UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

EDUARDO MARTINS MACHADO

Framework para Criação de Sistemas Supervisórios Dinâmicos em Dispositivos Móveis

Dissertação apresentada como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

Área de concentração: Engenharia de Computação e Sistemas Embarcados.

Prof. Dr. Carlos Eduardo Pereira
Orientador

Porto Alegre, março de 2012.

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Machado, Eduardo Martins

Framework para Criação de Sistemas Supervisórios Dinâmicos em Dispositivos Móveis / Eduardo Martins Machado – Porto Alegre: Programa de Pós-Graduação em Computação, 2012.
72 f.:il.

Orientador: Carlos Eduardo Pereira.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul, Instituto de Informática, Programa de Pós-Graduação em Computação, Porto Alegre, BR – RS, 2012.

1. Sistemas supervisórios. 2. Sistemas dinâmicos. 3. Computação móvel. I. Pereira, Carlos Eduardo, orient. II. Título.

Agradecimentos

Aos meus pais, Waldir e Maria do Rosário, que sempre estiveram presentes em todos os passos que dei até hoje, sempre me ensinando, amparando, motivando e principalmente acreditando, mesmo em momentos que nem eu mais tinha *fé*. "Se enxerguei longe foi porque me sentei nos ombros de gigantes."

À Helena, que desde pequeno sempre esteve presente e contribuiu na minha educação, com muito carinho.

Aos meus avôs, Armando e Jâime, *in memorian*, pelos pais maravilhosos que me deram.

Às minhas avós, Lilia e Dejanira, também pelos pais que me deram, e pelo carinho, amor, e a cada momento que passamos juntos.

À minha irmã, Renata, pelo apoio em vários momentos na vida, e pelas lágrimas sinceras quando me mudei para Porto Alegre.

À Lidiane, por somar sua vida à minha e pelo companheirismo a cada minuto gasto neste trabalho, por ideias, sugestões e inclusive algumas revisões.

Aos amigos que contribuíram com a amizade e em alguns momentos inclusive para com este trabalho, e em especial: Reiner e Dárlinton.

E não menos importante, ao meu orientador, Carlos Eduardo que me incentivou a superar todas as dificuldades, ajudou com todo empenho na minha adaptação ao novo estado e sempre fez o máximo para que o resultado do curso gerasse um bom trabalho; sem sua dedicação este trabalho não seria possível.

A todas essas pessoas, o meu mais sincero agradecimento: **Muito obrigado!**

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

3G	3ª Geração
API	Application Programming Interface
AWT	Abstract Window Toolkit
CDC	Connected Device Configuration
CLDC	Connected Limited Device Configuration
FP	Foundation Profile
GPRS	General Packet Radio Service
GUI	Graphical User Interface
HTTP	HyperText Transfer Protocol
IHM	Interface Homem-Máquina
IP	Internet Protocol
JAR	Java ARchive
Java EE	Java Enterprise Edition (J2EE)
Java ME	Java Micro Edition (J2ME)
Java SE	Java Standard Edition (J2SE)
JNI	Java Native Interface
JSR	Java Specification Request
JVM	Java Virtual Machine
KVM	Kylobyte Virtual Machine
MIDP	Mobile Information Device Profile
OSGi	Open Services Gateway Initiative
OTA	Over The Air
PBP	Personal Basis Profile
PDA	Personal Digital Assistent
PP	Personal Profile
RMI	Remote Method Invocation
SCADA	Supervisory Control and Data Acquisition

SDK	Software Development Kit
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UI	User Interface
UML	Unified Modeling Language
URL	Uniform Resource Locator
VB .Net	Visual Basic .Net
VM	Virtual Machine
WiFi	Wireless Fidelity
XML	eXtensible Markup Language

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

Esta dissertação apresenta o *framework* DynamicML para o desenvolvimento de aplicações (sistemas supervisórios) dinâmicas para telefones celulares.

A proposta é que o aplicativo seja criado usando a linguagem DynamicML; assim, através deste arquivo, que é enviado ao dispositivo móvel (com o *framework* previamente instalado), este é então interpretado e, finalmente, a aplicação gerada. Além disso, a qualquer momento o arquivo de configuração pode ser alterado e isso se refletirá imediatamente na aplicação. Assim, torna-se possível a adaptação/evolução de aplicações na plataforma Java ME em tempo de execução, além de proporcionar uma maneira de desenvolver aplicações de forma rápida, e com qualidade; abstraindo do desenvolvedor a maior parte da complexidade de programação.

A fim de permitir maior portabilidade nos celulares atuais, a proposta é implementada usando Java ME (Java Micro Edition) tendo como principal alvo a configuração CLDC (a mais difundida nos aparelhos atuais), mas que também é suportada na configuração CDC.

Existem vários esforços para permitir o carregamento de componentes em tempo de execução na plataforma Java ME: alguns têm como alvo a configuração CDC, e consequentemente exigem *hardware* avançado e/ou específico; outros têm propostas que exigem um enorme esforço de desenvolvimento, exigindo uma implementação para cada plataforma alvo.

A principal vantagem deste trabalho está no uso da DynamicML como uma metalinguagem para a descrição de aplicações, o que torna possível gerar a aplicação dinamicamente. Dessa forma, não é necessária nenhuma biblioteca extra (indisponível na maioria dos aparelhos) e nem o desenvolvimento de uma solução para cada plataforma alvo.

A validação da proposta foi realizada através do desenvolvimento de um protótipo aplicado a três casos de uso nas áreas de automação residencial e supervisão industrial. O protótipo consiste num sistema para celulares usando o *framework*, e uma arquitetura web para interagir com os sistemas supervisionados e simular o funcionamento do conjunto para demonstração.

Palavras-Chave: Sistemas supervisórios. Sistemas dinâmicos. Computação móvel.

Framework for Building Dynamical Supervisory Systems in Mobile Devices.

ABSTRACT

This dissertation presents the DynamicML, a framework for developing dynamic applications (supervisory systems) for mobile phones.

The proposal is that the application is created using the DynamicML language, so through this file, which is sent to the mobile device (pre-installed with the framework), is then interpreted and, finally, the application is generated. Also, the configuration file may be changed at any time and this will immediately reflect on the application. Thus, it becomes possible adaptation/evolution of applications at runtime on the Java ME platform, while providing a way to quickly develop applications, with quality; and abstracting most of the complexity of programming from the developer.

In order to allow greater portability in the today cell phones, the proposal is implemented using the Java ME (Java Micro Edition) having as the primary target the CLDC configuration (the most widespread in the current devices), but that is also supported in the CDC configuration.

There are several efforts to allow the loading of components at runtime in Java ME platform: some are targeted to the CDC configuration, and consequently require advanced and/or specific hardware, others have proposals requiring a huge development effort, requiring one implementation for each target platform.

The main advantage of this research is the use of DynamicML as a metalanguage for describing applications, which makes it possible to dynamically build the application. Thus, there is no need for extra library (not available on most devices) and even the development of a solution for each target platform.

The validation of the proposal was done by developing a prototype applied to three use cases in the fields of residential automation and industrial supervision. The prototype consists of a system for mobile phones using the framework, and a web architecture to interact with the supervised systems and simulate the operation of the set for demonstration.

Keywords: Supevisory systems. Dynamical systems. Mobile computing.

1 INTRODUÇÃO

Nos últimos anos houve um grande aumento tanto na qualidade quanto na quantidade de redes sem fio disponíveis, as quais possibilitam desde a simples navegação na web, passando por sistemas de localização, até sistemas supervisórios na área de automação.

Também os protocolos de comunicação aplicados à área de automação sofreram muitas mudanças nos últimos anos passando da tradicional rede analógica (4 a 20mA) para os diversos protocolos digitais atuais. Isto possibilitou aplicações mais sofisticadas que antes eram proibitivas devido ao forte impacto no desempenho da rede de comunicação.

