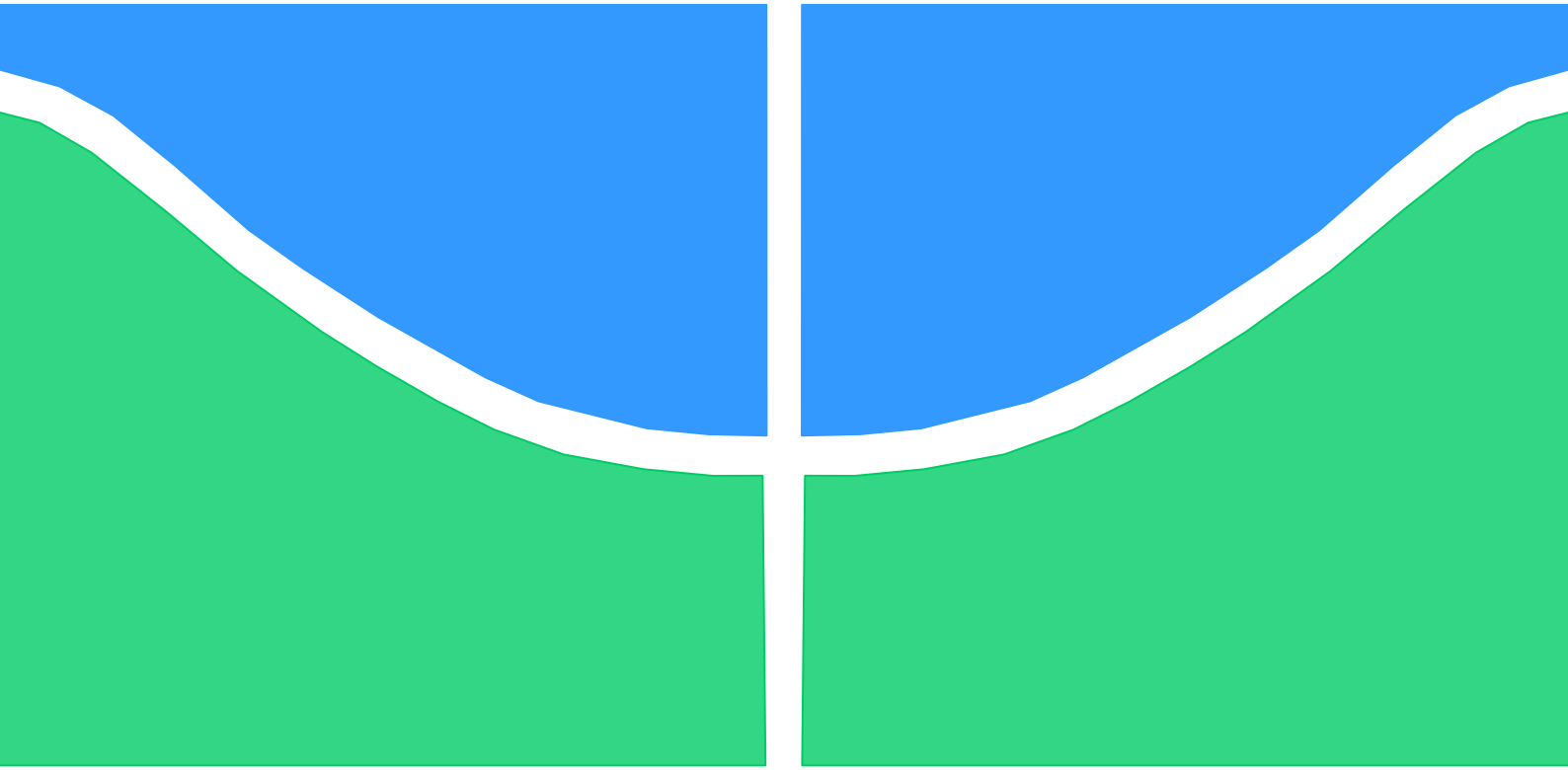

PROJETO DE GRADUAÇÃO

IMPLEMENTAÇÃO DE RÁDIO DEFINIDO POR SOFTWARE EM ARQUITETURA EMBARCADA

Ana Lúcia Alvares Alves
Samantha Irineu Andrade de Souza

Brasília, dezembro de 2007

UNIVERSIDADE DE BRASILIA

FACULDADE DE TECNOLOGIA

UNIVERSIDADE DE BRASILIA

Faculdade de Tecnologia

PROJETO DE GRADUAÇÃO

IMPLEMENTAÇÃO DE RÁDIO DEFINIDO POR SOFTWARE EM ARQUITETURA EMBARCADA

Ana Lúcia Alvares Alves
Samantha Irineu Andrade de Souza

Relatório submetido como requisito parcial para obtenção
do grau de Engenheiro Eletricista

Banca Examinadora

Prof. Adson Ferreira Rocha, UnB/ ENE (Orientador) _____

Prof. Janaína Gonçalves Guimarães, UnB/ ENE _____

Prof. Leonardo R.A.X. Menezes, UnB/ ENE _____

Dedicatórias

Dedico aos meus pais, Pedro e Magdala e aos meus amados irmãos, Larissa e Luciano.

Samantha Irineu Andrade de Souza

Dedico aos meus queridos pais Sandra Lúcia e Fernando Irineu..

Ana Lúcia Alves Álvares

Agradecimentos

Agradeço a Deus, pela minha saúde, disposição e coragem durante todo o período da faculdade. Aos meus pais por terem se esforçado muito durante toda a vida para me darem educação e conforto. Aos meus irmãos, principalmente minha irmã, por ter me emprestado o computador e dormido com a luz acesa ao som do teclado e também pela vida feliz que passamos juntos.
À minha querida companheira de projeto final, Samantha, que sempre foi uma grande amiga e se mostrou muito companheira mesmo nos momentos difíceis do projeto. Ao querido professor Eduardo Wolski, com quem tive a honra de trabalhar, pelo auxílio dado durante todo o projeto e pelo tempo cedido tão gentilmente.
Aos meus colegas da graduação, especialmente os meninos do semestre 2/2003, com quem fiz a maior parte do curso e meus amigos Pedro Henrique Milhomem, Rafael Ribeiro, Samantha Andrade e Rafael Schena. Aos meus amigos da vida (especialmente Bruna, Nadja, Fernanda, Carla, Nayara, Ana Lívia, Bia, João Guilherme, Keyne, Gui, Guilherme Coelho, Brenda ,Leilane Gabi, Emerson, Lorenzzo) que fizeram meu curso ser muito mais divertido fora da Universidade e a um amigo muito especial, Juan, por me servir de exemplo de pessoa e por me dar tanto apoio mesmo de tão longe.
Agradeço também aos meus colegas de Embratel, que me ensinaram tanto quanto ou mais que a Universidade. E, especialmente aos meus professores da Universidade de Brasília, principalmente aos professores do Departamento de Engenharia Elétrica.

Ana Lúcia Alves Álvares

Primeiramente, agradeço a Deus por ter colocado pessoas tão boas na minha vida para compartilhar alegrias e momentos de dificuldade. À minha mãe Sandra e meu pai Fernando, por serem meus exemplos de vida e que, mesmo de longe, acompanharam meu esforço e dedicação à Universidade, sempre com palavras de carinho. Aos meus irmãos Fernanda e Pedro Henrique que muito me ensinaram e ajudaram sempre que foi preciso.
Ao meu namorado Salles que agüentou meu mau-humor nos fins de semestres e me incentivou muito a estudar, fazendo companhia na biblioteca até tarde da noite.
Agradeço muito ao Professor Eduardo Wolski, uma pessoa brilhante que nos acompanhou durante cada passo do projeto e que tenho muito carinho e admiração.
À amiga e companheira de projeto Ana Lúcia, sempre otimista e dedicada, que me acompanhou nos períodos mais difíceis dentro e fora da Universidade.
Agradeço ao grupo de professores da UnB, em especial, ao Professor Lúcio Martins e Professor Ivan Camargo, que sempre atenderam minhas dúvidas com dedicação e carinho. Professores que muito me ensinaram e que, com certeza, levo seus exemplos para minha vida profissional. Enfim, agradeço aos amigos que fiz na Engenharia, um grupo que sempre se uniu nos momentos mais difíceis e que me proporcionou muitas risadas nesses 5 anos juntos.

Samantha Irineu Andrade de Souza

SUMÁRIO

LISTA DE FIGURAS

LISTA DE SÍMBOLOS

SIGLAS

AO	Ambiente Operacional
API	*Application Programming Interfaces*
ASICs	*Application Specific Integrated Circuit*
AWGN	*Additive White Gaussian Noise*
BPSK	*Binary Phase Shift Keying*
CD	Detector de colisão
CF	*Core Framework*
CORBA	*Common Object Request Broker Architecture*
COTS	Commercial *off-the-shelf*
DA	Digital-analógico
DoD	Departamento de Defesa Americano
DPD	Device Package Descriptor
DSP	*Digital Signal Processing*
FPGA	*Field-Programmable Gate Arrays*
GPDS	Grupo de Processamento Digital de Sinais
GPP	*General Purpose Processor*
I	*In phase*
IDL	*Interface Definition Language*
IEEE	*Institute of Electrical and Eletronics Engineers*
JPO	*Joint Program Office*
JTRS	*Joint Tactical Radio System*
LRU	*Line Replaceable Units*
MMU	*Memory Management Units*
MPRG	*Mobile and Portable Radio Research Group*
OMG	*Object Management Group*
OO	Orientado ao Objeto
ORB	*Object Request Broker*
OWD	*OSSIE Waveform Develope*r
PC	*Personal Computer*
PI	Propriedade Intelectual
PMCS	Sistema de Comunicação Modular Programável
POSIX	*Portable Operating System Interface*
PSK	*Phase Shift Keying*
Q	*Quadrature*
QPSK	*Quadrature Phase Shift Keying*
RDS	Rádio Definido por Software
RISC	*Reduced Instruction Set Computing*
RT	Tempo Real

RTOS	Sistema Operacional em Tempo Real
SCA	*Software Communication Architecture*
SCD	*Software Component Descriptor*
SO	Sistemas Operacionais (embarcados)
SPD	*Software Package Descriptor*
TI	Tecnologia da Informação
USB	*Universal Serial Bus*
USRP	*Universal Software Radio Peripheral*
VMWare	Máquina Virtual
WITS	*Wireless Information Transfer System*
XML	*Extensible Markup Language*

1 INTRODUÇÃO

1.1 IMPORTANCIA DO ESTUDO E MOTIVAÇÃO

Os padrões atuais dos sistemas de comunicações móveis estão cada vez mais diversificados (padrões distintos), comprometendo, com isso, sua abrangência, cuja compatibilidade é dependente do hardware. Nesse contexto que surge a necessidade do desenvolvimento de rádios cada vez mais flexíveis e cada vez menos dependentes de sua estrutura de hardware. A tendência, daqui para frente será a obtenção de interoperabilidade através da simples alteração de software. Esse é o principal objetivo da tecnologia que será enfoque desse projeto: o Rádio Definido por Software (RDS).

Um Rádio Definido por Software, **como será visto no Capítulo 2, inteiramente dedicado ao assunto,** é um rádio cujos parâmetros de utilização do espectro são determinados por software ou por hardware reconfigurável. O método através do qual essa reconfiguração irá ocorrer é variável. Pode ser introduzido no equipamento através da interface de rádio, mas igualmente através de um fio ou disco, ou, ainda, codificando o software ou o netlist diretamente através da interface de programação do dispositivo. Além da reconfigurabilidade, uma grande vantagem oferecida pelo desenvolvimento do Rádio Definido por Software é a diminuição dos custos de produção acarretados pela produção de dispositivos de hardware mais padronizados, com suas funcionalidades podendo ser introduzidas pelo software carregado no dispositivo. Comercialmente falando, haverá uma redução de custos, uma vez que os terminais físicos não deverão mais ser descartados a fim de que haja compatibilidade entre tecnologias diversas. Do ponto de vista militar, operações que envolvem grupos distintos operando com seus padrões específicos de rádio, poderiam convergir para uma interoperabilidade através do RDS.

As grandes motivações baseadas em interesses militares e comerciais fazem com que grande parte da pesquisa seja desenvolvida em instituições governamentais ou em grandes empresas privadas (Motorola, Siemens, Nokia, Samsung etc.)

Por se tratar de uma tecnologia recente, ela ainda está em fase de estudos e adaptações. Um exemplo disso é o desenvolvimento dos softwares utilizados. Um dos softwares mais utilizados em projetos de RDS é o GNU RADIO, que foi abordado no projeto final de um aluno da Universidade de Brasília [6]. O programa utilizado nesse projeto foi o OSSIE, uma implementação baseada em uma arquitetura denominada *Software Communication Architecture* (SCA).

1.2 OBJETIVOS

Neste trabalho, tem-se como principal objetivo a análise do programa OSSIE, suas ferramentas e principais aplicações e, principalmente, a arquitetura que deu origem a ele (SCA). Esse estudo é de grande valia, pois se trata de um programa mais arrojado e ambicioso que o Gnu Radio, possibilitando uma maior quantidade de opções no que se refere à implementação de um RDS.

Tínhamos como objetivos iniciais também, a implementação de um dispositivo embarcado, importantíssimo para a consideração dessa nova tecnologia RDS, uma vez que ela será muito utilizada em ambientes, na sua maioria, móveis. Entretanto, devido a problemas e erros que serão mais para frente enunciados, nos ateremos apenas a uma explicação sobre sistemas embarcados, servindo como uma sugestão a sua implementação em projetos futuros dentro da Universidade de Brasília.

Resumidamente, temos que os principais objetivos são:

- estudar conceitos de RDS;
- estudar as especificações de SCA;
- estudar o funcionamento e a aplicabilidade de sistemas embarcados;
- conhecer a instalação e a implementação de formas de onda usando OSSIE;
- avaliar o OSSIE como uma ferramenta de implementação do SCA.

1.3 ESTRUTURA DO TEXTO

O texto será dividido em 4 Capítulos. O primeiro capítulo mostrou uma breve introdução a respeito dos estudos realizados e do conteúdo que será abordado nesta monografia. No Capítulo 2 será possível tratar de forma mais profunda o assunto Rádio Definido por Software. O Capítulo 3 será responsável por uma explicação ampla sobre a arquitetura SCA (*Software Communication Architecture*). No Capítulo 4 abordamos o tema Sistemas Embarcados finalizando, assim, os três capítulos responsáveis pela Revisão Bibliográfica. O Capítulo 5 refere-se a parte prática do projeto, que explicará o funcionamento do OSSIE e exemplificará o uso desta ferramenta através de dois laboratórios. Por fim, o Capítulo 6 irá concluir os estudos realizados.

2 RÁDIO DESENVOLVIDO POR *SOFTWARE*

2.1 CONCEITOS E ARQUITETURA GERAL

O conceito de reconfigurabilidade é o propulsor para a implementação de Rádios Definidos por Software (RDS). Estes rádios, por sua vez, visam dar grande flexibilidade a aplicações e mesmo a características operacionais de rádio-frequência. A frequência de operação já não depende mais de capacitores variáveis ou do circuito eletrônico do aparelho: ela é definida por um programa que controla o funcionamento do equipamento.

Rádio Definido por Software é uma tecnologia emergente que está transformando profundamente a engenharia do sistema de rádio da forma como era conhecida. Segundo a definição de Joseph Mitola [1], "um Rádio Definido por Software (RDS) é, um rádio cuja modulação das formas de onda do canal são definidas em software. Isto é, as formas de onda são geradas como sinais digitais amostrados, convertidas de digitais para analógicas por um conversor DA de banda larga, que captura todos os canais do nó do RDS. O receptor, por sua vez, captura o sinal faz um abaixamento de frequência, e demodula a forma de onda do canal por meio de um software que roda sobre um processador de uso geral".

Consiste, em sua maioria, nos mesmos blocos funcionais básicos de qualquer sistema de comunicação digital. O sistema RDS projeta novas demandas para esses blocos a fim de fornecer banda múltipla, múltiplos serviços de operação e necessidade de reconfiguração para suportar vários padrões de interfaces aéreas.

Para alcançar a flexibilidade necessária, a fronteira do processamento digital deve ser movida o mais próxima possível da antena (fazendo isso, a maioria das funcionalidades do rádio pode ser executada processando o sinal digitalmente, permitindo reutilizar a plataforma de hardware; isso permite uma grande reconfigurabilidade se comparada aos rádios convencionais) e as aplicações específicas de circuitos integrados, que são usadas no processamento do sinal de banda-básica, devem ser substituídas por implementações programáveis.

Um sistema de RDS é um rádio cujos parâmetros de utilização dos espectros são determinados por software ou por hardware reconfigurável. Suporta uma grande quantidade de frequências diferentes, interfaces aéreas e software de aplicação; e muda sua configuração inicial para satisfazer as exigências do usuário. O método através do qual essa reconfiguração é realizada no dispositivo pode variar.

Funções que eram realizadas por hardware passam a ser desenvolvidas por software em processadores de sinais. O fato de essas funções serem executadas em software com base em hardware programável implica que o equipamento pode ser programado para transmitir e receber uma ampla gama de frequências. Entretanto, tais possibilidades geram problemas no que concerne à regulamentação. Como em um sistema RDS há um controle de software, é possível fazer um dispositivo de hardware funcionar ilegalmente, ou seja, executar operações que não foram determinadas para ele, em espectros que ultrapassam seus limites pré-definidos.

Os dispositivos de hardware para os sistemas de RDS trabalham com a premissa de que é primordial uma alta capacidade de processamento digital de sinais com baixo consumo de energia, tudo isso aliado a características de reconfigurabilidade aceitáveis. A indústria tem caminhado de forma a contribuir para isso. Os dispositivos *Digital Signal Processing* (DSP) e *Field-Programmable Gate Arrays* (FPGA) são dispositivos capazes de atender, parcialmente, os conceitos almejados pelos sistemas de RDS. Equacionar e estabilizar um ambiente operacional de software é algo ainda complexo.

Vale a pena ressaltar também algumas vantagens adicionais de um RDS:

- inventário das peças reduzido;
- uso de componentes de rádio frequência baratos;
- baixo custo de desenvolvimento;
- melhor suporte para usuário com necessidades especiais.

RDS é uma forma de desenvolvimento de sistemas de comunicação sem fio. Devido à sua flexibilidade realçada e o desenvolvimento rápido, implementação orientada ao software é a opção lógica quando desenvolvemos tecnologia sem fio, contanto que os projetos e os confinamentos operacionais possam ser encontrados.

Há diversas maneiras de implementar um RDS. Idealmente, os desenvolvedores querem um desenvolvimento otimizado que permita reconfigurabilidade realçada, flexibilidade e manutenção facilitada, enquanto se aumenta a reutilização da propriedade intelectual.

No sentido de aprimorar o desenvolvimento do RDS, é necessário dizer que esforços estão sendo feitos por vários governos, instituições de pesquisa e indústrias a fim de se estabelecer uma arquitetura padrão para o desenvolvimento de sistemas de RDS. Como uma das iniciativas mais promissoras destaca-se a arquitetura *Software Communications Architecture* (SCA) que será um dos objetos de estudo deste projeto.

Vale ressaltar também que os sistemas RDS já estão começando a alcançar potencial comercial. Quando o RDS se torna objetivo final, todo potencial de adaptabilidade pode criar possibilidades para novos tipos de serviços.

A arquitetura básica de um RDS ideal é bem simplória Fig. (2.1), onde FERF refere-se à *front-end* de rádio de frequência. Consiste em uma antena inteligente, um conversor analógico-digital e de um processador digital de sinais. Um dos pontos mais importantes dessa arquitetura é que o sinal permaneça em sua forma analógica apenas quando necessário, ou seja, apenas quando for recebido através da antena. As demais etapas seriam realizadas em software a fim de que a reconfigurabilidade e todos seus benefícios possam ser bem aproveitados. Os elementos dessa arquitetura serão apresentados de forma melhor na próxima seção.

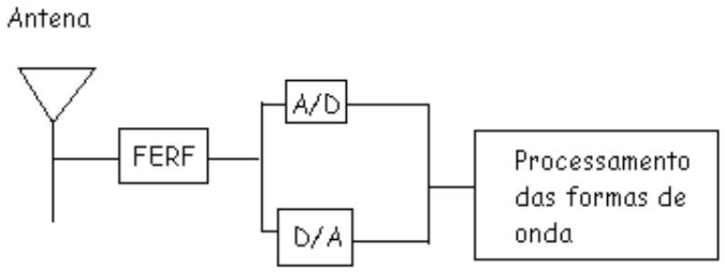

Figura 2.1: Arquitetura RDS

2.2 ELEMENTOS DE UM RDS

2.2.1 Antenas Inteligentes

No que se refere à reconfigurabilidade, os elementos da arquitetura RDS, embora fundamentais para seu funcionamento, podem ser fatores limitante à implementação. A própria antena do rádio já é um fator limitante à reconfiguração, quando se deseja uma alteração da frequência de transmissão e recepção do sinal. Nesse contexto que se investe em um a tecnologia em ascensão: a das antenas inteligentes.

O maior objetivo das antenas inteligentes é o de mudar seu tamanho de acordo com o comprimento de onda do sinal recebido, afinal de contas, sinais com frequências diferentes possuem comprimentos de onda diferentes; e também no que se refere à diretividade que está diretamente relacionado com o ganho da antena.

Podemos tornar uma antena inteligente através de várias modificações em sua estrutura. Primeiramente, sua estrutura física pode ser modificada pela adição de elementos. Depois, uma antena pode se tornar um sistema de antenas desenvolvido a fim de mudar o sinal da transmissão.

O conceito de se utilizar múltiplas antenas e o processamento de sinais inovador para servir células de forma mais inteligente já existe há muitos anos. De fato, várias aplicações de antenas inteligentes já foram usadas em sistemas de defesa há vários anos atrás.

O conceito de antenas inteligentes possui um erro em sua fala. Na verdade não existem antenas inteligentes, mas sim um sistema de antenas inteligente. Um sistema de antenas inteligente combina uma disposição da antena com a capacidade de transmitir e receber sinais de maneira adaptativa e com sensoreamento espacial. Ou seja, tal sistema pode mudar automaticamente a diretividade do seu padrão original em resposta à forma do sinal. Isso aumenta de forma incrível, o desempenho de um sistema móvel, por exemplo.

Uma antena simples trabalha para um ambiente de rádio-frequência simples. Antenas inteligentes são necessárias uma vez em que a quantidade de usuários, interferências e complexidade de propagação aumentam. Sua "inteligência"reside justamente na facilidade do processamento digital dos sinais.

