Universidade Federal de Santa Catarina
Departamento de Informática e Estatística
Curso de Bacharelado em Ciências da Computação

Marcus Vinicius Cruz Xavier

Trabalho de conclusão de curso

Telis ME: Uma versão de Telis para dispositivos móveis

Orientação: Luiz Fernando Bier Melgarejo

Florianópolis
fevereiro de 2007

Marcus Vinicius Cruz Xavier

Telis ME: uma versão de Telis para dispositivos móveis

Monografia apresentada ao Curso de Ciências da Computação como requisito parcial à obtenção do grau de Bacharel em Ciências da Computação.

Universidade Federal de Santa Catarina.

Orientador: Prof. Dr. Luiz Fernando Bier Melgarejo.

Florianópolis
fevereiro de 2007

Universidade Federal de Santa Catarina
Departamento de Informática e Estatística
Curso de Bacharelado em Ciências da Computação

Telis ME: Uma versão de Telis para dispositivos móveis

Marcus Vinicius Cruz Xavier

BANCA EXAMINADORA:

Prof. Luiz Fernando Bier Melgarejo
Orientador

Anderson Nielson
Banca

Prof. José Mazzuco Júnior
Banca

Florianópolis
fevereiro de 2007

Dedico este trabalho à música, à cerveja, e à mulher, sem as quais ele teria sido feito com o dobro da qualidade e levaria metade do tempo. Ou ele nem existiria...

Meu trabalho de conclusão de curso, foi resultado de cinco anos de intenso aprendizado. Seria impossível agradecer a alguém, sem cometer injustiças.

À todos que me ajudaram nesta jornada, amo todos vocês.

Apenas um agradecimento especial: ao meu orientador, por me mostrar que é possível orientar de maneira humana e efetiva.

Resumo

Assim como o interesse pelo desenvolvimento de aplicações para Internet explodiu alguns anos atrás, decorrente da popularização da rede, o interesse pelo desenvolvimento de aplicações para dispositivos móveis está aumentando sensivelmente, devido principalmente à disseminação dos telefones celulares.

O potencial desta plataforma em vista de sua grande popularidade é enorme, e como no caso da Internet, o desenvolvimento de software para celulares requer também um razoável domínio de conceitos e técnicas.

Telis é uma ferramenta de aprendizagem de programação, desenvolvida com o intuito de simplificar conceitos e técnicas, e integrar-se à tecnologias relacionadas a Internet. O objetivo deste trabalho é criar um modo de executar programas feitos em Telis em dispositivos móveis, mais especificamente em telefones celulares.

Desta forma estudantes sem conhecimento prévio de programação poderão usufruir de um ambiente de aprendizado simples e poderoso para gerar aplicações para seus telefones celulares, como hoje o fazem os alunos de graduação com suas aplicações para Web.

Abstract

Like the interest for Internet application development exploded few years ago, due the popularization of the network, the interest for building mobile devices applications is sensibly growing, due the dissemination of cell phones.

The potential of this platform, considering its popularity is huge, and like is the case of the Internet, the development of software for this platform requires a reasonable domain of concepts and techniques.

Telis is a programming learning tool, developed to simplify programming concepts and techniques, and integrate itself to Internet related technologies. The goal of this work is to create a way of execute programs developed in Telis in mobile devices, specifically in cell phones.

So, students without any previous knowledge of programming techniques, will be able to enjoy a simple and powerful learning environment to create applications to their cell phones, like do graduation students today, with their Web applications.

Sumário

Resumo 5
Abstract 7
Sumário 8
Índice de figuras 9
1. Introdução 10
1.1. Delimitação 12
2. Telis 12
2.1. A linguagem Telis 13
2.2. Conceitos e nomenclaturas de Telis 14
3. Java para dispositivos móveis (JME) 16
3.1. Máquina Virtual Java (JVM) 17
3.2. Bibliotecas de núcleo 17
3.3. MIDP (*Mobile Information Device Profile*) 18
3.3.1. Bibliotecas de manipulação de gráficos 20
4. Telis ME 20
4.1. O compilador 22
4.1.1. Implementação 23
4.1.2. Etapas do processo de transformação 26
4.1.3. Tradução de código 28
4.1.4. Compilação e publicação 30
4.1.5. Exemplo 31
4.2. Máquina Virtual Telis ME 33
4.2.1. Inicialização 34
4.2.2. Ciclo de vida do ator 34
4.2.3. Execução de uma ação 36
4.2.4. Contextos 37
4.2.5. Estímulos 38
4.2.6. Variáveis e agendas 40
5. Conclusão 41
5.1. Trabalhos futuros 42
6. Referências Bibliográficas 43
Anexo I 44

Índice de figuras

Figura 1: formação do núcleo de Java nas diversas distribuições.....18
Figura 2: MIDP disponível sobre CDC e a CLDC 19
Figura 3: Telis é construído sobre JSE. Telis ME sobre JME 20
Figura 4: em Telis ME o código do aplique é distribuído junto ao interpretador.....22
Figura 5: geração de código utilizando herança.....25
Figura 6: fluxo das etapas de transformação 27
Figura 7: Transformação do seDito em uma ação em Telis ME 28
Figura 8: exemplo de transformação de associação de variável 29
Figura 9: resumo do exemplo de transformação.....31
Figura 10: aplique publicado como XML 32
Figura 11: umModelo.java.....32
Figura 12: ExecutorDeAplique.java.....33
Figura 13: algoritmo de busca de instruções 36
Figura 14: classes que implementam a interface Empilhavel.....37
Figura 15: possíveis alterações na pilha de contextos.....38
Figura 16: exemplo ilustrado de ator emitindo estímulo.....40
Figura 1: em Telis ME o código do aplique é distribuído junto ao interpretador.....49
Figura 2: Transformação do seDito em uma ação em Telis ME 49
Figura 3: classes que implementam a interface Empilhavel.....51
Figura 4: possíveis alterações na pilha de contextos.....51

1. Introdução

"A rede Internet, tal como evoluiu nos últimos anos, permitiu inúmeros exemplos de trabalho cooperativo à distância. Tecnologias recentes (como Java), abriram novas perspectivas nessa direção, uma vez que possibilitam um ainda maior dinamismo e interação utilizando a rede.

Infelizmente, o domínio produtivo dessas novas tecnologias exige um longo tempo de aprendizagem, pela complexidade dos pré-requisitos conceituais envolvidos, sejam aqueles intrínsecos, sejam aqueles relativos ao ambiente de execução e comunicação (Orientação a Objetos, TCP/IP, etc)" [1].

Com o objetivo de diminuir a curva de aprendizado de programação, surgiu a linguagem Telis. Desenvolvida pela equipe do Edugraf, laboratório de software educacional – INE – CTC – UFSC, coordenado pelo professor Luis Fernando Bier Melgarejo, esta linguagem de programação se baseia em antecessores poderosos e simples como Forth, Logo e SmallTalk, e mantém características de distribuição e execução semelhantes a Ágora [1].

Telis foi desenvolvida de modo que todo seu ambiente propicia uma rica interação com a Internet, aproveitando o entusiasmo causado pela popularização da rede para estimular o interesse dos estudantes. A idéia de fazer com que Telis passe a interagir com dispositivos móveis surgiu do mesmo princípio.

Telis é utilizada hoje nas primeiras fases dos cursos de ciências da computação da engenharia de automação da UFSC, como ferramenta de aprendizagem de programação. A idéia básica é permitir que logo nos seus primeiros programas um estudante consiga desenvolver aplicações gráficas e publicá-las na Internet. Da mesma forma, a intenção é que, logo nos primeiros programas, sem conhecimento prévio de linguagens e técnicas de programação, um estudante possa desenvolver aplicativos para dispositivos móveis, e executá-los na própria sala de aula e, se possível, em seus próprios telefones celulares.

Segundo dados do IBGE, o número de residências com telefones celulares superou o número de residências com telefones fixos nos lares brasileiros já em 2005 [2], o que demonstra a rápida difusão desta tecnologia na sociedade brasileira.

A grande dificuldade da concretização deste trabalho reside nas diferenças existentes entre a capacidade computacional dos computadores pessoais e a dos dispositivos móveis.

Uma aplicação executada em um navegador instalado em um PC dispõe de recursos computacionais adequados para a execução de programas escritos em Telis: processadores rápidos, grandes quantidades de memória e de armazenamento, conexões de redes confiáveis e monitores de alta resolução e bilhões de cores. O universo dos dispositivos móveis é totalmente diferente: processadores lentos, monitores muito pequenos muitas vezes com apenas duas cores, pouquíssima memória e freqüentemente nenhuma capacidade de armazenamento.

Por exemplo, o requisito mínimo para execução de um programa em Java ME, utilizando MIDP 2.0 e CLDC 1.1, exige:

- um mostrador de 96 por 54 pontos e um bit de cor;
- 416 *kilobytes* de capacidade de armazenamento para as bibliotecas e mais o que for necessário para armazenar o programa;
- 40 *kilobytes* de memória de execução, chamada de memória volátil;
- processadores de 16 bits;
- algum dispositivo de entrada de dados;

Estas diferenças se refletem nas versões da tecnologia Java, no caso a *Enterprise Edition* (EE), a *Standard Edition* (SE) e a *Micro Edition* (ME). Esta última é versão disponibilizada para a plataforma de dispositivos móveis, naturalmente com recursos bem mais limitados do que as outras.

Para alcançar o objetivo do trabalho, a abordagem adotada foi a implementação de um compilador, integrado ao ambiente de Telis, que transforma um aplique publicado para Internet, em um aplique publicado para plataforma móvel, e um programa que executa o aplique em um dispositivo móvel compatível com a tecnologia Java ME.

Este conjunto de programas foi chamado de Telis ME, uma analogia à própria plataforma Java ME, sobre o qual é escrita a máquina virtual.

1.1. Delimitação

O ambiente de Telis é complexo demais para a plataforma de dispositivos móveis, pois exige um teclado, um mouse e um monitor que permita visualizá-lo. Além disso, o tempo necessário para a implementação de todo o ambiente é muito maior do que o tempo disponível para conclusão do trabalho.

Dadas estas limitações de recursos, foi decidido que Telis ME não inclui o ambiente de Telis, sendo então, uma plataforma que permite que programas escritos em Telis sejam executados em dispositivos móveis.

Telis ME disponibiliza, assim como o núcleo de JME para a tecnologia Java, um subconjunto dos recursos disponíveis em Telis. Alguns recursos que demandam muito processamento foram excluídos.

A ordem de implementação dos recursos foi baseada na usabilidade do produto final, ficando com menor prioridade, a manipulação de recursos gráficos, e com maior prioridade as primitivas e características inerentes da execução da linguagem, como manipulação de pilhas e listas, comunicação entre atores e estruturas de repetição.

Os recursos de Telis efetivamente implementados em Telis ME até a data da entrega do trabalho, estão disponíveis em uma tabela anexa, ou podem ser consultados diretamente no próprio código fonte do compilador, na classe `br.ufsc.edugraf.telis.me.compilador.Simbolos`.

2. Telis

Telis é totalmente implementada em Java. As aplicações (ou apliques) feitas em Telis são *applets* Java contendo os programas que são interpretados por uma máquina Telis que é executada em um navegador que possua uma JVM embutida.

O escopo principal das aplicações Telis é basicamente exibir e movimentar objetos gráficos e colher entradas de usuários, para que possa haver interação. Estes são requisitos típicos de jogos e simuladores, duas das áreas mais exploradas pelos

alunos quando estudando Telis. Estas são também duas das principais áreas de desenvolvimento em dispositivos móveis. A própria JME disponibiliza APIs especializadas em programação de jogos.

Outros tipos de aplicação, não apenas jogos e simuladores podem e são desenvolvidos em Telis e em JME, porém percebe-se uma tendência no usuários de ambas as plataformas em desenvolver aplicações deste tipo.

Para a exibição de gráficos e leitura de entrada de dados, Telis utiliza a biblioteca AWT (*Abstract Window Toolkit*), que não é compatível com a JME, o que constitui um dos principais obstáculos para a implementação de Telis ME. No lugar da AWT a JME oferece o pacote LCDUI, que é uma biblioteca gráfica muito inferior em recursos. Estas questões serão discutidas mais profundamente adiante.

2.1. A linguagem Telis

Telis oferece o aprendizado através do uso, simplificando conceitos e técnicas. Estas características fazem de Telis uma ferramenta apropriada ao aprendizado. Alguns exemplos de conceitos e técnicas simplificados em Telis são:

- Orientação a objetos, que em Telis toma uma forma intuitiva. Por exemplo, ao criar um modelo e enviar mensagens a um ator deste modelo, o usuário da linguagem está implicitamente criando uma classe e instanciando um objeto, mas sem necessariamente conhecer os conceitos de classe e objeto.
- Estruturas de dados, através do uso constante de pilhas e listas, recursos fundamentais no desenvolvimento de programas em Telis.
- Paralelismo: técnicas de sincronização de programas paralelos são abstraídas. Em Telis, cada ator é executado em seu próprio escopo, e possui sua própria pilha de dados e linha de execução. Desta forma um estudante que instancie mais de um ator em um mesmo aplique está executando um programa com várias linhas de execução.
- Comunicação de dados: as mensagens trocadas entre atores podem ultrapassar os limites do aplique, e até do computador em que o aplique está sendo executado.

- Aplicações distribuídas são desenvolvidas com freqüência pelos alunos, utilizando estímulos como base de comunicação.
- A manipulação de objetos gráficos é feita através de chamadas a agendas primitivas de qualquer ator. Ou seja, qualquer ator é um objeto com capacidade gráfica.

