UMA ARQUITETURA PARA O DESENVOLVIMENTO DE APLICAÇÕES DE VISÃO COMPUTACIONAL E PROCESSAMENTO DIGITAL DE IMAGENS EM SISTEMAS EMBUTIDOS

ANTÔNIO CELSO CALDEIRA JÚNIOR
Orientador: Antônio Otávio Fernandes

UMA ARQUITETURA PARA O DESENVOLVIMENTO DE APLICAÇÕES DE VISÃO COMPUTACIONAL E PROCESSAMENTO DIGITAL DE IMAGENS EM SISTEMAS EMBUTIDOS

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Minas Gerais como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

Belo Horizonte
19 de março de 2009

Caldeira Júnior, Antônio Celso
UMA ARQUITETURA PARA O DESENVOLVIMENTO DE APLICAÇÕES DE VISÃO COMPUTACIONAL E PROCESSAMENTO DIGITAL DE IMAGENS EM SISTEMAS EMBUTIDOS / Antônio Celso Caldeira Júnior. — Belo Horizonte, 2009
xii, 63 f. : il. ; 29cm

Dissertação (mestrado) — Universidade Federal de Minas Gerais
Orientador: Antônio Otávio Fernandes

I. Título.

[Folha de Aprovação]

Quando a secretaria do Curso fornecer esta folha,
ela deve ser digitalizada e armazenada no disco em formato gráfico.

Se você estiver usando o `pdflatex`,
armazene o arquivo preferencialmente em formato PNG
(o formato JPEG é pior neste caso).

Se você estiver usando o `latex` (não o `pdflatex`),
terá que converter o arquivo gráfico para o formato EPS.

Em seguida, acrescente a opção `approval={`*nome do arquivo*`}`
ao comando `\ppgccufmg`.

Àquele que me faz sentir especial a cada dia, mesmo sem dizer uma palavra, meu pai.

Agradecimentos

Primeiramente, agradeço à minha mãe, meu pai e minha irmã. Obrigado pelo amor incondicional, pela compreensão em todos os momentos e pela simples presença em minha vida. Às minhas avós Aline e Maria, modelos de vida, alegria e carinho, impossível descrever em palavras. Aos meus tios, em especial Tia Lúcia, que sempre me tratou de forma deliciosamente parcial. Finalmente a meus primos Anso, Teco e Kadu e demais familiares agradeço o apoio e a alegria daqueles sempre me incentivaram.

Ao professor e orientador Antônio Otávio, que acreditou em mim logo que iniciei minha carreira acadêmica na graduação e no mestrado. Professores Claudionor e Mário Campos, foi minha honra trabalhar com vocês nas disciplinas de Arquitetura. Aos demais professores, agradeço o ensinamento e formação.

Aos meus amigos do DCC da Grad031 e Grad041, em especial Guru, Bernardo e Farrer, agradeço por inúmeros trabalhos e infindáveis discussões que tornaram a computacação cada vez mais interessante e divertida. Aos amigos da PUC em especial Rodrigo e Rabelo, o pouco tempo compartilhado em sala de aula é inversamente proporcional ao carinho que tenho por vocês. Aos grandes amigos do SEBRAE, em especial Amorim, Ferrara, Gui, Japonês, Jejé e Romeiro, sempre presentes para rir e chorar. Aos amigos muito mais que virtuais Palmito, Pri e Grieco. Finalmente agradeço à Silvana, cujas conversas se tornaram imprescindíveis em vários momentos de minha conturbada vida. Aos colegas da Jasper, LECOM e iVision, em especial Alessandro, Yann e Glauber obrigado pela oportunidade diariamente enriquecedora de trabalhar com vocês.

A CAPES pelo subsídio durante a elaboração do mestrado.

"It is only in the mysterious equations of love that any logic or reason can be found."

(John Nash, A Beautiful mind)

Resumo

Este trabalho descreve uma arquitetura de *hardware* e *software* que auxilia no desenvolvimento de aplicações de Visão Computacional e Processamento Digital de Imagens em Sistemas Embutidos, especificamente *Smart Cameras*. *Smart Cameras* são câmeras que, além de capturar imagens, é capaz de extrair informações específicas para a aplicação. Sistemas Embutidos são sistemas compostos de *hardware* e *software* projetados para realizar tarefas específicas combinando processamento e dispositivos de entrada e saída necessários às aplicações. Visão Computacional pode ser definida como a capacidade de reconstruir, automaticamente, o modelo matemático a partir de uma imagem ou mais imagens de uma dada cena. O modelo matemático ou de computação gráfica é capaz de prover informações sobre a cena anteriormente registrada apenas em imagens. Este é então o processo inverso à Computação Gráfica em que o sistema de computação, a partir de um modelo matemático, gera imagens digitais. A área de Processamento Digital de Imagens é, desta forma, a base para o desenvolvimento de qualquer aplicação de Visão Computacional, pois ela é responsável pela manipulação de imagens, que são o primeiro insumo de uma aplicação de Visão Computacional. A restrição de recursos inerentes aos Sistemas Embutidos aliada à complexidade de Visão Computacional dificulta a construção de aplicações inseridas nestes dois contextos. A solução, proposta neste trabalho, é construir uma arquitetura que auxilia o desenvolvedor, abstraindo detalhes do Sistema Embutido e dos algoritmos de Visão Computacional.

Palavras-chave: Visão Computacional, Smart-cameras, Arquitetura

Abstract

This paper describes an architecture of hardware and software that helps the development of applications of Computer Vision and Digital Image Processing in Embedded Systems, specifically Smart Cameras. Smart Cameras are cameras that, in addition to image capturing, is capable of extracting application-specific information. Embedded systems are systems composed of hardware and software designed to perform specific tasks combining processing and input/output devices needed for applications. Computer Vision can be defined as the ability to rebuild, automatically, the mathematical model from one or more images of a given scene. The mathematical model, or computer graphics, is capable of providing information about the scene previously registered only in images. This is the reverse process to Computer Graphics where the computing system, from a mathematical model, generates digital images. The area of digital image processing is thus the basis for the development of any application of Computer Vision, because it is responsible for the manipulation of images that are the first input of an application of Computer Vision. The restriction of resources inherent in the systems combined with the complexity of Embedded Computer Vision hinders the construction of applications embedded in these two contexts. The solution, proposed in this paper, is to build an architecture that helps the developer, leaving aside details of Embedded System and Computer Vision algorithms.

Keywords: Computer Vision, Smart-Cameras, Architecture

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

1.1 Objetivos

Este trabalho é parte de um projeto maior que inclui construir uma arquitetura de *software* e *hardware* que auxilie no desenvolvimento de aplicações de Visão Computacional e Processamento Digital de Imagens em Sistemas Embutidos. Este trabalho trata apenas da arquitetura de *software*.

A arquitetura, como um todo, foi projetada com o intuito de: promover a reutilização dos recursos básicos (código) para garantir uma qualidade do sistema desenvolvido e reduzir custos; isolar a plataforma de *hardware* da aplicação desenvolvida; garantir a propriedade intelectual e a segurança da plataforma de *hardware*; possibilitar o desenvolvimento e execução de aplicações completas e reais em sistemas heterogêneos; abstrair detalhes da plataforma de *hardware*; encapsular restrições e limitações do *hardware*; fornecer um modelo intuitivo de programação.

Este trabalho define e descreve a arquitetura projetada. Além disso, ele define e descreve a implementação do *software* para a tradução, geração, transmissão, controle e execução do programa desenvolvido pelo usuário final até o Sistema Embutido, parte integrante da arquitetura projetada. A implementação da camada abstração de *hardware* será apresentada por seu autor como parte de sua dissertação de mestrado, ela

complementa o presente trabalho (detalhado em de Sousa Carmo [2009]).

1.2 Motivação

A abrangência de conhecimentos necessários para construir aplicações de Visão Computacional gera restrições de custo e qualidade no desenvolvimento. Aliado a isto, as particularidades no projeto de Sistemas Embutidos e os detalhes específicos de cada plataforma de desenvolvimento agravam o problema, aumentando ainda mais a necessidade de conhecimentos específicos em áreas diversas da Ciência da Computação. Desta forma, a motivação deste trabalho é criar uma ferramenta para programação de um Sistema Embutido específico: uma *Smart Camera* (detalhado em Shi [2009]).

1.2.1 *Smart Cameras*

Smart Cameras é definida como um Sistema de Visão que além de capturar imagens é capaz de: extrair informações relevantes para a aplicação; gerar eventos baseados nas informações da imagem capturada; tomar decisões que são usadas em sistemas inteligentes e autonomos. Elas são sistemas fechados que encapsulam até a interface de comunicação, por exemplo Ethernet. Notadamente mais compactas que os Sistemas de Visão Computacional baseados em computadores pessoais.

1.2.2 Sistemas Embutidos

Sistemas embutidos são sistemas compostos de *hardware* e *software* projetados para realizar tarefas específicas combinando processamento e dispositivos de entrada e saída necessários às aplicações. Normalmente nestes sistemas existem restrições de recursos, como, por exemplo, memória principal, espaço em disco ou *flash* (utilizados para armazenamento do *firmware*, Sistema Operacional), dispositivos de entrada e saída (monitor, teclado). Desta forma, para que os Sistemas Embutidos atendam aos requisitos de desempenho é necessário que um projeto bem definido seja realizado, afim de

reduzir o custo e tamanho do produto sem deixar de atender as restrições impostas pela aplicação (detalhado em staff [1999]; Malek et al. [2006]; Schmidt [2002]; Hennessy e Patterson [2003]; Patterson e Hennessy [1998]).

O desenvolvimento de aplicações para dispositivos de sistemas embutidos vem se tornando cada vez mais complexo e multidisciplinar, uma vez que a necessidade de aplicações completas é crescente. Isto exige que o desenvolvimento de aplicações para Sistemas Embutidos consiga prover aplicações multi-funcionais e compatíveis, mas, ao mesmo tempo, de desenvolvimento simples e de baixo custo. Os fabricantes de dispositivos para Sistemas Embutidos provêm ambientes de desenvolvimento proprietários e específicos para cada plataforma. Entretanto, os ambientes de desenvolvimento não possuem padrão estabelecido, ficando a critério do fabricante definir as características e recursos fornecidos. A independência de plataforma é um aspecto importante e motivador deste trabalho, pois a arquitetura projetada abstrai detalhes inerentes e específicos de cada plataforma, possibilitando a exclusão do ambiente de desenvolvimento fornecido pelo fabricante do processo. Isto garante a reutilização de código independente da plataforma utilizada. Entretanto, é necessário que o dispositivo em questão seja capaz de executar a interface com o baixo nível, parte da arquitetura aqui projetada (detalhado em staff [1999]; Malek et al. [2006]; Schmidt [2002]).

A grande quantidade de processamento exigido por aplicações de Processamento Digital de Imagens (detalhado em Gonzalez [2002]; Niblack [1984]) e consequentemente pela área de Visão Computacional aliado à disponibilidade de processadores otimizados para este tipo de aplicação, é um motivo de escolha desta área do conhecimento neste trabalho. Foi utilizado o processador *Blackfin* (detalhado em Devices [2005]), uma plataforma de *hardware* otimizada para processamento digital de imagem e vídeo. Esta plataforma foi utilizada no desenvolvimento da arquitetura resultante deste trabalho e aqui descrita para auxiliar na criação de aplicações eficientes em Visão Computacional e em Processamento Digital de Imagens.

1.2.3 Visão Computacional e Processamento Digital de Imagens

Existem várias definições de Visão Computacional (detalhado em Trucco e Verri [1998]). Uma delas trata da capacidade de reconstruir, automaticamente, o modelo matemático a partir de uma imagem ou mais imagens de uma dada cena. O modelo matemático, ou de computação gráfica, é capaz de prover informações sobre a cena registrada apenas em imagem. Este é, então, o processo inverso à Computação Gráfica, em que o sistema de computação, a partir de um modelo matemático, gera imagens digitais (como descrito em Coatrieux [2005]). A área de Processamento Digital de Imagens é, desta forma, a base para o desenvolvimento de qualquer aplicação de Visão Computacional, pois ela é responsável pela manipulação de imagens, que são o primeiro insumo de uma aplicação de Visão Computacional. Ela pode ser definida como processamento de uma imagem digital, representada por uma função de duas dimensões, $f(x, y)$, onde x e y são coordenadas espaciais no plano e a amplitude de f em qualquer par ordenado (x, y) é chamado de *intensidade* da imagem no dado ponto. Uma imagem é chamada de digital quando os valores de f são discretos e finitos. Não existe consenso quanto às fronteiras entre Visão Computacional e Processamento Digital de Imagens, uma delimitação aceitável é considerar que Processamento Digital de Imagens trata de processos cujas entradas e as saídas são imagens digitais e de processos que extraem atributos a partir de imagens digitais (como descrito em Gonzalez [2002]).

A área de Visão Computacional é, por natureza, multidisciplinar. Ela utiliza-se de conceitos, técnicas e abordagens de outras disciplinas principalmente da área da computação, mas não limita-se a ela. Algumas destas disciplinas são: processamento de sinais, física, matemática, inteligência artificial, robótica, neurobiologia (visão biológica), dentre outras (detalhado em Trucco e Verri [1998]; Forsyth e Ponce [2002]).