Como descrito por (PINTO, 2007), a própria área de automação apresentou um grande crescimento nos últimos anos. E (BOWEN, BUENNEMEYER e THOMAS, 2005) mostra como essa evolução tornou os sistemas supervisórios mais flexíveis e interconectados, além de apresentar vários desafios que surgem a partir dessa grande integração dos sistemas.

Com esses eventos, surgiram então vários trabalhos explorando o uso de redes sem fio na área de automação; o maior número deles colocando dispositivos móveis como ferramenta adicional ao sistema de automação, expondo seus prós e contras e propondo soluções para as atuais limitações.

O uso de redes sem fio na automação proporciona uma economia no custo de cabeamento e, além disso, permitiu a disponibilidade de dados de forma ubíqua, ou seja, virtualmente independente da localização do usuário. As próprias máquinas num sistema de automação passaram a ter mais flexibilidade de interação entre si.

Contudo, essa mudança também trouxe desafios, principalmente na interseção dessas duas áreas de pesquisa (redes sem fio e automação). Problemas que antes não existiam, ou eram menores, surgiram ou foram potencializados. Alguns destes novos desafios dizem respeito a: segurança no tráfego de dados e acesso, IHM (Interface Homem-Máquina), autonomia de bateria e alcance da rede.

Existem vários desafios relacionados à incorporação de redes sem fio a sistemas de automação. (WILLIG, MATHEUS e WOLISZ, 2005) em seu trabalho, abordam as questões relacionadas a confiabilidade e tempo real. Além destes, outro desafio importante refere-se a segurança, pois redes sem fio são naturalmente “inseguras”, uma vez que os dados trafegam no ar, estando ao alcance de qualquer um. Por sua vez, sistemas de automação, pela própria tarefa a que se propõem, têm a segurança como um requisito de alta importância.

Outra questão que deve ser pensada diz respeito à IHM (Interface Homem-Máquina). Nesse sentido, (YORK e PENDHARKAR, 2004) trazem algumas questões a

serem consideradas, que são: primeiro a da dimensão do aparelho, que é mínima com o intuito de facilitar a portabilidade, mas com isso acaba restringindo os modos de interação; outra questão é a da autonomia, uma vez que, por ser portátil, o dispositivo deve ter uma fonte de energia que o mantenha em funcionamento por longo tempo.

1.1 Sistemas de Automação

A ideia por traz da automação é a unificação do controle e de processos para facilitar a gerência de algum sistema ou máquina, disponibilizando dados para que o usuário possa acompanhar cada evento e assim poder adequar o sistema de forma a atender os requisitos dos usuários (PEROZZO e PEREIRA, 2006).

Inicialmente, pela complexidade e custo, a automação ficou restrita ao ambiente industrial; mas com o desenvolvimento do *hardware* e, consequente barateamento, a automação se espalhou para uma infinidade de aplicações, atingindo até as residências modernas, onde proporciona controle automatizado de várias tarefas da casa: como segurança, climatização, iluminação, etc.

Com a difusão da automação surgiram novas áreas de aplicação para essa tecnologia, como exemplo temos ambientes inteligentes e ambientes de suporte a ensino a distância. A primeira é baseada na visão de Weiser (1991), em que ele introduz o conceito de computação ubíqua; no qual os recursos de computação seriam onipresentes com o fim de oferecer informações e serviços de forma transparente a seus clientes (que podem ser seres humanos ou outros sistemas). A segunda é o uso de ambientes inteligentes para o auxílio ao ensino, no qual as informações e serviços se tornam disponíveis mesmo com o aluno fora da sala de aula e, além disso, permitem uma interação maior com o conteúdo.

Existem ainda outras áreas, como a computação sensível ao contexto; onde contexto é qualquer informação que pode ser usada para caracterizar a situação de uma entidade (pessoa, lugar, objeto) que é considerada relevante entre o usuário e a aplicação (ABOWD, DEY, *et al.*, 1999).

A própria área inicial da automação, a industrial, está se beneficiando com esses avanços da computação pervasiva; hoje já existem várias aplicações e muita pesquisa no uso desse paradigma para o chão de fábrica, principalmente em sistemas supervisórios (Sistemas de Supervisão e Aquisição de Dados – SCADA, do termo em inglês).

1.2 Computação Móvel

A computação móvel vem surgindo como uma nova proposta de paradigma computacional advinda da tecnologia de rede sem fio e dos sistemas distribuídos (PALAZZO M. DE OLIVEIRA, MACHADO, *et al.*, 2006).

No decorrer do tempo, a evolução dos sistemas de computação diminuiu o forte acoplamento do usuário aos recursos e ao ambiente computacional. A grande expectativa nesse assunto está na possibilidade de acessar os recursos computacionais a qualquer tempo e em qualquer lugar (deste que exista rede de comunicação) (PALAZZO M. DE OLIVEIRA, MACHADO, *et al.*, 2006).

O termo computação ubíqua (Ubiquitous Computing) é amplo e diz respeito à imersão em um ambiente onde existem agentes computacionais para interação; porém, existem algumas definições mais específicas, dependendo do(s) elemento(s) que

pode(m) se mover no contexto. Assim, existem termos que dão ênfase a diferentes aspectos:

- *Hardware* pode se mover (*nomadic computing*) (KLEINROCK, 1995);
- Usuário pode se mover entre um conjunto fixo de estações conectadas à rede (*wireless computing*) (KLEINROCK, 1995);
- Aplicação pode se mover (*mobile computation: mobile code / mobile agent*) (Mobile Agent, 2010);
- Usuário, com um equipamento portátil (*hardware*), executando aplicações com dados e código móvel, se locomove (*pervasive computing*). Esta definição é basicamente a usada por Weiser (1991) acrescida com algumas definições de Kleinrock (1995), que explicita o agente móvel do contexto.

Apesar da distinção de acordo com o contexto (como detalhado acima) existe certa confusão entre os conceitos usados como é possível ver na definição de *nomadic computing* em (YOURDICTIONARY.COM, 2010) e *wireless computing* em (BITPIPE.COM, 2010). Dessa forma, é correto dizer simplesmente computação ubíqua ou pervasiva (WEISER, 1991) apesar da, talvez excessiva, generalização.

Esta é uma área multidisciplinar, uma vez que envolve a interação de vários campos de pesquisa como: computação distribuída, computação móvel, redes de sensores, interação homem-máquina, inteligência artificial, além de outras áreas, dependendo do contexto do sistema (Ubiquitous Computing) e (WEISER, 1993).

1.3 Computação Móvel e Automação

A computação pervasiva vem se tornando um novo paradigma no desenvolvimento de sistemas computacionais, uma vez que estes deixam de ser "fisicamente estáticos", ou seja, é possível se beneficiar de seus serviços e recursos em qualquer local do mundo (onde exista um mínimo de infraestrutura de comunicação).

Essa mudança está, aos poucos, criando e ampliando uma série de serviços como a TV digital (acessível em telefones 3G), possibilidade de comunicação e interação com outras pessoas através de chamadas telefônicas com vídeos, serviços sensíveis a contexto (publicidade sob demanda, vitrines interativas, etc.), e disponibilização de vários sistemas de automatização de forma remota (AMOR, 2001).

A automação está abraçando esse novo paradigma e suas mais diversas áreas, a automação residencial e de escritórios vem ampliando seus serviços com o usuário que agora pode ter acesso ao sistema de forma remota em diversos dispositivos móveis; a automação industrial, motivada principalmente pela possibilidade de enorme economia nos custos de cabeamento, vem adotando a tecnologia sem fio em seus sistemas, e ainda, criando uma série de novos serviços (sistemas de manutenção inteligentes, sistemas SCADA, etc.) (AMOR, 2001).

