Universidade Federal de Pernambuco
Centro de Informática
Graduação em Ciência da Computação

TVDesigner: Uma Ferramenta para Criação de Aplicações MHP Interativas para TV Digital

Djaci Alves de Araujo Filho

TRABALHO DE GRADUAÇÃO

Recife
2008

Universidade Federal de Pernambuco
Centro de Informática

Djaci Alves de Araujo Filho

TVDesigner: Uma Ferramenta para Criação de Aplicações MHP Interativas para TV Digital

Monografia apresentada ao Centro de Informática da Universidade Federal de Pernambuco, como requisito parcial para obtenção do Grau de Bacharel em Ciência da Computação.

Orientador: *Prof. Ph.D.* Carlos André Guimarães Ferraz

Recife
2008

A minha família

AGRADECIMENTOS

Em primeiro lugar agradeço a Deus por tudo. Agradeço por todas as bênçãos e por tudo que ele permitiu que acontecesse em minha vida.

Agradeço aos meus pais Djaci e Luci pelo amor, compreensão, apoio, carinho e tudo que eles têm feito por mim.

Agradeço a minha tia Zezé que sempre tem demonstrado o seu amor por mim.

Agradeço a minha irmã Luciana por todos os momentos bons vividos.

Aos amigos da faculdade, que estiveram comigo todos esses anos sem nunca perder o bom humor e pelos bons momentos vividos.

Ao meu orientador Carlos Ferraz por me mostrar o caminho a seguir em mais esta fase da minha vida.

Aos amigos do Cenas que me proporcionaram ótimos momentos e por terem me ajudado neste trabalho.

A Andrino e Mariana por terem me apoiado no desenvolvimento deste trabalho.

RESUMO

Com a rápida evolução tecnológica estamos vivendo mudanças onde surgem novos equipamentos e mídias convergentes. No decorrer deste processo uma nova mídia que surge é a televisão digital trazendo recursos de interatividade.

Por se tratar de um mercado amplo e com uma demanda crescente, a produção de aplicações interativas deve ser otimizada. Tornando-as mais atrativas, diminuindo o tempo de desenvolvimento, suprindo a pendência de interatividade existente, com isso tornando as emissoras mais competitivas em relação às outras.

Este trabalho tem como objetivo propor uma ferramenta, o *TVDesigner*, que se encaixa na etapa de implementação de aplicações para TVD MHP, acelerando e facilitando o processo de desenvolvimento, provendo recursos gráficos e de geração de código.

Palavras-chave: televisão digital, MHP, interatividade, aplicações.

ABSTRACT

With the rapid technological growth we are living changes where new equipments and converging medias arise. During this process a new media that emerges is the digital television, enabling the interactivity.

This is a market with a broad and growing demand. Production of interactive applications must be optimized aiming to turn then more attractive, by reducing the development time, supplying the demand of interactivity, and increasing the competitive among the producers.

The report aim is to propose a tool, the TVDesigner, which is placed in the implementation phase of DTV MHP applications, facilitating and speeding up.the development process, by providing graphical user interface and code generation.

Keywords: digital television, MHP, interactive, applications.

SUMÁRIO

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

API	Application Programming Interface
DLL	Dynamic Link Library
DTV	Digital TV
DVB	Digital Video Broadcasting
EPG	Eletronic Program Guide
GEM	Globally Executable MHP
GUI	Graphical User Interface
HAVi	Home Audio/Video Interoperability
MHP	Multimedia Home Platform
STB	Set-Top-Box
TVD	TV Digital

1. INTRODUÇÃO

1.1. Contexto

Com a rápida evolução tecnológica estamos vivendo mudanças em que surgem novos equipamentos e mídias convergentes. A convergência é um processo de mudança qualitativa que liga dois ou mais mercados existentes e anteriormente distintos [9]. Esta faz com que uma única estrutura tecnológica seja utilizada para prover serviços, que anteriormente requeriam equipamentos, canais de comunicação, protocolos e padrões independentes.

No decorrer deste processo uma nova mídia que surge é a televisão digital, disponibilizando melhores recursos de som e imagem, além disso, possibilitando ao telespectador passar de um ator passivo a uma postura ativa, interagindo diretamente com a televisão, através das aplicações interativas como jogos, compras (*t-commerce*), EPG, propagandas, etc.

Estas evoluções e o crescimento do mercado refletem diretamente nos produtores de conteúdo televisivo. Por se tratar de um mercado amplo e em que existe uma demanda crescente, cria-se um grande desafio no que diz respeito à produção de aplicações em TVD (TV Digital) que melhor se adéqüem as necessidades da sua audiência. Este desenvolvimento, por ser feito de forma semi-artesanal, com documentação precária e dificuldade de reutilização e manutenção [7], consome muito tempo do processo, muitas vezes não aproveitando todo o potencial possível de uma aplicação interativa. Este processo deve ser otimizado com o intuito de tornar as aplicações mais atrativas, diminuindo o tempo de desenvolvimento, suprindo a pendência de interatividade existente, com isso tornando as emissoras mais competitivas em relação às outras [6].

1.2. Objetivos

Este trabalho tem como objetivo propor uma ferramenta, o *TVDesigner*, que se encaixa na etapa de implementação no processo de desenvolvimento de aplicações para TVD MHP, agilizando e facilitando o desenvolvimento,

disponibilizando recursos gráficos para a elaboração de telas e buscando eliminar a utilização de código na geração destas aplicações. Detalhes de implementação serão mostrados como forma de explanar como foram resolvidos alguns problemas.

Por fim realizamos um estudo de caso e faremos uma comparação no tempo de desenvolvimento com o processo clássico de desenvolvimento (utilização de código).

1.3. Estrutura do Trabalho

Este trabalho é composto de 6 capítulos. No capítulo 2, será abordado um processo de desenvolvimento adaptado para de aplicações para TV digital interativa no padrão MHP e outras tecnologias. No capitulo 3, serão mostrados a visão geral do *TVDesigner*, suas funcionalidades para o desenvolvimento de um aplicação interativa e a sua arquitetura.

No capitulo 4, serão mostrados detalhes de implementação da solução, e padrões utilizados na sua realização. No capitulo 5, serão mostrados a utilização da ferramenta na geração de uma aplicação TVD, e os resultados obtidos com a realização de um estudo de caso. Finalizando, no capitulo 6, faremos as considerações finais deste trabalho juntamente com sugestões e trabalhos futuros.

2. TRABALHOS RELACIONADOS

Inicialmente, neste capitulo, discutiremos um pouco sobre a TVD interativa e a necessidade de uma plataforma padronizada que suporte a execução de aplicações interativas, onde focaremos no *middleware* MHP. Em seguida falaremos dos processos de desenvolvimento de aplicações interativas para TVD. Por fim faremos as considerações dos tópicos abordados neste capitulo.

2.1. TV Digital Interativa

Com o inicio da televisão digital interativa surgem inúmeras oportunidades de mercado para a produção de serviços e aplicações. Durante os primeiros anos de TVD, em que grupos de pesquisa e empresas implantavam suas soluções, o único meio disponível para executar aplicações interativas, em escala de produção, eram os sistemas proprietários, onde o desenvolvimento da solução é feito verticalmente e toda a cadeia de produção, desde equipamentos de transmissão/recepção a softwares, pertence a uma única empresa. Esta forma não condiz com o mercado horizontal existente, onde várias empresas produzem soluções em vários níveis da cadeia de produção [10].

Devido ao fato de produção heterogênea e distribuída, faz-se necessário a criação de um padrão para a execução de aplicações interativas. O MHP, desenvolvido pelo projeto DVB (Digital Video Broadcast), surge tentado resolver este problema.

2.1.1. Multimedia Home Platform

Como chave para o sucesso em aplicações interativas multimídia em um futuro próximo, o MHP tem o objetivo de padronizar elementos de uma plataforma residencial (set-top boxes, televisores, entre outros) [6].

Sendo um *middleware* aberto produzido por um consórcio de mais de 300 empresas entre transmissoras, universidades, fabricantes de equipamentos, órgãos reguladores e mais de 35 países [3]. O MHP define uma interface entre o receptor

de TV e a rede que está conectada para dar suporte a serviços interativos [11]. Esta interface desacopla a aplicação dos detalhes específicos de *hardwares* e detalhes de implementação de *software*, possibilitando que uma aplicação gerada possa ser executada em qualquer tipo de equipamento como *set-top boxes* (simples e complexos), TVs digitais integradas e computadores multimídia.

MHP vem se estabelecendo como uma plataforma em comum para o desenvolvimento de aplicações para TVD, dentre vários fatores estão alguns citados por [1]:

- Qualquer pessoa possui permissão para implementar o *middleware*. A especificação pode ser baixada gratuitamente, e o único custo é o de uma pequena taxa para o teste de conformidade e taxas de propriedade intelectual do DVB;
- Aplicações são escritas em Java ou HTML, então não dependem de uma única plataforma de hardware ou sistema operacional. A especificação do MHP não entra em detalhes no que diz respeito à plataforma de *hardware* necessária, dando liberdade aos desenvolvedores;
- MHP definiu a especificação do GEM (Globally Executable MHP) que permite outras entidades construir padrões, que são compatíveis as normas do MHP, enquanto respeitam as necessidades técnicas e comerciais dos mercados locais.

Uma categorização de aplicações MHP se da considerando o tipo de interatividade disponibilizada ao usuário. Estas categorias descritas, em seguida, mostram também a evolução da complexidade das aplicações e do suporte à conexão dos terminais de acesso, sendo estas categorias descritas por [4] [6] como:

- ***Enhanced Broadcasting*:** A informações acessadas vindas unicamente do *middleware*, e os receptores que executam estes tipos de aplicações podem ser mais simples e baratos. Conhecida como interatividade local;
- ***Interactive Profile*:** Também chamado de canal de interatividade. Adiciona o conceito de canal de retorno, possibilitando uma comunicação bi-direcional, onde informações, especificas do usuário, podem ser enviadas e recebidas por este canal;

- ***Internet Profile*:** Suporte de conteúdos, como *e-mails* e *sites*, baixados diretamente da Internet;

2.2. Processos de Software

Um processo de desenvolvimento compõe um conjunto de atividades a serem executadas e que levam a produção de um *software*. Mesmo havendo uma diversidade e variâncias de processos existentes, estes se baseiam em quatro atividades fundamentais [14], que são:

- **Especificação de *software*:** Onde são definidas as funcionalidades do *software* e restrições de operação.
- **Projeto e implementação de *software*:** Ocorre a produção do *software* seguindo a especificação.
- **Validação de *software*:** Deve ser garantido que o *software* faz o que o cliente deseja.
- **Evolução de *software*:** Fase em que o *software* sofre mudanças de acordo com as necessidades do cliente.

De acordo com cada necessidade o processo pode sofrer adaptações que podem ocasionar em um melhor ou pior aproveitamento das boas práticas de engenharia de *software*. Essas adaptações e mudanças realizadas visam um aumento da qualidade, tanto do *software* entregue, quanto de todo o processo de desenvolvimento.

2.3. Processos de Software em TVD

Como mencionado anteriormente, o processo pode ser melhorado e/ou adaptado para melhor atender a uma demanda especifica. No caso de TV digital, onde existe um mercado que cresce rapidamente, o processo deve ser adaptado para tentar suprir os requisitos necessários ao desenvolvimento de uma aplicação de forma rápida.

Atualmente, aplicações de TVD são criadas para exercer funções especificas, como o caso de uma simples votação, compra de um determinado produto, ou

mesmo exibição de um conteúdo especifico. Mesmo com o objetivo de disponibilizar uma funcionalidade ao usuário, as aplicações de TV digital focam, durante o seu desenvolvimento, na identidade visual que será mostrada. Este fator faz com que desde o inicio do desenvolvimento exista a necessidade de um *feedback* visual constante ao cliente de como a aplicação será exibida.

As aplicações, em geral, possuem o conteúdo casual e não tendem a ser complexas a ponto de criar muitos módulos e entidades. Outro fator determinante é que desenvolvimento de uma aplicação deve sempre estar em sintonia com o interesse do cliente, na maioria dos casos as emissoras. A mudança de um design de uma aplicação, por exemplo, pode ser solicitado com urgência, pois o programa televisivo a que esta aplicação referencia mudo o layout.

Fatores citados acima fazem com que aplicações de TV digital, em geral, sejam pequenas e que todos os requisitos possam ser elicitados na etapa de especificação. Este fator possibilita, em alguns casos, a produção de uma aplicação completa em apenas um ciclo podendo se utilizar do modelo cascata, ao invés de realizar várias interações com o cliente.

2.3.1. Processo adaptado para TVD

Com base na necessidade de visualização de um protótipo para validação da solução proposta e o fato de muitas aplicações serem pequenas, existe um processo adaptado, utilizado por uma empresa de *software* no Recife, para TVD que divide o desenvolvimento nas seguintes etapas:

Requisitos

Nesta etapa são realizados o estudo de viabilidade, a elicitação dos requisitos do cliente, a documentação, a definição dos requisitos e o projeto da interface. Este ultimo focando na premissa de sempre possibilitar um *feedback* visual da aplicação ao cliente. Nesta fase, são gerados dois documentos para o cliente, que dizem respeito à solução proposta. O primeiro chamado de *wireframe* e o segundo *Mockup*. O primeiro elícita os requisitos funcionais de cada tela, os não funcionais,

as restrições e premissas da aplicação. Este documento exibe como a aplicação funcionará e também a disposição dos componentes gráficos, sem design, na tela da TV. O segundo documento representa o design da aplicação, visualizando como serão realmente as telas da aplicação quando exibidas na televisão.

Análise e Projeto

Após a elicitação dos requisitos do cliente passamos para a fase onde serão definidas a arquitetura do sistema, projetados os componentes, modelados os dados e o projeto será consolidado. Esta etapa é importante, pois determinará como se dará o desenvolvimento da aplicação. No caso de aplicações em TVD possuírem uma natureza, em geral, simples e o tempo ser reduzido, faz com que nesta etapa a documentação da arquitetura e do projeto do sistema possa até ser suprimida (sem dispensar os modelos criados) ou até mesmo feito de forma simplificada, isto deve ser decidido pela equipe.