Além disso, Telis possui características peculiares, algumas delas herdadas diretamente de outras linguagens:

- Os operadores pós-fixos usados em Telis, que a princípio são criticados pela maioria dos programadores experientes, oferecem aos usuários leigos uma familiaridade ao uso de algumas calculadoras de bolso;
- As operações são realizadas diretamente na pilha;
- Tipagem dinâmica, ou seja, um tipo de dados só é validado se for necessário, e isto só ocorre em tempo de execução;
- As primitivas da linguagem são em português;
- O tratamento de eventos é feito através de uma estrutura de estímulos e tratadores;

Algumas destas características, entre outras, fazem de Telis uma linguagem propícia a ser usada no aprendizado de programação.

2.2. Conceitos e nomenclaturas de Telis

Detalhar a linguagem e o ambiente Telis, não é o foco principal deste trabalho. O foco é detalhar como funciona a implementação de Telis ME, mas alguns conceitos serão explicados de forma resumida, para facilitar o entendimento do trabalho. Para se aprofundar mais sobre Telis, consulte o sítio do Edugraf [3].

Seguem alguns conceitos e nomenclaturas utilizados em Telis, úteis para o entendimento deste trabalho:

- **Aplique**: é uma aplicação Telis. É constituído por um ou mais atores e seu ponto de início são agendas iniciais dos atores indicados para serem criados no seu carregamento;

- **Ator**: é o agente de Telis, o equivalente à uma instancia de um objeto. Todas as instruções estão associadas a um ator. Também são chamados de processos, uma vez que possuem escopos de execução isolados entre si [1];

- **Mundo**: é o painel de visualização do aplique;

- **Modelo**: é o equivalente à classe. Define o comportamento de um ator de sua criação até sua destruição;

- **Agenda**: é um conjunto de instruções agrupadas. É o equivalente à um método;

- **Agenda Inicial**: é o equivalente à um construtor de uma classe. É a agenda que é executada quando um ator é criado;

- **Agenda Primitiva**: é uma instrução mínima para o interpretador. Consiste de operações básicas de manipulação de pilhas, listas, números e textos, além de instruções de comunicação. São executadas no escopo do ator. Também são chamadas apenas de *primitivas*;

- **Pilha**: é a pilha de dados manipulada por um ator. Em uma execução de um aplique Telis com vários atores, cada ator terá a sua própria pilha. Todas as operações de manipulação de dados ocorrem diretamente na pilha;

- **Lista**: é um conjunto de dados, primitivas ou agendas. Qualquer lista pode ser executada, ou disparada como um estímulo;

- **Estímulo**: é uma mensagem enviada pelo ambiente de execução ou por um ator para um determinado contexto, denominado em Telis de *alcance*. É a forma de implementação de tratamento de eventos em Telis;

- **Tratador**: é o conjunto formado por um filtro de estímulos e uma lista de tarefas a serem executadas pelo ator, caso o filtro aceite o estímulo. É o ponto de filtragem e tratamento de estímulos.

Para saber mais sobre os recursos de Telis implementados em Telis ME, consulte o tópico *Delimitação*.

3. Java para dispositivos móveis (JME)

A tecnologia Java é composta por três partes: a linguagem Java, a máquina virtual Java (JVM) e a API (*Application Programming Interface*, conjunto de bibliotecas de classe) Java. O conjunto formado por estes componentes é o que caracteriza a tecnologia Java, e a variação dos recursos disponíveis em cada componente, é o que caracteriza a versão disponibilizada para uma determinada plataforma.

A comunidade Java através do JCP (*Java Community Process*), especifica três plataformas para tecnologia Java: Java SE (*Standard Edition*) voltada para aplicações em computadores pessoais, Java EE (*Enterprise Edition*) voltada para servidores e Java ME (*Micro Edition*) voltada para dispositivos móveis. Esta última é a que concerne o desenvolvimento de softwares para dispositivos móveis, como celulares e PDAs (*Personal Data Assistant*), e é a plataforma de destino da migração de Telis.

A J2ME, como também é conhecida à versão atual da plataforma Java ME (JME), é o conjunto de especificações das tecnologias envolvidas no processo de desenvolvimento, distribuição e execução de softwares para dispositivos móveis em Java: a máquina virtual, a linguagem, as bibliotecas envolvidas no núcleo da linguagem, as bibliotecas adicionais, denominadas *profiles*, e os conjuntos de bibliotecas opcionais denominadas *optional packages*.

A especificação do núcleo das tecnologias envolvidas é chamada de configuração. Uma configuração é composta pela especificação de uma máquina virtual, das restrições à linguagem, e das chamadas bibliotecas de núcleo (*core libraries).*

Duas configurações são definidas pela JME: a CDC (*Connected Device Configuration*) que abrange os PDAs com maior capacidade computacional e a CLDC (*Connected Limited Device Configuration*) que abrange os dispositivos que possuem recursos mais escassos, entre eles os telefones celulares, foco deste trabalho.

3.1. Máquina Virtual Java (JVM)

A JVM (*Java Virtual Machine*) foi inicialmente projetada para ser executada em dispositivos móveis (PDAs) sob a sigla de KVM (*K Virtual Machine*). Isto faz com que o esforço atual de migração de Java para dispositivos móveis seja considerado uma "volta às raízes" [4].

As máquinas virtuais que as empresas disponibilizam em seus aparelhos, seguindo as especificações da CLDC, são chamadas KVM, como as primeiras máquinas desenvolvidas para Java, que nos primórdios era chamada de Oak.

Para cada configuração, é escolhido um subconjunto de instruções ou todo o conjunto, dependendo da disponibilidade de recursos de cada dispositivo. Desta forma o código interpretado pela JVM não é totalmente intercambiável entre plataformas.

Entre os recursos da JMV ausentes na KVM da CLDC estão:

- Carregadores de classe definidos pelo usuário
- Reflexão
- *Thread groups* e *daemon threads*
- Finalização de instâncias.
- Erros e exceções assíncronas.

Destas características que estão ausentes, a mais custosa para o desenvolvimento do trabalho é quanto à reflexão, uma vez que a máquina virtual de Telis utiliza este recurso para efetuar a chamada das primitivas e agendas.

Esta limitação foi determinante no processo de decisão da abordagem a ser tomada para a migração de Telis. Além de impedir aproveitamento da máquina virtual de Telis, criou a necessidade de um método otimizado para a execução, uma vez que a capacidade processamento nas plataformas móveis é muito limitado.

3.2. Bibliotecas de núcleo

As bibliotecas de núcleo, chamadas de *core libraries*, denotam as bibliotecas envolvidas com a linguagem (`java.lang.*, java.util.*`), as bibliotecas de

entrada e saída (`java.io.*`) e dos recursos que envolvem segurança, redes e internacionalização.

Estas bibliotecas são, na JME, um subconjunto das bibliotecas disponibilizadas na JSE, sendo definido que se uma classe está presente ela deve ser idêntica ou um subconjunto da classe equivalente distribuída na JSE.

Nenhum novo recurso pode ser adicionado e a semântica dos métodos e atributos deve permanecer a mesma, se existirem [7].

A CLDC é então a definição de um núcleo que é um subconjunto da CDC que por sua vez é um subconjunto do núcleo de Java, como está ilustrado na Figura 1. A figura mostra ainda os componentes que formam uma configuração. Cada setor do círculo representa uma das partes de uma configuração enquanto cada círculo concêntrico representa uma configuração.

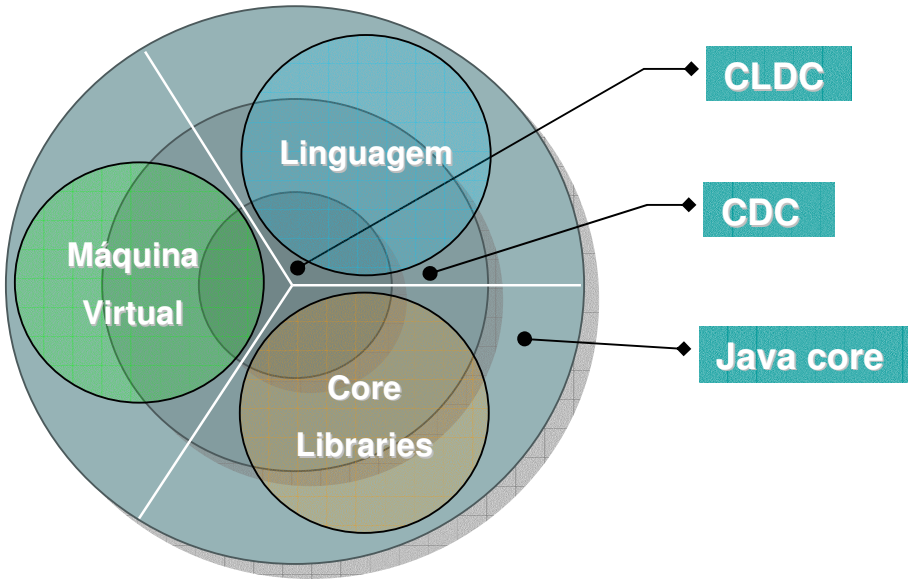

Figura 1: formação do núcleo de Java nas diversas distribuições

3.3. MIDP (*Mobile Information Device Profile*)

Além dos componentes que formam uma configuração, JME disponibiliza bibliotecas adicionais, que são incompatíveis com as outras versões de Java. Estas bibliotecas são construídas sobre as configurações CDC e CLDC, e são chamadas de

profiles. Além dos *profiles* outras especificações de bibliotecas são disponibilizadas, e chamadas de *optional packages.* Estas são bibliotecas opcionais pois definem APIs para recursos que podem ou não estar disponíveis em um determinado dispositivo.

MIDP (*Mobile Information Device Profile*) é o principal conjunto de bibliotecas construído sobre a plataforma ME, e é praticamente um padrão. Outros *profiles* menos conhecidos foram desenvolvidos: *KittyHawk*, *PDAP* (extensão do MIDP, ainda não terminado), *i-appli* (desenvolvido antes do MIDP).

É um erro bastante comum confundir JME com MIDP. Os desenvolvedores confundem os termos, pois hoje praticamente todo desenvolvimento para celulares em Java é feito sob o MIDP, uma vez que ele se tornou quase um padrão na industria. Hoje a maior parte dos aparelhos de todos os fabricantes adotaram MIDP em suas implementações de Java.

MIDP está disponível para as configurações CDC e CLDC. A plataforma alvo de Telis ME são os telefones celulares, que são abrangidos pela CLDC.

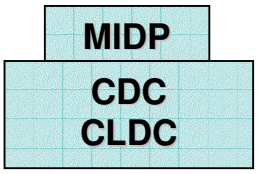

Figura 2: MIDP disponível sobre CDC e a CLDC

MIDP oferece bibliotecas para os seguintes escopos:

- Apresentação em monitores de tamanho limitado
- Meios para leitura de dados e ações de usuários
- Armazenamento persistente
- Envio e recebimento de mensagens
- Comunicação de dados
- Telefonia sem fio

3.3.1. Bibliotecas de manipulação de gráficos

O ambiente gráfico e os recursos de interação com o usuário, foram implementados em Telis ME usando o pacote `javax.microedition.lcdui.game`. Este pacote disponibiliza funcionalidades que satisfazem vários dos requisitos gráficos de um aplique Telis.

Um recurso útil fornecido por esta biblioteca é o conceito de manipulação de gráficos em camadas. Em Telis ME foram utilizados dois tipos de camadas fornecidos pelo pacote, uma para representar o fundo (`TiledLayer`) e outro para representar os atores se movendo na tela (`Sprite`).

O gerenciamento das camadas é feito pela classe `LayerManager`. Assim as requisições de atualização de tela são feitas para esta classe que desenha as camadas na ordem programada.

Os recursos são utilizados através de herança e composição. A classe `Ator` especializa a classe `Sprite` enquanto a classe `Mundo` utiliza a classe `TiledLayer` para desenhar o fundo do aplique.

4. Telis ME

Telis ME é o conjunto de programas que viabiliza a execução de código Telis em ambientes móveis, que suportem a tecnologia Java ME, ou seja, Telis ME é o produto final deste trabalho.

A Figura 3, mostra a analogia entre Telis e sua versão para dispositivos móveis Telis ME, e sua relação com as configurações de Java, sobre os quais são construídos.

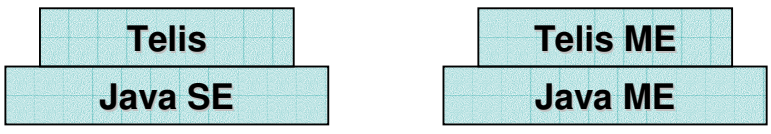

Figura 3: Telis é construído sobre JSE. Telis ME sobre JME

Os programas que constituem essa ferramenta são: um compilador que, à partir de um aplique Telis publicado na forma de XML, gera uma saída otimizada, e uma máquina virtual capaz de traduzir esta saída em ações. Cada símbolo oriundo do código fonte Telis é traduzido para uma ou várias ações da máquina Telis ME.

Como em Telis, um programa em Telis ME é chamado de aplique, e cada aplique é constituído de um ou mais atores e suas agendas. Cada ator possui uma área de dados (pilha) própria e um fluxo de execução independente dos outros atores que compõe o aplique.

A compilação é executada em cinco etapas, cada uma gerando produtos parciais que são usados como entrada da próxima etapa. Estas etapas serão discutidas posteriormente.