As tecnologias sem fio atuais ainda precisam atingir alguns requisitos de tempo real, que são importantes na área de automação e, principalmente, em sistemas críticos, que não podem falhar. Nesse sentido existem várias pesquisas sobre problemas e propostas de soluções (WILLIG, MATHEUS e WOLISZ, 2005). Felizmente, onde estas restrições (de tempo real) têm menor impacto, como em sistemas de automação residencial e de equipes de venda (do inglês *sales force*), a tecnologia sem fio já é uma realidade.

2 APLICAÇÕES DINÂMICAS

Um ambiente inteligente pode ser definido como um local rico em interação do ponto de vista de um usuário, tendo todo o tipo de facilidades para que o ambiente se adapte ao seu gosto (DUCATEL, BOGDANOWICZ, *et al.*, 2001). Mas para isso acontecer o usuário precisa antes ter em suas mãos algum dispositivo para controlar o ambiente; e como fazer isso para novos ambientes ou novos serviços? Exigir um complicado processo de instalação/configuração é ir na contramão do que significa um ambiente inteligente. Então o que fazer? O ideal é que o próprio ambiente indique para o usuário que nele existem serviços inteligentes disponíveis.

Para o usuário interagir com o serviço inteligente disponível, após o segundo indicar sua existência, é necessário que o primeiro tenha um dispositivo capaz de se comunicar com o ambiente e interagir com este, e qual a maneira mais simples e transparente? Como visto em (MARQUES, 2010) o número de celulares não para de crescer no Brasil, então é possível admitir que o aparelho celular torna-se um excelente candidato a plataforma de controle para ambientes inteligentes devido a sua quase onipresença junto aos usuários, além de a maioria destes serem dotados de plataformas de desenvolvimento de aplicações.

Mas um ambiente inteligente pode mudar, evoluir, adicionar novos serviços, e tudo isso deve ser refletido imediatamente no dispositivo do usuário. Então se torna necessário que a as aplicações possam mudar e evoluir de acordo com os serviços que representam, e essas mudanças devem ocorrer de forma transparente para o usuário, ou seja, a aplicação deve ser dinâmica, podendo alterar-se durante a sua execução.

No contexto do trabalho, é chamado de aplicação dinâmica, o *software* capaz de agregar, retirar ou modificar recursos (interfaces de usuário, do inglês *User Interfaces* ou UIs; serviços; ou dados) em tempo de execução, ou seja, não é necessário recompilar e nem mesmo reiniciar a aplicação. Dessa forma, uma vez iniciado o aplicativo, e durante todo o tempo de uso, este deve ser capaz de adicionar novos recursos que forem encontrados e prontamente disponibilizar para que o usuário possa usá-los; também deve ser capaz de, tão logo determinado recurso deixe de existir, remover a possibilidade de utilização pelo usuário; e finalmente, ser capaz de modificar configurações sinalizadas pelos servidores dos serviços de forma rápida e transparente ao usuário (Dynamic Definition).

Diferentemente do anterior, um sistema estático não muda após a compilação, ou seja, depois de criado, o sistema irá sempre apresentar a mesma lista de serviços e recursos. Neste caso, não importa onde o sistema esteja, ou se existem recursos disponíveis no ambiente que não foram previstos pelo arquiteto da aplicação na etapa de projeto, este irá sempre conter somente os recursos incluídos previamente. Em outras palavras, o sistema é imutável.

A proposta deste trabalho é explorar as melhorias possíveis em sistemas embarcados em dispositivos móveis (celulares, PDAs, etc.) na área de sistemas supervisórios e ambientes inteligentes, onde o resultado deste trabalho deverá permitir que as aplicações sejam capazes de incorporarem novos recursos e evoluírem (atualizar parcialmente sem a necessidade de recompilar ou mesmo reiniciar). Exemplos de casos de uso para validação poderiam ser:

- Interface gráfica pode ser facilmente atualizada em tempo de execução;
- Um supervisório recebe novos componentes para supervisão à medida que o usuário se move pela fábrica;
- Após a validação de usuário, o sistema recebe somente os componentes disponíveis àquele cliente (maior segurança).

2.1 A Plataforma

Os dispositivos móveis, atualmente, apresentam uma enorme variedade de plataformas de *hardware*, como redes de sensores, celulares, *smartphones*, entre outros; uma vasta coleção de *software*: vários sistemas operacionais (Lista de Sistemas Operacionais Embarcados), incluindo os sistemas fechados como o NokiaOS (Nokia OS), e mais de uma dezena de máquinas virtuais (Lista de Máquinas Virtuais Java). Na maioria absoluta dessa enorme variedade de sistemas é possível executar programas Java ME; assim para ampliar o escopo do trabalho foi escolhido o ambiente Java de desenvolvimento (reconhecidamente multiplataforma) (MAREJKA, 2008).

A Tabela 1 mostra um comparativo entre as plataformas de desenvolvimento Java ME e .Net Compact (Microsoft); existem outras, mas na comparação da tabela dá para perceber a importância e abrangência Java, em especial a plataforma CLDC) (KOSTAKOS).

Tabela 1 - Comparativo entre algumas plataformas de desenvolvimento.

	.Net Compact Framework	J2ME CDC (Connected Device Configuration)	J2ME CLDC (Connected Limited Device Configuration)
Requerimentos de Hardware	Alto	Alto	Médio/Baixo
Custo	Alto	Alto	Médio/Baixo
Mercado Foco	Enterprise	Enterprise	Consumer and enterprise
Linguagens Suportadas	C#, VB.Net	Java	Java
Plataformas	Pocket PC, Windows CE	Maioria das plataformas móveis, exceto Palm OS	Todas as plataformas móveis
Compatibilidade do Byte code	Padrão .Net CLR	Padrão Java 2	Incompatível com J2SE ou CDC
Compatibilidade da API	Subconjunto do .Net	Subconjunto do J2SE mais alguns pacotes padrões opcionais	Compatibilidade parcial com CDC mais alguns pacotes padrões opcionais
APIs Nativas	P/Invoke; consistente em todos os dispositivos suportados	JNI; específicas do SO ou do dispositivo	N/A
Ferramentas de Desenvolvimento	VS.Net 2003	Linha de comando, SDKs dos fabricantes de aparelhos, CodeWarrior, e WebSphere	Linha de comando, SDKs dos fabricantes de aparelhos, maioria das IDEs Java
Processo de Especificação	Única empresa	Comunidade	Comunidade
Gateway de Serviços	N/A	Executad gateways como OSGi servlets; executa clientes gateways via SDKs dos fabricantes de aparelhos	Executa clientes gateways via SDKs dos fabricantes de aparelhos
Modelo de Segurança	Modelo .Net simplificado	Gerenciamento de segurança Java completo	Modelo de gerenciamento de segurança Java limitado suplementado pela especificação OTA
Instalação de aplicativos	ActiveSync, Internet Explorer download	Sync, download	Especificação OTA formal
Gerenciamento do Ciclo de vida das aplicações	N/A	OSGi for gateway apps, J2EE Client Provisioning Specification for generic clients	Included in OTA spec, works with J2EE Client Provisioning Specification

Fonte: tradução feita a partir da tabela de (KOSTAKOS).

A plataforma Java se divide em quatro conjuntos de desenvolvimento: Enterprise Edition, Standard Edition, Micro Edition e Card (destacados na Figura 1); sendo a penúltima destinada à criação de sistemas embarcados em *hardware* mais simples (SUN MICROSYSTEMS) e (ORACLE DEVELOPER NETWORK, 2010). Como mostrado em (SUN MICROSYSTEMS) e (Java Platform, Micro Edition), o próprio Java ME se divide em vários conjuntos de acordo com as características específicas do *hardware* alvo; isto porque a arquitetura desta tecnologia foi dividida em configurações e perfis, onde a primeira é a especificação do ambiente de execução propriamente dito: a máquina virtual (neste caso conhecida por KVM, Kylobyte Virtual Machine) e um conjunto básico de classes; a segunda consiste em novas classes para prover acesso aos recursos específicos dos dispositivos (rede, interface, armazenamento de dados local, etc.).