Pela riqueza e abrangência da área de Visão Computacional, desenvolver sistemas de Visão Computacional pode exigir do programador maturidade em diferentes áreas do conhecimento. Independente da aplicação desenvolvida, fatalmente será necessário

capturar e processar uma imagem digital, pois o processo de Visão Computacional começa na aquisição da imagem da cena. O processo básico de aquisição e processamento da imagem através da aplicação de um filtro básico para reduzir ruído se torna constante no desenvolvimento de aplicações de Visão Computacional. Torna-se, pois, útil reutilizar métodos que façam este processamento sempre que for necessário utilizá-los.

Arcabouços de desenvolvimento de aplicações em Visão Computacional, como a biblioteca OpenCV (detalhado em Corporation [2007]), são amplamente difundidas e utilizadas para este fim. Estas bibliotecas, entretanto, exigem o conhecimento de suas funções e detalhes de sua implementação para gerar uma aplicação de alto desempenho. A curva de aprendizado pode não ser favorável e a programação não-trivial, incrementando, pois, a lista de pre-requisitos para o desenvolvimento de aplicações em Visão Computacional.

1.3 Organização do texto

O presente trabalho encontra-se organizado em 5 capítulos, distribuídos da seguinte forma. Este é o Capítulo 1 que contém a introdução e conceitos básicos relacionados ao tema do trabalho. O Capítulo 2 fornece uma discussão sobre trabalhos relacionados. O Capítulo 3 apresenta e detalha a arquitetura construída neste trabalho. O Capítulo 4 descreve os resultados obtidos e o estudo de caso, exemplificando a confecção de duas aplicações utilizando a arquitetura desenvolvida. O Capítulo 5 apresenta as conclusões e aponta trabalhos futuros.

Capítulo 2

Trabalhos Relacionados

Os trabalhos relacionados à presente dissertação são caracterizados por apresentarem um escopo maior que o definido aqui ou por serem específicos de uma outra área de aplicação, por exemplo, redes de sensores sem fio (detalhado em SensorNet [2009]). Desta forma, são apresentados abaixo alguns trabalhos relacionados com uma breve descrição de cada um.

2.1 WISDOM e YATOS

O WISDOM é parte de um trabalho desenvolvido junto ao extinto grupo de pesquisa SensorNet (detalhado em SensorNet [2009]) que consiste na criação de um sistema operacional (YATOS) e de um *middleware* (WISDOM) projetado especificamente para o contexto de Redes de Sensores sem Fio. Ele foi projetado no intuito de servir como plataforma de desenvolvimento para o grupo e, por isso, é um caso particular da utilização de *middleware* para facilitar o desenvolvimento de aplicações em sistemas embutidos (como descrito em Vieira [2004]).

O *middleware* projetado apresenta características multi-plataforma e multi-linguagem em baixo nível, isto é, pode ser utilizado para desenvolver aplicações em plataformas e linguagens heterogêneas, desde que configuradas apropriadamente. Também possui uma metodologia de programação gráfica e intuitiva, fazendo uso da lin-

guagem Java (como descrito em Deitel e Deitel [2001]; Horstmann e Cornell [1998]) para ser portável.

Este trabalho se assemelha ao descrito neste documento, entretanto o contexto dele são aplicações específicas de redes de sensores em contrapartida à Visão Computacional e Processamento Digital de Imagens em Sistemas Embutidos realizado neste trabalho.

2.2 μCLinux

O μCLinux (detalhado em μCLinux [2009]) é um sistema operacional baseado no kernel do Linux 2.0 (como descrito em Bovet e Cesati [2003]). Ele foi portado, inicialmente, para microcontroladores sem *Memory Management Unit* (MMU). As MMU's são dispositivos de hardware que traduzem endereços virtuais em endereços físicos.

O Projeto μCLinux, no entanto, cresceu e atualmente proporciona sistemas operacionais que utilizam os Kernels 2.0, 2.4 e 2.6 do Linux para diversas arquiteturas de *hardware* diferentes. Dentre elas está o Blackfin, que foi utilizado como primeira plataforma de *hardware* deste trabalho.

O μCLinux foi escolhido para ser utilizado como camada de abstração dos recursos e detalhes do Blackfin por ser semelhante ao sistema operacional Linux, gerenciar memória, gerenciar sistema de arquivos, implementar interfaces de rede em alto nível e ser multitarefa. Estas funcionalidades auxiliaram no desenvolvimento da interface *hardware* e *software*, cujo principal benefício é a abstração conseguida.

2.3 TinyOS

O TinyOS (detalhado em TinyOS [2007b]) é um sistema operacional baseado em eventos para redes de sensores sem fio. Ele foi desenvolvido utilizando a linguagem de programação nesC, como descrito em TinyOS [2007a]. O nesC é uma variação da linguagem C, detalhada em Kernighan [1988], otimizada para lidar com as limitações de memória das redes de sensores sem fio.

O TinyOS é composto por um escalonador de tarefas não-preemptivo que foi implementado através de uma fila FIFO (*first in first out*). Ele não possui gerenciamento de processos, memória virtual ou alocação dinâmica de memória.

O projeto do TinyOS foi uma parceria entre a Universidade da Califórnia e o Intel Research Lab, ambos em Berkeley. Muito embora ele seja amplamente utilizado, dado à sua simplicidade seu uso é usualmente restrito às redes de sensores sem fio e, portanto, muito específico.

2.4 Image Processing Library 98 - IPL

A biblioteca de processamento de imagens (IPL) (detalhado em Eriksen [2009]) é uma plataforma independente de manipulação de imagens em C/C++ (detalhado em Stroustrup [1997]). O seu objetivo é auxiliar na criação de novas técnicas de processamento bem como fornecer os métodos padrão para processar imagens. Ela é base da biblioteca OpenCV descrita na Seção 2.5.

2.5 Open Source Computer Vision Library - OpenCV

O Open Computer Vision Library (OpenCV) (detalhado em Corporation [2007]) é uma biblioteca produzida pela Intel e seu foco é na aplicação de Visão Computacional em tempo real. Esta biblioteca é otimizada para aproveitar as instruções de multimídia MMX (como descrito em Peleg e Weiser [1996]) contidas nos processadores Intel. Por este motivo, é utilizada usualmente em computadores de propósito geral.

Esta biblioteca é baseada na IPL, biblioteca descrita na Seção 2.4. Possui um amplo arsenal de funções variando da gerência da matriz de imagem na memória até a criação de uma interface gráfica (detalhado em Sharp et al. [2007]) para a aplicação.

A OpenCV foi utilizada como inspiração para a criação de uma pequena biblioteca utilizada neste trabalho, denominada μOpenCv, contendo um número reduzido de funções de processamento de imagens para validar a idéia de múltiplas APIs (*application programming inferface*).

2.6 Matlab

O Matlab (detalhado em Mathworks [2007]; Wikipedia [2007]) é uma ferramenta de modelagem, alto-nível e interativa utilizada amplamente em cursos de processamento digital de imagens por possibilitar a utilização simples dos procedimentos utilizados em disciplinas de matemática aplicada. Entretanto, o Matlab possui diversas limitações como a ausência de referências, sintaxe não-convencional e ambígua. Além disso, Matlab não foca em desempenho. Estes motivos aliados ao fato de ser código fechado dificultam a utilização deste recurso na ferramenta proposta. Outro aspecto importante é a necessidade de utilização em sistemas embutidos, onde as restrições de armazenamento de informações continuam presentes. Uma ferramenta completa como o Matlab não seria adequado à plataforma alvo deste trabalho.

2.7 ImageAnalysis - Analog Devices

A biblioteca ImageAnalysis da empresa Analog Devices, fabricante do Blackfin, fornece a implementação em *assembly* para o Blackfin das funções de processamento de imagem básicas suportadas por esta plataforma. Ela é otimizada para a utilização no Blackfin. A ImageAnalysis foi traduzida para a linguagem C e compilada para utilização do μCLinux (como descrito em Devices [2005]; Inc. [2009b]).

2.8 Outros trabalhos

Existem iniciativas da indústria para fornecer soluções de *middleware* para desenvolvimento de aplicações em sistemas embutidos (detalhado em Foster [2007]). Essas soluções tendem a ser genéricas e portanto não aproveitam recursos disponibilizados pelo *hardware* para melhorar o desempenho final da aplicação. Entretanto há poucas referências científicas sobre a construção destas ferramenta, este é mais um fator motivador para este trabalho.

Capítulo 3

Arquitetura Construída

A arquitetura construída reside entre o usuário e o sistema operacional do Sistema Embutido utilizado, Figura 3.1. A arquitetura possibilita a abstração de detalhes do Sistema Embutido utilizado bem como a confecção de aplicações em alto nível.

Figura 3.1. Localização da Arquitetura

A arquitetura foi desenhada para ser modular. Com este objetivo sua implementação foi dividida em dois módulos, ferramenta e middleware. O esquema presente na Figura 3.2 mostra a arquitetura desenvolvida segmentada em suas duas partes distintas.

A **ferramenta** faz a interface com o usuário (programador) de aplicações em Visão Computacional e Processamento Digital de Imagens. Suas principais funções são:

tradução e geração de código

provê ao usuário recursos para o desenvolvimento de aplicações, possibilitando

a confecção de aplicações em alto nível, transparente à plataforma de *hardware* utilizada.

transmissão e execução do programa

controle da transmissão do código gerado quando este será executado no processador escolhido e o controle da execução deste programa.

O **middleware** trabalha no baixo nível da arquitetura, isto é, faz a interface com o Sistema Operacional, μCLinux. Sua principal função é abstrair a camada de sistema operacional e, por conseguinte, detalhes de *hardware*.

O projeto da arquitetura como um todo e a implementação da ferramenta fazem parte desde trabalho, e estão descritas detalhadamente aqui. Entretanto a implementação do *middleware* é parte do trabalho de mestrado de Glauber Tadeu (detalhado em de Sousa Carmo [2009]), desta forma, foram incluídos no presente texto as características de projeto do *middleware*, detalhes de implementação ficaram a critério de seu autor. As decisões de projeto do middleware realizadas como parte desta dissertação estão detalhados na Seção 3.1.

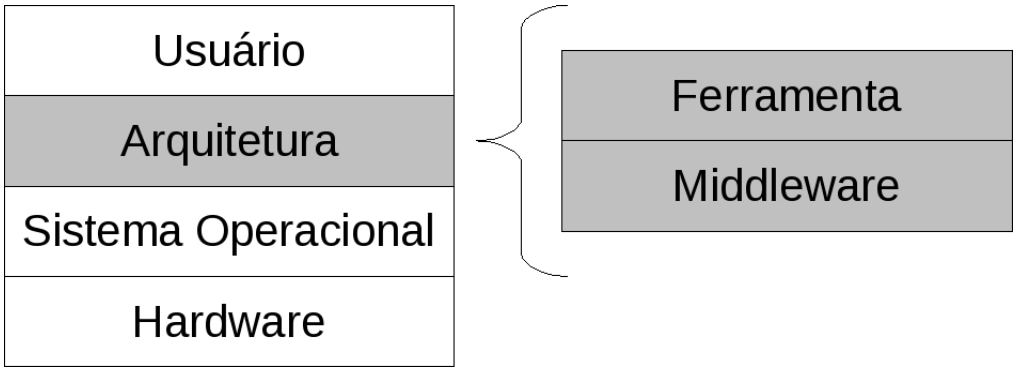

Figura 3.2. Composição da Arquitetura: Ferramenta e Middleware

Na Figura 3.3, é mostrado um diagrama detalhado da arquitetura projetada. Neste diagrama estão as interações presentes e os módulos responsáveis por cada uma delas. O usuário interage com a ferramenta, esta interação é realizada pelo módulo de Interface da Ferramenta. Este módulo tem por objetivo auxiliar o usuário na programação

da aplicação através da disponibilização dos recursos do hardware em alto nível bem como pela representação gráfica (árvore do XML construído) da aplicação. O módulo de interface interage com o módulo de processamento, cujas principais tarefas são de manter a estrutura da linguagem, gerar e executar o código XML a ser interpretado pelo middleware. O módulo de processamento interage com o módulo de comunicação para, através do protocolo detalhado na Seção 3.3.1 , fornecer ao *middleware* os recursos para a interpretação do XML. O *middleware*, por sua vez, possui um módulo de comunicação que implementa a parte do servidor do protocolo de comunicação; este módulo captura o XML e transfere o controle para o *parser* que processa o XML gerando uma lista de funções. A partir deste ponto não é mais utilizado o XML. O módulo de controle, então, utiliza as estruturas internas da lista de funções, APIs e gerência de arquivos para executar trechos da aplicação do usuário delegados pela aplicação. Por decisão de projeto, apenas a API Câmera tem comunicação direta com a câmera, simplificando o processo de adaptação da arquitetura a uma nova câmera.

Os detalhes de conceitos, características e implementação do *middleware* e da ferramenta estão detalhados nas subseções 3.1 e 3.2, respectivamente.

3.1 *Middleware*

Conforme (detalhado em Geihs [2001]), o *middleware* mascara a heterogeneidade de arquiteturas de computadores, sistemas operacionais, linguagens de programação, tecnologias de rede e facilita o desenvolvimento de aplicações. O *middleware* definido durante o presente trabalho mascara estas heterogeneidades e, por isso, é aliado à ferramenta apresentada na Seção 3.2 resultando na arquitetura projetada.