Implementação

Esta é uma fase onde é preparado o ambiente de desenvolvimento, implementado o sistema, realizados os testes unitários onde são geradas as imagens finais dos componentes que serão posicionados na aplicação. Nesta etapa de implementação como as aplicações de televisão digital, e no nosso caso MHP, possuem um padrão para exibição de componentes e de estrutura básica para execução, existem pontos que poderiam ser automatizados para agilizar o processo de desenvolvimento.

É neste ponto que o *TVDesigner* se insere propondo uma automatização no processo de preparação do ambiente e implementação do sistema, e facilitando a configuração dos componentes gráficos, tentando abstrair a utilização de código de linguagem de programação. Esta automação na produção de aplicações acarreta uma maior produtividade e uma diminuição dos erros de código, já que este estaria sendo evitado na fase inicial de desenvolvimento.

Testes

Esta fase é de extrema importância durante o processo de desenvolvimento, pois nela são testadas as lógicas internas do software e funcionalidades externas. Assegurando que as entradas produzem os resultados especificados. Como tarefas, temos o planejamento e o projeto de testes. Em seqüência os testes são implementados e realizados os testes e uma planilha de resultado é gerada. No caso de TVD os casos de teste são gerados a partir dos requisitos do *wireframe*. E a conformidade visual é testada de acordo com o *mockup*.

Implantação

Por fim temos a implatação, onde temos o planejamento da implantação, e onde é gerada a documentação de suporte e por fim a implantação é realizada.

2.4. Considerações

Neste capitulo foi discutido o processo de desenvolvimento de aplicações, e mencionado o problema de tempo que deve ser diminuído para redução de custos e aumento da produtividade para suprir a demanda por aplicações interativas existente. Sabendo que o processo deve ser agilizado propomos uma solução no próximo capitulo.

3. TV DESIGNER

Neste capitulo detalharemos a solução proposta, o *TVDesigner*. Nos tópicos seguintes mostraremos uma visão geral do projeto, suas funcionalidades no processo de desenvolvimento de uma de aplicação para TVD MHP e a arquitetura utilizada na sua realização.

3.1. Visão Geral

Como mencionado anteriormente, desejamos executar cada etapa do processo de desenvolvimento o mais rápido possível para suprir a demanda por aplicações interativas. Sabendo que a atividade de implementação de solução MHP baseia-se na utilização de código, onde a produtividade a probabilidade de ocorrência de erros é grande, o *TVDesigner* foi concebido com o intuito de prover uma ferramenta que auxilie no processo de desenvolvimento de aplicações MHP. A utilização desta ferramenta encaixa-se durante a etapa de implementação da solução, como um agente facilitador e automotizador na execução de algumas tarefas. Utilizando-se de recursos gráficos e de geração automática de código como pontos chave do sucesso da ferramenta.

3.2. Requisitos

Durante a etapa de elicitação de requisitos foram entrevistados funcionários de uma empresa de *software* para TV digital do Recife. Portanto, eles possuem o conhecimento necessário a respeito das necessidades que uma ferramenta com este objetivo deve suprir.

Como resultado os seguintes Requisitos Funcionais (RF) foram elicitados:

- [RF01] – O sistema deverá possuir um editor visual para facilitar o desenvolvimento de interfaces gráficas;
- [RF02] – Disponibilizar um diagrama de visualização de relacionamento entre telas para validar o fluxo da aplicação;

- [RF03] – Apresentação de uma barra de ferramentas com os componentes gráficos disponíveis para utilização;
- [RF04] - Visualização das propriedades dos componentes gráficos.
- [RF05] - *Drag and drop* dos componentes;
- [RF06] – Visualização de perspectivas de 'Design' e 'Código' para que seja possível realizar alterações manuais;
- [RF07] - Importação de imagens;
- [RF08] – Geração automática de código da aplicação;
- [RF09] - O sistema deverá dispor de itens de configuração na geração do projeto.

Além destes, alguns Requisitos Não Funcionais (RNF) também foram levados em consideração para a realização da solução:

- [RNF01] – O sistema deve ser intuitivo o bastante para o usuário saber o que deve ser feito;
- [RNF02] – O sistema necessita ser executado em sistemas operacionais dotados da .Net Framework 2.0;
- [RNF03] – A ajuda deve ser acessada a qualquer momento da execução da aplicação. Dispondo de textos explicativos instruindo ao usuário o que deve ser feito.

3.3. Arquitetura

A fim de tornar a solução mais modular e facilitar o controle, durante a etapa de projeto foi decida a utilização da arquitetura em camadas. Esta decisão possibilita um desacoplamento do sistema, facilitando posteriores manutenções e evoluções. Cada camada disponibiliza uma interface de serviços, por onde a camada superior pode acessar as funcionalidades.

Esta abordagem possibilita que a interface gráfica se preocupe apenas com a apresentação da aplicação acessando a camada de serviços para executar as funcionalidades do sistema. Isto possibilita, por exemplo, a mudança das regras de

negócio sem que afete outras partes do sistema, como as entidades e a interface. A figura 3.1 ilustra a arquitetura utilizada na solução.

Figura 3.1: Arquitetura do TVDesigner.

A camada de serviço provê todas as funcionalidades requeridas pela camada gráfica para que a aplicação funcione. Abaixo da camada da camada de serviços está camada de negócios, que possui controladores que gerenciam as regras de negócios das classes básicas. Em seguida temos a camada de persistência das entidades de negócio.

4. IMPLEMENTAÇÃO

Este capitulo tem o intuito de demonstrar detalhes de implementação da ferramenta *TVDesigner*, mostrando o módulos que compõem a solução, suas características, funcionalidades, e padrões utilizados para a geração destes componentes. O *TVDesigner* consiste de uma solução, na linguagem C# implementada utilizando o ambiente *Visual Studio .NET 2005*, dividido em 6 módulos.

4.1. Módulos

4.1.1. Entities

Este módulo, implementado em forma de DLL, contém as entidades de negócio da aplicação. Estas classes básicas são abstrações responsáveis por guardar as informações necessárias para a geração de uma aplicação de TV digital como telas, botões, caixas de texto e etc. A figura 4.1 ilustra a seguir as entidades mais importantes criadas:

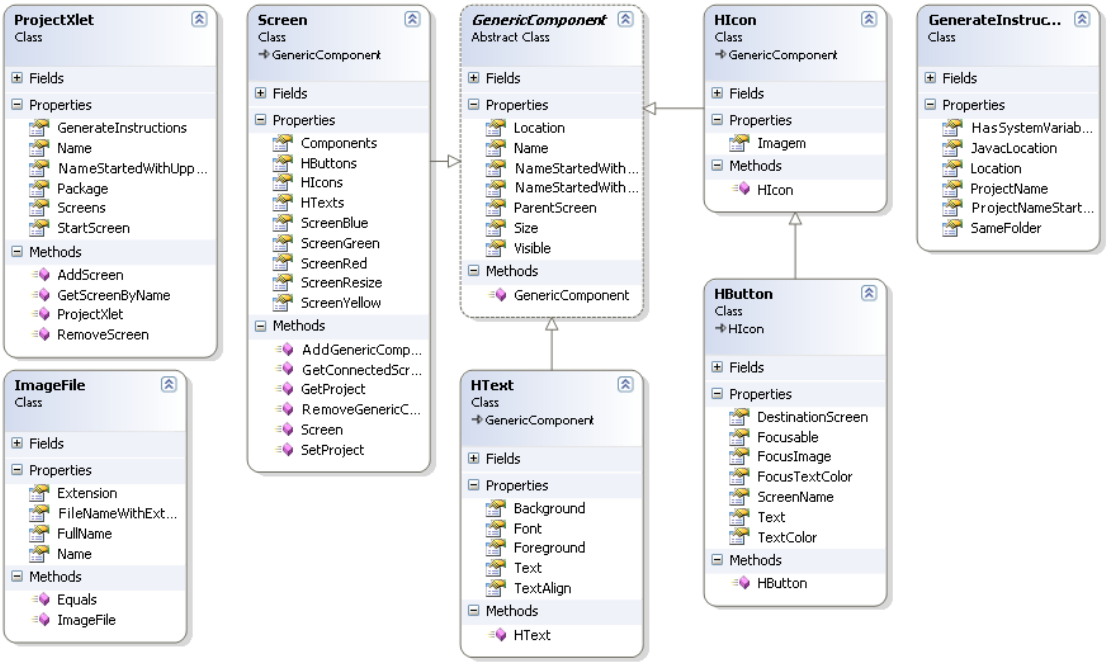

Figura 4.1: Diagrama das entidades do módulo *Entities*.

O detalhamento das classes é o que se segue:

1. A classe *ProjectXlet* representa um projeto criado pelo *TVDesigner*, armazenando o nome do projeto, que será utilizado para a geração da aplicação de TV digital MHP, a referência para a tela inicial da aplicação. Possuindo uma lista de telas da aplicação e informações de geração do projeto.
2. A classe *Screen*, que herda de *GenericComponent*, representa uma abstração criada para facilitar a implementação de aplicações para TV digital. Um *screen* possui uma lista de componentes, que podem ser botões, textos e imagens, que serão exibidos quando esta tela se tornar visível durante a execução da aplicação gerada.
3. A classe abstrata *GenericComponent* representa uma abstração de um componente que pode ser adicionando visivelmente em uma aplicação MHP. Possuindo nome, tamanho, localização na tela e um atributo *Visible* que ilustra se o componente está visível ou não, algumas ações durante a execução podem mudar o estado do componente.
4. A classe *HIcon* representa um componente que possui uma imagem relacionada, que será exibida na tela da aplicação de TV digital.
5. A classe *GenerateInstructions* representa uma abstração que tem o objetivo de armazenar informações necessárias para a geração de código da aplicação de TVD, sendo estas configuradas pelo usuário ao manipular um projeto criado pelo *TVDesigner*.
6. A classe *HText* representa um componente que possui características para exibição de um texto em uma aplicação de TVD.
7. A classe *HButton* representa um componente com atributos para a exibição de um botão. O botão pode conter um texto, a cor do texto, imagem de fundo, e imagem de fundo quando o botão está com foco. Possui também o atributo *Focusable* que representa se o botão pode ou não possuir foco. Estes botões sem foco podem executar ações a partir da utilização das teclas coloridas disponíveis em um controle de TVD.

4.1.2. Data Provider

Este módulo é responsável por garantir a persistência das entidades básicas quando salvamos um projeto gerados pelo *TVDesigner*. É responsável pela persistência de um *ProjectXlet*, já que este engloba todas as outras entidades de utilizadas no *TVDesigner*.

4.1.3. Business Provider

Este módulo representa a camada de serviços da aplicação. Por ele passam todas as execuções que solicitam alguma manipulação das classes básicas. Esta camada realiza as verificações das regras de negócio do sistema. Possuindo entidades controladoras para cada tipo de entidade de negócio.

4.1.4. Services Provider

Este módulo engloba todos os serviços disponibilizados pelos controladores da camada de negócio (*Business Provider*) em uma única interface, disponibilizando estas funcionalidades para a camada de apresentação.

4.1.5. Generator

O módulo *Generator* é responsável pela geração de arquivos que formam um projeto de uma aplicação TVD. Utilizando os dados armazenados pelas classes do módulo *Entities,* o *Generator* cria as pastas da solução gerada, os arquivos Java, com código para a aplicação MHP, manipulam as imagens utilizadas na aplicação TVD e cria um executável que permite a compilação dos arquivos. A figura 4.2 a seguir ilustra as entidades mais importantes criadas para este módulo:

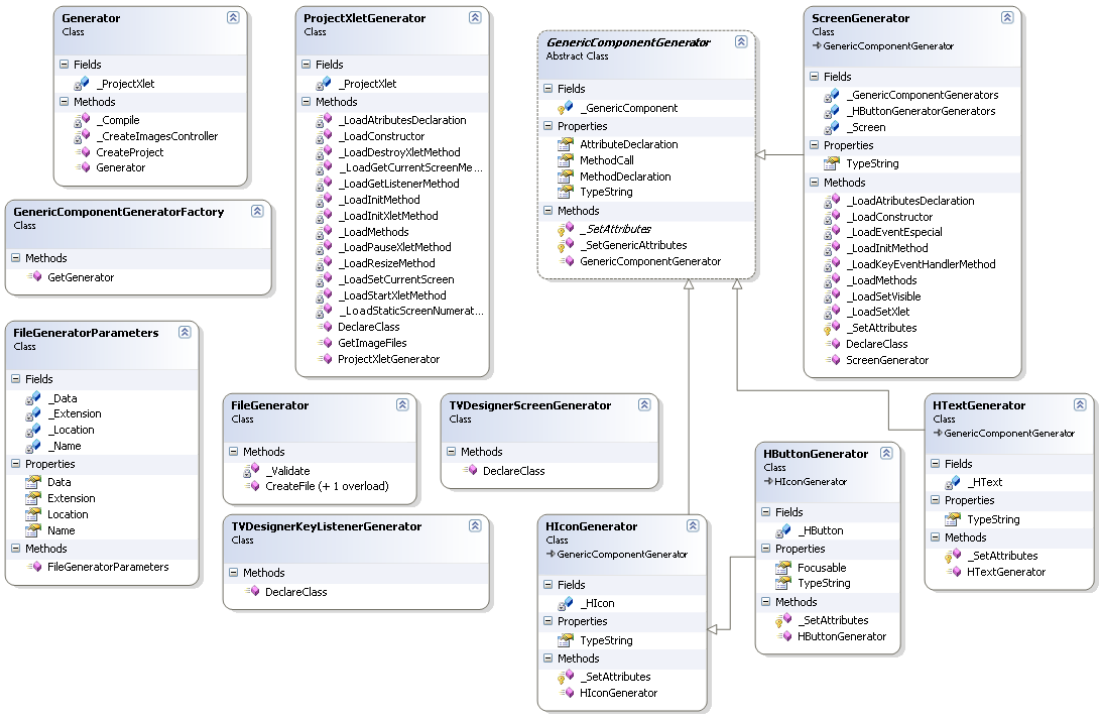

Figura 4.2: Diagrama das entidades do módulo *Generator*.