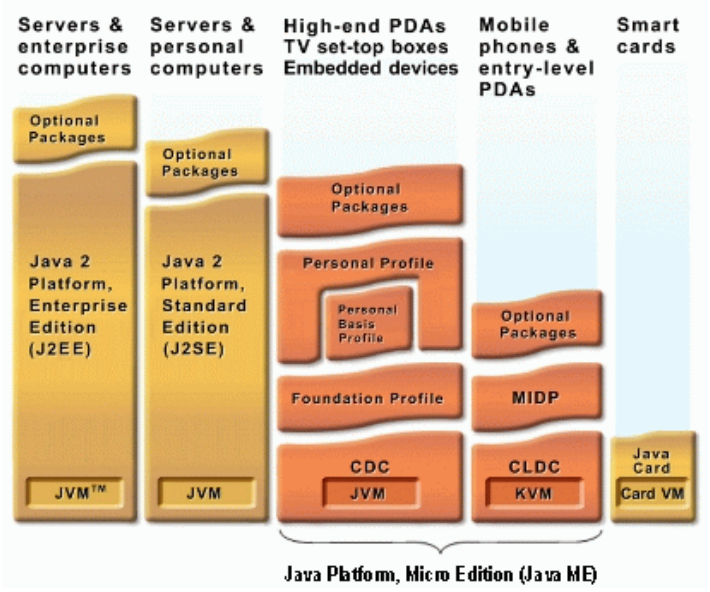

Figura 1 – Os componentes da plataforma Java ME (configurações e perfis) em comparação a outras tecnologias Java (SUN MICROSYSTEMS).

Ainda segundo em (SUN MICROSYSTEMS) e (Java Platform, Micro Edition), tem-se duas configurações, e dois e três perfis, respectivamente para cada configuração; como visto a seguir:

- CLDC (Connected Limited Device Configuration): subconjunto muito estrito de classes Java, o mínimo necessário para uma VM funcionar.
    - MIDP (Mobile Information Device Profile): projetado principalmente para celulares e semelhantes; provê uma GUI e acesso a rede, entre outros recursos.
    - IMP (Information Module Profile): específico para dispositivos sem tela de interface, ou esta muito simples; e conexão de rede mais limitada que na MIDP.
- CDC (Connected Device Configuration): subconjunto do Java SE; contém quase tudo que este (excluindo a parte de interface).
    - FP (Foundation Profile): destinada a dispositivos que possam implementar toda a especificação de uma JVM.
    - PBP (Personal Basis Profile): adiciona suporte a interfaces mais elaboradas com um pequeno subconjunto do AWT (Abstract Window Toolkit) ao perfil anterior.
    - PP (Personal Profile): extende o perfil anterior incluindo um maior subconjunto do AWT e adiciona suporte a *applets*.

Como visto na breve descrição da plataforma Java ME acima, para que este trabalho tenha um maior impacto e relevância a plataforma alvo ideal é a CLDC/MIDP, pois mesmo dispositivos bem simples poderiam se beneficiar. Outro fato que corrobora na escolha desta configuração/perfil é o número de dispositivos atualmente no mercado com estas características (sendo que mesmo os atuais lançamentos continuam a aumentar esta base no Brasil) (MAREJKA, 2008).

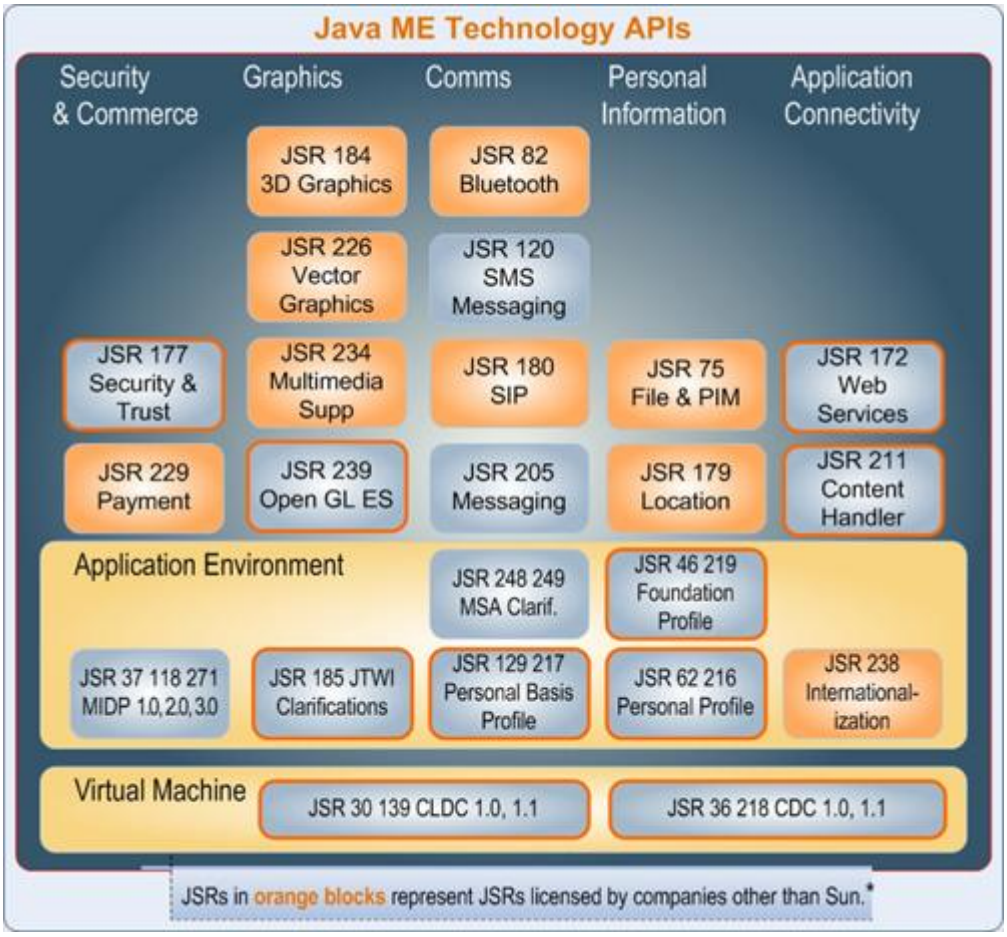

Figura 2 – Detalhe das JSRs da plataforma Java ME (ORACLE DEVELOPER NETWORK, 2010).

Desta forma optou-se pela plataforma Java Micro Edition para CLDC/MIDP; que é o conjunto mínimo (e mais restritivo na questão de recursos e APIs) do Java (para sistemas móveis com display de interface). A Figura 2 mostra a quantidade de APIs extras existentes para o Java ME, mas destas, poucas estão presentes na configuração CLDC. E apesar destas restrições, mesmo os atuais *smartphones*, com *hardware* mais poderoso, ainda dispõem de máquinas virtuais para Java ME CLDC/MIDP.

2.2 Desafios

Para obter um MIDlet (nome dado a um aplicativo Java na plataforma Java ME) dinâmico seria necessário poder carregar arquivos ".class" em tempo de execução. Contudo, esta não é uma opção; pois quando a plataforma Java ME foi projetada, o *hardware* tinha baixo desempenho e capacidade (processadores de baixo custo e pouquíssima memória), além de que na época, as restrições de segurança forçaram que a API do Java Me fosse extremamente enxuta como pode ser visto em (JCP, 2000) (JCP, 2007), desta forma não foi incluída a parte de reflexão Java (responsável por permitir carregamento dinâmico de código) (GIGUÈRE, 2002).