O produto final da transformação é um par de arquivos de extensão `.jad` e `.jar`. O arquivo `.jad` descreve o que contém o arquivo `.jar`, e este é o pacote de classes Java que denotam o programa. A plataforma de destino, seja um simulador ou um telefone celular que suporte Java, lê o arquivo `.jad` e oferece ao usuário a execução dos programas, chamados *Midlets*. Um aplique é um único *Midlet*, contendo todos os dados necessários para sua execução, incluindo a máquina virtual e a programação dos atores e suas agendas.

Neste ponto Telis ME se assemelha ao Telis. Quando um usuário carrega um aplique em um navegador, a máquina Telis, disponibilizada na forma de um arquivo `.jar` é carregada juntamente com a descrição do aplique, que, no caso de Telis, está disponibilizado em formato XML.

A execução é basicamente a tradução do código intermediário em ações que refletem alterações nos estados dos atores. Esta etapa foi otimizada, e o que acontece é que a interpretação é feita parcialmente na compilação e parcialmente na execução.

Diferentemente do Telis, um aplique Telis ME não é interpretado diretamente de um arquivo XML ou de qualquer outro arquivo armazenado. Ele passa por um processo de tradução e otimização que gera uma saída mista. Diz-se que Telis ME utiliza uma mescla de compilação e interpretação.

4.1. O compilador

O compilador Telis ME é embutido dentro do código do Telis. Ele utiliza o *parser* XML do ambiente Telis e obtém informações de diretório das configurações do próprio ambiente.

Ele é instanciado de forma independente, mas utiliza várias classes do pacote que contém a maquina virtual do Telis (`br.ufsc.edugraf.telisme.maquina`), além do conversor de XML para aplique.

A entrada do compilador é o aplique Telis já publicado para *Web*, em formato XML, e o produto final gerado pela transformação é a dupla de arquivos citados na introdução, o `[aplique].jad` e o `[aplique].jar`, como mostra a Figura 4:

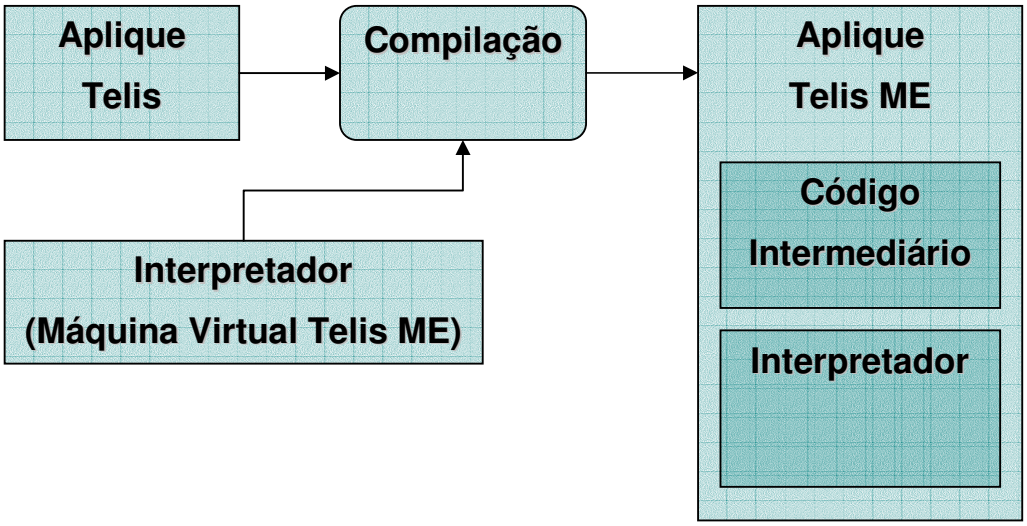

Figura 4: em Telis ME o código do aplique é distribuído junto ao interpretador

O arquivo `.jar` é a biblioteca de classes Java, e o `.jad` é um arquivo utilizado pelos ambientes Java ME para descrever os pacotes com os *Midlets*. Isso serve para auxílio na decisão sobre o carregamento de um *Midlet* em um dispositivo móvel.

4.1.1. Implementação

Na otimização da execução do aplique, o ponto crucial é o formato em que ele é armazenado. Se o Telis ME se comportasse como o Telis, a máquina virtual Telis ME seria responsável pela interpretação de um arquivo XML, e teria seu tamanho aumentado consideravelmente dada a inclusão de um *parser*. Além disso, a interpretação do arquivo XML consumiria tempo de processamento e memória volátil, recursos que em dispositivos móveis são escassos.

Em termos cronológicos, o formato do código intermediário, foi decidido a partir de uma análise de algumas opções, cada uma com suas vantagens e desvantagens. Abaixo a descrição curta de cada uma das opções:

- XML: seria a escolha de menor custo, uma vez que o aplique já é publicado para *Web* neste formato, mas como discutido anteriormente, não seria uma opção que levaria em conta os requisitos de performance das plataformas.
- Código em formato próprio com dados em formato texto: esta opção era atrativa no sentido de que permitiria uma revisão do código gerado, mesmo em tempo de execução, já no próprio dispositivo, mas ainda parecia bastante prejudicada em relação à performance, uma vez que as instruções, neste caso, seriam palavras que deveriam ser buscadas em uma tabela de espalhamento. Ainda necessita de um módulo de interpretação, porém bem mais leve que um *parser* XML.
- Código em formato próprio com dados binários: com esta opção o problema da performance seria amenizado, mas uma característica importante presente em outras opções, a de ser humanamente legível, seria perdida. Além disso, um código gerado de maneira errada poderia acarretar em erros graves e de difícil detecção dentro da máquina Telis ME. Esta opção elimina a necessidade de um interpretador, mas ainda requer o carregamento de um arquivo externo.
- Utilização do próprio código Telis: o código fonte Telis é internamente chamado de gíria. O conversor o transforma o código em objetos que representam palavras que podem ser reescritas em forma de texto. O código se assemelha muito a gíria em si, mas é muito mais comportado, e requer menos

processamento na interpretação. Esta alternativa apresenta as mesmas vantagens e desvantagens do código próprio textual, mas com uma grande vantagem: sua visualização é facilmente reconhecida como código Telis, o que permitiria ao usuário do aplique no dispositivo ter ainda algum contato com o mundo Telis. Como as alternativas anteriores, requer um módulo de interpretação.

- Geração de código fonte Java: nesta abordagem o compilador geraria arquivos contendo código fonte Java, que descreveriam o aplique, seus atores e suas agendas, e este código seria compilado já para plataforma de destino. Desta forma a verificação de sintaxe seria feita quando o código fosse compilado. Apresenta o melhor resultado em performance uma vez que as ações são criadas diretamente no construtor das classes, sem necessidade de ler qualquer arquivo externo. Além disso, dispensa a implementação de um módulo de interpretação, o que reduz o tamanho do *Midlet* e aumenta a performance. A depuração do código pode ser feita através de *logs*.

Duas foram as abordagens implementadas como teste: a que cria um código em formato texto e a que cria o código Java. Desta experiência constatou-se além das vantagens anteriores, a velocidade de implementação, que é muito mais rápida na geração de código Java, além da eliminação da necessidade de verificar o código do aplique. Assim, a implementação da tradução gerando código intermediário em formato de texto foi descontinuada.

Desta forma, o que acontece, é que a tradução gera código Java, que contém o código dos modelos de ator e a definição de seu comportamento, e este código é compilado, gerando código Java nativo, compatível com a CLDC.

O código gerado é dividido em classes que representam os modelos de ator derivadas da classe `telisme.maquina.Ator` e uma classe derivada da classe `telisme.maquina.Executor` que instancia os atores e cria seus fluxos de execução. As classes `Ator` e `Executor` estão contidas na máquina Telis ME, e estendem classes do MIDP, como mostra a Figura 5. Enquanto a classe gerada, que estende a classe ator, é declarada com o nome do modelo, a classe Executor é declarada com o nome fixo de `ExecutorDeAplique`. Os apliques são diferenciados

pelo pacote. Desta forma, a classe gerada por um modelo chamado `umModelo`, parte de um aplique chamado `umAplique` terá o nome: `telisme.apliques.umaplique.umModelo`.

O código da classe `Ator`, contém as reações a cada ação programada para um ator. As instruções alteram o estado interno das pilhas de dados e de execução do ator, de forma muito semelhante a atuação da própria máquina virtual Java.

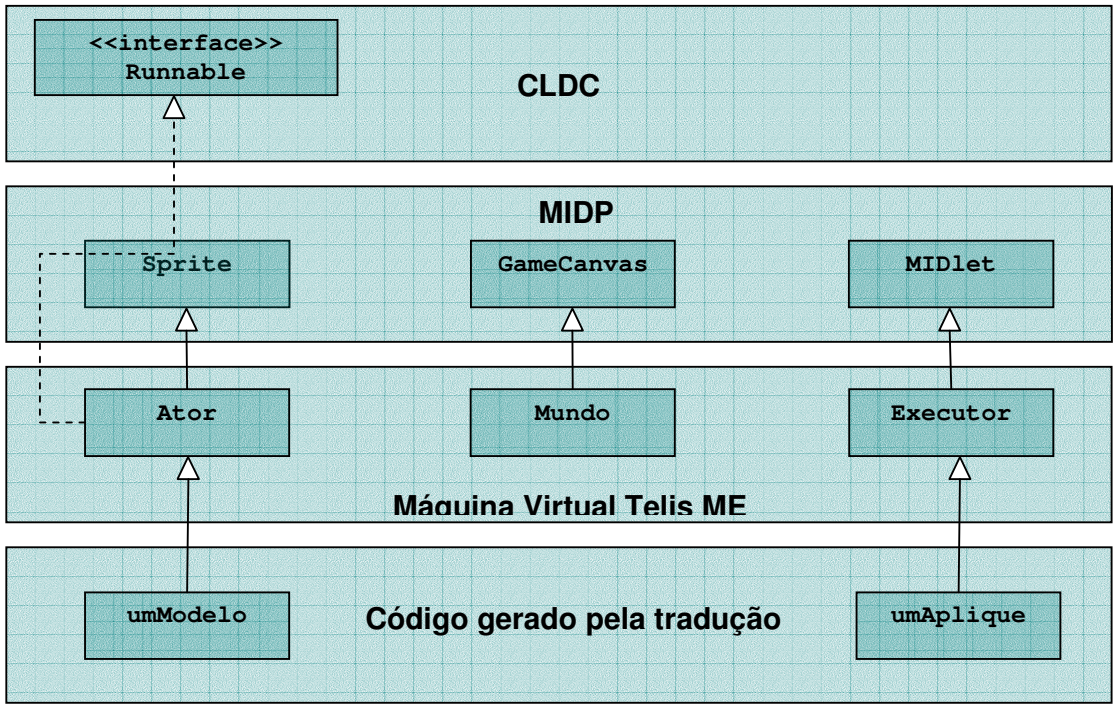

Figura 5: geração de código utilizando herança

As classes de atores programam suas agendas através da chamada do método `Ator.adicione`, que cria e adiciona ações nas listas de execução que representam as agendas. Cada operação sobre um ator é chamada de ação, e está diretamente ligada a pelo menos um símbolo contido no código das agendas dos atores compilados.

Não apenas as primitivas se tornam ações na máquina Telis ME. Todos os símbolos geram pelo menos uma ação. Por exemplo, o símbolo "`[`", que indica o início de uma lista, representa uma ação denominada `Ação._abrirLista` na máquina Telis ME. Desta forma a execução do aplique no PC ou no dispositivo móvel ficou muito semelhante.

As agendas primitivas denotadas por símbolos, como "`[`", "`/`" e "`*`", por exemplo, tem suas constantes marcadas com um "_" no início, nas suas definições na classe `Acao`. No caso dos exemplos citados, as constantes geradas no código gerado seriam: `Acao._abrirLista`, `Acao._dividir` e `Acao._multiplicar`, respectivamente.

4.1.2. Etapas do processo de transformação

O processo de transformação do aplique publicado para *Web* para o aplique publicado para dispositivos móveis é composto por várias etapas. Estas etapas foram separadas de acordo com a geração de produtos parciais:

1. **Tradução:** à partir dos objetos gerados pela classe `br.ufsc.edugraf.util.xml.ConversorXMLTelisParaAplique`, o compilador Telis ME traduz o código Telis no código intermediário.

2. **Compilação:** neste ponto ocorre o processo de otimização que transforma o código intermediário gerado em código CLDC executável por uma máquina virtual Java ME.

3. **Verificação:** a máquina virtual Java ME exige que o *bytecode* Java seja verificado por uma ferramenta chamada *Preverify*, disponível no JWT, antes que possa ser distribuída.

4. **Empacotamento:** criação do arquivo `.jar`, que contém além do código executável, as bibliotecas que constituem a máquina Telis ME.

5. **Publicação:** geração do arquivo `.jad`, que descreve o código binário gerado no passo anterior, em forma de texto, para prover informações sobre o aplique, incluindo por exemplo, o tamanho do pacote de classes a ser carregado e envio para o servidor, caso seja necessário.

Estas etapas envolvem diversos componentes além do compilador em si. São envolvidos no processo, classes do Telis, *scripts* de sistema operacional, e programas do próprio Java, como o compilador `javac.exe` e o verificador de código para JME `preverify.exe`, como mostra a Figura 6.