O detalhamento das classes é o que se segue:

1. A classe *Generator* é responsável pela geração do projeto. Esta entidade é criada unicamente para executar uma série de passos que acarretam na criação de uma aplicação de TVD. O método *CreateProject* é composto por um conjunto de passos:
   1.1. Criação das pastas do projeto.
   1.2. Criar a classe que implementa a interface *javax.tv.xlet.Xlet*, interface de entrada para as aplicações MHP.
   1.3. Criar a classe abstrata *TVDesignerScreen*, que é uma abstração utilizada para criar o conceito de telas em aplicações TVD.
   1.4. Criar a classe *TVDesignerKeyListener*, que é o *listener* de todos os eventos que acontecem na aplicação.
   1.5. Criar as classes que representam as telas da aplicação. Estas classes herdam de *TVDesignerScreen*.
   1.6. Copiar o arquivo *mhpstubs.jar*, que é utilizado na compilação de aplicações TVD MHP.

1.7. Criar a classe *ImagesController* responsável por manipular o carregamento das imagens utilizadas pela aplicação MHP.
1.8. Copiar todas as imagens utilizadas pela aplicação TVD para a pasta especifica de imagens criada no passo 1.1.
1.9. Criar executável de compilação. Este é gerada utilizando-se dos valores da entidade *GenerateInstructions*, como variáveis de sistema, local do compilador Java, endereço de destino, utilizando também o *mhpstubs.jar* que foi copiado previamente no passo 1.6.

2. A classe *ProjectXletGenerator* é responsável por utilizar as informações armazenadas na entidade *ProjectXlet* para criar o arquivo essencial para uma aplicação MHP, que é a classe Java que implementa a interface *javax.tv.xlet.Xlet.*
3. A classe abstrata *GenericComponentGenerator* é responsável por disponibilizar métodos de geração de código relativos a um componente gráfico, utilizando-se dos valores de um *GenericComponent*. A propriedades *AttributeDeclaration*, *MethodCall, MethodDeclaration, TypeString* possuem valores que são utilizados durante a geração de código.
4. A classe *ScreenGenerator* herda de *GenericComponentGenerator* e é utilizada para gerar o código de uma classe que herda de *TVDesignerScreen* utilizando-se dos valores armazenados em um Screen. Assim como um *Screen* possui uma lista de Componentes, o *ScreenGenerator* possui uma lista de *GenericComponentsGenerator* que utiliza na geração da classe.
5. A classe *GenericComponentGeneratorFactory* utiliza o padrão *Factory Method* [15], sendo o único meio de adquirimos uma instância de um elemento que herda de *GenericComponentGenerator*.
6. A classe *FileGeneratorParameters* representa uma abstração utilizada com os dados necessários para criar um arquivo de texto.
7. A classe *FileGenerator* utiliza um *FileGeneratorParameters* e executa a operação de criação e escrita do novo arquivo.
8. A classe *TVDesignerScreenGenerator* gera a classe *TVDesignerScreen*, que é uma classe Java que estende de *org.havi.ui.HContainer,* e cria o conceito de tela para aplicações TVD. Esta classe será herdada por cada tela que for gerada pela aplicação.

9. A classe *TVDesignerKeyListenerGenerator* gera a classe *TVDesignerKeyListener*, que é o *listener* da aplicação gerada. Todos os componentes possuem o mesmo *listener* unificando o tratamento de eventos da aplicação.
10. As classes *HIconGenerator*, *HButtonGenerator* e *HTextGenerator* herdam de *GenericComponentGenerator* e utilizam-se de um *HIcon, HButton e HText* respectivamente, para gerar o código que será utilizado dentro de uma tela gerada.

4.1.6. GUI

O *Graphical User Interface* (GUI) é o módulo responsável pela apresentação da solução *TVDesigner*. Implementado como uma aplicação desktop a GUI possibilita a interação do usuário para a criação das aplicações para TVD MHP, utilizando-se de facilidades como adição de componentes gráficos na tela, *drag and drop* entre outros. Através de recursos gráficos, este manipula as entidades de negócio, do módulo *Entites*, simplificando e agilizando o processo de produção.

4.2. Estrutura de Classes Gerada

Por se tratar de uma aplicação que será executada por *set-top boxes,* que possuem um baixo poder de processamento com o objetivo de diminuir os custos, as aplicações devem ser simples e pequenas, sem um grande número de entidades, nem possuir uma arquitetura complexa. Levando-se em conta estes fatores, a estrutura de classes utilizada para a criação de aplicações MHP, no *TVDesigner,* é ilustrada na figura 4.3.

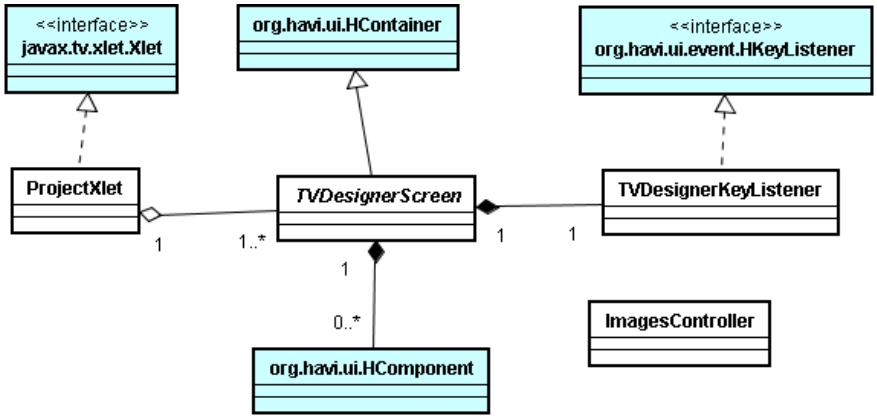

Figura 4.3: Diagrama de classes da aplicação MHP gerada.

O itens em azul são entidades já existentes na especificação sendo o *javax.tv.xlet.Xlet* a interface de entrada de uma aplicação MHP e as outras entidades fazendo parte do especificação do HAVi.

A entidade *ProjectXlet* implementa a interface do *Xlet* e possui uma lista de telas (*TVDesignerScreen*) estas estendem de *org.havi.ui.HContainer* para suportarem a adição de componentes visuais. O *TVDesignerKeyListener* implementa a interface *org.havi.ui.HKeyListener* e é o único *listener* de ações do controle remoto de toda aplicação. Esta abordagem unifica o tratamento de eventos na aplicação com o objetivo de facilitar o controle do fluxo da mesma. Por fim temos o *ImagesController,* que auxilia o carregamento de imagens nas aplicações MHP.

5. ESTUDO DE CASO

Neste capitulo será realizado um estudo de caso na criação de uma aplicação simples para TV digital em MHP. Faremos um comparativo com a implementação utilizado o método clássico de desenvolvimento e utilizando o *TVDesigner*.

5.1. Protótipo

Como estudo de caso realizaremos a implementação de uma aplicação simples de exibição de conteúdo, na forma de textos e imagens, ao usuário. Esta aplicação, que daremos o nome de *BirdsTV,* tem o objetivo de disponibilizar informações sobre pássaros ao usuário. A solução proposta possui a estrutura ilustrada na figura 51.

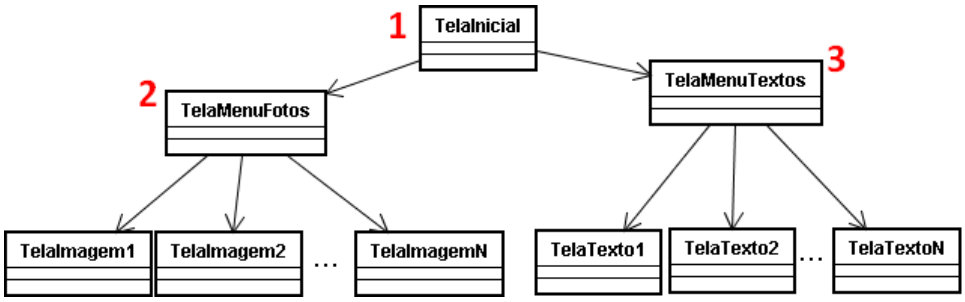

Figura 5.1: Diagrama de telas do protótipo

Como ilustrado acima, a aplicação é composta de (1) uma tela inicial que contem um menu para que possa ser escolhido o tipo de conteúdo a ser exibido (texto ou imagem). Após a escolha de um tipo de conteúdo a ser exibido, aparecerá (2, 3) uma tela que permitirá a escolha do conteúdo, do pássaro selecionado, a ser exibido. Escolhido o conteúdo este será exibido. Esta proposta mesmo sendo simples abrange os recursos disponíveis aos usuários de TV digital como navegação e disposição de informações.

Sabendo que esta aplicação é especifica a um grupo de pessoas interessadas no tema pássaro, esta funcionará durante o período de transmissão do programa televisivo que possui o mesmo tema. Dessa forma a aplicação será

produzida tentando não obstruir o vídeo que está sendo exibido ao fundo. Os menus serão posicionados nos nas margens laterais da tela.

5.2. Utilizando o TVDesigner

Para implementarmos o *BirdsTV*, proposto acima, utilizaremos o *TVDesigner*. No decorrer do processo de criação serão mostradas as particularidades e funcionalidades da ferramenta. Primeiramente teremos uma visão geral da solução ilustrada na figura 5.2:

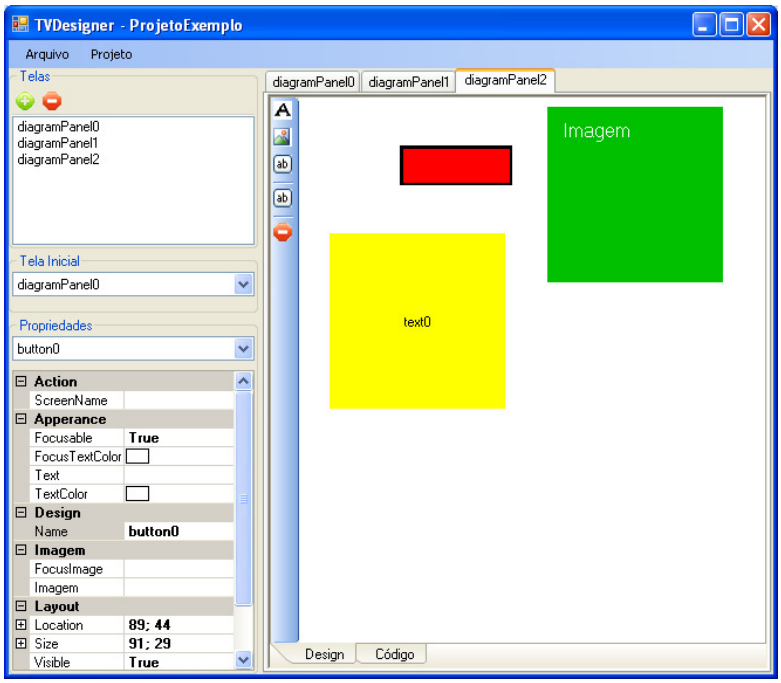

Figura 5.2: Visão geral da aplicação TVDesigner.

5.2.1. Passo 1 – Criação do Projeto

Ao executarmos a aplicação *TVDesigner* devemos acessar o Menu e selecionarmos a opção Novo Projeto, que atende o RF02. O passo 1 é ilustrado na figura 5.3:

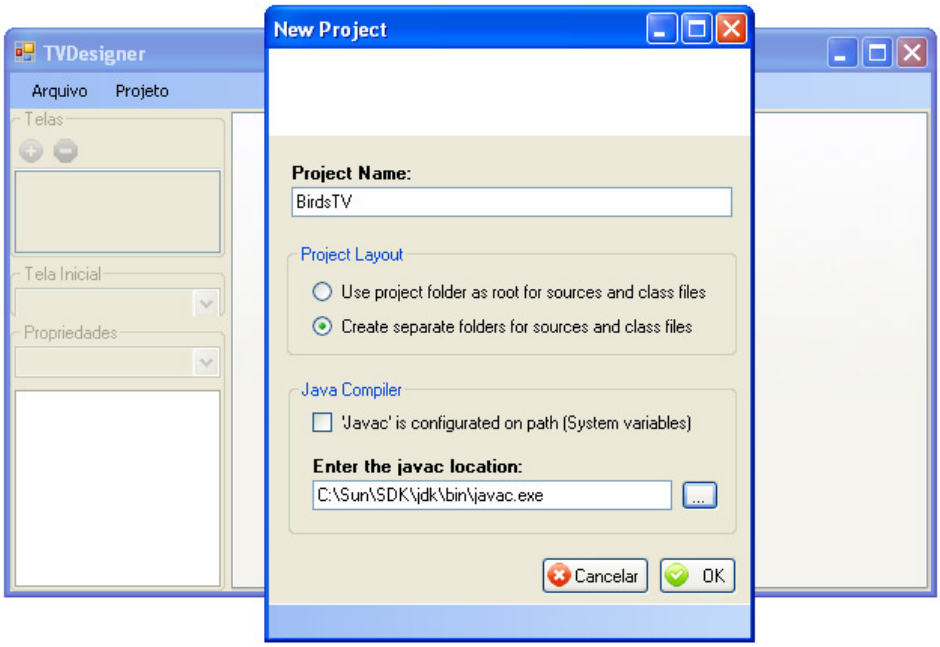

Figura 5.3: Tela de criação de projeto.