3.1.1 Definição da arquitetura

O principal objetivo do *middleware* é facilitar o desenvolvimento de aplicações para Sistemas Embutidos através da independência de plataforma. Isto é feito através da

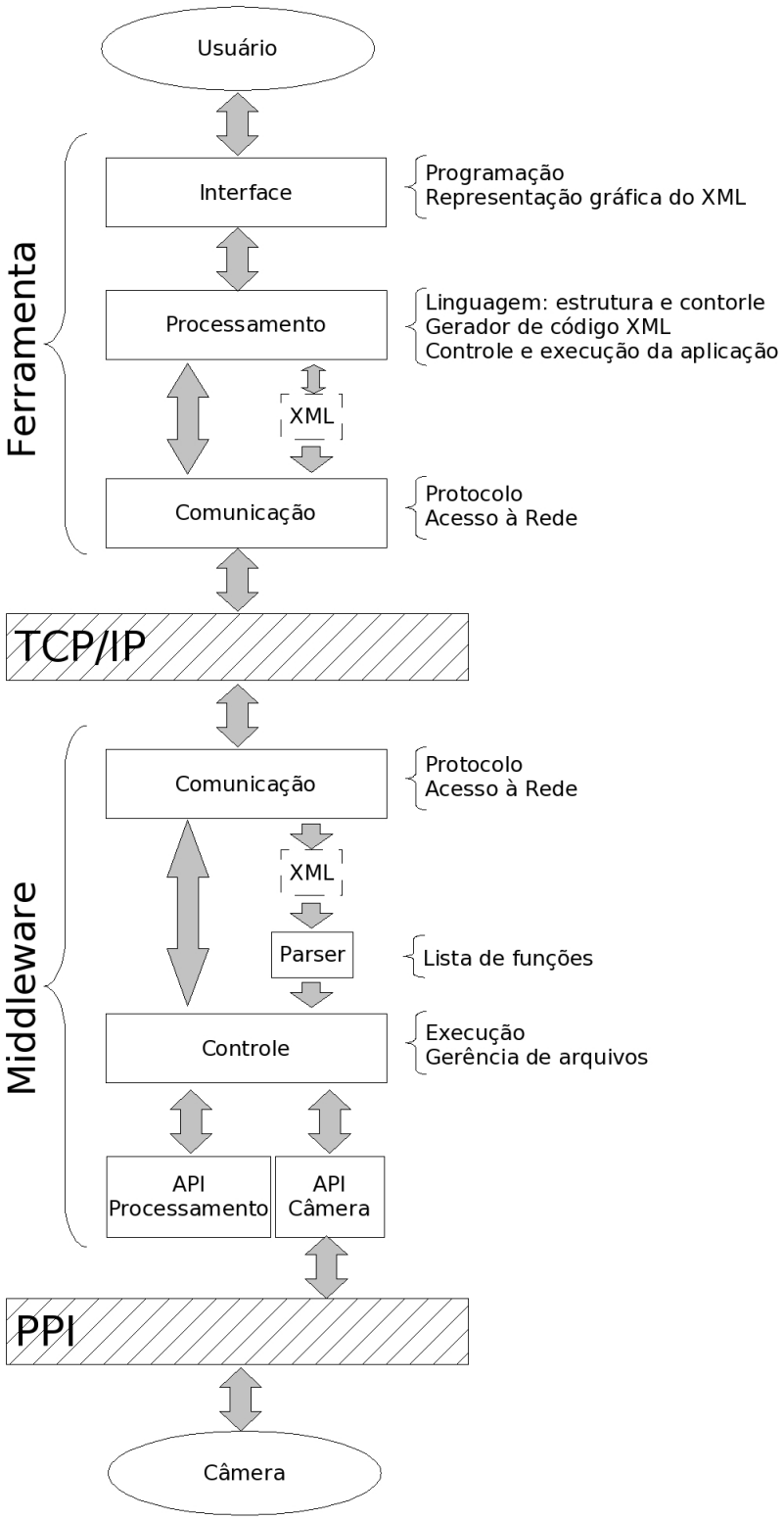

Figura 3.3. Arquitetura construída detalhada

abstração de detalhes do *hardware* possibilitados pela camada de Sistema Operacional incluída. A Figura 3.4 localiza o *middleware* perante o Sistema Embutido e a sua interface *hardware/software*; o *middleware* atua na interface entre o sistema operacional e a ferramenta, abstraindo os detalhes das camadas de baixo nível (sistema operacional e, por consequência, *hardware*) para as camadas de mais alto nível (ferramenta). A instância escolhida por este trabalho é definir um *middleware* que faça a interface especificamente para produzir aplicações de Processamento Digital de Imagens e Visão Computacional, desenvolvidas pela ferramenta e as arquiteturas computacionais compatíveis. Entretanto, o conceito utilizado não se limita a estas aplicações. A arquitetura implementada pode ser utilizada para o desenvolvimento de aplicações de propósito geral. De qualquer forma, utilizar plataformas de *hardware* otimizadas para tal é recomendado.

Figura 3.4. Localização do middleware

A arquitetura desenhada para o *middleware* está definida na Figura 3.5. Cada plataforma está representada por um retângulo. No nível mais alto, está uma ferramenta qualquer que implemente o protocolo de comunicação com o middleware. A ferramenta utiliza a camada de abstração proporcionada pelo *middleware* para interagir com cada plataforma de *hardware* de maneira transparente. A utilização deste protocolo permite que o *middleware* seja utilizado por qualquer ferramenta capaz de implementá-lo, e vice-versa. No nível mais baixo, estão as plataformas de *hardware*

que implementam a API de comunicação com o Sistema Operacional. Para validar o conceito deste trabalho, foram utilizadas duas plataformas: o Blackfin e um computador pessoal. A utilização de uma API possibilita que o *middleware* se comunique com qualquer plataforma capaz de implementá-la. Esta característica permite a abstração e transparência da arquitetura de *hardware* provida pelo middleware. Outros sistemas estão representados na definição da arquitetura do middleware; basta implementar a API de comunicação com o *middleware* para portá-lo.

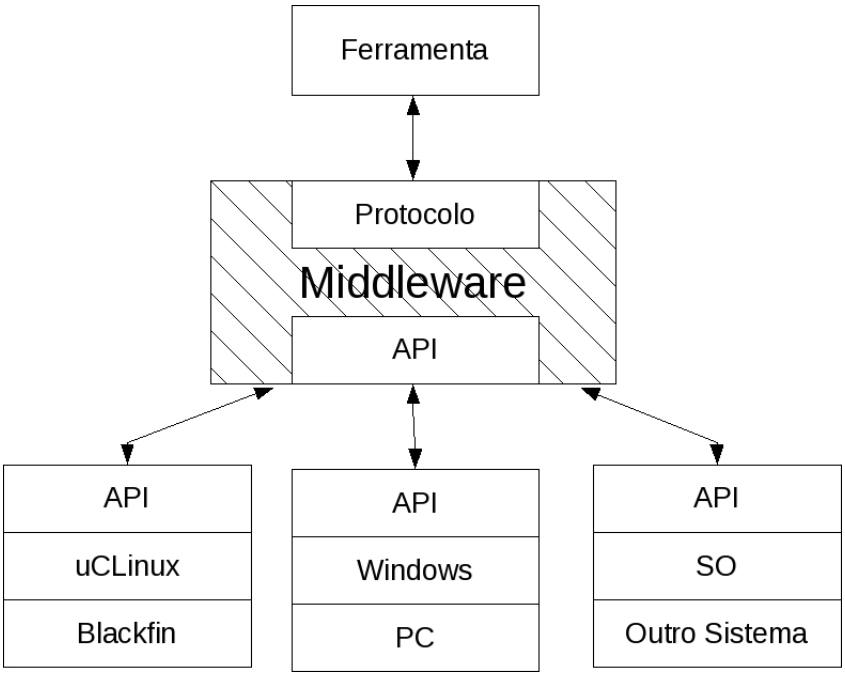

Figura 3.5. Definição da arquitetura do middleware

3.1.2 Características

O *middleware* desenvolvido consiste em uma camada de abstração entre a ferramenta e o sistema operacional provido pela arquitetura de hardware em questão. Esta camada facilita o desenvolvimento de aplicações de Processamento Digital de Imagens em Sistemas Embutidos pois a abstrai os detalhes das rotinas de configuração e operacação da plataforma em questão, permitindo que arquiteturas de hardware heterogêneas possam ser programadas de maneira semelhante. A partir do *middleware*, foi desenvolvida uma

ferramenta de desenvolvimento de aplicações de Visão Computacional e Processamento Digital de Imagens para Sistemas Embutidos, em especial, o Blackfin descrita na Seção 3.2. O *middleware* provê benefícios aos processos de desenvolvimento de software (detalhado em Sommerville [2006]; Pressman [2005]) através de:

Reutilização dos recursos básicos

reutilização de código é uma conhecida técnica de Engenharia de Software para garantir a qualidade um sistema

Isolamento

independência entre a arquitetura de hardware e a ferramenta

Segurança e garantia de propriedade intelectual

não é necessário que o usuário tenha acesso aos detalhes da plataforma de *hardware* para desenvolver suas aplicações.

Estas características são conseguidas através da capacidade do middleware, através do sistema operacional, abstrair heterogeneidades das arquiteturas de computadores, sistemas operacionais e linguagens de programação (detalhado em Geihs [2001]).

O *middleware* foi projetado para utilizar os recursos fornecidos pelo sistema operacional μCLinux (detalhado em μCLinux [2009]) em um primeiro momento. O μCLinux, por sua vez, está executando no processador Blackfin (como descrito em Devices [2005]). A escolha do BlackFin se deu por ele ser otimizado para processamento digital de imagens e vídeos e, portanto, se adequando ao escopo das aplicações referenciadas neste trabalho (Processamento Digital de Imagens e Visão Computacional). A escolha do μCLinux se deu por ser um sistema operacional de código aberto e semelhante ao sistema operacional Linux (como descrito em Bovet e Cesati [2003]; Silberschatz e Galvin [2000]) e, desta forma, flexível pois o código escrito para ele é portável para sistemas operacionais Linux padrão. Permitindo, ainda, a comparação de desempenho entre o sistema Linux padrão, executado em um computador e o Sistem Embutido.

3.2 Ferramenta

A ferramenta implementada neste trabalho tem por objetivo utilizar as funcionalidades do *middleware* para prover ao usuário um modelo de programação alto nível e transparente em relação ao *hardware*. É função da ferramenta gerenciar a tradução, geração, transmissão e controle da execução do programa definido pelo usuário.

A ferramenta construída gera código (XML) independente de linguagem e plataforma, desde que esta última implemente a API do middleware.

3.2.1 Definição da arquitetura

O principal objetivo de ferramenta é prover uma interface intuitiva e transparente para o usuário programar a aplicação de Visão Computacional e Processamento Digital de Imagens no Sistema Embutido escolhido. Além disso, a ferramenta é responsável pelo controle do fluxo de execução das aplicações desenvolvidas. Ou seja, a ferramenta determina quais trechos do código do usuário serão executadas na própria ferramenta e quais trechos serão executados no Sistema Embutido, por exemplo Blackfin. A Figura 3.6 localiza a ferramenta no Sistema Embutido e sua interface hardware/software; a ferramenta atua na interface entre o *middleware* e o usuário, proporcionando interface de programação intuitiva e transparente e controlando a execução do programa construído. Para este trabalho, foram implementadas funções da biblioteca ImageAnalysis e um subconjunto da biblioteca OpenCV, presentes no middleware.

A ferramenta foi desenvolvida com os seguintes requisitos em mente:

- Possibilitar a execução das aplicações de forma eficiente em diversos sistemas embutidos através da geração de código para cada compilador especificamente. As otimizações fornecidas pelos compiladores seriam assim aproveitadas.
- Possibilitar o desenvolvimento modularizado de aplicações.
- Abstrair o hardware. Permitindo que a arquitetura de hardware não seja exposta ao usuário.

Figura 3.6. Localização da ferramenta

- Encapsular restrições e limitações do hardware utilizado.
- Fornecer um modelo intuitivo de programação que abstraia detalhes do hardware utilizado para facilitar o desenvolvimento de aplicações.

A Figura 3.7 mostra um esquema de desenvolvimento sem *middleware* (esquerda) a arquitetura de *hardware* está exposta para o usuário que, além disso, o retrabalho para portar a aplicação em diversas arquiteturas existe. Enquanto que no esquema de desenvolvimento com a arquitetura projetada (direita) as plataformas de *hardware* são abstraídas (propriedade intelectual preservada) e a reutilização de código é explorada.

Sem a utilização da arquitetura projetada e implementada, o usuário tem conhecimento detalhado da arquitetura para qual a aplicação é desenvolvida, ele também terá o retrabalho de escrever a aplicação desejada em diferentes *toolkits*. Neste caso, para cada novo *hardware* utilizado, ele deve, inicialmente, reescrever a aplicação. Com a utilização da arquitetura proposta, o usuário utilizaria apenas a ferramenta proposta que geraria, transparentemente para o usuário, o código compatível com quaisquer *hardwares* suportados pela ferramenta. A ferramenta assegura a propriedade intelectual do fabricante do hardware, uma vez que a sua arquitetura não precisa ser exposta ao usuário para projetar aplicações.