Finalmente, vale a pena ressaltar os objetivos de um sistema de antenas inteligentes. Um dos objetivos é aumentar a qualidade do sinal dos sistemas de rádio-base através de uma transmissão mais focada de sinais de rádio enquanto a capacidade aumenta através do aumento reuso da frequência.

2.2.2 *FRONT-END* RF

Os sinais transmitidos e recebidos dos terminais do rádio podem percorrer o percurso da antena ao conversor A/D com a frequência bastante elevada ou podem percorrer o caminha inverso com frequência insuficiente. Para isso, faz-se uso concomitante ao dos conversores analógico/digital de um estágio de radio frequência denominado de *Front-end* RF.

Além das funções supracitadas, o *Front-End* Rf é responsável pela amplificação do sinal, filtragem anti-alias a fim de que o sinal possa ser digitalizado, seleção de canal, rejeição de interferência e controle da potência de transmissão.

Como se não bastasse o fato da antena ser fator limitante à reconfigurabilidade, o *front-end*, embora necessário, também o é. Seus filtros analógicos trabalham de forma ineficiente em uma vasta gama de frequência. O que se pode fazer é investir em hardware que opere em frequências distintas para atuar como *front-end* e em antenas "inteligentes"que variem seus tamanhos de acordo com o comprimento de onda e sua diretividade.

Embora um RDS ideal deva ter um *front-end* analógico mínimo, consistindo em um conversor analógico-digital na antena, qualquer implementação prática necessita de *front-end* RF e projetar um RF reconfig-

urável é algo bastante difícil.

O lado de transmissão do *front-end* RF captura o sinal do conversor digital-analógico, converte esse sinal para a transmissão de rádio frequência, amplifica o sinal para o valor desejado, limita a largura de banda do sinal filtrando-o a fim de evitar interferências e leva o sinal até a antena.

O lado receptor converte o sinal da antena a um centro abaixador de frequência até a nova frequência ficar compatível com o conversor analógico-digital, filtrando os ruídos, os canais indesejáveis e amplificando o sinal para um nível adequado para um conversor analógico-digital.

2.2.3 CONVERSORES ANALÓGICO-DIGITAL E DIGITAL-ANALÓGICO

Considerando o desempenho e o custo do Rádio Definido por Software, os conversores analógico digital e digital analógico estão entre os componentes mais importantes. São também os principais limitantes da arquitetura de RDS tanto devido ao custo quanto ao desempenho.

Em muitos casos, eles definem a largura de banda, a variação dinâmica e o consumo de potência do rádio. A largura de banda do conversor analógico-digital é um dos mais importantes desafios no desenvolvimento de RDS. Ela e a variação dinâmica do sinal analógico têm de ser compatíveis com o conversor analógico-digital.

2.2.4 PROCESSAMENTO DIGITAL

O processamento digital é a chave para qualquer RDS, isto é, o desenvolvimento de processamento digital programável torna possível reconfigurar qualquer interface aérea. O segmento do processamento digital de um RDS é funcionalmente similar a outros sistemas de comunicação digital.

A necessidade por reconfigurabilidade solicita o uso de hardware de processamento digital programável. A reconfiguração pode ser feita em vários níveis. Há muitos componentes paramétricos que são "fixos"(exemplo ASICs) e muitos *hardwares* que podem ser completamente reconfiguráveis (exemplo FPGA). Deve haver um compromisso entre programabilidade, tempo reconfigurável, potência de processamento, de consumo, custo etc.

Sinais digitais que são discretos no tempo, em comparação com sinais analógicos, possuem diversos benefícios. Dentre eles imunidade a ruídos, habilidade de gerar formas de onda arbitrárias, baixo "*switching time*"e circuitos com tamanhos menores.

É importante também discutir um pouco dos padrões relacionados com RDS. Eles são de grande importância considerando qualidade, confiabilidade, eficiência e compatibilidade. A indústria da Tecnologia da Informação (TI) não é uma exceção: padrões são essenciais do ponto de vista de compatibilidade, portabilidade de componentes de software e o desenvolvimento de produtos em geral.

Com frequência, a indústria da comunicação sem fio e os usuários finais têm de lidar com problemas crescentes da constante evolução dos padrões de interface aérea e diferentes padrões em países diferentes, incompatibilidade entre linhas sem fio e projetos mais antigos. RDS pode ser visto como uma tecnologia que traz a solução de muitos desses problemas. Em contrapartida, RDS devem adequar-se ao enorme número de padrões devido a operações multi-modo.

Algumas aplicações de Rádio Definido por software merecem ser destacadas, como a utilização em Estações Rádio-base.

2.3 APLICAÇÕES

O foco em RDS é principalmente aplicado no domínio comercial nas comunicações sem fio. As características de um Rádio Definido por Software são solicitadas por sistemas da 3^a. Geração da Telefonia Móvel assim como em sistemas de comunicações militares e civis governamentais. E a utilização do RDS em gerações futuras na área de telecomunicações o torna ainda mais desafiador visto que terá de promover uma ligação entre sistemas novos e antigos.

Deve ser ressaltado que toda capacidade da Segunda Geração (2G) deva ser "acomodada"nos terminais da Terceira Geração (3G) e também na infra-estrutura de transição e o impacto da arquitetura RDS como preferida e dominante não deve ser descartado. Como uma transição é significativamente influenciada pelo compromisso custo versus capacidade, o debate acerca de custo, complexidade, *performance*, tamanho, peso, consumo de potência nunca deve ser deixado de lado quando se enxerga um RDS como algo repleto de aplicações.

Pesquisas recentes indicam que o RDS é considerado uma tecnologia que desaloca implementações sem fio mais antigas e áreas de aplicação do governo, por exemplo. Isto é, as aplicações iniciais de RDS devem ser como pequenos "prêmios"compensados pelas vantagens de crescimento do RDS que cada implementação proporciona. À medida que o RDS se tornar mais popular, ele possivelmente se tornará a tecnologia dominante e preferida para muitas aplicações. Isso ocorrerá graças ao aumento de produtos que necessitam da tecnologia RDS e o aumento da quantidade de produtos que o aceitam.

Em aplicações militares é necessário que haja consolidação de aquisição de programas de rádio e o desejo de alcançar uma plataforma comum de comunicação através de serviços, ramificações ou conjuntos de operações táticas com diferentes objetivos. Como o RDS pode ser utilizado pelo governo, ele será escolhido com tecnologia dominante e sujeito a diversos níveis de financiamento.

2.4 ARQUITETURAS DE *SOFTWARE*

Há diversas maneiras de se implementar um Rádio Definido por Software. Idealmente, os desenvolvedores querem um desenvolvimento otimizado que permita uma reconfigurabilidade realçada, flexibilidade e manutenção facilitadas, concomitantemente ao incremento da reutilização da propriedade intelectual. Para obter os objetivos almejados, há diversas arquiteturas de software disponíveis. Algumas mais relevantes de disseminadas serão discutidas a seguir.

2.4.1 *Software Communication Architecture* (SCA)

A especificação SCA (*Software Communication Architecture*) procura atingir os objetivos destinados às arquiteturas de softwares que implementam RDS, principalmente através da premissa de criar uma independência entre *hardware* e *software*.

É uma arquitetura de sistema aberta, publicada pelo JTRS (*Joint Tactical Radio System*), um programa do Departamento de Defesa Americano (DoD) e pelo JPO (*Joint Program Office*). Esse programa foi criado com o intuito de fornecer ao departamento militar dos Estados Unidos uma família de rádios modulares digitais, programáveis, multi-banda e multi-modo a fim de aliviar os problemas de interoperabilidade de comunicação existentes entre todos os ramos militares.

Além disso, foi desenvolvido para aprimorar os sistemas de comunicação futuros, capturando os avanços recentes da tecnologia, realçando a interoperabilidade (supracitada) dos sistemas de comunicação e reduzindo os custos do desenvolvimento e da distribuição. Os principais objetivos do programa JTRS são:

- aumento da flexibilidade e interoperabilidade operacionais;
- redução de custos de suportabilidade;
- melhoria em termos da inserção de uma tecnologia mais fácil e com maior capacidade de aprimoramento;
- custo reduzido da aquisição e da operação do sistema.

A fim de atingir esses objetivos, a SCA foi estruturada para fornecer portabilidade de *software* para diferentes implementações; reduzir o tempo de desenvolvimento de novas formas de onda, reutilizando módulos de projetos anteriores; aumentar a quantidade de estruturas e arquiteturas comerciais e, por fim, reduzir os custos de desenvolvimento para esses padrões comerciais.

A SCA é desenvolvida para ir de ao encontro de aplicações comerciais e militares. Não se trata de uma especificação de sistema, pois pretende ser uma implementação independente, um conjunto de regras que confinam os projetos dos sistemas para conseguir os objetivos listados acima.

O próximo capítulo foi destinado a um estudo mais profundo dessa arquitetura.

2.4.2 *Wireless Information Transfer System* (WITS)

A WITS é uma arquitetura de rádio "obediente"desenvolvida pelo JTRS da Motorola baseada na arquitetura do Fórum SDR [2]. Ela foi desenvolvida pela Motorola a partir da grande experiência adquirida pelo envolvimento com sistemas de RDS através de programas como o SPEAKeasy, DMR e JTRS. Os sistemas baseados em WITS são utilizados pela Marinha dos Estados Unidos e a linha de produtos tende a se expandir para o mercado comercial.

A arquitetura do *software* é baseada na SCA, isto é, ela se apresenta em camadas e é uma arquitetura modular. A camada mais baixa está relacionada à abstração dos módulos de *hardware*. As entidades físicas são mapeadas em módulos de *hardware* definidos pela arquitetura, com exceção das antenas e dos amplificadores, que são dispositivos específicos e externos. A implementação de hardware é composta, em sua maioria, de *Line Replaceable Units* (LRUs) que são conectadas através de um conjunto de *Compact* PCI. A maior parte das unidades de processamento consiste de uma combinação de DSPs e ASICs. Os ASICs são comumente utilizados para comunicações com fio e processamento de rádio-frequência.

Cada módulo, incluindo os LRUs e os externos devem implementar o POSIX (*Portable Operanting System Interface*) APIs (*Application Programming Interfaces*), que são usados para interfaceamento com altos níveis da arquitetura [2]. O *software* de forma de onda existente não necessita de modificações quando um novo pedaço de *hardware* é adicionado uma vez que todos os elementos devem ser submissos ao POSIX, que é o sistema operacional do SCA e do WITS, por exemplo. As unidades de rádio frequência disponíveis, que utilizam *down-conversion*, não suportam altos índices de dados, mas para as apresentações militares presentes, o WITS é muito adequado e sua capacidade pode ser expandida.

3 *SOFTWARE COMMUNICATION ARCHITECTURE*

3.1 VISÃO GERAL

Há diversas maneiras de se implementar um Rádio Definido por *Software*. Idealmente, os desenvolvedores querem um desenvolvimento otimizado que permita uma reconfigurabilidade realçada, flexibilidade e manutenção facilitadas, concomitantemente ao incremento da reutilização da propriedade intelectual.

A especificação SCA (*Software Communication Architecture*) procura atingir estes objetivos, principalmente pela premissa de criar uma independência entre *hardware* e *software*. É uma arquitetura de sistema aberta, publicada pelo JTRS (*Joint Tactical Radio System*), um programa do Departamento de Defesa Americano (DoD) e pelo JPO (Joint Program Office). Esse programa foi criado com o intuito de fornecer ao departamento militar dos Estados Unidos uma família de rádios modulares digitais, programáveis, multi-banda e multi-modo a fim de aliviar os problemas de interoperabilidade de comunicação existentes entre todos os ramos militares.

Além disso, foi desenvolvido para aprimorar os sistemas de comunicação futuros, capturando os avanços recentes da tecnologia, realçando a interoperabilidade (supracitada) dos sistemas de comunicação e reduzindo os custos do desenvolvimento e da distribuição. Os principais objetivos do programa JTRS são:

- aumento da flexibilidade e interoperabilidade operacionais;
- redução de custos de suportabilidade;
- melhoria em termos da inserção de uma tecnologia mais fácil e com maior capacidade de aprimoramento;
- custo reduzido da aquisição e da operação do sistema.

A fim de atingir esses objetivos, a SCA foi estruturada para fornecer portabilidade de *software* para diferentes implementações; reduzir o tempo de desenvolvimento de novas formas de onda, reutilizando módulos de projetos anteriores; aumentar a quantidade de estruturas e arquiteturas comerciais e, por fim, reduzir os custos de desenvolvimento para esses padrões comerciais.

A SCA é desenvolvida para ir de ao encontro de aplicações comerciais e militares. Não se trata de uma especificação de sistema, pois pretende ser uma implementação independente, um conjunto de regras que confinam os projetos dos sistemas para conseguir os objetivos listados acima.

A estrutura da arquitetura de *software* é mostrada na figura 3-1 a seguir. Os principais benefícios dessa arquitetura são que ela maximiza o uso de produtos e protocolos comerciais, isola as aplicações pertencentes e não pertencentes ao núcleo do SCA dos dispositivos de *hardware* através de inúmeras camadas de uma infra-estrutura de *software* aberta e comercial além de fornecer portabilidade, reusabilidade e escalabilidade.

Ainda na figura 3-1, podemos observar que, de azul escuro se encontram os Commercial off-the-shelf, mais conhecidos daqui para frente como COTS. Trata-se de produtos de *hardware* e *software* comercializados para o público em geral. Dentre esse COTS merece destaque o CORBA que será delatado mais adiante.

De verde pode ser observado o Core Framework (CF) que como o próprio nome "Core"já diz, é o núcleo, ou seja, a parte essencial na arquitetura. É o conjunto de interfaces abertas de aplicações e serviços

que fornece uma abstração ao *software* que se encontra abaixo das camadas e ao *hardware* das camadas projetado para operações de *software*. Mais adiante será explicado um pouco mais a respeito do CF.

De colorido se encontram as *Applications*. Elas consistem de uma ou mais *Resources*. A interface *Resource* é a responsável por fornecer uma *Application Programming Interfaces* (API) comum para o controle e configuração de uma componente de *software*. Uma API é uma interface de código fonte que um sistema operacional ou uma biblioteca fornece para suportar um pedido de serviços que devem ser feitos por programas de computador. Os desenvolvedores da aplicação podem estender essa definição criando interface de Resource especializada para a aplicação. No mínimo a extensão herdaria a interface Resource.

Devices são tipos de *Resources* usados para aplicações como um substituto do *software* para dispositivos reais de *hardware*. ModemDevice, I/ODevice e SecurityDevice são exemplos que implementam as interfaces *Device*.

Na próxima seção a arquitetura terá uma explicação mais detalhada de seu funcionamento.

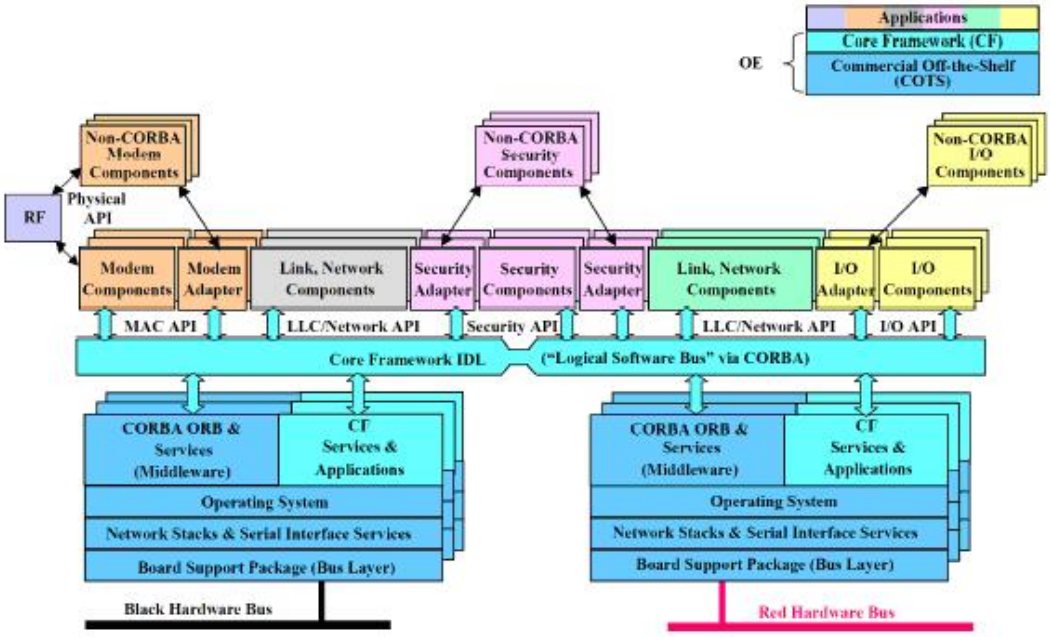

Figura 3.1: Estrutura do *Software* da Arquitetura SCA [3].

3.2 A ARQUITETURA DO *SOFTWARE*

E estrutura do *software* SCA divide o sistema em duas camadas:

1. Ambiente Operacional (AO), compreendido por:
    - Core Framework (CF),
    - CORBA Middleware,
    - Sistema Operacional POSIX,
    - Serviços.
2. *Applications*

3.2.1 O AMBIENTE OPERACIONAL

O Ambiente Operacional (AO), elemento chave da SCA, fornece os elementos necessários para que se rodem as aplicações. O objetivo do SCA de aprimorar o reuso de componentes e a tecnologia de inserção dos mesmos são possíveis graças à abstração da plataforma abaixo das camadas fornecida pelo AO. A partir da perspectiva de aplicação, o AO é sempre o mesmo, indiferentemente do quão diferentes sejam a camada física real e as plataformas lógicas. Essa plataforma abaixo das camadas fornecida pelo AO pode ser composta de um ou múltiplos processadores, de diferentes arquiteturas e características e distribuídos de diferentes placas, computadores, redes etc.

O AO também oferece mecanismos comuns de administrar e controlar aplicações e suas componentes. Nesse sentido, componentes de aplicação também possuem uma camada de abstração para suas implementações particulares. Ou seja, uma componente pode ser implementada usando dispositivos de *hardware* e *software* especializados, que podem ser escritos em diferentes linguagens de programação (C, C++ E Java), ou ser direcionados a rodar em uma plataforma ou outra.

O Ambiente Operacional é dividido em camadas, explicadas a seguir:

3.2.1.1 CORE FRAMEWORK

Antes de se explicar do que se trata o *Core Framework*, é interessante introduzir o conceito de *framework*. Um *framework* é um conjunto de classes que cooperam entre si e que maquia um projeto reutilizável para uma classe específica de *software*. Ele define a arquitetura e outras publicações sutis a fim de otimizar um certo domínio, permitindo que desenvolvedores se foquem em importantes aspectos das aplicações.

O *Core Framework* da SCA é o essencial da arquitetura de *software*. O "núcleo"que se ajusta das relações e dos perfis abertos e que fornece para *softwares* em sistemas embarcados a distribuição, a gerência, a interconexão, e a intercomunicação de seus componentes. É composto por um conjunto de interfaces abertas e serviços que fornecem desenvolvimentos de aplicações como uma forma comum de administrar as forma de onda e seus componentes.

O *Core Framework* consiste em:

- Base *Application Interfaces (Port, LifeCycle, TestableObject, PropertySet, PortSupplier, ResourceFactory e Resource)* que podem ser usadas por todas aplicações de *software*;
- Framework Control Interfaces (Application, ApplicationFactory, DomainManager, Device, LoadableDevice, ExecutableDevice, AggregateDevice e DeviceManager) que fornecem controle do sistema;
- *Framework Services Interfaces* que suporta aplicações que são e que não são do núcleo (*File, FileSystem, FileManager, e Timer*), e
- *Domain Profile* que descreve as propriedades dos dispositivos de *hardware (Device Profile)* e components de *software (Software Profile)* no sistema.

As interfaces do CF do SCA são definidas no CORBA Interface Description Language (IDL). Nessas interfaces IDL estão os métodos e atributos necessários para descrever uma componente. Interfaces descritas em IDL são programadas em linguagem independente e podem ser compiladas em diversas linguagens, incluindo, novamente as linguagens Java e C++.

3.2.1.2 CORBA MIDDLEWARE

A SCA foi desenvolvido na busca dos futuros sistemas de comunicação com a finalidade de aumentar a interoperabilidade de um sistema e reduzir custos de desenvolvimento e distribuição. Uma das bases da SCA é a utilização do CORBA como *middleware*. O CORBA é uma especificação desenvolvida pela OMG (*Object Management Group*). A OMG é uma organização internacional da indústria de computadores que aprova padrões abertos para aplicações orientadas ao objeto e que tem como objetivo dar suporte a publicações de problemáticas de interesses público ou privado com objetivos não comerciais.

A carta de princípios da OMG inclui o estabelecimento de diretrizes na indústria e especificações de gerenciamento de objetos para fornecer uma estrutura comum para desenvolvimento de aplicações. O objetivo primário é se alcançar sistemas baseados em objetos em ambientes distribuídos heterogêneos com características de reusabilidade, portabilidade e interoperabilidade.

O CORBA se baseia no uso do paradigma da Orientação a Objetos (OO). O conceito de OO pode ser usado para ambos: *software* e *hardware*. Essa prática oferece a mais ampla reusabilidade e portabilidade. É especialmente vantajoso para o RDS uma vez que a reconfigurabilidade torna possível a viabilidade de técnicas OO e independência da plataforma real utilizada necessária.

No contexto de redes de computador, o termo *middleware* é usado para denotar o centro das funções que viabilizam o uso de serviços de comunicação de forma fácil e distribuem serviços de aplicação. Em outras palavras, fornece meios de administrar aplicações ou serviços, mapear os nomes dos objetos que os fornecem, controlar conexão etc. Em comunicações móveis, o *middleware* pode funcionar para monitorar enlaces e notificar os usuários acerca de eventos significativos. Além disso, é parte essencial para o uso de serviços ininterruptos quando múltiplos padrões sem fio são usados.

CORBA é uma estrutura aberta, independente de fornecedor que oferece interfaces programáveis de plataformas independentes e modelos de aplicações computacionais distribuídos portáteis. É desejável para o desenvolvedor de novas aplicações e sua integração dentro do sistema, devido à independência de linguagens de programação, plataforma computacionais e protocolos de redes.

CORBA é usado no fornecimento de serviços de middleware que simplificam operações cliente-servidor padronizadas em um ambiente distribuído ao omitir os mecanismos de comunicação sob um barramento de *software* ORB (*Object Request Broker*). É usado pelo Core Framework como um mensageiro para o ambiente distribuído (fornece mensagens quando o processo ocorre de forma satisfatória). Ele é uma estrutura de plataforma cruzada que pode ser usado para padronizar operações entre o servidor e o cliente utilizando processo distribuído.