Neste passo é necessária a configurações de informações especificas da máquina que executa o *TVDesigner*, como o caso da localização do compilador Java, este será necessário no processo de geração da aplicação MHP. Posteriormente estas configurações também podem ser ajustadas dentro do Projeto. Feitas as configurações e confirmada a ação o projeto é criado.

5.2.2. Passo 2 – Criação das Telas

Após o projeto criado passaremos para etapa de criação das telas da aplicação. Esta ferramenta tem o objetivo, nesta proposta, de abstrair conceitos de programação, no caso MHP, do usuário da solução. Dispondo de recursos gráficos que permitem a criação das telas de uma aplicação e adição de componentes. A figura 5.4 ilustra esta etapa:

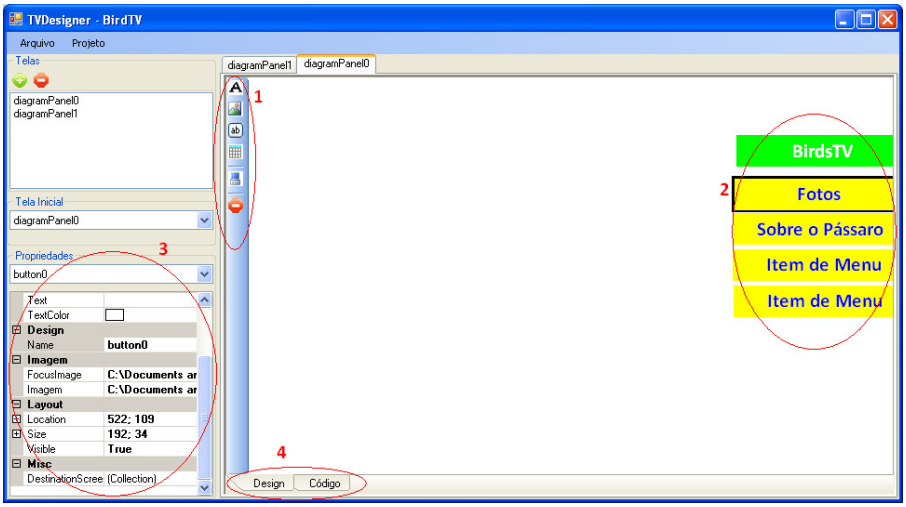

Figura 5.4: Exibição de uma tela sendo produzida.

Como ilustrado na figura acima, o *TVDesigner* incorpora muitas facilidades no processo de implementação. No item 1 estão dispostos os componentes gráficos, satisfazendo o RF03, que podem ser exibidos em uma tela, como botões, tabelas, imagens. No item 2 estão os componentes gráficos adicionados, que incorporam imagens (RF05, RF07). O fundo em brando representa o vídeo que estará sendo exibido no aparelho de televisão. Na região 3 são exibidos os valores do componente selecionado (RF04), possibilitando sua edição. E o item 4 possibilita a visualização do código produzido a partir das mudanças realizadas visualmente, satisfazendo o RF06. Nesta interação da solução proposta à manipulação do código ainda não está disponível, mas esta é uma funcionalidade desejada. Nesta imagem criamos a tela inicial da aplicação *BirdsTV* que possui um menu para o telespectador acessar a informações. A produção de todas as telas da aplicação segue mesmo padrão (Criar tela, adicionar componentes gráficos, configurar valores das propriedades dos componentes gráficos e criar relacionamentos de navegação entre telas), por isso não serão mostradas.

5.2.3. Passo 3 – Geração da Aplicação MHP

Depois de toda a modelagem das telas e configurações dos componentes, podemos gerar a aplicação MHP. Esta opção está ilustrada na Figura 5.5.

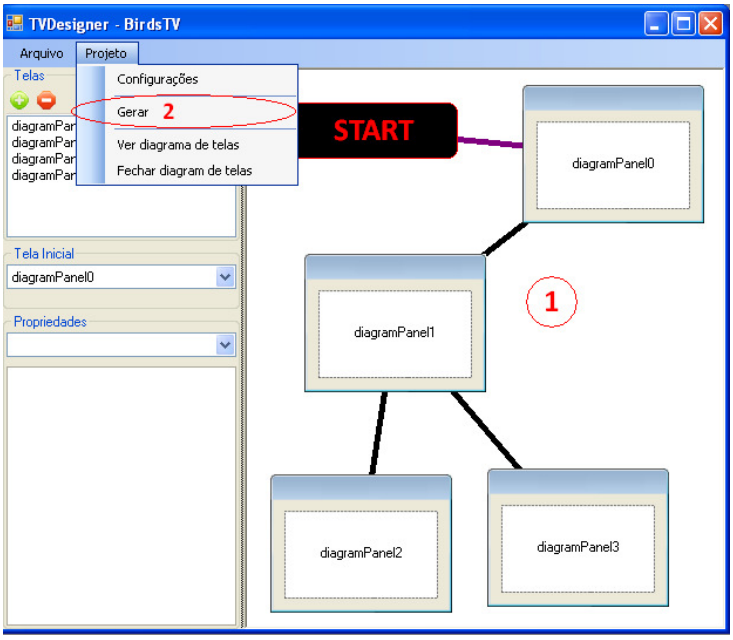

Figura 5.5: Menu Projeto e Diagrama de relacionamento da telas.

No item 1 está demonstrado o diagrama de relacionamento entres as telas, satisfazendo o RF02. No item 2 temos a opção de geração da aplicação de TVD MHP. As atividades para a geração foram mencionadas no item 4.1.2.

5.3. Aplicação Gerada

Após todo o processo de criação e configuração das telas, a aplicação gerada pela ferramenta é composta de um conjunto de arquivos Java. A seguir, mostraremos um pequeno trecho ilustrativo do código gerado para a aplicação *BirdsTV*:

```
package birdstv;

//imports
...

public class BirdsTV extends HContainer implements Xlet {

      public static final int TELAMENU = 0;
      public static final int TELAMENUIMAGENS = 1;
      public static final int TELAMENUTEXTOS = 2;
      public static final int TELAIMAGEM1 = 3;
      public static final int TELAIMAGEM2 = 4;
```

```
	public static final int TELATEXTO1 = 5;
	public static final int TELATEXTO2 = 6;

	protected HScene scene;
	protected XletContext context;
	protected TVDesignerScreen[] screens;
	protected TVDesignerScreen currentScreen;
	protected TVDesignerScreen previousScreen;
	protected TVDesignerKeyListener listener;
	protected AWTVideoSize awtVideoSize;


	...

	public void startXlet() throws XletStateChangeException {
		this.init();

		this.screens = new TVDesignerScreen[7];
		this.listener = new TVDesignerKeyListener(this);
		this.screens[0] = new TelaMenu(this);
		this.screens[1] = new TelaMenuImagens(this);
		this.screens[2] = new TelaMenuTextos(this);
		this.screens[3] = new TelaImagem1(this);
		this.screens[4] = new TelaImagem2(this);
		this.screens[5] = new TelaTexto1(this);
		this.screens[6] = new TelaTexto2(this);

		this.add(screens[0]);
		this.add(screens[1]);
		this.add(screens[2]);
		this.add(screens[3]);
		this.add(screens[4]);
		this.add(screens[5]);
		this.add(screens[6]);

		this.setCurrentScreen(this.screens[0]);

	}

	public void initXlet(XletContext arg0) throws
XletStateChangeException {
		this.context = arg0;
	}

	public void destroyXlet(boolean arg0) throws XletStateChangeException
{
		// TODO Auto-generated method stub
	}

	public void pauseXlet() {
		// TODO Auto-generated method stub
	}

	public void setCurrentScreen(int screen) {
		this.previousScreen = this.currentScreen;
		this.setCurrentScreen(this.screens[screen]);
	}

	public void setCurrentScreen(TVDesignerScreen newCurrentScreen) {
```

```
            if (this.currentScreen != null) {
                  this.currentScreen.setVisible(false);
            }
            this.currentScreen = newCurrentScreen;
            this.currentScreen.setVisible(true);
      }

      ...
}
```

5.4. Executando Aplicação Gerada

Após a criação, o código é compilado e podemos executá-lo. Para os momentos iniciais do desenvolvimento podemos testar aplicações utilizando um simulador chamado XletView [18]. Posteriormente, com a proximidade da etapa de implantação as aplicações MHP devem ser testadas em um ambiente real, com o intuito de tornar os testes o mais realista possível. A seguir são ilustradas algumas figuras da aplicação gerada.

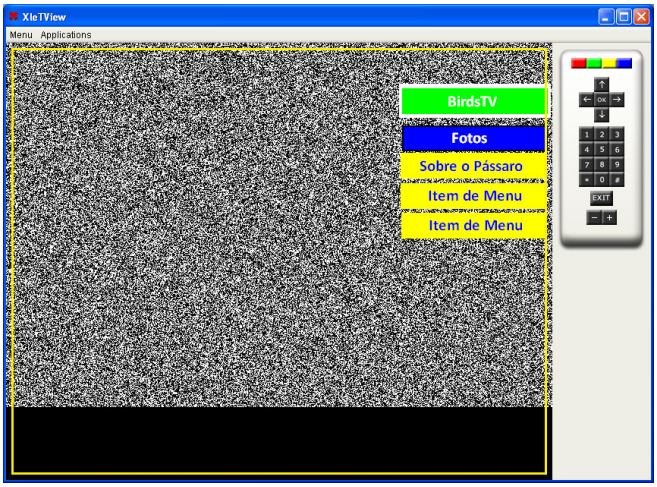

Figura 5.6: Tela Inicial do BirdsTV.

Esta tela possui o nome da aplicação e menu para escolha do conteúdo que deseja ser visualizado. Ao fundo está o vídeo do canal, no caso deste simulador o vídeo não é transmitido.

Figura 5.7: Tela de Menu de imagens.

Esta tela possui um menu de imagens que deve ser escolhido para visualizar a figura do pássaro em tamanho grande. Sempre tentando evitar obstruir a visibilidade do vídeo ao fundo, por isso os componentes gráficos estão localizados nos cantos da tela.

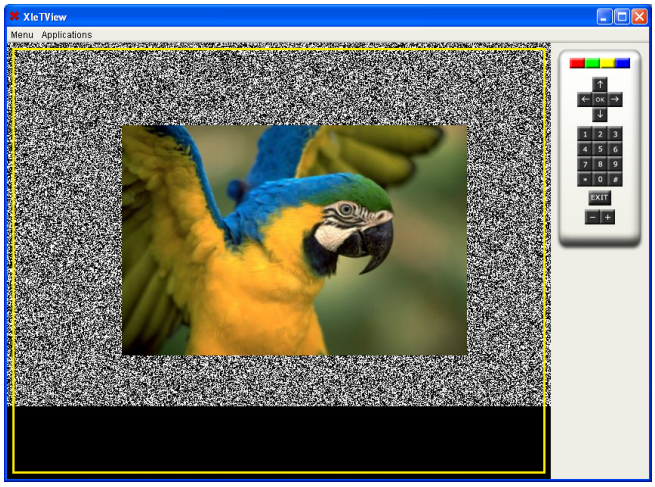

Figura 5.8: Tela de visualização de um foto grande de um pássaro.

No caso desta tela, como o usuário explicitamente, através da navegação da aplicação, desejou visualizar a foto em tamanho aumentado, A imagem é exibida cobrindo o vídeo de fundo.

5.5. Considerações

A utilização do *TVDesigner* possibilitou um aumento na produtividade na criação e manipulação das telas e dos elementos gráficos de uma aplicação. No caso do *BirdsTV* o processo de desenvolvimento da aplicação levou alguns minutos (levando-se em consideração que todos as imagens e informações já estavam disponíveis), onde algumas telas foram criadas, botões formaram menus navegáveis e imagens e textos sobre os pássaros foram exibidos sem a utilização de código para isto.

Este mesmo desenvolvimento feito de forma clássica (utilizando código) levaria um tempo para criação do projeto em um ambiente de desenvolvimento, configuração dos elementos gráficos nas telas (elementos gráficos diferentes possuem classes diferentes que os representam, não se podem utilizar o mesmo elemento de exibição de um texto para uma imagem). A utilização dos componentes diferenciados já faz com que o código gerado utilize a melhor opção para cada caso.

Com a ferramenta uma pessoa sem grandes conhecimentos técnicos sobre o assunto poderia facilmente criar a aplicação, enquanto no desenvolvimento utilizando código é necessária uma pessoa ou talvez uma equipe capacitada no desenvolvimento de aplicações MHP para realizar a mesma tarefa.

Contudo a ferramenta ainda está em fase inicial e impende que regras de negócio possam ser adicionadas e outros tipos de mudanças mais específicas possam ser realizadas. Estes fatores devem ser levados em conta na hora de se iniciar a implementação, mas o *TVDesigner* se mostrou bastante útil na criação de aplicações interativas e pode ser utilizado como ponto de partida para aplicações mais complexas.

6. CONCLUSÕES

Durante o trabalho foram apresentados conceitos sobre televisão digital e o as necessidades de mercado. Foram abordados conceitos sobre o processo de desenvolvimento e um processo adaptado existente para aplicações em TV digital. Este que se mostrou adequado as necessidades de interação constante com o cliente e de agilizar o desenvolvimento.

Apresentamos o *TVDesigner* mencionando suas características e funcionalidades. Em seguida, apresentamos sua estrutura mais detalhada. Posteriormente realizamos um estudo de caso, com o intuito de comprovar a eficácia da solução proposta, analisando os seus pontos fortes e fracos.

A etapa de implementação de soluções para TVD conseguiu ser acelerada utilizando a ferramenta proposta, com isso aumentando a produtividade. A ferramenta não soluciona todos os problemas, mas com isso esta torna-se o pontos de partida para outras iniciativas.

6.1. Trabalhos Futuros

Tivemos como objetivo propor uma ferramenta capaz de auxiliar algumas tarefas durante o processo de desenvolvimento. Ao que inicialmente a ferramenta se propôs, o *TVDesigner* cumpre todos os requisitos.