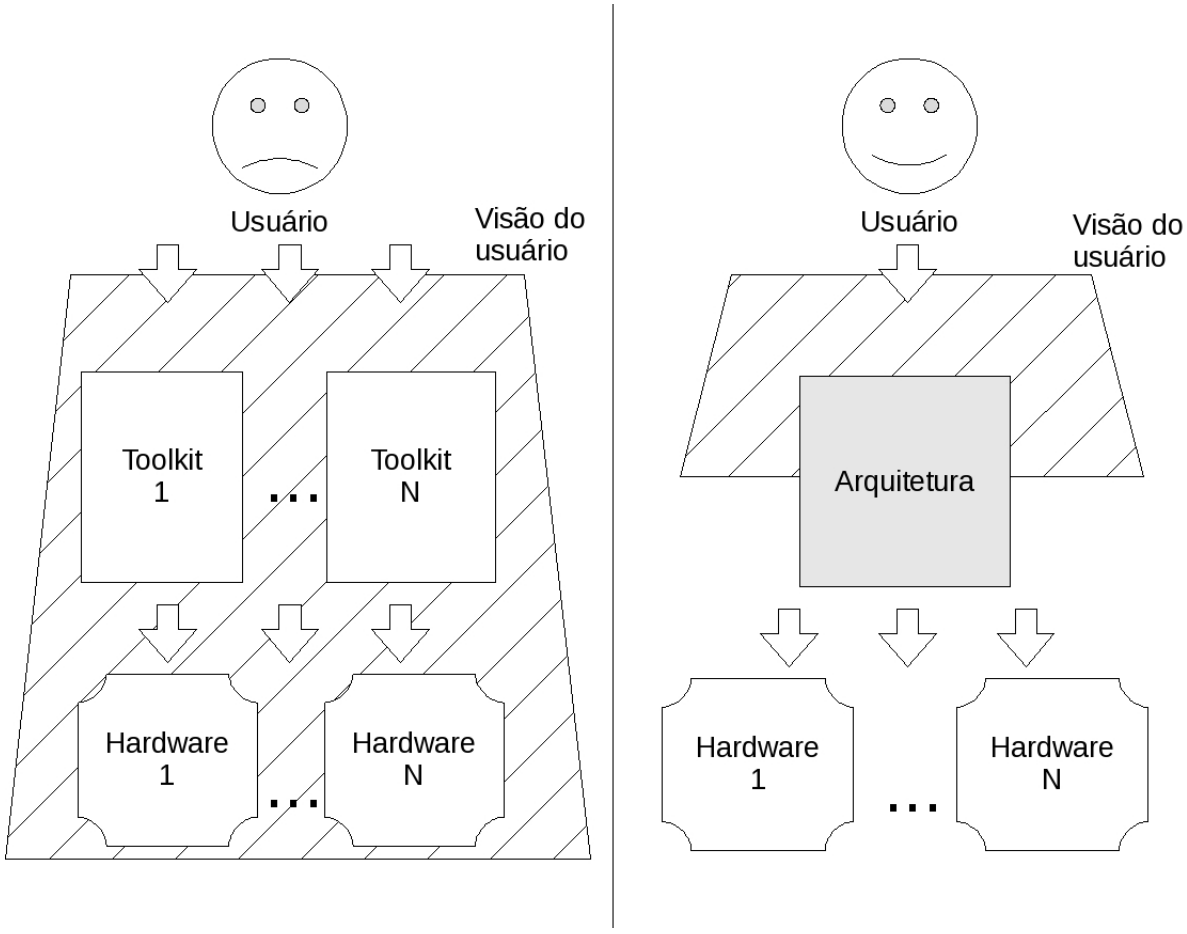

Figura 3.7. Esquema de desenvolvimento: comparação

3.2.2 Características

A ferramenta desenvolvida auxilia o desenvolvimento de aplicações de Visão Computacional e Processamento Digital de Imagens através da construção de uma árvore representativa das funções executadas pela aplicação. Esta árvore dá origem ao XML representativo da aplicação do usuário e é independente de plataforma de hardware, linguagem de programação e de API de processamento utilizada. Na Figura 3.8 está um exemplo de árvore do programa simples e linear, que apenas aplica o filtro de binarização (*threshold*) da API μOpenCv no *framebuffer*, gerado pela aplicação.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<blackfinProgram>
    <!--This is a Hello World program.-->
```

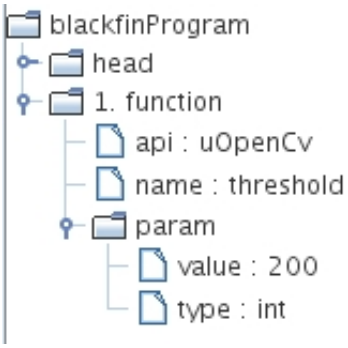

Figura 3.8. Construção de aplicações: Threshold