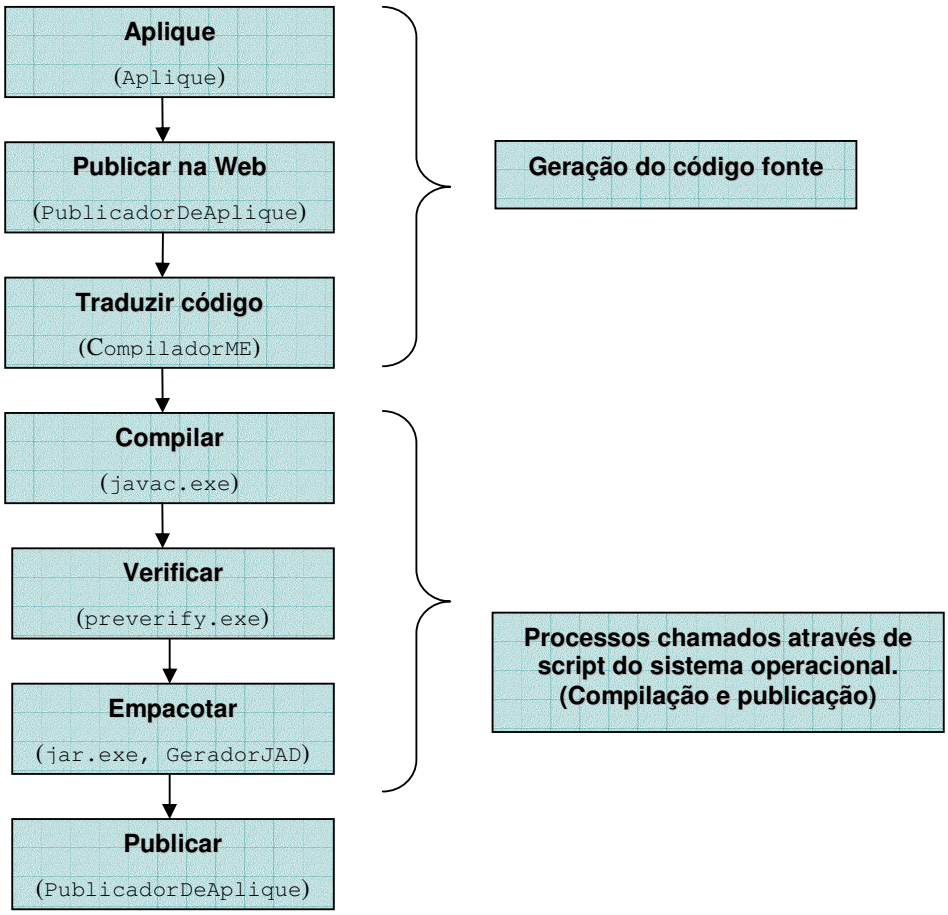

Figura 6: fluxo das etapas de transformação

Internamente, na máquina Telis, um aplique é representado por um objeto da classe `br.ufsc.edugraf.telis.maquina.Aplique`. O compilador desenvolvido para Telis ME, representado por um objeto da classe `br.ufsc.edugraf.telis.me.CompiladorME` têm como entrada os dados da classe `br.ufsc.edugraf.util.xml.ConversorXMLTelisParaAplique`, que carrega um aplique publicado pela classe `br.ufsc.edugraf.ambiente.impl.PublicadorDeApliques`. Em outras palavras, o compilador Telis ME precisa que um aplique seja primeiro publicado para Web, para depois carregá-lo e transformá-lo.

4.1.3. Tradução de código

O processo de tradução do código, é a geração de código Java, à partir de código Telis. É nesta fase que ocorre a maior interação entre o Telis e Telis ME, pois parte da interpretação do código é feita no código do ambiente Telis. Isto ocorre através de chamadas ao método `obterProximoAtor` da classe `ConversorXMLTelisParaAplique` que retorna uma instancia da classe `br.ufsc.edugraf.telisme.maquina.palavra.ModeloDeAtor`, que contém métodos e atributos que descreve os atores e suas agendas.

A classe `CompiladorME`, interage com as classes `CompiladorDeModelo` e `CompiladorDoExecutor`, que são responsáveis pela geração do código dos modelos e do executor, respectivamente. Estas classes estão contidas no pacote `br.ufsc.edugraf.telis.me.compilador`.

Para cada modelo do aplique o `CompiladorDeModelo` é chamado e gera o código Java em memória. Após todos os modelos estarem compilados em uma lista de *buffers*, eles são gravados em arquivos. O mesmo processo ocorre depois com o executor.

A compilação do modelo basicamente é uma troca de símbolos. Para cada símbolo encontrado no código do aplique, um ou mais símbolos são gerados na saída. Os símbolos gerados são *strings* de constantes definidas na classe `telisme.maquina.Acao`, da máquina virtual Telis ME, sendo adicionados a uma lista, através do método `Ator.adicione`. Para cada agenda é definida uma lista separada.

Para a maioria das primitivas, a tradução é feita de forma direta, como mostra a Figura 7.

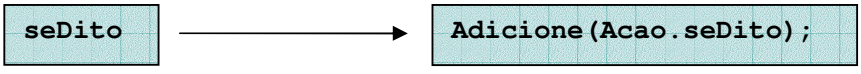

Figura 7: Transformação do seDito em uma ação em Telis ME

As exceções a esta regra são as constantes, as associações de variável e as chamadas de agenda. A principal diferença nestes casos é a adição de um parâmetro a ação.

- **Constantes:** as constantes são avaliadas pelo compilador e marcadas como texto ou número. Esta diferença é necessária na hora de avaliar um filtro de estímulos de um tratador. Se a constante for um texto, a ação `Acao._pushTexto` é criada, e se for um número, a ação criada é `Ação._pushNumero`. Neste caso, o parâmetro adicionado é o valor da constante.
- **Variáveis:** O tratamento de variáveis é uma tarefa diferenciada tanto em Telis como em Telis ME. Quando encontra uma associação, ao invés de gerar um objeto de *token*, o interpretador da máquina virtual Telis gera um objeto de *instrução*. Isto por que se uma variável está indefinida, a única primitiva que pode estar depois dela é a primitiva `associar`, caso contrário, um erro de variável não definida é gerado. O compilador do Telis ME gera uma ação (`Acao.associar`) e passa como parâmetro o nome da variável quê será criada e receberá valor contido no topo da pilha. As variáveis são adicionadas na tabela de símbolos do modelo quando são associadas, e quando são referenciadas, uma ação `Acao._pushVariavel`, com o nome da variável é criada.

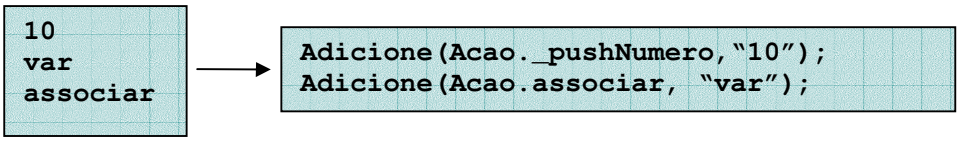

Figura 8: exemplo de transformação de associação de variável

- **Agendas:** as agendas são incluídas na tabela de símbolos do modelo no momento da criação do `CompiladorDeModelo`, uma vez que já são todas conhecidas. O parâmetro adicional que acompanha a ação criada para chamada de agenda (`Acao._executarAgenda`) é o nome da agenda a ser executada.

4.1.4. Compilação e publicação

A saída do tradutor são arquivos `.java` contendo o código fonte de cada modelo, e o código fonte do executor, que é a classe derivada da classe `Executor` da máquina Telis ME. A classe `Executor` estende a classe `java.microedition.midlet.MIDlet`, e é o ponto de partida do aplique.

As classes são geradas em um diretório pré-definido e conhecido pelo processo, que contém algumas ferramentas necessárias para a conclusão da transformação. Estas ferramentas são:

- Um *script* de sistema que limpa os diretórios temporários para geração dos novos arquivos;
- Um *script* de sistema que chama os programas que fazem parte do Java.

Este diretório pré-definido é conhecido através do método `obterRaizParaSalvamento()`, presente no pacote `br.ufsc.edugraf.telis.ambiente.configuracoes.Configuracao` do ambiente Telis. Abaixo dele é criado um diretório `me`, onde todo o processo é realizado. Em `me/tools`, encontram-se as ferramentas acima listadas.

Os apliques são gerados em um diretório com seu nome, criado no diretório `me/telisme/apliques`, e o compilador é chamado à partir do diretório `me`, então, as classes são declaradas como parte do pacote `telisme.apliques`, seguido do nome do aplique em letras minúsculas e sem acentos. Antes do próximo passo, o arquivo `MANIFEST.MF`, que deve fazer parte do arquivo `.jar`, e contém a descrição do mesmo, é gerado pela classe `br.ufsc.edugraf.telis.me.GeradorManifest`.

Um *script* de sistema operacional chama o `javac.exe`, para compilar o código fonte do aplique. Os arquivos `.class` são gerados em `me/classes`, onde devem estar os binários da máquina virtual Telis ME. Em seguida o *script* chama o `preverify.exe`, que é uma ferramenta de verificação do código necessária para adequar o código gerado pelo compilador Java para a KVM. O resultado são arquivos

`.class` gerados em `me/verified`, que já contém o aplique e a máquina virtual Telis ME, com *bytecode* já compatível com a CLDC.

O *script* de compilação necessita conhecer o diretório onde estão as classes compiladas da máquina virtual, pois os apliques necessitam delas para compilar, além das classes serem empacotadas junto aos apliques no final da transformação.

Após verificados, o *script* chama o utilitário `jar.exe` que gera o pacote no diretório de destino, selecionado pelo usuário. Finalmente a classe `br.ufsc.edugraf.telis.me.compilador.GeradorJAD` gera o arquivo `.jar`. Ao final do processo , um arquivo de *log* `BuildLog.txt` é gerado no diretório de saída.

Se o aplique estiver sendo publicado na Web, então o processo de compilação acontece em um diretório temporário, e só depois de obtidos os arquivos `.jar` e `.jad` é que eles são enviados para o servidor.

4.1.5. Exemplo

Exemplo com código de uma transformação de um aplique publicado como XML em código Java. A Figura 9 mostra de forma resumida os produtos da transformação:

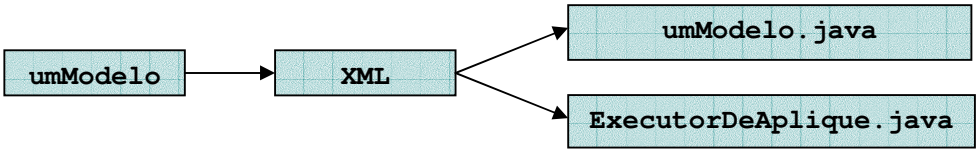

Figura 9: resumo do exemplo de transformação

```
<repositório>
<apliques>
<aplique    nome="umAplique"    atores="umModelo"    altura="500"
largura="500"  esquerda="0"  topo="0" />
</apliques>
<modelos>
<modelo nome="umModelo">
<agenda nome="umaAgendaLocal">
<![CDATA[
  nome
  mostrar
]]></agenda>
<agenda>
<![CDATA[
  "vinicius" nome associar
  umaAgendaLocal
]]>
</agenda>
</modelo>
</modelos>
</repositório>
```

Figura 10: aplique publicado como XML

```
package telis.apliques.umaplique;
import telisme.maquina.*;

public class umModelo extends Ator {
      public umModelo(Mundo m) {
            super(m);
            Agenda a = programa.obterAgendaInicial();
            a.adicione(Acao._pushTexto, "vinicius");
            a.adicione(Acao.associar, "nome");
            a.adicione(Acao._executarAgenda, "umaAgendaLocal");

            a = programa.crieAgenda("umaAgendaLocal");

            a.adicione(Acao._pushVariavel, "nome");
            a.adicione(Acao.mostrar);
      }
}
```

Figura 11: umModelo.java

```
package telisme.apliques.umaplique;
import telisme.maquina.*;

public class ExecutorDeAplique extends Executor {

      public ExecutorDeAplique() {
            super();

            atores.addElement(new umModelo(mundo));
      }
}
```

Figura 12: ExecutorDeAplique.java

Neste exemplo, um aplique simples, chamado de `umAplique` instancia um ator à partir de um modelo chamado `umModelo`. Este ator tem uma agenda inicial que atribui uma variável e executa uma agenda local. O XML do aplique pode ser visto na Figura 10.

Este aplique gera um pacote com a máquina virtual Telis ME e as classes `umModelo` e `ExecutorDeAplique`, contidos no pacote `telisme.apliques.umaplique`, e cujos códigos fontes encontram-se na Figura 11 e na Figura 12 respectivamente.

4.2. Máquina Virtual Telis ME

A máquina virtual Telis ME é basicamente um conjunto de classes que denotam os conceitos básicos de Telis, como atores, listas, mundo, etc... além de recursos que são conhecido apenas internamente nesta versão, como ações, tipos, contadores, e algumas interfaces largamente utilizadas durante a implementação.

Basicamente um objeto executor ativa atores contidos em uma lista. Os atores, tem suas agendas programadas na construção dos objetos, e executam cada um em uma nova *thread*. As ações programadas nas agendas alteram os estados internos dos atores, em especial a pilha de dados e a pilha de contextos.

A execução da agenda inicial pode ser interrompida quando o ator receber um estímulo e estiver programado para tratar aquele estímulo. Neste caso, um outro

contexto é empilhado, e o contexto original só é restaurado ao fim da execução do estímulo.

4.2.1. Inicialização

O produto final da transformação é o par de arquivos `[Aplique].jar` e `[Aplique].jad`, que são, o pacote de classes e o seu descritor. Dentro do pacote `[Aplique].jar` está o pacote `telisme.apliques.[aplique]`, que contém a classe `ExecutorDeAplique`, que é derivada da classe `telisme.maquina.Executor`. Esta, por sua vez estende a classe `java.microedition.midlet.Midlet`. Um programa JME inicia no método `startApp` de uma classe que estenda a classe `Midlet`, sendo esse então o ponto inicial da execução do aplique.