Da mesma forma que vários *softwares* são produzidos de forma incremental, este passou por apenas mais uma interação com esta entrega. Muitas melhorias e novas funcionalidades podem ser adicionadas a ferramenta para torná-la mais completa e que facilite ainda mais o desenvolvimento de aplicações para TVD.

Dentre as funcionalidades que podem ser adicionadas com trabalhos futuros citamos algumas:

- Permitir manipulação de código, que reflitam na interface gráfica. Isto também possibilitará a existência de lógicas mais complexas para as aplicações;
- Suporte a geração de aplicações para outras plataformas como OpenTV, Ginga-J, Ginga-NCL, entre outras;

- Integração com simuladores;
- Geração de documentação baseada nos comentários feitos no código fonte;
- Adição de mais componentes gráficos e possibilitar a criação de componentes próprios;
- Maior suporte a áudio e vídeo.

REFERÊNCIAS

[1]. Interactive TV Web, http://www.interactivetvweb.org/

[2]. MORRIS, S.; SMITH-CHAIGNEAU, A. Interactive TV Standards: A Guide to MHP, OCAP, and JavaTV. Focal Press Inc. 2005

[3]. TV Without Borders, http://www.tvwithoutborders.com/

[4]. Multimedia Home Platform Specification, http://www.mhp.org/

[5]. The Digital Video Broadcasting Project, http://www.dvb.org/

[6]. VRBA, V; CVRK, L; SÝKORA, M, Framework for digital TV applications. In: International Conference on Networking, International Conference on Systems and International Conference on Mobile Communications and Learning Technologies, 2006, New York. **Proceedings**… Washington, DC: IEEE Computer Society, 2006. P. 184.

[7]. FILGUEIRAS, L. V. L.; ALMAS, A.; SCHLITTLER-SILVA, J. P. A.; et al . Processos de software para a TV interativa. In: IV Fórum de Oportunidades em Televisão Digital Interativa (TVDI'2006), 2006, Poços de Caldas MG. Anais do IV Fórum de Oportunidades em Televisão Digital Interativa. Poços de Caldas, MG: Pontifícia Universidade Católica de Minas Gerais,, 2006.

[8]. KUNERT, Tibor (2003): Interaction Design Patterns in the Context of Interactive TV Applications. In: Proceedings of IFIP INTERACT03: Human-Computer Interaction 2003, Zurich, Switzerland. p. 691.

[9]. STOBBE, A; JUST, T, “IT, telecoms & New Media: The dawn of technological convergence”, E-conomics Deutsche Bank Research, No 56, May, 2006.

[10]. Multimedia Home Platform. Complete set of Open Middleware Standards for Interactive TV. Fact sheet 06/08, disponível em http://www.mhp.org.

[11]. BERGIN, H, Sterring the Future of Interactive TV with MHP. GEM The Key To Common Platform For US iTV. Digital Video Broadcasting.

[12]. HANSEN, B, Design for Interactive television. British Broadcasting Corporation. 2005.

[13]. SUN, Java Code Conventions. http://java.sun.com/docs/codeconv/

[14]. SOMMERVILLE, I. Engenharia de Software. 6.ed. Addison-Wesley Pub. Co., São Paulo, 2003.

[15]. Erich Gamma , Richard Helm , Ralph Johnson , John Vlissides, Design patterns: elements of reusable object-oriented software, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, 1995 C.J. KRAAL, Developing for Interactive Services in Digital TV. Irdeto Consultants, Holland.

[16]. VEIGA, E. G; TAVARES, T. A., Um Modelo de Processo para o Desenvolvimento de Programas para TV Digital e Interativa baseado em Metodologias Ágeis. In Primeiro Workshop em Desenvolvimento Rápido de Aplicações, 2007, Porto de Galinhas, Pernambuco.

[17]. BERGIN, H, MHP & Next Generation DVB Comes to Brazil. Digital Video Broadcasting.

[18]. XletView, Disponível em http://xletview.sourceforge.net/

APÊNDICE I - ENTREVISTA

Neste apêndice, será mostrada a entrevista realizada com engenheiros de software e designers de uma empresa de software do Recife. As pessoas entrevistadas são especialistas no desenvolvimento de aplicações para TV digital.

O objetivo desta entrevista é de tentar elicitar as necessidades que devem ser supridas e funcionalidades disponibilizadas por uma ferramenta que auxilie o processo de desenvolvimento.

1. Perguntas

- Qual a sua posição na empresa?

Sobre TV digital.

- A quanto tempo trabalha com televisão digital?
- Quais os tipos de tecnologias você conhece?
- Quais os tipos de tecnologias você trabalha/trabalhou?
- Quais tipos de aplicações você já desenvolveu?
- Quais as dificuldades encontradas no desenvolvimento?
- Quais os pontos positivos no processo de desenvolvimento?
- Quais as funcionalidades seriam interessantes para facilitar/agilizar a etapa de implantação?

APÊNDICE II – DETALHAMENTO DOS CASOS DE USO

Neste apêndice, será mostrada a definição de alguns termos e conceitos que serão utilizados. Em seguida ilustraremos o diagrama de funcionalidades do *TVDesigner* e por fim mostraremos o detalhamento dos caso de uso da aplicação.

2. Conceitos

1.1. Identificação dos Casos de Uso

Os casos de uso devem ser identificados com um identificador único. A numeração inicia com o identificador [UC01] e prossegue sendo incrementada à medida que forem surgindo novos casos de uso.

Os fluxos alternativos devem ser identificados com um identificador inicado por [FA01] e prossegue sendo incrementado à medida que forem surgindo novos fluxos alternativos dentro de um caso de uso. A contagem é reiniciada para cada caso de uso.

1.2. Prioridades dos Casos de Uso

A prioridade de cada caso de uso pode ser classificada como “essencial”, “importante” e “desejável”, de acordo com a descrição abaixo:

- **Essencial:** O caso de uso é imprescindível à aplicação. Se não for atendido ocasiona o impedimento do funcionamento da aplicação.
- **Importante:** Se não atendido não ocasiona o impedimento de funcionamento da aplicação, mas causa o funcionamento de forma insatisfatória.
- **Desejável:** Sua ausência não compromete a aplicação, são funcionalidades consideradas extras. Podem ser deixados para versões posteriores da solução, caso não haja tempo hábil para implementá-los na versão especificada.

3. Atores

A solução possuirá como ator a entidade **Usuário,** que representa a pessoa que utilizará a solução. Podendo ser programados, designers ou até uma pessoa não envolvida com o desenvolvimento de aplicações para TV digital.

4. Casos de uso

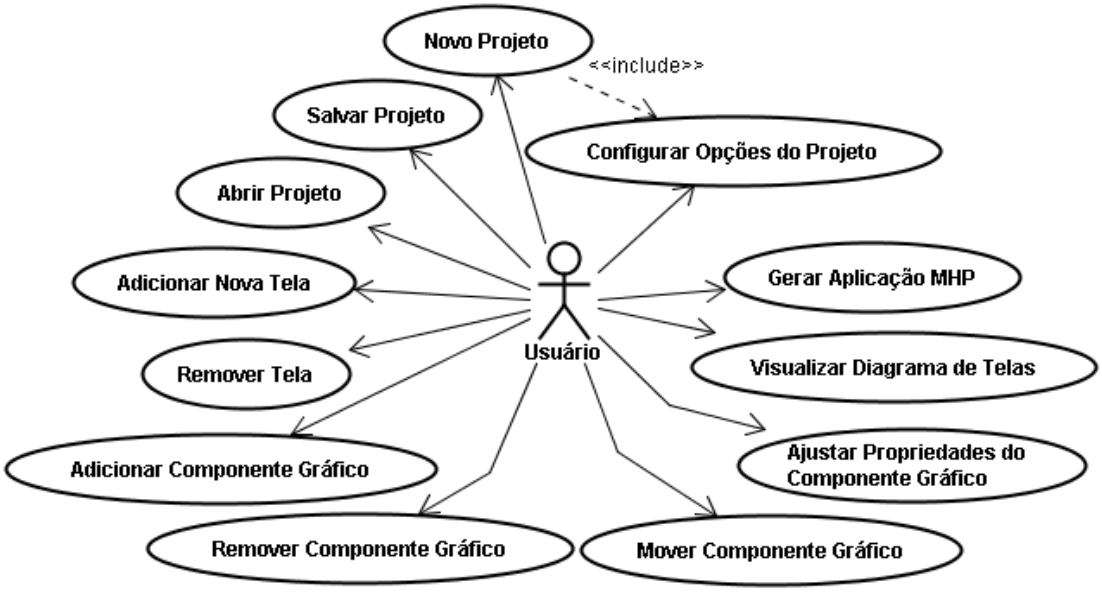

Figura A. 1: Diagrama de casos de uso

- **[UC01] Novo Projeto**

Prioridade	Essencial
Ator(es)	Usuário
Descrição	Funcionalidade que permite a criação de Projeto no *TVDesigner*
Pré-Condições	NA
Pós-Condições	Projeto criado no *TVDesigner*
Fluxo de Eventos	
Fluxo Principal	1. O usuário seleciona a opção 'Novo Projeto'; 2. Configura as Opções de projeto [UC02];

	3. Confirmar criação do projeto.

- **[UC02] Configurar Opções do Projeto**

Prioridade	Essencial
Ator(es)	Usuário
Descrição	Funcionalidade que permite a configuração das opções do Projeto
Pré-Condições	Existir um projeto aberto no *TVDesigner*
Pós-Condições	Alterações das configurações realizadas com sucesso
Fluxo de Eventos	
Fluxo Principal	1. O usuário seleciona a opção 'configurações do projeto'; 2. Altera os valores desejados; 3. Confirma alteração dos valores [FA01].
[FA01]	1. O usuário cancela alteração

- **[UC03] Adicionar Nova Tela**

Prioridade	Essencial
Ator(es)	Usuário
Descrição	Esta funcionalidade permite a adição de uma nova tela a um projeto no *TVDesigner*
Pré-Condições	Existir um projeto aberto
Pós-Condições	Tela adicionada com sucesso
Fluxo de Eventos	
Fluxo Principal	1. O usuário seleciona a opção 'Adicionar tela'; 2. A tela é criada; 3. A tela é mostrada para edição ao usuário.

- **[UC04] Remover Tela**

Prioridade	Essencial

Ator(es)	Usuário
Descrição	Esta funcionalidade permite a remoção de uma tela de um projeto no *TVDesigner*
Pré-Condições	Existir uma tela selecionada na lista de telas
Pós-Condições	Tela removida com sucesso; Remoção dos relacionamentos vinculados a tela removida.
Fluxo de Eventos	
Fluxo Principal	1. O usuário seleciona a tela que deseja remover; O usuário seleciona a opção 'Remover'.

- **[UC05] Adicionar Componente Gráfico**

Prioridade	Essencial
Ator(es)	Usuário
Descrição	Esta funcionalidade permite a adição de um componente gráfico a uma tela do Projeto
Pré-Condições	Existir uma tela sendo exibida
Pós-Condições	Componente Gráfico adicionado com sucesso
Fluxo de Eventos	
Fluxo Principal	1. O usuário seleciona o tipo de componente gráfico que deseja adicionar; 2. O usuário posiciona onde deseja criar o novo componente gráfico; 3. Confirma a criação do componente gráfico; 4. Componente gráfico criado com sucesso.

- **[UC06] Remover Componente Gráfico**

Prioridade	Essencial
Ator(es)	Usuário
Descrição	Esta funcionalidade permite a remoção de um componente gráfico de uma tela do Projeto
Pré-Condições	Possuir um componente gráfico selecionado

Pós-Condições	Componente Gráfico removido com sucesso; Remoção dos relacionamentos vinculados ao componente gráfico.
Fluxo de Eventos	
Fluxo Principal	1. O usuário seleciona o componente gráfico; 2. O usuário seleciona a opção de 'Remover';.

- **[UC07] Mover Componente Gráfico**

Prioridade	Essencial
Ator(es)	Usuário
Descrição	Esta funcionalidade permite o deslocamento de um componente gráfico de uma tela
Pré-Condições	Possuir um componente gráfico selecionado
Pós-Condições	Componente Gráfico movido para nova posição
Fluxo de Eventos	
Fluxo Principal	1. O usuário utiliza *drag and drop* para mover o componente gráfico; 2. O usuário libera o componente gráfico; 3. O componente gráfico assume o novo posicionamento

- **[UC08] Ajustar Propriedades do Componente Gráfico**

Prioridade	Essencial
Ator(es)	Usuário
Descrição	Esta funcionalidade permite a manipulação dos valores das propriedades de um componente gráfico
Pré-Condições	Possuir um componente gráfico selecionado
Pós-Condições	Novos valores das propriedades são alterados no componente gráfico
Fluxo de Eventos	
Fluxo Principal	1. O usuário seleciona uma propriedade que deseja alterar o valor;

	2. O usuário altera o valor desta propriedade; 3. O usuário confirma a alteração do valor desta propriedade [FA01] [FA02]
[FA01]	1. O usuário adiciona um valor inválido; 2. A propriedade não é alterada.
[FA02]	1. O usuário cancela a alteração dos valores; 2. A propriedade não é alterada.

- **[UC09] Visualizar Diagrama de Telas**

Prioridade	Essencial
Ator(es)	Usuário
Descrição	Esta funcionalidade permite visualizar o relacionamento das telas entre as telas existentes em um projeto do *TVDesigner*
Pré-Condições	Possuir um projeto aberto no *TVDesigner*
Pós-Condições	É exibido o diagrama de relacionamento entre as telas
Fluxo de Eventos	
Fluxo Principal	1. O usuário seleciona a opção 'Ver diagrama de telas'; 2. O diagrama é exibido.

