
**CENTRO UNIVERSITÁRIO DE BRASÍLIA - UniCEUB**
**CURSO DE ENGENHARIA DE COMPUTAÇÃO**

**LEONARDO LIMA CORDEIRO DA COSTA**

# IMPLEMENTAÇÃO DE REDE DE DISPOSITIVOS DOMÉSTICOS COM CONTROLE VIA TV DIGITAL E TRANSMISSÃO DE DADOS VIA PLC

**Orientador:** Prof.ª M.S. Maria Marony Sousa Farias

Brasília
Dezembro, 2010.

LEONARDO LIMA CORDEIRO DA COSTA

# IMPLEMENTAÇÃO DE REDE DE DISPOSITIVOS DOMÉSTICOS COM CONTROLE VIA TV DIGITAL E TRANSMISSÃO DE DADOS VIA PLC

Trabalho apresentado ao Centro Universitário de Brasília (UniCEUB) como pré-requisito para a obtenção de Certificado de Conclusão de Curso de Engenharia de Computação.
Orientador: Prof.ª M.S. Maria Marony Sousa Farias

Brasília

Dezembro, 2010.

LEONARDO LIMA CORDEIRO DA COSTA

# IMPLEMENTAÇÃO DE REDE DE DISPOSITIVOS DOMÉSTICOS COM CONTROLE VIA TV DIGITAL E TRANSMISSÃO DE DADOS VIA PLC

Trabalho apresentado ao Centro Universitário de Brasília (UniCEUB) como pré-requisito para a obtenção de Certificado de Conclusão de Curso de Engenharia de Computação.
Orientador: Prof.ª M.S. Maria Marony Sousa Farias

Este Trabalho foi julgado adequado para a obtenção do Título de Engenheiro de Computação, e aprovado em sua forma final pela Faculdade de Tecnologia e Ciências Sociais Aplicadas - FATECS.

_____

Prof. Abiezer Amarilia Fernandez
Coordenador do Curso

**Banca Examinadora:**

_____

Profa. Maria Marony Sousa Farias, Mestre em Engenharia Elétrica.
Orientadora

_____

Prof. Gil Renato Ribeiro Goncalves, Doutor em Física.
UniCEUB

_____

Prof. Francisco Javier de Obaldia Diaz, Mestre em Engenharia Elétrica.
UniCEUB

_____

Prof. Luis Claudio Lopes de Araujo, Mestre em Matemática Pura.
UniCEUB

*Dedico este trabalho a minha família, que com amor incondicional, sempre me apoiou e acreditou em mim.*

## AGRADECIMENTOS

Primeiramente, agradeço a Deus, por toda a saúde, paz e proteção. Agradeço a Ele por estar sempre ao meu lado. Por ter me dado força e sabedoria para vencer mais esta etapa da vida.

Agradeço meus pais, Ana Cristina e José Carlos. Se hoje sou um cidadão digno e do bem, devo tudo a eles. É sem dúvida o amor mais incondicional que existe.

Aos meus avós, Maria Yeda e José Carlos, que, são também, meus outros pais. Sou abençoado, pois Deus me deu a oportunidade de ter dois pais e duas mães.

A minha irmã, Ticiana. Amiga e companheira, amor que não pode ser expressado em palavras.

Aos meus grandes amigos, que sempre apoiaram e entenderam os momentos delicados dessa caminhada que é a graduação.

Aos novos grandes amigos que fiz, durante os cinco anos de curso. A turma de Engenharia de Computação que compomos, será lembrada, para sempre, pela união e companheirismo.

A todos os professores, que, durante os cinco anos de curso, se esforçaram em tornar cada um dos alunos prontos para exercer de forma digna a profissão de Engenheiros de Computação.

À instituição UniCEUB, que proporciona a todos os alunos condições ideais para o melhor aprendizado possível.

Agradeço, em especial, à professora Marony e ao professor Javier, que com muita dedicação e paciência, acreditaram no meu projeto e me ajudaram a concluir este trabalho.

Muito obrigado.

**SUMÁRIO**

## LISTA DE FIGURAS

## LISTA DE TABELAS

## LISTA DE ABREVIATURAS E SIGLAS

| | |
|---|---|
| ABNT | Associação Brasileira de Normas Técnicas |
| Anatel | Agência Nacional de Telecomunicações |
| API | Application Programming Interface |
| ANEEL | Agência Nacional de Energia Elétrica |
| ARIB | Association of Radio Industries and Businesses |
| ATSC | Advanced Television Systems Committee |
| BPL | Broadband over Power Lines |
| CDM | Code Division Multiplexing |
| CPqD | Centro de Pesquisa e Desenvolvimento em Telecomunicações |
| DTV | Digital TV |
| DVB | Digital Video Broadcasting |
| DQPSK | Differential Quadrature Phase Shift Keying |
| FDM | Frequency Division Multiplexing |
| Ginga-CC | Ginga-Common Core |
| Ginga-J | Ginga-Java |
| HD | High Definition |
| HDTV | High Definition TV |
| HE-AAC | High-Efficiency Advanced Audio Coding |
| HTML | HyperText Markup Language |
| HTTP | HyperText Transfer Protocol |
| IBGE | Instituto Brasileiro de Geografia e Estatística |
| IP | Internet Protocol |
| ISDB | Integrated Services Digital Broadcasting |
| ISDB-T | Integrated Services Digital Broadcasting Terrestrial |
| ISDB-TB | Integrated Services Digital Broadcasting Terrestrial Brasil |
| JPEG | Joint Photographic Experts Group |
| LCD | Liquid Crystal Display |
| LED | Light-emitting diode |
| MHP | Multimedia Home Platform |
| Mbps | Mega bits por segundo |
| MHz | Mega hertz |

| | |
|---|---|
| MPEG | Moving Picture Experts Group |
| NBR | Norma da Associação Brasileira de Normas Técnicas |
| NCM | Nested Context Model |
| NHK | Nippon Hoso Kyokai |
| NTSC | National Television System Commitee |
| NCL | Nested Context Language |
| OFDM | Orthogonal Frequency Division Multiplexing |
| PDA | Personal Digital Assistant |
| PIC | Programmable Interface Controller |
| PLC | PowerLine Communication |
| PNAD | Pesquisa Nacional por Amostra de Domicílios |
| PUC-Rio | Pontifícia Universidade Católica do Rio de Janeiro |
| QAM | Quadrature Amplitude Modulation |
| QPSK | Quadrature phase-shift keying |
| RF | Radio frequency |
| RFCs | Request for Comments |
| RGB | Modelo de cores Red Green Blue |
| SBTVD | Sistema Brasileiro de TV Digital |
| SDTV | Standard Definition Television |
| SDK | Software Development Kit |
| SEBRAE | Agência de Apoio ao Empreendedor e Pequeno Empresário |
| SSH | Secure Shell |
| TDM | Time Division Multiplexing |
| TCP/IP | Transmission Control Protocol/Internet Protocol |
| UFPB | Universidade Federal da Paraíba |
| UTP | Unshielded Twisted Pair |
| UniCEUB | Centro Universitário de Brasília |
| XML | Extensible Markup Language |

**RESUMO**

O homem sempre trabalhou para tornar suas atividades do dia-a-dia mais fáceis, seja criando novas tecnologias exclusivamente para este fim, ou aproveitando e aprimorando tecnologias existentes. Automação residencial consiste na aplicação destas tecnologias em residências com o objetivo de diminuir os trabalhos domésticos aumentando a qualidade de vida, a segurança, além buscar o uso racional dos recursos dentro de uma residência. A chegada da TV Digital aos lares trouxe uma enorme inovação na utilização do aparelho de TV, a interatividade. O telespectador passa a ter uma papel em frente a TV e suas ações, agora, causam diferentes reações graças a grande possibilidade de aplicações oferecidas pela interatividade. Assim, o objetivo deste projeto é o desenvolvimento de um protótipo de automação residencial com o controle sendo exercido pelo telespectador, utilizando uma aplicação para TV Digital que é executada dentro de um *set-top-box* virtual de TV Digital. A aplicação foi desenvolvida dentro das normas da ABNT e se enquadra nos padrões do Sistema Brasileiro de TV Digital. O projeto utiliza os conceitos das linguagens *Nested Context Language* (NCL) e LUA, adotadas pelo Sistema Brasileiro de TV Digital, do *middleware* Ginga, todos responsáveis pela interatividade da TV Digital, e propõe a comunicação entre a TV Digital e o *hardware* controlador dos dispositivos domésticos por meio da rede elétrica, utilizando-se assim a tecnologia *PowerLine Communication*, comunicação pela rede elétrica.

Palavras Chave: TV Digital, SBTVD, middleware, Ginga, NCL, LUA, interatividade, microcontrolador, automação residencial, PLC, rede elétrica

**ABSTRACT**

Men have always worked to make his day-to-day activities easier, creating new Technologies exclusively for this purpose, or using and improving existing Technologies. Home automation consists on the application of this Technologies in homes with the objective of reducing the home work improving quality of life, security, and also seeking for the rational use of resources within a residence. The arrival of digital TV to the residences brought a huge innovation to the use of the TV device, the interactivity. The televiewer has now a role when sitting in front of TV, and his actions, now, cause different reactions, due to the great possibilities of applications offered by the interactivity. Thus, the objective of this work is the development of a prototype for home automation where the control is done by a televiewer using an application for Digital TV being executed inside a virtual set-top-box. The application was developed following the ABNT rules and fits the Brazilian Digital TV System. The prototype uses concepts of Nested Context Language (NCL) e LUA programming languages, adopted by the Brazilian Digital TV System, the middleware Ginga, all of them responsible for the interactivity offered by the Digital TV, and proposes the communication between the Digital TV and the hardware controlling the home devices over power line, using then the PowerLine Communication technology.

Key words: Digital TV, Brazilian Digital TV System, middleware, Ginga, NCL, LUA, interactivity, microcontroller, home automation, PLC, power line

# CAPÍTULO 1 – INTRODUÇÃO

A chegada da interatividade proporcionada pela TV Digital trouxe um leque infindável de aplicações. Desde o *t-commerce* e *t-banking* (realização de compras e transações bancárias, respectivamente, utilizando a TV), passando pelas enquetes interativas até a possibilidade de aplicações nativas ao *set-top-box* (nome dado ao *hardware* conversor do sinal digital) independentes da programação das emissoras. O campo de desenvolvimento destas aplicações já está bastante amplo, mas ainda há muito a se descobrir.

A vontade que o homem tem em tornar suas tarefas domésticas do dia-a-dia mais fáceis e práticas faz com que, constantemente, e ao surgimento de cada nova tecnologia, ele se esforce em aplicá-la no campo da automação residencial. E é exatamente desta área que este trabalho trata.

## 1.1 – Motivação

"A digitalização da televisão representa muito mais que uma melhoria de imagem, a alta definição. Ela representa um novo meio de comunicação de massa, uma tecnologia que permite a convergência da TV com outras mídias" (TEIXEIRA, ALMEIDA, JÚNIOR, 2005). Como pode ser visto, um dos principais objetivos da digitalização da TV é o acesso das massas sociais a tecnologias nunca antes vistas. Oferecer às diferentes classes sociais a oportunidade da inclusão digital.

O aprimoramento de tecnologias amplamente conhecidas pela sociedade é uma outra motivação para este trabalho. Segundo a Pesquisa Nacional por Amostra de Domicílio, PNAD, 95,1% dos lares brasileiros, em 2008, tinham pelo menos um aparelho de TV em casa. E segundo o Instituto Brasileiro de Geografia e Estatística, IBGE, em 2008, a rede elétrica estava presente em 98,6% dos lares brasileiros.

Há ainda a motivação ambiental, já que a proposta do trabalho visa também o uso racional dos recursos dentro de uma residência utilizando uma infraestrutura parcialmente pronta (aparelho de TV e rede elétrica). Ou seja, as adaptações necessárias, em uma residência, para que haja uma implementação comercial, seriam mínimas.

## 1.2 – Objetivo Geral

O objetivo geral deste trabalho é propor um protótipo de automação residencial utilizando inovações desenvolvidas para tecnologias amplamente conhecidas pela sociedade.

O trabalho constitui-se no desenvolvimento de uma aplicação para TV Digital, residente em um *set-top-box* virtual (Ginga-NCL *Virtual Set-top Box*), instalado em uma máquina virtual executada em um notebook, comunicando-se com um servidor *web* embarcado em um kit de desenvolvimento utilizando um microcontrolador, um controlador *ethernet* e quatro LEDs, simulando quatro lâmpadas em uma residência. A troca de dados entre o *set-top-box* e o kit de desenvolvimento ocorrerá por meio da tecnologia PLC, que utiliza a infraestrutura da rede elétrica. Aplicação interativa para TV Digital será desenvolvida utilizando as linguagens NCL e Lua, adotadas pelo Sistema Brasileiro de TV Digital.

## 1.3 – Objetivos específicos

Os objetivos específicos deste trabalho são:

- Desenvolvimento da aplicação para TV Digital utilizando as linguagens NCL e Lua;
- Instalação do ambiente Ginga-NCL *Virtual Set-top-box* em uma máquina virtual;
- Criação de uma infraestrutura de comunicação através de rede elétrica entre o Ginga-NCL *Virtual Set-top-box* e o kit de desenvolvimento contendo o PIC e os LEDs a serem controlados, utilizando a tecnologia PLC;
- Documentar todos os testes e resultados obtidos.

## 1.4 – Procedimento Metodológico

Primeiramente foi feita pesquisa relacionada aos temas Automação Residencial, TV Digital, Ginga, linguagem NCL, linguagem Lua, PICs, PLC e redes nas mais diversas fontes com o objetivo de obter um status a cerca do estado da arte desses temas e adquirir um embasamento teórico para o desenvolvimento do trabalho. Em seguida foi idealizado o protótipo proposto por este trabalho.

Então, foi dado início ao desenvolvimento do protótipo sendo instalada a máquina virtual Ginga-NCL *Virtual Set-top-box*, desenvolvendo a aplicação em NCL e Lua. Após isto,

foi feita pesquisa de mercado e adquirido o kit de desenvolvimento da ACEPIC Tecnologia e Treinamento LTDA contendo um microcontrolador e um controlador *ethernet* para comunicação. Foi feita nova pesquisa de mercado, agora para aquisição dos *HomePlugs* TP-Link modelo TL-PA201. Durante o desenvolvimento do protótipo, resultados foram anotados e evidências colhidas.

## 1.5 – Estrutura do trabalho

Além deste capítulo introdutório, esta monografia está estruturada em mais três capítulos e organizada da seguinte maneira:

- O capítulo 2 apresenta os problemas atualmente existentes e apresenta a proposta de solução para os problemas apresentados na integração dos conceitos de automação residencial, TV Digital e da tecnologia PLC;
- O capítulo 3 apresenta as bases teóricas utilizadas para pesquisa e desenvolvimento do trabalho e do protótipo proposto;
- O capítulo 4 apresenta um protótipo de automação residencial utilizando as tecnologias TV Digital e PLC.
- O capítulo 5 apresenta os testes e as validações dos resultados obtidos;
- E por fim, o capítulo 6 apresenta as conclusões obtidas após o desenvolvimento deste trabalho, sugerindo propostas de trabalhos futuros.

## 1.6 – Resultados esperados

Como resultado, é esperado que a comunicação entre a TV Digital e o kit de desenvolvimento funcione corretamente através da tecnologia PLC, utilizando os *HomePlugs*, e os comandos enviados a partir do controle remoto, simulado pelo teclado do *notebook*, cheguem ao circuito, permitindo assim o controle de acender e/ou apagar os LEDs.

# CAPÍTULO 2 - APRESENTAÇÃO DO PROBLEMA

## 2.1 – Automação Residencial

Sempre preocupado com sua qualidade de vida e bem-estar, o homem trabalha a fim de tornar seu dia-a-dia mais fácil e prático. Assim, esforça-se na evolução das tecnologias de modo que possa aplicá-las de forma eficiente em suas atividades cotidianas.

A palavra Domótica, que é formada pela junção da palavra latina "*Domus*", que significa casa, com a palavra robótica refere-se à aplicação de diversas tecnologias em residências com o objetivo de diminuir os trabalhos domésticos melhorando a qualidade de vida, a segurança, além buscar o uso racional dos recursos dentro de uma residência.

A automação residencial pode ser implementada por meio de dispositivos com inteligência artificial como, por exemplo, uma janela que "percebe" a incidência solar por meio de sensores e automaticamente fecha a persiana. Outra alternativa é o controle pelo usuário enviando os comandos para que estes sejam executados por mecanismos eletrônicos, como um comando de voz para abrir a porta, ou envio de mensagens a partir de um celular ou *palm-top*. A figura 2.1 mostra diferentes aplicações de automação residencial.


Figura 2.1: Aplicações de automação residencial

Fonte: (COMPLEXX TECNOLOGIA)

Enfim, existem diversas formas de se implementar o conceito de *smart home* com as tecnologias que dispomos hoje, e há no mercado inúmeras empresas especializadas neste tipo de serviço. Porém, dada as constantes evoluções da tecnologia, novas ferramentas para este fim surgem quase que diariamente. O dia-a-dia das pessoas cada vez mais corrido, o apelo ambiental pelo uso racional de recursos como a energia elétrica e a necessidade cada vez maior de segurança faz com que a motivação por novos meios de implementação de automação residencial seja infindável.

Existem diversos dispositivos que podem ser usados como central de controle para automação residencial: celulares, computadores, *PDAs*, etc.. A chegada da TV Digital oferecendo grande interatividade entre usuário e a TV inclui este aparelho no grupo de centrais de controle para automação residencial.

## 2.2 – TV Digital

Desde 1992, ano em que começaram a ser divulgados dados acerca da quantidade de lares com televisores, este número vem crescendo. Devido a enorme inserção da TV na vida dos brasileiros, o seu uso é visto como simples, e novidades devem ser facilmente aprendidas.

Apesar de ser um dos principais meios de comunicação no Brasil, a televisão analógica não oferece ao telespectador a oportunidade de interagir com o aparelho, ou seja, ele assiste à programação podendo apenas mudar de canal, aumentar o volume, e fazer configurações de imagem na tela do aparelho.

Nos anos 70, cientistas do NHK (*Nippon Hoso Kyokai*) *Science & Technical Research Laboratories* deram início aos estudos do que mais tarde viria a ser a TV Digital. A direção da rede pública de TV do Japão NHK juntamente com um consórcio de cem estações comerciais decidiram investir no desenvolvimento de uma TV de alta definição com o intuito de disponibilizar na casa do usuário, uma qualidade de imagem e som semelhante às do cinema.

No Brasil, a TV Digital surgiu em 1996 com as transmissões das operadoras de TV por assinatura SKY e DirecTV. Era, porém, bem diferente da TV Digital que temos hoje, pois não tinha imagem em alta definição e sua interatividade era muito limitada.

Quando as discussões para a implantação da TV digital como conhecemos hoje, iniciaram-se, três sistemas de transmissão eram estudados:

- o modelo ATSC americano;

- o modelo DVB europeu;
- e o modelo ISDB japonês.

A Anatel, juntamente com o Centro de Pesquisa e Desenvolvimento em Telecomunicações, CPqD, deu início ao processo de avaliação técnica e econômica da situação brasileira para que pudessem selecionar o modelo de transmissão de TV Digital que seria adotado no Brasil. Em 2003, através do decreto 4.901, foi fundado o Comitê do Sistema Brasileiro de TV Digital.

Após os estudos, foi proposto pelo Ministério das Comunicações o Sistema Brasileiro de TV Digital e "em 2006, V. Exa. Luiz Inácio Lula da Silva, presidente do Brasil, assinou o decreto 5620/2006, que trata sobre a implantação do SBTVD Terrestre e seu padrão como sendo o ISDB-$T_B$, já que foi desenvolvido com base no sistema japonês ISDB-T - *Integrated Services Digital Broadcasting Terrestrial"* (BASTOS, 2010).

"Ao ser adotado no Brasil, o ISDB recebeu atualizações tecnológicas nas partes de áudio, vídeo e interatividade. Então o "B" é para contemplar essas atualizações, mas ele não é mais um sistema de TV Digital; ele é o ISDB-T reconhecido pelas organizações internacionais que regulam as telecomunicações no mundo" (FÓRUM SBTVD, 2010).

O Sistema Brasileiro de TV Digital "oferece uma série de diferenciais em relação aos sistemas adotados pelo resto do mundo. Junta os eficientes padrões de compressão digital de áudio e vídeo introduzidos pelo Brasil, com a base técnica de transmissão do sistema japonês" (BASTOS, 2010) que atende a equipamentos portáteis. Ou seja, a transmissão em alta definição e a interatividade podem ser acessadas a partir de aparelhos de televisão fixos e dispositivos móveis, como, por exemplo, celulares.

No dia 02 de dezembro de 2007, foi realizada no Brasil, a primeira transmissão digital aberta na cidade de São Paulo. Hoje, a grande maioria das capitais brasileiras tem acesso à transmissão HD (*High Definition*, alta definição) através do SBTVD que além de ter o sinal aberto, é livre e gratuito.

A transmissão do SBTVD iniciou-se no fim de 2007 e segue conforme o cronograma abaixo mostrado na figura 2.2 a seguir:

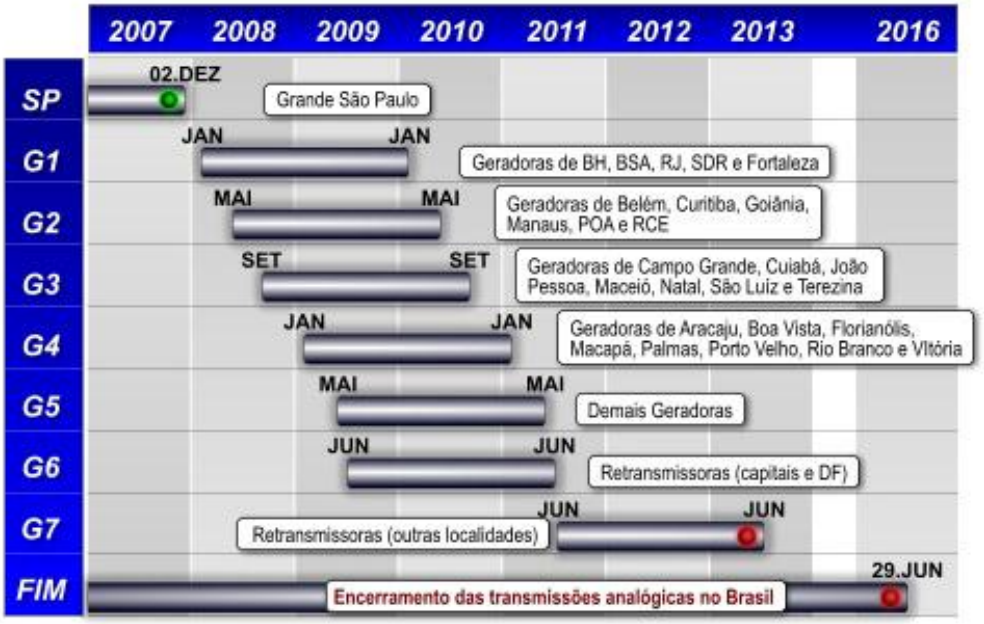


Figura 2.2: Cronograma TV Digital no Brasil

Fonte: (FÓRUM SBTVD, 2006)

Desta forma, até meados de 2012, todo o Brasil deve ser coberto pelo sinal digital e a partir de primeiro de julho de 2013, o Ministério das Comunicações somente outorgará a exploração do serviço de radiodifusão de sons e imagens para a transmissão em tecnologia digital. Em junho de 2016, está previsto o encerramento das transmissões analógicas no Brasil.

Um dos maiores diferenciais proporcionado pela TV Digital é a interatividade que ela proporciona. O telespectador passa a ter um papel ativo em frente à televisão e a transmitir dados de seu interesse como resposta a uma enquete, consulta de saldo de conta corrente, ou requisição de compra de algum produto. Para dar suporte a essa interatividade, o SBTVD adotou o *middleware* Ginga.

## 2.3 – Ginga

"*Middleware* é uma camada de *software* posicionada entre o código das aplicações e a infra-estrutura de execução" (SOARES, 2008). É um *software* que age como um intermediador na comunicação entre as aplicações e o *hardware* sobre o qual elas executarão.

O nome Ginga foi escolhido por ser um *software* brasileiro e em reconhecimento à cultura, arte e contínua luta por liberdade e igualdade do povo brasileiro. Ginga é o nome do *Middleware* Aberto do Sistema Brasileiro de TV Digital (SBTVD), instalado nos conversores (*set-top-boxes*). É constituído por um conjunto de tecnologias padronizadas e inovações brasileiras que o tornam a especificação de *middleware* mais avançada e a melhor solução para os requisitos do país.

O *middleware* do SBTVD possui duas funções principais: a primeira é tornar o desenvolvimento de novas aplicações mais simples, já que elas funcionarão sobre o Ginga, independentemente do sistema operacional e do *hardware* do *set-top-box,* e a segunda é dar o necessário suporte à interatividade.

O *middleware* Ginga é fruto de anos de trabalho de desenvolvimento liderados pelos laboratórios Telemídia, da Pontífice Universidade Católica do Rio de Janeiro e LAViD da Universidade Federal da Paraíba.