A arquitetura CORBA define o ORB (*Object Request Broker*), que é o barramento padrão para estabelecer e simplificar a troca de dados entre sistemas distribuídos heterogêneos. Um objeto CORBA é uma entidade virtual (ou seja, não existe se não for concretizado pela sua implementação escrita em uma linguagem de programação) capaz de ser localizada por um ORB. É uma infra-estrutura aberta e independente de fabricante que fornece interfaces e modelos independentes de plataforma para aplicações computacionais distribuídas e portáteis. Além disso, o ORB é responsável por todos os mecanismos requeridos para encontrar o objeto, preparar a implementação de objeto para receber a requisição e executá-la.

CORBA atua de forma que os compontentes do *software* possam se comunicar de forma transparente, mesmo tendo que interoperar com outros *softwares*, em outro sistema operacional e em outra ferramenta de desenvolvimento. O aspecto relevante de CORBA para RDS é que funciona como uma camada de *middleware* responsável pela independência das funções de *software* em relação ao *hardware* utilizado. A SCA e, conseqüentemente, CORBA foram adotados como padrão pelo Fórum de Radio Definido por Software.

A carta de princípios da OMG inclui o estabelecimento de diretrizes na indústria e especificações de gerenciamento de objetos para fornecer uma estrutura comum para desenvolvimento de aplicações. O objetivo primário é se alcançar sistemas baseados em objetos em ambientes distribuídos heterogêneos com

características de reusabilidade, portabilidade e interoperabilidade.

Todas as interfaces do *Core Framework* são definidas em IDL (*Interface Definition Language*). O CORBA também está definido em IDL. Essa é uma linguagem baseada em C++ que não possui algoritmos nem variáveis, ou seja, é puramente declarativa, e, portanto, é independente da linguagem de programação utilizada para acessá-la. Ao IDL SCA definem as operações e os atributos que servem como um contrato entre componentes.

Para compreender melhor a arquitetura CORBA é necessário conhecer a descrição do modelo de objetos [OMG 95] que fornece conceitos e terminologias de objetos usados pela arquitetura CORBA. Dentre os ORBs definidos pela arquitetura CORBA, o mais robusto e de desempenho mais elevado é o OMNIORB. É usado para linguagens C++ e Python e está livremente disponível sob os termos do GNU. Dentro do *middleware* é importante que se referencie também o XML (Extensible Markup Language). Como o próprio nome diz, trata-se de uma "*markup language*", ou seja, um tipo de linguagem que descreve o formato de um documento.

Foi projetada especificamente para entrega de informação através da "World Wide Web". Sua definição consiste a uma sintaxe reduzida ao essencial. Criando-se um documento XML, novos elementos podem ser criados e quaisquer nomes podem ser designados a eles. Entretanto, XML pode ser usado para descrever virtualmente qualquer tipo de documento, de uma música a um RDS.

3.2.1.3 SISTEMA OPERACIONAL - POSIX

A fim de aprimorar a portabilidade das aplicações, a arquitetura SCA necessita de um padrão Portable Operanting System Interface (POSIX). POSIX é o nome comercial dado pelo IEEE (*Institute of Electrical and Eletronics Engineers*) a esse padrão aceito pela indústria. Ele e suas extensões em tempo real são compatíveis com o que é necessário para suportar a especificação do OMG CORBA. A completa obediência ao POSIX engloba mais características que as necessárias para controlar uma implementação típica.

3.2.2 APLICAÇÕES

As aplicações, *Applications*, são os programas que executam as funções de um produto específico do SCA. Elas devem satisfazer as exigências de uma especificação e não estão definidas pelo SCA exceto quando se conectaram ao AO.

3.2.2.1 EXIGÊNCIAS GERAIS DA APLICAÇÃO

- **Serviços SO (Sistema Operacional)**: As aplicações serão limitadas a usar os serviços do SO.
- **Serviços do CORBA**: As aplicações serão limitadas a usar CORBA e serviços do CORBA.
- **Interfaces CF**: As aplicações implementarão as interfaces CF usando seu IDL correspondente.

3.2.2.2 INTERFACES DE APLICAÇÃO

Aplicações consistem em um de diversos componentes. Estes Componentes podem ser um CORBA-habilitado ou um CORBA-não habilitado. Para componentes de CORBA-habilitado, além de suportar as interfaces de aplicação do CF, os componentes podem executar e usar as interfaces específicas do componente para dados e/ou controle. As interfaces fornecidas por um componente serão descritas no arquivo Software Component Descriptor fornecido pela porta. Interfaces utilizadas por um componente serão descritas no arquivo Software Component Descriptor assim como as que são usadas pela porta. Uma Aplicação pode ainda ter interfaces externas opcionais além das interfaces de aplicação.

3.2.2.3 DISPOSITIVOS LÓGICOS

Um Dispositivo lógico é um *software-proxy* para um Dispositivo de *hardware*. Cada dispositivo de *hardware* usado por um componente da aplicação da Resource terá uma interface de Dispositivo lógica associada. As interfaces de dispositivo lógico incluem o Dispositivo (Devices), *o LoadableDevice*, o *ExecutableDevice*, e o *AggregateDevice*.

- **Serviços SO**: Os Dispositivos lógicos não serão limitados a usar os serviços designados como principais no SCA.
- Serviços do CORBA: Os Dispositivos lógicos serão limitadas a usar CORBA e serviços do CORBA.
- **Interfaces CF**: Os Dispositivos lógicos implementam uma das seguintes interfaces CF: *Device*, *LoadableDevice* ou *ExecutableDevice*.
- **Perfil**: Cada Dispositivos lógicos terá um SPD, SCD e DPD e um ou mais *Properties Descriptor*, citado anteriormente.

3.2.2.4 REGRAS GERAIS DO *SOFTWARE*

Esta seção identificará regras e recomendações específicas para a arquitetura do software que ainda não foram especificamente descritas em outras seções.

- Novo Software: O *software* desenvolvido para um produto aderente a SCA será projetado em uma linguagem padrão de uma ordem mais elevada. O objetivo do novo desenvolvimento será fornecer um SW que é independente dos detalhes da plataforma e do ambiente, assegurando edições mínimas de portabilidade.
- *Software* Herdado (*Legacy Software*): não necessita ser reescrito em uma linguagem padrão de uma ordem mais elevada. O *Legacy Software* será conectado à estrutura do núcleo de acordo com sua especificação.

3.3 ESTRUTURA DE *HARDWARE*

Mesmo que um dos maiores objetivos da concepção do RDS seja executar a maior quantidade de funções possível no domínio da programação digital, isto é, em *software*, os padrões de *hardware* ainda exercem um papel fundamental sob o ponto de vista da modulação. Idealmente, diferentes fornecedores deveriam ser capazes de desenvolver módulos de rádio usando interface padrão.

A estrutura do *hardware* também utiliza o conceito do OO (orientado ao objeto) para definir uma típica partição de *hardware* dentro de sistemas realizáveis. A principal finalidade da estrutura de *hardware* é a publicação completa e detalhada das relações e dos atributos, uma vez que os sistemas foram construídos. A modularidade do *hardware* facilita a inserção da tecnologia enquanto os elementos programáveis futuros aumentam a potencialidade, ou seja, quanto mais modular o *hardware* se tornar, mais fácil será sua inserção bem-sucedida quando o assunto é Rádio Definido por *Software*.

A especificação SCA estabelece uma estrutura de implementação independente com exigências para o desenvolvimento do JTRS. Estas exigências são compreendidas de especificações de interface e comportamentais das APIs (*Application Program Interfaces*) e de regras. O objetivo desta especificação é assegurar a portabilidade e a configurabilidade do *software* e do *hardware* e assegurar a interoperabilidade dos produtos desenvolvidos usando o SCA.

É uma "estrutura de arquitetura"que é necessária nas áreas em que algo é reutilizado (RDS). O SCA define o *hardware* e o *software* em diferentes níveis de detalhes permitindo a maior portabilidade e reusabilidade de seus componentes.

Para o *hardware*, a parte física e as diferenças do produto através dos domínios são tão diversas que aspectos físicos ordinários não podem ser obtidos para todas implementações. Entretanto, utilizando a descrição para *hardware* Orientada ao Objeto, representada como uma classe de *hardware*, todas as implementações potenciais do sistema são incluídas em uma simples estrutura.

A definição de *hardware* se limita no nível da estrutura, com regras que fornecem a orientação da execução de domínios e plataformas. Já a definição de *software* pode ser aplicada diretamente à implementação por causa da sua independência geral da implementação do *hardware*.

As interfaces, comportamentos e regras que definem a conformidade com a SCA são identificadas e são partes das suas especificações. Esses elementos são selecionados para aumentar a portabilidade, interoperabilidade e configurabilidade de um *software* e de um *hardware* ao permitir que a flexibilidade se dirija às exigências e limitações do domínio.

A SCA foi desenvolvida para sistemas de rádio em tempo real, embarcados a fim de suportar o *Joint Technical Architecture* (JTA) quer que ele seja aplicado. O JTA é o responsável por validar os padrões para todo comando do DoD.

3.4 ARQUITETURA DE *SOFTWARE*

Os principais benefícios da arquitetura de software são:

- aumentar o uso comercial de protocolos e produtos;
- isolar as aplicações do núcleo do hardware com as múltiplas camadas "aberta"(públicas) do software;
- fornecer portabilidade, reusabilidade e escalabilidade a um ambiente de processo distribuído a partir do uso do CORBA.

3.4.1 CAMADA DE OPERAÇÃO DO SISTEMA

A Arquitetura de Software inclui funções de sistemas embarcados operando em tempo real para fornecer sustentação às aplicações (incluindo aplicações no *Core Framework*). A arquitetura requer uma relação de sistema operando no modo padrão para serviços de sistema a fim de facilitar a portabilidade das operações.

3.4.2 CAMADA DE APLICAÇÃO

3.4.2.1 APLICAÇÕES

Aplicações consistem em uma ou mais *Resources*. A *interface Resource* fornece uma API comum para o controle e a configuração de um componente de *software*. Os desenvolvedores do aplicativo podem estender essas definições criando interfaces do *Resource* especializadas para a aplicação.

Aplicações do núcleo, que são parte do *Core Framework*, suportam as aplicações que não estão no núcleo, fornecendo a função de controle, assim como as definições de relação de padrão.

3.4.2.2 ADAPTADORES

Adaptadores são *Resources* ou dispositivos usados para suportar o uso de elementos no domínio fora do CORBA. Eles são usados em uma implementação para fornecer uma translação entre os componentes que não utilizam o CORBA e as *Resources* que o utilizam.

Os adaptadores se tornaram particularmente usuais para suportar elementos tais como Modem, elementos de Segurança e *Hosts* que não rodam sobre CORBA. É importante observar que o link de forma de onda e as Resources de rede não são afetados pela inclusão ou não dos Adaptadores.

3.4.2.3 CONCEITOS FUNCIONAIS DE RÁDIO DEFINIDO POR *SOFTWARE*

MODELO DE REFERÊNCIA DO *SOFTWARE*: o modelo de referência do software é descrito baseado no Modelo de Referência do Sistema de Comunicação Modular Programável (PMCS). Esse modelo forma a base para a SCA através de:

1. Introdução dos vários papéis funcionais que as entidades do software podem executar, sem ditar um modelo estrutural destes elementos;

2. Introdução do controle e da interface do tráfego de dados entre as entidades funcionais do software.

O Modelo de Referência identifica funcionalidades relevantes, mas não impõe uma arquitetura. SCA realiza o Modelo de Referência do *Software* definindo um padrão único de funcionalidades denominado *Resource*. Todas as aplicações são compreendidas na Resource e utilizando Dispositivo (Device). *Resources* e Dispositivos específicos podem ser identificados através de suas entidades funcionais correspondentes no Modelo de Referência do *Software*.

Funcionalidade do dispositivo de Modem:o dispositivo de Modem fornece um padrão para o controle e a interface de um modem, retendo diversas execuções da antena inteligente, do RF e das funções do modem, por exemplo. As funções realizadas pelo dispositivo de Modem vão variar dependendo da forma de onda almejada assim como a alocação de hardware e software e não são ordenadas pelo *Core Framework*.

3.4.3 SISTEMA DE CONTROLE

SCA fornece especificações para interfaces, serviços e formatação de dados para o controle dos recursos. Cada recurso estabiliza seu próprio parâmetro de controle com o *Domain Manager* através do domínio do perfil.

O uso do CORBA nos dá um sentido para se ter o domínio e o controle da aplicação, embora ela esteja em uma interface comum (pública). *Serial Device* e *Ethernet Device* são exemplos de interfaces externas disponíveis para o uso. Esses exemplos mostram que sistemas de controle operacionais operam com humanos ou com quaisquer interfaces de máquinas localmente ou remotamente e interagem de maneira a facilitar a portabilidade.

3.4.4 PROTOCOLOS DE REDES EXTERNAS

Protocolos de redes externas definem a comunicação entre um sistema de rádio aderente a SCA e seus sistemas de pares.

Através de protocolos externos de rede, implementados pelas aplicações de um sistema de rádio aderente a SCA e seus pares, uma rede de nós é formada interconectando repetidores, bridges, gateways,

roteadores etc. Protocolos de redes externas vão se interconectar tipicamente em diversas camadas utilizando:

1. interconexões físicas de camadas com função de repetição;
2. interconexão de camadas de ligação com a função bridge;
3. interconexão das camadas de rede através de roteamento;
4. interconexão das camadas superiores através do uso de gateways.

As diferentes categorias de interoperabilidade são esboçadas abaixo baseadas no modelo OSI. Pode haver múltiplos níveis de interoperabilidade dentro do mesmo sistema.

1. Interoperabilidade da camada física. Os protocolos da rede externa fornecem uma interface com compatibilidade física, incluindo a interface de sinalização, mas não incluindo as camadas mais altas. Esse nível de interoperabilidade é adequado para uma simples disposição de bit sobre bit ou para operações entre duas interfaces.
2. Interoperabilidade da camada de ligação. Os protocolos da rede externa fornecem ligação entre as camadas atuando através de todas as interfaces físicas. Esse nível de interoperabilidade é adequado para permitir que o rádio seja utilizado como um meio de transporte e para permitir que esse mesmo rádio seja utilizado como transporte também em outras redes. As decisões inteligentes de roteamento ou do switching são limitadas ao roteamento local da camada 2.
3. Interoperabilidade da camada de rede. Os protocolos da rede externa fornecem interoperabilidade no processo de endereçamento da camada de rede. O rádio e as redes interoperando são sub-redes de uma mesma rede que os relaciona. Nesse nível, as decisões inteligentes do switching e do roteamento podem ser feitas fim-a-fim.
4. Interoperabilidade na camada HOST (camadas 4 a 7). Aplicações embarcadas podem trocar informações com hosts unidos à rede. Um exemplo disso é um rádio de mão que contém "Consciência da Situação"embarcada. Neste exemplo, o rádio fornece traduções da mensagem para permitir que dois hosts, que de outra maneira seriam incompatíveis, comuniquem-se.

Tipicamente, cada forma de onda ou protocolo de rede vai ser implementado por um único conjunto de um ou mais entidades de protocolos. Um único conjunto de entidades de protocolos implementa a pilha específica de protocolos por uma forma de onda ou protocolo de rede. Um sistema de rádio implementando múltiplas aplicações de formas de onda pode ter entidades de protocolos múltiplos em cada camada de protocolo.

CORBA é o mecanismo preferido de transferência. Usando serviços associados com CORBA, as exigências da segurança para uma execução particular podem ser encontradas. Uma introdução atrasada de um mecanismo de transferência diferente requer uma análise cuidadosa dos serviços de segurança que podem ser fornecidos por aquele mecanismo de transferência.

3.5 ARQUITETURA DE *HARDWARE*

Particionando-se o *hardware* em classes, enfatizando os elementos físicos do sistema e como eles são compostos de elementos funcionais. Essas classes definem elementos comuns que dividem atributos físicos (características e interfaces) que carregam excesso à execução para plataformas de domínios específicos. Essa mesma estrutura é aplicada a todos os domínios.

Os atributos são os parâmetros que definem os dispositivos de hardware de domínio neutro e os valores designados a eles satisfazem as exigências para uma execução selecionada. Os dispositivos de hardware, que são a implementação física dessas classes, terão valor para atributos relevantes, baseados nas exigências da existência de uma plataforma física e na obtenção de desempenho.

3.5.1 AMBIENTE DE OPERAÇÃO

3.5.1.1 AMBIENTE DE OPERAÇÃO

O ambiente de processamento e as funções executadas na arquitetura impõem a esta diferentes restrições. O perfil do ambiente de operação do SCA é definido a fim de suportar a portabilidade das formas de ondas, a escalabilidade da arquitetura e a viabilidade comercial. O CORBA *Object Request Broker* (ORB), o CF *Framework Control InterfacE*, *Framework Services Interface* e os dispositivos de *Hardware* não são limitados a usar serviços designados como principais pelo perfil.

3.5.2 SERVIÇOS E *MIDDLEWARE*

Serviços e *Middleware* são subdivididos conforme mostrado na Fig. 3.2.

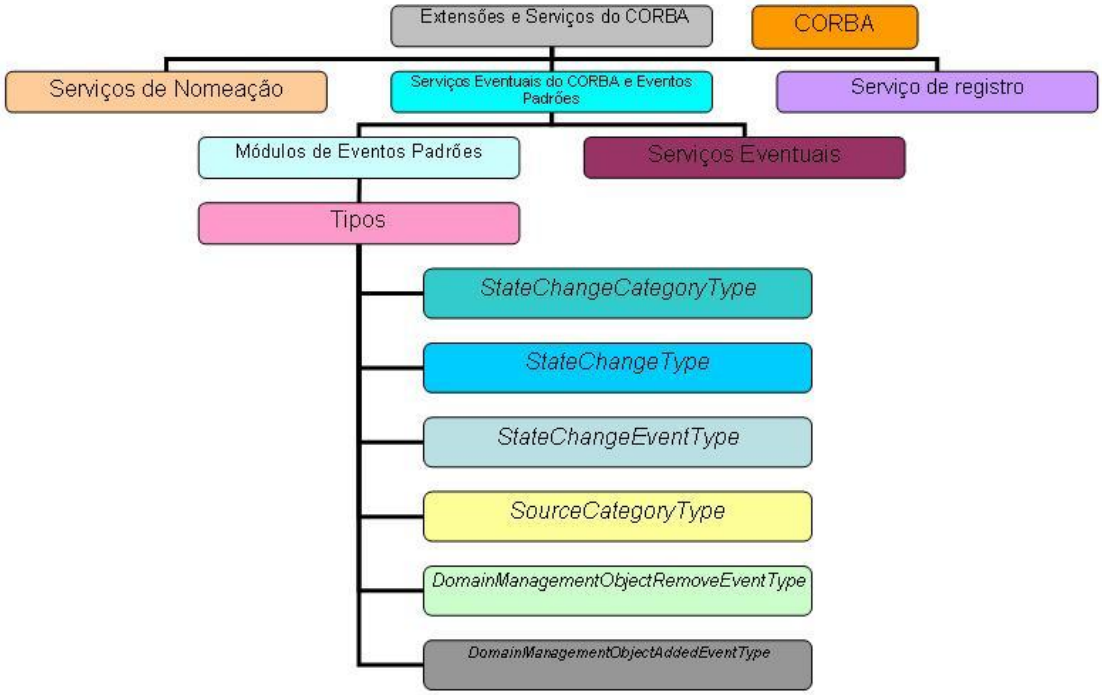

Figura 3.2: Subdivisão dos serviços de *Middleware*.

Nota-se presenta na Fig. 3.2 divisões e subdivisões. Está presente o CORBA, que o AO usará o *middleware* que, no mínimo, fornece os serviços e as potencialidades de minimumCORBA; a **Extensão e Serviços do CORBA**; os **Serviços de Nomeação**; os **Serviço de Registro**; os **Serviços Eventuais do CORBA e Eventos Padrões**; e os **Serviços Eventuais do CORBA**.

Serviços Eventuais do CORBA:

Um serviço eventual do CORBA (por exemplo, OMG) será fornecido no OE. O serviço eventual de CORBA permite a comunicação entre os objetos do consumidor e do fornecedor, em que os componentes

do consumidor são independentes dos componentes do fornecedor, e vice- versa. O serviço eventual do CORBA é baseado na aproximação do *Push Model*, onde os produtores promovem eventos para o consumidor. Este Serviço tem potencial para criar os canais do evento, que permitirão que os múltiplos fornecedores se comuniquem com consumidores simultaneamente. O canal do Evento é ao mesmo tempo fornecedor e consumidor de um evento.

Módulos de Eventos Padrões:

O módulo de Eventos Padrões (*StandardEvent*) contém os tipos definidos que serão usados para passar eventos dos produtores do evento aos consumidores do evento.

Tipos:

- ***StateChangeCategoryType***: É uma relação utilizada no *StateChangeEventType*. É usado para identificar a categoria de uma mudança do estado que ocorreu.
- ***StateChangeType***: É uma relação utilizada no *StateChangeEventType*. É usado para identificar os estados específicos da origem do evento antes e depois que a mudança do estado ocorreu.
- ***StateChangeEventType***: É a estrutura usada para indicar que o estado de um evento mudou. O produtor do evento emitirá esta estrutura em um canal do evento em nome da fonte do evento.
- ***SourceCategoryType***: Relação utilizada no *DomainManagementObjectAddedEventType* e no *DomainManagementObjectRemovedEventType*. É usado para identificar o tipo de objeto que foi adicionado ou removido do domínio.
- ***DomainManagementObjectRemovedEventType***: Estrutura utilizada para identificar que a origem de um evento foi removida do domínio. O produtor do evento emitirá esta estrutura em um canal do evento em nome da fonte do evento.
- ***DomainManagementObjectAddedEventType***: Estrutura utilizada para identificar que a origem de um evento foi adicionada no domínio. O produtor do evento emitirá esta estrutura em um canal do evento em nome da fonte do evento.

3.5.2.1 *CORE FRAMEWORK* - ESTRUTURA DO NÚCLEO

A especificação do CF inclui uma descrição detalhada da finalidade de cada interface. A Figura 3.3 descreve os elementos chaves do CF e a relação IDL entre estes componentes.

Um componente do *DomainManager* controla as aplicações do *software*, o *ApplicationFactories*, dispositivos do *hardware* (representados por dispositivos do *software*) e o *DeviceManagers* dentro do sistema. Aplicação é um tipo de *Resource* das várias *Resources* de *software* existentes. Algumas podem controlar diretamente os dispositivos internos de *hardware* do sistema. Estas *Resources* são Dispositivos lógicos que implementam as interfaces: *Device, LoadableDevice, ou ExecutableDevice*. Por exemplo, o *NetworkResource* pode executar uma função da camada de rede. Cada *Resource* pode se comunicar com outros *Resources* e Dispositivos e podem ser distribuídos a um ou mais dispositivo de *hardware* pelo *DomainManager* baseado em vários fatores, incluindo os dispositivos de *hardware* que o *DeviceManager* conhece, a disponibilidade destes dispositivos, o comportamento de um *Resource* e as exigências do carregamento da *Respource*.