- **[UC10] Salvar Projeto**

Prioridade	Essencial
Ator(es)	Usuário
Descrição	Esta funcionalidade permite salvar um projeto criado, no *TVDesigner*, para posterior manipulação
Pré-Condições	Possuir um projeto aberto no *TVDesigner*
Pós-Condições	O projeto é salvo em um arquivo com sucesso
Fluxo de Eventos	
Fluxo Principal	1. O usuário seleciona a opção 'Salvar Projeto'; 2. O usuário seleciona o local de destino onde será salvo o projeto;

	3. O usuário confirma para Salvar o projeto [FA01];
[FA01]	1. O usuário cancela opção; 2. O projeto não é salvo.

- **[UC11] Abrir Projeto**

Prioridade	Essencial
Ator(es)	Usuário
Descrição	Esta funcionalidade permite abrir um projeto criado no *TVDesigner* para edição
Pré-Condições	NA
Pós-Condições	O projeto é aberto para edição no *TVDesigner*
Fluxo de Eventos	
Fluxo Principal	1. O usuário seleciona a opção 'Abrir Projeto'; 2. O usuário selecionado o arquivo de projeto criado pelo *TVDesigner*; 3. O usuário confirma ação de 'Abrir Projeto' [FA01];
[FA01]	3. O usuário cancela a operação; 4. O projeto não é aberto.

- **[UC12] Gerar Aplicação MHP**

Prioridade	Essencial
Ator(es)	Usuário
Descrição	Esta funcionalidade permite a criação de código da aplicação MHP que está sendo modela visualmente pelo *TVDesigner*
Pré-Condições	Possuir um projeto aberto no *TVDesigner*; Possuir uma tela selecionada como tela inicial
Pós-Condições	Pasta com código, imagens utilizadas e arquivos compilados é gerada.
Fluxo de Eventos	
Fluxo Principal	1. O usuário seleciona a opção 'Gerar Projeto'; 2. As pastas são criadas';

	3. Os arquivos Java são criados; 4. As imagens são copiadas para uma pasta destino; 5. É executada uma compilação do código gerado [FA01];
[FA01]	1. Se as configurações de projeto em relação ao compilador Java não estiverem corretas. A compilação não é executada.

APÊNDICE III – CÓDIGOS GERADOS

Neste apêndice, será mostrado o código gerado pela ferramenta, o *TVDesigner*, no desenvolvimento da aplicação *BirdsTV.* O próximos itens representam as classes geradas.

1. BirdsTV

```
package birdstv;


import javax.media.Player;
import javax.tv.media.AWTVideoSize;
import javax.tv.media.AWTVideoSizeControl;
import javax.tv.xlet.XletContext;
import javax.tv.service.selection.ServiceContentHandler;
import javax.tv.service.selection.ServiceContextFactory;
import javax.tv.xlet.Xlet;
import javax.tv.xlet.XletStateChangeException;
import org.havi.ui.HContainer;
import org.havi.ui.HScene;
import org.havi.ui.HSceneFactory;
import org.havi.ui.HScreen;
import java.awt.Rectangle;
import java.awt.Color;


public class BirdsTV extends HContainer implements Xlet {

      public static final int TELAMENU = 0;
      public static final int TELAMENUIMAGENS = 1;
      public static final int TELAMENUTEXTOS = 2;
      public static final int TELAIMAGEM1 = 3;
      public static final int TELAIMAGEM2 = 4;
      public static final int TELATEXTO1 = 5;
      public static final int TELATEXTO2 = 6;

      protected HScene scene;
      protected XletContext context;
      protected TVDesignerScreen[] screens;
      protected TVDesignerScreen currentScreen;
      protected TVDesignerScreen previousScreen;
      protected TVDesignerKeyListener listener;
      protected AWTVideoSize awtVideoSize;


      protected void init() {

        this.scene = HSceneFactory.getInstance().getFullScreenScene(
        HScreen.getDefaultHScreen().getDefaultHGraphicsDevice());

        this.scene.setLayout(null);
```

```
        this.scene.setLocation(0, 0);

        Rectangle rect = scene.getBounds();
            System.out.println(rect.toString());
        this.setBounds(rect);

        this.setVisible(true);
        this.setBackground(Color.black);

            this.scene.add(this);
        this.scene.setVisible(true);

            this.requestFocus();
    }

      public void startXlet() throws XletStateChangeException {
            this.init();

            this.screens = new TVDesignerScreen[7];
            this.listener = new TVDesignerKeyListener(this);
            this.screens[0] = new TelaMenu(this);
            this.screens[1] = new TelaMenuImagens(this);
            this.screens[2] = new TelaMenuTextos(this);
            this.screens[3] = new TelaImagem1(this);
            this.screens[4] = new TelaImagem2(this);
            this.screens[5] = new TelaTexto1(this);
            this.screens[6] = new TelaTexto2(this);

            this.add(screens[0]);
            this.add(screens[1]);
            this.add(screens[2]);
            this.add(screens[3]);
            this.add(screens[4]);
            this.add(screens[5]);
            this.add(screens[6]);

            this.setCurrentScreen(this.screens[0]);

      }

      public void initXlet(XletContext arg0) throws
XletStateChangeException {
            this.context = arg0;
      }

      public void destroyXlet(boolean arg0) throws XletStateChangeException
{
            // TODO Auto-generated method stub
      }

      public void pauseXlet() {
            // TODO Auto-generated method stub
      }

      public void setCurrentScreen(int screen) {
            this.previousScreen = this.currentScreen;
            this.setCurrentScreen(this.screens[screen]);
      }

      public void setCurrentScreen(TVDesignerScreen newCurrentScreen) {
```

```
            if (this.currentScreen != null) {
                  this.currentScreen.setVisible(false);
            }
            this.currentScreen = newCurrentScreen;
            this.currentScreen.setVisible(true);
      }

      public TVDesignerScreen getPreviousScreen() {
            return this.previousScreen;
      }

      public TVDesignerKeyListener getListener() {
            return this.listener;
      }

      public TVDesignerScreen getCurrentScreen() {
            return this.currentScreen;
      }


    /**
       * Resize the video.
       *
       * @param size - AWTVideoSize (currentSize, finalSize)
       */
      public void resize(AWTVideoSize size) {

        if (this.awtVideoSize == null) {
            this.awtVideoSize = size;
        } else if (this.awtVideoSize.equals(size)) {
            return;
        }
            ServiceContextFactory serviceContextFactory = null;

            javax.tv.service.selection.ServiceContext serviceContext =
null;

            try {
                  serviceContextFactory =
ServiceContextFactory.getInstance();
                  serviceContext =
serviceContextFactory.getServiceContext(context);
            }
            catch (Exception ex) {
                  ex.printStackTrace();
            }

            if (serviceContext != null) {
                  ServiceContentHandler[] serviceContentHandler =
serviceContext.getServiceContentHandlers();
                  Player player = null;

                  if (serviceContentHandler.length > 0) {

                        player = (Player) serviceContentHandler[0];

                  }

                  if (player != null) {
                        AWTVideoSizeControl awtVideoSizeControl =
```

```
(AWTVideoSizeControl)
player.getControl("javax.tv.media.AWTVideoSizeControl");

                awtVideoSizeControl.setSize(size);
                  }
            }
      }
}
```

2. TVDesignerScreen

```
package birdstv;

import org.havi.ui.HContainer;
import javax.tv.media.AWTVideoSize;

public abstract class TVDesignerScreen extends HContainer {

    protected javax.tv.media.AWTVideoSize awtVideoSize;
    protected TVDesignerScreen lastScreen;
      protected BirdsTV xlet;


    public abstract void keyEvent(java.awt.event.KeyEvent event);

    protected void setLastScreen(TVDesignerScreen lastScreen) {
        this.lastScreen = lastScreen;
    }

    public abstract void setXlet(BirdsTV xlet);
}
```

3. TVDesignerKeyListener

```
package birdstv;


public class TVDesignerKeyListener implements
org.havi.ui.event.HKeyListener {

    public final static int KEY_RED      = 403;
    public final static int KEY_GREEN    = 404;
    public final static int KEY_YELLOW   = 405;
    public final static int KEY_BLUE     = 406;

    private BirdsTV xlet;

    public TVDesignerKeyListener(BirdsTV xlet){
        this.xlet = xlet;
    }

    public void keyTyped(java.awt.event.KeyEvent arg0) {}
    public void keyReleased(java.awt.event.KeyEvent arg0) {}
    public void keyPressed(java.awt.event.KeyEvent event) {
```

```
          if (this.xlet.getCurrentScreen() == null)
                return;

          this.xlet.getCurrentScreen().keyEvent(event);
        System.out.println("KEY_EVENT");
        this.xlet.repaint();
    }
}
```

4. ImagesController

```
package birdstv;

import java.awt.MediaTracker;
import java.awt.Toolkit;
import java.awt.Image;

import org.havi.ui.HContainer;

public class ImagesController extends HContainer {

      private final String path = "data/";

      private static ImagesController instance = null;

      private ImagesController()
      {
      }

      public static ImagesController getInstance() {
            if (instance == null) {
                  instance = new ImagesController();
            }
            return instance;
      }

      public Image getImage(String name) {
            if (name == null) {
                  return null;
            }

            Image image = Toolkit.getDefaultToolkit().getImage(path +
name);

            System.out.println("Carregando imagem: " + path + name + "
...");

            MediaTracker mediaTracker = new MediaTracker(this);
            mediaTracker.addImage(image,0);
          try{
                  mediaTracker.waitForAll();
                }
          catch (InterruptedException ex){
            image = null;
            ex.printStackTrace();
          }
          if (mediaTracker.isErrorAny())
```

```
            image = null;

            if (image == null)
                  System.out.println("Imagem '" + name +"' ta null");
            else
                  System.out.println("Imagem carregada.");

          return image;

      }
}
```

5. TelaMenu

```
package birdstv;

import org.havi.ui.HContainer;

public class TelaMenu extends TVDesignerScreen {

      private org.havi.ui.HIcon[] focusableButtons;
      private int count;
      private org.havi.ui.HStaticIcon picture1;
      private org.havi.ui.HIcon button0;
      private org.havi.ui.HIcon button1;
      private org.havi.ui.HIcon button2;
      private org.havi.ui.HIcon button3;

      public TelaMenu(BirdsTV xlet) {
            super();
            this.xlet = xlet;
            this.init();
      }

      protected void init() {
            this.setVisible(false);
            this.setLocation(0, 0);
            this.setSize(720, 480);
            java.awt.Rectangle source = new java.awt.Rectangle(0, 0, 720,
480);
            java.awt.Rectangle dest = new java.awt.Rectangle(0, 0, 720,
480);
            this.awtVideoSize = new javax.tv.media.AWTVideoSize(source,
dest);

            this.add(getPicture1());
            this.add(getButton0());
            this.add(getButton1());
            this.add(getButton2());
            this.add(getButton3());

            this.getButton0().addKeyListener(this.xlet.getListener());
            this.getButton1().addKeyListener(this.xlet.getListener());
            this.getButton2().addKeyListener(this.xlet.getListener());
            this.getButton3().addKeyListener(this.xlet.getListener());
            this.addKeyListener(this.xlet.getListener());
```

```
            this.focusableButtons = new org.havi.ui.HIcon[4];

            this.focusableButtons[0] = getButton0();
            this.focusableButtons[1] = getButton1();
            this.focusableButtons[2] = getButton2();
            this.focusableButtons[3] = getButton3();


            this.focusableButtons[0].requestFocus();
      }

      public org.havi.ui.HStaticIcon getPicture1() {
            if (this.picture1 == null) {
                  this.picture1 = new org.havi.ui.HStaticIcon();
                  this.picture1.setVisible(true);
                  this.picture1.setLocation(518, 54);
                  this.picture1.setSize(200, 45);
                  java.awt.Image image =
ImagesController.getInstance().getImage("BirdsTVImage.PNG");
                  if (image != null) {
                        this.picture1.setGraphicContent(image,
org.havi.ui.HVisible.NORMAL_STATE);
                  }
            }
            return this.picture1;
      }

      public org.havi.ui.HIcon getButton0() {
            if (this.button0 == null) {
                  this.button0 = new org.havi.ui.HIcon();
                  this.button0.setVisible(true);
                  this.button0.setLocation(522, 109);
                  this.button0.setSize(192, 34);
                  java.awt.Image image =
ImagesController.getInstance().getImage("buttonFoto.PNG");
                  if (image != null) {
                        this.button0.setGraphicContent(image,
org.havi.ui.HVisible.NORMAL_STATE);
                  }
                  java.awt.Image selectedImage =
ImagesController.getInstance().getImage("buttonFotoFocus.PNG");
                  if (selectedImage != null) {
                        button0.setGraphicContent(selectedImage,
org.havi.ui.HVisible.FOCUSED_STATE);
                  }
            }
            return this.button0;
      }

      public org.havi.ui.HIcon getButton1() {
            if (this.button1 == null) {
                  this.button1 = new org.havi.ui.HIcon();
                  this.button1.setVisible(true);
                  this.button1.setLocation(520, 145);
                  this.button1.setSize(192, 34);
                  java.awt.Image image =
ImagesController.getInstance().getImage("buttonTexto.PNG");
                  if (image != null) {
                        this.button1.setGraphicContent(image,
org.havi.ui.HVisible.NORMAL_STATE);
```

```
                }
                java.awt.Image selectedImage =
ImagesController.getInstance().getImage("buttonTextoFocus.PNG");
                if (selectedImage != null) {
                      button1.setGraphicContent(selectedImage,
org.havi.ui.HVisible.FOCUSED_STATE);
                }
          }
          return this.button1;
    }

    public org.havi.ui.HIcon getButton2() {
          if (this.button2 == null) {
                this.button2 = new org.havi.ui.HIcon();
                this.button2.setVisible(true);
                this.button2.setLocation(520, 185);
                this.button2.setSize(192, 34);
                java.awt.Image image =
ImagesController.getInstance().getImage("buttonMenu1.PNG");
                if (image != null) {
                      this.button2.setGraphicContent(image,
org.havi.ui.HVisible.NORMAL_STATE);
                }
                java.awt.Image selectedImage =
ImagesController.getInstance().getImage("buttonMenu1Focus.PNG");
                if (selectedImage != null) {
                      button2.setGraphicContent(selectedImage,
org.havi.ui.HVisible.FOCUSED_STATE);
                }
          }
          return this.button2;
    }

    public org.havi.ui.HIcon getButton3() {
          if (this.button3 == null) {
                this.button3 = new org.havi.ui.HIcon();
                this.button3.setVisible(true);
                this.button3.setLocation(521, 224);
                this.button3.setSize(192, 34);
                java.awt.Image image =
ImagesController.getInstance().getImage("buttonMenu1.PNG");
                if (image != null) {
                      this.button3.setGraphicContent(image,
org.havi.ui.HVisible.NORMAL_STATE);
                }
                java.awt.Image selectedImage =
ImagesController.getInstance().getImage("buttonMenu1Focus.PNG");
                if (selectedImage != null) {
                      button3.setGraphicContent(selectedImage,
org.havi.ui.HVisible.FOCUSED_STATE);
                }
          }
          return this.button3;
    }

    public void keyEvent(java.awt.event.KeyEvent event) {
          switch(event.getKeyCode()) {
                case java.awt.event.KeyEvent.VK_ENTER: {
                      if (this.count == 0) {
```

```
	this.xlet.setCurrentScreen(BirdsTV.TELAMENUIMAGENS);

	this.xlet.getCurrentScreen().setLastScreen(this);
;
				return;
			}
			else if (this.count == 1) {

	this.xlet.setCurrentScreen(BirdsTV.TELAMENUTEXTOS);

	this.xlet.getCurrentScreen().setLastScreen(this);
;
				return;
			}
			break;
		}
		case java.awt.event.KeyEvent.VK_UP: {
			if (this.focusableButtons == null) {
				return;
			}
			this.count--;
			if(this.count < 0) {
				this.count = this.focusableButtons.length-1;
			}
			this.focusableButtons[ this.count ].requestFocus();
			break;
		}
		case java.awt.event.KeyEvent.VK_DOWN: {
			if (this.focusableButtons == null) {
				return;
			}
			this.count = (this.count + 1) %
this.focusableButtons.length;
			this.focusableButtons[this.count].requestFocus();
			break;
		}
		case java.awt.event.KeyEvent.VK_ESCAPE: {
			if (this.lastScreen != null) {
				this.xlet.setCurrentScreen(this.lastScreen);
			}
			break;
		}
	}
	}

	public void setXlet(BirdsTV xlet) {
		this.xlet = xlet;
	}

	public void setVisible(boolean visible) {
		super.setVisible(visible);
		if (this.awtVideoSize != null) {
			this.xlet.resize(this.awtVideoSize);
		}
		this.requestFocus();
		if (this.focusableButtons != null && this.count >= 0 &&
this.count < this.focusableButtons.length) {
			this.focusableButtons[count].requestFocus();
		}
	}
```

```
}
```