O *framework* OSGi (*Open Services Gateway Initiative*) que é uma plataforma de serviços Java, também foi portado para o Java ME através da JSR-232 (JCP, 2008). Porém, devido à forma como funciona, o *framework* exige a API de reflexão Java, que está disponível somente na configuração CDC do Java ME (JCP, 2005) (JCP, 2006).

Existem alguns projetos que tornaram possível este *framework* na CLDC, mas com grandes restrições como será mostrado mais a frente.

O RMI (execução de métodos remotos, do inglês *Remote Method Invocation*), especificado em (JCP, 2002) poderia ser usado para alcançar parcialmente o objetivo de se adicionar novos recursos dinamicamente a um MIDlet (parcialmente, pois o RMI realiza a execução remotamente, não incorporando código ao sistema local do dispositivo móvel). Contudo, este, também faz uso da API de reflexão, assim esta solução também só é viável em sistemas com configuração CDC como visto na especificação em (JCP, 2007).

Dessa forma, existem basicamente dois grandes desafios para alcançar o objetivo de se ter uma aplicação dinâmica em Java ME CLDC/MIDP:

- Falta de reflexão: ausência da API de reflexão Java na plataforma ME, e ausência de uma JSR (pacotes de APIs opcionais) para tal.
- Necessidade de pré-verificação: um código Java, para executar na plataforma ME, precisa ser pré-verificado pela ferramenta do SDK após já estar compilado. E quando este é instalado no aparelho, é feita apenas uma rápida verificação (graças à tarefa anterior) (J2ME Programming, 2010) e (DAY, 2000).

3 TRABALHOS CORRELATOS

Tendo-se especificada a plataforma alvo e seu funcionamento (como visto no capítulo anterior as limitações impostas e as tecnologias disponíveis que podem auxiliar na tarefa de obter sistemas dinâmicos em Java ME, na configuração CLDC) pode-se pensar no obstáculo e ver quais são as opções para conseguir um sistema dinâmico nas condições propostas. Inicialmente serão discutidas as técnicas usadas nos diversos trabalhos encontrados, e em seguida detalhar cada uma das pesquisas relacionadas.

As técnicas usadas para possibilitar a carga dinâmica de componentes foram as quatro listadas na sequência:

- Modificação da KVM
- Uso da JSR-232
- OSGi encapsulado no MIDlet
- Uso de agente nativo e OSGi

Todas as técnicas citadas têm como objetivo criar MIDlets dinâmicos, com alguns prós e contras alcançam a meta, mas de forma limitada (como mostrado no item seguinte onde as propostas são analisadas a fundo). Abaixo é mostrada a ideia básica de cada uma das abordagens.

A primeira técnica propõe a abordagem mais direta: modificar a KVM definida pela JSR-30 ou JSR-139 e adicionar as classes ausentes específicas sobre reflexão e assim permitir nativamente carga dinâmica de código. Essa técnica foi abordada nos seguintes trabalhos: (PETREA e GRIGORAS, 2006), (JAMVM GROUP, 2010) e (/K/ EMBEDDED JAVA SOLUTIONS, 2010).

A segunda usa o pacote opcional Mobile Operational Management (OSGi para Java, definido na JSR-232), mas na configuração CDC. Assim também é possível carregar novos conjuntos de código em tempo de execução. Permite ainda atualização de componentes da aplicação (os módulos, no contexto do OSGi). Essa técnica foi usada nos seguintes trabalhos: (BEERS, 2007), (CONCIERGE GROUP, 2009), (RELLERMEYER e ALONSO, 2007) e (HAIGES, 2004).

A terceira já aborda o uso do OSGi na configuração CLDC do Java Me encapsulando o *framework* dentro do MIDlet, assim como todos os módulos necessários; desta forma a aplicação pode carregar dinamicamente os componentes encapsulados com ela, mas não pode fazer o mesmo com código fora do JAR (Java ARchive) do MIDlet. Trabalhos que desenvolvidos com essa técnica: (KRIENS, 2006), (JADABS GROUP, 2004), (FREI e ALONSO, 2004) e (FREI e ALONSO, 2005).

A quarta usa a mesma técnica que a anterior, mas adicionando um novo recurso: o agente nativo, para que seja possível carregar novos componentes que não estavam

anteriormente encapsulados com a aplicação (KRIENS, 2006), (PROSYST, 2010) e (PROSYST, 2006).

3.1 Análise das Soluções

No item anterior, deste mesmo capítulo, foram apenas relacionadas as ideias principais de cada uma das abordagens para permitir a carga dinâmica de componentes de *software* em tempo de execução, mas sem aprofundar nos detalhes e nem avaliar seus benefícios e limitações.

Neste item cada proposta será descrita e analisada em detalhes no seu funcionamento, também serão listados alguns trabalhos encontrados de cada uma. A análise foi baseada, em maior parte, mas não somente, na documentação dos próprios projetos; em alguns casos foi possível encontrar casos de uso e avaliações de terceiros.

A avaliação que se segue não tem como objetivo medir o desempenho ou classificar a qualidade dos projetos; a única intenção é avaliar os casos e em quais situações eles cobrem e resolvem o problema (sistemas dinâmicos para Java ME CLDC) e quais são suas limitações.

3.1.1 Modificação da KVM

A especificação da CLDC, descrita em (JCP, 2000) e (JCP, 2007), que define a KVM básica capaz de executar em dispositivos com recursos computacionais extremamente escassos, omitiu (exatamente por questões de segurança e performance) a API de reflexão Java; o que passou a impedir a carga dinâmica de componentes de *software* Java em tempo de execução (GIGUÈRE, 2002).

Assim, a primeira e "mais simples" ideia seria redefinir e reimplementar a KVM, tornando possível o carregamento de novas classes Java; para tal é necessário criar um novo *class loader* (PETREA e GRIGORAS, 2006).

De acordo com a proposta de (PETREA e GRIGORAS, 2006), é possível solucionar o problema e assim obter MIDlets dinâmicos; contudo a complexidade de se implementar a KVM não é trivial e além disso seria necessário fazer isso para cada plataforma alvo, pois o MIDlet Java é multiplataforma, mas a KVM é específica para cada *hardware* e/ou sistema operacional.

Ainda segundo (PETREA e GRIGORAS, 2006), ainda falta criar um protótipo da proposta. Já em (JAMVM GROUP, 2010) e (/K/ EMBEDDED JAVA SOLUTIONS, 2010) existe uma implementação funcional, mas para plataformas específicas; e como mencionado anteriormente o custo e a complexidade da tarefa de implementação desta solução a torna menos interessante aos desenvolvedores.

Existem vários projetos de implementação alternativos de VM's, mas a maioria tem como alvo a plataforma Java SE e não a ME (plataforma alvo deste trabalho). Para a plataforma móvel Java existem alguns projetos, dentre eles destacam-se: MikaVM em (/K/ EMBEDDED JAVA SOLUTIONS, 2010), e JamVM em (JAMVM GROUP, 2010). MikaVM tem como alvo a configuração CDC (principal motivação é o uso com OSGi), mas novamente não atende à plataforma alvo deste trabalho. A JamVM apesar de implementar uma JVM (especificação completa Java), diz ser compacta o suficiente para rodar em dispositivos projetados para CLDC que usam KVM (os testes no site do projeto foram realizados com aparelhos com *hardware* de maior capacidade); porém por ser uma implementação da especificação completa, este projeto inclui suporte completo

às APIs de reflexão e JNI, que são os impeditivos de sistemas dinâmicos em JAVA ME CLDC.

Portanto, se o projeto JamVM se confirmar estável e flexível o suficiente para executar em sistemas CLDC com o *hardware* mínimo da plataforma, seria uma boa alternativa para sistemas abertos; pois sistemas fechados como o NokiaOS, Symbian (versões *low-end*) e outros, a KVM é "fixa", não podendo ser substituída.