Ele pode ser dividido em três subsistemas principais:

- Ginga-CC: Ginga *Common-Core* é o "coração" do *middleware* e oferece o suporte básico para os ambientes declarativos e procedural. "Ginga-CC é o subsistema lógico que provê todas as funcionalidades comuns ao suporte dos ambientes declarativo, Ginga-NCL, e imperativo, Ginga-J" (SOARES, 2008). E é apenas este subsistema do Ginga que é adaptado ao *hardware* onde ele será instalado, ou seja, é este módulo que provê o nível de abstração da plataforma de *hardware* e sistema operacional. Além disto, ele faz a exibição dos objetos de mídia como áudio, vídeo, texto e imagem pois fornece o controle do plano gráfico para o modelo especificado para o ISDB-TB e é responsável pelos dados obtidos através do canal de interatividade ou de retorno, que é o módulo responsável pelo acesso à camada de rede do *set-top-box*.
- Ginga-J: desenvolvido pela UFPB, o Ginga-J é um ambiente imperativo que tem como função prover uma infraestrutura de execução de aplicações desenvolvidas na linguagem Java. É dividido em três módulos: a máquina virtual Java, o núcleo e suas APIs, também chamadas APIs verde do Ginga-J, e o módulo responsável pelo suporte às APIs específicas do Ginga-J, chamadas de APIs amarela e vermelha do Ginga-J.
- Ginga-NCL: desenvolvida nos laboratórios da Pontifícia Universidade Católica do Rio de Janeiro, é a inovação totalmente brasileira do SBTVD. Tem como função "prover uma infra-estrutura de apresentação de aplicações baseadas em documentos hipermídia escritos em linguagem NCL, com facilidades para a especificação de aspectos de interatividade,

> sincronismo espaço-temporal de objetos de mídia, adaptabilidade e suporte a múltiplos dispositivos”(PORTAL DO SOFTWARE PÚBLICO BRASILEIRO, 2010). “O ambiente tem por base a linguagem NCL (uma aplicação XML) e sua linguagem de *script* Lua, ambas desenvolvidas nos laboratórios da Pontifícia Universidade Católica do Rio de Janeiro” (SOARES, 2008).

O Fórum do Sistema Brasileiro de TV Digital Terrestre publicou junto à ABNT uma série de Normas que padronizam a TV Digital. Os documentos são de livre acesso e fazem parte do grupo "Codificação de dados e especificações de transmissão para radiodifusão digital". As normas aprovadas e publicadas até março de 2008 são as seguintes:

- ABNT NBR 15606-1 - Televisão digital terrestre - Codificação de dados e especificações de transmissão para radiodifusão digital - Parte 1: Codificação de dados;
- ABNT NBR 15606-2 - Televisão digital terrestre - Codificação de dados e especificações de transmissão para radiodifusão digital - Parte 2: Ginga-NCL para receptores fixos e móveis - Linguagem de aplicação XML para codificação de aplicações;
- ABNT NBR 15606-3 - Televisão digital terrestre - Codificação de dados e especificações de transmissão para radiodifusão digital - Parte 3: Especificação de transmissão de dados;
- ABNT NBR 15606-5 - Televisão digital terrestre - Codificação de dados e especificações de transmissão para radiodifusão digital - Parte 5: Ginga-NCL para receptores portáteis - Linguagem de aplicação XML para codificação de aplicações.

O Brasil, ao adotar as soluções do Ginga em seu sistema de TV Digital, dá um enorme passo rumo a inclusão social das mais variadas classes, pois, sendo uma especificação livre e completa, qualquer programador pode desenvolver aplicações interativas com os mais variados objetivos. Isto dá ao Brasil o posto de primeiro país a oferecer um conjunto de soluções livres para TV digital, e torna o Sistema Brasileiro de TV Digital o mais avançado sistema de TV digital terrestre, não apenas por usar as tecnologias mais avançadas, mas, principalmente, por dispor de tecnologias inovadoras, como o Ginga.

## 2.4 – PLC – *PowerLine Communication*

Segundo a Resolução Normativa nº 3752009, de 25 de agosto de 2009, da Agência Nacional de Energia Elétrica, ANEEL, PLC (*PowerLine Communication*) é o sistema de telecomunicações que utiliza a rede elétrica como meio físico para a comunicação digital ou analógica de sinais, tais como: *internet*, vídeo, voz, entre outros, incluindo *Broadband over*

*PowerLine* – BPL. É uma tecnologia de rede emergente que reusa o sistema de fios elétricos de uma residência para comunicação entre dispositivos e a *internet* (ZAHARIADIS, 2003).

Por utilizar a infraestrutura já disponível da rede elétrica, o PLC apresenta uma grande vantagem do ponto de vista econômico para transmissão de dados, em relação a outras tecnologias. “A principal idéia por trás do PLC é a redução de custo operacional e gastos para realização de novas redes de telecomunicações” (HRASNICA, HAIDINE, LEHNERT, 2004). E tem ainda um papel muito importante do ponto de vista social: pode representar a democratização do acesso aos principais meios de comunicação. O uso da tecnologia pode ser usado para transmissão de TV a cabo, internet banda larga e vários outros serviços.

O potencial de mercado para a tecnologia PLC é gigantesco já que a grande maioria das residências estão ligadas na rede elétrica. Mas a tecnologia de transmissão de dados pela é rede elétrica é ainda pouco difundida no Brasil devido aos principais obstáculos que enfrenta: ruído, atenuação e distorção (DUQUE, 2001). Os principais obstáculos para transmissão de dados sobre a rede elétrica são a topologia física, as características físicas do cabeamento elétrico e as interferências geradas pela transmissão de corrente elétrica (ZAHARIADIS, 2003). Como o meio de propagação do sinal não é de caráter exclusivo, a presença de ruídos e fatores que prejudicam a transmissão é inevitável. Esses ruídos podem ser provocados por aparelhos eletroeletrônicos, ligados à rede elétrica (TAVEIRA, 2004).

Existem, hoje no mundo, diversos projetos em desenvolvimento acerca da tecnologia PLC. E graças ao aproveitamento de infraestrutura e fiação já presente na grande maioria dos lares, esta surge com um potencial enorme para comunicação para os mais diversos fins, e no caso proposto, trata-se de uma comunicação entre a TV Digital, preparada para comunicação *ethernet* e um dispositivo de controle para automação residencial.

## 2.5 – Proposta do Trabalho

A proposta deste trabalho é desenvolver um protótipo para utilização em automação residencial aproveitando-se de evoluções de tecnologias largamente conhecidas pela sociedade. O protótipo utilizará a interatividade alcançada pela chegada da TV Digital e a possibilidade de transmissão de dados pela rede elétrica utilizado a tecnologia PLC.

Será desenvolvida uma interface gráfica utilizando documento hipermídia da linguagem NCL para acesso das funcionalidades de dispositivos domésticos. Este aplicativo será armazenado no *set-top-box* da TV. Quando o usuário selecionar a ação (acender/apagar o LED) desejada, será enviado o devido comando a uma placa microcontrolada com um servidor *web* embarcado. Estes dados serão enviados através da rede elétrica sendo o sinal modulado/demodulado pelos aparelhos *HomePlug*.

# CAPÍTULO 3 – REFERENCIAL TECNOLÓGICO

Neste capítulo serão expostas as tecnologias utilizadas no projeto do protótipo proposto, como a TV Digital e as tecnologias que compõem o Sistema Brasileiro de TV Digital, e a tecnologia PLC, utilizada na comunicação do protótipo.

## 3.1 – A Linguagem NCL

A linguagem NCL é a parte das especificações do Sistema Brasileiro de TV Digital usada pela máquina de apresentação do *middleware* Ginga, o Ginga-NCL, do SBTVD. É uma linguagem declarativa adotada pelo Sistema Brasileiro de TV Digital para autoria de documentos hipermídia.

O desenvolvimento em linguagens declarativas não requer um especialista em programação, uma vez que linguagens declarativas são muito intuitivas e descrevem de forma simples e direta os resultados que se espera que o programa atinja. “Nas linguagens declarativas, o programador fornece apenas o conjunto das tarefas a serem realizadas, não estando preocupado com os detalhes de como o executor da linguagem realmente implementará essas tarefas”(BARBOSA, SOARES, 2008). “A linguagem enfatiza a declaração descritiva de um problema ao invés de sua decomposição em implementações algorítmicas, não necessitando, em geral, de tantas linhas de código para definir certa tarefa.” (CARVALHO, SANTOS, DAMASCENO, SILVA, SAADE, 2009).

Os documentos desenvolvidos na linguagem NCL (*Nested Context Language*), são baseados no modelo conceitual NCM (*Nested Context Model*) com sincronização espacial e temporal para exibição dos objetos de mídia. NCL é uma linguagem de “cola” que reúne diferentes objetos de mídia, sejam eles áudio, vídeo, um documento HTML ou mesmo um objeto procedural, em uma apresentação multimídia.

### 3.1.1 – *Nested Context Model*

O NCM, Modelo de Contextos Aninhados, adota os conceitos de nós e elos para composição de documentos hipermídia. Um nó representa a abstração das mídias no documento além de trazer informações como a forma de apresentação do objeto de mídia. Existem dois tipos de nós: nós de conteúdo, que trazem informações sobre a mídia utilizada

pelo documento, e nós de composição que possuem um conjunto de nós de conteúdo e nós de composição e um conjunto de elos, sendo utilizado para dar estrutura e organização a um documento hipermídia.

Um documento hipermídia deve ser desenvolvido de forma que apresente algumas informações básicas:

- Onde: A primeira informação a ser dada são as áreas disponíveis para exibição de objetos de mídia. No modelo NCM, esta informação é chamada região, e indica a posição e o tamanho da área onde o nó deverá ser exibido. Neste ponto, ainda não é definido o que será apresentado em cada região.
- Como: Esta informação diz respeito a como o objeto de mídia, ou o nó será apresentado e é dada utilizando-se de um elemento chamado descritores. Os descritores informam características como a região onde será apresentado, o volume do som, a transparência a duração da exibição, e outras.
- O que: O elemento mídia representa cada nó de um documento e indica cada objeto de mídia do documento hipermídia. É este elemento que representa os conteúdos que serão exibidos. No elemento mídia também é informado a qual descritor o nó está relacionado. Uma mídia deve, obrigatoriamente estar dentro de um nó chamado contexto, que é um nó de composição. Este nó representa um documento completo ou apenas parte de um documento.
- Quando: Para se definir o primeiro nó a ser apresentado utiliza-se o elemento chamado porta, e a para definição da ordem de apresentação dos nós em uma exibição hipermídia utilizam-se o elementos elos (*links*). As portas definem os nós a serem apresentados quando um nó de contexto é iniciado e os *links* definem os relacionamentos de sincronização entre os nós e a interatividade do programa" (CARVALHO, SANTOS, DAMASCENO, SILVA, SAADE, 2009).

### 3.1.2 – Estrutura de um documento NCL

A linguagem NCL é baseada em XML, ou seja, os documentos são organizados de forma hierárquica em uma árvore onde cada elemento possui um elemento pai e elementos filhos. Essa organização se assemelha a da árvore genealógica de uma família onde cada indivíduo seria um elemento XML.

Os documentos NCL tem sua estrutura dividida em duas partes: cabeçalho e corpo. Estas partes estão dentro de um nó pai chamado <*ncl*> que é o primeiro elemento definido em um documento NCL.

3.1.2.1 – Cabeçalho

No cabeçalho do documento NCL, marcado pelas *tags* <*head*><*/head*> são definidos os parâmetros de apresentação dos objetos de mídia utilizando-se as regiões e os descritores, e os elementos responsáveis pela sincronização da apresentação, através dos conectores.

3.1.2.2 – Corpo

O corpo do documento NCL é determinado pelas *tags* <*body*><*/body*>. Dentro destas *tags* deve ser definido o elemento porta, que indica o início da apresentação dos objetos de mídia, os objetos de mídia e os descritores devem ser relacionados, além de serem definidos os elementos *links* que determinam a ordem de sincronismo em que os objetos de mídia, previamente definidos, devem ser exibidos.

**3.2 – A Linguagem Lua**

Lua é uma linguagem de programação brasileira, desenvolvida no laboratório Tecgraf da PUC-Rio, em 1993. É uma linguagem de *script* projetada para estender aplicações, portanto, ela é rápida, leve e dinâmica. A linguagem de *script* Lua, escolhida para fazer parte do SBTVD, juntamente com a linguagem NCL compõem o padrão de interatividade Ginga-NCL.

Utilizando apenas a linguagem declarativa NCL, o subsistema Ginga-NCL ficaria bastante limitado às barreiras de uma linguagem declarativa e seria difícil executar tarefas como processamento matemático, manipulação sobre textos e uso do canal de interatividade, além de outras. Assim, o SBTVD encontrou a solução de se criar uma nova classe de objetos de mídia Lua, os quais são chamados de NCLua e "por meio de elementos de mídia, scripts NCLua podem ser inseridos em documentos NCL, trazendo poder computacional adicional às aplicações declarativas"(SANT'ANNA, CERQUEIRA, SOARES, 2008).

Assim como ocorre a abstração dos objetos de mídia usados para imagens e vídeos, os scripts NCLua são apenas referenciados e relacionados em um documento NCL, devendo ser escritos em arquivos separados do documento NCL.

### 3.3 – TV Digital

Desde 1950, quando nos Estados Unidos, iniciou-se a transmissão em cores, o mundo não percebia grandes mudanças na forma de se assistir TV. E esta novidade demorou a chegar ao Brasil. Apenas em 19 de fevereiro de 1972, cerca de 500 televisores receberam imagens em cores da Festa da Uva, diretamente de Caxias do Sul.

A TV analógica persiste na maioria dos lares brasileiros, mas desde dezembro de 2007 este quadro vem mudando. A chegada da interatividade, trazida pela TV Digital, aliada à significante melhora na qualidade da imagem e som vem convencendo a todos de que a TV Digital traz muitas vantagens em relação à TV analógica.

A maior destas vantagens é a interatividade que dará ao telespectador um papel ativo quando estiver assistindo à programação das emissoras. Um toque no botão do controle remoto, e ele poderá ler o que aconteceu no capítulo anterior da novela, um outro toque em outro botão e surgirá uma breve descrição sobre o personagem em foco, as respostas a enquetes em programas de auditório também poderão ser dadas com apenas um toque no controle remoto, comprar um produto anunciado em uma promoção no intervalo do jogo de futebol também estará ao alcance do controle. Ou seja, a TV Digital chega com a responsabilidade de mudar de vez a forma de se assistir TV.

E não é apenas a programação das emissoras que permitirá interatividade na TV Digital. As aplicações para TV Digital podem ser divididas em três categorias:

- Aplicações dependentes do conteúdo televisivo;
- Independentes do conteúdo televisivo;
- Independentes do conteúdo televisivo e residente no *set-top-box*.

A figura 3.1 mostra os diferentes receptores da TV Digital:


Figura 3.1: TV Digital

Fonte: (FÓRUM SBTVD)

### 3.3.1 – História da TV até a TV Digital

Antes das primeiras transmissões de boa qualidade, em Berlim, a história da TV é confusa, porém sabe-se que diversos cientistas deram suas contribuições para produzirem o que no início era apenas um aparelho de rádio com um disco giratório mecânico que produzia uma imagem do tamanho de um selo postal. As primeiras transmissões de boa qualidade ocorreram em 1935, com a intenção de se transmitir as Olimpíadas de Berlim. Porém, durante a Segunda Guerra Mundial, as transmissões foram interrompidas, com exceção da própria Alemanha. A figura 3.2 mostra um exemplar dos primeiros aparelhos de TV da história:

Figura 3.2: Televisão da década de 30

Fonte (CAMARGO, 2009)

No início dos anos 1950, as televisões já eram cerca de 10 milhões nos Estados Unidos, e o mundo tentava um grande avanço nesta área: transmissão de imagem a cores. Porém todas as propostas sugeriam um novo sistema em que não era possível aproveitar estes 10 milhões de aparelhos. Foi criado, então, o *National Television System Commitee*, comitê especial responsável por encontrar uma solução para este problema.

E o comitê chegou a uma solução que recebeu o nome de suas iniciais: NTSC. O sistema baseava-se no sistema preto e branco que trabalhava com níveis de luminância (Y) e acrescentou a cor (C). O princípio baseia-se na decomposição da luz branca em três cores primárias: *Red* (vermelho), *Green* (verde) e o *Blue* (azul), daí o conhecido padrão RGB.

A história da TV no Brasil inicia-se em 1950 com a criação da Rede Tupi. Assis Chateaubriand, criador da Rede Tupi trouxe dos Estados Unidos cerca de duzentos televisores e os espalhou por São Paulo. O sucesso foi tanto que apenas seis anos depois, o número de aparelhos de TV no país já ultrapassava os 1,5 milhões. Desde então, a TV é um dos principais meios de comunicação do pais e um importante fator da cultura popular da sociedade.

Mas, desde a Rede Tupi, transmitindo imagens em preto e branco, a TV no Brasil demorou a dar outro grande passo. Apenas em 19 de fevereiro de 1972, cerca de 500 televisores receberam a primeira transmissão oficial a cores, em ocasião da Festa da Uva, diretamente de Caxias do Sul, e a emissora responsável pelo feito foi a TV Difusora.

As imagens a cores só chegaram ao Brasil na década de 70, mas desde a mesma década, um outro gigantesco passo na história da televisão já era estudado. No Japão, a direção da rede pública de TV do Japão *Nippon Hoso Kyokai* (NHK) juntamente com um consórcio de 100 estações comerciais, dá carta branca aos cientistas do *NHK Science & Technical Research Laboratories* para desenvolver uma TV de alta definição (que seria chamada de HDTV). A intenção era oferecer aos telespectadores uma experiência semelhante a vivida nas salas de cinema, com imagem e som de alta definição.

### 3.3.2 – Padrões de TV Digital no Mundo

A implantação da TV Digital no mundo iniciou-se em diferentes épocas em cada pais. No Reino Unido, por exemplo, o processo iniciou-se em 1998 e 65,9% das residências já tinham acesso à TV Digital em setembro de 2005. Nos Estados Unidos o início foi em 2002 e no Japão em 2003 (TELECO, 2010).

Por meio de estudos, cada país define o sistema de TV Digital que melhor se adapta a suas condições de infraestrutura, econômicas e sociais. Atualmente, existem três importantes sistemas em desenvolvimento que são objeto de estudos dos países que pretendem introduzir adotar o sistema de TV digital. O sistema norte-americano ATSC - *Advanced Television Systems Committee*, o sistema predominantemente europeu DVB - *Digital Vídeo Broadcasting* e o sistema japonês, conhecido como ISDB - *Integrated Services Digital Broadcasting*.

3.3.2.1 – O sistema norte-americano ATSC

O sistema ATSC é o padrão da TV Digital dos Estados Unidos e já foi adotado por outros oito países é o que oferece a melhor qualidade de imagem entre os três sistemas citados anteriormente. O nome de seu middleware é DASE (*DTV Application Software Environment*).

3.3.2.2 – O sistema europeu DVB

É conhecido por ser o mais versátil dos principais sistemas, o que facilita a transmissão de diversos canais na mesma freqüência. Este sistema trabalha na freqüência de 8 MHz, o que o deixa em desvantagem em relação aos padrões japonês e americano, que operam em 6 MHz, mesmo espectro usado no Brasil para a TV aberta. O padrão europeu tem ainda uma outra desvantagem: necessita de um canal adicional para a transmissão de TV portátil, o que leva o consumidor final a sofrer com a tarifação.

O middleware do DVB é o MHP (*Multimídia Home Platform*) que prioriza a segurança e interoperabilidade. Uma das vantagens e semelhanças com o Ginga, é que ele tem seu código aberto.

3.3.2.3 – O sistema japonês ISDB

O padrão japonês é apontado como o mais flexível pois atende melhor aos requisitos de mobilidade e portabilidade. Ele é uma evolução do sistema DVB-T, e teve o início de seu desenvolvimento na década de 70 pelo laboratório de pesquisa da rede de TV NHK.

Para se chegar ao ISDB, o DVB sofreu algumas melhorias:

- Foi acrescentado um "*Interleaver*" temporal para melhorar o desempenho na presença de interferências concentradas, tais como o ruído impulsivo;

- A banda de RF de 6MHz foi subdividida em 13 segmentos independentes, com a possibilidade de serem enviadas 3 programações diferentes ao mesmo tempo, por exemplo: uma em QPSK, outra em 16QAM e outra em 64QAM;
- Foi acrescentado o modo 4K;
- Foi acrescentado o método de modulação DQPSK (*Differential Quaternary Phase Shift Keying*).

Este padrão adotou o middleware ARIB (*Association of Radio Industries and Business*).

A tabela 01 apresenta um comparativo entre os três padrões apresentados:

Tabela 01 - Características dos três sistemas abertos com relação à difusão terrestre

| **Sistema** | **DVB** | **ATSC** | **ISDB** |
|---|---|---|---|
| **Modulação** | COFDM | 8-VSB | COFDM |
| **Multiplexação** | MPEG-2 Sistemas | MPEG-2 Sistemas | MPEG-2 Sistemas |
| **Codificação de Vídeo** | MPEG-2 Vídeo | MPEG-2 Vídeo | MPEG-2 Vídeo |
| **Codificação de Áudio** | MPEG-2 Layer II | Dolby AC3 | MPEG-2 AAC |
| **Middleware** | MHP | DASE | ARIB-Std-B24 |
| **Largura de Banda** | 6 a 8 MHz | 6 a 8 MHz | 6 a 8 MHz |
| **Taxa de Transmissão** | De 4,98 a 31,67 Mbps | 19,4 Mbps | De 3,65 a 23,23 Mbps |

Fonte: Piccioni, 2005

A tabela 02, disponível no Portal Teleco, mostra os sistemas adotados por diferentes países da America Latina e faz um breve comentário a respeito da implementação em cada um:

Tabela 02 - Implantação da TV Digital na America Latina

| **País** | **Padrão** | **Implantação** |
|---|---|---|
| **México** | ATSC | O padrão foi adotado em julho de 2004. |
| **Brasil** | ISDB-T | Lançada no dia 02 de dezembro de 2007 na cidade de São Paulo. |
| **Uruguai** | DVB-T e DVB-H | No dia 28 de agosto de 2007 o governo anunciou a escolha do padrão europeu. Embora a decisão tenha sido tomada, o presidente recém-eleito, José Mujica, anunciou que poderá desistir do padrão europeu e optar pelo sistema ISDB-T. |
| **Colômbia** | DVB-T | Em 28 de agosto de 2008, a Comissão Nacional de Televisão da Colômbia anunciou a adoção do sistema europeu. |

| País | Padrão | Implantação |
|---|---|---|
| **Peru** | ISDB-T | Em 23 de abril de 2009, o governo peruano anunciou a escolha do padrão ISDB. Com a notícia, o Peru se tornou o primeiro país na América do Sul a aderir ao padrão denominado nipo-brasileiro. De acordo com o cronograma peruano, os sinais digitais estarão disponíveis em Lima em março do próximo ano. A previsão é que as transmissões analógicas sejam encerradas em 15 anos. |
| **Argentina** | ISDB-T | Em 28 de agosto de 2009, os governos brasileiro e argentino assinaram um convênio bilateral para a implantação do sistema de TV Digital nipo-brasileiro na Argentina. O país foi o segundo na América do Sul a aderir ao padrão. |
| **Chile** | ISDB-T | O governo do Chile anunciou, em 14 de setembro de 2009, a adesão ao padrão ISDB-T. A previsão é de que as primeiras transmissões digitais no país sejam realizadas a partir de 2010. |
| **Venezuela** | ISDB-T | O governo da Venezuela anunciou, no dia 6 de outubro de 2009, a decisão de adotar sistema japonês de televisão digital. O sistema de televisão analógico deverá desativado no país em 2018. |
| **Equador** | ISDB-T | Em 26 de março de 2010, o governo do Equador anunciou a adesão ao sistema ISDB-T. Com a decisão, o Equador se torna o sexto país da América Latina a aderir oficialmente ao padrão. O prazo de implantação do sistema é estimado em sete anos e o desligamento definitivo das transmissões analógicas deverá acontecer em 2017. |
| **Costa Rica** | ISDB-T | A escolha pelo padrão nipo-brasileiro foi homologada em 7 de maio de 2010. |
| **Paraguai** | ISDB-T | Em 2 de junho, o Paraguai tornou-se o oitavo país da América Latina a aderir ao padrão ISDB-T |
| **Bolívia** | ISDB-T | Em 5 de julho, o governo boliviano anunciou a adoção do padrão nipo-brasileiro a TV Digital no país. A implementação da nova tecnologia no país levará dois anos. As primeiras transmissões serão realizadas em 2011. |
| **Cuba** | - | O país está testando pelo menos três sistemas de televisão digital. Embora, não tenha prazo fixo para que seja tomada uma decisão, espera anunciar o padrão até o final deste ano. |
| **Nicarágua** | - | O governo do país avalia a implementação do padrão ISDB-T. |

Fonte: (TELECO, 2010)

### 3.3.3 – Implantação da TV Digital no Brasil

Em 1999, foi estabelecido um termo de cooperação técnica entre a Anatel e o CPqD e ambos deram início ao processo de avaliação técnica e econômica para a tomada de decisão quanto ao padrão de transmissão digital a ser aplicado no Brasil ao Serviço de Radiodifusão de Sons e Imagens. “A escolha do CPqD para a prestação desses serviços considerou não apenas o histórico de serviços prestados à Agência e às empresas operadoras da antiga Telebrás, mas o elevado domínio técnico das tecnologias de compressão digital de sons e imagens” (DTV, 2010).