6. TelaMenuImagens

```
package birdstv;

import org.havi.ui.HContainer;

public class TelaMenuImagens extends TVDesignerScreen {

      private org.havi.ui.HIcon[] focusableButtons;
      private int count;
      private org.havi.ui.HIcon button1;
      private org.havi.ui.HIcon button2;
      private org.havi.ui.HIcon button3;
      private org.havi.ui.HIcon button4;

      public TelaMenuImagens(BirdsTV xlet) {
            super();
            this.xlet = xlet;
            this.init();
      }

      protected void init() {
            this.setVisible(false);
            this.setLocation(0, 0);
            this.setSize(720, 480);
            java.awt.Rectangle source = new java.awt.Rectangle(0, 0, 720,
480);
            java.awt.Rectangle dest = new java.awt.Rectangle(0, 0, 720,
480);
            this.awtVideoSize = new javax.tv.media.AWTVideoSize(source,
dest);

            this.add(getButton1());
            this.add(getButton2());
            this.add(getButton3());
            this.add(getButton4());

            this.getButton1().addKeyListener(this.xlet.getListener());
            this.getButton2().addKeyListener(this.xlet.getListener());
            this.getButton3().addKeyListener(this.xlet.getListener());
            this.getButton4().addKeyListener(this.xlet.getListener());
            this.addKeyListener(this.xlet.getListener());

            this.focusableButtons = new org.havi.ui.HIcon[4];

            this.focusableButtons[0] = getButton1();
            this.focusableButtons[1] = getButton2();
            this.focusableButtons[2] = getButton3();
            this.focusableButtons[3] = getButton4();


            this.focusableButtons[0].requestFocus();
      }
```

```
	public org.havi.ui.HIcon getButton1() {
		if (this.button1 == null) {
			this.button1 = new org.havi.ui.HIcon();
			this.button1.setVisible(true);
			this.button1.setLocation(589, 4);
			this.button1.setSize(96, 95);
			java.awt.Image image =
ImagesController.getInstance().getImage("araraAzulSmall.jpg");
			if (image != null) {
				this.button1.setGraphicContent(image,
org.havi.ui.HVisible.NORMAL_STATE);
			}
			java.awt.Image selectedImage =
ImagesController.getInstance().getImage("araraAzulSmallFocus.jpg");
			if (selectedImage != null) {
				button1.setGraphicContent(selectedImage,
org.havi.ui.HVisible.FOCUSED_STATE);
			}
		}
		return this.button1;
	}

	public org.havi.ui.HIcon getButton2() {
		if (this.button2 == null) {
			this.button2 = new org.havi.ui.HIcon();
			this.button2.setVisible(true);
			this.button2.setLocation(550, 102);
			this.button2.setSize(140, 95);
			java.awt.Image image =
ImagesController.getInstance().getImage("araraCanindeSmall.jpg");
			if (image != null) {
				this.button2.setGraphicContent(image,
org.havi.ui.HVisible.NORMAL_STATE);
			}
			java.awt.Image selectedImage =
ImagesController.getInstance().getImage("araraCanindeSmallFocus.jpg");
			if (selectedImage != null) {
				button2.setGraphicContent(selectedImage,
org.havi.ui.HVisible.FOCUSED_STATE);
			}
		}
		return this.button2;
	}

	public org.havi.ui.HIcon getButton3() {
		if (this.button3 == null) {
			this.button3 = new org.havi.ui.HIcon();
			this.button3.setVisible(true);
			this.button3.setLocation(559, 201);
			this.button3.setSize(125, 95);
			java.awt.Image image =
ImagesController.getInstance().getImage("araraVermelhaSmall.jpg");
			if (image != null) {
				this.button3.setGraphicContent(image,
org.havi.ui.HVisible.NORMAL_STATE);
			}
			java.awt.Image selectedImage =
ImagesController.getInstance().getImage("araraVermelhaSmallFocus.jpg");
			if (selectedImage != null) {
				button3.setGraphicContent(selectedImage,
```

```
org.havi.ui.HVisible.FOCUSED_STATE);
                }
            }
            return this.button3;
      }

      public org.havi.ui.HIcon getButton4() {
            if (this.button4 == null) {
                  this.button4 = new org.havi.ui.HIcon();
                  this.button4.setVisible(true);
                  this.button4.setLocation(595, 298);
                  this.button4.setSize(91, 140);
                  java.awt.Image image =
ImagesController.getInstance().getImage("papagaioSmall.jpg");
                  if (image != null) {
                        this.button4.setGraphicContent(image,
org.havi.ui.HVisible.NORMAL_STATE);
                  }
                  java.awt.Image selectedImage =
ImagesController.getInstance().getImage("papagaioSmallFocus.jpg");
                  if (selectedImage != null) {
                        button4.setGraphicContent(selectedImage,
org.havi.ui.HVisible.FOCUSED_STATE);
                  }
            }
            return this.button4;
      }

      public void keyEvent(java.awt.event.KeyEvent event) {
            switch(event.getKeyCode()) {
                  case java.awt.event.KeyEvent.VK_ENTER: {
                        if (this.count == 0) {

      this.xlet.setCurrentScreen(BirdsTV.TELAIMAGEM1);

      this.xlet.getCurrentScreen().setLastScreen(this);
;
                              return;
                        }
                        else if (this.count == 1) {

      this.xlet.setCurrentScreen(BirdsTV.TELAIMAGEM2);

      this.xlet.getCurrentScreen().setLastScreen(this);
;
                              return;
                        }
                        break;
                  }
                  case java.awt.event.KeyEvent.VK_UP: {
                        if (this.focusableButtons == null) {
                              return;
                        }
                        this.count--;
                        if(this.count < 0) {
                              this.count = this.focusableButtons.length-1;
                        }
                        this.focusableButtons[ this.count ].requestFocus();
                        break;
                  }
```

```
                  case java.awt.event.KeyEvent.VK_DOWN: {
                        if (this.focusableButtons == null) {
                              return;
                        }
                        this.count = (this.count + 1) %
this.focusableButtons.length;
                        this.focusableButtons[this.count].requestFocus();
                        break;
                  }
                  case java.awt.event.KeyEvent.VK_ESCAPE: {
                        if (this.lastScreen != null) {
                              this.xlet.setCurrentScreen(this.lastScreen);
                        }
                        break;
                  }
            }
      }

      public void setXlet(BirdsTV xlet) {
            this.xlet = xlet;
      }

      public void setVisible(boolean visible) {
            super.setVisible(visible);
            if (this.awtVideoSize != null) {
                  this.xlet.resize(this.awtVideoSize);
            }
            this.requestFocus();
            if (this.focusableButtons != null && this.count >= 0 &&
this.count < this.focusableButtons.length) {
                  this.focusableButtons[count].requestFocus();
            }
      }

}
```

7. TelaMenuTextos

```
package birdstv;

import org.havi.ui.HContainer;

public class TelaMenuTextos extends TVDesignerScreen {

      private org.havi.ui.HIcon[] focusableButtons;
      private int count;
      private org.havi.ui.HIcon button0;
      private org.havi.ui.HIcon button1;
      private org.havi.ui.HIcon button2;
      private org.havi.ui.HIcon button3;

      public TelaMenuTextos(BirdsTV xlet) {
            super();
            this.xlet = xlet;
            this.init();
      }
```

```
	protected void init() {
		this.setVisible(false);
		this.setLocation(0, 0);
		this.setSize(720, 480);
		java.awt.Rectangle source = new java.awt.Rectangle(0, 0, 720,
480);
		java.awt.Rectangle dest = new java.awt.Rectangle(0, 0, 720,
480);
		this.awtVideoSize = new javax.tv.media.AWTVideoSize(source,
dest);

		this.add(getButton0());
		this.add(getButton1());
		this.add(getButton2());
		this.add(getButton3());

		this.getButton0().addKeyListener(this.xlet.getListener());
		this.getButton1().addKeyListener(this.xlet.getListener());
		this.getButton2().addKeyListener(this.xlet.getListener());
		this.getButton3().addKeyListener(this.xlet.getListener());
		this.addKeyListener(this.xlet.getListener());

		this.focusableButtons = new org.havi.ui.HIcon[4];

		this.focusableButtons[0] = getButton0();
		this.focusableButtons[1] = getButton1();
		this.focusableButtons[2] = getButton2();
		this.focusableButtons[3] = getButton3();


		this.focusableButtons[0].requestFocus();
	}

	public org.havi.ui.HIcon getButton0() {
		if (this.button0 == null) {
			this.button0 = new org.havi.ui.HIcon();
			this.button0.setVisible(true);
			this.button0.setLocation(522, 93);
			this.button0.setSize(192, 34);
			java.awt.Image image =
ImagesController.getInstance().getImage("araraAzulMenu.PNG");
			if (image != null) {
				this.button0.setGraphicContent(image,
org.havi.ui.HVisible.NORMAL_STATE);
			}
			java.awt.Image selectedImage =
ImagesController.getInstance().getImage("araraAzulMenuFoco.PNG");
			if (selectedImage != null) {
				button0.setGraphicContent(selectedImage,
org.havi.ui.HVisible.FOCUSED_STATE);
			}
		}
		return this.button0;
	}

	public org.havi.ui.HIcon getButton1() {
		if (this.button1 == null) {
			this.button1 = new org.havi.ui.HIcon();
			this.button1.setVisible(true);
			this.button1.setLocation(522, 129);
```

```
                this.button1.setSize(192, 34);
                java.awt.Image image =
ImagesController.getInstance().getImage("araraCanindeMenu.PNG");
                if (image != null) {
                      this.button1.setGraphicContent(image,
org.havi.ui.HVisible.NORMAL_STATE);
                }
                java.awt.Image selectedImage =
ImagesController.getInstance().getImage("araraCanindeMenuFoco.PNG");
                if (selectedImage != null) {
                      button1.setGraphicContent(selectedImage,
org.havi.ui.HVisible.FOCUSED_STATE);
                }
          }
          return this.button1;
    }

    public org.havi.ui.HIcon getButton2() {
          if (this.button2 == null) {
                this.button2 = new org.havi.ui.HIcon();
                this.button2.setVisible(true);
                this.button2.setLocation(522, 165);
                this.button2.setSize(192, 34);
                java.awt.Image image =
ImagesController.getInstance().getImage("araraVermelhaMenu.PNG");
                if (image != null) {
                      this.button2.setGraphicContent(image,
org.havi.ui.HVisible.NORMAL_STATE);
                }
                java.awt.Image selectedImage =
ImagesController.getInstance().getImage("araraVermelhaMenuFoco.PNG");
                if (selectedImage != null) {
                      button2.setGraphicContent(selectedImage,
org.havi.ui.HVisible.FOCUSED_STATE);
                }
          }
          return this.button2;
    }

    public org.havi.ui.HIcon getButton3() {
          if (this.button3 == null) {
                this.button3 = new org.havi.ui.HIcon();
                this.button3.setVisible(true);
                this.button3.setLocation(522, 200);
                this.button3.setSize(192, 34);
                java.awt.Image image =
ImagesController.getInstance().getImage("papagaioMenu.PNG");
                if (image != null) {
                      this.button3.setGraphicContent(image,
org.havi.ui.HVisible.NORMAL_STATE);
                }
                java.awt.Image selectedImage =
ImagesController.getInstance().getImage("papagaioMenuFoco.PNG");
                if (selectedImage != null) {
                      button3.setGraphicContent(selectedImage,
org.havi.ui.HVisible.FOCUSED_STATE);
                }
          }
          return this.button3;
    }
```

```
	public void keyEvent(java.awt.event.KeyEvent event) {
		switch(event.getKeyCode()) {
			case java.awt.event.KeyEvent.VK_ENTER: {
				if (this.count == 0) {

	this.xlet.setCurrentScreen(BirdsTV.TELATEXTO1);

	this.xlet.getCurrentScreen().setLastScreen(this);
;
					return;
				}
				else if (this.count == 1) {

	this.xlet.setCurrentScreen(BirdsTV.TELATEXTO2);

	this.xlet.getCurrentScreen().setLastScreen(this);
;
					return;
				}
				break;
			}
			case java.awt.event.KeyEvent.VK_UP: {
				if (this.focusableButtons == null) {
					return;
				}
				this.count--;
				if(this.count < 0) {
					this.count = this.focusableButtons.length-1;
				}
				this.focusableButtons[ this.count ].requestFocus();
				break;
			}
			case java.awt.event.KeyEvent.VK_DOWN: {
				if (this.focusableButtons == null) {
					return;
				}
				this.count = (this.count + 1) %
this.focusableButtons.length;
				this.focusableButtons[this.count].requestFocus();
				break;
			}
			case java.awt.event.KeyEvent.VK_ESCAPE: {
				if (this.lastScreen != null) {
					this.xlet.setCurrentScreen(this.lastScreen);
				}
				break;
			}
		}
	}

	public void setXlet(BirdsTV xlet) {
		this.xlet = xlet;
	}

	public void setVisible(boolean visible) {
		super.setVisible(visible);
		if (this.awtVideoSize != null) {
			this.xlet.resize(this.awtVideoSize);
		}
```

```
            this.requestFocus();
            if (this.focusableButtons != null && this.count >= 0 &&
this.count < this.focusableButtons.length) {
                  this.focusableButtons[count].requestFocus();
            }
      }

}
```