As *Resources* que estão sendo controlados pelo *DomainManager* são objetos de CORBA que implementam a interface *Resource*. Alguns podem ser dependentes de outros e esta interface fornece uma maneira consistente de adicionar ou remover qualquer uma dentro do sistema.

Interface Base das Aplicações

São definidas pelas exigências da estrutura do núcleo (CF) e executadas por seus desenvolvedores.

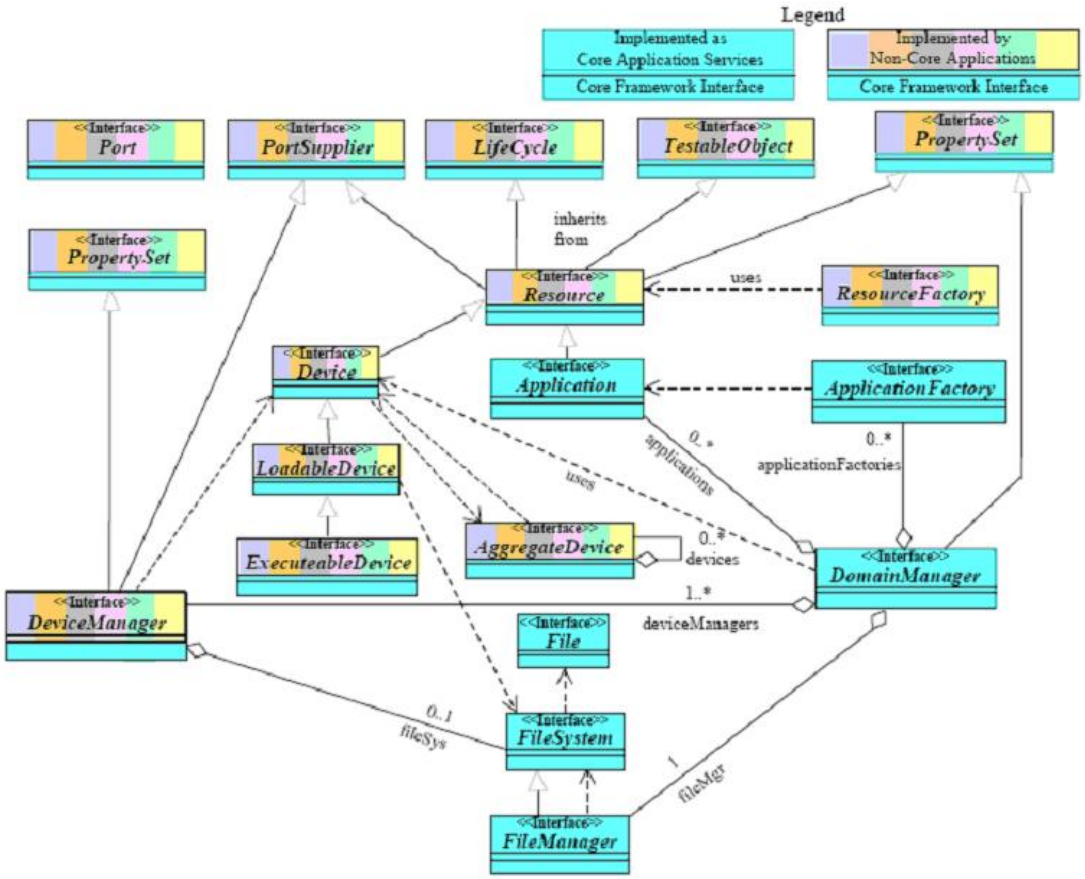

Figura 3.3: Elementos chave do CF e a relação IDL [3].

- *Port*: Esta interface fornece operações para controlar ligações entre as portas. É importante comentar que a especificação do CORBA define somente o mínimo de cada tipo básico de IDL. O tamanho real dos tipos de dados depende da linguagem e da arquitetura usada na CPU. Usando estes tipos de dados básicos de CORBA, a portabilidade é mantida entre os componentes implementados em diferentes arquiteturas de CPU e as linguagens.
- *LifeCycle*: Esta interface define as operações genéricas para inicializar ou liberar instantaneamente dados de componentes específicos e/ou processar elementos.
- *TestableObject*: Interface que define um conjunto de operações que podem ser usadas para testar os componentes implementados.
- *PortSupplier*: Interface que fornece a operação GetPort para os componentes fornecedores de portas.
- *PropertySet*: interface que define as operações configure e query para acessar o componente properties/attributes.
- *Resource*: A interface Resource fornece um API comum para o controle e a configuração de um componente de software.
- *ResourceFactory*: Interface usada para criar ou retirar uma Resource. O mecanismo Factory fornece a separação entre os recursos cliente-usuário (por exemplo, rede, link, modem, I/O, etc...) e fornece um mecanismo padrão para obter uma Resource sem saber sua identidade.

Não é preciso requerer uma Aplicação para usar a interface *ResourceFactory* para se obter, criar ou remover *Resources*. Um *software* determinará que aplicações do *ResourceFactories* será usadas pela interface *ApplicationFactory*.

Estrutura de Controle das Interfaces

O controle da estrutura dentro de um domínio é realizado pelas seguintes interfaces: *DomainManagement, devices e DeviceManagement*. As interfaces do *Domain Management* são *Application, ApplicationFactory, e DomainManager*. Estas interfaces gerenciam o registro das Aplicações, Dispositivos e Dispositivos de Gerenciamento dentro do domínio assim como controlam suas aplicações no domínio.

- *Applications* (Aplicações): Esta classe fornece a interface para o controle, configuração e status (em um determinado momento) de uma aplicação no domínio. A interface Aplicações herda a interface IDL da *Resource*. Um exemplo criado da aplicação pode conter componentes da *Resource* e/ou componentes de non-CORBA.
- *ApplicationFactory*: Classe que fornece uma interface para pedir a criação de um tipo específico de Aplicação no domínio. É projetada usando o *Factory Design Pattern* e o perfil do software determina o tipo de aplicação que é criada pelo *ApplicationFactory*.
- *DomainManagement*: É a interface de controle e configuração do domínio do sistema. Pode ser dividida em três categorias: *Human Computer Interface* (HCI), Registro e administração do CF.
- *Devices* (Dispositivos): Um dispositivo é um tipo de *Resource* dentro do domínio e tem as mesmas exigências da interface dela. Esta interface define capacidades e atributos adicionais para todo o dispositivo lógico no domínio. Um dispositivo lógico é parte de um conjunto de dispositivos de *hardware*.
- *LoadableDevice*: Esta interface estende a interface Devices adicionando a este um software de procedimento de carregamento descarregando.
- *ExecutableDevice*: Esta interface estende a interface de LoadableDevice adicionando um procedimento de execução e término ao Dispositivo.
- *AggregateDevice*: Permite o procedimento "agregar"que pode remover ou adicionar Dispositivos. Esta interface pode ser herdada ou providenciada por uma porta por qualquer Dispositivo capacitado para agregar.
- *DeviceManagement*: Interface usada para gerenciar um conjunto de serviços e Dispositivos lógicos.

Estrutura de Serviços da Interface

Pode ser implementada usando a IDL do CF.

- *File*: Interface com a habilidade de ler e escrever arquivos no CF. O File descreve, ainda, onde ocorrerá a próxima leitura ou escrita de arquivo.
- *FileManagement*: Através desta interface, múltiplos arquivos do sistemas podem ser acessados.

Perfil do Domínio

Os dispositivos de *hardware* e componentes de *software* que formam o domínio do sistema SCA são descritos como um conjunto de arquivos que são chamados de Perfil do Domínio. Estes arquivos descrevem a identidade, capacidade, propriedade, inter-dependência e a localização dos dispositivos de *hardware* e componentes de software que formam o sistema. Estes dados são mostrados pelo XML. Para finalidade

desta especificação de SCA, os elementos do vocabulário de XML foram baseados na especificação dos componentes de CORBA do OMG. O vocabulário de XML dentro de cada um destes arquivos descreve um aspecto distinto dos recursos do *hardware* e do *software*.

- *Software Package Descriptor*: Identifica a implementação de um componente de software. Em geral, fornece dados sobre os pacotes do *software*, como nomes, autor e proprietário do arquivo, etc.
- *Software Component Descriptor*: Contém informações sobre um componente específico de software SCA. Apresenta também informações sobre os componentes usados ou fornecidos por uma interface.
- *Software Assembly Descriptor*: Contém informação sobre os componentes que formam uma aplicação. O *ApplicationFactory* utiliza essa informação quando cria uma aplicação.
- *Properties Descriptor*: contém informações sobre as propriedades aplicáveis a um pacote de software ou a um pacote de dispositivo. Informa também os tipos de configurações, testes, execução e alocação de um componente.
- *Device Package Descriptor*: Identifica a classe do Dispositivo.
- *Device Configuration Descriptor*: Contém informações de como achar o Domain Device e fornece informações sobre configurações ao Dispositivo.
- *Profile Descriptor*: Fornece o nome completo do arquivo para qualquer *Software Package Descriptor, Software Assembly Descriptor* ou *Device Configuration Descriptor*.
- *DomainManagement Configuration Descriptor*: Contém informações de configuração para o *DomainManagement*.

Tipos Bases da Estrutura do Núcleo

Estes tipos são modelos usados na interface do CF.

- *DataType*: Este tipo é uma estrutura modelo do CORBA IDL que pode ser usada para manter qualquer tipo básico de CORBA.
- *DeviceSequence*: Define uma Seqüência ilimitada de Dispositivos.
- *FileException*: Relata a ocorrência de erros.
- *InvalidFileName*: Indica que um arquivo com nome inválido foi aceito no serviço de operação do arquivo.
- *InvalidObjectReference*: indica uma referência de objeto CORBA inválida.
- *InvalidProfile*: indica um perfil inválido.
- *OctetSequence*: É um tipo de seqüência ilimitada do CORBA.
- *Properties*: usado para definir seqüência de nomes e pares de valores.
- *StringSequence*: Define seqüência de variáveis.
- *UnknownProperties*: indica as propriedades desconhecidas pelo componente.
- *DeviceAssignmentType*: define uma estrutura que associa um componente ao Dispositivo onde este deve ser executado.
- *DeviceAssignmentSequence*: Provê uma seqüência ilimitada de D*eviceAssignmentType* no CF.
- *ErrorNumberType*: mostra tipos de erro.

4 SISTEMAS EMBARCADOS

4.1 APRESENTAÇÃO

Sistemas embarcados são sistemas computacionais dedicados a executar um conjunto específico de tarefas de maneira contínua e, na maioria das vezes, sem travamentos e panes. Estas plataformas são normalmente simples, constituídas por microprocessadores de baixa velocidade com recursos limitados a atender estritamente as aplicações para as quais foram projetadas.

Tais sistemas devem atender a uma grande variedade de características comuns - incluindo resposta em tempo real - e de requisitos, tais como essencialidade e eficiência. A tecnologia de Sistemas Embarcados é essencial à obtenção de informações seguras, uma das metas da moderna tecnologia da informação.

Seguindo o sucesso da tecnologia da informação para escritório e aplicações de fluxo de trabalho, os sistemas embarcados são considerados como a mais importante aplicação na área de tecnologia da informação durante os próximos anos. Devido a essa expectativa, o termo era pós-PC foi criado. Esse termo denota ao fato de que, no futuro, o padrão do PC como é hoje será o tipo de hardware menos dominante. Processadores e softwares serão usados em sistemas muito menores e, em muitos casos, poderão ser invisíveis. Eis a razão do termo "o desaparecimento do computador".

Atualmente, o mercado dos chamados dispositivos embarcados abrange uma ampla variedade de consumidores e produtos, sendo um dos ramos promissores dentro da área de tecnologia uma vez que um simples FPGA pode ser programado para desempenhar praticamente qualquer função.

4.2 *HARDWARE* DE SISTEMAS EMBARCADOS

Uma das características mais importantes dos sistemas embarcados é que, no projeto, deve-se levar em consideração tanto *hardware* quanto *software*. O reuso de componentes de *hardware* e de *software* é um dos principais diferenciais dessa metodologia construída baseada na plataforma.

O *hardware* para sistemas embarcados é muito menos padronizado do que o *hardware* de PCs. Muitos sistemas de controle apresentam uma malha de realimentação. Nestes sitemas, as informações acerca do ambiente físico são extraídas através de sensores, que transformam sinais físicos em sinais elétricos, tipicamente contínuos e analógicos.

Hardware para sistemas embarcados é muito menos padronizado do que o *hardware* de PCs. Em muitos sistemas de controle, o hardware de sistemas embarcados é usado em *loops*. Nesses *loops*, informações acerca do desenvolvimento físico se tornam possíveis graças a sensores.

Tipicamente sensores geram seqüências contínuas de valores analógicos. Conversões apropriadas são feitas em dois tipos de circuitos: circuitos "*sample-and-hold*"e conversores analógico-digital (A/D). Após essa conversão, as informações podem ser processadas digitalmente. Resultados gerados podem ser indicados e usados no controle físico de atuadores. Uma vez que a maioria dos atuadores é analógica, a conversão de digital pra analógico também é necessária.

4.2.1 ENTRADAS

4.2.1.1 SENSORES

Sensores podem ser criados para quaisquer aparelhos físicos. Há sensores de peso, velocidade, aceleração, corrente elétrica, tensão, temperatura, pressão etc.

Recentemente, uma grande quantidade de sensores pôde ser criada e muito do progresso e do desenvolvimento de sistemas inteligentes se deve ao desenvolvimento da tecnologia de desses aparelhos. Como foi citado anteriormente, há dois tipos principais de sensores, são eles:

- circuitos "*Sample-and-hold*": Sabe-se que os computadores trabalham no domínio discreto do tempo. Isso quer dizer que se pode processar seqüências discretas de valores. Logo, valores no domínio contínuo devem ser convertidos para o domínio discreto. Esse é o propósito desse tipo de circuito.

  Essencialmente, esse circuito consiste de um transistor de *clock* e de um capacitor. O transistor opera como uma chave. Cada tempo da chave é fechado por um sinal de *clock*, nisso, o capacitor é carregado a fim de que sua tensão assuma um valor VE (tensão de entrada). Após abrir novamente a chave, a tensão vai permanecer praticamente a mesma até a chave ser fechada novamente. Cada um desses valores armazenados no capacitor pode ser considerado como elemento de uma seqüência discreta de valores Vx, gerada a partir de uma seqüência contínua VE.

  Um circuito "*sample-and-hold*"ideal deve ser capaz de mudar a tensão no capacitor em um curto e arbitrário período de tempo. Dessa forma, a tensão de entrada em um instante particular pode ser transferida ao capacitor e cada elemento da seqüência discreta deve corresponder à tensão de entrada em um determinado tempo.

  Na prática, entretanto, o transistor deve manter a chave fechada por um curto período de tempo a fim de realmente carregar e descarregar o capacitor. A tensão armazenada no capacitor corresponderá à tensão média através de uma pequena janela de tempo.

- conversores A/D: Uma vez que nosso sistema embarcado está restrito a computadores digitais, trabalha-se com valores discretos representando nossos sinais de entrada. A conversão de valores analógicos para valores digitais, é feita por esse tipo de conversor. Há uma grande escala de conversores A/D que variam características de velocidade e precisão.

4.2.1.2 COMUNICAÇÃO

A informação tem de estar disponível antes de ser processada em um sistema embarcado. Ela pode ser comunicada através de vários canais. Canais são entidades abstratas caracterizadas por propriedades essenciais de comunicação, como máxima capacidade de transferência de informação e parâmetros de ruído. A probabilidade de erros de comunicação pode ser computada usando técnicas teóricas de comunicação. As entidades físicas que permitem a comunicação são chamadas de meios de comunicação. Importantes classes de meios incluem: meios "*wireless*", fibra óptica e cabos.

Há uma grande variedade de exigências de comunicação entre as várias classes de sistemas embarcados. Em geral, conectar diferentes componentes de hardware embarcados está longe de ser uma tarefa trivial. Algumas exigências podem ser identificadas:

- comportamento no tempo real: a maioria das redes de computadores é baseada no protocolo Ethernet. Para versões do Ethernet de 10Mbits/s e 100Mbits/s, pode haver colisões entre vários parceiros de comunicação. Ou seja, vários parceiros estão tentando se comunicar ao mesmo tempo e os sinais dos fios estão corrompidos. Quando isso ocorre, os parceiros têm de interromper a comunicação, esperar algum tempo e, então, tentar novamente. O tempo de espera é escolhido aleaoriamente, então não

é muito comum que a próxima tentativa também resulte em colisão. Esse método é denominado detector de colisão (CD). Para o CD, o tempo de colisão pode se tornar grande se elas se repetirem diversas vezes, mesmo isso não ocorrendo repetidas vezes (como foi citado anteriormente).

- Exemplos
  - Comunicação "*wireless*": tipo de comunicação que está se tornando cada vez mais popular ao passo que a largura de banda está se tornando um recurso escasso.
  - Circuitos Integrados de Aplicações Específicas (ASICs): para aplicações de alta performance e em larga escala, circuitos integrados de aplicações específicas podem ser desenvolvidos. Entretanto, o custo do desenvolvimento e da manufatura de chips é um pouco alto. Entretanto, ASICs são apropriados apenas se o máximo de eficiência no uso de energia for necessário, se o mercado aceitar os custos ou se a produção em larga escala diminuir os preços.
  - Processadores: a grande vantagem dos processadores é sua flexibilidade. Com processadores, o comportamento dos sistemas embarcados pode mudar apenas mudando o software que roda esses processadores. Mudanças de comportamento podem ser necessárias a fim de corrigir erros de produção, para mudar o sistema para um outro melhor a fim de adicionar benefícios ao sistema anterior. Devido a isso os processadores estão ficando muito populares.

Processadores embarcados têm de ser eficientes e eles não precisam ser compatíveis com os comumente utilizados nos computadores. Portanto, suas arquiteturas podem ser diferentes daquelas encontradas nos PCs. Do ponto de vista da eficiência, há um grande número de diferenças:

- eficiência de energia: A arquitetura deve ser otimizada pela sua eficiência de energia e tem-se de ter certeza de que não se está perdendo eficiência no processo de geração de energia. Por exemplo, compiladores gerando 50% de cabeçalho em termos do número de ciclos levará a eficiência do ASICs, possivelmente para mais do que 50%, se a tensão de alimentação e a freqüência do clock aumentarem a fim de obterem limites. Há um grande número de técnicas disponíveis em que se pode aumentar a energia dos processadores.
- eficiência do tamanho do código: minimizar o tamanho do código é muito importante para sistemas embarcados, uma vez que o disco rígido não está disponível e a capacidade de memória é tipicamente limitada também.
- lógica reconfigurável Em muitos casos, ASICs são caros e soluções baseadas em software são muito lentas ou consomem muita energia. Lógica reconfigurável fornece uma solução se os algoritmos puderem ser facilmente implementados. Pode ser tão rápida quanto um hardware especial. Além disso, ao contrário de um hardware especial, a função pode ser mudada reconfigurando dados. Devido a essas propriedades, lógica reconfigurável é bastante usada nas seguintes áreas:
  - Protótipos rápidos: ASICs modernos podem ser muito complexos e o desenvolvimento de ferramentas pode durar muito tempo. Entretanto, geralmente, é desejável gerar um protótipo que pode ser usado como experimento de um sistema que se comporta quase como o sistema final. O protótipo pode ser mais oneroso e maior do que o sistema final. Além disso, seu consumo de energia também pode ultrapassar o do sistema final. Tal sistema, pois, pode ser usado para checar o comportamento fundamental do futuro sistema.
  - Aplicações de baixo volume: se o tamanho do mercado é muito pequeno para justificar o desenvolvimento de um ferramenta ASICs especial, a lógica reconfigurável pode ser a tecnologia de hardware certa para aplicações que não podem ser implementadas em software. Hardware reconfigurável tipicamente inclui memória RAM para estocar configurações durante operações normais de hardware. Tal RAM é, normalmente, volátil (a informação é armazenada apenas enquanto a potência é aplicada). Logo, os dados de configuração devem ser copiados numa configuração RAM at power-up.

– FPGA: a forma mais comum de se reconfigurar um hardware. Tal instrumento é programado "no campo"após sua fabricação.

4.2.2 MEMÓRIAS

Dados, programas e configurações de FPGA devem ser armazenados em algum tipo de memória. Isso deve ser feito de um modo eficiente. Eficiência, neste caso, significa um tempo de resposta eficiente, tamanho do pacote e energia eficientes. Eficiência no tamanho do pacote demanda um bom compilador e pode ser melhorada com a compressão desse pacote. Hierarquias de memórias podem ser exploradas a fim de se conseguir uma resposta no tempo e energia eficientes. As razões citadas anteriormente é que grandes memórias demandam mais energia por acesso e são mais lentas que memórias pequenas.

4.2.3 SAÍDAS

Instrumentos de saída de sistemas embarcados incluem os *displays*. A tecnologia de *displays* é uma área extremamente importante. Um grande volume de informação existe a respeito dessa tecnologia. Os maiores esforços em pesquisa e desenvolvimento atuais tratam das novas tecnologias de *displays* incluindo displays orgânicos.

Instrumentos de saída analógicos e digitais são usados. No caso da saída analógica, a informação digital deve ser, primeiramente, convertida por conversores D/A.

4.2.4 ATUADORES

Há um grande número de atuadores. Eles vão dos enormes, que são capazes de mover toneladas aos menores necessários para suportarem pesos minúsculos com dimensões da ordem de áreas micrométricas. Como exemplo, um atuador que terá grande importância no futuro será aquele que utiliza tecnologia de sistemas micrométricos. Atualmente, os cientistas buscam fabricar atuadores minúsculos que podem ser inseridos no corpo humano, por exemplo. Com isso, a quantidade de drogas introduzidas em nosso corpo pode ser adaptada para sua real necessidade, não atingindo outras regiões do organismo. Isso permite uma melhor medicação de pacientes.

4.3 SOFTWARE PADRÃO: SISTEMA OPERACIONAL DE DISPOSITIVOS EMBARCADOS E MIDDLEWARE

Nem todos os componentes de sistemas embarcados necessitam ser projetados começando do zero, pois existem componentes padrões que podem ser reusados. Estes componentes exigem um pré-conhecimento do projeto e constituem a chamada Propriedade Intelectual (PI). A reutilização da PI é uma técnica chave para lidar com a complexidade crescente dos projetos.