3.1.2 Uso da JSR-232

A JSR-232 (JCP, 2008) define uma API opcional para dar suporte ao OSGi, que é uma especificação de uma plataforma de serviços, em (OSGI ALLIANCE, 2010). O OSGi é um *framework* Java para sistemas que requerem longos tempos em execução, atualizações dinâmicas, e mínima interferência ao ambiente de execução. Segundo (BEERS, 2007), inicialmente o OSGi foi aplicado aos sistemas de automação e *gateways* residenciais. Recentemente, o *framework* ganhou aplicações em novas áreas, passando a ser usado desde telefones móveis e até em carros. A Figura 3 é mostra a plataforma OSGi e como esta se relaciona com a pilha do Java ME.

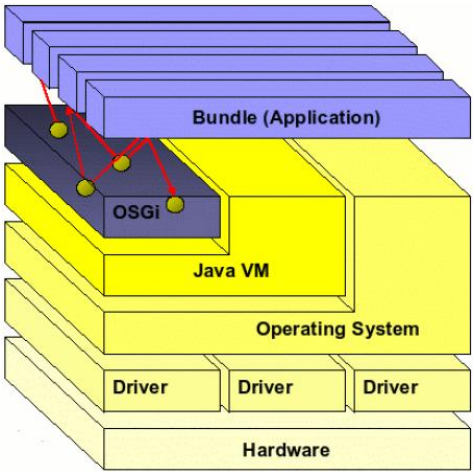

Figura 3 – Plataforma OSGi.

A Figura 4 mostra uma comparação entre as configurações CLDC e CDC com a JSR-232 do Java ME, é destacada a vantagem OSGi do uso de módulos independentes como serviços e como é possível ainda manter a compatibilidade com sistemas CLDC.

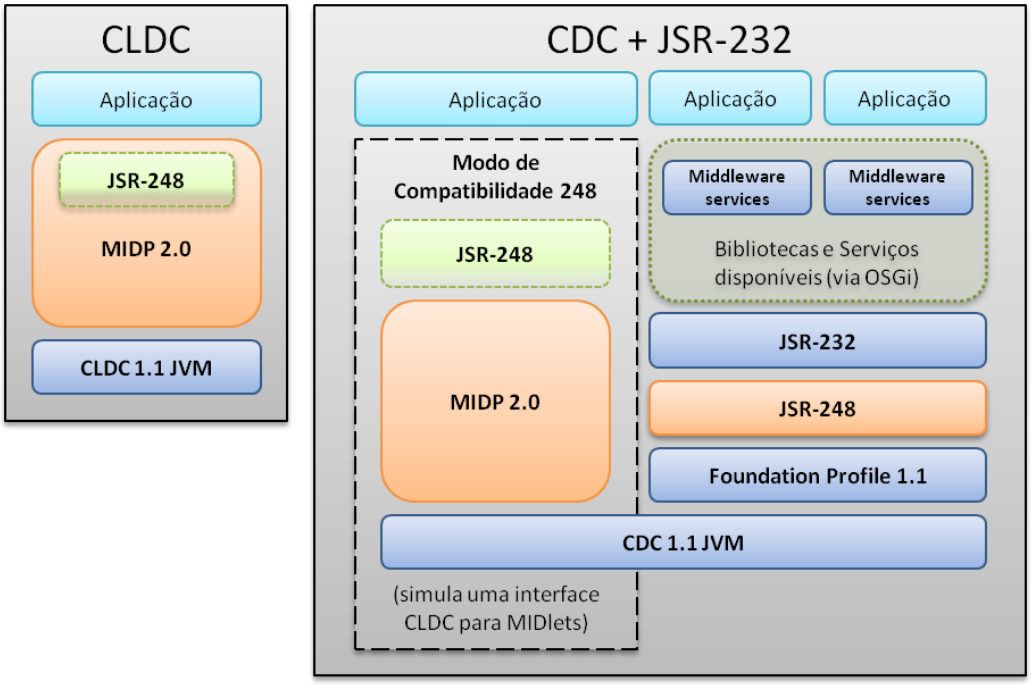

Figura 4 – Comparativo entre as configurações CLDC e CDC com suporte a OSGi do ponto de vista de uma aplicação.

Infelizmente, para atender, o conjunto mínimo da especificação OSGi, a CLDC é inviável devido ao problema da ausência da API de reflexão Java. Assim, com esta JSR é possível implementar sistemas dinâmicos somente nos perfis da configuração CDC do Java ME.

Existem várias implementações usando esta abordagem, dentre as que têm código aberto, citam-se: (APACHE FELIX, 2010), (ECLIPSE EQUINOX, 2010), (KNOPFLERFISH GROUP, 2010) e (CONCIERGE GROUP, 2009).

Com isto, foi possível verificar que a solução é válida, mas novamente não atende à plataforma alvo deste trabalho, e continua a lacuna para a configuração CLDC.

3.1.3 OSGi encapsulado no MIDlet

Para resolver o problema do OSGi com o CLDC, (FREI e ALONSO, 2004) e (FREI e ALONSO, 2005) usaram a ideia de empacotar todo o *framework* OSGi e arquivos requeridos num único arquivo Jar, e incluir este no container do MIDlet. Assim é possível carregar os serviços incluídos, de forma semelhante ao que ocorre com módulos do Kernel Linux.

Usando a ideia de encapsular todo o *framework* OSGi foi desenvolvido o projeto Jadabs, que é inclusive de código aberto (JADABS GROUP, 2004).

Como é possível ver na Figura 5, quando se usa OSGi é possível carregar novos módulos em qualquer instante (pois o container do *framework* é flexível) e assim extender a aplicação; contudo, como mostrado na mesma Figura, o conteúdo de um MIDlet não pode ser alterado depois que este é instalado (devido à simplicidade da CLDC a verificação é muito limitada; assim deve ser feita, antes, uma pré-verificação fora do dispositivo móvel) (JADABS GROUP, 2004).

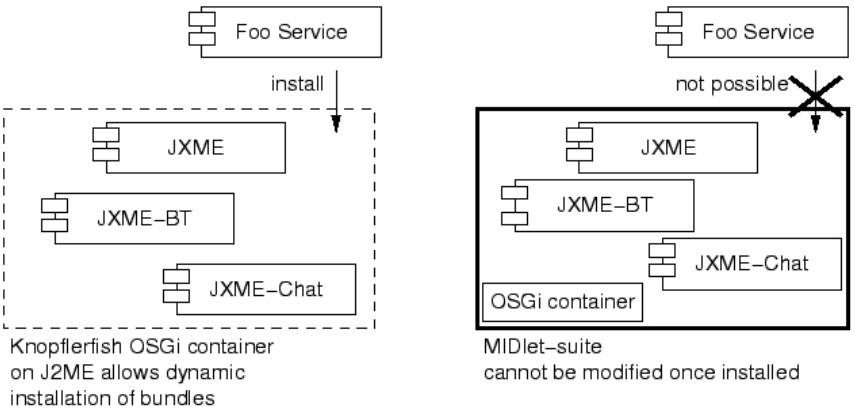

Figura 5 – Comparação dos contêineres OSGi e MIDlet (JADABS GROUP, 2004).

A seguir, é mostrado como realmente funciona a ideia usada no Jadabs. Todos os módulos requeridos devem ser encapsulados juntos com o contêiner OSGi num único arquivo Jar, e depois processado pelo pré-verificador.