A classe `ExecutorDeAplique` apenas adiciona os atores instanciados à lista `atores`, atributo da classe `Executor`. Todo o código de ativação dos atores está na classe base.

Como em Telis, os atores executam suas tarefas em um mundo, que neste caso um objeto `GameCanvas` associado a um *form* da biblioteca MIDP. Este *form* é criado no momento da chamada do `startApp` e passado para cada ator instanciado.

A classe `telisme.maquina.Ator` implementa a interface `java.lang.Runnable`, e todo o paralelismo da linguagem é baseado neste recurso. Quando o executor ativa um ator, ele associa o ator a uma instância da classe `java.lang.Thread` e dispara uma nova *thread*, o que quer dizer que o método `Ator.run` é implicitamente chamado em outra *thread* de execução.

Inicialmente antes de entrar no seu ciclo de vida, o ator empilha em seu contexto a sua agenda inicial, programando-a para ser executada uma única vez.

4.2.2. Ciclo de vida do ator

Quando ativo, um ator está em um laço de repetição executado na implementação do método abstrato `Runnable.run`. A cada repetição o método `Ator.viva` é executado, e quando seu valor de retorno é falso o ator morre, ou em

outras palavras a *thread* é finalizada naturalmente pela saída do método `Runnable.run`.

Como visto, ao iniciar seu ciclo de vida, o ator empilha sua agenda inicial para ser executada. Em Telis ME uma agenda é tratada como uma lista, que é executada até o fim repetidamente até que a avaliação de uma condição, que é empilhada junto à ela, retorne falso. A condição é representada pela interface `telisme.maquina.Avaliavel`. No caso de uma agenda, a condição empilhada com a lista é um objeto da classe `telisme.maquina.Contador`, com o número de repetições igual a um.

A cada chamada do método `Ator.viva`, uma ação é executada ou uma lista, que pode ser executada posteriormente, é criada.

Na entrada do método, a lista em execução é lida do topo da pilha de listas para execução. Se ainda houverem elementos a serem processados nesta lista, o ator processa este elemento, ou cria novas listas para serem empilhadas.

A varredura de uma lista é possível através da classe `telisme.maquina.Lista` que tem a capacidade de fornecer seus elementos de forma seqüencial, através dos métodos `Lista.proxItem` e `Lista.temMais`.

Se não houverem mais itens nesta lista, uma avaliação é feita na condição de execução desta lista, e se a condição for verdadeira, a lista é reiniciada e seu primeiro elemento é executado. Se a condição for falsa, uma nova lista é elegida para ser executada, sendo esta retirada do topo da pilha de listas para execução, que é alimentada também pelos estímulos recebidos pelo ator.

Se não houverem mais listas a serem executadas, o ator entra no estado de hibernação, de onde só sai quando morre ou quando recebe um estímulo.

A Figura 13 mostra o algoritmo de busca de instruções:

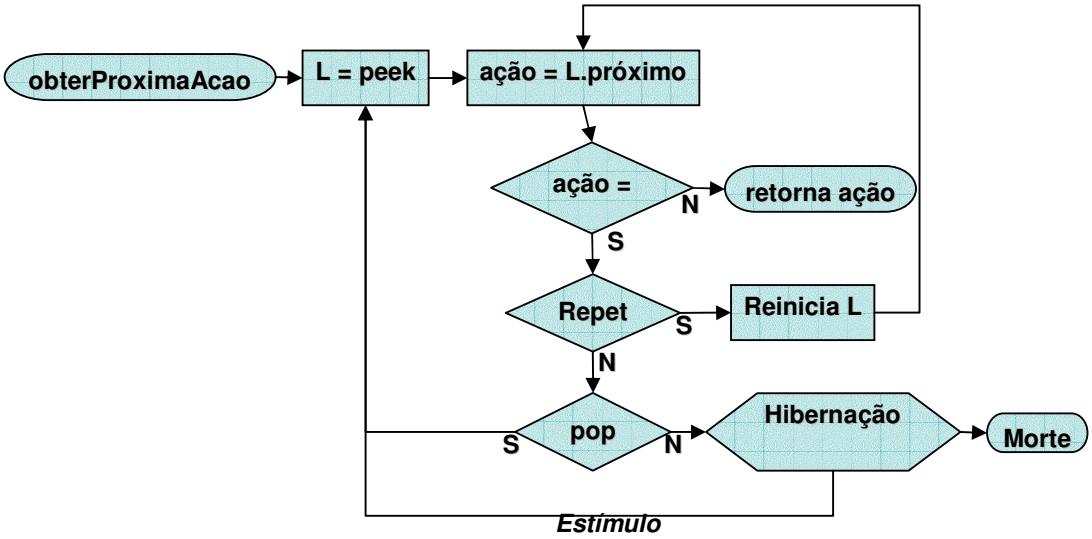

Figura 13: algoritmo de busca de instruções

4.2.3. Execução de uma ação

Como foi visto anteriormente, cada ação a ser executada no método `Ator.viva` é lida de uma lista, que pode ser uma lista em execução, uma agenda ou um estímulo. As ações que indicam o início e o fim da criação de uma lista são denominados `Acao._abrirLista` e `Acao._fecharLista`. Quando encontradas estas ações a execução vai para um estado especial, em que os elementos lidos da lista de execução, não são executados, e sim armazenados em uma nova lista, para futuro uso, ou processamento.

Em execução normal, um elemento lido da lista de execução é uma ação, ou é uma lista que está contida na lista que está em execução. Neste caso, a lista é empilhada, e o ciclo continua, procurando pelo próximo elemento da lista de execução.

No caso normal, o elemento lido de uma lista em execução é uma ação, que é processada de acordo com o seu código.

As operações são realizadas na pilha de dados. A pilha de dados utiliza tipos abstratos para manipular os dados, através da interface `Empilhavel`, como podemos observar na Figura 14.

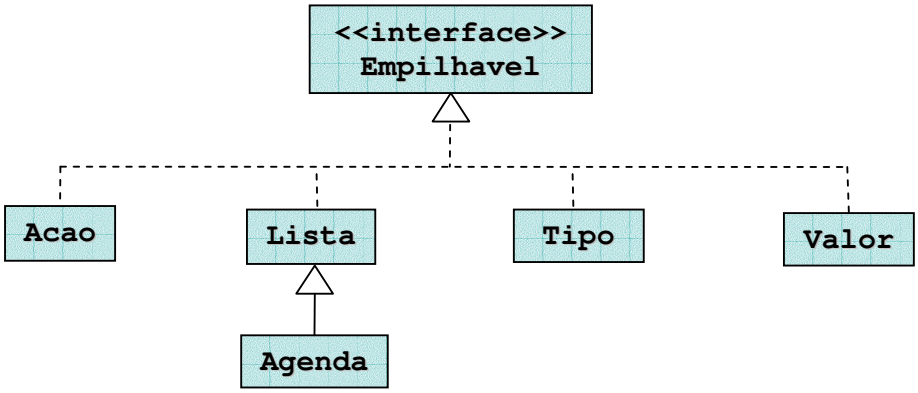

Figura 14: classes que implementam a interface Empilhavel

4.2.4. Contextos

Até agora foi visto como as ações são eleitas para serem executadas, e para isso falamos das listas de execução, e da existência de uma pilha de listas para execução. Esta pilha, controla o empilhamento não apenas das listas de execução, mas também dos contextos de execução. Estes contextos são representados pela classe `telisme.maquina.Contexto`, e são gerenciadas pela classe `PilhaDeContextos` do mesmo pacote.

Agora será exclarecido que além de cada ator possuir uma pilha de execução isolada, uma nova pilha é criada para ele cada vez que ele vai tratar um estímulo. Esta pilha é valida enquanto o código do estímulo estiver sendo executado, e é destruída depois disso, pois em Telis os estímulos devem ser executados sem alterar a pilha atual.

Figura 15: possíveis alterações na pilha de contextos

Os atributos de um contexto são: a pilha atual, a lista em execução, os dados necessários para a criação das listas contidas em uma agenda ou estímulo, a condição de repetição da lista de execução e uma marca que informa se aquele contexto foi gerado por um estímulo.

A pilha de contextos é alterada por chamadas de agendas, chamadas de primitivas de execução de listas, pela inicialização do ator e pelos estímulos, como mostra a Figura 15. No caso dos estímulo há um tratamento especial, para que durante o tratamento de um estímulo, nenhum outro estímulo seja aceito.

Para executar uma nova lista, seja ela um estímulo, uma agenda ou um laço de repetição, o ator empilha um novo contexto, com a lista e a condição de repetição. No caso de um estímulo ou da agenda inicial, uma nova pilha de dados é criada também.

4.2.5. Estímulos

Um estímulo é constituído por uma lista. Ao lançar um estímulo o agente criador do estímulo, que pode ser o ator ou o mundo, pede ao executor que estimule todos os atores que ele criou. O executor então chama o método `telisme.maquina.estimuleSe`, passando como parâmetro o estímulo, para que

cada tratador de estímulo dos atores verifiquem se devem ou não tratar aquele estímulo. A verificação é feita apenas para os atores que não estão tratando um estímulo e que não tenham emitido aquele estímulo, pois em Telis um ator não atende aos próprios estímulos.

A lista de tratadores dos atores é então varrida e o método `Empilhavel.combina` da interface `telisme.maquina.Empilhavel` é chamado na lista que representa o filtro de cada tratador. Todos os elementos de uma lista, e a própria classe `Lista` implementam o método `combina`. A implementação de `Empilhavel.combina` da lista chama o método para todos os seus elementos. Por fim uma lista de tratadores que responderão ao estímulo é retornada.

As listas de execução dos tratadores são empilhadas com novos contextos, contendo uma nova pilha. Esta nova pilha contém apenas a lista que gerou o estímulo.

Este processo ocorre em de forma mutuamente excludente à obtenção da próxima ação a ser executada, eliminando a possibilidade de erros de concorrência.

Na Figura 16 observa-se a dinâmica da geração de um estímulo por um ator. Na figura, o ator 1 gera o estímulo `[10 "string"]`. Os atores 2 e 3 possuem alguns tratadores de estímulos instalados, mas apenas o ator 2 possui tratadores com filtros que aceitam o estimulo gerado. No caso observa-se o filtro `[Número Texto]` para o tratador 1 e `[10 Texto]` para o tratador 2. Em ambos os casos a chamada ao método `Lista.combina` resulta em verdadeiro, fazendo com que este estímulo seja tratado pelo ator 2.

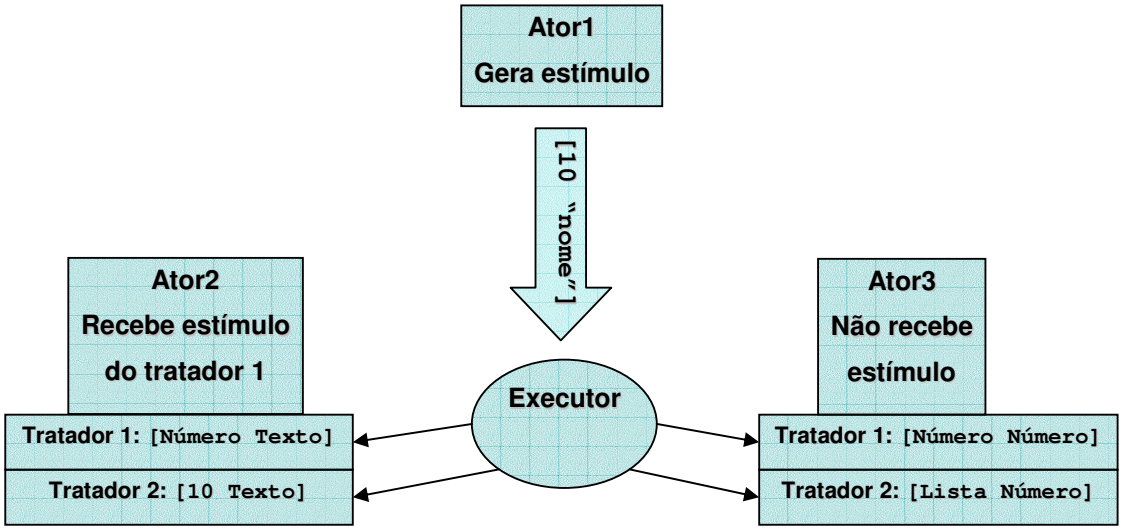

Figura 16: exemplo ilustrado de ator emitindo estímulo

4.2.6. Variáveis e agendas

No caso de variáveis e agendas, há um tratamento especial: para estes símbolos, a máquina faz uso de tabelas de espalhamento.

As agendas, como são criadas no carregamento do ator, à exceção das agendas dinâmicas, ficam contidas em uma tabela de espalhamento que é um atributo da classe `telisme.maquina.Programa`. Para invocar uma agenda o compilador gera a ação `Acao._executarAgenda`, e passa o nome da agenda como parâmetro. O nome é procurado na tabela e se encontrado, a agenda é executada.

Para as agendas dinâmicas, depois de criadas, devem também ser adicionadas ao programa do ator.

No caso das variáveis, a tabela de espalhamento é um atributo da classe `telisme.maquina.Ator`. As variáveis são adicionadas à tabela no momento da associação, que ocorre quando a ação `Acao.associar` é processada. O valor contido no topo da pilha é movido para a variável. Quando encontra uma referência a uma variável já associada, o compilador gera a ação `Acao._pushVariavel`, passa como parâmetro o nome da variável. A máquina então, ao processar a ação, procura o nome na tabela de espalhamento e, se encontrado, o valor contido na variável é copiado para o topo da pilha.