```
    <head>
        <name>Example</name>
        <author>Antonio</author>
        <version>1.0</version>
        <date>20081010</date>
    </head>
    <function>
        <api>u/gOpenCv</api>
        <name>threshold</name>
        <param>
            <value>200</value>
            <type>int</type>
        </param>
    </function>
</blackfinProgram>
```

No Código 3.2.2 está mostrado o XML correspondente à árvore mostrada na Figura 3.8. A contrapartida da árvore do programa é o XML. Ele define o fluxo de processamento realizado e determina nomes de APIs, funções, parâmetros e retornos. Desta forma, é preciso implementar, no sistema embutido, um servidor capaz de decodificar as informações contidas no XML e executar as operações necessárias. Além disso, ou-

tras informações, como o cabeçalho do XML, definem informações sobre o autor do programa, data e versão, através da etiqueta *head*.

Para o desenvolvimento da aplicação, o usuário escolhe a ordem das construções possibilitadas pela linguagem e as funções de APIs com seus respectivos parâmetros e os adiciona, um à um, à árvore de funções. Uma vez criada a árvore é gerado o XML, utilizado pela ferramenta para a execução do programa. O *middleware* é responsável pela execução apenas das APIs de processamento. A ferramenta é responsável pela execução das demais construções da linguagem, isto é, controle de fluxo e framebuffers. Na Figura A.1, está a janela de construção da aplicação e sua respectiva árvore de funções. Em construções mais complexas é possível interagir com os nodos da árvore estruturada da aplicação para detalhar cada construção, como constantes, parâmetros, expressões, etc. No Apêndice A, estão as telas construídas para a ferramenta.

É possível escolher entre duas APIs e processamento distintas, ou uma das construções da linguagem.

analog

ImageAnalysis da Analog Devices

- sobel
- erode
- dilate
- skeleton
- median
- perimeter

uopencv

subconjunto inspirado na biblioteca OpenCV da Intel

- threshold
- lineprofile

- negative
- histogram
- countWhite
- getRect
- floodfill
- conv

control

Construções da linguagem detalhadas na Seção 3.2.2.1.

Uma característica fundamental da ferramenta, e da arquitetura como um todo, foi a definição do modelo de *framebuffers* utilizado. A estrutura define um modelo de acumulador com lista de *framebuffers*, isto é, existe o *framebuffer* principal (acumulador). Todas as operações são executadas sobre este *framebuffer* e gravadas sobre ele. Aliada ao acumulador, existe uma lista de *framebuffers* que é utilizada para manipulação dos dados do acumulador para a memória. Por exemplo, caso o usuário queira salvar o *framebuffer* antes de uma determinada operação, basta copiar o acumulador para a lista através da operação de *push*. As operações de manipulação do acumulador e sua lista associada estão detalhadas na Seção 3.2.2.

3.2.2.1 Linguagem

Foi definida uma linguagem para o desenvolvimento de aplicações que é utilizada para compilar um XML. A linguagem foi definida e refinada a fim de atender aos requisitos mínimos de possibilitar a confecção de aplicações completas e reais. Desta forma, foram definidas as seguintes construções de propósito geral. A linguagem definida aqui é doravante denominada linguagem XML.

decl

Declaração de variáveis; reservar espaço na memória principal. Na implementação

adotada toda variável é um vetor. Variáveis são do tipo byte. Declarações devem ser escritas no início do programa. Elas não são contadas como funções do fluxo de execução (não pode-se pular para uma declaração).

assign

Assinalamento de variáveis: assinala um valor à variável previamente declarada

goto

Jump incondicional; pula para a posição definida no fluxo de execução da aplicação. Na implementação os goto's nada mais são que if que a condição é verdadeira sempre. O endereço é absoluto (não é possível pular para trás utilizando um endereço negativo).

if

Jump condicional; altera a qual será próxima construção no fluxo de execução se a expressão é verdadeira. As expressões utilizadas foram baseadas no padrão definido de C.

function

Função de API; executa uma função de uma das APIs definidas e disponibilizadas. Funções possuem parâmetros e retorno definidos pela linguagem, quando existentes. Esta construção é executada pelo middleware.

return

Retorno da aplicação; marca qual variável será retornada quando a operação halt for executada.

halt

Fim da aplicação; marca o final da aplicação.

Estas construções foram definidas tomando como base a linguagem C. Desta forma, é possível traçar um paralelo entre as duas linguagens. A Tabela 3.1 mostra as construções em C e a sua contrapartida na linguagem definida.

Além das construções de propósito geral, foram implementadas construções de propósito específico à arquitetura geral de câmeras digitais. Estas construções visam controlar o acesso ao framebuffer, sua profundidade, captura e armazenamento persistente. Assim, uma estrutura de pilha de framebuffers foi criada. As seguintes construções controlam a pilha:

capture

Captura a imagem do sensor para o framebuffer. Esta construção é executada pelo middleware.

push

Adiciona a imagem do framebuffer à pilha.

pop

Move a imagem da pilha ao framebuffer.

save

Salva a imagem do framebuffer em arquivo.

load

Carrega a imagem do arquivo no framebuffer.

Como estas funções são de propósito específico não há contrapartida na linguagem C. Na Tabela 3.2.

C	XML
5*int var;	<decl> <var>var</var> <size>1</size> <type>int</type> </decl>
6*var = 0;	<assign> <var>var</var> <position>0</position> <value>0</value> <type>int</type> </assign>
2*out: goto out;	<goto> <address>9</address> </goto>
3*if (var > 8192) goto out;	<if> <expression>var > 8192</expression> <goto>9</goto> </if>
13*var = function(128)	<function> <api>api</api> <name>function</name> <param> <value>128</value> <type>int</type> </param> <return> <var>var</var> <pos>0</pos> <type>int</type> </return> </function>
7*return var;	<return> <var>var</var> <pos>0</pos> <size>1</size> <type>int</type> </return> <halt/>

Tabela 3.1. Paralelo entre XML e C: construções de propósito geral

instrução	XML
4*capture	<capture> <height>512</height> <widht>512</widht> </capture>
3*push	<push> <pos>0</pos> </push>
3*pop	<pop> <pos>0</pos> </pop>
3*save	<save> <name>filename</name> </save>
3*load	<load> <name>filename</name> </load>

Tabela 3.2. Paralelo entre XML e C: construções de propósito específico

.

3.2.3 Implementação

A ferramenta construída é constituída de dois grandes módulos: **interface** 3.2.3.1 e **processamento** 3.2.3.2.

3.2.3.1 Interface

A **interface** é composta por quatro módulos: construtor de aplicações, visualizador de aplicações, configurador de conexão e gerenciador de conexão. Eles estão ilustrados na Figura 3.9. No restante desta seção estão detalhadas as implementações destes quatro módulos componentes da ferramenta.

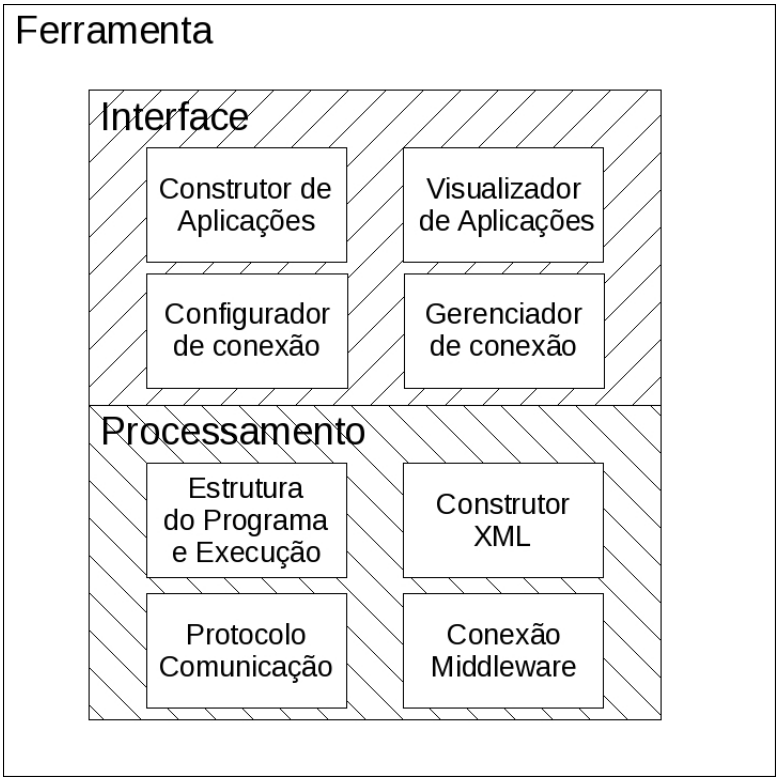

Figura 3.9. Descrição detalhada da ferramenta

O módulo construtor de aplicações tem por objetivo prover uma interface intuitiva, simples e de fácil aprendizado para que o usuário construa aplicações de Visão Computacional em Sistemas Embutidos. Desta forma, o construtor de aplicações cria a árvore de funções que determina a estrutura da aplicação criada. Isto é feito através da sele-

ção ordenada de funções a serem adicionadas à árvore. As funções são disponibilizadas por APIs que executam sobre o sistema operacional do sistema embutido escolhido. As APIs disponibilizadas para este trabalho estão descritas na Seção 3.2.2. Ele foi implementado utilizando-se a interface provida pelo *middleware* através da criação de documentos XML DOM.

O módulo visualizador de aplicações tem como saída uma árvore no modelo daquela mostrada na Figura 3.8. Ele tem por objetivo fornecer uma representação gráfica simples para o usuário da aplicação que está sendo confeccionada. Sua implementação consiste na criação da classe XmlTree que escreve a estrutura de um documento XML DOM na estrutura de dados JTree do Java, desenhando-a na tela.

Os módulos configurador e gerenciador de conexão configuram o acesso ao Sistema Embutido escolhido através de informações de IP e Porta para conexão. Eles são implementados utilizando-se a interface com o middleware.

A linguagem escolhida para implementação da ferramenta foi Java. Por prover portabilidade, possibilidade de descarregamento em página internet via *webstart*) e construção de GUI nativas. A ferramenta contém 6018 linhas de código java (como descrito em Horstmann e Cornell [1998]).

3.2.3.2 Processamento

O **processamento** é composto de basicamente quatro módulos: construtor XML, estrutura do programa e execução, conexão à plataforma e protocolo comunicação ilustrados na Figura 3.9. No restante desta seção estão detalhadas as implementações e decisões de implementação referentes a estes três módulos componentes do processamento.

O módulo construtor XML tem por objetivo construir o documento XML. Para tal, foi utilizada a tecnologia Document Object Model (DOM); uma plataforma que permite acessar e alterar conteúdo, estrutura e estilo de documentos como por exemplo XML. Esta tecnologia foi escolhida pela facilidade e flexibilidade na criação e alteração do

XML através da definição da estrutura em árvore do documento. Outras tecnologias, como a Simple API for XML (SAX) e até mesmo escrever um parser XML próprio foram sondadas (como descrito em Horstmann e Cornell [2004]). A geração do XML não é parte crítica do desempenho da arquitetura construída, pois o XML é gerado apenas uma vez e carregado no DSP em um segundo momento. Assim, foi feita a opção pela tecnologia mais flexível, DOM.

O módulo controle de execução tem por objetivo interpretar o XML gerado. Foi implementada uma máquina de estados para a execuçao da aplicação que considera cada construção presente no XML como uma operação atômica. As operações são executadas uma a uma, ou no ambiente da ferramenta ou no ambiente do middleware. As operações de API de processamento (etiqueta *function* no XML) e a operação de captura de imagem da câmera (etiqueta *capture* no XML) são executadas no middleware. As demais operações são executadas na ferramenta. Na Figura 3.10, está presente um diagrama representando as responsabilidades na execução da aplicação do usuário. A ferramenta utiliza o XML para gerar as estruturas presentes no programa, como, por exemplo, a lista de variáveis declarada. É a ferramenta que possui o controle sobre a execução da aplicação, definindo quais funções serão executadas por ela própria e quais serão executadas pelo middleware. As construções de controle de fluxo de execução (*decl, assign, if, goto, return e halt*) e de manipulação de *framebuffers* (*push, pop, save e load*) são executadas pela ferramenta por serem de alto nível. As construções de utilização das APIs de processamento (*function*) e de captura são (*capture*) são executadas pelo middleware. Para a execução de construções pelo middleware, a ferramenta gera XMLs contendo apenas as informações necessárias ao *middleware* para executar a construção desejada. O processo de geração e transmissão destes XMLs para o *middleware* é efetuado no início da execução, antes de decodificar a primeira construção.

A implementação da execução se deu por uma máquina de estados simples, controlada pelo iterador *Program Counter*, **pc**, que determina qual a construção está sendo

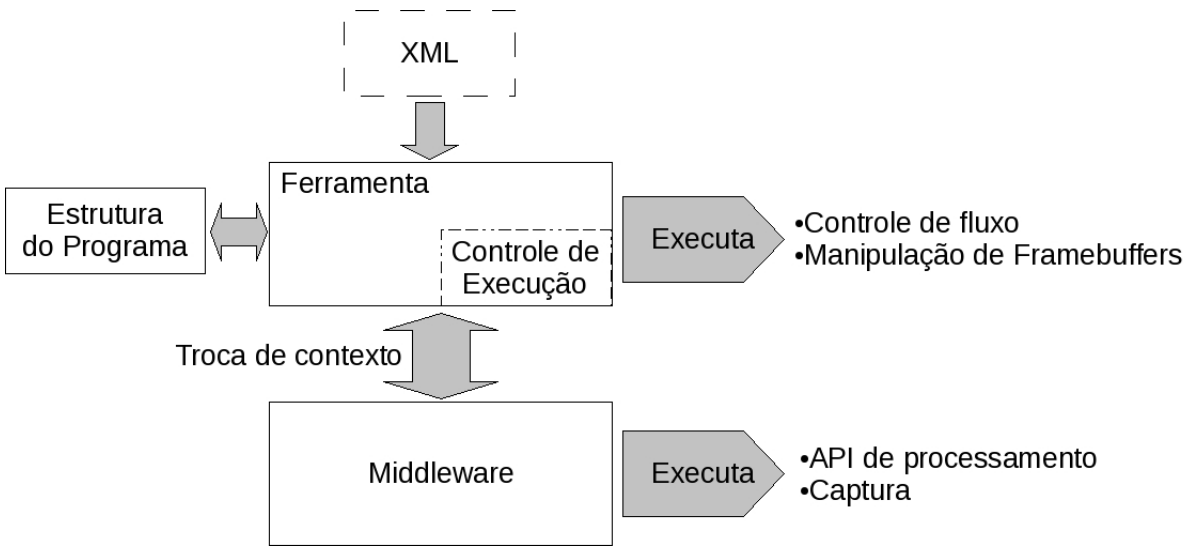

Figura 3.10. Papéis na execução da aplicação

executada. Para cada construção são realizados os seguintes passos.

Identificação

Identificação e análise da construção atual. Consiste na decodificação das etiquetas presentes no trecho do XML referente à construção atual e a preparação destas informações para a execução.

Execução

Execução da construção atual. Atualizam-se as estruturas do programa. Caso a instrução seja executada pelo middleware, é realizada a atualização do contexto da aplicação no *middleware* e na ferramenta. Esta atualização consiste na transferência do acumulador entre o *middleware* e a ferramenta, antes e depois da execução da construção.

Próxima instrução

Incremento do **pc**.

O módulo conexão plataforma tem por objetivo implementar a conexão e comunicação entre o *middleware* e a plataforma de hardware. O tipo de conexão adotado foi TCP/IP utilizando *Socket* (detalhado em Peterson e Davie [2007]). O fundamental

motivo para a escolha deste protocolo é o fato do Blackfin implementá-lo. Aliado a isso, a facilidade e flexibilidade que o protocolo TCP/IP através da internet provê à aplicação que utiliza o *middleware* descrito nesta parte do trabalho. É possível, desta forma, programar (e controlar) o dispositivo de hardware à distância, desde que ele esteja conectado à Internet e implemente o acesso para o middleware. O módulo de conexão à plataforma de hardware, assim, controla *buffers* e *timeouts* referentes à programação via *sockets*.

O módulo protocolo de comunicação tem por objetivo implementar as funções básicas providas pela API do sistema embutido. Estas funções vão desde recuperação da informação da versão do software até à interpretação e execução de um programa XML descarregado, passando pela transferência de arquivos entre a aplicação e a plataforma de hardware.

Java foi a linguagem de programação escolhida para implementação do middleware. Muito embora esta escolha de projeto possa afetar negativamente o seu desempenho, a interface com a ferramenta de exemplo construída foi facilitada e este é o maior motivo para esta decisão de projeto. Além disso o fator portabilidade que é inerente à linguagem Java também foi fator preponderante à adoção desta linguagem. Vale lembrar que a interface de programação não é focada em desempenho e sim o código por ela gerado, fazendo esta decisão aceitável. A implementação do módulo processamento contém 1817 linhas de código java.

3.3 Interface entre Ferramenta e Middleware

A interface entre a ferramenta implementada e o *middleware* projetado é realizada através de uma camada de rede. Foi utilizado o protocolo TCP/IP para estabelecer a conexão entre a ferramenta e o middleware. Toda comunicação é realizada através do protocolo de comunicação em nível de aplicação, ele está definido e detalhado na Seção 3.3.1.

3.3.1 Protocolo de comunicação

O protocolo implementado é baseado em ASCII (*American Standard Code for Information Interchange*) (detalhado em Inc. [2009a]). É utilizado o modelo cliente-servidor Silberschatz e Galvin [2000] em que a ferramenta é o cliente e o *middleware* o servidor. O caso de uso principal consiste em uma transação entre cliente e servidor. Uma transação consiste na troca de mensagens, em código ASCII entre cliente e servidor para completar uma operação. A lista de possíveis transações está listada abaixo. O protocolo completo está mostrado em detalhes na Tabela 3.3.

Resetar

Restaura o sistema para suas configurações padrão iniciais.

Transmitir arquivo

Transferência de arquivo do cliente para o servidor.

Atualizar versão

Transmissão da versão do servidor.

Atualizar imagem

Transmissão da imagem do acumulador para o cliente.

Carregar imagem

Carrega da imagem do cliente para o acumulador.

Capturar imagem

Captura imagem do sensor acoplado ao servidor para o acumulador.

Selecionar XML

Seleciona qual arquivo XML deve ser carregado no servidor.

Carregar XML

Carrega o programa XML a ser executado no servidor.

Executar XML

Executa o programa XML carregado no servidor.

Tabela 3.3: Protocolo de comunicação

Mensagem	Origem	Destino	Descrição
Resetar			
R	Cliente	Servidor	Comando de reset, restaura o sistema a suas configurações iniciais. Utilizado para sincronizar o inicio da comunicação.
A\n	Servidor	Cliente	Resposta ao comando de reset, informando que o servidor restaurou as configurações iniciais e está pronto para receber comandos. Caso o cliente não receba a resposta em um tempo determinado, esta deverá reenviar o comando de reset por mais duas vezes. Se não obtiver sucesso nas tentativas, informar ao usuário que não foi possível estabelecer conexão com o servidor.
Transmitir arquivo			
F	Cliente	Servidor	Comando informando que o cliente deseja transmitir um arquivo.
FA\n	Servidor	Cliente	Servidor responde ao cliente confirmando o recebimento do comando e fica aguardando tamanho do arquivo.

Continua na próxima página

Tabela 3.3 – continuado da página anterior

Mensagem	**Origem**	**Destino**	**Descrição**
TTTT TTTT	Cliente	Servidor	Cliente envia o tamanho do arquivo, em bytes, para o servidor. O tamanho máximo deve ser de 99999999 bytes uma vez que cada algarismo é representado por um ASCII.
TC\n	Servidor	Cliente	Servidor responde confirmando a recepção do tamanho. O último byte transmitido corresponde ao último algarismo que compõe o tamanho, um meio primitivo de controle de erro via eco.
NNNN NNNN. NNN	Cliente	Servidor	Cliente envia o nome com o qual o arquivo deverá ser salvo no servidor. O nome poderá ser composto de 1 a 8 caracteres (alfanuméricos, _ e -), com uma extensão de até 3 caracteres. O limite entre o nome e a extensão é marcado com um . (ponto). Não é obrigatório a existência da extensão do arquivo.
N\n	Servidor	Cliente	Servidor informa que recebeu o nome corretamente e já está pronto para receber os dados.
DDDD (...)	Cliente	Servidor	Recebendo o comando de entendimento do nome, os dados serão enviados pelo cliente, byte a byte.

Continua na próxima página

Tabela 3.3 – continuado da página anterior

Mensagem	**Origem**	**Destino**	**Descrição**
FOK\n	Servidor	Cliente	O servidor confirma o recebimento de todos os dados e encerra a transmissão de arquivos. Este controle é feito através do tamanho do arquivo previamente enviado.

Atualizar versão

V	Cliente	Servidor	Requisição da versão do servidor.
VVV DDMM AAAA	Servidor	Cliente	Transmite uma *string* com a versão e a data de compilação do servidor em execução. Cada número da versão indica a versão de uma parte específica do sistema.

Atualizar imagem

I	Cliente	Servidor	Requisição do envio da imagem do acumulador.
IA\nC\n TTTT\n TTTT\n	Servidor	Cliente	Responde ao comando I com o envio da imagem corrente. IA indica que o comando foi aceito. O parâmetro C define se a imagem é colorida (C) ou P&B (P), os quatro próximos bytes é o número de linhas, em ASCII e os quatro seguintes o número de colunas.
D	Cliente	Servidor	Cliente confirma para o servidor que recebeu o tamanho da imagem, autorizando o envio dos dados.

Continua na próxima página

Tabela 3.3 – continuado da página anterior

Mensagem	**Origem**	**Destino**	**Descrição**
DDDD (...)\n	Servidor	Cliente	Tendo recebido a confirmação do cliente de que recebeu o tamanho da imagem, o servidor envia os dados que formam a imagem. Ver na sessão Arquivos de Imagem como eles devem ser interpretados, entre outros detalhes importantes.

Carregar imagem

A	Cliente	Servidor	Cliente informa ao servidor que deseja abrir uma imagem de um arquivo.
AA\n	Servidor	Cliente	Servidor informa que está pronto para receber o nome.
NNNN NNNN. NNN	Cliente	Servidor	Cliente envia o nome do arquivo que contém a imagem no servidor. Para validação de conceito apenas arquivos tipo .pgm. (detalhado em Poskanzer [2003]) são suportados.
NA\n	Servidor	Cliente	Servidor informa que o arquivo foi aberto com sucesso.
NE\n	Cliente	Servidor	Servidor informa que houve erro na abertura do arquivo.

Capturar imagem

C	Cliente	Servidor	Cliente informa ao servidor que uma imagem deve ser capturada do sensor.

Continua na próxima página

Tabela 3.3 – continuado da página anterior

Mensagem	**Origem**	**Destino**	**Descrição**
CA\n	Servidor	Cliente	Servidor informa ao cliente que está pronto para capturar.
T HHHH WWWW	Cliente	Servidor	Cliente informa ao servidor o tamanho da imagem que deverá ser salva no acumulador. 4 bytes para a altura (HHHH) , 4 bytes para a largura (WWWW) que é descrita em ASCII. O tamanho máximo da imagem é de 9999x9999.
TA\ou TE\	Servidor	Cliente	Servidor informa ao cliente que terminou de capturar a imagem. TA\no caso de captura sem erros. TE\no caso de capturar com erros.
Selecionar XML			
O	Cliente	Servidor	Aplicação solicita acesso a opções.
OA\n	Servidor	Cliente	Servidor confirma modo de opções.
X	Cliente	Servidor	Aplicação solicita opção do XML.
XA\n	Servidor	Cliente	Servidor confirma opção XML.
NNNN NNNN. NNN	Cliente	Servidor	Cliente envia o nome do arquivo XML a ser carregado no próximo comando de carga.
NA\n	Servidor	Cliente	Servidor confirma recebimento de nome do XML.
Carregar XML			
L	Cliente	Servidor	Cliente solicita a abertura do XML.

Continua na próxima página

Tabela 3.3 – continuado da página anterior

Mensagem	**Origem**	**Destino**	**Descrição**
LA\n	Servidor	Cliente	Servidor informa que o XML foi aberto e carregado com sucesso.
LE\n	Cliente	Servidor	Servidor informa que houve erro na abertura ou leitura do XML (erro com o arquivo ou de sintaxe).
Executar XML			
X	Cliente	Servidor	Aplicação solicita a execução das funções carregadas do XML.
TT\n DDDD (...)\nXA\n	Servidor	Cliente	Servidor informa o tamanho do retorno da última função executada e o valor (conteúdo) do retorno, seguido de XA, indicando que funções foram executadas com sucesso. Cada um destes campos é separado por \n. Caso não haja retorno, o tamanho será 00 seguido de dois \n.
00\n \nXE\n	Servidor	Cliente	Servidor informa o tamanho zero de retorno seguido de dois \n e XE, indicando que ocorreu um erro na execução.

Capítulo 4

Resultados

A arquitetura foi projetada para o desenvolvimento intuitivo de aplicações com geração de código eficiente para diversos Sistemas Embutidos que possam servir de suporte no projeto de aplicações em sistemas embutidos.

Pode-se citar algumas funcionalidades e características oferecidas pela arquitetura:

- Abstração de restrições dos detalhes da plataforma utilizada.
- Geração de código para execução em cada plataforma desejada de forma transparente para o usuário.
- Fornecimento de um modelo intuitivo de programação, idealmente, gráfica.

4.1 Estudo de Caso: Presença ou Ausência

Esta foi a primeira aplicação escrita utilizando-se a arquitetura de *software* e *hardware* desenvolvida neste trabalho. O objetivo deste programa é responder à pergunta: "Existe um objeto na imagem dada?".

Por motivos didáticos, o primeiro passo é definir o fluxo de execução da aplicação. Na Figura 4.1, está definido o fluxo de execução desta aplicação. Inicialmente, é preciso inicializar as variáveis e estruturas que serão utilizadas na aplicação. O segundo passo é binarizar a imagem. O terceiro passo é contar a quantidade de *pixels* brancos da

imagem. O quarto passo é decidir se a quantidade de *pixels* brancos é ou não suficiente para considerar que o objeto está presente na imagem. O quinto, e último passo, é retornar o resultado.

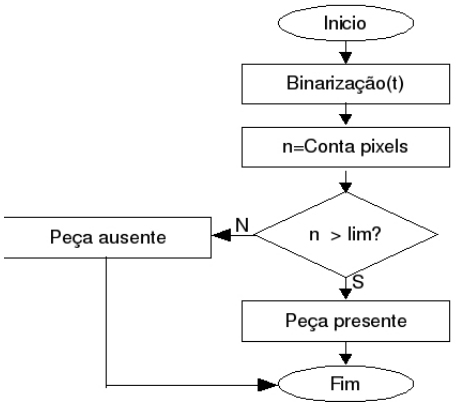

Figura 4.1. Fluxo de execução da aplicação de presença ou ausência

Para servir de controle, a aplicação descrita acima foi codificada na linguagem C e utilizando a arquitetura deste trabalho apresentada na Figura 4.2.