O decreto 4.901 de 2003 instituiu, além de outros, o Comitê do Sistema Brasileiro de TV Digital responsável pelos estudos que definiriam o padrão a ser adotado no país. Os objetivos do Comitê eram:

- promover a inclusão social, a diversidade cultural do País e a língua pátria por meio do acesso à tecnologia digital, visando à democratização da informação;
- estimular a pesquisa e o desenvolvimento e propiciar a expansão de tecnologias brasileiras e da indústria nacional relacionadas à tecnologia de informação e comunicação;
- planejar o processo de transição da televisão analógica para a digital, de modo a garantir a gradual adesão de usuários a custos compatíveis com sua renda;
- viabilizar a transição do sistema analógico para o digital, possibilitando às concessionárias do serviço de radiodifusão de sons e imagens, se necessário, o uso de faixa adicional de radiofreqüência, observada a legislação específica;

Após os estudos, em 23 de junho de 2006, através do decreto 5.820, o governo brasileiro informou oficialmente ao governo japonês a escolha do padrão japonês ISDB-T (*Integrated Services Digital Broadcasting Terrestrial*) de TV Digital.

Os principais pontos definidos no decreto são:

- O decreto definiu que o Sistema Brasileiro de Televisão Digital Terrestre (SBTVD-T) adotará, como base, o padrão de sinais do ISDB-T e possibilitará transmissão digital em alta definição (HDTV) e em definição padrão (SDTV); transmissão digital simultânea para recepção fixa, móvel e portátil, e interatividade.
- As emissoras de TV receberão um canal de radiofreqüência com largura de banda de 6 MHz para cada canal analógico que possuam. Elas terão um prazo máximo de 2 anos para iniciar a transmissão digital neste novo canal.
- A transmissão analógica continuará ocorrendo, simultaneamente à digital, por um período de 10 anos a partir da publicação do decreto. Ao fim deste período as emissoras de TV devem devolver os canais utilizados para a transmissão analógica. A partir de julho de 2013 somente serão outorgados canais para a transmissão em tecnologia digital.
- Deverão ser consignados pelo menos quatro canais digitais para a exploração direta pela União Federal como canal do Poder Executivo, Canal de Educação, Canal de Cultura e Canal de Cidadania.

Essa preferência pelo padrão japonês era justificada pela capacidade do sistema atender a equipamentos portáteis, permitindo que o público assista TV a partir de dispositivos móveis, além de proporcionar alta definição e interatividade para terminais fixos e móveis.

3.3.3.1 – Sistema Brasileiro de TV Digital (SBTVD)

Os estudos do Comitê do Sistema Brasileiro de TV Digital foram concluídos foi apresentado o Sistema Brasileiro de TV Digital, o SBTVD, baseado no sistema japonês ISDB-T, com o acréscimo da letra “B” ao final, se tornando o ISDB-TB. Tal mudança deve-se às diversas implementações sofridas.

A principal delas sendo o “casamento” entre a base técnica de transmissão do sistema japonês com os padrões de compressão digital de áudio e vídeo introduzidos pelo Brasil, que são mais modernos e eficientes do que os adotados por outros padrões. O SBTVD adotou o padrão MPEG-4, também conhecido como H.264, para codificação de vídeo, e o HE-AAC v2 para o áudio.

O SBTVD adotou ainda tecnologias desenvolvidas pela Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio) e pela Universidade Federal da Paraíba (UFPB) que agregam importantes diferenciais que são a mobilidade e a interatividade. O *middleware* Ginga, adotado pelo SBTVD foi desenvolvido nos laboratórios da PUC-Rio e da UFPB e sua adoção possibilita a geração de empregos com mão de obra qualificada, incentivo à pesquisa e desenvolvimento, fortalecendo e expandindo o mercado de software e conteúdo interativo no Brasil, mercado esse de alto conteúdo tecnológico e de alto valor agregado.

O SBTVD realizou sua primeira transmissão no dia 02 de dezembro de 2007 e atualmente, segue o cronograma mostrado na figura 2.2.

## 3.4 – Middleware Ginga

“*Middleware* é uma camada de software posicionada entre o código das aplicações e a infraestrutura de execução” (SOARES, 2008). “Seu objetivo é oferecer às aplicações suporte necessário para seu rápido e fácil desenvolvimento, alem de esconder os detalhes das camadas inferiores, bem como a heterogeneidade entre os diferentes sistemas operacionais e hardwares, definindo, para os que produzem conteúdo, uma visão única do aparelho” (TELEMIDIA, 2007). É um software que age como um intermediador na comunicação entre as aplicações e o hardware sobre o qual elas executarão.

Desta forma, é atribuída ao *middleware* enorme responsabilidade, pois será ele o responsável pela comunicação entre as duas principais frentes da televisão: a indústria de produção de conteúdo, que é a responsável pelo desenvolvimento das aplicações para TV

Digital, e a indústria de *set-top-boxes*, ou conversores digitais. O insucesso na definição deste *middleware*, impossibilitaria o correto funcionamento da TV Digital no Brasil, já que o desenvolvimento de aplicações especificamente para determinados aparelhos seria absolutamente antidemocrático e culminaria na não disseminação da tecnologia da TV Digital. Assim, o SBTVD adotou o *middleware* Ginga.

No contexto da TV Digital, as aplicações podem ser particionadas em dois conjuntos: o das aplicações declarativas e o das aplicações procedurais. "Uma aplicação declarativa é aquela em que sua entidade "inicial" é do tipo "conteúdo declarativo". Analogamente, uma aplicação procedural é aquela em que sua entidade "inicial" é do tipo "conteúdo procedural"" (TELEMIDIA, 2007). E neste âmbito, o *middleware* age como a máquina de execução para ambos os tipos de aplicações, independentemente do *hardware* ou dos sistemas operacionais.

A figura 3.3 mostra a divisão em camadas do middleware Ginga:

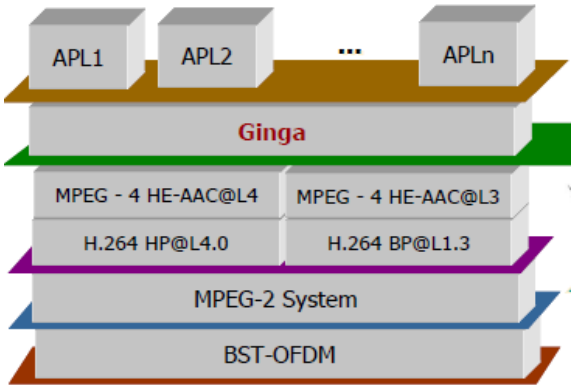


Figura 3.3: Camada Middleware Ginga

Fonte: (Soares e Barbosa, 2009)

Para o desenvolvimento de conteúdos declarativos para a TV Digital Brasileira, é utilizada a linguagem declarativa NCL. O conteúdo procedural é baseado em uma linguagem não declarativa que pode seguir diferentes paradigmas. A linguagem procedural mais usual em ambientes de TV Digital é o Java.

Para estes dois tipos de aplicações, o *middleware* Ginga dispõe de dois subsistemas lógicos responsáveis pelo processamento destas aplicações. O Ginga-NCL é o subsistema responsável pelo processamento das aplicações declarativas representadas por documentos NCL. O Ginga-J é o subsistema lógico que processa as aplicações procedurais desenvolvidas na linguagem Java.

### 3.4.1 – O Ambiente Declarativo do *Middleware* Ginga

Conforme foi dito, o subsistema lógico do *middleware* Ginga responsável pelo processamento dos documentos declarativos NCL que formam as aplicações declarativas é o Ginga-NCL. Pelo fato de a linguagem NCL ter sido desenvolvida nos laboratórios da PUC-Rio, o Ginga-NCL é uma das principais inovações adotadas pelo SBTVD em relação ao Sistema de TV Digital Japonês, o ISDB-T.

Ele tem por base a linguagem NCL que é uma linguagem de cola que tem como seu foco o sincronismo de mídias, a adaptabilidade e o suporte a múltiplos dispositivos e utiliza a, também brasileira, linguagem de script LUA para os trechos de código em que são necessárias intervenções procedurais.

O Ginga-NCL possui um módulo chave que merece destaque: Formatador NCL. Ele é o responsável por receber um documento NCL e controlar sua apresentação, fazendo com que as relações de sincronismo entre os objetos de mídia existentes sejam respeitadas (TELEMIDIA, 2007). Como as aplicações para TV Digital devem ter como função principal o sincronismo espacial e temporal dos diversos objetos de mídia que as comporão, é de extrema importância que a relação de sincronismo entre o fim da exibição de um vídeo e o início da exibição de outro vídeo, ou o inicio da exibição simultânea de duas imagens ocorra de forma fiel ao planejado para que a execução da aplicação seja bem sucedida.

Além disso, há acoplado ao Formatador NCL do Ginga-NCL, uma máquina virtual LUA. Ela fornece uma interface que permite as aplicações NCL trocarem dados com programas LUA que tornam as aplicações NCL capazes de utilizar-se do eficiente processamento de *scripts*.

Desta forma, fica mais simples de se entender o funcionamento do Formatador NCL: São gerados diversos eventos, como, por exemplo, apresentação de um vídeo ou texto, seleção de um botão ou uma imagem, durante as exibições dos objetos de mídia feitas pelos exibidores MPEG, JPEG, HTML. Estes eventos podem gerar ações em outros objetos de mídia, tais como parar, iniciar ou pausar suas apresentações. Ações essas que serão geradas pelo Formatador NCL quando os exibidores reportarem os eventos a ele. O Ginga define uma interface padrão que todo o exibidor acoplado ao sistema deve obedecer para reportar seus eventos e serem comandados por ações geradas pelo Formatador.

### 3.4.2 – O Ambiente Procedural do *Middleware* Ginga

Assim como existe o ambiente declarativo, o *middleware* Ginga dispõe de um subsistema lógico responsável pelo processamento das aplicações procedurais. Foi adotado, para este papel, o Ginga-J, que é baseado nas tecnologias Java (Máquina Virtual e algumas APIs). O Ginga-J adota uma série de inovações, mas também mantém a compatibilidade com a maioria dos *middlewares* dos sistemas de TV Digital do mundo.

Alguns importantes requisitos identificados para a TV Digital Brasileira não eram supridos por nenhum *middleware* existente no mundo, assim, para sanar este problema além de manter a compatibilidade com outras APIs, a definição Ginga-J é composta por APIs projetadas para suprir todas as funcionalidades necessárias para a implementação de aplicativos para televisão digital, desde a manipulação de dados multimídia até protocolos de acesso.

## 3.5 – PLC - Transmissão de dados pela Rede elétrica

Um dos grande desafios do Brasil, na questão social, é a inclusão digital para todas as classes sociais. Seja por dificuldades econômicas, ou por limitações de distância que não justificariam o esforço econômico para alcançar algumas tecnologias a municípios ou comunidades distantes, muitas pessoas ainda não sabem o que é a *internet*, por exemplo

Segundo a tabela 03 a seguir, disponível no Portal Teleco, diferentes fonte divergem quanto ao total de brasileiros com acesso a internet. Sabe-se porém que o número ainda está muito distante do total da população.

Tabela 03 - Usuários de Internet no Brasil

| **Milhões** | **2005** | **2006** | **2007** | **2008** | **2009** |
|---|---|---|---|---|---|
| **Fonte: PNAD** | 32,1 | | | 55,9 | 67,9 |
| **Fonte: TIC Domicílios** | | 35,3 | 44,9 | 53,9 | 63 |
| **Fonte Ibope** | | 32,5 | 39 | 62,3 | 66,3 |

Fonte: (TELECO, 2010)

Essas pessoas que não tem nenhum acesso à grande rede, encontram muitas dificuldades, hoje em dia, para se manter atualizadas com notícias e informações, ou até realizar importantes atividades. A Receita Federal, por exemplo, a partir de 2011, só aceitará declarações de imposto de renda enviadas pela *internet*, ou entregues em disquete (esta sendo uma mídia pouquíssimo usada).

A fim de resolver este problema, diversos projetos são estudados e novas tecnologias surgem com grande frequência. Uma delas já é realidade e tem grande potencial de se tornar popular, o PLC, que realiza a transmissão de dados utilizando como canal, ou meio, a rede elétrica.

O propósito de um sistema de comunicação é entregar uma mensagem de uma fonte de informação em um formato reconhecível a um usuário, com estes fisicamente separados. Para fazer isso, o transmissor modifica a mensagem para uma forma apropriada à transmissão através do canal. Essa modificação é realizada por um processo conhecido como modulação (HAYKIN, 2004).

Para que a mensagem possa ser enviada do remetente ao destinatário, é preciso que haja uma forma de tratamento do sinal para que ela possa trafegar pelo canal utilizado como forma de transmissão. O tratamento que o sinal recebe ao ser enviado é chamado de modulação, e o receptor recria a mensagem enviada pelo processo de demodulação. Para estes tratamentos, neste projeto são utilizados modems específicos para a tecnologia chamados *HomePlugs.*

Os avanços da tecnologia PLC só foram possíveis graças a avanços nas técnicas de modulação e demodulação do sinal. A presença de ruído e a distorção no sinal recebido impossibilitam a recriação exata da mensagem original. A degradação do sinal no sistema como um todo é influenciada pelo tipo de modulação usado, sendo “algumas técnicas mais sensíveis a ruídos e distorções que outras”(VARGAS, 2004).

E para a combinação de diferentes sinais para a transmissão simultânea sobre o mesmo canal, deve ser atribuído, ainda, um outro tratamento a este sinal chamado de multiplexação. Existe diferentes métodos de multiplexação de sinal. Dentre os básicos podem ser citados:

- F*requency-Division Multiplexing* (FDM): Multiplexação por divisão de frequência. Divide o sinal em diferentes frequências e coloca cada sinal em uma frequência específica.
- *Time-Division Multiplexing* (TDM): “usa modulação por pulsos para posicionar os sinais em diferentes fatias de tempo”(VARGAS, 2004”)

- *Code-Division Multiplexing* (CDM): “no qual cada sinal é identificado por uma seqüência (código) diferente”(VARGAS, 2004).

As técnicas mais utilizadas na tecnologia PLC são baseadas nessas técnicas de multiplexação citadas:

- *Orthogonal Frequency-Division Multiplexing (*OFDM): técnica utilizada pelos dispositivos *HomePlug* utilizados neste projeto.
- *Spread Spectrum;*

Estas técnicas de modulação e multiplexação são utilizadas no intuito de que a tecnologia PLC supere os principais problemas que enfrenta:

- Ruído;
- Atenuação;
- Distorção.

A figura 3.4 mostra os principais problemas enfrentados pela transmissão de dados sobre a rede elétrica:

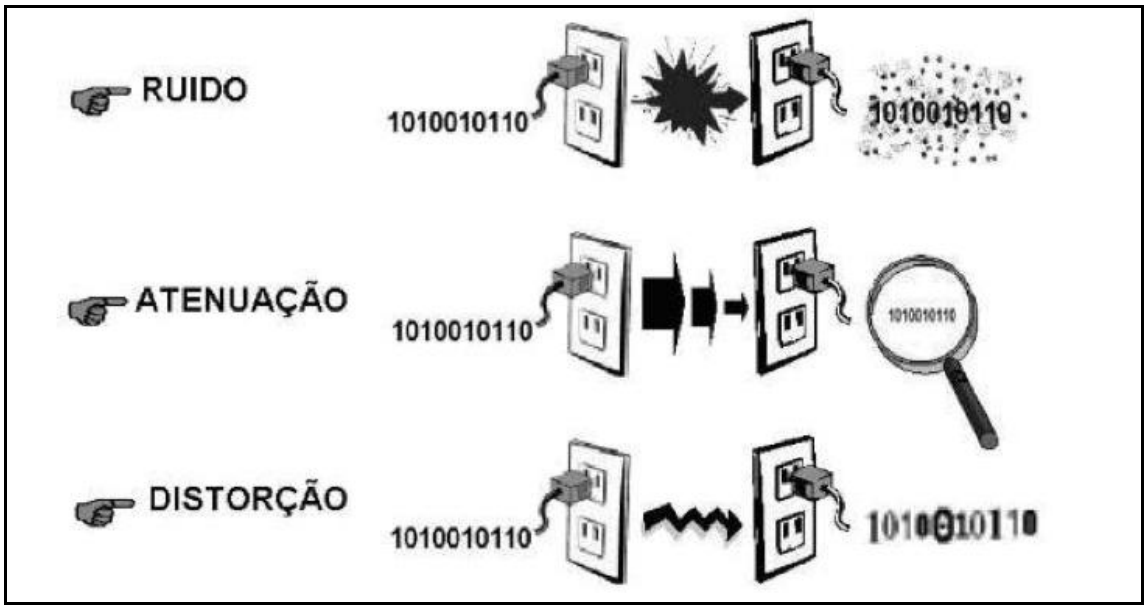


Figura 3.4: Principais obstáculos enfrentados na transmissão de dados

Fonte: (DUQUE, 2001).

Entende-se por ruído todos os sinais presentes no canal e que não transportam informação útil. Assim, como a freqüência utilizada pelo PLC varia de 1 a 30 MHz, existem

vários aparelhos geradores de ruídos capazes de dificultar  a transmissão de dados na rede elétrica.

A atenuação é a propriedade do sinal diminuir sua amplitude durante a propagação. Os três principais fatores causadores da atenuação são o material do cabo utilizado, a freqüência do sinal e a distância percorrida. Quanto maior a distância e a freqüência, maior é a atenuação do sinal. A atenuação do sinal pode, ainda, variar com o tempo, devido ao ligamento e desligamento de aparelhos na rede elétrica. Na rede elétrica, diferentemente das outras redes em geral, o fator predominante para a atenuação é a indutância do cabo utilizado e não a capacitância, pois as impedâncias dos aparelhos que são ligadas à rede são em geral menores que a impedância característica do cabo. A atenuação não é um grande fator de empecilho para a propagação do sinal, pois é possível aumentar em certa faixa a amplitude do sinal a ser transmitido para contornar seus efeitos. Se, entretanto, a atenuação for muito grande, uma solução seria aumentar muito o nível do sinal. Isso, entretanto é inviável, pois acarreta problemas relacionados à superação do nível de emissão que é regulamentado para as redes PLC (TAVEIRA, 2004).

A distorção é causada por vários fatores sendo, o principal, as múltiplas reflexões de sinais que ocorrem pelo descasamento que existe entre as várias partes da rede elétrica. Esse fator é extremamente importante em redes de baixa tensão, pois existe um número muito grande de ligações sendo praticamente todas descasadas. Uma tomada não sendo utilizada, por exemplo, funciona como um estube em aberto. Como há uma grande distorção, a resposta em freqüência passa a ser não linear (TAVEIRA, 2004).

### 3.5.1 – OFDM

Multiplexação é a técnica que possibilita que vários sinais possam ser enviados ao mesmo tempo e em um mesmo canal. Existem diferentes técnicas sendo as mais importantes a multiplexação por divisão em freqüência, por divisão no tempo e por divisão em códigos.

A técnica FDM permite que múltiplos sinais trafeguem numa faixa de freqüência única. Uma variação dessa tecnologia é a técnica OFDM que consiste em modular um grande número de portadoras de banda estreita distribuídas lado a lado. A figura 3.5 ilustra as diferenças entre as duas técnicas de modulação. No sistema FDM, as portadoras estão suficientemente espaçadas de modo a poderem ser recebidas utilizando filtros convencionais. Entretanto, para tornar a filtragem possível, bandas de guarda têm que ser introduzidas entre

essas portadoras, o que resulta em uma diminuição da eficiência espectral. Por outro lado, na OFDM, ao invés de se utilizar uma banda de guarda entre subportadoras para poder separá-las na recepção, emprega-se uma sobreposição das mesmas, resultando em um ganho espectral de até de 50% em relação à técnica FDM.

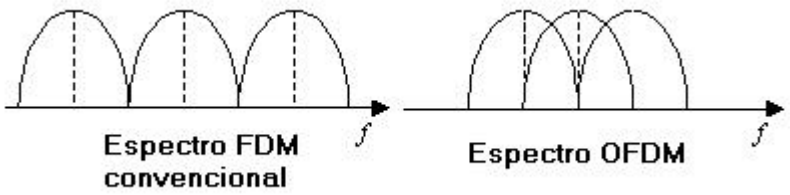


Figura 3.5: Modulação FDM e OFDM
Fonte: (PINHEIRO, 2005)

A utilização de modulação por multi-portadoras ortogonais tem recebido grande atenção nos sistemas de telecomunicação. Na tecnologia PLC, a técnica OFDM controla as freqüências em tempo real, com o sistema alternando o carregamento dos sinais de acordo com a presença ou não de ruídos.

Para aproveitar as melhores condições possíveis do link, os sinais são carregados e transmitidos em várias freqüências simultâneas e com níveis de carregamento diferentes. Assim, altas taxas de transmissão, boa performance e confiabilidade são garantidos. Com isso, o sistema pode facilmente se adaptar às mudanças das condições de transmissão da rede elétrica, podendo ainda utilizar filtros para a proteção de serviços especialmente sensíveis a esses tipos de interferências.

### 3.5.2 – PLC e TV Digital em Barreirinhas

A cerca de 270 Km de São Luiz, capital do Maranhão, com aproximadamente 40000 habitantes, Barreirinhas foi escolhida pela Comissão de Inclusão Digital do Fórum Aptel PLC Brasil como a cidade que receberá o projeto piloto de PLC.

O projeto, coordenado pela FITec Inovações Tecnológicas, visa estabelecer um modelo de conectividade para os principais alvos sociais, em localidades que além da falta de infraestrutura tem o problema do baixo nível de renda da população, proporcionando assim,

acesso às populações de baixa renda a serviços de tecnologia moderna, ampliando a participação na distribuição da mesma através do acesso a informação. Este tipo de investimento acaba trazendo grande benefícios para áreas chaves como a educação. O projeto potencializa o uso de recursos educacionais como no caso da TV Escola.

O projeto pretende “iluminar” (termo utilizado para a inserção dos sinais de dados na rede elétrica transformando-a em uma internet) 14 pontos de conectividade: 01 Oficina de artesanato, 03 escolas, 01 biblioteca, 05 estabelecimentos de saúde, além da prefeitura, secretarias da educação, saúde e trabalho e ação social. Aderbal Borges, Diretor de Desenvolvimento de Negócios da FITec, comenta: "Em Barreirinhas, 14 serão os locais de prestação de serviços públicos: escolas, bibliotecas, postos de saúde, oficinas de artesanato e secretarias municipais, interconectadas através de tecnologias PLC. Eles terão acesso à Internet através de conteúdos programáticos específicos para cada ponto de conectividade, de acordo com as respectivas vocações. Dessa forma, esta experiência se mostra perfeitamente condizente com o Serviço de Comunicação Digital (SCD) e com os Programas de Inclusão Digital em planejamento pelo Governo" (CHEROBINO, 2008).

Além de oferecer oportunidades na área da educação, por exemplo, o projeto piloto do PLC em Barreirinhas deu a oportunidade da cidade ser a primeira cidade brasileira a ter TV Digital com interatividade utilizando a rede elétrica como canal de retorno. Como benefício para a população pode ser citado o seguinte exemplo: enquanto assiste a telenovela, o telespectador pode acessar uma aplicação e verificar se há disponibilidade de horário para ir a uma consulta médica em determinado dia. A iniciativa conta com conteúdo local fornecido por empresas como SEBRAE e a prefeitura da cidade (CHEROBINO, 2008).

### 3.6 – Automação residencial

Os constantes avanços tecnológicos e a infindável busca pela melhoria da qualidade de vida, reduzindo o trabalho doméstico, aumentando o bem estar e a segurança dos habitantes e uma utilização racional e planejada dos diversos meios de consumo faz com que o ramo da automação residencial esteja sempre em crescimento.

A automação residencial pode se dar por meio de inteligência artificial, quando por exemplo um sensor verifica a incidência dos raios de sol na janela e automaticamente envia um comando para que a persiana feche, ou a automação pode ocorrer aplicando-se técnicas

que executem uma atividade determinada por um ser humano de forma automática, por exemplo enviar, de um dispositivo móvel, o comando para apagar uma lâmpada.

O termo domótica, quee também pode ser referenciado por expressões como "*smart building*", "*intelligent building*", "edifícios inteligentes", procura uma melhor integração através da automatização nas áreas de segurança, de comunicação e de controle, e gestão de recursos.

Dado o grande apelo e o caos ambiental em que vivemos, a automação residencial não está ligada apenas a busca por maior conforto diminuindo as atividades domésticas. Ela está ligada ao uso consciente e eficaz da energia e dos demais recursos naturais.

A TV Digital, uma tecnologia recente e com muito a se descobrir na questão das aplicações, graças a sua interatividade, aparece com um grande potencial para centro de controle de automação em uma residência. Por ser um aparelho bem conhecido pela sociedade, seu uso não requer muito tempo de treinamento e sua presença nos lares é grande.

### 3.7 – Normas ABNT

Conforme estabelece o decreto 5.820 de 29 de junho de 2006, o Fórum Brasileiro de TV Digital solicitou à ABNT, através do seu Módulo Técnico, a constituição da CEET de TV Digital, que elaborasse as normas que atendessem ao SBTVD.