5. Conclusão

Os objetivos do trabalho foram alcançados. Telis ME manteve-se simples e poderoso como Telis, e logo nos primeiros testes, foi possível verificar a semelhança de comportamento dos apliques, quando executando na máquina Telis ou executando na máquina Telis ME.

O trabalho visava criar uma estrutura para que as primitivas fossem implementadas gradualmente em Telis ME, e esse objetivo foi alcançado, e além disso aproximadamente 35% das primitivas de Telis (na versão que foi usada), foram implementadas em Telis ME, priorizando as primitivas de comunicação, lógica e manipulação de dados. As primitivas do Dix não entraram no percentual.

Telis ME necessita de um dispositivo que suporte CLDC 1.1, e esta ainda não é a configuração mais comum encontrada na maioria dos telefones celulares, mas isto tem caráter temporal. Praticamente todos os novos telefones já estão sendo distribuídos com ela.

Observou-se também, que a performance do sistema é consideravelmente degradada a medida que se aumenta o número de atores em um aplique. Isto por que a plataforma destino ainda tem restrições de performance quanto ao paralelismo.

Uma observação particular do autor, é que Telis realmente cumpre sua função de ferramenta de aprendizado. Mesmo um estudante que não seja um iniciante é capaz de expandir seus conhecimentos ao utilizar esta ferramenta.

Com a evolução da plataforma de dispositivos móveis, a expectativa é que eles tornem-se cada vez mais próximos dos computares, aumentando sua capacidade de processamento e melhorando sua interface com o usuário. Desta forma, chegará o dia em que todo o ambiente Telis possa ser executado nestes dispositivos, sem a necessidade de uma versão especial, o que dá um caráter temporário ao Telis ME.

Até lá, Telis ME pode cumprir bem a função de estimular o aprendizado, e prover uma fonte de estudos sobre Java ME para os membros do Edugraf.

5.1. Trabalhos futuros

- Inicialmente, aumentar o número de primitivas suportados por Telis ME, para que cada vez haja menos diferenças entre as versões.
- Como citado na conclusão, Telis ME necessita da CLDC 1.1. Seria muito interessante ter uma versão ainda mais reduzida, que suportasse a CLDC 1.0, pois hoje, a maioria dos celulares ainda disponibiliza somente esta plataforma. Para isto seria necessário criar um escalonador interno e remover o tratamento de ponto flutuante, duas das maiores restrições da CLDC 1.0.
- Atualizar a versão de Telis suportada por Telis ME. Novos conceitos como comunicação direta e moldes não foram integrados, e a versão de Telis suportada por Telis ME já está defasada em um ano.
- Efetuar um estudo aprofundado sobre a performance de Telis ME, para que a ferramenta possa ser executada também em celulares com menor capacidade computacional.

6. Referências Bibliográficas

[1] Página oficial do Telis. Manual do Telis, Programando para Internet. Disponível em http://twiki.edugraf.ufsc.br/bin/view/Telis/ProgramandoNaInternet. Último acesso em 22/01/2007.

[2] Página do IBGE. Pesquisa Nacional por Amostra de Domicílios 2005. Disponível em http://www.ibge.gov.br/home/presidencia/noticias/noticia_visualiza.php?id_noticia=686. Último acesso em 22/01/2007.

[3] Página oficial do Telis. Disponível em http://twiki.edugraf.ufsc.br/bin/view/Telis/WebHome. Último acesso em 22/01/2007.

[4] Sun Developer Network. A Survey of J2ME. Disponível em http://developers.sun.com/techtopics/mobility/getstart/articles/survey/. Último acesso em 22/01/2007.

[5] JSR 118: Mobile Information Device Profile for Java 2.0 Micro Edition. Version 2.0. Disponível para download em http://jcp.org/en/jsr/detail?id=118. Último acesso em 22/01/2007.

[6] Página oficial da tecnologia Java ME. Disponível em http://java.sun.com/javame/index.jsp. Último acesso em 22/01/2007.

[7] JSR 139: Connected Limited Device Configuration Specification. Version 1.1. Disponível para download em http://jcp.org/en/jsr/detail?id=37. Último acesso em 22/01/2007.

Anexo I

O código fonte abaixo extraído da classe `br.ufsc.edugraf.telis.me.Simbolos` contém as primitivas implementadas por Telis ME até a data da impressão do trabalho.

```
crieNodo("[", TokenME.PRIMITIVA, "_abrirLista");
crieNodo("]", TokenME.PRIMITIVA, "_fecharLista");
crieNodo("+", TokenME.PRIMITIVA, "_somar");
crieNodo("-", TokenME.PRIMITIVA, "_subtrair");
crieNodo("*", TokenME.PRIMITIVA, "_multiplicar");
crieNodo("/", TokenME.PRIMITIVA, "_dividir");
crieNodo("\\", TokenME.PRIMITIVA, "_modulo");
crieNodo("mostrar", TokenME.PRIMITIVA, "mostrar");
crieNodo("mostraretiqueta", TokenME.PRIMITIVA, "mostrarEtiqueta");
crieNodo("frente", TokenME.PRIMITIVA, "frente");
crieNodo("direita", TokenME.PRIMITIVA, "direita");
crieNodo("comrastros", TokenME.PRIMITIVA, "comRastros");
crieNodo("semrastros", TokenME.PRIMITIVA, "semRastros");
crieNodo("executar", TokenME.PRIMITIVA, "executar");
crieNodo("vezesrepetir", TokenME.PRIMITIVA, "vezesRepetir");
crieNodo("prasempre", TokenME.PRIMITIVA, "praSempre");
crieNodo("sedito", TokenME.PRIMITIVA, "seDito");
crieNodo("dizer", TokenME.PRIMITIVA, "dizer");
crieNodo("descansar", TokenME.PRIMITIVA, "descansar");
crieNodo("número", TokenME.PRIMITIVA, "Numero");
crieNodo("texto", TokenME.PRIMITIVA, "Texto");
crieNodo("lista", TokenME.PRIMITIVA, "Lista");
crieNodo("teclado", TokenME.PRIMITIVA, "TECLADO");
crieNodo("soletrar", TokenME.PRIMITIVA, "soletrar");
crieNodo("comotexto", TokenME.PRIMITIVA, "comoTexto");
crieNodo("limparpilha", TokenME.PRIMITIVA, "limparPilha");
crieNodo("obternumerodeelementosdapilha", TokenME.PRIMITIVA,
        obterNumeroDeElementosDaPilha");
crieNodo("fixarcordosrastros", TokenME.PRIMITIVA, "fixarCorDosRastros");
crieNodo("fixardireção", TokenME.PRIMITIVA, "fixarDirecao");
crieNodo("obterposição", TokenME.PRIMITIVA, "obterPosicao");
crieNodo("inserirnofim", TokenME.PRIMITIVA, "inserirNoFim");
crieNodo("tirar", TokenME.PRIMITIVA, "tirar");
crieNodo("obteridentidade", TokenME.PRIMITIVA, "obterIdentidade");
crieNodo("=", TokenME.PRIMITIVA, "_igual");
crieNodo("invisível", TokenME.PRIMITIVA, "invisivel");
crieNodo("visível", TokenME.PRIMITIVA, "visivel");
crieNodo("intangível", TokenME.PRIMITIVA, "intangivel");
crieNodo("tangível", TokenME.PRIMITIVA, "tangivel");
crieNodo("severdade", TokenME.PRIMITIVA, "seVerdade");
crieNodo("sefalso", TokenME.PRIMITIVA, "seFalso");
```

```
crieNodo("geraraleatório", TokenME.PRIMITIVA, "gerarAleatorio");
crieNodo("andarpara", TokenME.PRIMITIVA, "andarPara");
crieNodo("fixaríconesimples", TokenME.PRIMITIVA, "fixarIcone");
crieNodo("fixarícone", TokenME.PRIMITIVA, "fixarIcone");
crieNodo("obterdireção", TokenME.PRIMITIVA, "obterDirecao");
crieNodo("absoluto", TokenME.PRIMITIVA, "absoluto");
crieNodo("obtercordoponto", TokenME.PRIMITIVA, "obterCorDoPonto");
crieNodo("obtercordosrastros", TokenME.PRIMITIVA, "obterCorDosRastros");
crieNodo("suicidar", TokenME.PRIMITIVA, "suicidar");
crieNodo("trocar", TokenME.PRIMITIVA, "trocar");
crieNodo("copiar", TokenME.PRIMITIVA, "copiar");
crieNodo("obtertamanho", TokenME.PRIMITIVA, "obterTamanho");
crieNodo("maiorouigual", TokenME.PRIMITIVA, "maiorOuIgual");
crieNodo("menorouigual", TokenME.PRIMITIVA, "menorOuIgual");
crieNodo("maiorque", TokenME.PRIMITIVA, "maiorQue");
crieNodo("menorque", TokenME.PRIMITIVA, "menorQue");
crieNodo("primeiro", TokenME.PRIMITIVA, "primeiro");
crieNodo("semprimeiro", TokenME.PRIMITIVA, "semPrimeiro");
crieNodo("enquantofalso", TokenME.PRIMITIVA, "enquantoFalso");
crieNodo("substituirelemento", TokenME.PRIMITIVA, "substituirElemento");
crieNodo("obterelemento", TokenME.PRIMITIVA, "obterElemento");
crieNodo("entãosenão", TokenME.PRIMITIVA, "entaoSenao");
crieNodo("verdadeiro", TokenME.PRIMITIVA, "_pushVerdadeiro");
crieNodo("falso", TokenME.PRIMITIVA, "_pushFalso");
crieNodo("contargiro", TokenME.PRIMITIVA, "contarGiro");
crieNodo("carimbar", TokenME.PRIMITIVA, "carimbar");
```

TELIS ME: UMA VERSÃO DE TELIS PARA DISPOSITIVOS MÓVEIS

Marcus Vinicius Cruz Xavier[1], Luiz Fernando Bier Melgarejo[2]

Curso de Bacharelado em Ciências da Computação
Departamento de Informática e Estatística
Universidade Federal de Santa Catarina , Brasil, 88040-900
{vinix[1], melga[2]}@inf.ufsc.br

RESUMO

Assim como o interesse pelo desenvolvimento de aplicações para Internet explodiu alguns anos atrás, decorrente da popularização da rede, o interesse pelo desenvolvimento de aplicações para dispositivos móveis está aumentando sensivelmente, devido principalmente à disseminação dos telefones celulares.

O potencial desta plataforma em vista de sua grande popularidade é enorme, e como no caso da Internet, o desenvolvimento de software para celulares requer também um razoável domínio de conceitos e técnicas.

Telis é uma ferramenta de aprendizagem de programação, desenvolvida com o intuito de simplificar conceitos e técnicas, e integrar-se à tecnologias relacionadas a Internet. O objetivo deste trabalho é criar um modo de executar programas feitos em Telis em dispositivos móveis, mais especificamente em telefones celulares.

Palavras chave: dispositivos móveis, linguagens, Telis, JME.

ABSTRACT

Like the interest for Internet application development exploded few years ago, due the popularization of the network, the interest for building mobile devices applications is sensibly growing, due the dissemination of cell phones.

The potential of this platform, considering its popularity is huge, and like is the case of the Internet, the development of software for this platform requires a reasonable domain of concepts and techniques.

Telis is a programming learning tool, developed to simplify programming concepts and techniques, and integrate itself to Internet related technologies. The goal of this work is to create a way of execute programs developed in Telis in mobile devices, specifically in cell phones.

Keywords: mobile devices, languages, Telis, JME.

INTRODUÇÃO

Com o objetivo de diminuir a curva de aprendizado de programação, surgiu a linguagem Telis. Desenvolvida pela equipe do Edugraf, laboratório de software educacional – INE – CTC – UFSC, coordenado pelo professor Luis Fernando Bier Melgarejo, esta linguagem de programação se baseia em antecessores poderosos e simples como Forth, Logo e SmallTalk, e mantém características de distribuição e execução semelhantes a Ágora [1].

Telis foi desenvolvida de modo que todo seu ambiente propicia uma rica interação com a Internet, aproveitando o entusiasmo causado pela popularização da rede para estimular o interesse dos estudantes. A idéia de fazer com que Telis passe a interagir com dispositivos móveis surgiu do mesmo princípio.

Segundo dados do IBGE, o número de residências com telefones celulares superou o número de residências com telefones fixos nos lares brasileiros já em 2005 [2], o que demonstra a rápida difusão desta tecnologia na sociedade brasileira.

A grande dificuldade da concretização deste trabalho reside nas diferenças existentes entre a capacidade computacional dos computadores pessoais e a dos dispositivos móveis.

Para alcançar o objetivo do trabalho, a abordagem adotada foi a implementação de um compilador, integrado ao ambiente de Telis, que transforma um aplique publicado para Internet, em um aplique publicado para plataforma móvel, e um programa que executa o aplique em um dispositivo móvel compatível com a tecnologia Java ME.

Este conjunto de programas foi chamado de Telis ME, uma analogia à própria plataforma Java ME, sobre o qual é escrita a máquina virtual.

TELIS

Telis é totalmente implementada em Java. As aplicações (ou apliques) feitas em Telis são *applets* Java contendo os programas que são interpretados por uma máquina Telis que é executada em um navegador que possua uma JVM embutida.