```
1  int presenca()
2  {
3          int numWhite = 0, ret;
4          threshold(128);
5          numWhite = countWhite();
6          if ( numWhite > 8192 )
7              ret = 1;
8          else
9              ret = 0;
10         return ret;
11 }
```

Pode-se perceber a semelhança de ambos códigos C e da Arquitetura com o fluxo de execução. Ambas as implementações acontecem de forma quase direta da codificação

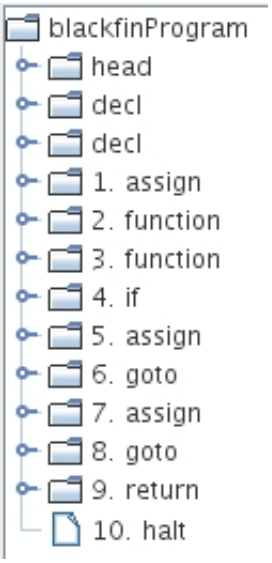

Figura 4.2. Presença ou Ausência na Arquitetura

Linha C	Elemento Arquitetura	Linha XML	Fluxo
3	0-1	10-22	Primeiro passo
4	2	23-30	Segundo passo
5	3	31-38	Terceiro passo
6-9	4-8	39-58	Quarto passo
10	9-10	59-65	Quinto passo

Tabela 4.1. Mapeamento entre abordagens: presença ou ausência

do fluxo de execução. A Tabela 4.1 do mapeamento entre o fluxo de execução, do código C e do código da Arquitetura.

Na Figura 4.2, estão omitidos os dados de cada marcação no XML. Entretanto, eles podem ser obtidos analisando-se o código XML gerado a partir da definição da aplicação da arquitetura. O código XML é mostrado no Apêndice B.

4.2 Estudo de Caso: Presença ou Ausência com controle de erro

Esta aplicação foi escolhida por mostrar as funcionalidades de propósito específico de controle de framebuffer implementadas na linguagem. Ela consiste na aplicação de

Presença ou Ausência adicionada de controle de erro, isto é, um objeto é considerado presente apenas se estiver entre um intervalo de contagem pre-definido. Ou seja, histerese (detalhado em Trucco e Verri [1998]).

Por motivos didáticos o primeiro passo é definir o fluxo de execução da aplicação. Na Figura 4.3, está definido o fluxo de execução desta aplicação. Inicialmente, é preciso inicializar as variáveis e estruturas que serão utilizadas na aplicação. O segundo passo é binarizar a imagem utilizando um limiar variável. O terceiro passo é contar a quantidade de *pixels* brancos da imagem. O quarto passo é decidir se a quantidade de *pixels* brancos é ou não suficiente para considerar que o objeto está presente na imagem, caso o julgamento seja indecisivo, volta-se ao inicio incrementando o limiar e tentando novamente. O quinto, e último passo, é retornar o resultado.

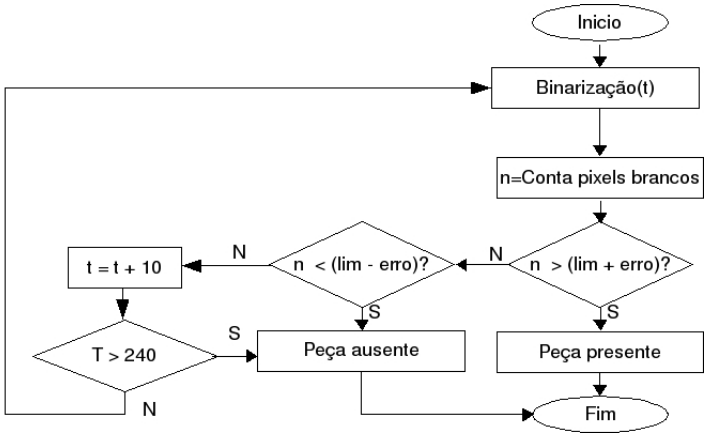

Figura 4.3. Fluxo de execução da aplicação de presença ou ausência com controle de erro

Para servir de controle, a aplicação descrita acima foi codificada na linguagem C e utilizando a arquitetura deste trabalho apresentada na Figura 4.4.

```
1 int presenca_erro()
2 {
3    int numWhite = 0, ret = 0, threshold;
4    push(0);
5    threshold=118;
6    while( threshold < 240 )
```