Foram criados, então, grupos de trabalho em cada uma das frentes de especificações estabelecidas pelo Módulo Técnico. Esses grupos de trabalho criaram as seguintes normas:

- Transmissão:
  - ABNT NBR 15601
- Codificação:
  - ABNT NBR 15602 – Parte 1
  - ABNT NBR 15602 – Parte 2
  - ABNT NBR 15602 – Parte 3
- Multiplexação:
  - ABNT NBR 15603 – Parte 1
  - ABNT NBR 15603 – Parte 2
  - ABNT NBR 15603 – Parte 3
- Receptores:

    - ABNT NBR 15604
- Segurança:
    - ABNT NBR 15605
- Middleware:
    - ABNT NBR 15606 – Parte 1
    - ABNT NBR 15606 – Parte 2
    - ABNT NBR 15606 – Parte 3
    - ABNT NBR 15606 – Parte 4
    - ABNT NBR 15606 – Parte 5
- Canal de Interatividade:
    - ABNT NBR 15607
- Guia de Operação:
    - ABNT NBR 15608 – Parte 1
    - ABNT NBR 15608 – Parte 2
    - ABNT NBR 15608 – Parte 3

# CAPÍTULO 4 – PROTÓTIPO DE AUTOMAÇÃO RESIDENCIAL

Este trabalho consiste no desenvolvimento de um protótipo de automação residencial utilizando um *set-top-box* virtual baseado no modelo brasileiro de TV Digital como ambiente de controle dos dispositivos domésticos a serem automatizados e a tecnologia PLC para comunicação entre o middleware e o dispositivo eletrônico contendo os LEDs. Por se tratar de um protótipo foram utilizados quatro LEDs que simulam qualquer outro dispositivo que se deseje controlar.

## 4.1 – Apresentação Geral do Protótipo

O protótipo pode ser dividido em dois diferentes módulos que devem ser interligados:

- uma máquina virtual Ginga-NCL *Virtual Set-top-box*, utilizando o *software VMware Fusion* versão 3.1.0 instalado em um notebook, que simula o funcionamento de um *set-top-box* real;
- um kit de desenvolvimento ACEPIC NET com módulo *ethernet* fabricado pela ACEPIC que utiliza um microcontrolador 18F4620 e um microcontrolador *ethernet* ENC28J60, além dos quatro LEDs a serem controlados.

Esses módulos são interligados através de uma régua, ligada à rede elétrica da companhia fornecedora de energia por uma tomada convencional. Esta régua simula a rede elétrica de uma residência. Além disso, nos pontos da máquina virtual e do kit de desenvolvimento, com cabos UTP, são conectados dispositivos *HomePlugs* que tratam o sinal para que possam trafegar na rede elétrica.

A figura 4.1 a seguir mostra a topologia do protótipo:

A: Rede elétrica da Companhia Energética de Brasília;

B: Régua elétrica;

C: *HomePlugs* TL-PA201;

D: Kit de desenvolvimento ACEPIC NET com os LEDs a serem acionados;

E: Ginga-NCL Set-top-box;

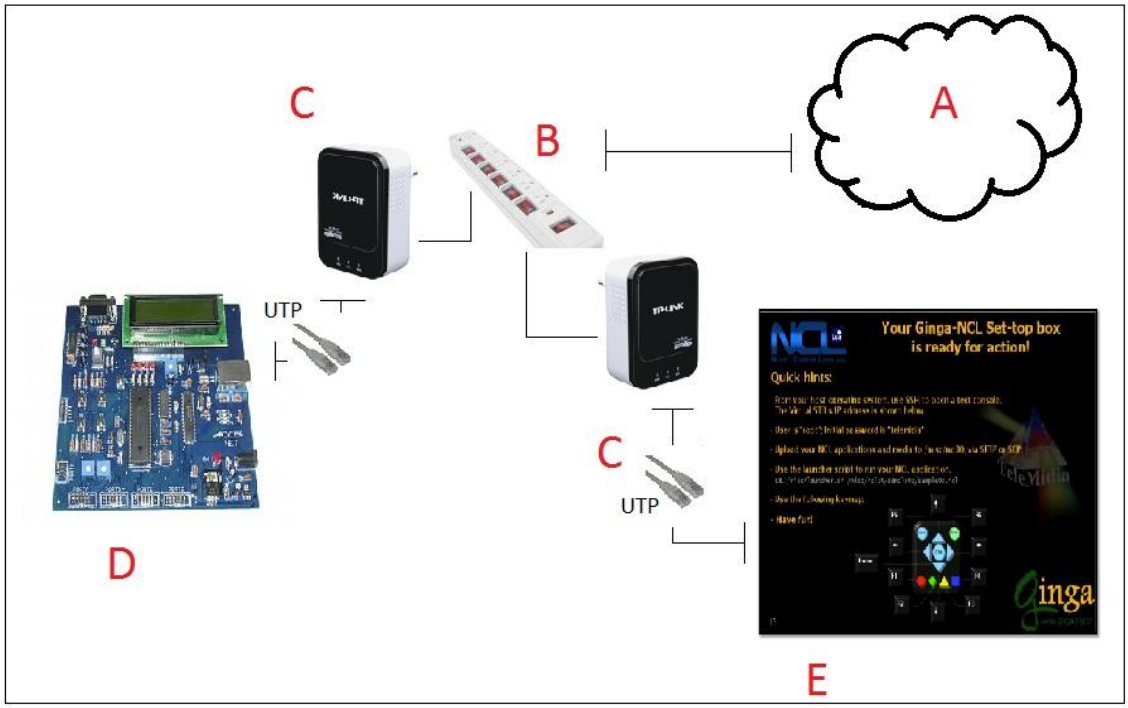


Figura 4.1: Topologia do protótipo

Fonte: (O autor)

Foram selecionados dois vídeos promocionais chamados “Momento UniCEUB” que são exibidos em *loop* simulando uma transmissão das emissoras de televisão. Durante a exibição dos vídeos, é mostrado um ícone interativo (i) na cor vermelha, indicando que a seleção do botão de cor vermelha do controle remoto, simulado pela tecla [F1] no teclado do computador inicia a aplicação interativa.

## 4.2 – Custos

Como todo projeto, o desenvolvimento deste envolveu custos na aquisição do kit de desenvolvimento e os *HomePlugs*. A tabela 04 mostra os custos envolvidos:

Tabela 04 - Custos envolvidos na implementação do protótipo

| Equipamento | Quantidade | Custo total (R$) |
|---|---|---|
| Adaptadores HomePlug para tecnologia PLC, modelo TL-PA201 200MBPS | 2 | 278,00 |
| Kit de Desenvolvimento ACEPIC NET c/ módulo Ethernet | 1 | 360,00 |
| Total de equipamentos | 4 | 638,00 |

Fonte: (O autor)

## 4.3 – Implementação do protótipo

Para o desenvolvimento do protótipo foram seguidas as seguintes etapas:

1. Instalação da máquina virtual Ginga-NCL *Set-top-box*;
2. Desenvolvimento da interface gráfica de automação residencial para TV Digital utilizando a linguagem NCL dentro dos padrões do Sistema Brasileiro de TV Digital;
3. Implementação dos *scripts* Lua para conexão com o kit de desenvolvimento dentro dos padrões do Sistema Brasileiro de TV Digital;
4. Implementação do servidor web, desenvolvido na linguagem C pela ACEPIC Tecnologia e Treinamento, no kit de desenvolvimento;
5. Implantação de todo o protótipo utilizando os *HomePlugs* TL-PA201 interligando todos os módulos do protótipo;

## 4.4 – Os módulos do protótipo

### 4.4.1 – Máquina Virtual

Uma máquina virtual é a implementação em *software* que utiliza os recursos de *hardware* de uma máquina hospedeira e, conforme a configuração, simula de forma fiel um outro computador real. Assim, para facilitar a distribuição e a implementação do Ginga-NCL, a equipe do Laboratório TeleMídia da PUC-Rio criou e configurou a máquina fedora-fc7-ginga-i386 que simula de forma bastante semelhante uma implementação do Ginga-NCL embarcada em *set-top-boxes* reais. A figura 4.2 mostra a configuração de *hardware* da máquina virtual fedora-fc7-ginga-i386:

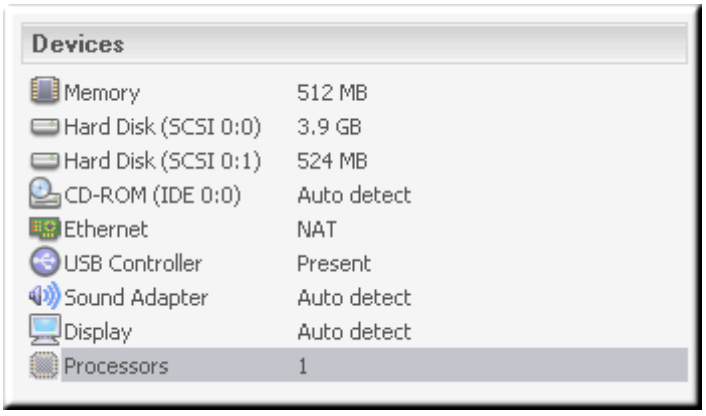


Figura 4.2: Configuração de hardware da máquina virtual fedora-fc7-ginga-i386

Fonte: (PORTAL DO SOFTWARE PÚBLICO BRASILEIRO, 2010)

Nesta máquina virtual foi configurada a distribuição Fedora Core 7 do sistema operacional Linux. A instalação foi otimizada incluindo apenas os pacotes de *software* essenciais para o desenvolvimento do *middleware* Ginga e para a execução do Ginga-NCL versão C++, a versão do player NCL que conta com os mais avançados recursos de apresentação de aplicações declarativas.

Para executar esta máquina virtual, foi utilizado o *software* de virtualização *VMWare Fusion* versão 3.1.0. O computador utilizado para o protótipo foi um *notebook* MacBook Pro com o sistema operacional Mac OS X, versão 10.6.4. A figura 4.3 mostra a configuração do notebook utilizado para executar a máquina virtual fedora-fc7-ginga-i386:

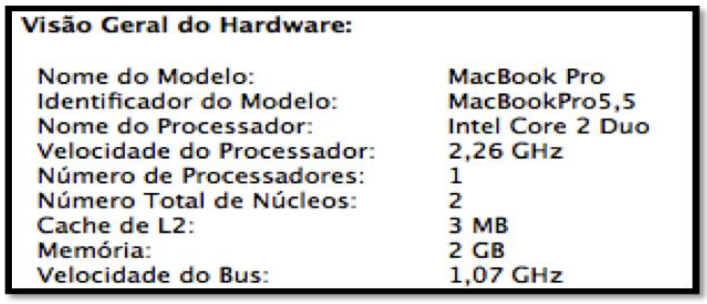


Figura 4.3: Configuração de *hardware* do notebook MacBook Pro

Fonte: (O autor)

4.4.1.1 – Código NCL

O documento NCL é o responsável por exibir na tela do *notebook*, que está simulando o *set-top-box* e a televisão, a interface gráfica da aplicação interativa para que o usuário possa enviar os comandos de acender ou apagar os LEDs presentes no kit de desenvolvimento.

Para o desenvolvimento dos códigos NCL foi utilizado o software Eclipse SDK na versão 3.6.0 instalado no *notebook* MacBook Pro, conforme mostrado na figura 4.4:

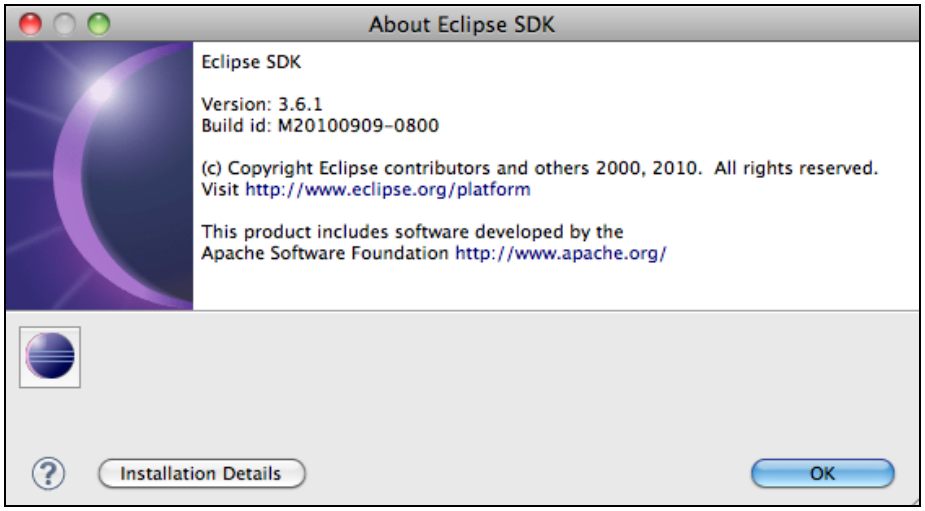


Figura 4.4: Eclipse SDK versão 3.6.0

Fonte: (Menu do *software* Eclipse SDK)

Os códigos estão organizados em um diretório principal chamado *ProjetoFinal*. Dentro deste diretório estão os arquivos NCL *masterConnectorBase.ncl*, que contém a base de conectores, que será explicado mais a frente, e o arquivo *projFinal_nclMain.ncl*, que é o arquivo principal do projeto, pois ele contém as instruções para exibição da interface gráfica da aplicação interativa. Há ainda os diretórios *media*, que contém as mídias (imagens e vídeos) exibidas durante a execução da aplicação NCL, e *lua* com os *scripts* Lua responsável pela conexão do *set-top-box* com o kit de desenvolvimento e o envio do comando para acender ou apagar os LEDs.

O arquivo *projFinal_nclMain.ncl*, que pode ser visto em detalhes no APÊNDICE A, é o principal documento NCL, pois é o responsável por exibir o ícone interativo (i) que permite ao telespectador que tenha acesso à aplicação que oferece os controles de acender ou apagar os LEDs.

Este arquivo, assim como todos arquivos NCL, possui algumas estruturas pré definidas que devem ser seguidas.

4.4.1.1.1 – Cabeçalho

Possui duas *tags* responsáveis por determinar informações básicas em um documento NCL:

1. *<?xml version="1.0" encoding="ISO-8859-1"?>*: esta *tag* determina a versão 1.0 do xml e a codificação ISO-8859-1 que corresponde a utilização de caracteres ocidentais.
2. *<ncl id="projFinal_nclMain" xmlns="http://www.ncl.org.br/NCL3.0/EDTVProfile">*: aqui inicia-se o documento NCL. Esta *tag* é chamada de nó NCL e possui dois atributos: '*id'* que é utilizado para identificar a aplicação, e o '*xlmns'* que identifica o perfil NCL que está sendo usado no documento.

4.4.1.1.2 – Regiões

Após o cabeçalho, são definidas as regiões de exibição das mídias na tela de exibição. As regiões são definidas dentro da *tag* <*regionBase*> que deve ser criada dentro da *tag* <*head*> do documento NCL.

Foram definidas as seguintes regiões que serão utilizadas para exibição das mídias durante a exibição da aplicação:

- *rgMain*: é a região principal da aplicação e determina a área total da tela;
- *rgVideo*: é a região de exibição dos vídeos que simulam a programação das emissoras;
- *rgBtnInterativo*: indica o local da tela onde é exibido o ícone interativo (i);
- *rgMenuInterativo*: região onde é exibido o botão Automação Residencial;
- *rgBtnSair*: região onde é exibido o botão sair;
- *rgBtnLampada1*: região onde é exibido o botão que indica o primeiro LED;
- *rgBtnAcenderLampada1*: região onde é exibido o botão de acender o primeiro LED;
- *rgBtnApagarLampada1*: região onde é exibido o botão de apagar o primeiro LED;
- *rgBtnLampada2*: região onde é exibido o botão que indica o segundo LED;
- *rgBtnAcenderLampada2*: região onde é exibido o botão de acender o segundo LED;
- *rgBtnApagarLampada2*: região onde é exibido o botão de apagar o segundo LED;

- *rgBtnLampada3*: região onde é exibido o botão que indica o terceiro LED;
- *rgBtnAcenderLampada3*: região onde é exibido o botão de acender o terceiro LED;
- *rgBtnApagarLampada3*: região onde é exibido o botão de apagar o terceiro LED;
- *rgBtnLampada4*: região onde é exibido o botão que indica o quarto LED;
- *rgBtnAcenderLampada4*: região onde é exibido o botão de acender o quarto LED;
- *rgBtnApagarLampada4*: região onde é exibido o botão de apagar o quarto LED;

4.4.1.1.3 – Descritores

Os descritores, como o próprio nome diz, descrevem a apresentação de cada mídia exibida durante a aplicação. Após as *tags* de criação das regiões, foi criada a *tag* <*descriptorBase*> que definem os descritores, ligando-os às regiões, e seus atributos.

São definidos os atributos *focusIndex*, que define a ordem de foco de cada mídia durante a execução da aplicação, *focusBorderColor*, que define a cor da borda da mídia selecionada, e *moveDown* e *moveUp*, que determinam a ordem de navegação das mídias pois se trata de um menu vertical.

As figuras 4.5, 4.6 e 4.7 mostram os descritores criados no documento principal da aplicação:

```
<!--descritor do vídeo-->
<descriptor id="dVideo" region="rgVideo"/>
<!--descritor do botão de interatividade-->
<descriptor id="dBtnInterativo" region="rgBtnInterativo"/>
<!--descritor do menu de interatividade-->
<descriptor id="dMenuInterativo" region="rgMenuInterativo" focusIndex="1" focusBorderColor="gray" focusSelSrc="2"/>
<!--descritor do botão para sair da interatividade-->
<descriptor id="dBtnSairInteratividade" region="rgBtnSair"/>
<!--descritor do botão para sair da lampada1-->
<descriptor id="dBtnSairLampada1" region="rgBtnSair"/>
<!--descritor do botão para sair da lampada2-->
<descriptor id="dBtnSairLampada2" region="rgBtnSair"/>
<!--descritor do botão para sair da lampada3-->
<descriptor id="dBtnSairLampada3" region="rgBtnSair"/>
<!--descritor do botão para sair da lampada4-->
<descriptor id="dBtnSairLampada4" region="rgBtnSair"/>
```

Figura 4.5: Descritores

Fonte:(O autor)

```
<!--descritor do botão lampada1-->
<descriptor id="dBtnLampada1" region="rgBtnLampada1" focusIndex="2" focusBorderColor="gray" moveDown="3" focusSelSrc="5"/>
<!--descritor do botão lampada2-->
<descriptor id="dBtnLampada2" region="rgBtnLampada2" focusIndex="3" focusBorderColor="gray" moveDown="4" moveUp="2"/>
<!--descritor do botão lampada3-->
<descriptor id="dBtnLampada3" region="rgBtnLampada3" focusIndex="4" focusBorderColor="gray" moveDown="5" moveUp="3"/>
<!--descritor do botão lampada4-->
<descriptor id="dBtnLampada4" region="rgBtnLampada4" focusIndex="5" focusBorderColor="gray" moveUp="4"/>
```

Figura 4.6: Descritores

Fonte:(O autor)

```
<!--descritor do botão AcenderLampada1-->
<descriptor id="dBtnAcenderLampada1" region="rgBtnAcenderLampada1" focusIndex="6" focusBorderColor="gray" moveDown="7"/>
<!--descritor do botão ApagarLampada1-->
<descriptor id="dBtnApagarLampada1" region="rgBtnApagarLampada1" focusIndex="7" focusBorderColor="gray" moveUp="6"/>
<!--descritor do botão AcenderLampada2-->
<descriptor id="dBtnAcenderLampada2" region="rgBtnAcenderLampada2" focusIndex="8" focusBorderColor="gray" moveDown="9"/>
<!--descritor do botão ApagarLampada2-->
<descriptor id="dBtnApagarLampada2" region="rgBtnApagarLampada2" focusIndex="9" focusBorderColor="gray" moveUp="8"/>
<!--descritor do botão AcenderLampada3-->
<descriptor id="dBtnAcenderLampada3" region="rgBtnAcenderLampada3" focusIndex="10" focusBorderColor="gray" moveDown="11"/>
<!--descritor do botão ApagarLampada3-->
<descriptor id="dBtnApagarLampada3" region="rgBtnApagarLampada3" focusIndex="11" focusBorderColor="gray" moveUp="10"/>
<!--descritor do botão AcenderLampada4-->
<descriptor id="dBtnAcenderLampada4" region="rgBtnAcenderLampada4" focusIndex="12" focusBorderColor="gray" moveDown="13"/>
<!--descritor do botão ApagarLampada4-->
<descriptor id="dBtnApagarLampada4" region="rgBtnApagarLampada4" focusIndex="13" focusBorderColor="gray" moveUp="12"/>
```

Figura 4.7: Descritores

Fonte:(O autor)

4.4.1.1.4 – Conectores

Os conectores, definidos na tag <*connectorBase*>, definem a ordem de exibição das mídias por meio de ação sobre condição. Ou seja, quando algo ocorre, é disparada uma ação.

Para este projeto, foi gerado o arquivo *masterConnectorBase.ncl* armazenado no mesmo diretório *ProjetoFinal*. O código deste arquivo pode ser visto em detalhes no APÊNDICE B. Este arquivo possui uma série de conectores que serão utilizados pelas *tags* <*link*> no documento principal *projFinal_nclMain.ncl*. Porém, para que esses conectores possam ser utilizados pelo documento principal da aplicação NCL, é necessário que ele seja carregado dentro da *tag* <*connectorBase*>, pelo atributo *documentURL*.

4.4.1.1.5 – Mídias

As mídias são os arquivos de imagem e vídeo que serão exibidos durante a execução da aplicação. Os nós de mídia são criados dentro da *tag* <*body*>, que já é o corpo do documento, e devem definir alguns atributos:

- *id*: nomeia o nó de mídia;
- *type*: determina o tipo de mídia do nó, se é imagem, vídeo ou áudio e também informa o formato do arquivo;
- *src*: informa o caminho onde o arquivo está armazenado;
- *descriptor*: define o descritor responsável pelo nó de mídia.

Foram criados dois nós de mídia para os vídeos que são exibidos em *loop*, um nó de mídia para cada imagem (*.jpg*), que funcionam como os botões da aplicação.

A figura 4.8 abaixo mostra os nós de mídia de vídeo:

```
<!--nós de mídia de Vídeo-->
<media id="video1" type="video/mpeg" src="media/video1.mpg" descriptor="dVideo"/>
<media id="video2" type="video/mpeg" src="media/video2.mpg" descriptor="dVideo"/>
```

Figura 4.8: Mídias de vídeo

Fonte: (O autor)

4.4.1.1.6 – Porta

A *tag* <*port*> indica o início da aplicação. É definida, nesta *tag*, qual a primeira mídia a ser exibida, logo no inicio da execução da aplicação NCL.

4.4.1.1.7 – *Links*

Os *links* utilizam a base de conectores carregada do documento *masterConnectorBase.ncl* para definir o fluxo de execução da aplicação. Ou seja, nas *tags* <*link*> são definidas as ações a serem executadas em cada condição. Por exemplo, é usado o conector *onEndStart* para a execução do *loop* dos vídeos, pois o *link lEndVideo1StartVideo2Loop* indica que quando for concluída a exibição do arquivo de mídia *video1.mpeg*, deverá ser iniciada a exibição do arquivo de mídia *video2.mpeg*.

4.4.1.2 – Código Lua

Os *scripts* desenvolvidos na linguagem Lua utilizados no projeto são responsáveis por realizar a conexão com o kit de desenvolvimento e enviar a ele os comandos de acender ou apagar o LED selecionado pelo usuário. Eles são chamados quando os botões Acender ou Apagar de cada LED foram selecionados. Os *scripts* foram desenvolvidos utilizado o *software* Eclipse SDK na versão 3.6.0 instalado no *notebook* MacBook Pro. A figura 4.9 mostra o *script* Lua desenvolvido para enviar o comando que acende o LED 1, o script pode ser visto em detalhes no APÊNDICE C:

```
require "util"
require "http"


local function enviaComandoLed(endKitDesn)
    --Variável que recebe a resposta
    local resposta = http.request(endKitDesn, "GET")
    --imprime a resposta do servidor
    if resposta then
       print("\n\n\n--------------------", resposta, "\n\n\n")
    end
end

util.coroutineCreate(enviaComandoLed, "http://192.168.0.104/?lcd=LED 1 ON&led1=1")
```

Figura 4.9: *Script* acende1.lua

Fonte: (O autor)

O *script* utiliza os módulos *http.lua* e *util.lua*, desenvolvidos pelo professor Manoel Campos da Silva Filho, professor do Instituto Federal de Educação, Ciência e Tecnologia do Tocantins e Mestrando em Engenharia Elétrica na Universidade de Brasília, na Área de TV Digital, disponíveis em http://manoelcampos.com. Além destes módulos, utiliza ainda a classe *tcp.lua* disponibilizada no portal do laboratório Telemidia da PUC-Rio, desenvolvedora da linguagem de programação Lua, e o módulo de conversão de/para Base 64 disponível na comunidade *lua-users* (http://lua-users.org/wiki/BaseSixtyFour).

O *script* realiza uma conexão com o servidor *web*, embarcado no kit de desenvolvimento, enviando o método *GET* do protocolo *HTTP* pela *url* *http://192.168.0.104/?lcd=LED 1 ON&led1=1*. O parâmetro *lcd=LED 1 ON* exibe no *display* LCD do kit de desenvolvimento a mensagem *LED 1 ON* (LED 1 ACESO), e o parâmetro *led1=1* envia o comando para acender o LED 1. Caso o parâmetro fosse *led1=0*, por exemplo, o LED 1 seria apagado, conforme mostra a figura 4.10, que mostra o *script* responsável por apagar o LED 1, que pode ser visto em detalhes no APÊNCIDE G:

```
require "util"
require "http"


local function enviaComandoLed(endKitDesn)
    --Variável que recebe a resposta
    local resposta = http.request(endKitDesn, "GET")
    --imprime a resposta do servidor
    if resposta then
       print("\n\n\n--------------------", resposta, "\n\n\n")
    end
end

util.coroutineCreate(enviaComandoLed, "http://192.168.0.104/?lcd=LED 1 OFF&led1=0")
```

Figura 4.10: *Script* apaga1.lua

Fonte: (O autor)

4.4.2 – O kit de desenvolvimento ACEPIC NET

Kits de desenvolvimento são circuitos eletrônicos projetados tendo em vista a utilização de vários periféricos em uma só placa. O kit de desenvolvimento ACEPIC NET, fabricado pela ACEPIC Tecnologia e Treinamento representa, no protótipo deste projeto, a central de inteligência dos dispositivos domésticos. Ele utiliza um PIC 18F4620 e baseia-se na comunicação *ethernet* usando o controlador ENC28J60, sendo ambos PIC e controlador *ethernet* fabricados pela Microchip, além de quatro LEDs representando quatro dispositivos domésticos.