Os componentes padrões de *software* que podem ser reutilizados incluem: Sistemas Operacionais Embarcados (SO), banco de dados *real-time* e outras formas de *middleware*. O último termo representa o *software* que fornece uma camada intermediária entre o OS e o *software* de aplicação.

4.3.1 SISTEMAS OPERACIONAIS EMBARCADOS

4.3.1.1 EXIGÊNCIAS GERAIS

Podemos citar algumas das principais características dos sistemas operacionais embarcados:

- Devido à grande variedade de sistemas embarcados, existe também uma grande variedade de requisitos para as funcionalidades de um SO embarcado. Para que um sistema trabalhe de forma eficiente, não é possível utilizar um SO que promova a união de todas essas funcionalidades. Portanto, é preciso que o SO seja flexível. A configurabilidade é, consequentemente, uma das principais características dos SOs embarcados.
- Há uma grande variedade de dispositivos periféricos empregados em sistemas embarcados. Muitos sistemas embarcados não possuem disco rígido, teclado, monitor ou um mouse. Não há notavelmente nenhum dispositivo que necessite ser suportado por todas as versões de SOs, com exceção talvez do temporizador do sistema. Portanto, faz sentido usar dispositivos relativamente lentos, tais como discos e redes, com tarefas especiais, ao invés de integrar seus drivers no sistema operacional.
- Mecanismos de proteção nem sempre são necessários, uma vez que sistemas embarcados são projetados tipicamente para uma única finalidade e programas não testados nunca são rodados. Depois de ser testado, um *software* é suposto seguro. Na maioria das vezes, sistemas embarcados não possuem mecanismos de proteção. Isso também se aplica ao input/output. Diferentemente de aplicações desktop, não há nenhuma razão de se implementar instruções de I/O porque as instruções e tarefas privilegiadas podem fazer seu próprio I/O, o que reduz as despesas das operações.
- Interrupções podem ser empregadas por qualquer sistema. Para aplicações em desktop, seria perigoso permitir que qualquer sistema usasse interrupções diretamente. Como os sistemas embarcados podem ser considerados testados completamente, não é necessário proteção e existe um controle eficiente sobre uma variedade de dispositivos. É possível permitir interrupções diretamente (iniciar/parar) das tarefas. Isso é substancialmente mais eficiente do que ir até o SO para a mesma finalidade. Contudo, se uma tarefa específica for conectada diretamente a alguma interrupção, pode ser difícil adicionar uma outra tarefa que também necessita ser iniciada por algum evento.
- Muitos sistemas embarcados são sistemas em tempo real (RT) e, por isso, o SO usado neste sistema deve ser um Sistema Operacional em Tempo Real (RTOS).

4.3.2 *MIDDLEWARE*

4.3.2.1 BANCOS DE DADOS *REAL-TIME*

O banco de dados *real-time* fornece uma maneira conveniente e estruturada de armazenar e de alcançar a informação. Desta maneira, os bancos de dados fornecem um API para a leitura e escrita da informação. Uma seqüência de operação de leitura e escrita é chamada transação. Uma exigência freqüente é que as transações não afetem o estado do banco de dados, a menos que sejam executadas até seu término total. Por isso, as mudanças causadas por transações não são normalmente consideradas finais até que estejam encaminhadas. Muitas transações requerem ser atômicas, isto é, o resultado final (o novo resultado do banco de dados) gerado por alguma transação deve ser o mesmo tanto se a transação for totalmente terminada ou não.

O estado do banco de dados resultante de uma transação deve ser consistente. As exigências de consistência incluem, por exemplo, que os valores dos pedidos de leitura pertencentes à mesma transação não descrevam um estado que nunca existiu no ambiente modelado pelo banco de dados. Além disso, para outro usuário do banco de dados, estados intermediários resultantes de uma execução parcial de uma

transação não devem ser visíveis (as transações devem ser desempenhadas como se fossem executadas isoladamente). Finalmente, os resultados das transações devem ser persistentes. Essa propriedade é chamada de durabilidade. Junto, as quatro propriedades apresentadas são conhecidas como ACID.

4.3.2.2 ACESSO A OBJETOS REMOTOS

Existem pacotes de *software* especiais que facilitam o acesso aos serviços remotos. O CORBA (*Common Object Request Broker Architecture*) é um exemplo disso. Com CORBA, objetos remotos podem ser acessados através de interfaces padronizadas. Clientes se comunicam com os topos locais, imitando o acesso aos objetos remotos. Estes clientes mandam informações sobre o objeto a ser acessado e parâmetros ao *Object Request Broker*, ORB. Com isso, o ORB determina a localização do objeto a ser acessado e envia esta informação por um protocolo padrão, por exemplo, o protocolo IIOP. Esta informação é então encaminhada ao objeto através de um esboço e a informação pedida pelo objeto é retornada usando o ORB outra vez.

5 PROJETO OSSIE EMBARCADO

5.1 HISTÓRICO E DESCRIÇÃO GERAL DO PROJETO

Inicialmente tínhamos como objetivo final do projeto o desenvolvimento de um receptor de sinais de rádio (mais especificamente um receptor FM) utilizando uma arquitetura embarcada. Para isso ser possível, utilizaríamos uma placa denominada *Universal Software Radio Peripheral*, mais conhecida como USRP, entre o *front end* RF e o computador. Os dispositivos de *front end* do projeto são pequenas placas-filha acopladas à USRP.

A fim de embarcar esse receptor e torná-lo independente do computador, fornecendo mobilidade a este, seria necessária a utilização de um processador embarcado. Em princípio utilizaríamos o processador NIOS II, um dispositivo de baixa velocidade muito utilizado em sistemas embarcados que já estaria disponível na Universidade de Brasília. Esse processador utiliza o Sistema Operacional Micro-C Linux, que é ideal para controladores, como o NIOS II, que não possuem MMU (*Memory Management Units* Ű Unidades de Gerência de Memória).

A ferramenta utilizada para gerar a forma de onda e demodular o sinal FM proposto no projeto é o OSSIE. O OSSIE foi desenvolvido pela Virginia Tech com o intuito de fornecer gratuitamente uma implementação do *framework* SCA disponível para estudos e pesquisas de Rádios Definido por Software. Como o funcionamento do OSSIE a princípio era desconhecido, foi necessário, além de estudar a estrutura da arquitetura SCA, ter-se como um objetivo específico rodar o OSSIE em um computador que contivesse o Fedora 5 como sistema operacional. Uma vez que se tivesse um maior entendimento e uma maior familiaridade com tal ferramenta, poderíamos dar continuidade visando alcançar o produto final do nosso projeto de graduação.

Nos tópicos seguintes descreveremos mais detalhadamente as características da USRP, do processador NIOS e do programa OSSIE.

5.1.1 A USRP

A USRP é um dispositivo que permite a criação de um Rádio Definido por Software em qualquer computador que possua uma entrada USB 2 (*Universal Serial Bus*). Ela é um componente de *hardware* barata que possibilita a implementação de sistemas de Rádio Definido por Software em tempo real. Foi desenvolvida com o foco voltado para ser utilizada com o GNU Radio. Entretanto foram realizadas experiências bem-sucedidas utilizando a placa com outros programas, como o programa OSSIE, utilizado em nosso projeto e apresentado mais à frente.

Além das placas filhas que funcionam como front end RF, a USRP possui um FPGA que realiza funções de interpolação e decimação dentre outras, tornando a placa um dispositivo extremamente multifuncional. Essa placa possui um conversor analógico-digital/ digital-analógico. Com isso, ela também atua como front end, ou seja, ela também executa funções típicas desse dispositivo.

Os dispositivos relatados acima, podem ser visualizados na Fig. 5.1 que mostra a placa USRP.

5.1.2 O NIOS II

O Nios II consiste em um processador de 32-bits RISC (*Reduced Instruction Set Computing*, Conjunto Reduzido de Instruções Computacionais) de propósito geral, desenvolvido para atender uma grande escala de dispositivos embarcados. As principais característica do Nios II são: conjunto de instruções, espaço

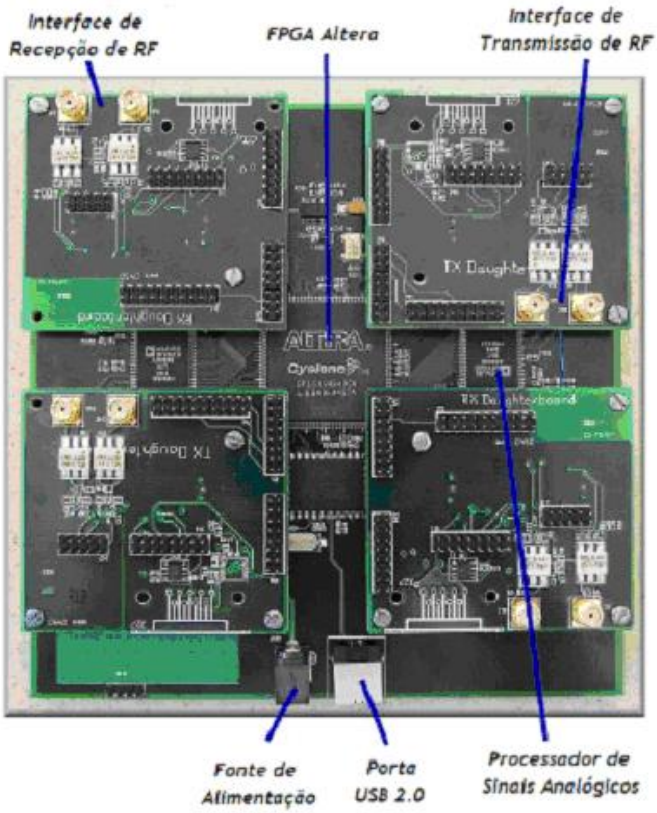

Figura 5.1: Placa USRP [4].

de endereçamento e data path de 32-bits; 32 registradores de propósito geral; 32 fontes de interrupções externas; instruções dedicadas ao cálculo de multiplicações com 64-bits e 128-bits; acesso a uma variedade de periféricos on-chip, e interfaces para acesso a memórias e periféricos off-chip; oferece cerca de 2 GBytes de espaço de endereçamento; e customização de até 256 instruções.

A utilização do processador NIOS seria de grande valia para o projeto, pois os idealizadores do OSSIE não testaram ainda um processador que não contém MMU (Memory Management Units).

A MMU é um dispositivo de hardware que transforma endereços virtuais em endereços físicos. Na MMU, o valor no registro de re-locação é adicionado a todo o endereço lógico gerado por um processo do utilizador na altura de ser enviado para a memória. Ou seja, o programa do utilizador manipula endereços lógicos; nunca vendo endereços físicos reais.

5.1.3 O OSSIE

O OSSIE é um Rádio Definido por Software de fonte aberta desenvolvida no Virgínia Tech. Ele tem a intenção de viabilizar a pesquisa e educação em Rádio Definido por Software e comunicações sem fio. O pacote de programas inclui um Core Framework RDS baseado no SCA do JTRS, ferramentas para o rápido desenvolvimento dos componentes RDS e formas de onda (aplicações). Além disso, desenvolve uma biblioteca de componentes pré-fabricados e formas de onda.

A implementação de fonte aberta SCA do projeto do OSSIE embarcado é uma iniciativa do grupo MPRG (Mobile and Portable Radio Research Group) do Virgínia Tech com a intuito de fornecer uma plataforma simples, fácil de expandir e de fonte aberta (disponibilidade do código fonte) para o desen-

volvimento de forma de onda seguindo as especificações da arquitetura SCA.

Seu desenvolvimento começou no ano de 2003 e continua até os dias de hoje. O projeto OSSIE é escrito em C++ utilizando o OmniORB CORBA ORB, que está disponível livremente. Atualmente o desenvolvimento está focado no sistema operacional Linux.

A missão do Virgínia Tech com a implementação do OSSIE é desenvolver o mais simples, mais portátil implementação da SCA de fonte aberta e as ferramentas para um protótipo veloz a fim de melhorar a educação e pesquisa em Rádio Definido por Software.

O OSSIE foi desenvolvido utilizando o Sistema Operacional Fedora Core (4, 5 ou 6). Pode ser possível que rode em outras versões do Linux, entretanto essas não possuem garantia pelos idealizadores no Virgínia Tech. Há uma restrição quanto à utilização do Micro-C Linux apresentada em um documento.

Os dispositivos de hardware complementares utilizados a fim de embarcar o OSSIE pelo Virgínia Tech foram: a USRP (Universal Software Radio Peripheral) e suas placas filhas. As placas filhas utilizadas no nosso projeto foram a Basic TX e a Basic RX.

O OSSIE está sendo aprimorado dia após dia pelos seus idealizadores. Existem várias versões para o programa, entretanto apenas uma pode ser considerada estável. Essa versão é disponibilizada no web site como a versão 0.6.1, e foi utilizando ela que conseguimos realizar experiências sobre o que o OSSIE é capaz de executar. Essas experiências serão apresentadas a seguir.

5.2 IMPLEMENTAÇÃO E EXECUÇÃO DO OSSIE

Até o momento presente, não conseguimos instalar o OSSIE com êxito em nenhuma de suas versões. Entretanto pudemos verificar algumas funções do OSSIE, pois seus idealizadores disponibilizam para os usuários uma Máquina Virtual que já contém o programa instalado. O tópico 5.2.1 explicará os requisitos e os pacotes necessários para ter-se o OSSIE instalado em um ambiente nativo (o próprio computador) assim como apresentará os passos da instalação que não obtivemos êxito e as possíveis causas de seus erros. O subitem 5.2.2 iniciará o conceito de Máquina Virtual para que então, em 5.2.3, possamos mostrar os resultados obtidos com a utilização desta, através da realização de dois laboratórios que são disponibilizados pelo sítio do oficial OSSIE.

5.2.1 INSTRUÇÕES PARA INSTALAÇÃO DO OSSIE

5.2.1.1 PRÉ-REQUISITOS

O core framework do OSSIE foi desenvolvido com sucesso para ser instalado no Fedora 4, 5 ou 6 ou distribuições SUSE do Linux. Os passos apresentados a seguir são exclusivamente para versões 5 e 6 do Fedora.

Para instalar e rodar o OSSIE é necessário que se tenha anteriormente o Xerces-C e OmniORB em seu computador. O segundo passo, requer que o Amara, omniORBpy e wxPython também estejam instalados.

5.2.1.2 FERRAMENTAS DO OSSIE

Os erros da versão 0.6.0 foram reparados, assim como suportes para novas interfaces e para o omniORB 4.1.0 foram inclusas na nova versão 0.6.1 do OSSIE. Outra novidade nesta versão é a biblioteca SigProc, que facilitará escrever novas componentes.

Assim como as ferramentas do OSSIE, diversas amostras de formas de onda estão disponíveis para

download no sítio da Virginia Tech. Essas ferramentas da versão 0.6.1 podem ser resumidas em três passos, os quais devem ser completados antes de se seguir para o próximo. A primeira etapa consiste em instalar e verificar uma cópia do *Core Framework* e dos dispositivos da plataforma, que permitirão rodar formas de ondas no computador. O segundo passo a ser tomado é a instalação e verificação do funcionamento de amostras de formas de ondas e de componentes, assim como do OWD (OSSIE Waveform Developer), que é uma ferramenta de interface gráfica que permite esboçar, editar e gerar novas componentes e formas de ondas. O último passo é a instalação de dispositivos e componentes adicionais para o uso da USRP que permitirão receber e demodular formas de ondas AM e FM em suas respectivas bandas. Abaixo são mostrados os pacotes que devem ser instalados em cada um dos passos acima descritos:

Passo 1

cf/ossie0.6.0.tar.gz (ossie0.6.1.tar.gz)
cf/standardinterfaces0.6.0.tar.gz (standardinterfaces0.6.1.tar.gz)
platform/gpp0.6.0.tar.gz (gpp0.6.1.tar.gz)
platform/nodebooter0.6.0.tar.gz (nodebooter0.6.1.tar.gz)
platform/nb_test0.6.0.tar.gz (nb_test0.6.1.tar.gz)

Passo 2

waveforms/ossiedemo0.6.0.tar.gz (ossiedemo0.6.1.tar.gz)
components/ChannelDemo0.6.0.tar.gz (ChannelDemo0.6.1.tar.gz)
components/RxDemo0.6.0.tar.gz (RxDemo0.6.1.tar.gz)
tools/wavLoader0.6.0.tar.gz (wavLoader0.6.1.tar.gz)
tools/WaveDev0.6.0.tar.gz (WaveDev0.6.1.tar.gz)

Passo 3

cf/SigProc0.6.1.tar.gz
platform/USRP0.6.0.tar.gz (USRP0.6.1.tar.gz)
waveforms/USRP_Ctrl0.6.0.tar.gz (USRP_Ctrl0.6.1.tar.gz)
platform/Sound_Out0.6.0.tar.gz (Sound_Out0.6.1.tar.gz)
components/FM_Demodulator0.6.0.tar.gz
(FM_Demodulator0.6.1.tar.gz)
components/am_demod0.6.0.tar.gz (am_demod0.6.1.tar.gz)
components/Decimator0.6.0.tar.gz (Decimator0.6.1.tar.gz)

5.2.1.3 INSTALAÇÃO DO OSSIE

√ ***Xerces-C***

Pode ser encontrado no sítio http://xml.apache.org/xercesc/XercesC a fim de criar ou customizar as variáveis de ambiente, deve-se seguir os passos:

```
$ export CC=mysupperfastCC
$ export CXX=mysuperfastC++
$ export XERCESCROOT=diretório gerado da extração do xercescsrc2.7.0.tar.gz
$ cd $XERCESCROOT/src/xercesc
$ ./runConfigure-plinux
$ make
$ su -c "make install"
```

√ ***omniORB***

É aconselhavel baixar a versão 4.1.0 disponível no sítio http://omniorb.sourceforge.net/download.html pois as outras ainda não foram testadas. Pode-se então extrair o arquivo em qualquer diretório que se desejar.

```
$ cd <omniORBV.V.V> (onde V.V.V se refere a versão do omniORB que está sendo baixada)
$ mkdir build
$ cd build
$ ../configure
$ make
$ su -c "make install"
```

Deve-se editar e copiar as configurações do omniORB como root.

```
$ cp $OMNIORB_TOP/sample.cfg /etc/omniORB.cfg
```

Editar o arquivo omniORB.cfg: na linha 317, deve-se trocar InitRef = NameService=corbaname::my.host.name para InitRef = NameService=corbaname::localhost e certificar-se de que a linha não está comentada.

Atualização dos locais aonde se encontram as bibliotecas compartilhadas: Editar o arquivo: /etc/ld.so.conf e adicionar a linha /usr/local/lib Neste ponto deve-se rodar ldconfig no diretório /sbin/.

Configurando o omniNames: Criar um diretório chamado "logs"em qualquer local que se deseje.

```
$ mkdir /caminho/logs
```

Crie um arquivo omniNames.sh e coloque as linhas que se seguem dentro deste:

```
# !/bin/sh
killall omniNames [omniEvents]
rm /caminho/logs/omninames*
omniNames start logdir
/caminho/logs &
```

Agora, como root:

```
# chmod 755 omniNames.sh
# cp omniNames.sh /usr/local/bin
```

Assim, este script "omniNames.sh"poderá ser acessado como root de qualquer diretório.

√ **Construindo e instalando o OSSIE:**

```
$ cd <diretório onde foi extraído o ossie 0.6.1>
$ ./reconf
$ ./configure
$ make
$ su -c "make install"
```

√ **Construindo e instalando standardInterfaces, nodebooter and GPP**

```
$ mkdir /home/sca
$ chown nomedousuário.nomedogrupo /home/sca
$ cd <diretório standardInterfaces>
$ ./reconf
$ ./configure
$ make
$ su -c "make install"
$ cd <diretório do nodebooter>
$ ./reconf
$ ./configure
$ make
$ su -c "make install"
$ cd <diretório do GPP>
$ ./reconf
$ ./configure
$ make
$ su -c "make install"
```

Agora, deve-se mais uma vez rodar o ldconfig como root no diretório /sbin.

√ **Extraindo o nb_test:**

```
$ cd /home/sca
$ cd xml
$ su -c "tar -xzvf <caminho do nb_test.tar.gz>"
```

√ **Testando a instalação:**

Em um novo terminal do Linux deve-se iniciar o omniNames, digitando como root:

```
$ omniNames.sh
```

Em uma outra janela, testa-se o nodebooter no diretório mostrado a seguir:

```
$ cd /home/sca/xml/nb_test
$ nodeBooter -D -d PC.dcd.xml
```

Se o teste ocorrer com sucesso, a ultima linha mostrará a mensagem "Device Registered". Para finalizar o nodebooter e o omniNames, deve-se apertar simultaneamente Ctrl C.

Essa é a etapa da instalação do programa que apresentou o erro o qual não conseguimos resolver. Isso ocorre porque a estrutura de diretórios vem sendo modificada todos os dias pelos integrantes do projeto realizado na Virginia Tech. Atualmente, o sítio apresenta uma estrutura de diretórios onde as componentes e os Devices devem estar no diretório /sdr. Como dessa forma não obtivemos sucesso na instalação, encontramos um documento que apresentava uma chamada "nova estrutura". Esta também foi testada sem êxito.

Como já citado anteriormente, a Virginia Tech disponibiliza uma Máquina Virtual (que será apresentada mais adiante) com o OSSIE 0.6.1 já instalado e funcionando perfeitamente. Com isso, tentamos utilizar a mesma estrutura no Fedora nativo e o erro persistiu.A estrutura de diretórios da Máquina Virtual está apresentada na Fig. 5.2.