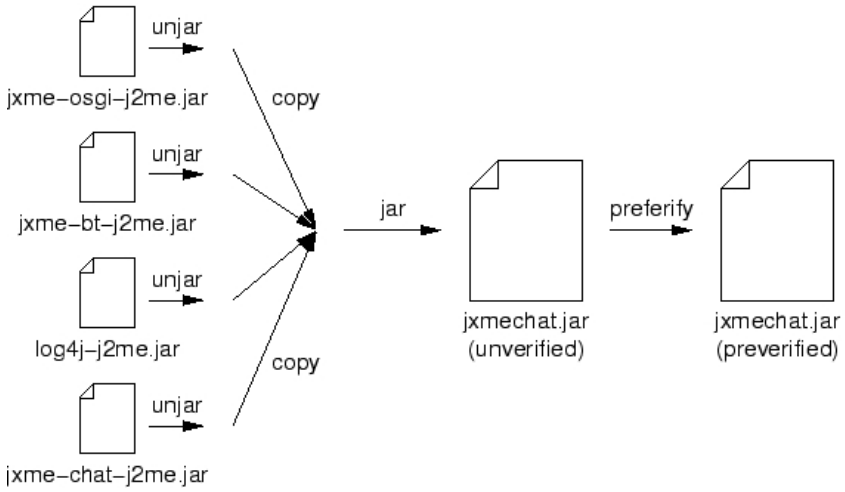

Figura 6 – Técnica do Jadabs para usar OSGi com Java ME CLDC (JADABS GROUP, 2004).

Esta abordagem permite o uso do OSGi em qualquer KVM CLDC, mas somente módulos que estejam empacotados no container do MIDlet serão possíveis de ser usados via OSGi em tempo de execução na aplicação (JADABS GROUP, 2004).

Com esta solução obtém-se parcialmente o dinamismo de sistemas embarcados Java na configuração CLDC. Parcialmente porque continua sendo impossível adicionar novos módulos após o MIDlet ser instalado no dispositivo (devido à forma como funciona a KVM: verificação/pré-verificação).

3.1.4 Uso de agente nativo e OSGi

Para alcançar o objetivo (PROSYST, 2010) usou uma técnica bem criativa, e complementar à usada pelo Jadabs, discutida no item anterior. Nesta nova proposta foi introduzido um novo componente de *software* como parte da solução, e este, chamado de “agente nativo”: um componente extra aos MIDlets no dispositivo móvel (é um *software* externo à plataforma Java). Esse sistema é criado especificamente para a

plataforma alvo (daí o “nativo” do nome), sendo desenvolvido na linguagem nativa do dispositivo.

O agente nativo é usado para executar operações específicas (que não podem ser feitas pelo *framework* CLDC devido às limitações da KVM CLDC). Sendo usado principalmente para as operações de ciclo de vida dos módulos e do *framework* CLDC (instalação, atualização e desinstalação) (PROSYST, 2006).

Ainda segundo (PROSYST, 2006), o agente nativo faz interface com toda a pilha da plataforma Java e com o sistema operacional, como mostrado na Figura 7. Essa abordagem torna possível a carga dinâmica de módulos, mas novamente, de forma parcial. Pois todos os módulos devem estar previamente empacotados no MIDlet do *framework* OSGi.

Essa estratégia torna possível instalar e usar novos MIDlets à medida que são necessários, e estes podem instalar, executar, desinstalar qualquer módulo presente no contêiner OSGi; mas cada MIDlet individualmente é imutável, não podendo ser alterado ou extendido em tempo de execução (dinamicamente), para tal seria necessário instalar uma nova versão deste MIDlet com as modificações (PROSYST, 2010).

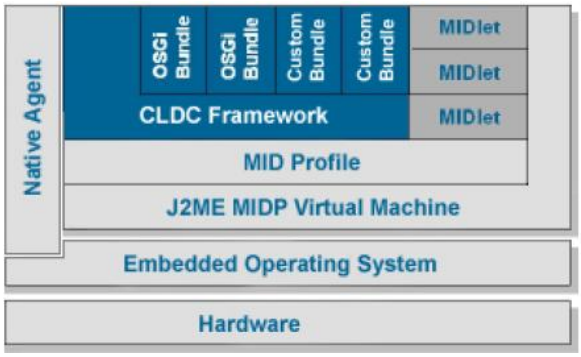

Figura 7 – Plataforma da solução OSGi da ProSyst (PROSYST, 2006).

Até agora foi visto que o OSGi era, inicialmente, incompatível com a plataforma Java ME na configuração CLDC; para vencer esta incompatibilidade (como visto em detalhes no item anterior) o *framework* OSGi foi incorporado ao contêiner do MIDlet, e aqui foi usado o conceito de agente nativo para o mesmo fim. Porém agora se tem um novo desafio: somente os módulos previamente empacotados no contêiner podem ser usados pela aplicação. Chegou-se mais perto de uma solução definitiva, mas ainda não satisfaz o proposto: incorporar componentes dinamicamente em tempo de execução (e que a princípio não estavam previstos).

Partindo da ideia do agente nativo, a ProSyst pretende prover a capacidade de incorporar novos módulos (sem previsão de lançamento, pois segundo informação no site da empresa, este recurso não é prioridade no momento). Atualmente em fase de desenvolvimento já é possível adicionar novos módulos ao MIDlet OSGi, desde que estes já estejam no sistema de arquivos do dispositivo móvel. Esta tarefa também é realizada pelo agente nativo que irá criar um novo contêiner MIDlet, adicionando o novo módulo. Neste processo, a aplicação (se em execução) será encerrada, o agente nativo irá adicionar o novo módulo ao contêiner do MIDlet e iniciá-lo novamente (PROSYST, 2010).

Esta solução, quando pronta, pode também atingir o objetivo proposto neste trabalho; contudo não é muito agradável ao usuário ter sua aplicação encerrada toda

hora que tiver um novo serviço disponível. Principalmente no cenário proposto neste trabalho (ambientes inteligentes) perturbações durante o uso do sistema não são desejáveis, pois pode se tratar de um sistema crítico, além de prejudicar a interação do usuário com o *software*. Usando um sistema supervisório como exemplo, seria muito ruim entrar num edifício ou indústria e a cada andar ou setor, respectivamente, a aplicação ficar encerrando e reiniciando para adicionar novos recursos ou serviços disponíveis em cada novo local visitado.

4 PROPOSTA: O *FRAMEWORK* DYNAMICML

Como visto no capítulo anterior, ainda não é possível criar uma solução satisfatória (na definição de sistema dinâmico, capítulo 2) para ser usada eficientemente no contexto proposto por este trabalho: sistemas supervisórios e ambientes inteligentes, na plataforma Java ME, na configuração CLDC.

Os benefícios e limitações de vários trabalhos foram avaliados no capítulo anterior, mas em resumo temos o seguinte:

- Modificação da KVM: grande complexidade, e requer o desenvolvimento para cada dispositivo alvo diferente.
- JSR-232: não serve para dispositivos CLDC, ou seja, sistemas com grandes restrições de hardware (e a maioria absoluta dos dispositivos atuais).
- OSGi encapsulado: o dinamismo é parcial, pois qualquer módulo tem de estar previamente disponível no contêiner do MIDlet.
- Agente nativo: dinamismo parcial ou interação homem-máquina prejudicada (como explicado no último item do capítulo anterior).

Todos os trabalhos encontrados são unânimes quanto às dificuldades impostas pela CLDC (principalmente pela falta da API de reflexão) para se obter aplicações dinâmicas; e como visto anteriormente, todas as soluções propostas tem suas restrições.

Assim, a ideia de propor um sistema que possa satisfazer as características de um sistema dinâmico (definido no capítulo 2) no contexto de sistemas supervisórios e ambientes inteligentes é um grande desafio.

O resultado deste trabalho pode ter um impacto positivo na qualidade e difusão destes sistemas em dispositivos móveis, proporcionando novas aplicações ou expandido a base de aparelhos compatíveis para sistemas já existentes.