Os LEDs presentes no kit foram nomeados de L1 à L4 e estão conectados à porta B do microcontrolador nos pinos RB4, RB5, RB6 e RB7, conforme pode ser visto na figura 4.11 que representa o esquema elétrico da placa:

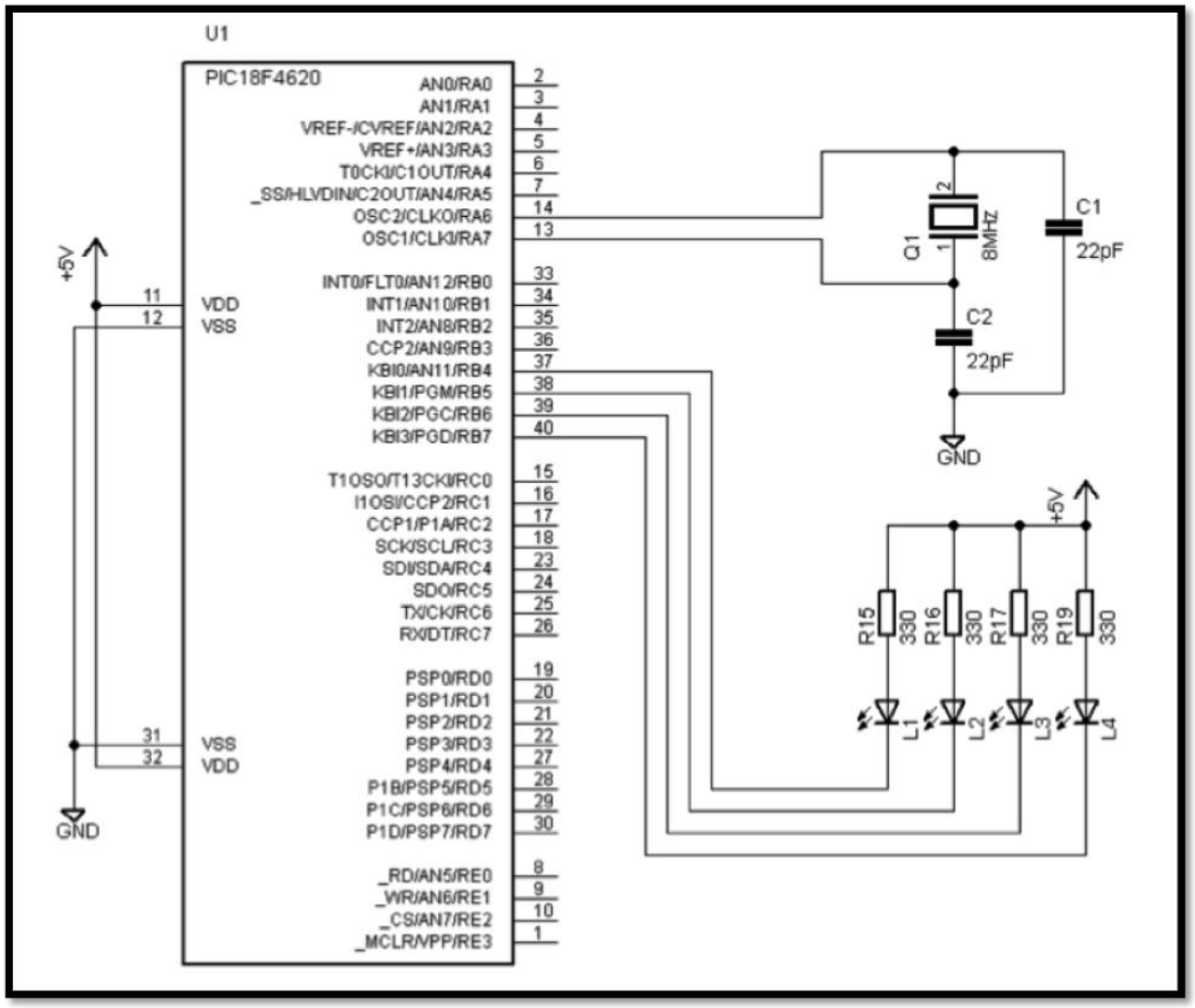


Figura 4.11: Ligação dos LEDs ao PIC
Fonte: (Manual do kit de desenvolvimento ACEPIC NET)

O LCD tem conectado seus pinos de controle na porta E do microcontrolador, sendo RE0 para o pino RS e RE1 para o pino EM e seus pinos de dados são conectados nos pinos RD4, RD5, RD6 e RD7. A figura 4.12 mostra a ligação do display LCD ao PIC 18F4620:

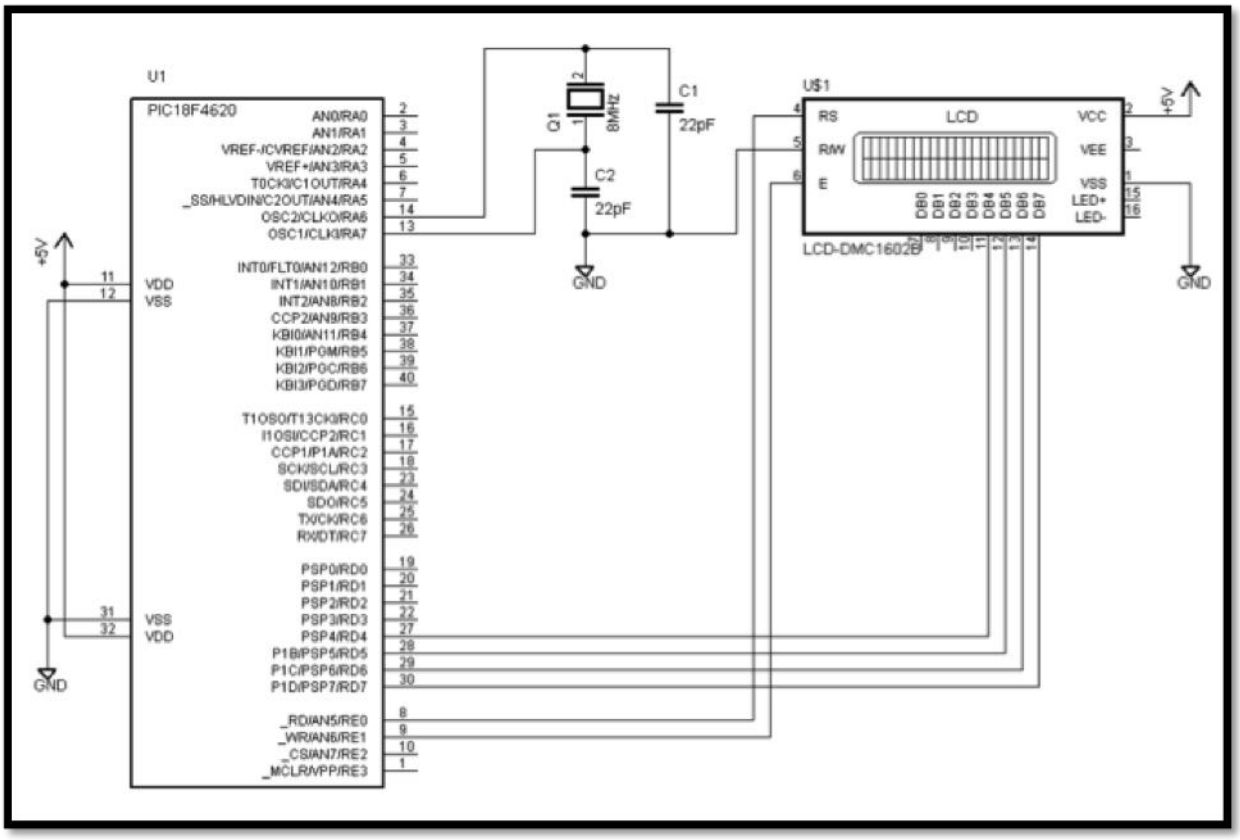


Figura 4.12: Ligação do display LCD ao PIC
Fonte: (Manual do kit de desenvolvimento ACEPIC NET)

O módulo *ethernet* ENC28J60 e a memória 25LC256 são conectados ao microcontrolador através dos pinos RC3 (SCK), RC4 (SO) e RC5 (SI) sendo o pino CS para o módulo conectado ao RD1 e o pino CS da memória conectado ao RB3 do PIC.

O kit de desenvolvimento é alimentado por uma fonte de 12V e 400mA de corrente.

4.4.2.1 – Código C do Microcontrolador

A programação do kit de desenvolvimento ACEPIC NET com o microcontrolador PIC 18F4620 e o microcontrolador *ethernet* ENC28J60 foi feita utilizando servidor *web* desenvolvido na linguagem de programação C, que pode ser visto em detalhes no ANEXO E. Este servidor é responsável pelas configurações de rede do kit de desenvolvimento, para que o *set-top-box* virtual possa se conectar a ele e enviar os comandos de acender ou apagar os

LEDs, e pelo tratamento dos dados que chegam a ele pela rede. Ele é responsável também pelo acionamento do LED e do *display* LCD correto à medida de que os comando chegam.

No kit de desenvolvimento utilizado neste projeto, foi embarcado um servidor *web* desenvolvido pela ACEPIC Tecnologia e Treinamento LTDA. Neste código, está incluído a pilha TCP/IP desenvolvida pela fabricante do controlador *ethernet* ENC28J60 e do PIC 18F4620 presentes no kit de desenvolvimento, a Microchip. Esta pilha implementa diversos protocolos no microcontrolador e foi desenvolvida levando em consideração todas as especificações contidas nas RFCs destes protocolos, pois "seu objetivo é permitir a conexão de microcontroladores à *Internet*, de forma transparente, como se fosse um "*host*" qualquer, ou seja, o fato de ser um dispositivo microcontrolado e não um computador, por exemplo, não deveria ser percebido"( SIQUEIRA, 2010).

Para que pudesse ser utilizado neste protótipo, foram necessárias alterações na configuração de rede do código do servidor *web*, como definição de endereço IP, *gateway* e submáscara de rede. Estas configurações foram definidas na função *IPAddrInit()* do arquivo *ccstcpip.h* do projeto do servidor web. A figura 4.13 mostra o trecho do código onde essas configurações foram definidas:

```
void MACAddrInit(void) {
   MY_MAC_BYTE1=0x00;
   MY_MAC_BYTE2=0x04;
   MY_MAC_BYTE3=0xa3;
   MY_MAC_BYTE4=0x00;
   MY_MAC_BYTE5=0x00;
   MY_MAC_BYTE6=0x00;
}

void IPAddrInit(void) {
   //IP address of this unit
   MY_IP_BYTE1=192;
   MY_IP_BYTE2=168;
   MY_IP_BYTE3=0;
   MY_IP_BYTE4=104;

   //network gateway
   MY_GATE_BYTE1=192;
   MY_GATE_BYTE2=168;
   MY_GATE_BYTE3=0;
   MY_GATE_BYTE4=1;//1;

   //subnet mask
   MY_MASK_BYTE1=255;
   MY_MASK_BYTE2=255;
   MY_MASK_BYTE3=255;
   MY_MASK_BYTE4=0;
}
```

Figura 4.13: Configurações de rede do kit de desenvolvimento
Fonte: (O autor)

Para acionar os LED do kit de desenvolvimento, o servidor *web* recebe uma solicitação via método *GET* do protocolo *HTTP* com os parâmetros *"lcd=a"*, onde *a* indica a mensagem a ser mostrada no *display* LCD do kit de desenvolvimento, e o parâmetro *"ledx=y"* onde *x* é o LED a ser comandado e *y* é 0 ou 1, sendo 0 o comando para apagar o LED, e 1 o comando para acendê-lo. O método *GET* é enviado, por meio do protocolo *HTTP*, pela TV Digital.

### 4.4.3 – *HomePlugs* TP Link TL-PA201

Para a comunicação entre o kit de desenvolvimento e a máquina virtual Ginga-NCL Set-top-box utilizando a rede elétrica como meio de transmissão de dados, foram utilizados dois dispositivos *HomePlug*, padrão AV, modelo TP-Link TL-PA201 AV 200Mbps. Um ligado ao *notebook* e outro ao kit de desenvolvimento. A figura 4.14 ilustra do dispositivo HomePlug:


Figura 4.14: HomePlug TP-Link modelo TL-PA201
Fonte: (Site da TP-Link)

Este dispositivo utiliza a técnica OFDM de modulação, e adota um sistema de segurança baseado na criptografia 128-bit AES (*Advanced Encryption Standard*). Sua taxa de transmissão pode atingir 200mbps.

# CAPÍTULO 5 – TESTES E VALIDAÇÃO DOS RESULTADOS

Este capítulo evidencia os testes realizados com o protótipo proposto e documenta os resultados obtidos.

Com o intuito de simular a infraestrutura de uma residência, foi montado o ambiente utilizando um *notebook*, simulando um aparelho de TV e um *set-top-box*, com a visualização e acesso à aplicação interativa e um kit de desenvolvimento com um servidor *web* embarcado realizando o controle dos dispositivos da residência. Para a comunicação foram utilizados dois dispositivos *HomePlugs* TP-Link TL-PA201 AV 200Mbps. A figura 5.1 mostra o protótipo proposto:


Figura 5.1: Protótipo proposto

Fonte: (O autor)

## 5.1 – Inicialização do Ginga-NCL *Set-top-box*

O primeiro passo a se tomar no teste do protótipo, é a inicialização do software *VMWare Fusion* com a máquina virtual Ginga-NCL *Virtual Set-top-box*. É iniciado o processo de *boot* e a tela do *boot loader* GRUB aparece com o Ginga-NCL *Virtual Set-top-box* selecionado por *default*, conforme a figura 5.2:

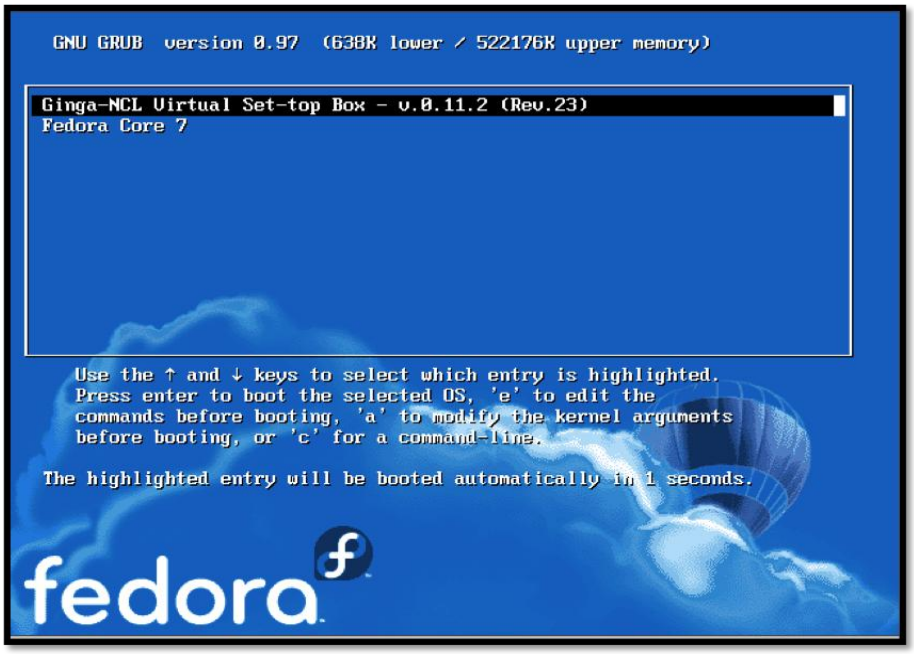


Figura 5.2: Boot loader GRUB

Fonte: (O autor)

Após o processo de inicialização da máquina virtual, surge a tela inicial do Ginga-NCL *Set-top-box* contendo o mapeamento das teclas do controle remoto sobre o teclado do PC:

- [F1] - Botão Vermelho;
- [F2] - Botão Verde;
- [F3] - Botão Amarelo;
- [F4] - Botão Azul;
- [F5] - Botão Menu;
- [F6] - Botão Info;
- [Enter] - Botão OK;
- [Setas] - Botões de setas direcionais;
- [Números] - Botões numéricos.

A figura 5.3 mostra a tela inicial da máquina virtual com o mapeamento das teclas:


Figura 5.3: Tela inicial do Ginga-NCL Set-top-box

Fonte: (O autor)

## 5.2 – Implementação da aplicação interativa no Ginga-NCL *Set-top-box*

Uma vez desenvolvida a aplicação interativa no Eclipse SDK, é necessário migrá-la para a máquina virtual Ginga-NCL *Set-top-box*, ambiente onde ela é executada. Para isto, foi utilizado o *software Fugu* na versão 1.2.0 para transferência de arquivos entre o computador hospedeiro e a máquina virtual. A figura 5.4 mostra a transferência do diretório *ProjetoFinal* realizada com sucesso:

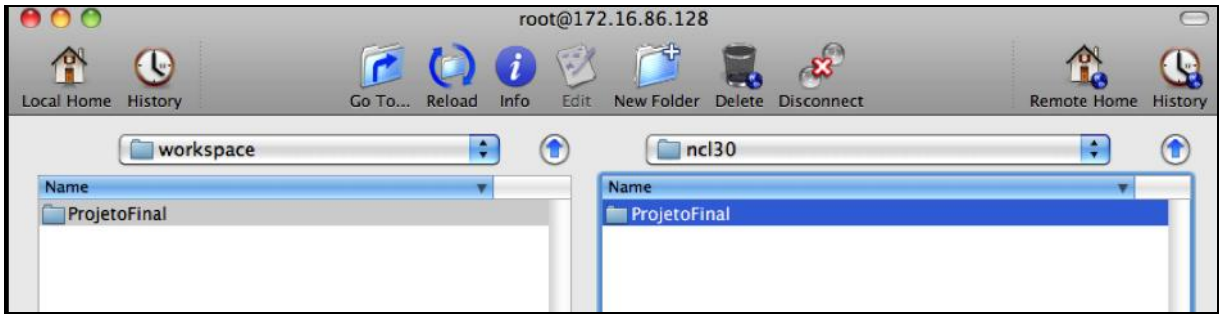


Figura 5.4: *Software Fugu* utilizado para transferência de arquivos

Fonte: (O autor)

A aplicação interativa precisa ser iniciada pelo comando mostrado na tela inicial da máquina virtual Ginga-NCL *Set-top-box*, figura 5.3. Para isso é necessário que se realize uma conexão SSH com a máquina virtual. Conforme é mostrado na figura 5.3, para a conexão SSH, deve ser utilizado o usuário *root*, e a senha *telemidia*. A figura 5.5 mostra essa conexão:

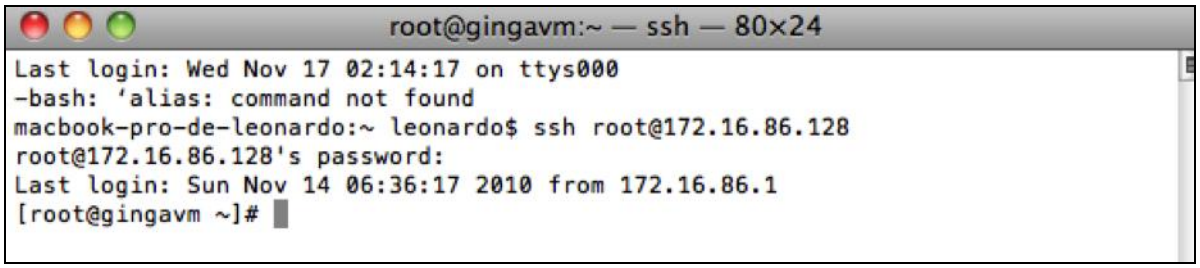


Figura 5.5: Conexão SSH com a máquina virtual

Fonte: (O autor)

Agora que a conexão com a máquina virtual foi realizada, o próximo passo é a inicialização da aplicação interativa. A aplicação foi armazenada no diretório *ProjetoFinal*, dentro do diretório */misc/ncl30* da máquina virtual. A aplicação é iniciada com o seguinte comando: *misc/launcher.sh /misc/ncl30/ProjetoFinal/projFinal_nclMain.ncl*, pois, conforme dito no capítulo 4 deste trabalho, o documento *projFinal_nclMain.ncl* é o principal documento NCL da aplicação interativa. A figura 5.6 mostra o comando a ser digitado:

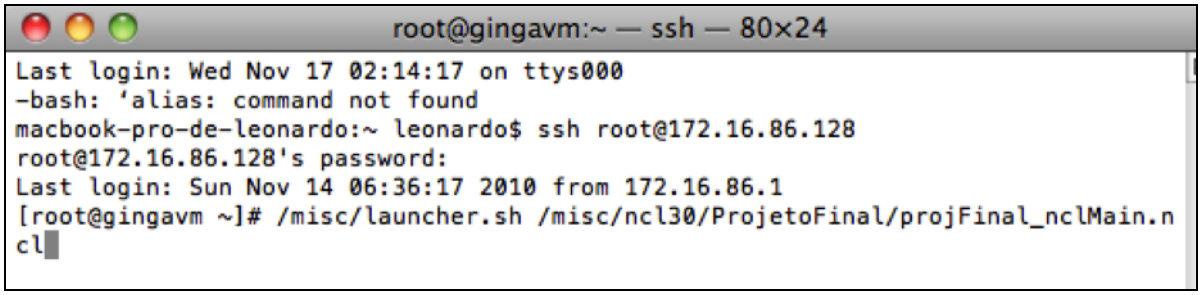


Figura 5.6: Comando para inicialização da aplicação interativa

Fonte: (O autor)

## 5.3 – A aplicação interativa

Ao iniciar a aplicação interativa, os vídeos promocionais do UniCEUB são inicializados, simulando a programação da emissora de TV. A figura 5.7 mostra o início do vídeo e o ícone interativo (i), no canto inferior direito da tela, em vermelho indicando haver uma interatividade disponível para o telespectador naquele momento:


Figura 5.7: Tela da máquina virtual simulando a tela da TV

Fonte: (O autor)

Como foi mostrado na figura 5.3, o botão vermelho do controle remoto é simulado, no teclado do notebook, pela tecla [F1]. Portanto, ao selecionar a tecla [F1], é iniciada a aplicação e o botão “Automação Residencial” é mostrado, conforme mostrado na figura 5.8:


Figura 5.8: Tela da máquina virtual simulando a tela da TV

Fonte: (O autor)

Para sair da aplicação, o telespectador pode novamente clicar no botão vermelho do controle remoto (tecla [F1] do notebook), conforme pode ser visto pelo botão Sair em vermelho na figura 5.8. Esta ação fará com que o ícone interativo (i) volte a ser exibido.

No início da aplicação, o botão "Automação residencial" pode ser visto selecionado pela sua borda cinza. O botão de seleção é feita com o clique no botão *OK* do controle remoto ([Enter] do teclado do notebook), conforme visto na figura 5.3. Quando o telespectador seleciona o botão "Automação Residencial", são apresentados para ele os botões referentes aos LEDs do kit de desenvolvimento, conforme mostra a figura 5.9:


Figura 5.9: Tela da máquina virtual simulando a tela da TV

Fonte: (O autor)

Nestes botões, a navegação é feita pelas teclas direcionais "para cima" e "para baixo" do controle remoto ([Setas] direcionais "para cima" e "para baixo" do teclado do notebook). Quando selecionado o botão referente a um dos LEDs, são mostrados os botões "Acender" e "Apagar". Nestes botões, a navegação também é feita pelas teclas direcionais "para cima" e "para baixo" do controle remoto ([Setas] direcionais "para cima" e "para baixo" do teclado do notebook).