A idéia proposta por esta estrutura é de se ter um local comum para a instalação das componentes e formas de ondas do sistema. Na figura acima, o local comum de instalação é o diretório /sca, geralmente criado no /home. Existem então três diretórios dentro deste principal: o bin, o xml e o waveforms. Estes contêm todos os arquivos necessários usados para rodar uma forma de onda. O bin deve conter todos os arquivos binários (executáveis) para cada uma das componentes e dos Devices que tenham sido instalados. O xml contém subdiretórios com o nome de cada componente e Device instalados no sistema e, dentro

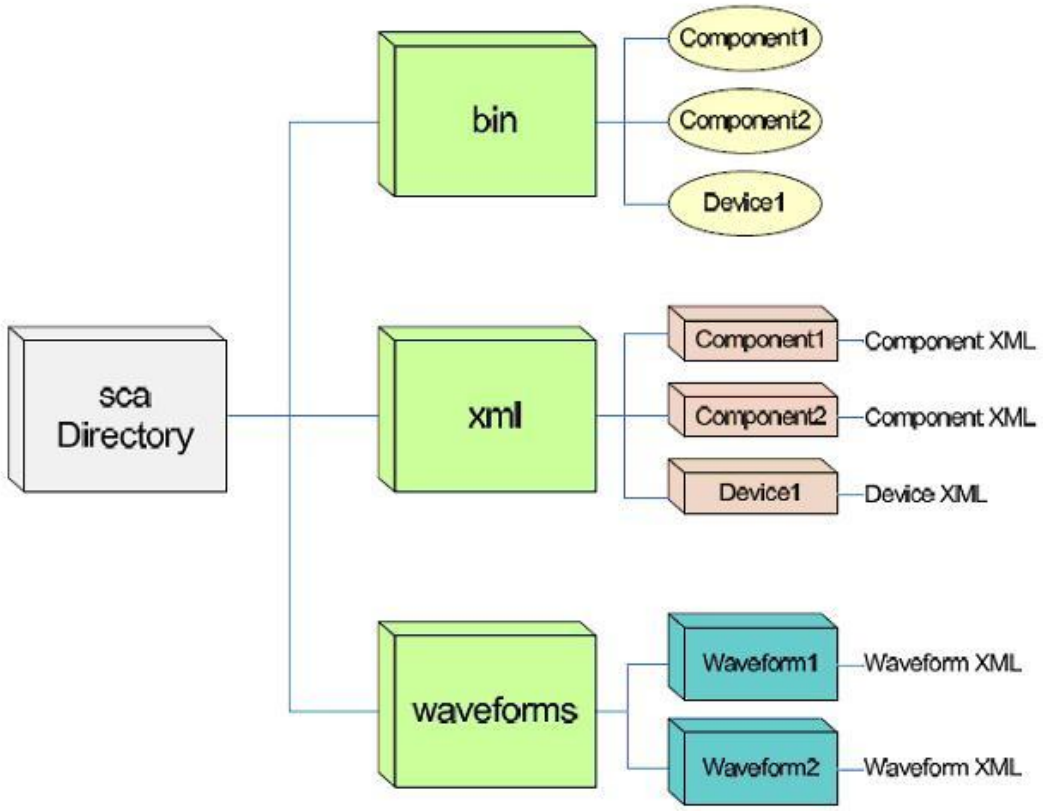

Figura 5.2: Estrutura de diretórios da Máquina Virtual [5].

desses, pode ser encontrado uma série de arquivos ".xml". Enfim, no diretório waveform, estarão os arquivos "xml"de cada forma de onda instalada no sistema. Contudo, os seguintes passos foram efetuados na tentativa de uma futura correção do erro, uma vez que o próprio site informa que este pode ocorrer na execução do comando nodeBooter.

√ **Instalando o *wavLoader*, o OWD e o *omniORBpy*:**

O omniORBpy source pode ser encontrado no sítio http://omniorb.sourceforge.net/.

```
$ tar - xzvf omniORBpyV.V.tar.gz (V.V se refere a versão que está sendo instalada)
$ cd <omniORBpy>
$ ./configure
$ make
$ su - c "make install"
```

√ ***Amara*:**

Pode-se encontrar o Amara no seguinte sítio: http://uche.ogbuji.net/tech/4suite/amara/. Para fazer o download do file "Amara1.0allinone.tar.gz"deve-se navegar em ftp://ftp.4suite.org/pub/Amara/ e clicar com o botão direito do mouse selecionando "salvar como". Para descompactar o arquivo seguem as instruções adiante:

```
$ tar - xzvf Amara1.0allinone.tar.gz
$ cd <diretório do Amara1.0allinone>
$ su - c "python setup.py install"
```

√ ***wxPython***

Baixe a ultima versão do *wxPython.*

```
$ mkdir idl_py
$ cd idl_py
$ cp <caminho do arquivo ossie idl >/*.idl .
$ omniidl -p<caminho do omniORBpy > bpython*
```

O caminho do ossie idl é tipicamente ”/usr/local/include/ossie/” e o caminho do omniORBpy é o ”/usr/local/lib/python<version>/sitepackages”.

```
$ cd /usr/lib/python<version>/sitepackages/
$ su - c "vim ossie.pth"
```

O ossie.pth deverá ter o seguinte conteúdo:

```
<caminho do idl_py>
/usr/local/lib/python2.x/sitepackages/
```

Novamente deve-se rodar o ldconfig no diretório /sbin como root.

√ **Baixando o *wavLoader*:**

Uma cópia do wavLoader deverá ser criada em /usr/bin e dentro do diretório onde estarão as formas de onda criadas.

```
$ cp <caminho do wavLoader>/wavLoader.py /home/sca/waveforms/<nome da forma de onda>
```

√ **Forma de onda demo do OSSIE**

Essa forma de onda consiste em três componentes: o RxDemo, o TxDemo e o ChannelDemo. Este código demonstra a comunicação e a aparência básica de uma forma de onda.

O TxDemo atua no ossie_demo como um controlador assembly. O RxDemo e ChannelDemo completam a cadeia de componentes da forma de onda do ossie_demo.

```
$ cd <caminho do ossie_demo>
$ ./reconf
$ ./configure
$ make
$ su -c "make install"
```

O mesmo então é feito para cada uma das componentes.

```
$ cd <diretório do TxDemo>
$ ./reconf
$ ./configure
$ make
$ su -c "make install"
$ cd <diretório do ChannelDemo>
$ ./reconf
$ ./configure
$ make
$ su -c "make install"
$ cd <diretório do RxDemo>
$ ./reconf
$ ./configure
$ make
$ su -c "make install"
```

Desta forma tem-se baixado no computador todos pacotes necessários para se rodar o OSSIE. Para que diversas formas de onda sejam implementadas pelo OSSIE, este dispõe de uma ferramenta de uma

interface gráfica, fácil de ser utilizada, denominada OWD (OSSIE Waveform Developer). O OWD pôde ser utilizado com sucesso com o uso do OSSIE 0.6.1 pré-instalado na Máquina Virtual, apresentada no próximo tópico.

5.2.2 A VIRTUALIZAÇÃO E A MÁQUINA VIRTUAL (VMWare)

Virtualização é uma tecnologia de software comprovada que transforma rapidamente a paisagem de TI e fundamentalmente mudando a forma como as pessoas utilizam o computador.

A máquina poderosa x86 foi originalmente desenvolvida para rodar somente sistemas de operações simples e aplicações simples, entretanto a virtualização "quebra"essa obrigatoriedade, tornando possível rodar sistemas de múltiplas operações e aplicações na mesma máquina ao mesmo tempo, ampliando a utilização e a flexibilidade do hardware.

É uma tecnologia que pode beneficiar qualquer um que utilize o computador, desde profissionais da área de TI a organizações governamentais. Engloba milhões de pessoas ao redor do mundo que usam virtualização para economizar dinheiro, tempo e energia alcançando mais do que o hardware de um computador sozinho.

Em sua essência, a virtualização permite que se transforme hardware em software. Utilizando-se software para virtualizar os recursos de hardware Ű incluindo a CPU, a RAM, o disco rígido e o controlador de rede - criando uma máquina virtual repleta de funcionalidades que rodam suas aplicações e sistemas operacionais exatamente como um computador convencional.

Múltiplas máquinas virtuais podem dividir os recursos de hardware sem interferir uma na outra, permitindo que se rode diversos sistemas operacionais em um mesmo computador.

O acesso à VMWare se faz através da inserção de uma fina camada de software diretamente no hardware do computador ou em um hospedeiro do sistema operacional. Essa camada de software cria uma máquina virtual e contém um monitor de máquina virtual ou "Hyperviso"que aloca recursos de hardware dinamicamente e de forma transparente a fim de multiplicar os sistemas operacionais pode rodar ao mesmo tempo em um computador físico simples sem ao menos ter conhecimento disso.

Entretanto, virtualização em uma simples máquina física como um computador é só o começo. VMWares oferecem um robusta plataforma de virtualização que podem subir para centenas de computadores físicos interconectados e unidades de armazenamento a fim de formar uma infraestrutura virtual completa.

Com a VMWare foi possível a realização de dois laboratórios propostos no sítio do OSSIE que permitiu maior conhecimento do programa e foi possível obter o espectro de freqüência e o gráfico de Magnitude de algumas formas de onda. Entretanto, com esses experimentos não nos sentimos satisfeitas em relação à utilização do programa, pois não utilizamos a placa USRP.

Tal utilização não foi possível devido a uma incompatibilidade existente entre a USB2 da placa e a USB do computador. Tal incompatibilidade foi atestada, pois, durante todo nosso projeto, caminhamos utilizando duas máquinas concomitantemente: um notebook pertencente ao programa Rádio Definido por Software e um computador localizado na sala de hardware do GPDS (Grupo de Processamento Digital de Sinais) do SG-11 da Universidade de Brasília. As duas máquinas apresentaram o mesmo tipo de erro. Além disso, é sabido, entretanto não divulgado "oficialmente", que algumas VMwares apresentam incompatibilidade com a entrada USB.

5.2.3 LABORATÓRIOS REALIZADOS COM A MÁQUINA VIRTUAL

Antes de explicar como os laboratórios foram efetuados e de mostrar os resultados obtidos é importante entender o funcionamento do OWD, que representa uma ferramenta simples e essencial para gerar formas de onda.

Existem algumas considerações que devem ser levadas em conta antes de se projetar formas de onda utilizando o OWD. Primeiramente, a aparência desejada do rádio deve ser considerada e traduzida de forma apropriada para o componente de software padrão. Em segundo lugar, as componentes input e output devem ser analisadas para que métodos apropriados sejam escolhidos, como a escolha do controlador Assembly. A consideração final envolve a estratégia no desenvolvimento de Devices. Todas essas implicações devem ser cuidadosamente resolvidas antes de se implementar uma forma de onda.

Para iniciar o OWD, deve-se navegar até o diretório src/WaveDev/wavedev na linha de comando do Linux e digitar python wd.py. A janela principal desta ferramenta irá se abrir conforme mostrado na Fig. 5.3.

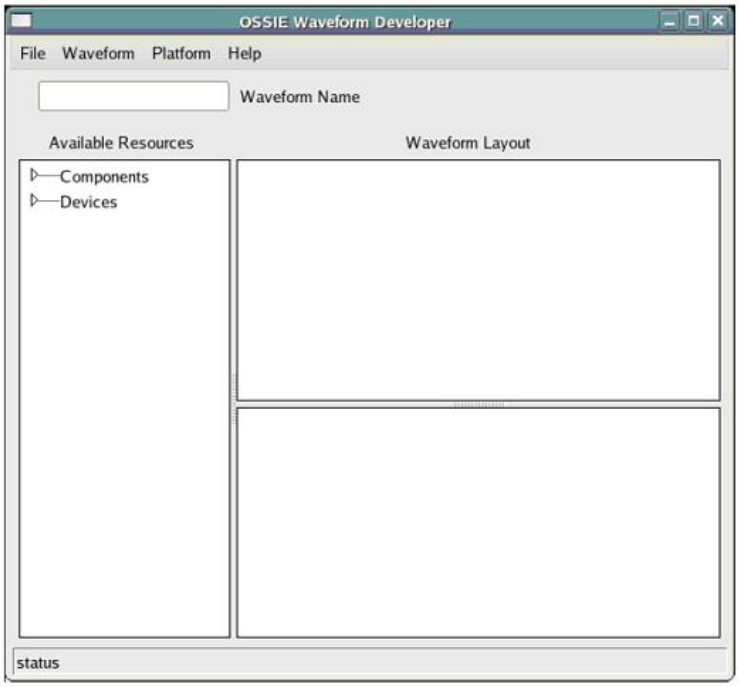

Figura 5.3: Menu Principal do OWD

É importante definirmos algumas siglas que serão utilizadas frequentemente no laboratório:

- GPP = General Purpose Processor (o processador do computador).
- CORBA = Common Object Request Broker Architecture
- SCA = Software Communications Architecture
- OSSIE = Implementação do SCA
- OWD = OSSIE Waveform Developer
- Fedora Core 5 = Versão do Linux disponível gratuitamente
- waveform = seu sistema padrão de comunicação
- component = uma função do sistema de comunicação que é implementada. As components geralmente implementam funções muito simples, porém, se forem conectadas umas com as outras, pode-se conseguir a implementação de funções mais complexas.

- device = hardware que implementará as funções das components e das waveforms. Exemplo: GPPs, FPGAs, USRP, etc.
- node = é um Device ou um grupo deles.
- uses port = normalmente (mas nem sempre) representam as portas output
- provides port = normalmente (mas nem sempre) representam as portas input
- QPSK= A modulação QPSK é uma técnica de modulação derivada do PSK, porém neste caso, são utilizados parâmetros de fase e quadratura da onda portadora para modular o sinal de informação. Como agora são utilizados dois parâmetros, existem mais tipos possíveis de símbolos nesta constelação, o que permite que sejam transmitidos mais bits por símbolo. Por exemplo, se quisermos transmitir 2 bits por símbolo, ao invés de 1 bit por símbolo como no caso PSK acima, neste caso, como teremos 4 tipos de símbolos possíveis, a portadora pode assumir 4 valores de fase diferentes, cada um deles correspondendo a um bit, como por exemplo 45^o, 135^o, 225^o e 315^o.
- PSK = (Phase Shift Keying) é uma forma de modulação em que a informação do sinal digital é embutida nos parâmetros de fase da portadora. Neste sistema de modulação, quando há uma transição de um bit 0 para um bit 1 ou de um bit 1 para um bit 0, a onda portadora sofre uma alteração de fase de 180 graus. Esta forma de particular do PSK é chamada de BPSK (Binary Phase Shift Keying). Quando não há nenhuma destas transições, ou seja, quando bits subseqüentes são iguais, a portadora continua a ser transmitida com a mesma fase.
- AWGN = O canal de comunicação via satélite é um canal AWGN (Additive White Gaussian Noise) onde predominam fortes atenuações e muitas vezes grandes atrasos de propagação do sinal. O termo AWGN é utilizado em modelamentos matemáticos para caracterizar aqueles canais onde o tipo de ruído responsável por degradar a comunicação é um ruído branco adicionado ao sinal. Este tipo de ruído é um dos mais "bem comportados"e a teoria acerca do desenvolvimento de receptores ótimos para utilização em canais AWGN já se tornou clássica. O ruído branco é um sinal aleatório e tem um modelamento matemático que o considera como possuindo largura de faixa infinita, média nula e correlação nula entre suas amplitudes tomadas a instantes de tempo distintos, ou seja, o valor da amplitude do ruído em um determinado instante independe daquele observado em outro instante de tempo qualquer. O termo gaussiano se deve ao fato desse tipo de ruído possuir uma função densidade de probabilidade gaussiana com média nula, com desvio padrão igual à sua tensão rms (root-mean-square) e variância igual à potência dissipada em um resistor de 1W.
- XML = arquivos que fazem parte integral de uma forma de onda SCA e devem ser apropriadamente gerados pelo OWD.

5.2.3.1 LABORATÓRIO 1

O objetivo deste exercício de laboratório é possibilitar aos estudantes a oportunidade de construir uma forma de onda e experimentar novos valores para obter-se uma melhor resposta do sistema. Será construída uma simples forma de onda que simula um sistema de comunicação QPSK. Os dados do QPSK serão mandados para um amplificador linear antes de serem enviados para o canal do AWGN. Depois que o ruído for adicionado ao sinal, os dados resultantes serão plotados. Feito isso, o ganho será incrementado para obtenção de um sinal melhor.

Tendo aberto a janela principal do OWD, devemos nomear a nossa waveform (ou forma de onda) na caixa a isso destinada. Chamaremos nossa forma de onda de lab1 e para a salvarmos devemos utilizar o menu principal do OWD - "File -> Save Project As...", conforme a Fig. 5.4.

Devemos agora adicionar um node ao nosso projeto. Este representará o computador onde as components rodarão. No menu principal do OWD navegue em Platform e em seguida selecione Add Deployment

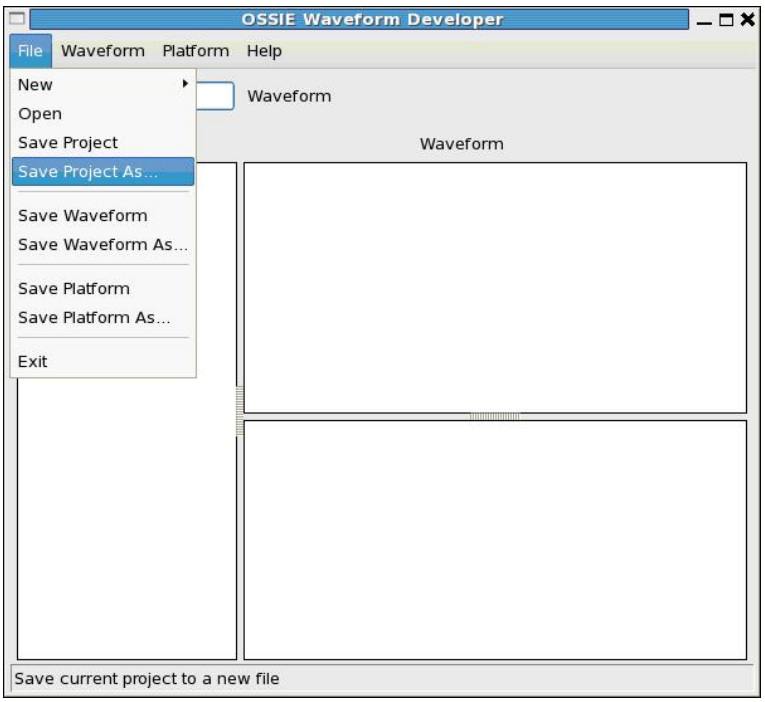

Figura 5.4: Salvando o Projeto

Node. Nomeie seu node como "Node1". Ao clicarmos no triângulo ao lado de Devices, expandiremos a lista que mostrará todos que são disponibilizados, assim, conforme a Fig. 5.5, devemos adicionar um Device ao Node1 criado anteriormente. Para isso, com o botão direito do mouse em GPP, selecionaremos a opção "Add to node"e nomearemos como GPP1.

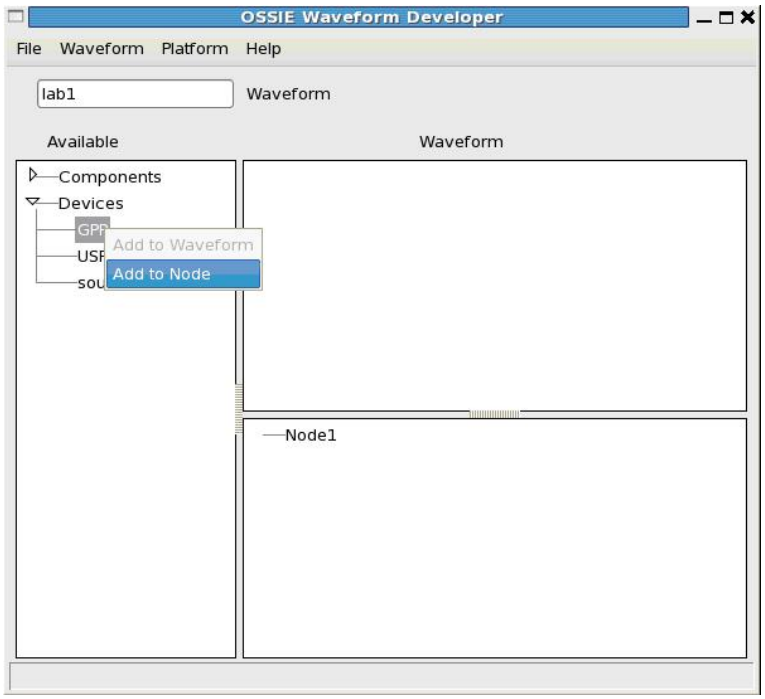

Figura 5.5: Lista de *Devices*

Para verificar se o GPP realmente foi adicionado ao Node1, pode-se observar se abaixo deste aparecerá

o GPP1. O próximo passo é adicionar as componentes necessárias para a forma de onda que pretendemos obter. Utilizaremos o QPSKmod, amplifier, AWGNChannel_complexShort e o Graph, conforme mencionado no objetivo deste laboratório. Devemos então expandir a lista de components e para cada uma dessas citadas, clicar com o botão direito e selecionar "Add to Waveform". Conforme as componentes vão sendo adicionadas, estas aparecerão na parte direita do OWD conforme Fig. 5.6.

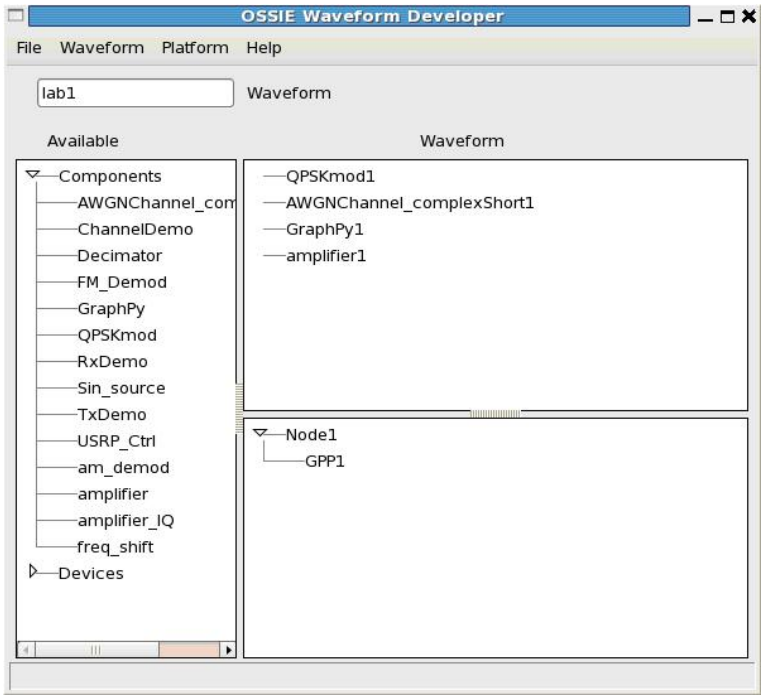

Figura 5.6: Adicionando as Componentes ao Projeto

Podemos associar cada uma das componentes a um node e device. Para tal, deve-se clicar duas vezes em cima do componente waveform QPSKmod1 e com isso abrirá uma nova janela denominada OSSIE Component Editor mostrada na Fig. 5.7. Abaixo de Deployment, devemos selecionar Node1 para node e GPP1 para Device. Isso será repetido para as outras três componentes, porém, somente o QPSKmod1 será marcado como o controlador Assembly. Cada forma de onda deve ter apenas um controlador Assembly. Quando rodamos uma forma de onda, a primeira coisa que acontece é a chamada da função "start"do controlador Asssembly. Portanto, este deve ter a autonomia de se iniciar primeiro e então controlará as demais componentes, iniciando-as ou desligando-as conforme é precisado. Quando voltarmos ao menu principal do OWD, perceberemos que o controlador Assembly, ou seja, a componente QPSKmod1, estará em negrito.