```
7    {
8       pop(0);
9       threshold = threshold + 10;
10      if ( threshold > 240 )
11        return 0; // ausente
12      push(0);
13      threshold(threshold);
14      numWhite = countWhite();
15      if ( numWhite > 8192 + 4096 )
16        return 1; // presente
17      else if ( numWhite > 8192 - 4096 )
18        ;
19      else
20        return 0;
21   }
22   return ret;
23 }
```

Ambas as implementações (código C e código da Arquitetura) refletem de forma quase direta da codificação do fluxo de execução. A Tabela 4.2 do mapeamento entre o fluxo de execução, do código C e do código da Arquitetura.

Na Figura 4.4, estão omitidos os dados de cada marcação no XML. Entretanto, eles podem ser obtidos analisando-se o código XML gerado a partir da definição da aplicação da arquitetura. O código XML é mostrado no Apêndice **??**.

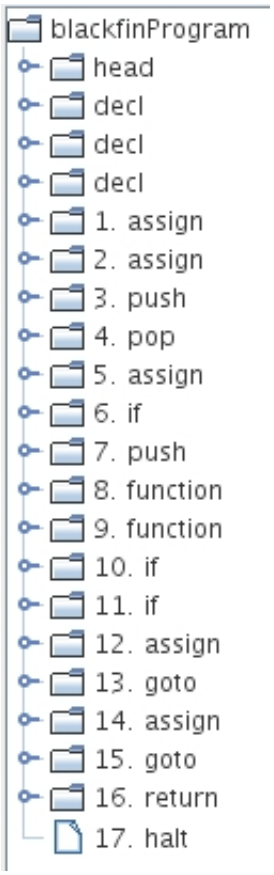

Figura 4.4. Presença ou Ausência com controle de erro na Arquitetura

Linha C	Elemento Arquitetura	Linha XML	Fluxo
3-5	0-3	10-37	Primeiro passo
6-13	4-8	38-60	Segundo passo
14	9	61-68	Terceiro passo
15-20	10-11	69-76	Quarto passo
21-22	12-17	77-98	Quinto passo

Tabela 4.2. Mapeamento entre abordagens: presença ou ausência com controle de erro

Capítulo 5

Conclusão

O presente trabalho projeta uma arquitetura que abrange as camadas de *software*, passando pela a interface humano-computador, até a camada baixo nível de *driver* de *hardware*. O trabalho também compreende a implementação das camadas de mais alto nível da arquitetura, consolidando as funcionalidades de interface com o usuário, tradução e geração de código.

A arquitetura foi projetada com o intuito de suportar aplicações de Visão Computacional e Processamento Digital de Imagens em Sistemas Embutidos. Entretanto, o resultado final foi uma arquitetura de propósito geral. Ela pode ser utilizada para o desenvolvimento de qualquer tipo de aplicação, desde que as APIs de processamento, implementadas no middleware. A arquitetura implementa uma linguagem de propósito geral, que não tem restrições conceituais ao utilizar sistemas regulares, não embutidos. Em particular para sistemas Linux, basta compilar o *middleware* para a arquitetura alvo em questão.

A arquitetura, como um todo, apresenta as características de: fornecer um modelo intuitivo de programação através da interface gráfica de programação implementada pela ferramenta; promover o reutilização dos recursos básicos (código) para garantir uma qualidade do sistema desenvolvido através da utilização de APIs de processamento; garantir a propriedade intelectual e a segurança da plataforma de *hardware* através da

inclusão do middleware; isolar a plataforma de *hardware* da aplicação desenvolvida através da adoção do Sistema Operacional μCLinux; possibilitar o desenvolvimento e execução de aplicações completas e reais em sistemas heterogêneos; abstrair detalhes da plataforma de *hardware*; encapsular restrições e limitações do *hardware* através da adoção de um Sistema Operacional.

Para trabalhos futuros na arquitetura, recomenda-se a melhora na implementação do protocolo de comunicação entre a ferramenta e o middleware, utilizando-se a construção de verdadeiros pacotes de TCP (*transmission control protocol*) ao invés de utilizar um protocolo baseado em ASCII, pois este, fatalmente, apresenta um *overhead* pois a informação, neste caso, é códificada utilizando-se apenas o |textitbyte ASCII. Outra possibilidade é automatizar a manipulação das bibliotecas do *middleware* pela ferramenta, garantindo ao usuário a habilidade de utilizar quaisquer APIs que lhe convenham sem ter conhecimento da arquitetura.

Especificamente sobre a ferramenta, parte de implementação descrita neste trabalho, as tarefas realizadas são quatro: interface com o usuário, tradução, geração de código e execução da aplicação. A totalidade destas tarefas não é o caminho crítico da arquitetura projetada. Uma vez desenvolvida a aplicação ela é descarregada através do *middleware* e executada de maneira isolada. Desta forma, a performance da ferramenta de desenvolvimento do programa não está ligada à performance da aplicação, uma vez que a ferramenta não está no contexto de otimização.

Para trabalhos futuros na ferramenta, recomenda-se analisar a qualidade do código gerado pela ferramenta no sistema embutido alvo. Como o código gerado pela ferramenta é interpretado, espera-se que o seu desempenho seja degradado. A transição de código interpretado, para a sua contrapartida compilada aliada à otimização, pode fornecer ganho de desempenho para aplicações críticas neste quesito. Outra possibilidade é a adoção de um sistema de programação baseado em diagramas, melhorando ainda mais a facilidade no modelo de programação adotado. Além disso, pode-se melhorar a arquitetura de execução de aplicações, possibilitando a execução distruibída da apli-

cação, utilizando-se mais de um sistema embutido ao mesmo tempo, introduzindo o paralelismo.

Referências Bibliográficas

A. A. Jerraya, W. W. (2005). *Hardware/software interface codesign for embedded systems.* Computer, 38(2):63-69 edição.

A. Sangiovanni-Vincentelli, G. M. (2003). *Application-defined scheduling in Ada.* ACM Press, in irtaw'03: proceedings of the 12th international workshop on real-time ada edição.

Bovet, D. P. e Cesati, M. (2003). *Understanding the Linux Kernel.* O'Reilly.

Coatrieux, J. L. (2005). Computer vision and graphics: frontiers, interfaces, crossovers and overlaps in science. *Engineering in Medicine and Biology Magazine*, 24(1):16–19.

Corporation, I. (2007). Open cv library. `http://www.intel.com/technology/computing/opencv/`.

de Sousa Carmo, G. T. (2009). *DSCam: uma plataforma hardware-software para operações de Visão Computacional. Dissertação (Mestrado).* UFMG – Universidade Federal de Minas Gerais.

Deitel, H. M. e Deitel, P. J. (2001). *Java How to Program.* Prentice Hall PTR, Upper Saddle River, NJ, USA.

Devices, A. (2005). Adsp-bf537 blackfin processor - hardware reference. `http://www.analog.com/processors/blackfin/`.

Edwards, G. T.; Schmidt, D. C. e Gokhale, A. (2004). Integrating publisher/subscriber services in component middleware for distributed real-time and embedded systems.

In *ACM-SE 42: Proceedings of the 42nd annual Southeast regional conference*, pp. 171–176, New York, NY, USA. ACM Press.

Elias T. Silva, J.; Wagner, F. R.; Freitas, E. P. e Pereira, C. E. (2006). Hardware support in a middleware for distributed and real-time embedded applications. In *SBCCI '06: Proceedings of the 19th annual symposium on Integrated circuits and systems design*, pp. 149–154, New York, NY, USA. ACM Press.

Eriksen, R. D. (2006. (acesso em 20-03-2009)). *Image Processing Library 98 - Version 2.20*. `http://www.mip.sdu.dk/ipl98/`.

Forsyth, D. A. e Ponce, J. (2002). *Computer Vision: A Modern Approach*. Prentice Hall Professional Technical Reference.

Foster, A. (2007). *The advent of COTS middleware use in embedded systems*. `http://www.embedded-computing.com/articles/id/?2017`.

Fummi, F.; Perbellini, G.; Pietrangeli, R. e Quaglia, D. (2007). Interactive presentation: A middleware-centric design flow for networked embedded systems. In *DATE '07: Proceedings of the conference on Design, automation and test in Europe*, pp. 1048–1053, San Jose, CA, USA. EDA Consortium.

Geihs, K. (2001). Middleware challenges ahead. *Computer*, 34(6):24–31.

Geihs, K. (June 1984). *Middleware challenges ahead*. IEEE Computer, 34(6):24-31 edição.

Gonzalez, W. (2002). *Digital Image Processing*. Prentice Hall, 2nd edition edição.

Hennessy, J. L. e Patterson, D. A. (2003). *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Horstmann, C. e Cornell, G. (1998). *Core Java 2, Volume 1: Fundamentals*. Prentice Hall PTR, Upper Saddle River, NJ, USA.

Horstmann, C. e Cornell, G. (2004). *Core Java(TM) 2, Volume II–Advanced Features (7th Edition)*. Prentice Hall PTR, Upper Saddle River, NJ, USA.

Inc., A. D. (Acesso em 2009a). Ascii: American standard code for information interchange. `http://www.asciitable.com`.

Inc., A. D. (Acesso em 2009b). Blackfin image analysis. `ftp://ftp.analog.com/pub/www/technology/dsp/Blackfin/benchmarks/vdsp35/Blackfin_ImageAnalysis.zip`.

Jerraya, A. A. (2004). *Long term trends for embedded system design*. IEEE Computer Society, in dsd, pages 20-26 edição.

Kernighan, B. W. (1988). *The C Programming Language*. Prentice Hall Professional Technical Reference.

Malek, S.; Seo, C. e Medvidovic, N. (2006). Tailoring an architectural middleware platform to a heterogeneous embedded environment. In *SEM '06: Proceedings of the 6th international workshop on Software engineering and middleware*, pp. 63–70, New York, NY, USA. ACM Press.

Mathworks (2007). Matlab - the language of technical computing. `http://www.mathworks.com/products/matlab/`.

μCLinux (2007. (acesso em 20-03-2009)). μclinux. `http://www.uclinux.org`.

Niblack, W. (1984). *An Introduction to Digital Image Processing*. Prentice Hall.

Patterson, D. A. e Hennessy, J. L. (1998). *Computer organization and design (2nd ed.): the hardware/software interface*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Peleg, A. e Weiser, U. (1996). Mmx technology extension to the intel architecture. *Micro*, 16(4):42–50.

Peterson, L. L. e Davie, B. S. (2007). *Computer Networks: A Systems Approach, Fourth Edition (The Morgan Kaufmann Series in Networking)*. Morgan Kaufmann, 4 edição.

pOSEK (2007). A super-small, scalable real-time operating system of high-volume, deeply embedded applications. `http://www.isi.com/products/posek/index.htm`.

Poskanzer, J. (2003). Pgm format specification. `http://netpbm.sourceforge.net/doc/pgm.html`.

Pressman, R. (2005). *Software Engineering: A Practitioner's Approach*. McGraw-Hill Professional, 6th edition edição.

Schmidt, D. C. (1999). Middleware techniques and optimizations for real-time, embedded systems. In *ISSS '99: Proceedings of the 12th international symposium on System synthesis*, p. 12, Washington, DC, USA. IEEE Computer Society.

Schmidt, D. C. (2002). Middleware for real-time and embedded systems. *Commun. ACM*, 45(6):43–48.

Schmidt, D. C.; Gokhale, A.; Schantz, R. E. e Loyall, J. P. (2004). Middleware r&d challenges for distributed real-time and embedded systems. *SIGBED Rev.*, 1(1):6–12.

SensorNet (Acesso em 20-03-2009). Sensornet. `http://www.sensornet.dcc.ufmg.br/index.html`.

Sharp, H.; Rogers, Y. e Preece, J. (2007). *Interaction Design: Beyond Human Computer Interaction*. Wiley.

Shi, Y. (2009). *Smart Cameras for Machine Vision*. Springer.

Silberschatz, A. e Galvin, P. B. (2000). *Operating System Concepts*. John Wiley & Sons, Inc., New York, NY, USA.

Sommerville, I. (2006). *Software Engineering*. Addison-Wesley, 8th edition edição.

staff, I. C. (1999). Middleware's role, today and tomorrow. *IEEE Concurrency*, 7(2):70–80.

Stroustrup, B. (1997). *The C++ Programming Language, Third Edition*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Suseela, A. L. e Kumar, V. L. (2005). Embedded systems in real time applications, design & architecture. *Ubiquity*, 6(28):2–2.

Tao, W. e Majumdar, S. (2002). Application level performance optimizations for corba-based systems. In *WOSP '02: Proceedings of the 3rd international workshop on Software and performance*, pp. 95–103, New York, NY, USA. ACM Press.

TinyOS, A. (2007a). nesc - network embedded systems c. `http://www.tinyos.net`.

TinyOS, A. (2007b). Tinyos. `http://www.tinyos.net`.

Trucco, E. e Verri, A. (1998). *Introductory Techniques for 3-D Computer Vision*. Prentice Hall PTR, Upper Saddle River, NJ, USA.

Vieira, L. F. M. (2004). *Middleware para Sistemas Embutidos e Redes de Sensores . Dissertação (Mestrado)*. UFMG – Universidade Federal de Minas Gerais.

Wikipedia (2007). Matlab. `http://en.wikipedia.org/wiki/MATLAB`.

Wolf, W. (2001). *Computers as components: principles of embedded computing system design*. Morgan Kaufmann Publishers Inc.

Apêndice A

Telas da Ferramenta

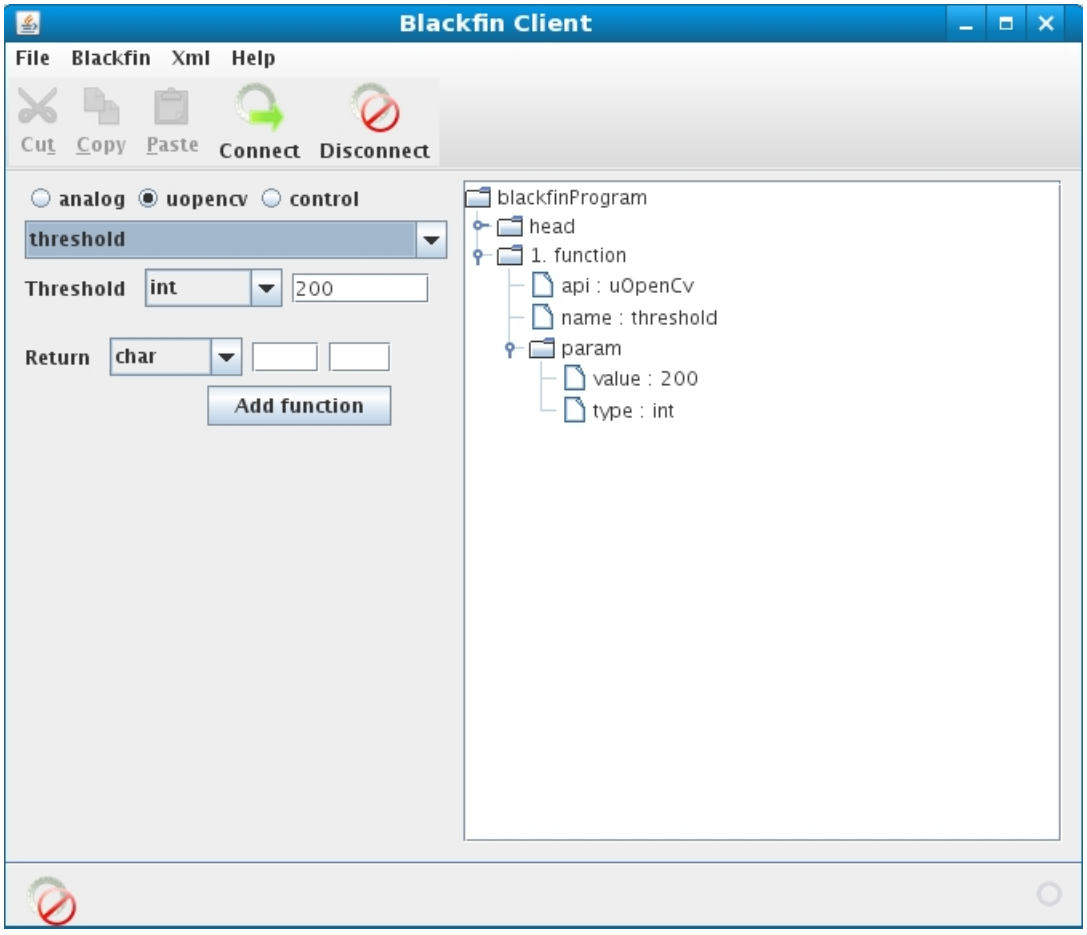

Figura A.1. Construção de aplicação

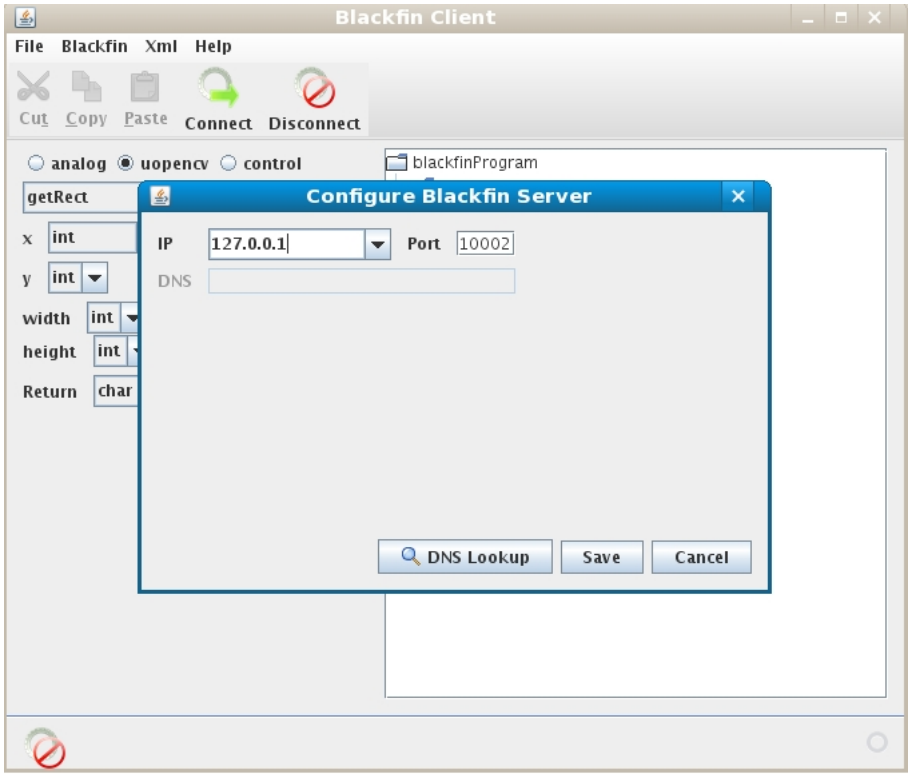

Figura A.2. Configuração de endereço do servidor

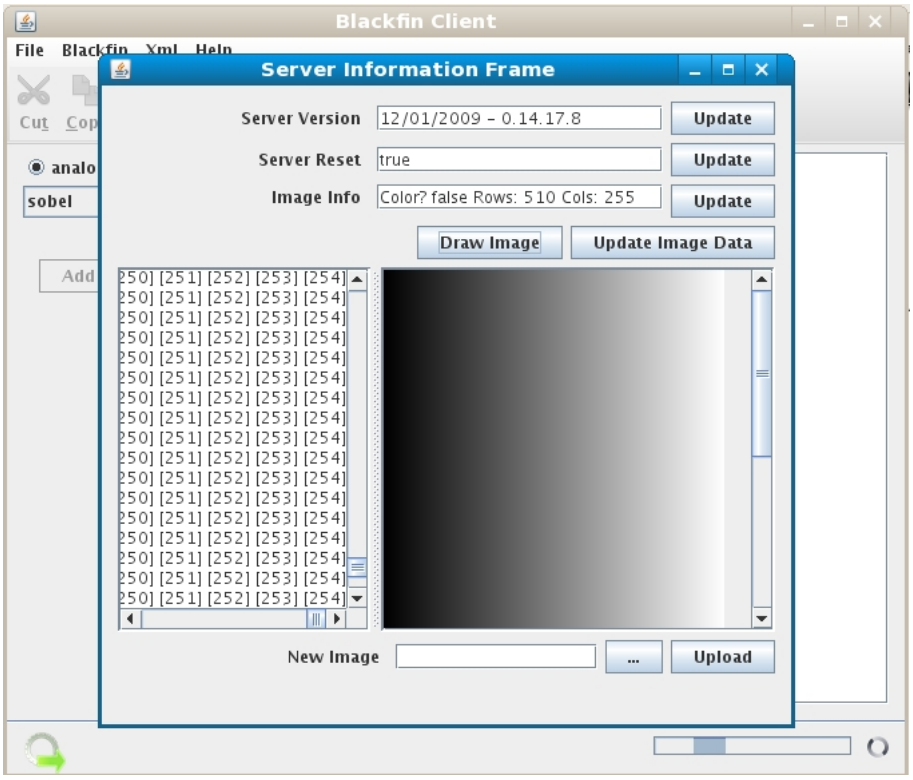

Figura A.3. Informações sobre o servidor utilizado

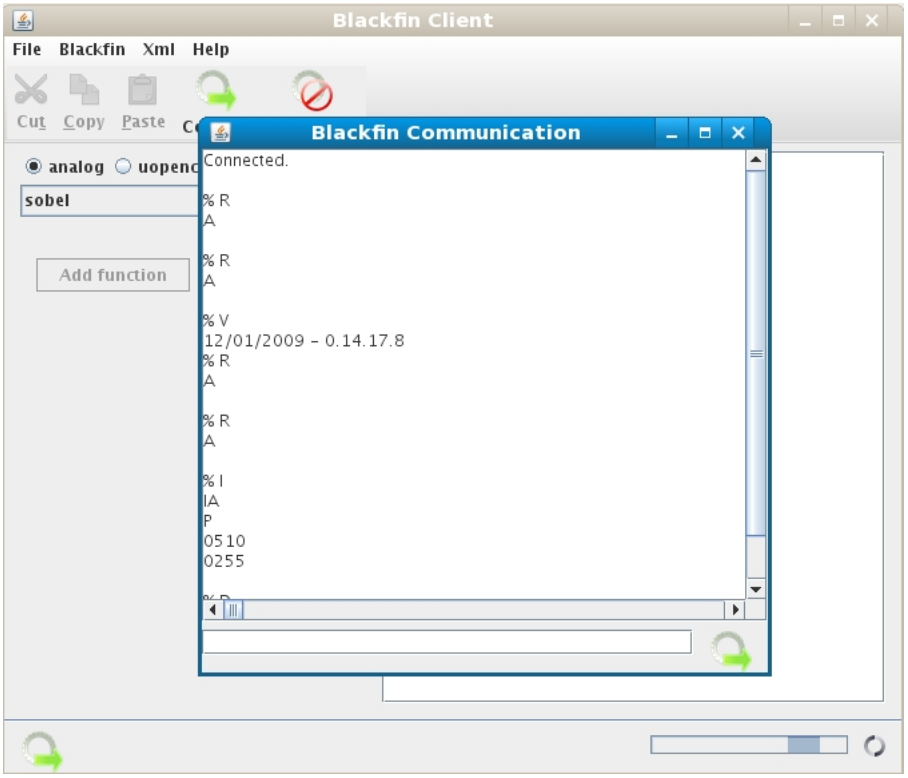

Figura A.4. Comunicação com o servidor (protocolo ASCII)

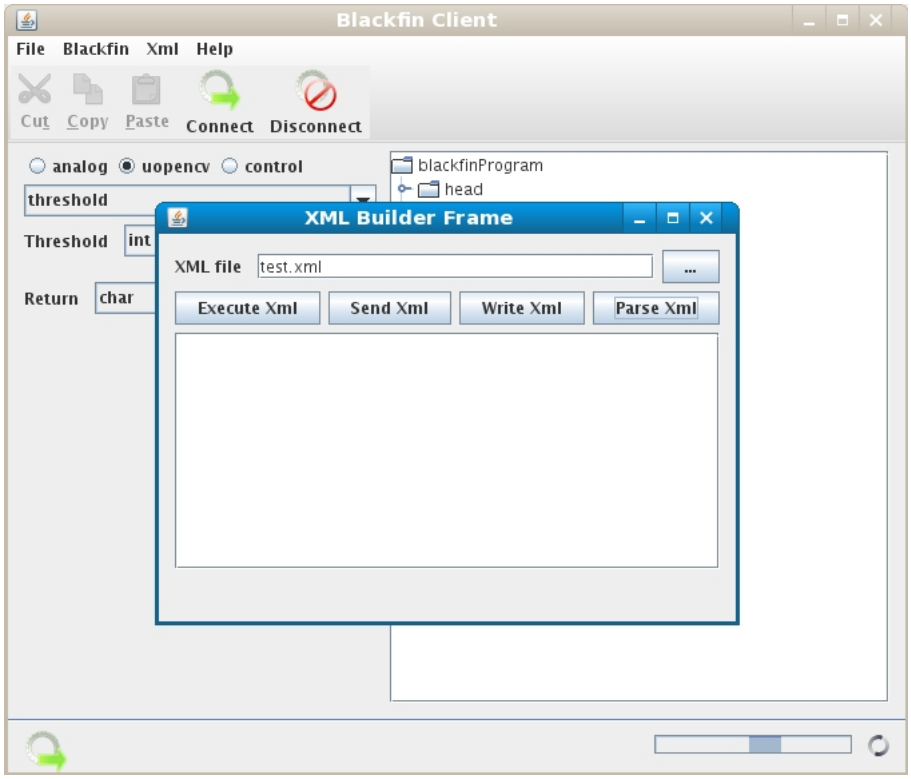

Figura A.5. Construção do XML

Apêndice B

Aplicação Presença ou Ausência: código XML gerado

Neste apêndice, está listado o código XML gerado para a aplicação de presença ou ausência de objeto descrita na Seção 4.1.