8. TelaImagem1

```
package birdstv;

import org.havi.ui.HContainer;

public class TelaImagem1 extends TVDesignerScreen {

      private org.havi.ui.HIcon[] focusableButtons;
      private int count;
      private org.havi.ui.HStaticIcon picture1;

      public TelaImagem1(BirdsTV xlet) {
            super();
            this.xlet = xlet;
            this.init();
      }

      protected void init() {
            this.setVisible(false);
            this.setLocation(0, 0);
            this.setSize(720, 480);
            java.awt.Rectangle source = new java.awt.Rectangle(0, 0, 720,
480);
            java.awt.Rectangle dest = new java.awt.Rectangle(0, 0, 720,
480);
            this.awtVideoSize = new javax.tv.media.AWTVideoSize(source,
dest);

            this.add(getPicture1());

            this.addKeyListener(this.xlet.getListener());

      }

      public org.havi.ui.HStaticIcon getPicture1() {
            if (this.picture1 == null) {
                  this.picture1 = new org.havi.ui.HStaticIcon();
                  this.picture1.setVisible(true);
                  this.picture1.setLocation(93, 96);
                  this.picture1.setSize(410, 400);
                  java.awt.Image image =
ImagesController.getInstance().getImage("araraAzulBig.jpg");
                  if (image != null) {
                        this.picture1.setGraphicContent(image,
org.havi.ui.HVisible.NORMAL_STATE);
                  }
            }
```

```
            return this.picture1;
      }

      public void keyEvent(java.awt.event.KeyEvent event) {
            switch(event.getKeyCode()) {
                  case java.awt.event.KeyEvent.VK_ESCAPE: {
                        if (this.lastScreen != null) {
                              this.xlet.setCurrentScreen(this.lastScreen);
                        }
                        break;
                  }
            }
      }

      public void setXlet(BirdsTV xlet) {
            this.xlet = xlet;
      }

      public void setVisible(boolean visible) {
            super.setVisible(visible);
            if (this.awtVideoSize != null) {
                  this.xlet.resize(this.awtVideoSize);
            }
            this.requestFocus();
            if (this.focusableButtons != null && this.count >= 0 &&
this.count < this.focusableButtons.length) {
                  this.focusableButtons[count].requestFocus();
            }
      }

}
```

9. TelaImagem2

```
package birdstv;

import org.havi.ui.HContainer;

public class TelaImagem2 extends TVDesignerScreen {

      private org.havi.ui.HIcon[] focusableButtons;
      private int count;
      private org.havi.ui.HStaticIcon picture2;

      public TelaImagem2(BirdsTV xlet) {
            super();
            this.xlet = xlet;
            this.init();
      }

      protected void init() {
            this.setVisible(false);
            this.setLocation(0, 0);
            this.setSize(720, 480);
            java.awt.Rectangle source = new java.awt.Rectangle(0, 0, 720,
480);
            java.awt.Rectangle dest = new java.awt.Rectangle(0, 0, 720,
```

```
480);
            this.awtVideoSize = new javax.tv.media.AWTVideoSize(source,
dest);

            this.add(getPicture2());

            this.addKeyListener(this.xlet.getListener());

      }

      public org.havi.ui.HStaticIcon getPicture2() {
            if (this.picture2 == null) {
                  this.picture2 = new org.havi.ui.HStaticIcon();
                  this.picture2.setVisible(true);
                  this.picture2.setLocation(152, 109);
                  this.picture2.setSize(457, 304);
                  java.awt.Image image =
ImagesController.getInstance().getImage("araraCanindeBig.PNG");
                  if (image != null) {
                        this.picture2.setGraphicContent(image,
org.havi.ui.HVisible.NORMAL_STATE);
                  }
            }
            return this.picture2;
      }

      public void keyEvent(java.awt.event.KeyEvent event) {
            switch(event.getKeyCode()) {
                  case java.awt.event.KeyEvent.VK_ESCAPE: {
                        if (this.lastScreen != null) {
                              this.xlet.setCurrentScreen(this.lastScreen);
                        }
                        break;
                  }
            }
      }

      public void setXlet(BirdsTV xlet) {
            this.xlet = xlet;
      }

      public void setVisible(boolean visible) {
            super.setVisible(visible);
            if (this.awtVideoSize != null) {
                  this.xlet.resize(this.awtVideoSize);
            }
            this.requestFocus();
            if (this.focusableButtons != null && this.count >= 0 &&
this.count < this.focusableButtons.length) {
                  this.focusableButtons[count].requestFocus();
            }
      }

}
```

10. TelaTexto1

```
package birdstv;

import org.havi.ui.HContainer;

public class TelaTexto1 extends TVDesignerScreen {

	private org.havi.ui.HIcon[] focusableButtons;
	private int count;
	private org.havi.ui.HText text0;

	public TelaTexto1(BirdsTV xlet) {
		super();
		this.xlet = xlet;
		this.init();
	}

	protected void init() {
		this.setVisible(false);
		this.setLocation(0, 0);
		this.setSize(720, 480);
		java.awt.Rectangle source = new java.awt.Rectangle(0, 0, 720,
480);
		java.awt.Rectangle dest = new java.awt.Rectangle(0, 0, 720,
480);
		this.awtVideoSize = new javax.tv.media.AWTVideoSize(source,
dest);

		this.add(getText0());

		this.addKeyListener(this.xlet.getListener());

	}

	public org.havi.ui.HText getText0() {
		if (this.text0 == null) {
			this.text0 = new org.havi.ui.HText();
			this.text0.setVisible(true);
			this.text0.setLocation(83, 252);
			this.text0.setSize(480, 150);

	this.text0.setHorizontalAlignment(org.havi.ui.HText.HALIGN_LEFT);

	this.text0.setVerticalAlignment(org.havi.ui.HText.VALIGN_TOP);
			this.text0.setTextContent("AmeaÃ§ada de extinÃ§Ã£o, a
arara-azul-grande Ã© o maior psitacÃ-deo do mundo, com 98 centÃ-metros de
comprimento, penas centrais da cauda com 55 cm, 1,5 kg de peso. Bico muito
grande, negro, com aparÃªncia de ser maior que o prÃ³prio crÃ¢nio, sem
dentes na maxila, porÃ©m com pronunciado entalhe na mandÃ-bula. Anel
perioftÃ¡lmico, pÃ¡lpebras e uma faixa na base da mandÃ-bula amarelos. A
lÃ-ngua Ã© negra com uma tarja amarela longitudinal. Os machos e as
fÃªmeas quase nÃ£o apresentam dimorfismo sexual. Os machos normalmente
sÃ£o mais robustos, principalmente no bico, com a cabeÃ§a mais quadrada. A
cauda tambÃ©m Ã© maior. A espÃ©cie Ã© monÃ³gama, permanecendo unidos por
toda a vida. Os ovos sÃ£o redondos com a incubaÃ§Ã£o ao redor de 30 dias.
Os filhotes abandonam o ninho com 15 semanas de idade.",
org.havi.ui.HVisible.ALL_STATES);
```

```
                  java.awt.Font font = new java.awt.Font("Arial Black",
java.awt.Font.PLAIN, 20);
                  this.text0.setFont(font);
                  java.awt.Color color = new java.awt.Color(0);
                  this.text0.setForeground(color);
                  java.awt.Color backgroundColor = new java.awt.Color(0);
                  this.text0.setBackground(backgroundColor);
            }
            return this.text0;
      }

      public void keyEvent(java.awt.event.KeyEvent event) {
            switch(event.getKeyCode()) {
                  case java.awt.event.KeyEvent.VK_ESCAPE: {
                        if (this.lastScreen != null) {
                              this.xlet.setCurrentScreen(this.lastScreen);
                        }
                        break;
                  }
            }
      }

      public void setXlet(BirdsTV xlet) {
            this.xlet = xlet;
      }

      public void setVisible(boolean visible) {
            super.setVisible(visible);
            if (this.awtVideoSize != null) {
                  this.xlet.resize(this.awtVideoSize);
            }
            this.requestFocus();
            if (this.focusableButtons != null && this.count >= 0 &&
this.count < this.focusableButtons.length) {
                  this.focusableButtons[count].requestFocus();
            }
      }

}
```

11. TelaTexto2

```
package birdstv;

import org.havi.ui.HContainer;

public class TelaTexto2 extends TVDesignerScreen {

      private org.havi.ui.HIcon[] focusableButtons;
      private int count;
      private org.havi.ui.HText text1;

      public TelaTexto2(BirdsTV xlet) {
            super();
            this.xlet = xlet;
            this.init();
      }
```

```
	protected void init() {
		this.setVisible(false);
		this.setLocation(0, 0);
		this.setSize(720, 480);
		java.awt.Rectangle source = new java.awt.Rectangle(0, 0, 720,
480);
		java.awt.Rectangle dest = new java.awt.Rectangle(0, 0, 720,
480);
		this.awtVideoSize = new javax.tv.media.AWTVideoSize(source,
dest);

		this.add(getText1());

		this.addKeyListener(this.xlet.getListener());

	}

	public org.havi.ui.HText getText1() {
		if (this.text1 == null) {
			this.text1 = new org.havi.ui.HText();
			this.text1.setVisible(true);
			this.text1.setLocation(116, 252);
			this.text1.setSize(480, 150);

	this.text1.setHorizontalAlignment(org.havi.ui.HText.HALIGN_LEFT);

	this.text1.setVerticalAlignment(org.havi.ui.HText.VALIGN_TOP);
			this.text1.setTextContent("A arara-vermelha (Ara
chloropterus) Ã© uma ave psitaciforme, nativa das florestas do PanamÃ¡ ao
Brasil, Paraguai e Argentina. A sua alimentaÃ§Ã£o Ã© baseada em sementes,
frutas, coquinhos. A arara-vermelha mede atÃ© 90 centÃ-metros de
comprimento e pesa atÃ© 1,5 quilo. Cada postura Ã© composta por ovos de 5
centÃ-metros, incubados por 29 dias. 0 ninho dessa arara Ã© feito em ocos
de Ã¡rvores, mas ela tambÃ©m se aproveita de buracos em paredes rochosas
para colocar os ovos, os quais sÃ£o chocados apenas pela fÃªmea, que fica
no ninho. Quem cuida de garantir a alimentaÃ§Ã£o tanto da fÃªmea como dos
filhotes Ã© o macho, que nessa espÃ©cie Ã© fiel, mantendo a mesma
companheira a vida inteira.", org.havi.ui.HVisible.ALL_STATES);
			java.awt.Font font = new java.awt.Font("Arial Black",
java.awt.Font.PLAIN, 20);
			this.text1.setFont(font);
			java.awt.Color color = new java.awt.Color(0);
			this.text1.setForeground(color);
			java.awt.Color backgroundColor = new java.awt.Color(0);
			this.text1.setBackground(backgroundColor);
		}
		return this.text1;
	}

	public void keyEvent(java.awt.event.KeyEvent event) {
		switch(event.getKeyCode()) {
			case java.awt.event.KeyEvent.VK_ESCAPE: {
				if (this.lastScreen != null) {
					this.xlet.setCurrentScreen(this.lastScreen);
				}
				break;
			}
		}
	}
```

```
	public void setXlet(BirdsTV xlet) {
		this.xlet = xlet;
	}

	public void setVisible(boolean visible) {
		super.setVisible(visible);
		if (this.awtVideoSize != null) {
			this.xlet.resize(this.awtVideoSize);
		}
		this.requestFocus();
		if (this.focusableButtons != null && this.count >= 0 &&
this.count < this.focusableButtons.length) {
			this.focusableButtons[count].requestFocus();
		}
	}

}
```

_____

Prof. Carlos André Guimarães Ferraz

Orientador

_____

Djaci Alves de Araújo Filho

Aluno