Neste ponto, faremos as conexões necessárias das portas de cada componente. Clicando com o botão direito do mouse em QPSKmod1 e selecionando "connect"a janela de conexões se abrirá (Fig. 5.8). No lado esquerdo deve-se selecionar "OutPortTx1"e no lado direito selecionar "datain", abaixo de amplifier1. Apertando "connect"a conexão feita deverá aparecer no espaço mais abaixo conforme ilustra a Fig.5.8.

Clicando com o botão direito em amplifier1 e selecionando "connect"deve-se, agora, iluminar do lado esquerdo do painel de conexões o "dataOut"e do lado direito o "inPort"abaixo do "AWGNChannelcomplexshort"conforme apresentado na Fig. 5.9. Feito isso aperta-se "Connect"e a nova conexão aparecerá na caixa de conexões.

Utilizando o mesmo procedimento, deve-se conectar a "outPort"do AWGNChannelcomplexshort com a porta disponível abaixo de GraphPy (ilustração da Fig. 5.10)

Neste ponto, devemos gerar os arquivos que definem uma waveform. Selecione "Waveform - genarate"no

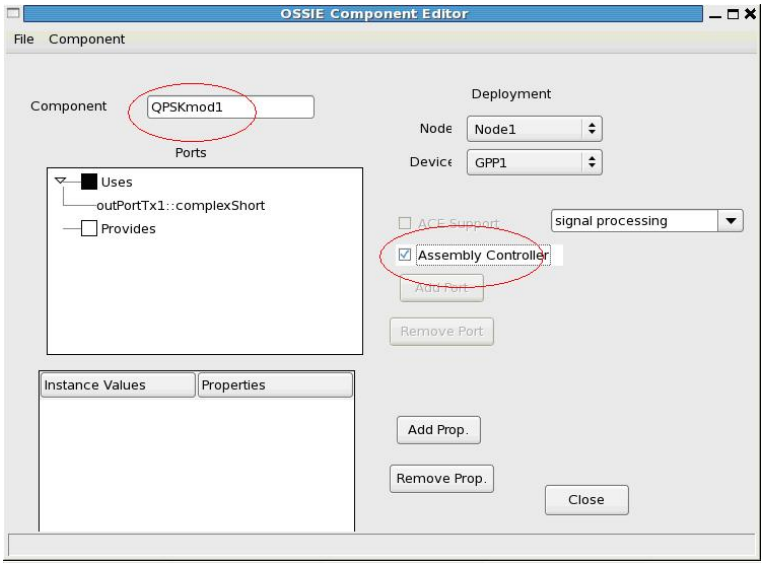

Figura 5.7: OSSIE *Component Editor*

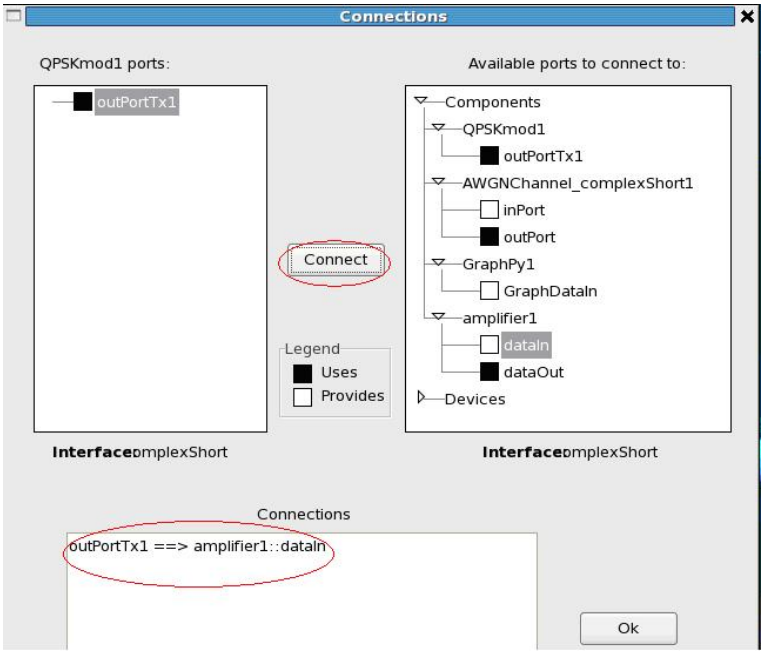

Figura 5.8: Janela de Conexões do QPSKmod1

menu OWD. Na janela que se abrirá, apresentará o diretório /home/ossie/src/WaveDev/wavedev. Não deve ser salvo aqui! E sim no /home/ossie/src/waveforms/<nome_da_sua_waveform>, neste caso, /home/ossie/src/waveforms/lab1. Salve seu projeto novamente e feche o OWD.

Ao se navegar no diretório /home/ossie/src/waveforms pode-se observar um novo diretório chamado lab1. Este contém diversos arquivos XML. Entrando em lab1 deve-se instalar a nossa forma de onda utilizando os comandos abaixo:

```
./reconf
./configure
su -c "make install"
```

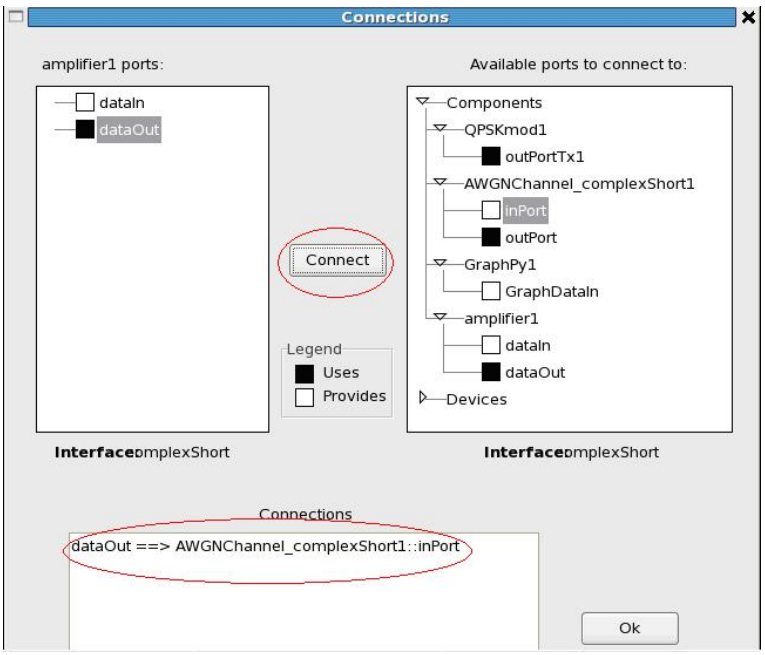

Figura 5.9: Janela de Conexões do Amplifier1

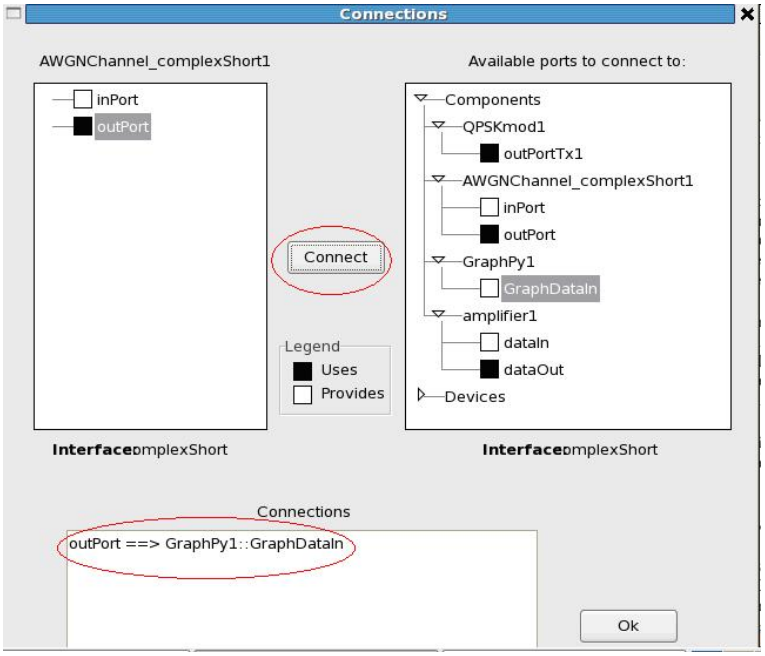

Figura 5.10: Janela de Conexões do AWGNChannel

Entrando no diretório /home/sca/waveforms/lab1 perceberemos que foi salva uma cópia dos arquivos XML. Devemos rodar o "CORBA Naming Service"digitando como root omniNames.sh. A mensagem "checkpointing completed"deverá aparecer.

Em seqüência, devemos iniciar o Domain Management e registrar nosso Device rodando o programa nodeBooter digitando nodeBooter -D -d DeviceManager.dcd.xml. A mensagem "Device Registered"deverá aparecer. Em outro terminal do Linux, deve-se navegar até o diretório /home/sca/waveforms/lab1 e iniciar o wavLoader digitando wavLoader.py lab1_DAS.xml. Escolhendo a opção 1 e em seguida "s", serão

plotados os seguintes gráficos, referente ao diagrama de constelação e ao espectro de frequência (Figs .5.11 e 5.12, respectivamente).

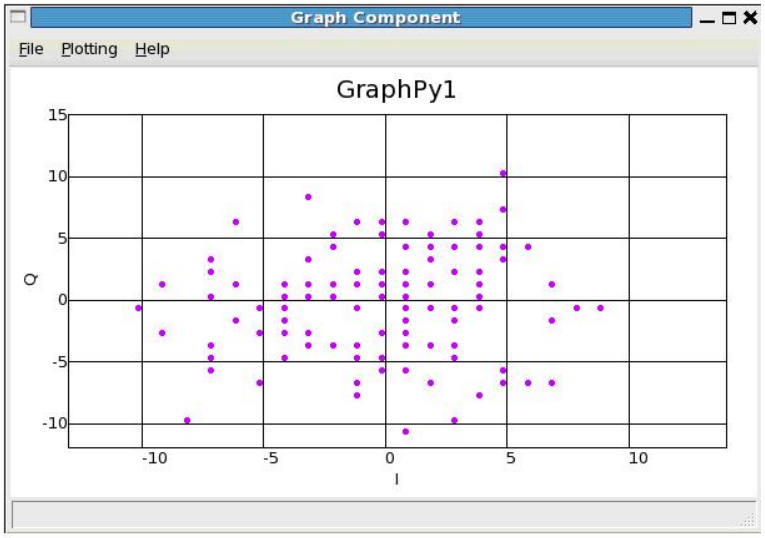

Figura 5.11: Diagrama de Constelação

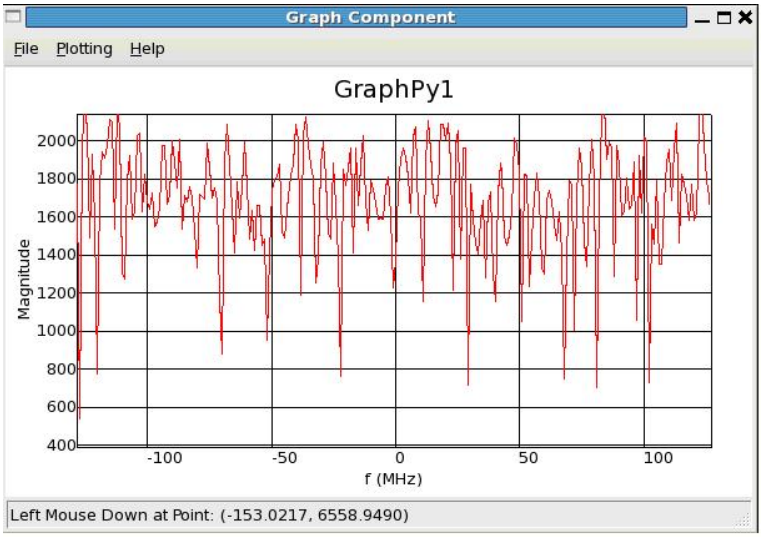

Figura 5.12: Espectro de Frequência

O resultado apresentado na Fig. 5.11 não é o que se esperava para uma modulação QPSK sob influência de um ruído AWGN. Isso ocorreu porque os valores do ganho I e ganho Q do amplificador já são pré-definidos pelo OWD. Esses valores são de 0db (1 linear) e, por serem pequenos, encontram-se tão próximos que se resumem, apenas, em uma nuvem de ruído. Faremos, então, um incremento para comparar com o resultado obtido e para verificar se conseguiremos alcançar o resultado esperado.

Primeiramente devemos parar e desinstalar, no terminal do WavLoader, a forma de onda obtida , reiniciar o OWD e abrir o projeto que foi salvo. Clicando com o botão direito do mouse em "amplifier1"e selecionando "Edit"a tela mostrada na Fig. 5.13 aparecerá. Clique no valor 1 dos ganhos e os modifique para, por exemplo, 32 (linear).

Deve-se fechar o OSSIE Component Editor e novamente gerar os arquivos XML, em Waveforms-genarate, como já foi feito anteriormente.

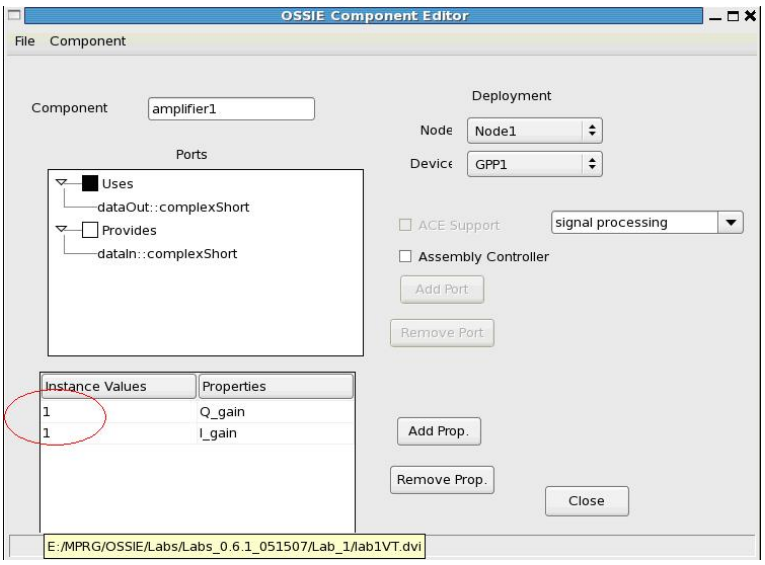

Figura 5.13: Editando os Valores dos Ganhos

Em /home/ossie/src/waveforms/lab1 é necessário repetir a instalação. E novamente usando-se o comando WavLoader, teremos os seguintes gráficos (Figs. 5.14 e 5.15).

Observando a Fig. 5.14 pode-se perceber que os símbolos individuais do canal de comunicação utilizado começam a se distinguir de uma nuvem única de ruído conforme o aumento do ganho I e Q do amplificador, obtendo-se assim o resultado esperado para constelação de um sistema QPSK transmitido através de um canal AWGN.

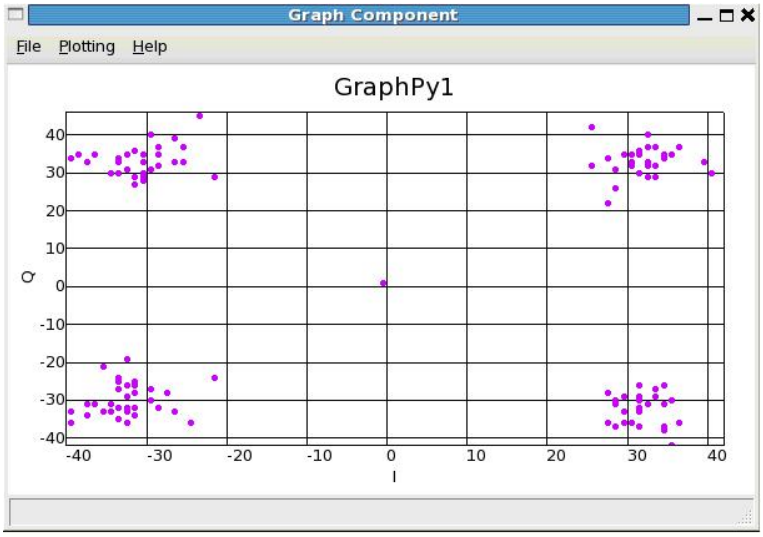

Figura 5.14: Novo Diagrama de Constelação

5.2.3.2 LABORATÓRIO 2

Esse laboratório tem como objetivo principal ensinar a construir uma nova component utilizando o OWD e adicionar a esta novas funcionalidades.

Primeiramente deve-se abrir o OWD através do diretório /home/ossie/WaveDev/wavedev, digitando-se

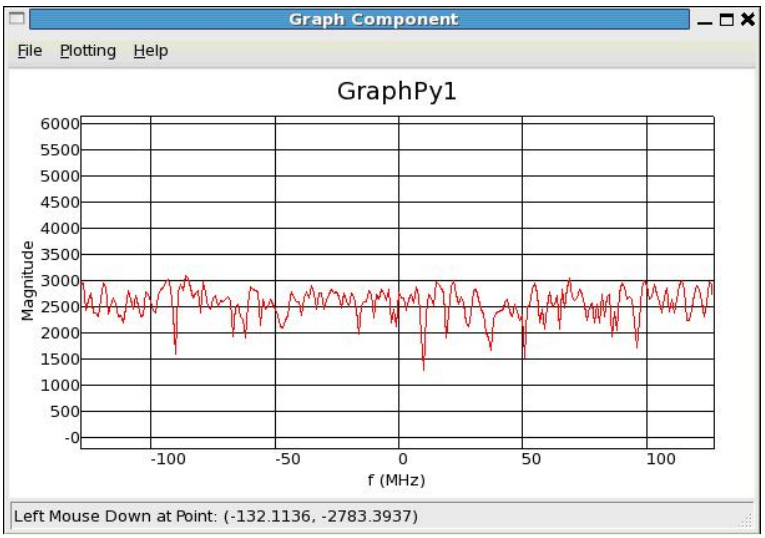

Figura 5.15: Novo Espectro de Frequencia

python wd.py. Nomeie sua forma de onda como lab2, por exemplo.

Começaremos esse laboratório criando uma nova componente no OWD. Para que isso seja possível, navegue no menu do OWD em "Waveform - Add Component"conforme mostrado na Fig. 5.16.

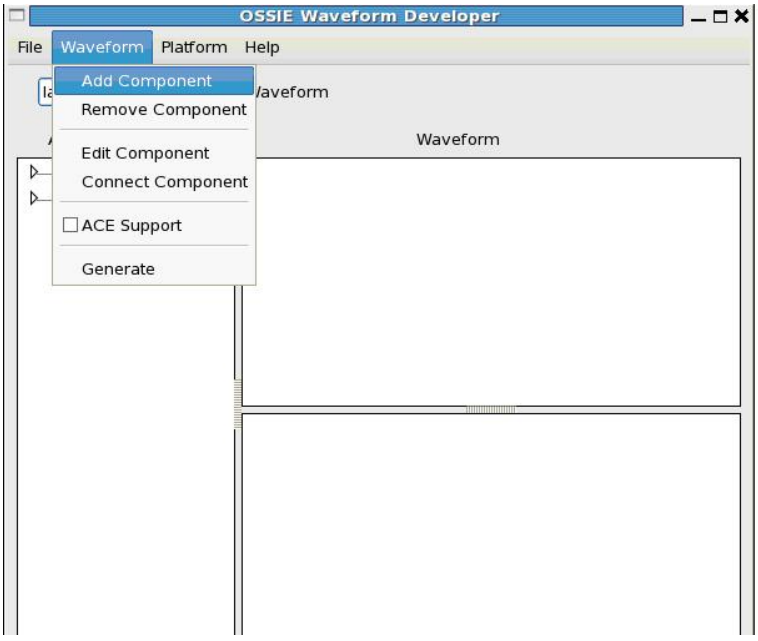

Figura 5.16: Adicionando Componentes

A janela do OSSIE Component Editor irá se abrir. Na caixa "Component", escreva o nome de sua nova componente: Amplifier_IQ. Defina, então, sua Uses Port clicando com o botão direito no quadrado preto "Uses"e selecionando "add". Clique no triângulo ao lado de "standardInterfaces"para expandir a lista de opções. Selecione opção "complexshort". Nomeie sua porta como "outPort"e certifique-se que sua porta está definida como "uses", Fig. 5.17.

Do Component Editor, precisaremos adicionar as propriedades do ganho I e do ganho Q. Clique no botão "add Prop"como mostra a Fig. 5.18.

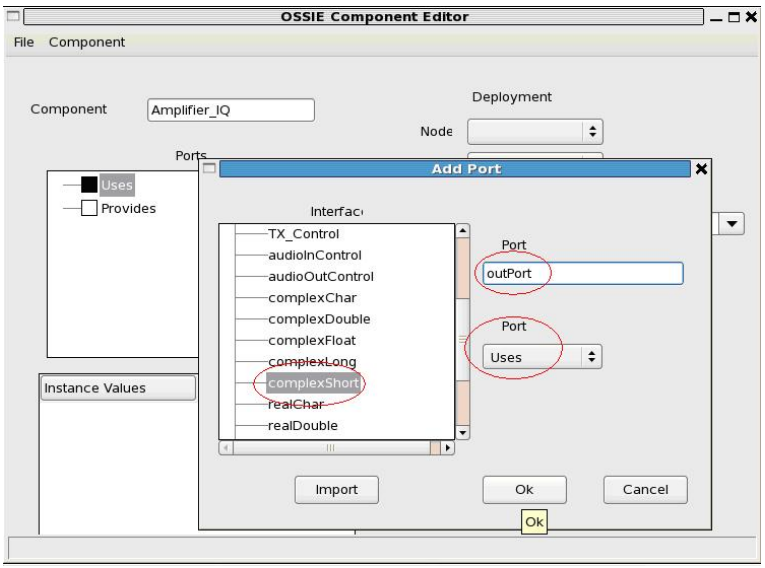

Figura 5.17: OSSIE Component Editor

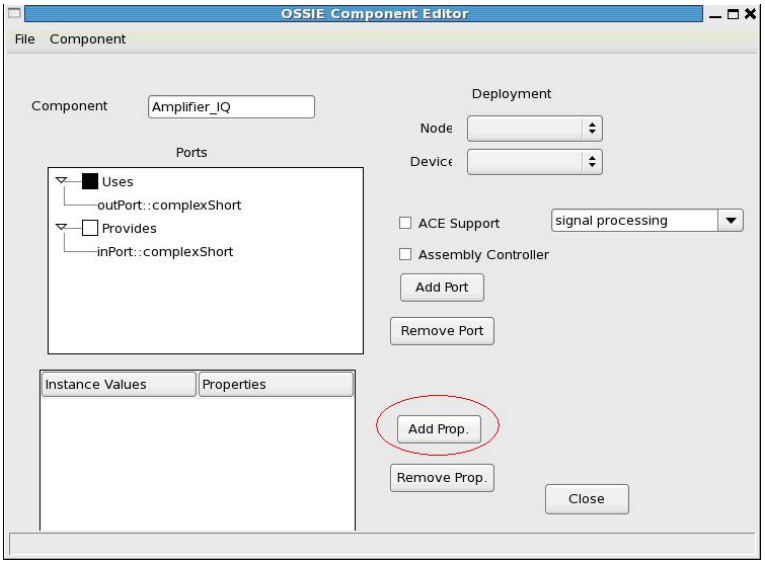

Figura 5.18: Adicionando Propriedades

Uma nova janela se abrirá. Nomeie sua propriedade como "I_gain"e embaixo de "value"selecione 1 e o "+". Em seguida aperte "Add Property"conforme ilustrado na Fig. 5.19.