O escopo principal das aplicações Telis é basicamente exibir e movimentar objetos gráficos e colher entradas de usuários, para que possa haver interação. Estes são requisitos típicos de jogos e simuladores, duas das áreas mais exploradas pelos alunos quando estudando Telis. Estas são também duas

das principais áreas de desenvolvimento em dispositivos móveis. A própria JME disponibiliza APIs especializadas em programação de jogos.

A linguagem Telis

Telis oferece o aprendizado através do uso, simplificando conceitos e técnicas. Estas características fazem de Telis uma ferramenta apropriada ao aprendizado. Alguns exemplos de conceitos e técnicas simplificados em Telis são:

- Orientação a objetos;
- Estruturas de dados;
- Paralelismo;
- Comunicação de dados;
- Aplicações distribuídas;
- Manipulação de objetos gráficos.

Além disso, Telis possui características peculiares, algumas delas herdadas diretamente de outras linguagens:

- Operadores pós-fixos;
- Operações realizadas direto na pilha;
- Tipagem dinâmica;
- Palavras reservadas em português.
- Tratamento de eventos feito através de uma estrutura de estímulos e tratadores;

Algumas destas características, entre outras, fazem de Telis uma linguagem propícia a ser usada no aprendizado de programação.

Conceitos e nomenclaturas de Telis

Seguem alguns conceitos e nomenclaturas utilizados em Telis, úteis para o entendimento deste trabalho[3]:

- **Aplique**: é uma aplicação Telis;
- **Ator**: é o agente de Telis, o equivalente à uma instancia de um objeto;
- **Mundo**: é o painel de visualização do aplique;
- **Modelo**: é o equivalente à classe;
- **Agenda**: é o equivalente à um método;
- **Agenda Inicial**: é o equivalente à um construtor de uma classe. É a agenda que é executada quando um ator é criado;
- **Agenda Primitiva**: é uma instrução mínima para o interpretador. Também são chamadas apenas de *primitivas*;
- **Pilha**: é a pilha de dados manipulada por um ator. Em uma execução de um aplique Telis com vários atores, cada ator terá a sua própria pilha;
- **Lista**: é um conjunto de dados, primitivas ou agendas. Qualquer lista pode ser executada, ou disparada como um estímulo;
- **Estímulo**: é a forma de implementação de tratamento de eventos em Telis;
- **Tratador**: é o conjunto formado por um filtro de estímulos e uma lista de tarefas a serem executadas pelo ator;

JAVA PARA DISPOSITIVOS MÓVEIS (JME)

A tecnologia Java é composta por três partes: a linguagem Java, a máquina virtual Java (JVM) e a API (*Application Programming Interface*, conjunto de bibliotecas de classe) Java. O conjunto formado por estes componentes é o que caracteriza a tecnologia Java, e a variação dos recursos disponíveis em cada componente, é o que caracteriza a versão disponibilizada para uma determinada plataforma.

A comunidade Java através do JCP (*Java Community Process*), especifica três plataformas para tecnologia Java: Java SE (*Standard Edition*) voltada para aplicações em computadores pessoais, Java EE (*Enterprise Edition*) voltada para servidores e Java ME (*Micro Edition*) voltada para dispositivos móveis. Esta última é a que concerne o desenvolvimento de softwares para dispositivos móveis, como celulares e PDAs (*Personal Data Assistant*), e é a plataforma de destino da migração de Telis.

A J2ME, como também é conhecida à versão atual da plataforma Java ME (JME), é o conjunto de especificações das tecnologias envolvidas no processo de desenvolvimento, distribuição e execução de softwares para dispositivos móveis em Java: a máquina virtual, a linguagem, as bibliotecas envolvidas no núcleo da linguagem, as bibliotecas adicionais, denominadas *profiles*, e os conjuntos de bibliotecas opcionais denominadas *optional packages*.

A especificação do núcleo das tecnologias envolvidas é chamada de configuração. Uma configuração é composta pela especificação de uma máquina virtual, das restrições à linguagem, e das chamadas bibliotecas de núcleo (*core libraries)*.

Duas configurações são definidas pela JME: a CDC (*Connected Device Configuration*) que abrange os PDAs com maior capacidade computacional e a CLDC (*Connected Limited Device Configuration*) que abrange os dispositivos que possuem recursos mais escassos, entre eles os telefones celulares, foco deste trabalho.

Máquina Virtual Java (JVM)

A JVM (*Java Virtual Machine*) foi inicialmente projetada para ser executada em dispositivos móveis (PDAs) sob a sigla de KVM (*K Virtual Machine*). Isto faz com que o esforço atual de migração de Java para dispositivos móveis seja considerado uma "volta às raízes" [4].

As máquinas virtuais que as empresas disponibilizam em seus aparelhos, seguindo as especificações da CLDC, são chamadas KVM.

Para cada configuração, é escolhido um subconjunto de instruções ou todo o conjunto, dependendo da disponibilidade de recursos de cada dispositivo. Desta forma o código interpretado pela JVM não é totalmente intercambiável entre plataformas.

As limitações impostas pelas configurações foram determinantes no processo de decisão da abordagem a ser tomada para a migração de Telis.

Além de impedir aproveitamento da máquina virtual de Telis, criou a necessidade de um método otimizado para a execução, uma vez que a capacidade processamento nas plataformas móveis é muito limitado.

Bibliotecas de núcleo

As bibliotecas de núcleo, chamadas de *core libraries*, denotam as bibliotecas envolvidas com a linguagem (`java.lang.*`, `java.util.*`), as bibliotecas de entrada e saída (`java.io.*`) e dos recursos que envolvem segurança, redes e internacionalização.

Estas bibliotecas são, na JME, um subconjunto das bibliotecas disponibilizadas na JSE, sendo definido que se uma classe está presente ela deve ser idêntica ou um subconjunto da classe equivalente distribuída na JSE.

Nenhum novo recurso pode ser adicionado e a semântica dos métodos e atributos deve permanecer a mesma, se existirem [7].

MIDP

Além dos componentes que formam uma configuração, JME disponibiliza bibliotecas adicionais, que são incompatíveis com as outras versões de Java. Estas bibliotecas são construídas sobre as configurações CDC e CLDC, e são chamadas de *profiles*. Além dos *profiles* outras especificações de bibliotecas são disponibilizadas, e chamadas de *optional packages.* Estas são bibliotecas opcionais pois definem APIs para recursos que podem ou não estar disponíveis em um determinado dispositivo.

MIDP (*Mobile Information Device Profile*) é o principal conjunto de bibliotecas construído sobre a plataforma ME, e é praticamente um padrão. Outros *profiles* menos conhecidos foram desenvolvidos: *KittyHawk*, *PDAP* (extensão do MIDP, ainda não terminado), *i-appli* (desenvolvido antes do MIDP).

MIDP está disponível para as configurações CDC e CLDC. A plataforma alvo de Telis ME são os telefones celulares, que são abrangidos pela CLDC.

MIDP oferece bibliotecas para os seguintes escopos:

- Exibição de gráfica;
- Leitura de entrada dados;
- Armazenamento persistente;
- Envio e recebimento de mensagens;
- Comunicação de dados;
- Telefonia sem fio.

O ambiente gráfico e os recursos de interação com o usuário, foram implementados em Telis ME usando o pacote `javax.microedition.lcdui.game`. Este pacote disponibiliza funcionalidades que satisfazem vários dos requisitos gráficos de um aplique Telis.

Um recurso útil fornecido por esta biblioteca é o conceito de manipulação de gráficos em camadas. Em Telis ME foram utilizados dois tipos de camadas fornecidos pelo pacote, uma para representar o fundo (`TiledLayer`) e outro para representar os atores se movendo na tela (`Sprite`). As camadas são gerenciadas pela classe `LayerManager`.

TELIS ME

Telis ME é o conjunto de programas que viabiliza a execução de código Telis em ambientes móveis, que suportem a tecnologia Java ME, ou seja, Telis ME é o produto final deste trabalho. É constituído de um compilador e um interpretador.

Como em Telis, um programa em Telis ME é chamado de aplique, e cada aplique é constituído de um ou mais atores e suas agendas. Cada ator possui uma área de dados (pilha) própria e um fluxo de execução independente dos outros atores que compõe o aplique.

A compilação é executada em cinco etapas, cada uma gerando produtos parciais que são usados como entrada da próxima etapa. Estas etapas serão discutidas posteriormente.

O produto final da transformação é um par de arquivos de extensão `.jad` e `.jar`. O arquivo `.jad` descreve o que contém o arquivo `.jar`, e este é o pacote de classes Java que denotam o programa. A plataforma de destino, seja um simulador ou um telefone celular que suporte Java, lê o arquivo `.jad` e oferece ao usuário a execução dos programas, chamados *Midlets*. Um aplique é um único *Midlet*, contendo todos os dados necessários para sua execução, incluindo a máquina virtual e a programação dos atores e suas agendas.

A execução é basicamente a tradução do código intermediário em ações que refletem alterações nos estados dos atores. Esta etapa foi otimizada, e o que acontece é que a interpretação é feita parcialmente na compilação e parcialmente na execução.

Diferentemente do Telis, um aplique Telis ME não é interpretado diretamente de um arquivo XML ou de qualquer outro arquivo armazenado. Ele passa por um processo de tradução e otimização que gera uma saída mista. Diz-se que Telis ME utiliza uma mescla de compilação e interpretação.

O compilador

O compilador Telis ME é embutido dentro do código do Telis. Ele utiliza o *parser* XML do ambiente Telis e obtém informações de diretório das configurações do próprio ambiente.

Ele é instanciado de forma independente, mas utiliza várias classes do pacote que contém a maquina virtual do Telis (`br.ufsc.edugraf.telisme.maquina`), além do conversor de XML para aplique.

A entrada do compilador é o aplique Telis já publicado para *Web*, em formato XML, e o produto final gerado pela transformação é a dupla de arquivos citados na introdução, o `[aplique].jad` e o `[aplique].jar`, como mostra a Figura 4:

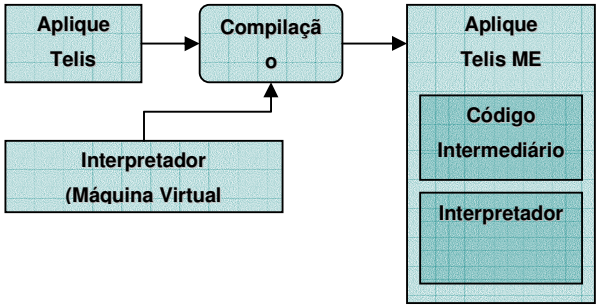

Figura 17: em Telis ME o código do aplique é distribuído junto ao interpretador

Implementação

Na otimização da execução do aplique, o ponto crucial é o formato em que ele é armazenado. Se o Telis ME se comportasse como o Telis, a máquina virtual Telis ME seria responsável pela interpretação de um arquivo XML, e teria seu tamanho aumentado consideravelmente dada a inclusão de um *parser*. Além disso, a interpretação do arquivo XML consumiria tempo de processamento e memória volátil, recursos que em dispositivos móveis são escassos.

Dadas estas restrições a abordagem selecionada dentre várias, foi a de traduzir código Telis em código Java, compilar o código Java e gerar um aplique já no formato de *bytecode* compatível com a CLDC. Esta abordagem tem a vantagem de não necessitar de *parsing* para interpretação das instruções e também de validar o código antes de ele ser executado em um dispositivo.

O código gerado é dividido em classes que representam os modelos de ator derivadas da classe `telisme.maquina.Ator` e uma classe derivada da classe `telisme.maquina.Executor` que instancia os atores e cria seus fluxos de execução. As classes `Ator` e `Executor` estão contidas na máquina Telis ME, e estendem classes do MIDP.

As classes de atores programam suas agendas através da chamada do método `Ator.adicione`, que cria e adiciona ações nas listas de execução que representam as agendas. Cada operação sobre um ator é chamada de ação, e está diretamente ligada a pelo menos um símbolo contido no código das agendas dos atores compilados.

Etapas do processo de transformação

O processo de transformação do aplique publicado para *Web* para o aplique publicado para dispositivos móveis é composto por várias etapas. Estas etapas foram separadas de acordo com a geração de produtos parciais:

6. **Tradução:** à partir dos objetos gerados pelo conversor de XML do Telis, o compilador Telis ME traduz o código Telis no código intermediário.
7. **Compilação:** neste ponto ocorre o processo de otimização que transforma o código intermediário gerado em código CLDC.
8. **Verificação:** a máquina virtual Java ME exige que o *bytecode* Java seja verificado por uma ferramenta chamada *Preverify*, disponível no JWT, antes que possa ser distribuída.
9. **Empacotamento:** criação do `.jar`.
10. **Publicação:** geração do arquivo `.jad` e envio para o servidor, caso seja necessário.

Estas etapas envolvem diversos componentes além do compilador em si. São envolvidos no processo, classes do Telis, *scripts* de sistema operacional, e programas do próprio Java, como o compilador `javac.exe` e o verificador de código para JME `preverify.exe`.

Tradução de código

O processo de tradução do código, é a geração de código Java, à partir de código Telis. É nesta fase que ocorre a maior interação entre o Telis e Telis ME, pois parte da interpretação do código é feita no código do ambiente Telis. Isto ocorre através de chamadas à métodos de classes do ambiente e da máquina Telis.

A compilação do modelo basicamente é uma troca de símbolos. Para cada símbolo encontrado no código do aplique, um ou mais símbolos são gerados na saída. Os símbolos gerados são *strings* de constantes definidas na classe `telisme.maquina.Acao`, da máquina virtual Telis ME, sendo adicionados a uma lista, através do método `Ator.adicione`. Para cada agenda é definida uma lista separada.