Supondo que o telespectador queira acender o LED 3 do kit de desenvolvimento, ao clicar no botão OK do controle remoto (tecla [Enter] do notebook), serão mostrados os botões conforme a figura 5.10:


Figura 5.10: Tela da máquina virtual simulando a tela da TV

Fonte: (O autor)

Quando selecionado o botão “Acender” ou “Apagar”, o comando é enviado ao kit de desenvolvimento e o LED é aceso ou apagado. As figuras 5.11 e 5.12 mostram o kit de desenvolvimento ao receber e processar o comando de acionamento do LED 3. A mensagem *LED 3 ON* e *LED 3 OFF* no *display* LCD, indicam o comando recebido em cada uma das situações:

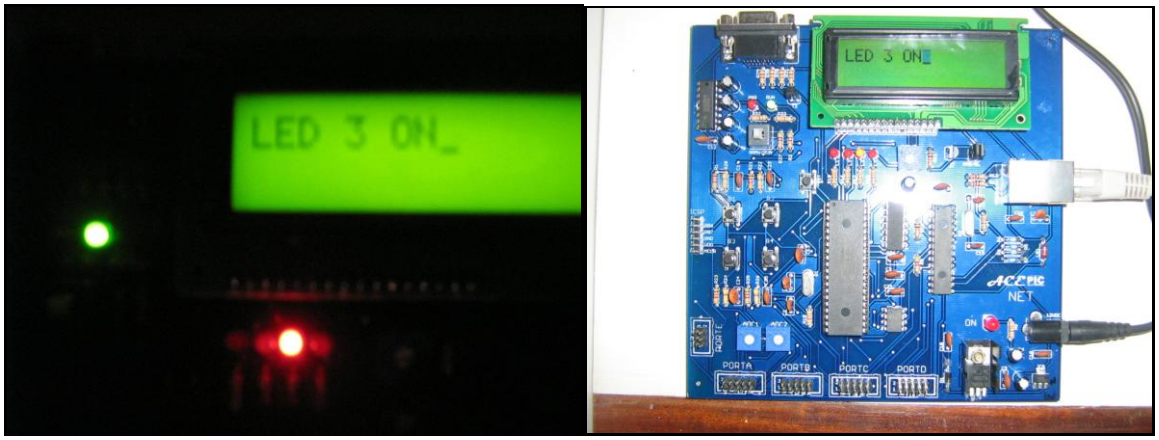


Figura 5.11: Comando LED 3 ON recebido

Fonte: (O autor)

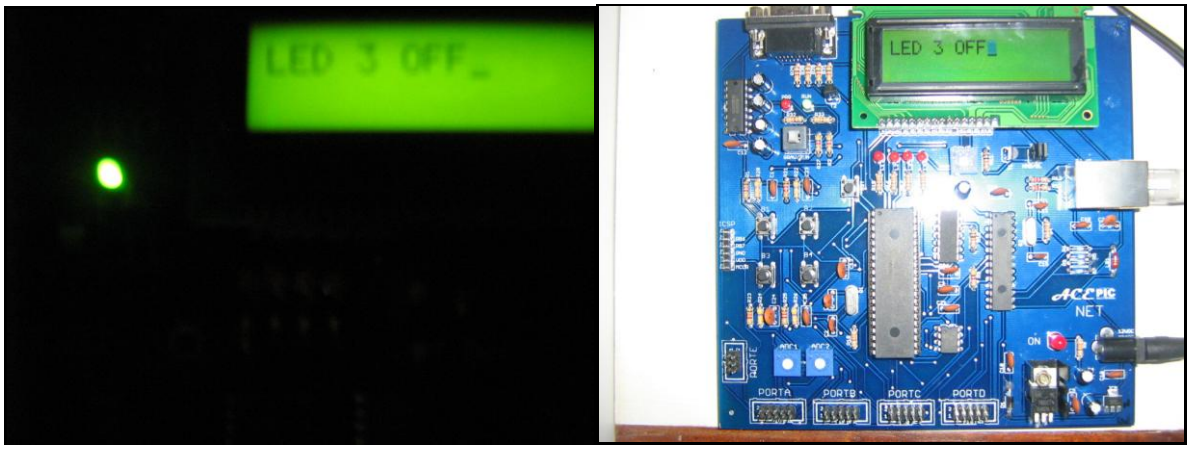


Figura 5.12: Comando LED 3 OFF recebido

Fonte: (O autor)

Conforme pode ser visto, os testes ocorreram de forma satisfatória e mostraram que a implementação do protótipo é viável. Para uma situação real, seriam necessárias algumas adaptações e serem abordados aspectos de interface, níveis de voltagem, uso de adaptadores e reguladores de tensão para os diversos dispositivos domésticos a serem ligados ao PIC, como lâmpadas de maior potência, motores e/ou eletrodomésticos, mantendo a lógica da implementação da aplicação interativa para TV Digital.

# CAPÍTULO 6 – CONCLUSÃO

Neste trabalho foi proposta a implementação de um protótipo de automação residencial utilizando a TV Digital como centro de controle (*set-top-box*), a tecnologia PLC para a transmissão de dados e um circuito eletrônico contendo um PIC para o acendimento de quatro LEDs simulando quatro dispositivos domésticos em uma residência. A comunicação entre *o set-top-box* e o circuito eletrônico foi realizado por meio da tecnologia PLC (*PowerLine Communication*).

A possibilidade de aplicação real de um projeto de automação residencial utilizando os recursos da TV Digital, se mostrou viável, já que as tecnologias utilizadas são eficientes. No entanto, por tais tecnologias ainda serem recentes, o projeto, aqui descrito, trata de um protótipo.

Desta forma, é possível concluir através dos estudos realizados para a concepção deste projeto, que a chegada da TV Digital, além das mudanças já proporcionadas como a interatividade, a significativa melhora da imagem e som ainda causará mudanças consideráveis na forma como os telespectadores assistem TV. Não bastasse isso, foi possível concluir também, que a TV Digital, ainda apresenta inúmeras possibilidades a serem exploradas e por isso ainda causará muitas mudanças nas telecomunicações. Atualmente já existem algumas aplicações de interatividade para diversos fins, mas ainda há muito o que se desenvolver.

## 6.1 – Sugestões para Trabalhos Futuros

Durante a realização do trabalho, surgiram diversas possibilidades de aprimoramento que são sugeridas como propostas para trabalhos futuros:

- Desenvolvimento e aprimoramento do servidor web desenvolvido pela ACEPIC para envio de retorno ao *set-top-box* indicando *status* dos LED;
- Utilização de outros dispositivos domésticos como um ventilador, que utilizaria diferentes componentes eletrônicos;
- Aprimoramento da interface gráfica da TV Digital em NCL;
- Customização deste projeto, para que possa ser utilizado a partir da TV Digital para dispositivos móveis.

# REFERÊNCIAS

ABNT. ABNT NBR 15601-1, Televisão digital terrestre — Sistema de Transmissão. ABNT, primeira edição, 2007.

ABNT NBR 15606-2, Codificação de dados e especificações de transmissão para radiodifusão digital, Parte 2: Ginga-NCL para receptores fixos e móveis – Linguagem de aplicação XML para codificação de aplicações. ABNT, primeira edição, 2007.

ABNT NBR 15606-4, Codificação de dados e especificações de transmissão para radiodifusão digital, Parte 4: Ginga-J – Ambiente para execução de aplicações procedurais. ABNT, primeira edição, 2010.

ABNT NBR 15606-5, Codificação de dados e especificações de transmissão para radiodifusão digital, Parte 5: Ginga-NCL para receptores portáteis – Linguagem de aplicação XML para codificação de aplicações. ABNT, primeira edição, 2008.

ABNT NBR 15607-1, Canal de Interatividade, Parte 1: Protocolos, interfaces físicas e interfaces de software. ABNT, primeira edição, 2008.

ABNT NBR 15608-1, Guia de Operação, Parte 1: Sistema de transmissão – Guia para implementação da ABNT NBR 15601:2007. ABNT, primeira edição, 2008.

ABNT NBR 15608-2, Guia de Operação, Parte 2: Codificação de vídeo, áudio e multiplexação – Guia para implementação da ABNT NBR 15602:2007. ABNT, primeira edição, 2008.

ABRANTES, Talita. A Web em 110 Volts. Revista INFO EXAME n.284, São Paulo, Outubro, 2009

ARIB. ARIB Standard. STD-B31. Transmission System for Digital Terrestrial Television Broadcasting. Ver. 1.5. Jul. 2003.

ATSC. Advanced Television Systems Committee Inc, 2006. Especificação e documentação do padrão ATSC. Disponível em: <http://www.atsc.org/standards.html>. Acessado em Março, 2010.

AVILA, Flavio Rocha de; Pereira, Carlos Eduardo. TECNOLOGIA PLC - A NOVA ERA DA COMUNICAÇÃO DE DADOS EM BANDA LARGA. CETA - Centro de Excelência em Tecnologias Avançadas SENAI-RS; Programa de Pós-Graduação em Engenharia Elétrica - UFRGS. 2007.

BARBOSA, Simone Diniz Junqueira; SOARES, Luiz Fernando Gomes. TV Digital no Brasil se faz com Ginga.

BASTOS, Arilson e FERNANDES, Sérgio. Televisão Digital. 1. ed. Rio de Janeiro. Antenna Edições Técnicas Ltda, 2004.

BASTOS, Wanessa de Sousa. Solução de interatividade para TV Digital em padrão ISDB-TB utilizando canal de retorno via internet. Trabalho apresentado ao Centro Universitário de Brasília (UniCEUB) para a obtenção de Certificado de Conclusão de Curso de Engenharia de Computação. Brasília, 2010.

BBC. Digital TV. Disponível em <http://www.bbc.co.uk/digital/tv/>. Acessado em Agosto, 2010.

Decreto n.º 5.820, de 29 de Junho de 2006. Diário Oficial da União, República Federativa do Brasil, Poder Executivo, Brasília, DF, 30 jun. 2006.

CAMARGO, Camila Porto de. História da Televisão. Disponível em <http://www.baixaki.com.br/tecnologia/2397-historia-da-televisao.htm>. Acessado em Outubro 2010.

CARVALHO, Rafael; SANTOS, Joel André Ferreira dos; DAMASCENO, Jean Ribeiro; SILVA, Julia Varanda da; SAADE, Débora Christina Muchaluat. Introdução às Linguagens NCL e Lua: Desenvolvendo Aplicações Interativas para TV Digital. Laboratório MídiaCom. Universidade Federal Fluminense. Outubro, 2009.

CHEROBINO,Vinicius. Barreirinhas usará rede elétrica como base para TV Digital interativa. Disponível em <http://idgnow.uol.com.br/telecom/2008/08/29/barreirinhas-usara-rede-eletrica-como-base-para-tv-digital-interativa/>. Acessado em Novembro, 2010.

COMER, Douglas E. Redes de Computadores e Internet; trad. Marinho Barcelos – 2.ed – Porto Alegre; Bookman, 2001

CPqD. Arquitetura de Referência. OS 40.541. Campinas-SP, 2005.

DTV, Site Oficial da TV Digital Brasileira. Disponível em <http://www.dtv.org.br>. Acessado em Agosto, 2010.

DVB. Digital Video Broadcasting Project. 2003. Site oficial do projeto DVB. Disponível em: <http://www.dvb.org>. Acessado em Agosto, 2010.

ELLWANGER, Fábio, BALBINOT, Gustavo. A Linguagem de Programação Lua. Disponível em http://www.inf.unisinos.br/~barbosa/paradigmas/consipa3/53/artigos/a3.pdf. Acessado em Novembro, 2010.

FERNANDES, J.; LEMOS, G.; SILVEIRA, G. Introdução à Televisão Digital Interativa: Arquitetura, Protocolos, Padrões e Práticas. In: JAI-SBC. Salvador, 2004.

FERREIRA, Marcus Vinicius de Almeida. PLC - Power Line Communication. Departamento de Telecomunicações – Laboratório MIDIACOM Universidade Federal Fluminense (UFF).

FÓRUM SBTVD. Fórum do Sistema Brasileiro de TV Digital. Disponível em: <http://www.forumsbtvd.org.br>. Acessado em Setembro, 2010.

GINGA. Sobre o Ginga. Ginga, 2006. Disponível em: <http://www.ginga.org.br/sobre.html>. Acessado em Setembro, 2010.

HALID Hrasnica,Abdelfatteh Haidine,Ralf Lehnert. Broadband powerline communications networks: network design. 2004.

HAYKIN, S. Sistemas de Comunicação: Analógicos e Digitais. 4a. Edição. Editora Bookman, 2004.

HOMEPLUG – PowerLine Alliance Disponível em: <http://www.homeplug.org/>. Acessado em Novembro, 2010.

JARDIM, Felipe Barbosa Maia; Gomes, Antônio Ricardo Leocádio. PLC – POWER LINE COMMUNICATIONS: DEFINIÇÕES, VANTAGENS, DESVANTAGENS E REQUISITOS. Disponível em: <http://www.webartigos.com/articles/41249/1/PLC---Power-Line-Communication/>. Acessado em Novembro, 2010.

KRONEMBERGER, Ingo Henrique Mamede. Implementação de um sistema indoor de comunicação de dados, pela rede eletria, em um circuito isolado. Monografia apresentada ao Curso de Engenharia da Computação, como requisito parcial para obtenção do grau de Engenheiro de Computação. Brasília, 2009.

Lua – The Programming Language. Disponível em: <http://www.lua.org/>. acessado em Novembro, 2010.

MAIA, Orlewilson Bentes. Uma infraestrutura de comunicação entre dispositivos domésticos e o modelo brasileiro de TV Digital. Dissertação apresentada ao Programa de Pós-Graduação em Engenharia Elétrica da UFA. Manaus, 2009.

MENDES, Luciano Leonel. SBTVD: Uma visão sobre a TV digital no Brasil. T&C Amazônica, ano V, número 12, out. 2007.

MONTEZ, Carlos; BECKER, Valdecir. TV Digital Interativa: Conceitos e Tecnologias. In: WebMidia e LA-Web 2004 – Joint Conference. Ribeirão Preto, SP, Outubro de 2004.

Nested Context Language. Disponível em <http://www.ncl.org.br/>. Acessado em Agosto, 2010.

NETO, C.S.S; Soares, L.F.G; Rodrigues, R.F; Barbosa, S.D.J: Tutorial Construindo Programas Audiovisuais Interativos Utilizando a NCL 3.0 e a Ferramenta Composer; PUC-RJ, Telemídia, SERG; 2ª Ed, 2007.

OLIVEIRA, Bruno Julian Dias de; Barbosa, Hildegard Paulino; Silva, Julio Cesar Ferreira da; Tavares, Tatiana Aires. Uma casa no controle da TV: Desenvolvimento de um Programa para TV Digital para Controle de Dispositivos Domésticos. Laboratório de Aplicações de Vídeo Digital UFPB, João Pessoa, 2008.

OLIVEIRA, Bruno Julian Dias de. Um estudo de caso entre Ginga-J e Ginga-NCL no âmbito de aplicações interativas residentes. Monografia apresentada junto ao curso de Ciência da Computação da Universidade Federal da Paraíba. João Pessoa, 2010.

COMPLEXX TECNOLOGIA. O que é Automação residencial. Diponível em: <http://www.complexx.com.br/engenharia/servicos/index.asp?cod=11>. Acessado em Novembro 2010.

PINHEIRO, José Mauricio Santos. Multiplexação Ortogonal por Divisão de Freqüência. Disponível em: < http://www.projetoderedes.com.br/artigos/artigo_multiplexacao_ortogonal_por_divisao_de_frequencia.php>. Acessado em Dezembro, 2010.

PORTAL DO SOFTWARE PÚBLICO BRASILEIRO. Disponível em: <http://www.softwarepublico.gov.br/dotlrn/clubs/ginga/gingancl/one-community?page_num=0>. Acessado em Agosto, 2010.

SOARES, Luiz Fernando Gomes Soares; Barbosa, Simone Diniz Junqueira Barbosa. TV Digital interativo no Brasil se faz com Ginga, Fundamentos, Padrões, Autoria Declarativa e Usabilidade. Departamento de Informática, PUC-Rio. Rio de Janeiro, 2008.

SOARES, L.F.S; Barbosa, S.D.J. Programando em NCL. 1 ed. Rio de Janeiro. Campus, 2009.

SOARES, L.F.S; Rodrigues, R.F. Nested Context Model 3.0 Part 1 – NCM Core, Monografias em Ciências da Computação de Informática, PUC-Rio, No. 18/05. Rio de Janeiro, mai. 2005.

TAVEIRA, Danilo Michalczuk. Redes PLC. Maio de 2004. Disponível em http://www.gta.ufrj.br/grad/04_1/redesplc/, acessado em Novembro, 2010.

TEIXEIRA, Érico Barbosa Teixeira; ALMEIDA, Filipe Sousa Bragança Ferreira de; JÚNIOR, Luiz Alberto da Silva. TV Digital e SBTVD-T. Artigo para a etapa de classificação do II Concurso Teleco de Trabalhos de Conclusão de Curso (TCC) 2006. Dísponível em http://www.teleco.com.br/tutoriais/tutorialsbtvd/pagina_1.asp, acessado em Agosto, 2010.

TELECO. Inteligência em Telecomunicações. Disponível em: <http://www.teleco.com.br/>. Acessado em Agosto, 2010.

TELEMIDIA. Ambiente para Desenvolvimento de Aplicações Declarativas para a TV Digital Brasileira. Laboratório Telemídia, Depto. Informática, PUC-Rio Rio de Janeiro, 2006.

VARGAS, Alessandra Antunes. Estudo sobre Comunicação de Dados via Rede Elétrica para Aplicações de Automação Residencial/Predial. Projeto de Diplomação apresentado como requisito parcial para obtenção do grau de Engenheiro de Computação da Universidade Federal do Rio Grande do Sul, Porto Alegre, 2004.

VIEIRA, D. A. et al. Ginga@Home – Ginga and Zigbee Integration for Controlling Devices in Smart Homes, 2008.

ZAHADIADIS, Theodore B., Home networking Technologies and Standards. 2003.

**APÊNDICE A – projFinal_nclMain.ncl**

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- Generated by NCL Eclipse -->
<ncl id="projFinal_nclMain" xmlns="http://www.ncl.org.br/NCL3.0/EDTVProfile">
	<head>

		<!--REGIÕES-->
		<regionBase>

			<!--região principal da tela-->
			<region id="rgMain" width="100%" height="100%">

				<!--região do vídeo-->
				<region id="rgVideo" width="100%" height="100%"
zIndex="10"/>

				<!--região do botão de interatividade-->
				<region id="rgBtnInterativo" right="7%" top="80%"
width="71" height="83" zIndex="100"/>

				<!--região do menu de interatividade-->
				<region id="rgMenuInterativo" left="10%" top="10%"
width="220" height="35" zIndex="100"/>

				<!--região do botão sair-->
				<region id="rgBtnSair" right="7%" top="80%" width="65"
height="40" zIndex="100"/>

				<!--região do botão lampada1-->
				<region id="rgBtnLampada1" left="7%" top="10%"
width="110" height="32" zIndex="100"/>

				<!--região do botão AcenderLampada1-->
				<region id="rgBtnAcenderLampada1" left="7%" top="7%"
width="107" height="30" zIndex="100"/>

				<!--região do botão ApagarLampada1-->
				<region id="rgBtnApagarLampada1" left="7%" top="14%"
width="107" height="30" zIndex="100"/>

				<!--região do botão lampada2-->
				<region id="rgBtnLampada2" left="7%" top="20%"
width="110" height="32" zIndex="100"/>

				<!--região do botão AcenderLampada2-->
				<region id="rgBtnAcenderLampada2" left="7%" top="7%"
width="107" height="30" zIndex="100"/>

				<!--região do botão ApagarLampada2-->
				<region id="rgBtnApagarLampada2" left="7%" top="14%"
```

```
width="107" height="30" zIndex="100"/>

                        <!--região do botão lampada3-->
                        <region id="rgBtnLampada3" left="7%" top="30%"
width="110" height="32" zIndex="100"/>

                        <!--região do botão AcenderLampada3-->
                        <region id="rgBtnAcenderLampada3" left="7%" top="7%"
width="107" height="30" zIndex="100"/>

                        <!--região do botão ApagarLampada3-->
                        <region id="rgBtnApagarLampada3" left="7%" top="14%"
width="107" height="30" zIndex="100"/>

                        <!--região do botão lampada4-->
                        <region id="rgBtnLampada4" left="7%" top="40%"
width="110" height="32" zIndex="100"/>

                        <!--região do botão AcenderLampada4-->
                        <region id="rgBtnAcenderLampada4" left="7%" top="7%"
width="107" height="30" zIndex="100"/>

                        <!--região do botão ApagarLampada4-->
                        <region id="rgBtnApagarLampada4" left="7%" top="14%"
width="107" height="30" zIndex="100"/>

                </region>
        </regionBase>

        <!--DESCRITORES-->
        <descriptorBase>
                <!--descritor do vídeo-->
                <descriptor id="dVideo" region="rgVideo"/>
                <!--descritor do botão de interatividade-->
                <descriptor id="dBtnInterativo" region="rgBtnInterativo"/>
                <!--descritor do menu de interatividade-->
                <descriptor id="dMenuInterativo" region="rgMenuInterativo"
focusIndex="1" focusBorderColor="gray" focusSelSrc="2"/>
                <!--descritor do botão para sair da interatividade-->
                <descriptor id="dBtnSairInteratividade" region="rgBtnSair"/>
                <!--descritor do botão para sair da lampada1-->
                <descriptor id="dBtnSairLampada1" region="rgBtnSair"/>
                <!--descritor do botão para sair da lampada2-->
                <descriptor id="dBtnSairLampada2" region="rgBtnSair"/>
                <!--descritor do botão para sair da lampada3-->
                <descriptor id="dBtnSairLampada3" region="rgBtnSair"/>
                <!--descritor do botão para sair da lampada4-->
                <descriptor id="dBtnSairLampada4" region="rgBtnSair"/>
                <!--descritor do botão lampada1-->
                <descriptor id="dBtnLampada1" region="rgBtnLampada1"
focusIndex="2" focusBorderColor="gray" moveDown="3" focusSelSrc="5"/>
```

```
                        <!--descritor do botão lampada2-->
                        <descriptor id="dBtnLampada2" region="rgBtnLampada2"
focusIndex="3" focusBorderColor="gray" moveDown="4" moveUp="2"/>
                        <!--descritor do botão lampada3-->
                        <descriptor id="dBtnLampada3" region="rgBtnLampada3"
focusIndex="4" focusBorderColor="gray" moveDown="5" moveUp="3"/>
                        <!--descritor do botão lampada4-->
                        <descriptor id="dBtnLampada4" region="rgBtnLampada4"
focusIndex="5" focusBorderColor="gray" moveUp="4"/>
                        <!--descritor do botão AcenderLampada1-->
                        <descriptor id="dBtnAcenderLampada1"
region="rgBtnAcenderLampada1" focusIndex="6" focusBorderColor="gray"
moveDown="7"/>
                        <!--descritor do botão ApagarLampada1-->
                        <descriptor id="dBtnApagarLampada1"
region="rgBtnApagarLampada1" focusIndex="7" focusBorderColor="gray" moveUp="6"/>
                        <!--descritor do botão AcenderLampada2-->
                        <descriptor id="dBtnAcenderLampada2"
region="rgBtnAcenderLampada2" focusIndex="8" focusBorderColor="gray"
moveDown="9"/>
                        <!--descritor do botão ApagarLampada2-->
                        <descriptor id="dBtnApagarLampada2"
region="rgBtnApagarLampada2" focusIndex="9" focusBorderColor="gray" moveUp="8"/>
                        <!--descritor do botão AcenderLampada3-->
                        <descriptor id="dBtnAcenderLampada3"
region="rgBtnAcenderLampada3" focusIndex="10" focusBorderColor="gray"
moveDown="11"/>
                        <!--descritor do botão ApagarLampada3-->
                        <descriptor id="dBtnApagarLampada3"
region="rgBtnApagarLampada3" focusIndex="11" focusBorderColor="gray"
moveUp="10"/>
                        <!--descritor do botão AcenderLampada4-->
                        <descriptor id="dBtnAcenderLampada4"
region="rgBtnAcenderLampada4" focusIndex="12" focusBorderColor="gray"
moveDown="13"/>
                        <!--descritor do botão ApagarLampada4-->
                        <descriptor id="dBtnApagarLampada4"
region="rgBtnApagarLampada4" focusIndex="13" focusBorderColor="gray"
moveUp="12"/>
                </descriptorBase>

                <!--IMPORT DA BASE DE CONECTORES-->
                <connectorBase>
                        <!--conectores que definem como os elos são ativados e o que eles
disparam-->
                        <importBase alias="connBase"
documentURI="masterConnectorBase.ncl"/>
                </connectorBase>
        </head>

        <body>
```

```
                <!--nós de mídia de Vídeo-->
                <media id="video1" type="video/mpeg" src="media/video1.mpg"
descriptor="dVideo"/>
                <media id="video2" type="video/mpeg" src="media/video2.mpg"
descriptor="dVideo"/>

                <!--media scripts lua-->
                <media id="acendeLampada1" src="lua/acende1.lua"/>
                <media id="apagaLampada1" src="lua/apaga1.lua"/>
                <media id="acendeLampada2" src="lua/acende2.lua"/>
                <media id="apagaLampada2" src="lua/apaga2.lua"/>
                <media id="acendeLampada3" src="lua/acende3.lua"/>
                <media id="apagaLampada3" src="lua/apaga3.lua"/>
                <media id="acendeLampada4" src="lua/acende4.lua"/>
                <media id="apagaLampada4" src="lua/apaga4.lua"/>

                <!--imagem botão interativo-->
                <media id="btnInterativo" type="image/jpeg" src="media/botaoInterativo.jpg"
descriptor="dBtnInterativo"/>

                <!--imagem menu automação residencial-->
                <media id="menuAutoResi" type="image/jpeg"
src="media/menuInterativo.jpg" descriptor="dMenuInterativo"/>

                <!--imagem botão sair da interatividade-->
                <media id="btnSairInteratividade" type="image/jpeg" src="media/sair.jpg"
descriptor="dBtnSairInteratividade"/>

                <!--imagem botão sair da lampada1-->
                <media id="btnSairLampada1" type="image/jpeg" src="media/sair.jpg"
descriptor="dBtnSairLampada1"/>
                <!--imagem botão sair da lampada2-->
                <media id="btnSairLampada2" type="image/jpeg" src="media/sair.jpg"
descriptor="dBtnSairLampada2"/>
                <!--imagem botão sair da lampada3-->
                <media id="btnSairLampada3" type="image/jpeg" src="media/sair.jpg"
descriptor="dBtnSairLampada3"/>
                <!--imagem botão sair da lampada3-->
                <media id="btnSairLampada4" type="image/jpeg" src="media/sair.jpg"
descriptor="dBtnSairLampada4"/>