Repita o procedimento para criar o "Q_gain".

No OSSIE Component Editor, gere sua nova componente navegando em "Component: Generate Component". Selecione o seguinte caminho de diretório: /home/ossie/src/component. Feche o OWD após salvar.

Navegue até o diretório onde gerou-se a nova componente e verifique a existência dos arquivos: amplifier_IQ.cpp e amplifier_IQ.h. Se abrirmos o amplifier IQ.h, verificaremos as seguintes linhas:

```
CORBA::Short simple_0_value;
CORBA::Short simple_1_value;
```

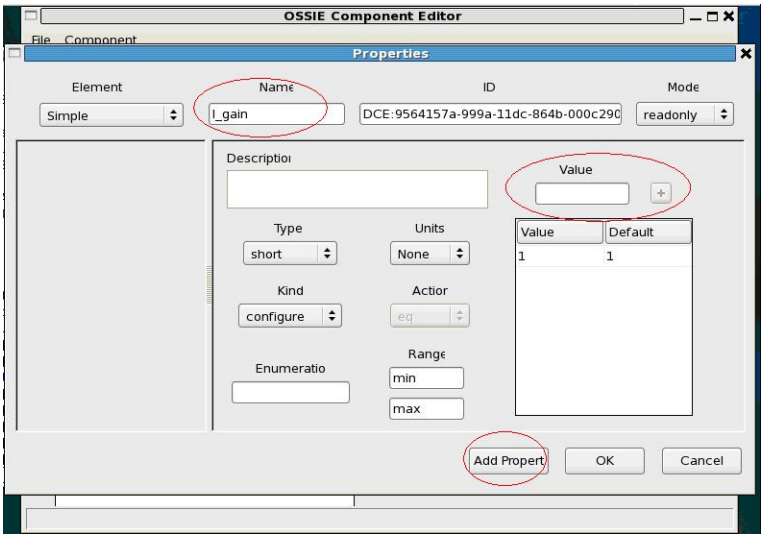

Figura 5.19: Janela Properties

Estas, representam a declaração das variáveis que conterão o valor das duas propriedades.

Entrando em amplifier_IQ.cpp, faremos algumas modificações a fim de adicionar funcionalidades ao Amplifier_IQ. Primeiramente, navegue até o void amplifier IQ i::configure. Pode-se notar dois "if"dentro de um loop "for". Estes são responsáveis por referenciar o valor da propriedade depositado no xml ao apropriado UUID (universally unique identifier). Agora, procure por void process data(void*data) e encontre a linha "amplifier_IQ i *Class_instance_name = (amplifier_IQ_i*) data"e mude "Class_instance_name"para "amplifier". Mude também "Class_instance_name"para "amplifier"nas próximas três linhas dentro do loop "while". Após a linha "amplifier_IQ_i*amplifier = (amplifier_IQi*) data"adicione a linha:

```
short I_gain, Q_gain;
```

Esta linha é a declaração das variáveis de ganho.

Edite a linha :

```
I_out_0.length();
Q_out_0.length();
```

Para que se torne:

```
I_out_0.length(I_in_0_length);
Q_out_0.length(Q_in_0_length);
```

Encontre também o comentário: /*insert code here to do work*/ e cole depois deste o seguinte código:

```
I_gain = (short) amplifier->simple_0_value; //get the properties
Q_gain = (short) amplifier->simple_1_value;
for (unsigned int i = 0; i < I_in_0_length; ++i)
(I_out_0)[i] = (*I_in_0)[i] * I_gain;
(Q_out_0)[i] = (*Q_in_0)[i] * Q_gain;
//close the for loop
```

Salve e feche o editor. Deve-se instalar a nova componente no OWD. Para isso, navegue para o seguinte diretório:

```
cd /home/ossie/src/component/amplifier IQ
```

E execute os seguintes comandos:

```
./reconf
./configure
su -c "make install"
```

Agora, similarmente ao primeiro laboratório, construiremos uma waveform só que utilizaremos a nova componente criada, ao invés do amplifier. Tendo aberto a janela principal do OWD, devemos nomear a nossa waveform (ou forma de onda) na caixa a isso destinada. Chamaremos nossa forma de onda de lab2 e para a salvarmos devemos utilizar o menu principal do OWD - "File -> Save Project As...".

Devemos agora adicionar um node ao nosso projeto. Este representará o computador onde as components rodarão. No menu principal do OWD navegue em Platform e em seguida selecione Add Deployment Node. Nomeie seu node como "Node1". Ao clicarmos no triângulo ao lado de Devices, expandiremos a lista que mostrará todos que são disponibilizados, assim, devemos adicionar um Device ao Node1 criado anteriormente. Para isso, com o botão direito do mouse em GPP, selecionaremos a opção "Add to node"e nomearemos como GPP1.

O próximo passo é adicionar as waveforms necessárias. Utilizaremos o QPSKmod, amplifier_IQ, AWGNChannel_complexShort e o Graph. Devemos então expandir a lista de components e para cada uma dessas citadas, clicar com o botão direito e selecionar "Add to Waveform". Conforme as componentes vão sendo adicionadas, estas aparecerão na parte direita do OWD.

Podemos associar cada uma das componentes a um node e device. Para tal, deve-se clicar duas vezes em cima da waveform QPSKmod1 e com isso abrirá uma nova janela denominada OSSIE Component Editor. Abaixo de Deployment, devemos selecionar Node1 para node e GPP1 para Device. Isso será repetido para as outras três componentes, porém, somente o QPSKmod1 será marcado como o controlador Assembly.

As conexões de cada uma das componentes estão mostradas na Fig. 5.20. Para que estas sejam feitas, clique com o botão direito em cada uma delas e selecione "connect".

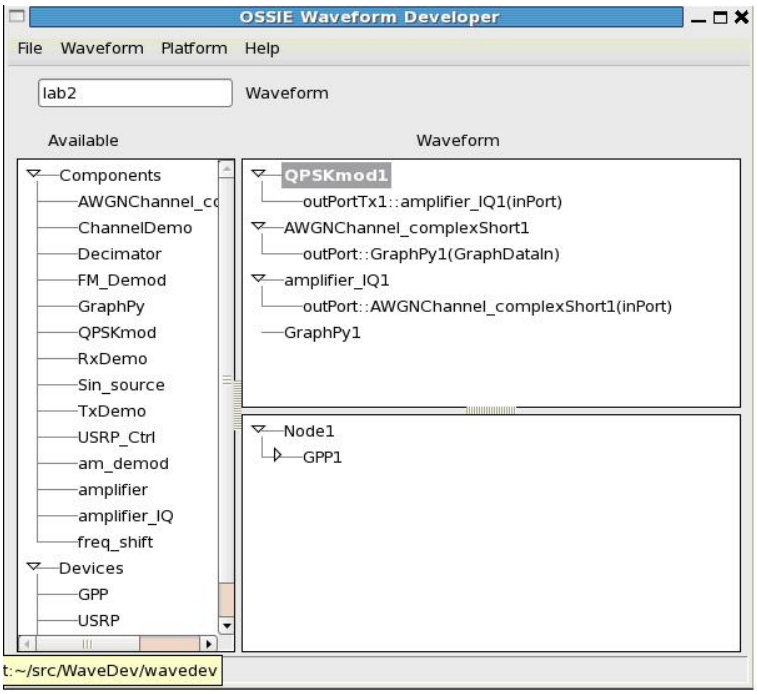

Figura 5.20: Conexões das Componentes

Neste ponto, devemos gerar os arquivos que definem uma waveform. Selecione "Waveform- genarate"no menu OWD. Na janela que se abrirá, apresentará o diretório /home/ossie/src/WaveDev/wavedev. Não deve

ser salvo aqui! E sim no /home/ossie/src/waveforms/<nome_da_sua_waveform>, neste caso, /home/ossie/src/waveforms/
Salve seu projeto novamente e feche o OWD.

Ao se navegar no diretório /home/ossie/src/waveforms pode-se observar um novo diretório chamado lab2. Este contém diversos arquivos XML. Entrando em lab2 deve-se instalar a nossa forma de onda utilizando os comandos abaixo:

```
./reconf
./configure
su -c "make install"
```

Entrando no diretório /home/sca/waveforms/lab2 perceberemos que foi salva uma cópia dos arquivos XML. Devemos rodar o "CORBA Naming Service"digitando como root omniNames.sh. A mensagem "checkpointing completed"deverá aparecer.

Em seqüência, devemos iniciar o Domain Management e registrar nosso Device rodando o programa nodeBooter digitando nodeBooter -D -d DeviceManager.dcd.xml. A mensagem "Device Registered"deverá aparecer. Em outro terminal do Linux, deve-se navegar até o diretório /home/sca/waveforms/lab2 e iniciar o wavLoader digitando wavLoader.py lab2_DAS.xml. Escolhendo a opção 1 e em seguida "s", será plotado os seguintes gráficos, referente ao ganho I/Q e ao espectro de freqüência para valores de ganho I e ganho Q iguais a 32 linear (Figs. 5.21 e 5.22, respectivamente).

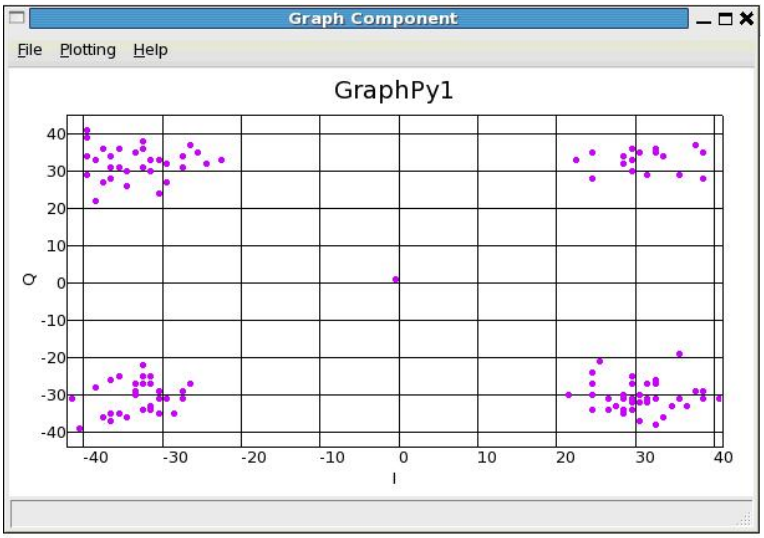

Figura 5.21: Diagrama de Constelação para Ganho I e Q iguais a 32

Para valores de ganho I e Q iguais a 1, temos o seguinte gráfico I/Q (Fig. 5.23). Como citado anteriormente, como os valores dos símbolos individuais do canal AWGN estão muito próximos, esses não são distinguidos formando uma nuvem de ruídos.

Selecionando valores diferentes para esses dois parâmetros, ganho I = 30 e ganho Q = 50, obtivemos o gráfico mostrado na Fig. 5.24. Desta forma conseguiu-se mais uma vez alcançar o resultado esperado para um sistema QPSK transmitido por um canal AWGN.

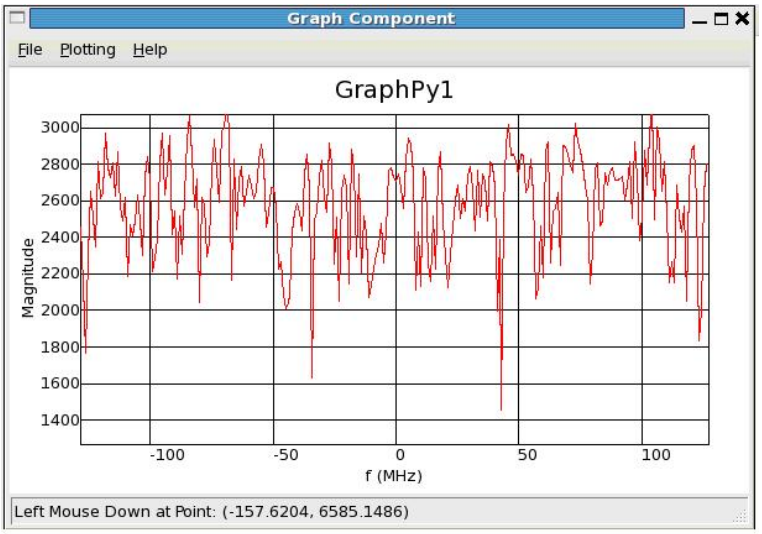

Figura 5.22: Espectro de Frequencia

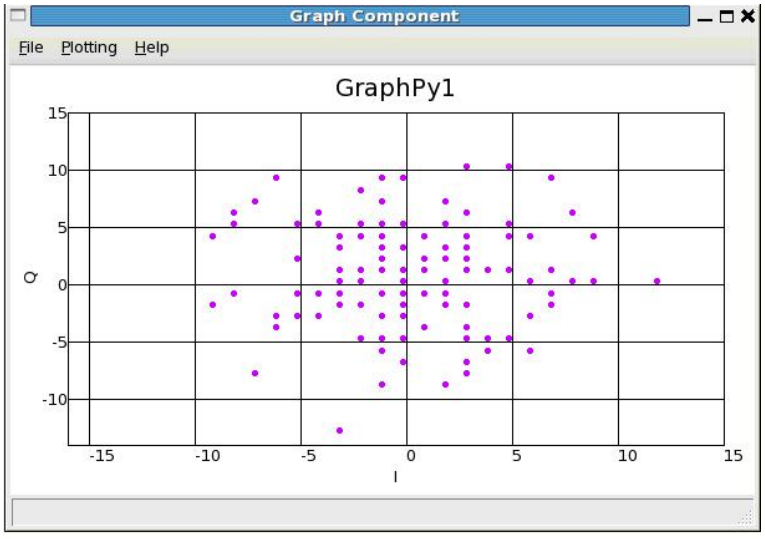

Figura 5.23: Diagrama de Constelação para Ganho I e Q iguais a 1

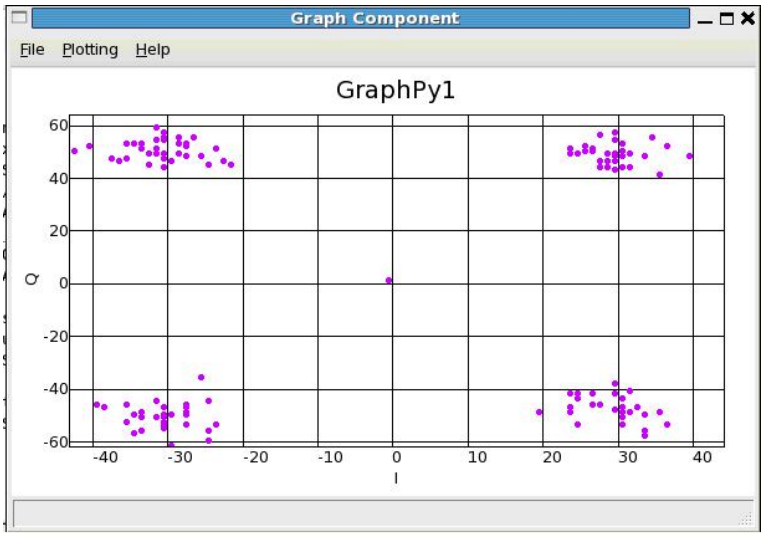

Figura 5.24: Diagrama de Constelação para Ganho I=30 e Q=50

6 CONCLUSÃO

Como foi explicado no capítulo 4, o produto final deste projeto de graduação não pôde ser implementado uma vez que o OSSIE ainda se apresenta como uma ferramenta instável. Sua estrutura de diretórios e seus arquivos XML foram modificados ocasionando diversos erros no OWD e na execução do comando nodeBooter. Esses erros se deram por incompatibilidade dos arquivos que estão disponíveis para serem baixados e as versões para os quais esses foram destinados.

O erro que ocorria ao abrir o OWD, quando este era instalado no Fedora nativo, foi contornado depois de muito estudo sobre a arquitetura SCA e os arquivos XML. Foram feitas diversas mudanças e correções na estrutura que o sítio do OSSIE fornece e, utilizou-se como base de comparação, a versão disponibilizada na VMWare. Com isso percebeu-se que quando a instalação está sendo efetuada, diversos arquivos são copiados para um diretório /sdr, porém, o OWD não consegue enxergar esses arquivos quando assim são dispostos, não apresentando as componentes necessárias para que os laboratórios sejam executados. Assim que este erro foi corrigido, tentou-se implementar uma forma de onda. O OWD funcionou sem que outro problema fosse acusado, porém, no ato da instalação da forma de onda criada, conforme explicado no capítulo 4, outra incompatibilidade de estrutura de diretório ocorreu. Os arquivos XML gerados deveriam ser copiados para o diretório /home/sca/waveforms para que, então, pudéssemos rodar o nodeBooter e o wavLoader. Não conseguimos reestruturá-los para que funcionasse corretamente, como foi feito para as componentes do OWD.

Em relação à VMWare obtivemos vários resultados interessantes. Um ponto relevante é que, apesar da estruturação disponibilizada na WMWare estar funcionando para a implementação dos dois laboratórios apresentados no capítulo 4, ou seja, ser aparentemente uma versão estável do OSSIE, o comando nodeBooter também apresentou erro quando submetido ao teste de instalação. Também não encontramos o motivo para tal erro. Desta forma, só foi possível a realização do laboratório porque, ao instalarmos a forma de onda, este gera uma cópia, para a estrutura correta, dos arquivos necessários para se rodar o nodeBooter.

A VMWare facilitou o entendimento do OSSIE como uma implementação da arquitetura SCA. Com esta ferramenta, implementamos uma forma de onda utilizando o sistema de modulação QPSK através de um canal AWGN. Foi possível também criar nossa própria componente, um amplificador I Q, e fazê-lo funcionar exatamente como o amplificador que o OWD nos fornece.

Os resultados experimentais foram realizados com sucesso, obtendo-se um diagrama de constelação de acordo com o que era esperado. A VMWare apresentou problemas quando resolvemos executar o terceiro laboratório, referente à utilização da placa USRP. A conexão desta ao computador se faz através de um cabo USB, e a Máquina Virtual, apresenta uma incompatibilidade com esta entrada, não reconhecendo, assim, a USRP como um hardware acoplado ao computador.

Uma vez que não foi possível cumprir com um dos objetivos específicos do projeto, tornou-se difícil embarcar o sistema, pois, como o OSSIE não funcionou de forma correta no computador com uso do Fedora 5, dificilmente funcionaria em uma versão mais leve do Linux a ser embarcado.

Contudo, acredita-se que o OSSIE está sendo aprimorado. Durante nosso projeto, o sítio que o disponibiliza e apresenta todas as suas documentações, projetos e passos para instalação, foi modificado por 4 vezes, e, cada uma das novas estruturas propostas, foram implementadas e testadas sem sucesso. Estas se apresentaram de forma diferente da contida na VMWare, que foi a nossa única fonte confiável do funcionamento do programa.

Há algumas semanas atrás, as instruções de instalação foram retiradas do sítio do OSSIE. Ainda não se tem conhecimento se tal fato acarretará em nova modificação de estrutura, que é o que se deseja, ou se simplesmente foi retirada por não estar funcionando corretamente.

Visto que diversos estudos foram realizados visando à parte prática, este projeto se apresenta como uma referência teórica sobre RDS, SCA e sistemas embarcados, assim como apresenta uma ferramenta prática, o OWD, que ajuda a compreender melhor a arquitetura SCA e suas aplicações.

O nosso Projeto Final teve grande valia para estudos futuros, pois fomos pioneiras na utilização do OSSIE dentro da Universidade de Brasília e pretendemos, com isso, estimular sua utilização como ferramenta para implementação de projetos de Rádio Definido por Software.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] MITOLA, J. Software Radio Architecture. [S.l.]: Wiley, 2000.

[2] Site http://ieeexplore.ieee.org/

[3] Preparado por Joint Tactical Radio System (JTRS) Joint Program Office, Software Communication Architecture Specification Ű JTRS 5000 SCA V3.0, 2004

[4] Ferreira, Leandro Gabriel Bastos - AVALIAÇÃO DE ARQUITETURAS DE PROCESSAMENTO DIGITAL DE RF EM SOFTWARE PARA USO EM RDS, 2006.

[5] DePriest, Jacob A. A Practical Approach to Rapid Prototyping of SCA Waveforms, 2006.

[6] Schena, Rafael, DESENVOLVIMENTO DE UM DIGITAL DOWN CONVERTER (DDC) PARA UM PROTÓTIPO EMBARCADO DE RÁDIO DEFINIDO POR SOFTWARE, 2007

[7] Carlos R. Aguayo Gonzalez and Jeffrey H. Reed , The SCA:: Explained. Carlos R. Aguayo Gonzalez and Jeffrey H. Reed, Bradley Dept. of Electrical and Computer Engineering Virginia Tech Blacksburg, Virginia 24061.

[8] Isomäki, Petri. An Overview of Software Defined Radio Technologies. University of Turku, Department of Information Technology

[9] J. H. Reed, Software Radio: A modern Approach to Radio Engineering, Prentice Hall, 2002.

[10] LATHI, B. P. Modern Digital and Analog Communications Systems. [S.l.]: Oxford University Press, 1998.

[11] M. Robert, P. Balister e J. Reed, Implementation of OSSIE on OMAP, 2006

[12] Site http://ossie.wireless.vt.edu/trac

[13] TUTTLEBEE, W. Software Defined Radio: Enabling Technologies. [S.l.]: Wiley, 2002.