```
1  <?xml version="1.0" encoding="UTF-8" standalone="no"?>
2  <blackfinProgram>
3      <!--Retorna 1 se objeto presente, 0 se objeto ausente.-->
4      <head>
5          <name>Presenca</name>
6          <author>Antonio</author>
7          <version>1.0</version>
8          <date>20090307</date>
9      </head>
10     <decl>
11         <var>ret</var>
12         <size>1</size>
13         <type>int</type>
14     </decl>
```

```
15      <decl>
16          <var>numWhite</var>
17          <size>1</size>
18          <type>int</type>
19      </decl>
20      <assign>
21          <var>numWhite</var>
22          <position>0</position>
23          <value>0</value>
24      </assign>
25      <function>
26          <api>u/gOpenCv</api>
27          <name>threshold</name>
28          <param>
29              <value>128</value>
30              <type>int</type>
31          </param>
32      </function>
33      <function>
34          <api>u/gOpenCv</api>
35          <name>countWhite</name>
36          <return>
37              <var>numWhite</var>
38              <pos>0</pos>
39          </return>
40      </function>
41      <if>
42          <expression>numWhite &gt; 8192</expression>
```

```
43              <goto>7</goto>
44          </if>
45          <assign>
46              <var>ret</var>
47              <position>0</position>
48              <value>0</value>
49          </assign>
50          <goto>
51              <address>9</address>
52          </goto>
53          <assign>
54              <var>ret</var>
55              <position>0</position>
56              <value>1</value>
57          </assign>
58          <goto>
59              <address>9</address>
60          </goto>
61          <return>
62              <var>ret</var>
63              <pos>0</pos>
64              <size>1</size>
65        </return>
66          <halt/>
67  </blackfinProgram>
```

?????@@Neste apêndice, está listado o código XML gerado para a aplicação de presença ou ausência com controle de erro descrita na Seção 4.2.

```
1  <?xml version="1.0" encoding="UTF-8" standalone="no"?>
```

```
2  <blackfinProgram>
3      <!--This is a Hello World program.-->
4      <head>
5          <name>Presenca com controle de erro</name>
6          <author>Antonio</author>
7          <version>1.0</version>
8          <date>20090307</date>
9      </head>
10     <decl>
11         <var>ret</var>
12         <size>1</size>
13         <type>int</type>
14     </decl>
15     <decl>
16         <var>numWhite</var>
17         <size>1</size>
18         <type>int</type>
19     </decl>
20     <decl>
21         <var>threshold</var>
22         <size>1</size>
23         <type>int</type>
24     </decl>
25     <assign>
26         <var>numWhite</var>
27         <position>0</position>
28         <value>0</value>
29     </assign>
```

```
30      <assign>
31          <var>threshold</var>
32          <position>0</position>
33          <value>118</value>
34      </assign>
35      <push>
36          <pos>0</pos>
37      </push>
38      <pop>
39          <pos>0</pos>
40      </pop>
41      <assign>
42          <var>threshold</var>
43          <position>0</position>
44          <value>threshold + 10</value>
45      </assign>
46      <if>
47          <expression>threshold &gt; 240</expression>
48          <goto>12</goto>
49      </if>
50      <push>
51          <pos>0</pos>
52      </push>
53      <function>
54          <api>u/gOpenCv</api>
55          <name>threshold</name>
56          <param>
57              <value>threshold</value>
```

```
58              <type>int</type>
59          </param>
60      </function>
61      <function>
62          <api>uOpenCv</api>
63          <name>countWhite</name>
64          <return>
65              <var>numWhite</var>
66              <pos>0</pos>
67          </return>
68      </function>
69      <if>
70          <expression>numWhite &gt; 8192 + 4096</expression>
71          <goto>14</goto>
72      </if>
73      <if>
74          <expression>numWhite &gt; 8192 - 4096</expression>
75          <goto>5</goto>
76      </if>
77      <assign>
78          <var>ret</var>
79          <position>0</position>
80          <value>0</value>
81      </assign>
82      <goto>
83          <address>16</address>
84      </goto>
85      <assign>
```

```
86              <var>ret</var>
87              <position>0</position>
88              <value>1</value>
89          </assign>
90          <goto>
91              <address>16</address>
92          </goto>
93          <return>
94              <var>ret</var>
95              <pos>0</pos>
96              <size>1</size>
97          </return>
98          <halt/>
99  </blackfinProgram>
```