Para a maioria das primitivas, a tradução é feita de forma direta, como mostra a Figura 7.

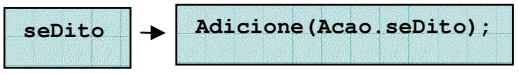

Figura 18: Transformação do seDito em uma ação em Telis ME

As exceções a esta regra são as constantes, as associações de variável e as chamadas de agenda. A principal diferença nestes casos é a adição de um parâmetro a ação.

- **Constantes:** as constantes são avaliadas pelo compilador e marcadas como texto ou número. Esta diferença é necessária na hora de avaliar um filtro de estímulos de um tratador.
- **Variáveis:** O tratamento de variáveis é uma tarefa diferenciada tanto em Telis como em Telis ME. Quando encontra uma associação, ao invés de gerar um objeto de token, o interpretador da máquina virtual Telis gera um objeto de instrução.

O compilador do Telis ME gera uma ação (`Acao.associar`) e passa como parâmetro o nome da variável quê será criada e receberá valor contido no topo da pilha.

- **Agendas:** as agendas são incluídas na tabela de símbolos do modelo no momento da criação do modelo, uma vez que já são todas conhecidas.

Compilação e publicação

A saída do compilador arquivos `.java` contendo o código fonte de cada modelo, e o código fonte do executor, que é a classe derivada da classe `Executor` da máquina Telis ME. A classe `Executor` estende a classe `MIDlet`, do pacote `java.microedition.midlet`, e é o ponto de partida do aplique.

As classes são geradas em um diretório pré-definido, abaixo do diretório raiz de Telis, que contém algumas ferramentas necessárias para a conclusão da transformação.

Os apliques são gerados em um diretório com seu nome, sem que desta forma cada aplique é isolado pelo seu pacote, na hora da compilação.

Depois da compilação, e antes das fases posteriores, o arquivo `MANIFEST.MF` é gerado.

Um *script* de sistema operacional chama o `javac.exe`, para compilar o código fonte do aplique. Em seguida o *script* chama o `preverify.exe`, O resultado são arquivos `.class` que já contém o aplique e a máquina virtual Telis ME, com *bytecode* já compatível com a CLDC.

Após verificados, o *script* chama o utilitário `jar.exe` que gera o pacote no diretório de destino, selecionado pelo usuário. Finalmente o arquivo `.jar` é gerado. Ao final do processo , um arquivo de *log* `BuildLog.txt` é gerado no diretório de saída.

Se o aplique estiver sendo publicado na Web, então o processo de compilação acontece em um diretório temporário, e só depois de obtidos os arquivos `.jar` e `.jad` é que eles são enviados para o servidor.

MÁQUINA VIRTUAL TELIS ME

A máquina virtual Telis ME é basicamente um conjunto de classes que denotam os conceitos básicos de Telis, como atores, listas, mundo, etc... além de recursos que são conhecido apenas internamente nesta versão, como ações, tipos, contadores, e algumas interfaces largamente utilizadas durante a implementação.

Basicamente um objeto executor ativa atores contidos em uma lista. Os atores, tem suas agendas programadas na construção dos objetos, e executam cada um em uma nova *thread*. As ações programadas nas agendas alteram os estados internos dos atores, em especial a pilha de dados e a pilha de contextos.

A execução da agenda inicial pode ser interrompida quando o ator receber um estímulo e estiver programado para tratar aquele estímulo. Neste caso, um outro contexto é empilhado, e o contexto original só é restaurado ao fim da execução do estímulo.

Inicialização

Um programa JME inicia no método `startApp` de uma classe que estenda a classe `Midlet`, sendo esse então o ponto inicial da execução do aplique. A classe `Executor` é a responsável por isso na máquina Telis ME.

A classe `ExecutorDeAplique` apenas adiciona os atores instanciados à lista `atores`, atributo da classe `Executor`. Todo o código de ativação dos atores está na classe base.

Como em Telis, os atores executam suas tarefas em um mundo, que neste caso um objeto `GameCanvas` associado a um *form* da biblioteca MIDP. Este *form* é criado no momento da chamada do `startApp` e passado para cada ator instanciado.

Cada ator é associado a uma *thread* e todo o paralelismo da linguagem é baseado neste recurso.

Inicialmente antes de entrar no seu ciclo de vida, o ator empilha em seu contexto a sua agenda inicial, programando-a para ser executada uma única vez.

Ciclo de vida do ator

Quando ativo, um ator está em um laço de repetição executado na implementação do método abstrato `Runnable.run`. A cada repetição o método `Ator.viva` é executado, e quando seu valor de retorno é falso o ator morre, ou em outras palavras a *thread* é finalizada naturalmente pela saída do método `Runnable.run`.

A cada chamada do método `Ator.viva`, uma ação é executada ou uma lista, que pode ser executada posteriormente, é criada.

Na entrada do método, a lista em execução é lida do topo da pilha de listas para execução. Se ainda houverem elementos a serem processados nesta lista, o ator processa este elemento, ou cria novas listas para serem empilhadas.

Se não houverem mais itens nesta lista, uma avaliação é feita na condição de execução desta lista, e se a condição for verdadeira, a lista é reiniciada e seu primeiro elemento é executado. Se a condição for falsa, uma nova lista é elegida para ser executada, sendo esta retirada do topo da pilha de listas para execução, que é alimentada também pelos estímulos recebidos pelo ator.

Se não houverem mais listas a serem executadas, o ator entra no estado de hibernação, de onde só sai quando morre ou quando recebe um estímulo.

Execução de uma ação

Como vimos anteriormente, cada ação a ser executada no método `Ator.viva` é lida de uma lista, que pode ser uma lista em execução, uma agenda ou um estímulo. Quando encontradas ações de criação de listas a execução vai para um estado especial, em que os elementos lidos da lista de execução, não são executados, e sim armazenados em uma nova lista, para futuro uso, ou processamento.

Em execução normal, um elemento lido da lista de execução é uma ação, ou é uma lista que está contida na lista que está em execução. Neste caso, a lista é empilhada, e o ciclo continua, procurando pelo próximo elemento da lista de execução.

No caso normal, o elemento lido de uma lista em execução é uma ação, que é processada de acordo com o seu código.

As operações são realizadas na pilha de dados. A pilha de dados utiliza tipos abstratos para manipular os dados, através da interface `Empilhavel`, como podemos observar na Figura 14.

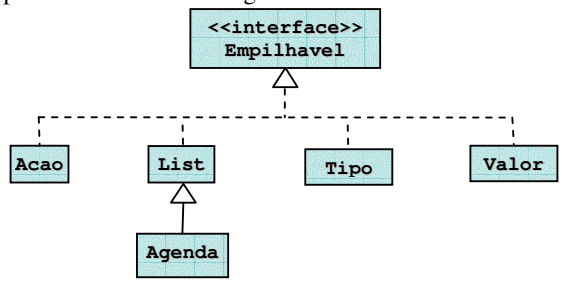

Figura 19: classes que implementam a interface Empilhavel

Contextos

Cada ator possui uma pilha de execução isolada, e além disso, uma nova pilha é criada para ele cada vez que ele vai tratar um estímulo. Este controle é feito por uma pilha de contextos, e esta pilha é alterada por determinadas situações, como mostra a Figura 15:

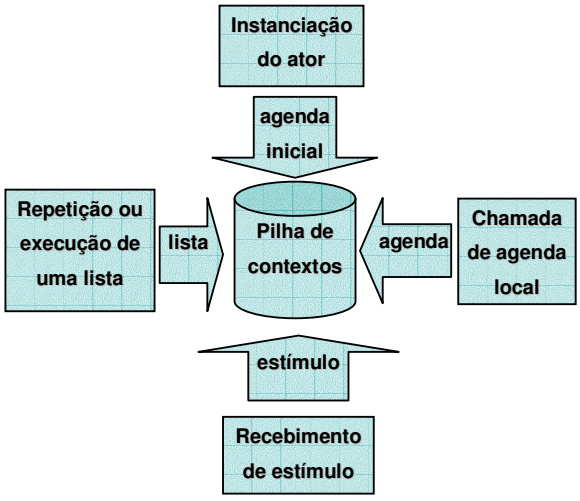

Figura 20: possíveis alterações na pilha de contextos

Os atributos de um contexto são: a pilha atual, a lista em execução, os dados necessários para a criação das listas contidas em uma agenda ou estímulo, a condição de repetição da lista de execução e uma marca que informa se aquele contexto foi gerado por um estímulo.

Estímulos

Um estímulo é constituído por uma lista. Ao lançar um estímulo o agente criador do estímulo, que pode ser o ator ou o mundo, pede ao executor que estimule todos os atores que ele criou. O executor então chama o método `telisme.maquina.estimuleSe`, passando como parâmetro o estímulo, para que cada tratador de estímulo dos atores verifiquem se devem ou não tratar aquele estímulo. A verificação é feita apenas para os atores que não estão tratando um estímulo e que não tenham emitido aquele estímulo, pois em Telis um ator não atende aos próprios estímulos.

Este processo ocorre em de forma mutuamente excludente à obtenção da próxima ação a ser executada, eliminando a possibilidade de erros de concorrência.

Variáveis e agendas

No caso de variáveis e agendas, há um tratamento especial: para estes símbolos, a máquina faz uso de tabelas de espalhamento.

As agendas, como são criadas no carregamento do ator, à exceção das agendas dinâmicas, ficam contidas em uma tabela de espalhamento que é um atributo da classe que representa o programa do ator. Para invocar uma agenda o compilador gera a ação `Acao._executarAgenda`, e passa o nome da agenda como parâmetro. O nome é procurado na tabela e se encontrado, a agenda é executada.

Para as agendas dinâmicas, depois de criadas, devem também ser adicionadas ao programa do ator.

No caso das variáveis, a tabela de espalhamento é um atributo da classe `telisme.maquina.Ator`. As variáveis são adicionadas à tabela no momento da associação, que ocorre quando a ação `Acao.associar` é processada. O valor contido no topo da pilha é movido para a variável. Quando encontra uma referência a uma variável já associada, o compilador gera a ação `Acao._pushVariavel`, passa como parâmetro o nome da variável. A máquina então, ao processar a ação, procura o nome na tabela de espalhamento e, se encontrado, o valor contido na variável é copiado para o topo da pilha.

CONCLUSÃO

Os objetivos do trabalho foram alcançados. Telis ME manteve-se simples e poderoso como Telis, e logo nos primeiros testes, foi possível verificar a semelhança de comportamento dos apliques, quando

executando na máquina Telis ou executando na máquina Telis ME.

O trabalho visava criar uma estrutura para que as primitivas fossem implementadas gradualmente em Telis ME, e esse objetivo foi alcançado, e além disso aproximadamente 35% das primitivas de Telis (na versão que foi usada), foram implementadas em Telis ME, priorizando as primitivas de comunicação, lógica e manipulação de dados. As primitivas do Dix não entraram no percentual.

Telis ME necessita de um dispositivo que suporte CLDC 1.1, e esta ainda não é a configuração mais comum encontrada na maioria dos telefones celulares, mas isto tem caráter temporal. Praticamente todos os novos telefones já estão sendo distribuídos com ela.

Observou-se também, que a performance do sistema é consideravelmente degradada a medida que se aumenta o número de atores em um aplique. Isto por que a plataforma destino ainda tem restrições de performance quanto ao paralelismo.

Uma observação particular do autor, é que Telis realmente cumpre sua função de ferramenta de aprendizado. Mesmo um estudante que não seja um iniciante é capaz de expandir seus conhecimentos ao utilizar esta ferramenta.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] Página oficial do Telis. Manual do Telis, Programando para Internet. Disponível em http://twiki.edugraf.ufsc.br/bin/view/Telis/ProgramandoNaInternet. Último acesso em 22/01/2007.

[2] Página do IBGE. Pesquisa Nacional por Amostra de Domicílios 2005. Disponível em http://www.ibge.gov.br/home/presidencia/noticias/noticia_visualiza.php?id_noticia=686. Último acesso em 22/01/2007.

[3] Página oficial do Telis. Disponível em http://twiki.edugraf.ufsc.br/bin/view/Telis/WebHome. Último acesso em 22/01/2007.

[4] Sun Developer Network. A Survey of J2ME. Disponível em http://developers.sun.com/techtopics/mobility/getstart/articles/survey/. Último acesso em 22/01/2007.

[5] JSR 118: Mobile Information Device Profile for Java 2.0 Micro Edition. Version 2.0. Disponível para download em http://jcp.org/en/jsr/detail?id=118. Último acesso em 22/01/2007.

[6] Página oficial da tecnologia Java ME. Disponível em http://java.sun.com/javame/index.jsp. Último acesso em 22/01/2007.

[7] JSR 139: Connected Limited Device Configuration Specification. Version 1.1. Disponível para download em http://jcp.org/en/jsr/detail?id=37. Último acesso em 22/01/2007.

Código Fonte

O Código fonte do trabalho, está disponível no CD, no arquivo `/TelisWorkspace.zip`, pois é muito extenso para ser anexado ao relatório. Além disso, o código foi compactado, pois o sistema de arquivos do CD não suporta o tamanho dos nomes completos das classes.

Os códigos podem ser encontrados nos seguintes diretórios, dentro do arquivo .ZIP:

Máquina Virtual Telis ME:

`/TelisWorkspace/TelisME_MaquinaVirtual/telisme/maquina`

Compilador ME:

`/TelisWorkspace/Telis/br/usfc/edugraf/telis/me/compilador`