                <!--imagem botão lampada1-->
                <media id="btnLampada1" type="image/jpeg" src="media/lampada1.jpg"
descriptor="dBtnLampada1"/>

                <!--imagem botão lampada2-->
                <media id="btnLampada2" type="image/jpeg" src="media/lampada2.jpg"
descriptor="dBtnLampada2"/>

                <!--imagem botão lampada3-->
```

```
            <media id="btnLampada3" type="image/jpeg" src="media/lampada3.jpg"
descriptor="dBtnLampada3"/>

            <!--imagem botão lampada4-->
            <media id="btnLampada4" type="image/jpeg" src="media/lampada4.jpg"
descriptor="dBtnLampada4"/>

            <!--imagem botão acender lampada1-->
            <media id="btnAcenderLampada1" type="image/jpeg"
src="media/acender.jpg" descriptor="dBtnAcenderLampada1"/>

            <!--imagem botão pagar lampada1-->
            <media id="btnApagarLampada1" type="image/jpeg" src="media/apagar.jpg"
descriptor="dBtnApagarLampada1"/>

            <!--imagem botão acender lampada2-->
            <media id="btnAcenderLampada2" type="image/jpeg"
src="media/acender.jpg" descriptor="dBtnAcenderLampada2"/>

            <!--imagem botão pagar lampada2-->
            <media id="btnApagarLampada2" type="image/jpeg" src="media/apagar.jpg"
descriptor="dBtnApagarLampada2"/>

            <!--imagem botão acender lampada3-->
            <media id="btnAcenderLampada3" type="image/jpeg"
src="media/acender.jpg" descriptor="dBtnAcenderLampada3"/>

            <!--imagem botão pagar lampada3-->
            <media id="btnApagarLampada3" type="image/jpeg" src="media/apagar.jpg"
descriptor="dBtnApagarLampada3"/>

            <!--imagem botão acender lampada4-->
            <media id="btnAcenderLampada4" type="image/jpeg"
src="media/acender.jpg" descriptor="dBtnAcenderLampada4"/>

            <!--imagem botão pagar lampada3-->
            <media id="btnApagarLampada4" type="image/jpeg" src="media/apagar.jpg"
descriptor="dBtnApagarLampada4"/>

            <!--PORTA DE INÍCIO-->
            <port id="pInicio" component="video1"/>

            <!--Quando começa o vídeo1, inicía o botão interativo-->
            <link xconnector="connBase#onBeginStartDelay"
id="lBeginVideo1StartBtnInteratividade">
                  <bind role="onBegin" component="video1"/>
                  <bind role="start" component="btnInterativo"/>
            </link>

            <!--Quando termina video1, inicia vídeo2 (LOOP)-->
            <link xconnector="connBase#onEndStart" id="lEndVideo1StartVideo2Loop">
```

```
                <bind role="onEnd" component="video1"/>
                <bind role="start" component="video2"/>
            </link>

            <!--Quando termina video2, inicia vídeo1 (LOOP)-->
            <link xconnector="connBase#onEndStart" id="lEndVideo2StartVideo1Loop">
                <bind role="onEnd" component="video2"/>
                <bind role="start" component="video1"/>
            </link>

            <!--Inicia Menu Interativo-->
            <link xconnector="connBase#onKeySelectionStartNStopN"
id="lIniciaInteratividade">
                <bind role="onSelection" component="btnInterativo">
                    <bindParam name="keyCode" value="RED"/>
                </bind>
                <bind role="start" component="menuAutoResi"/>
                <bind role="start" component="btnSairInteratividade"/>
                <bind role="stop" component="btnInterativo"/>
            </link>
            <!--Finaliza Menu Interativo-->
            <link xconnector="connBase#onKeySelectionStartNStopN"
id="lFinalizaInteratividade">
                <bind role="onSelection" component="btnSairInteratividade">
                    <bindParam name="keyCode" value="RED"/>
                </bind>
                <bind role="start" component="btnInterativo"/>
                <bind role="stop" component="menuAutoResi"/>
                <bind role="stop" component="btnSairInteratividade"/>
                <bind role="stop" component="btnLampada1"/>
                <bind role="stop" component="btnLampada2"/>
                <bind role="stop" component="btnLampada3"/>
                <bind role="stop" component="btnLampada4"/>
                <bind role="stop" component="btnAcenderLampada1"/>
                <bind role="stop" component="btnApagarLampada1"/>
                <bind role="stop" component="btnAcenderLampada2"/>
                <bind role="stop" component="btnApagarLampada2"/>
                <bind role="stop" component="btnAcenderLampada3"/>
                <bind role="stop" component="btnApagarLampada3"/>
                <bind role="stop" component="btnAcenderLampada4"/>
                <bind role="stop" component="btnApagarLampada4"/>
            </link>
            <!--Inicia BtnsLampadas-->
            <link xconnector="connBase#onKeySelectionStartNStopN"
id="selecionaMenuAutoResi">
                <bind role="onSelection" component="menuAutoResi"/>
                <bind role="start" component="btnLampada1"/>
                <bind role="start" component="btnLampada2"/>
                <bind role="start" component="btnLampada3"/>
                <bind role="start" component="btnLampada4"/>
                <bind role="stop" component="menuAutoResi"/>
```

```
		</link>

		<!--Inicia Btns para acender ou apagar lampada1-->
		<link xconnector="connBase#onKeySelectionStartNStopN"
id="selecionaLampada1">
			<bind role="onSelection" component="btnLampada1"/>
			<bind role="start" component="btnAcenderLampada1"/>
			<bind role="start" component="btnApagarLampada1"/>
			<bind role="start" component="btnSairLampada1"/>
			<bind role="stop" component="btnSairInteratividade"/>
			<bind role="stop" component="btnLampada1"/>
			<bind role="stop" component="btnLampada2"/>
			<bind role="stop" component="btnLampada3"/>
			<bind role="stop" component="btnLampada4"/>
		</link>
		<!--Acende Lampada1-->
		<link xconnector="connBase#onKeySelectionStart" id="scriptLuaAcende1">
			<bind role="onSelection" component="btnAcenderLampada1"/>
			<bind role="start" component="acendeLampada1"/>
		</link>
		<!--Apaga Lampada1-->
		<link xconnector="connBase#onKeySelectionStart" id="scriptLuaApaga1">
			<bind role="onSelection" component="btnApagarLampada1"/>
			<bind role="start" component="apagaLampada1"/>
		</link>
		<!--selecionar botão para sair da lampada1-->
		<link xconnector="connBase#onKeySelectionStartNStopN"
id="sairLampada1">
			<bind role="onSelection" component="btnSairLampada1">
				<bindParam name="keyCode" value="RED"/>
			</bind>
			<bind role="start" component="btnSairInteratividade"/>
			<bind role="start" component="btnLampada1"/>
			<bind role="start" component="btnLampada2"/>
			<bind role="start" component="btnLampada3"/>
			<bind role="start" component="btnLampada4"/>
			<bind role="stop" component="btnAcenderLampada1"/>
			<bind role="stop" component="btnApagarLampada1"/>
			<bind role="stop" component="btnSairLampada1"/>
		</link>

		<!--Inicia Btns para acender ou apagar lampada2-->
		<link xconnector="connBase#onKeySelectionStartNStopN"
id="selecionaLampada2">
			<bind role="onSelection" component="btnLampada2"/>
			<bind role="start" component="btnAcenderLampada2"/>
			<bind role="start" component="btnApagarLampada2"/>
			<bind role="start" component="btnSairLampada2"/>
			<bind role="stop" component="btnSairInteratividade"/>
			<bind role="stop" component="btnLampada1"/>
			<bind role="stop" component="btnLampada2"/>
```

```
                <bind role="stop" component="btnLampada3"/>
                <bind role="stop" component="btnLampada4"/>
            </link>
            <!--Acende Lampada2-->
            <link xconnector="connBase#onKeySelectionStart" id="scriptLuaAcende2">
                <bind role="onSelection" component="btnAcenderLampada2"/>
                <bind role="start" component="acendeLampada2"/>
            </link>
            <!--Apaga Lampada2-->
            <link xconnector="connBase#onKeySelectionStart" id="scriptLuaApaga2">
                <bind role="onSelection" component="btnApagarLampada2"/>
                <bind role="start" component="apagaLampada2"/>
            </link>
            <!--selecionar botão para sair da lampada2-->
            <link xconnector="connBase#onKeySelectionStartNStopN"
id="sairLampada2">
                <bind role="onSelection" component="btnSairLampada2">
                    <bindParam name="keyCode" value="RED"/>
                </bind>
                <bind role="start" component="btnSairInteratividade"/>
                <bind role="start" component="btnLampada1"/>
                <bind role="start" component="btnLampada2"/>
                <bind role="start" component="btnLampada3"/>
                <bind role="start" component="btnLampada4"/>
                <bind role="stop" component="btnAcenderLampada2"/>
                <bind role="stop" component="btnApagarLampada2"/>
                <bind role="stop" component="btnSairLampada2"/>
            </link>

            <!--Inicia Btns para acender ou apagar lampada3-->
            <link xconnector="connBase#onKeySelectionStartNStopN"
id="selecionaLampada3">
                <bind role="onSelection" component="btnLampada3"/>
                <bind role="start" component="btnAcenderLampada3"/>
                <bind role="start" component="btnApagarLampada3"/>
                <bind role="start" component="btnSairLampada3"/>
                <bind role="stop" component="btnSairInteratividade"/>
                <bind role="stop" component="btnLampada1"/>
                <bind role="stop" component="btnLampada2"/>
                <bind role="stop" component="btnLampada3"/>
                <bind role="stop" component="btnLampada4"/>
            </link>
            <!--Acende Lampada3-->
            <link xconnector="connBase#onKeySelectionStart" id="scriptLuaAcende3">
                <bind role="onSelection" component="btnAcenderLampada3"/>
                <bind role="start" component="acendeLampada3"/>
            </link>
            <!--Apaga Lampada3-->
            <link xconnector="connBase#onKeySelectionStart" id="scriptLuaApaga3">
                <bind role="onSelection" component="btnApagarLampada3"/>
                <bind role="start" component="apagaLampada3"/>
```

```
                </link>
                <!--selecionar botão para sair da lampada3-->
                <link xconnector="connBase#onKeySelectionStartNStopN"
id="sairLampada3">
                        <bind role="onSelection" component="btnSairLampada3">
                                <bindParam name="keyCode" value="RED"/>
                        </bind>
                        <bind role="start" component="btnSairInteratividade"/>
                        <bind role="start" component="btnLampada1"/>
                        <bind role="start" component="btnLampada2"/>
                        <bind role="start" component="btnLampada3"/>
                        <bind role="start" component="btnLampada4"/>
                        <bind role="stop" component="btnAcenderLampada3"/>
                        <bind role="stop" component="btnApagarLampada3"/>
                        <bind role="stop" component="btnSairLampada3"/>
                </link>

                <!--Inicia Btns para acender ou apagar lampada4-->
                <link xconnector="connBase#onKeySelectionStartNStopN"
id="selecionaLampada4">
                        <bind role="onSelection" component="btnLampada4"/>
                        <bind role="start" component="btnAcenderLampada4"/>
                        <bind role="start" component="btnApagarLampada4"/>
                        <bind role="start" component="btnSairLampada4"/>
                        <bind role="stop" component="btnSairInteratividade"/>
                        <bind role="stop" component="btnLampada1"/>
                        <bind role="stop" component="btnLampada2"/>
                        <bind role="stop" component="btnLampada3"/>
                        <bind role="stop" component="btnLampada4"/>
                </link>
                <!--Acende Lampada4-->
                <link xconnector="connBase#onKeySelectionStart" id="scriptLuaAcende4">
                        <bind role="onSelection" component="btnAcenderLampada4"/>
                        <bind role="start" component="acendeLampada4"/>
                </link>
                <!--Apaga Lampada4-->
                <link xconnector="connBase#onKeySelectionStart" id="scriptLuaApaga4">
                        <bind role="onSelection" component="btnApagarLampada4"/>
                        <bind role="start" component="apagaLampada4"/>
                </link>
                <!--selecionar botão para sair da lampada4-->
                <link xconnector="connBase#onKeySelectionStartNStopN"
id="sairLampada4">
                        <bind role="onSelection" component="btnSairLampada4">
                                <bindParam name="keyCode" value="RED"/>
                        </bind>
                        <bind role="start" component="btnSairInteratividade"/>
                        <bind role="start" component="btnLampada1"/>
                        <bind role="start" component="btnLampada2"/>
                        <bind role="start" component="btnLampada3"/>
                        <bind role="start" component="btnLampada4"/>
```

```
                    <bind role="stop" component="btnAcenderLampada4"/>
                    <bind role="stop" component="btnApagarLampada4"/>
                    <bind role="stop" component="btnSairLampada4"/>
                </link>

        </body>
</ncl>
```

**APÊNDICE B – masterConnectorBase.ncl**

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- Generated by NCL Eclipse -->
<ncl id="masterConnectorBase" xmlns="http://www.ncl.org.br/NCL3.0/EDTVProfile">
	<head>
		<connectorBase>
			<causalConnector id="onBeginStart">
				<simpleCondition role="onBegin"/>
				<simpleAction role="start" max="unbounded" qualifier="par"/>
			</causalConnector>

			<causalConnector id="onBeginStartDelay">
				<connectorParam name="delay"/>
				<simpleCondition role="onBegin"/>
				<simpleAction role="start" delay="$delay" max="unbounded"
qualifier="par"/>
			</causalConnector>

			<causalConnector id="onEndStart">
				<simpleCondition role="onEnd"/>
				<simpleAction role="start"/>
			</causalConnector>

			<causalConnector id="onKeySelectionStartNStopN">
			  <connectorParam name="keyCode" />
			  <simpleCondition role="onSelection" key="$keyCode" />
			  <compoundAction operator="par">
				<simpleAction role="start" max="unbounded"
qualifier="par"/>
				<simpleAction role="stop" max="unbounded"
qualifier="par"/>
			  </compoundAction>
			</causalConnector>

			<causalConnector id="onKeySelectionStopStart">
				<connectorParam name="keyCode"/>
				<simpleCondition role="onSelection" key="$keyCode"/>
				<compoundAction operator="par">
					<simpleAction role="stop" max="unbounded"
qualifier="par"/>
					<simpleAction role="start" max="unbounded"
qualifier="par"/>
				</compoundAction>
			</causalConnector>
			<causalConnector id="onKeySelectionStart">
				<connectorParam name="keyCode"/>
				<simpleCondition role="onSelection" key="$keyCode"/>
				<compoundAction operator="par">
					<simpleAction role="start" max="unbounded"
qualifier="par"/>
```

```
                    </compoundAction>
                </causalConnector>
            </connectorBase>
        </head>
</ncl>
```

**APÊNDICE C – acende1.lua**

```
require "util"
require "http"


local function enviaComandoLed(endKitDesn)
   --Variável que recebe a resposta
   local resposta = http.request(endKitDesn, "GET")
   --imprime a resposta do servidor
   if resposta then
      print("\n\n\n----------", resposta, "\n\n\n")
   end
end

util.coroutineCreate(enviaComandoLed, "http://192.168.0.104/?lcd=LED 1
OFF&led1=0")
```

**APÊNDICE D – acende2.lua**

```
require "util"
require "http"


local function enviaComandoLed(endKitDesn)
   --Variável que recebe a resposta
   local resposta = http.request(endKitDesn, "GET")
   --imprime a resposta do servidor
   if resposta then
      print("\n\n\n----------", resposta, "\n\n\n")
   end
end

util.coroutineCreate(enviaComandoLed, "http://192.168.0.104/?lcd=LED 2
ON&led2=1")
```

**APÊNDICE E – acende3.lua**

```
require "util"
require "http"


local function enviaComandoLed(endKitDesn)
   --Variável que recebe a resposta
   local resposta = http.request(endKitDesn, "GET")
   --imprime a resposta do servidor
   if resposta then
      print("\n\n\n----------", resposta, "\n\n\n")
   end
end

util.coroutineCreate(enviaComandoLed, "http://192.168.0.104/?lcd=LED 3
ON&led3=1")
```

**APÊNDICE F – acende4.lua**

```
require "util"
require "http"



local function enviaComandoLed(endKitDesn)
   --Variável que recebe a resposta
   local resposta = http.request(endKitDesn, "GET")
   --imprime a resposta do servidor
   if resposta then
      print("\n\n\n----------", resposta, "\n\n\n")
   end
end

util.coroutineCreate(enviaComandoLed, "http://192.168.0.104/?lcd=LED 4
ON&led4=1")
```

**APÊNDICE G – apaga1.lua**

```
require "util"
require "http"


local function enviaComandoLed(endKitDesn)
   --Variável que recebe a resposta
   local resposta = http.request(endKitDesn, "GET")
   --imprime a resposta do servidor
   if resposta then
      print("\n\n\n----------", resposta, "\n\n\n")
   end
end

util.coroutineCreate(enviaComandoLed, "http://192.168.0.104/?lcd=LED 1
OFF&led1=0")
```

**APÊNDICE H – apaga2.lua**

```
require "util"
require "http"



local function enviaComandoLed(endKitDesn)
   --Variável que recebe a resposta
   local resposta = http.request(endKitDesn, "GET")
   --imprime a resposta do servidor
   if resposta then
      print("\n\n\n----------", resposta, "\n\n\n")
   end
end

util.coroutineCreate(enviaComandoLed, "http://192.168.0.104/?lcd=LED 2
OFF&led2=0")
```

**APÊNDICE I – apaga3.lua**

```
require "util"
require "http"


local function enviaComandoLed(endKitDesn)
   --Variável que recebe a resposta
   local resposta = http.request(endKitDesn, "GET")
   --imprime a resposta do servidor
   if resposta then
      print("\n\n\n----------", resposta, "\n\n\n")
   end
end

util.coroutineCreate(enviaComandoLed, "http://192.168.0.104/?lcd=LED 3
OFF&led3=0")
```

**APÊNDICE J – apaga4.lua**

```
require "util"
require "http"



local function enviaComandoLed(endKitDesn)
   --Variável que recebe a resposta
   local resposta = http.request(endKitDesn, "GET")
   --imprime a resposta do servidor
   if resposta then
      print("\n\n\n----------", resposta, "\n\n\n")
   end
end

util.coroutineCreate(enviaComandoLed, "http://192.168.0.104/?lcd=LED 4
OFF&led4=0")
```

**ANEXOS A – http.lua**

```
---NCLua HTTP v0.9.8: Módulo para envio e recebimento de requisições HTTP em aplicações NCLua para TV Digital,
	--possibilitando também, o download de arquivos por meio de tal protocolo.<br/>
	--Utiliza a classe tcp disponibilizada no
	-- <a href="http://www.telemidia.puc-rio.br/~francisco/nclua/tutorial/index.html">Tutorial de NCLua</a>
	-- e o módulo de conversão de/para base64 disponível em <a href="http://lua-users.org/wiki/BaseSixtyFour">Lua Users</a><p/>
	--@licence: <a href="http://creativecommons.org/licenses/by-nc-sa/2.5/br/">http://creativecommons.org/licenses/by-nc-sa/2.5/br/</a>
	--@author Manoel Campos da Silva Filho<br/>
	--Professor do Instituto Federal de Educação, Ciência e Tecnologia do Tocantins<br/>
	--Mestrando em Engenharia Elétrica na Universidade de Brasília, na área de TV Digital<br/>
	--<a href="http://manoelcampos.com">http://manoelcampos.com</a>
	--@class module
	--@release Devido o módulo tcp realizar chamadas assíncronas,
	--a chamada da função tcp.execute retorna imediatamente, não aguardando
	--a resposta da requisição HTTP enviada. Para resolver tal problema e tornar
	--a chamada síncrona, permitindo que a requisição seja enviada por uma função
	--e a resposta obtida a partir do retorno da mesma função, todas as funções do módulo http
	-- (que realização conexões HTTP) devem ser chamadas dentro de uma co-rotina criada
	--pelo programador. Tal co-rotina pode ser facilmente criada com o uso
	--da função coroutineCreate do módulo util. Veja os exemplos para mais detalhes.
	--<p/>

	require "tcp"
	require "base64"
	require "util"

	local _G, tcp, print, util, base64, string, coroutine, table, type =
```

```
_G, tcp, print, util, base64, string, coroutine, table, type

module "http"

---Envia uma requisição HTTP para um determinado servidor
--@param url URL para a página que deseja-se acessar. A mesma pode incluir um número de porta,
--não necessitando usar o parâmetro port.
--@param method Método HTTP a ser usado: GET ou POST. Se omitido, é usado GET.
--onde a requisição deve ser enviada
--@param userAgent Nome da aplicação/versão que está enviando a requisição. Opcional
--@param headers Headers HTTP adicionais a serem incluídos na requisição (como por exemplo, os parâmetros de uma requisição POST). Opcional
--@param body String com o conteúdo a ser adicionado à requisição, já
--no formato URL-Encode, ou uma tabela, contendo pares de paramName=value,
--representando parâmetros a serem enviados. Opcional.
--Deve estar no formato URL Encode. Opcional
--@param user Usuário para autenticação básica. Opcional
--@param password Senha para autenticação básição. Opcional
--@param port Porta a ser utilizada para a conexão. O padrão é 80, no caso do valor ser omitido.
--A porta também pode ser especificada diretamente na URL. Se for indicada uma porta lá e aqui
--no parâmetro port, a porta da url é que será utilizada e a do parâmetro port será ignorada.
--@return Retorna a resposta da requisição HTTP
function request(url, method, userAgent, headers, body, user, password, port)
  if method == nil or method == "" then
    method = "GET"
  end
                port = port or 80
  method = string.upper(method)
```

```
        if method ~= "GET" and method ~= "POST" then
          error("Parâmetro method deve ser GET ou POST")
        end

        local co = false
        local protocol, host, port1, path = splitUrl(url)
        --Se existir uma número de porta dentro da URL, o valor do parâmetro port é
ignorado e
        --recebe a porta contida na URL.
        if port1 ~= "" then
          port = port1
        end
        if protocol == "" then
          protocol = "http://"
          url = protocol .. url
        end

        function f()
                tcp.execute(
                   function ()
                      tcp.connect(host, port)
                      --conecta no servidor
                      print("Conectado a "..host.." pela porta " .. port)

                                                --Troca espaços na URL por %20
                      url = string.gsub(url, " ", "%%20")
                      local request = {}

                                                      local fullUrl = ""
                                                if port == 80 then
                                                        fullUrl = url
                                                else
                                                        fullUrl = protocol .. host ..
":" ..port .. path
```

```
                                                    end
                --TODO: O uso de HTTP/1.1 tava fazendo com que a app congelasse
                --ao tentar obter toda resposta de uma requisição.
                --No entanto, pelo q sei, o cabeçalho Host: usado abaixo
                --é específico de HTTP 1.1, mas isto não causou problema.
                        table.insert(request, method .." "..fullUrl.." HTTP/1.0")

                        if userAgent and userAgent ~= "" then
                          table.insert(request, "User-Agent: " .. userAgent)
                        end
                        if headers and headers ~= "" then
                          table.insert(request, headers)
                        end
                        --O uso de Host na requisição é necessário
                        --para tratar redirecionamentos informados
                        --pelo servidor (código HTTP como 301 e 302)
                        table.insert(request, "Host: "..host)
                        if user and password and user ~= "" and password ~= "" then
                          table.insert(request, "Authorization: Basic " ..
                                base64.enc(user..":"..password))
                        end
                if body and body ~= "" then
                  if type(body) == "table" then
                    body = util.urlEncode(body)
                  end
                  --length of the URL-encoded body data
                  table.insert(request, "Content-Length: " .. #body.."\n")
                  table.insert(request, body)
                end
                                        table.insert(request, "\n")
                --Pega a tabela contendo os dados da requisição HTTP e gera uma string para
ser enviada ao servidor
                                        local requestStr = table.concat(request, "\n")
                        print("\n--------------------Request: \n\n"..requestStr)
```

```
                    --envia uma requisição HTTP para obter o arquivo XML do feed RSS
                    tcp.send(requestStr)
                    --obtém todo o conteúdo do arquivo XML solicitado
                    local response = tcp.receive("*a") --parâmetro "*a" = receber todos os
dados da requisição de uma vez só
                    if response == nil then
                            print("Erro ao receber dados da conexao TCP")
                else
                 print("\n\n----------------------------Resposta da requisição obtida\n\n")
                        end

                    tcp.disconnect()
                            print("\n-------------------------Desconectou")
                        coroutine.resume(co, response)
                   end
                )
                print("\n-------------------------Saiu da body function")
            end

            print("\n-------------------------Iniciar co-rotina (resume)")
            coroutine.resume(coroutine.create(f))
            print("\n-------------------------Terminou resume")
            co = coroutine.running()
            print("\n-------------------------Co-rotina suspensa (yield)")
            local response =  coroutine.yield()
            print("\n-------------------------Co-rotina finalizada (terminou yield)")
            return response
        end

        ---Baixa um arquivo xml a partir de um servidor web.
        --@param url URL para a página que deseja-se acessar. A mesma pode incluir um
número de porta,
        --não necessitando usar o parâmetro port.
```

```
--@param method Método HTTP a ser usado: GET ou POST. Se omitido, é usado GET.
--onde a requisição deve ser enviada
--@param userAgent Nome da aplicação/versão que está enviando a requisição. Opcional
--@param headers Headers HTTP adicionais a serem incluídos na requisição (como por exemplo, os parâmetros de uma requisição POST). Opcional
--@param body String com o conteúdo a ser adicionado à requisição.
--Deve estar no formato URL Encode. Opcional
--@param user Usuário para autenticação básica. Opcional
--@param password Senha para autenticação básição. Opcional
--@param port Porta a ser utilizada para a conexão. O padrão é 80, no caso do valor ser omitido.
--A porta também pode ser especificada diretamente na URL. Se for indicada uma porta lá e aqui
--no parâmetro port, a porta da url é que será utilizada e a do parâmetro port será ignorada.
--@return Em caso de erro retorna nil. Em caso de sucesso, retorna uma string contendo o código XML.
function getXml(url, method, userAgent, headers, body, user, password, port)
   --(url, method, userAgent, headers, body, user, password)
   local response = request(url, method, userAgent, headers, body, user, password, port)
   if response then
      print("Dados da conexao TCP recebidos")
      print(response)
      --Como a resposta da requisição será um arquivo XML,
      --e essa resposta conterá um cabeçalho HTTP,
      --é preciso remover esse cabeçalho e salvar
      --apenas o conteúdo XML válido. Este inicia em um sinal <
      local i = string.find(response, "?xml version=")
      if i then
        --Remove o cabeçalho HTTP da resposta,
        --deixando somente o código XML
```

```
            response = string.sub(response, i-1, #response)

            --Apenas para depuração, pois a função usa o módulo io,
            --não disponível no Ginga
            --util.createFile(response, "response.xml")
          else
            print("O marcador de início do XML (<) não foi encontrado")
            response = nil
          end
        end
        return response
      end

      ---Envia uma requisição HTTP para uma URL que represente um arquivo,
      --e então faz o download do mesmo.
      --@param url URL para a página que deseja-se acessar. A mesma pode incluir um
número de porta,
      --não necessitando usar o parâmetro port.
      --@param fileName Caminho completo para salvar o arquivo localmente.
      --Só deve ser usado para depuração, pois passando-se
      --um nome de arquivo, fará com que a função use o módulo io,
      --não disponível no Ginga. Para uso em ambientes
      --reais (Set-top boxes), deve-se passar nil para o parâmetro
      --@param userAgent Nome/versão do cliente http. Opcional
      --@param user Usuário para autenticação básica. Opcional
      --@param password Senha para autenticação básição. Opcional
      --@param port Porta a ser utilizada para a conexão. O padrão é 80, no caso do valor
ser omitido.
      --A porta também pode ser especificada diretamente na URL. Se for indicada uma
porta lá e aqui
      --no parâmetro port, a porta da url é que será utilizada e a do parâmetro port será
ignorada.
      --@return Se o parâmetro fileName for diferente de nil,
      --retorna true em caso de sucesso, e false em caso de erro.
```

```
--Caso contrário, retorna o conteúdo do arquivo, caso o mesmo
--seja obtido, caso contrário, retorna false.
function getFile(url, fileName, userAgent, user, password, port)
   --(url, method, userAgent, headers, body, user, password)
   local response = request(url, "GET", userAgent, nil, nil, user, password, port)

   if response then
      --print(response, "\n")
      print("Dados da conexao TCP recebidos")
      --Verifica se o código de retorno é OK
      if string.find(response, "200 OK") then
         --O corpo da mensagem, que contém o arquivo
         --a ser baixado, inicia após duas quebras
         --de linha (cada quebra de linha
         --possui dois caracteres: \r\n)
         local i = string.find(response, "\r\n\r\n")
         --A adição de 4 na posição i é usado para
         --pular as duas quebras de linha (cada quebra
         --possui dois caracteres: \r\n) e obter
         --o conteúdo do arquivo, que está após elas
         response = string.sub(response, i+4, #response)
         if fileName == nil then
            return response
         else
            util.createFile(response, fileName, true)
            return true
         end
      end
      return false
   else
      print("Erro ao receber dados da conexao TCP")
      return false
   end
end
```

```
---Obtém o valor de um determinado campo de uma resposta HTTP
--@param response Conteúdo da resposta HTTP de onde deseja-se extrair
--o valor de um campo do cabeçalho
--@param fieldName Nome do campo no cabeçalho HTTP
function getHttpHeader(response, fieldName)
  --Procura a posição de início do campo
  local i = string.find(response, fieldName .. ":")
  --Se o campo existe
  if i then
    --procura onde o campo termina (pode terminar com \n ou espaço
    --a busca é feita a partir da posição onde o campo começa
    local fim = string.find(response, "\n", i) or string.find(response, " ", i)
    return string.sub(response, i, fim)
  else
    return nil
  end
end

---Obtém uma URL e divide a mesma em protocolo, host, porta e path
--@param url URL a ser dividida
--@return Retorna o protocolo, host, porta e o path obtidas da URL.
--Caso algum destes valores não exita na URL, é retornada uma string vazia no seu
lugar.
function splitUrl(url)
  --TODO: O uso de expressões regulares seria ideal nesta função
  --por meio de string.gsub

  local protocolo = ""
  local separadorProtocolo = "://"
  --procura onde inicia o nome do servidor, que é depois do separadorProtocolo
  local i = string.find(url, separadorProtocolo)
  if i then
    protocolo = string.sub(url, 1, i+2)
```

```
                --soma o tamanho do separadorProtocolo em i para pular o
separadorProtocolo,
            --que identifica o protocolo,
                --e iniciar na primeira posição do nome do host
                i=i+#separadorProtocolo
        else
            --se a URL não possui um protocolo, então o nome
            --do servidor inicia na primeira posição
            i = 1
        end

        local host, porta, path = "", "", ""
        --procura onde termina o nome do servidor,
        --na primeira barra após o separadorProtocolo
        local j = string.find(url, "/", i)
        --se encontrou uma barra, copia o nome do servidor até a barra,
        --pois após ela, é o path
        if j then
            host = string.sub(url, i, j-1)
            path  = string.sub(url, j, #url)
        else
            --senão, não há um path após o nome do servidor, sendo o restante da url
            --o nome do servidor
            host = string.sub(url, i)
        end

        --verifica se há um número de porta dentro do host (a porta vem após os dois pontos)
        i = string.find(host, ":")
        if i then
          porta = string.sub(host, i+1, #host)
          host = string.sub(host, 1, i-1)
        end

        return protocolo, host, porta, path
```

```
end
```

**ANEXOS B – tcp.lua**

```
---Módulo para realização de conexões TCP.
	--Utiliza co-rotinas de lua para simular multi-thread.
	--Fonte: <a href="http://www.telemidia.puc-
rio.br/~francisco/nclua/index.html">Tutorial de NCLua</a>
	--@class module

	-- TODO:
	-- * nao aceita `tcp.execute` reentrante

	--Declara localmente módulos e funçõe globais pois, ao definir
	--o script como um módulo, o acesso ao ambiente global é perdido
	local _G, coroutine, event, assert, pairs, type, print
	   = _G, coroutine, event, assert, pairs, type, print
	local s_sub = string.sub

	module 'tcp'

	---Lista de conexões TCP ativas
	local CONNECTIONS = {}

	---Obtém a co-rotina em execução
	--@returns Retorna o identificador da co-rotina em execução
	local current = function ()
	   return assert(CONNECTIONS[assert(coroutine.running())])
	end

	---(Re)Inicia a execução de uma co-rotinas. Estas, são criadas
	--suspensas, assim, é necessário resumí-las para entrarem
	--em execução.
	--@param co Co-rotina a ser resumida
	--@param ... Todos os parâmetros adicionais
	--são passados à função que a co-rotina executa.
	--Quando a co-rotina é suspensa com yield, ao ser resumida
```

```
--novamente, estes parâmetros extras passados na chamada de resume
--são retornados pela yield. Isto é usado, por exemplo, na co-rotina da
--função receive, para receber a resposta de uma requisição TCP. Assim,
--ao iniciar, co-rotina da função é suspensa para que fique aguardando
--a resposta da requisição TCP. Quando a função tratadora de eventos (handler)
--recebe os dados, ela resume a co-rotina da função receive. Os dados
--recebidos são passados à função resume, e estes são retornados pela função
--yield depois que a co-rotina é reiniciada.
local resume = function (co, ...)
   assert(coroutine.status(co) == 'suspended')
   assert(coroutine.resume(co, ...))
   if coroutine.status(co) == 'dead' then
      CONNECTIONS[co] = nil
   end
end

---Função tratadora de eventos. Utilizada para tratar
--os eventos gerados pelas chamadas às funções da classe tcp.
--@param evt Tabela contendo os dados do evento capturado
function handler (evt)
   if evt.class ~= 'tcp' then return end

   if evt.type == 'connect' then
      for co, t in pairs(CONNECTIONS) do
         if (t.waiting == 'connect') and
            (t.host == evt.host) and (t.port == evt.port) then
            t.connection = evt.connection
            t.waiting = nil
            --Continua a execução da co-rotina,
            --fazendo com que a função connect, que causou
            --o disparo do evento connect, capturado
            --por esta função (handler), seja finalizada.
            resume(co)
            break
```

```
        end
    end
    return
end

if evt.type == 'disconnect' then
    for co, t in pairs(CONNECTIONS) do
        if t.waiting and
           (t.connection == evt.connection) then
            t.waiting = nil
            resume(co, nil, 'disconnected')
        end
    end
    return
end

        --Evento disparado quando existem dados a serem recebidos
        --após a chamada da função receive.
if evt.type == 'data' then
    for co, t in pairs(CONNECTIONS) do
        if (t.waiting == 'data') and
        (t.connection == evt.connection) then
            --O atributo value da tabela evt contém os dados
            --recebidos. Assim, continua a execução da função que disparou
            --este evento (função receive). O valor de evt.value
            --é retornado pela função coroutine.yield, chamada
            --dentro da função receive (que ficou suspensa
            --aguardando os dados serem recebidos).
            --Desta forma, dentro da função receive, o retorno
            --de coroutine.yield contém os dados recebidos.
            resume(co, evt.value)
        end
    end
    return
```

```
    end
end
event.register(handler)



---Função que deve ser chamada para iniciar uma conexão TCP.
--@param f Função que deverá executar as rotinas
--para realização de uma conexão TCP, envio de requisições
--e obtenção de resposta.
--@param ... Todos os parâmetros adicionais
--são passados à função que a co-rotina executa.
--@see resume
function execute (f, ...)
    resume(coroutine.create(f), ...)
end

---Conecta em um servidor por meio do protocolo TCP.
--A função só retorna quando a conexão for estabelecida.
--@param host Nome do host para conectar
--@param port Porta a ser usada para a conexão
function connect (host, port)
    local t = {
        host    = host,
        port    = port,
        waiting = 'connect'
    }
    CONNECTIONS[coroutine.running()] = t

    event.post {
        class = 'tcp',
        type  = 'connect',
        host  = host,
        port  = port,
```

```
        }

        --Suspende a execução da co-rotina.
        --A função atual (connect) só retorna quando
        --a co-rotina for resumida, o que ocorre
        --quando o evento connect é capturado
        --pela função handler.
        return coroutine.yield()
    end

    ---Fecha a conexão TCP e retorna imediatamente
    function disconnect ()
        local t = current()
        event.post {
            class      = 'tcp',
            type       = 'disconnect',
            connection = assert(t.connection),
        }
    end

    ---Envia uma requisição TCP ao servidor no qual se está conectado, e retorna imediatamente.
    --@param value Mensagem a ser enviada ao servidor.
    function send (value)
        local t = current()
        event.post {
            class      = 'tcp',
            type       = 'data',
            connection = assert(t.connection),
            value      = value,
        }
    end
```

```
---Recebe resposta de uma requisição enviada previamente
--ao servidor.
--@param pattern Padrão para recebimento dos dados.
--Se passado *a, todos os dados da resposta são
--retornados de uma só vez, sem precisar fazer
--chamadas sucessivas a esta função.
--Se omitido, os dados vão sendo retornados parcialmente,
--sendo necessárias várias chamadas à função.
function receive (pattern)
   pattern = pattern or '' -- TODO: '*l'/number
   local t = current()
   t.waiting = 'data'
   t.pattern = pattern

   if s_sub(pattern, 1, 2) ~= '*a' then
      --Suspende a execução da função, até que
      --um bloco de dados seja recebido.
      --Ela só é resumida depois que
      --a função handler (tratadora de eventos)
      --receber um bloco de dados. Nesse momento,
      --a função receive retorna o bloco de dados.
      --Tendo entrado neste if, o parâmetro pattern será
      --diferente de '*a', logo, serão necessárias
      --várias chamadas sucessívas a receive para obter
      --toda a resposta da requisição enviada previamente
      --por meio da função send.
      --A função receive retorna nil quando não houver
      --mais nada para ser retornado.
      return coroutine.yield()
   end

   --Chegando aqui, é porque o parâmetro pattern é igual
   --a '*a', indicando que a função só deve retornar depois
   --que toda a resposta da requisição enviada previamente,
```

```
    --por meio da função send, tiver sido retornada.
    local all = ''
    while true do
        --Suspende a execução da função, até que
        --um bloco de dados seja recebido.
        --Ela só é resumida depois que
        --a função handler (tratadora de eventos)
        --receber um bloco de dados. Nesse momento,
        --a função receive retorna o bloco de dados.
        --Se o resultado for nil, a função finaliza
        --devolvendo todos os blocos de resposta recebidos,
        --concatenados. Não sendo nil, a função suspende a execução
        --até receber novo bloco.
        local ret = coroutine.yield()
        if ret then
            all = all .. ret
        else
            return all
        end
    end
end
```

**ANEXOS C – util.lua**

```
---Módulo de funções de uso geral v1.3.3
--@author Manoel Campos da Silva Filho
--<a href="http://manoelcampos.com">http://manoelcampos.com</a>
--@license Atribuição-Uso não-comercial-Compartilhamento pela mesma licença
http://creativecommons.org/licenses/by-nc-sa/2.5/br/
--@class module

local _G, io, print, string, coroutine, canvas, tonumber, pairs, type =
    _G, io, print, string, coroutine, canvas, tonumber, pairs, type

module "util"

---Clona uma tabela
--@param tb Tabela ser clonada
--@return Retorna a nova tabela
function cloneTable(tb)
  local result = {}
  for k, v in pairs(tb) do
    result[k] = v
  end
  return result
end

---Imprime uma tabela, de forma recursiva
--@param tb A tabela a ser impressa
--@param level Apenas usado internamente para
--imprimir espaços para representar os níveis
--dentro da tabela.
function printable(tb, level)
  level = level or 1
  local spaces = string.rep(' ', level*2)
  for k,v in pairs(tb) do
    if type(v) ~= "table" then
```

```
        print(spaces .. k..'='..v)
      else
        print(spaces .. k)
        level = level + 1
        printable(v, level)
      end
  end
end

---Quebra uma string para que a mesma tenha linhas
--com um comprimento máximo definido, não quebrando
--a mesma no meio das palavras.
--@param Text String a ser quebrada
--@param maxLineSize Quantidade máxima de caracteres por linha
--@return Retorna uma tabela onde cada item é uma linha
--da string quebrada.
function breakString(text, maxLineSize)
  local t = {}
  local str = text
  local i, fim, countLns = 1, 0, 0

  if (str == nil) or (str == "") then
    return t
  end

  str = string.gsub(str, "\n", " ")
  str = string.gsub(str, "\r", " ")

  while i <= #str do
    countLns = countLns + 1
    if i > #str then
      t[countLns] = str
    else
      fim = i+maxLineSize-1
```

```
        if fim > #str then
          fim = #str
        else
                --se o caracter onde a string deve ser quebrada
                --não for um espaço, procura o próximo espaço
                if string.byte(str, fim) ~= 32 then
                  fim = string.find(str, ' ', fim)
                  if fim == nil then
                    fim = #str
                  end
                end
        end
        t[countLns]=string.sub(str, i, fim)
        i=fim+1
      end
    end

    return t
end



---Imprime um texto na tela, quebrando o mesmo nos limites
--horizontais da área do canvas.
--@param areaWidth Largura a área disponível para impressão
--@parma x Posição x onde o texto deve ser impresso
--@param initialY Posição y inicial a ser impresso o texto
--@param text Texto a ser impresso, sendo quebrado em
--linhas para caber horizontalmente na largura
--definida para impressão
function paintBreakedString(areaWidth, x, initialY, text)
    --Text Width e Text Height de um caractere minúsculo
    local tw, th = canvas:measureText("a")

    --Estima quantos caracteres cabem dentro da largura
```

```
    --definida para a exibição de uma mensagem do Twitter
    local charsByLine = tonumber(string.format("%d", areaWidth / tw))

    --Quebra o texto em diversas linhas,
    --gerando uma tabela onde cada item é uma linha que
    --foi quebrada. Isto é usado para que o texto seja
    --exibido sem sair da tela.
    local textTable = breakString(text, charsByLine)
    local y = initialY
    --Percorre a tabela gerada a partir da quebra do texto
    --em linhas, e imprime cada linha na tela
    for k,ln in pairs(textTable) do
        canvas:drawText(x, y, ln)
        y = y + th
        print("----------------------"..ln)
    end
end

---Desenha um texto na tela
--@param x Posição horizontal a ser impresso o texto
--@param y Posição vertical a ser impresso o texto
--@param text texto a ser desenhado
--@param fontName Nome da fonte a ser utilizada para imprimir o texto. Opcional
--@param fontSize Tamanho da fonte. Opcional
--@param fontColor Cor da fonte. Opcional
function paintText(x, y, text, fontName, fontSize, fontColor)
    if fontName and fontSize then
      canvas:attrFont(fontName, fontSize)
    end
    if fontColor then
      canvas:attrColor(fontColor)
    end

    --width e height do canvas
```

```
    local cw, ch = canvas:attrSize()
    canvas:drawText(x, y, text)
end


--DEVIDO AO USO DO MÓDULO IO, ESTA FUNÇÃO
--NÃO É PERMITIDA NO CONTEXTO DE TVD,
--POIS O MÓDULO IO NÃO FAZ PARTE DO GINGA
---Verifica se um arquivo existe
--@param fileName Nome do arquivo a ser verificado
--@return Retorna true se o arquivo existir
--[[
function fileExists(fileName)
  local file = io.open(fileName)
  if file then
    io.close(file)
    return true
  else
    return false
  end
end
--]]


---Cria um arquivo com o conteúdo informado em text.
--Devido a função utilizar o módulo io, não disponível
--no Ginga, a mesma deve ser utilizada apenas
--para depuração, em ambientes de teste.
--Se o arquivo já existir, substitui.
--@param content Conteúdo a ser adicionado no arquivo
--@param fileName Nome do arquivo a ser gerado.
--@return Retorna true caso o arquivo seja salvo com sucesso.
function createFile(content, fileName, binaryFile)
    binaryFile = binaryFile or false
```

```
    local mode = ""
    if binaryFile then
      mode = "w+b"
    else
      mode = "w+"
    end
    file, err = io.open(fileName, mode)
    if file == nil then
          print("Erro ao abrir arquivo "..fileName.."\n".. err)
          return false
    else
          print("Arquivo", fileName, "criado com sucesso")
       file:write(content)
       file:close()
       return true
    end
end

---Função para converter uma tabela para o formato URL-Encode,
--também chamado de Percent Encode, segundo RFC 3986.
--Fonte: http://www.lua.org/pil/20.3.html. Gerada a partir das funções
--escape e encode, gerando uma só.
--@param t Tabela contendo os pares param=value
--que representam os parâmetros a serem codificados para o formato URL-Encode,
--ou String contendo o texto a ser codificado.
--@return Retorna uma string codificada em URL-Encode
function urlEncode(t)
          local function escape (s)
            s = string.gsub(s, "([&=+%c])", function (c)
                return string.format("%%%02X", string.byte(c))
              end)
            s = string.gsub(s, " ", "+")
            return s
          end
```

```
        if type(t) == "string" then
          return escape(t)
        else
                local s = ""
                for k,v in pairs(t) do
                  s = s .. "&" .. escape(k) .. "=" .. escape(v)
                end
                return string.sub(s, 2)     -- remove first `&'
        end
      end

      --Conta o total de elementos em uma tabela indexada com chaves string,
      --pois o operador # não funciona para obter o total de elementos de tais tabelas.
      --@param Tabela a ser contato o total de elementos
      --@return Retorna o total de elementos da tabela
      function count(tb)
        local i = 0
        for k, v in pairs(tb) do
          i = i + 1
        end
        return i
      end

      ---Verifica se uma tabela contém apenas um elemento
      --@param tb Tabela ser verificada
      --@return Retorna true caso a tabela contenha apenas um elemento.
      function hasSingleElement(tb)
        --Para tabelas mais complexas, geradas a partir de um XML este código não
funciona,
        --congelando a aplicação.
        --local k=next(tb)
        --return k~=nil and next(tb,k)==nil
```

```
    local i = 0
    for k, v in pairs(tb) do
        i = i + 1
        if i > 1 then
            return false
        end
    end

    return i == 1
end

--Obtém o primeiro elemento de uma tabela
--@param Tabela de onde deverá ser obtido o primeiro elemento
--@return Retorna o primeiro elemento da tabela
function getFirstElement(tb)
   if type(tb) == "table" then
       --O uso da função next não funciona para pegar o primeiro elemento. Trava aqui
      --k, v = next(tb)
      --return v
      for k, v in pairs(tb) do
          return v
      end
   else
     return tb
   end
end

--Obtém a primeira chave de uma tabela
--@param Tabela de onde deverá ser obtido o primeiro elemento
--@return Retorna a primeira chave da tabela
function getFirstKey(tb)
   if type(tb) == "table" then
       --O uso da função next não funciona para pegar o primeiro elemento. Trava aqui
      --k, v = next(tb)
```

```
        --return k
        for k, v in pairs(tb) do
            return k
        end
    else
     return tb
    end
end

---Percorre uma tabela recursivamente. Se ela contém apenas um elemento,
--a tabela a qual ele pertence (a externa) é eliminada, ficando apenas a tabela interna,
--passando esta a ser a tabela principal. Repete isto até chegar no item mais interno da
tabela.
--Assim, uma tabela como nivel1 = { nivel2 = nivel3 = {desc = "mouse", valor = 99}}
--se transforma em {desc="mouse", valor = 99}
--Outra tabela como nivel1 = { nivel2 = nivel3 = {pais = "Brasil"}}
--se transforma em pais = "Brasil", sem nenhuma tabela.
--@param xmlTable Table lua gerada a partir de código XML
--@return Retorna a nova tabela simplificada. Se dentro de toda a estrutura
--da tabela original só existia um campo com valor, tal valor é retornado
--como uma variável simples.
function simplifyTable(tb)
  local tmp = tb
  ---[[
  while type(tmp) == "table" and hasSingleElement(tmp) do
    tmp = getFirstElement(tmp)
  end
  --]]
  return tmp
end
```

```
---Cria uma co-rotina para execução de uma determinada função.
--@param f Função body a ser executada pela co-rotina
--@param ... Parâmetros adicionais que serão passados à função
--body da co-rotina, passada no parâmetro f.
function coroutineCreate(f, ...)
    coroutine.resume(coroutine.create(f), ...)
end
```

**ANEXOS D – base64.lua**

```
#!/usr/bin/env lua

---<a href="http://lua-users.org/wiki/BaseSixtyFour">http://lua-
users.org/wiki/BaseSixtyFour</a><br/>
--Lua 5.1+ base64 v3.0 (c) 2009 by Alex Kloss <alexthkloss@web.de><br/>
--licensed under the terms of the LGPL2<br/>
--@class module

-- character table string
local
b='ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/'

--local _G, string, ipairs, print = _G, string, ipairs, print



---Encode a string to base64 format
--@param data String to be encoded
--@return Returns the encoded string
function enc(data)
    return ((data:gsub('.', function(x)
        local r,b='',x:byte()
        for i=8,1,-1 do r=r..(b%2^i-b%2^(i-1)>0 and '1' or '0') end
        return r;
    end)..'0000'):gsub('%d%d%d?%d?%d?%d?', function(x)
        if (#x < 6) then return '' end
        local c=0
        for i=1,6 do c=c+(x:sub(i,i)=='1' and 2^(6-i) or 0) end
        return b:sub(c+1,c+1)
    end)..({ '', '==', '=' })[#data%3+1])
end

---Decode a string from base64 format
```

```
--@param data String to be decoded
--@return Returns the decoded string
function dec(data)
   data = string.gsub(data, '[^'..b..'=]', '')
   return (data:gsub('.', function(x)
      if (x == '=') then return '' end
      local r,f='',(b:find(x)-1)
      for i=6,1,-1 do r=r..(f%2^i-f%2^(i-1)>0 and '1' or '0') end
      return r;
   end):gsub('%d%d%d?%d?%d?%d?%d?%d?', function(x)
      if (#x ~= 8) then return '' end
      local c=0
      for i=1,8 do c=c+(x:sub(i,i)=='1' and 2^(8-i) or 0) end
      return string.char(c)
   end))
end

-- command line if not called as library
if (arg ~= nil) then
        local func = 'enc'
        for n,v in ipairs(arg) do
                if (n > 0) then
                        if (v == "-h") then print "base64.lua [-e] [-d] text/data" break
                        elseif (v == "-e") then func = 'enc'
                        elseif (v == "-d") then func = 'dec'
                        else print(_G[func](v)) end
                end
        end
else
        module('base64',package.seeall)
end
```

**ANEXOS E – Código Kit de Desenvolvimento ACEPIC NET**

GRAVADO NO CD QUE ACOMPANHA A MONOGRAFIA.