4.1 Contribuição

O objetivo geral dessa Dissertação de Mestrado é propor um *framework* para possibilitar a criação de sistemas dinâmicos móveis, na plataforma Java ME, configuração CLDC, e principalmente na área de sistemas supervisórios.

A principal contribuição deste trabalho em relação às pesquisas correlatas atualmente encontradas é a plena possibilidade de criar aplicações dinâmicas (como definido no capítulo 2) no contexto proposto.

Outros benefícios alcançados foram: (i) o desenvolvimento de uma metalinguagem para a especificação de aplicações, e (ii) um conjunto de ferramentas para rápida prototipação de *software*.

Em relação à metalinguagem, por criar uma forma de mais alto nível para "programar" uma aplicação, usando uma linguagem em formato XML e descritiva, torna possível que um maior número de pessoas, e não somente especialistas, possam entender e desenvolver aplicações Java ME usando o resultado deste trabalho.

A respeito do conjunto de ferramentas para rápida prototipação: tem-se o *framework* em si, e a linguagem DynamicML, que auxiliam no projeto e rápido desenvolvimento de protótipos para validação de sistemas; ou seja, é gerada inicialmente somente a interface para avaliação e validação da mesma junto ao cliente, e isso sem tomar tempo da equipe de desenvolvimento.

4.2 Arquitetura

Dados os desafios impostos pela configuração CLDC do Java ME, e depois de avaliar os trabalhos relacionados e suas contribuições surgiu a ideia de usar a própria filosofia Java (interpretar código) para resolver o problema proposto. Entenda-se aqui por interpretar código como a técnica de "escrever" uma aplicação através de uma metalinguagem, no formato XML, e a partir daqui chamada de DynamicML. E o *framework* proposto será usado para ler e interpretar um arquivo escrito nesta linguagem, e assim criar efetivamente a aplicação no dispositivo móvel. Toda a representação gráfica (interface com usuário) e comunicação de dados são gerenciadas pelo motor do *framework*.

Assim, o *framework* deve permitir aos desenvolvedores de sistemas para dispositivos móveis criar MIDlets dinâmicos, e sem depender de implementações de JSRs pelos dispositivos, de *hardware* sofisticado e nem mesmo de uma KVM específica.

O MIDlet deverá usar a biblioteca do *framework* através de uma API definida; todas as configurações dinâmicas serão definidas através de um arquivo XML que pode estar previamente contido no dispositivo ou pode ser recebido via qualquer interface de comunicação que suporte HTTP, TCP ou UDP (Wifi, Bluetooth, GPRS, etc.).

No cenário de ambientes inteligentes, um exemplo é a descoberta de novos serviços disponibilizados por algum equipamento no ambiente; assim ao descobrir a existência do novo serviço o MIDlet buscará o arquivo de configuração, este será processado pelo motor do *framework* e após isto o MIDlet se adaptará de acordo com o conteúdo do arquivo de configuração.

Neste momento, é importante salientar que a proposta não aborda o problema de descoberta de serviços em si; desta forma o trabalho está sendo feito assumindo que de alguma forma (seja usando alguma outra ferramenta ou software) o novo serviço é encontrado e informado ao gerenciador do *framework* DynamicML. O objetivo deste trabalho é propor uma ferramenta capaz de prover a capacidade de adaptação dinâmica de um MIDlet sem interromper a sua execução.

A solução proposta funciona da seguinte forma: o usuário entra em um ambiente com seu aparelho móvel; se neste local houver um serviço disponível (controle de iluminação, por exemplo), o servidor mestre do ambiente deverá informar o dispositivo móvel da disponibilidade deste serviço. Ao ser informado da presença do novo serviço, o *framework* pode avisar o usuário e pedir autorização ou requisitar seu arquivo de configuração automaticamente. A configuração do serviço, um arquivo DynamicML, é então enviada ao dispositivo móvel, como visto na Figura 8. Em seguida este arquivo é

processado e interpretado pelo motor do *framework*, e após este processo o sistema criar o MIDlet, ou adaptar caso já exista, de forma dinâmica, de acordo com o arquivo recebido.

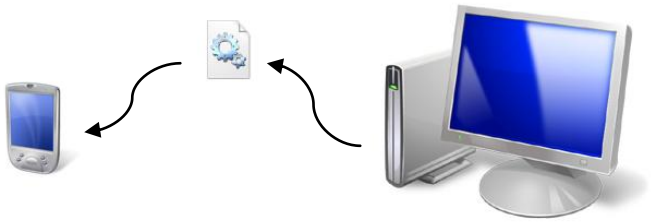

Figura 8 – Descoberta de novo serviço e recepção do arquivo de configuração pelo MIDlet.

Cada serviço tem um arquivo de configuração único, e este é criado usando a metalinguagem DynamicML; assim criamos a descrição de uma aplicação genérica e independente do dispositivo alvo. Toda e qualquer peculiaridade do aparelho móvel será adaptada no momento em que o motor do *framework* interpretar o arquivo de configuração para gerar o código da aplicação para execução. Isso permite adaptar o visual e a usabilidade da melhor forma possível para cada aparelho automaticamente. Esta é uma importante contribuição deste trabalho, como mostrado na Figura 9, um mesmo serviço pode ser acessado facilmente em dispositivos bem diferentes.

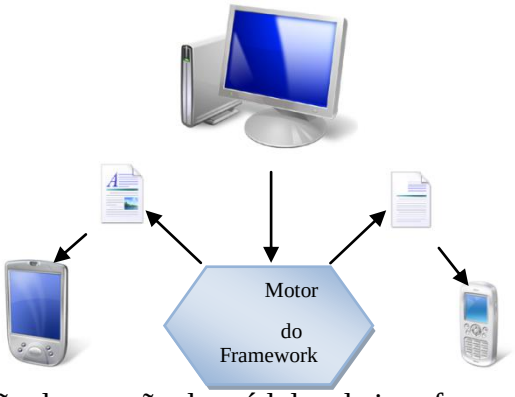

Figura 9 – Demonstração da geração de módulos de interface para múltiplos aparelhos móveis (de um mesmo serviço).

Uma opção é usar o sistema não para adicionar serviços ou novos recursos ao MIDlet, mas para somente criar interfaces de usuários atualizáveis; ou seja, quando houver apenas pequenas correções no aplicativo, não seria necessário atualizar todo o sistema. Bastaria modificar o arquivo de configuração e disparar um sinal para os dispositivos móveis carregarem o novo arquivo. A nova interface estaria instantaneamente pronta para uso, sem nenhuma interrupção no trabalho do usuário.

Com o sistema proposto será possível ter sistemas dinâmicos: que possam adicionar novos componentes, atualizar componentes, GUI padronizada e previsível (independente do dispositivo); tudo isto na plataforma Java ME e configuração CLDC.

4.3 A Linguagem DynamicML

Para programação de aplicações dinâmicas usando o *framework* proposto neste trabalho foi criada uma metalinguagem para descrição de aplicações: a DynamicML. No projeto da linguagem, foi feita a opção por usar como base o XML (que é um padrão

mais que consagrado para estruturação de dados hierárquicos), e aproveitando assim, a existência de inúmeras ferramentas para este formato.

Abaixo segue um exemplo de arquivo descrevendo uma aplicação na linguagem DynamicML. Um manual completo sobre a linguagem encontra-se no apêndice deste trabalho. Os detalhes do código serão explicados em seguida.

```
<?xml version="1.0" encoding="UTF-8"?>
<DynamicML>
    <Service Name="DynamicMJ" Version="100000">

        <Description>Descrição deste Serviço.</Description>

        <URL type="upload">http://skynet-s.no-ip.org:6969/DynamicMJ_Server/SendData</URL>
        <URL type="download">http://skynet-s.no-ip.org:6969/DynamicMJ_Server/GetData</URL>

        <UIList>
            <UI id="TESTE_GUI">
                <ComponentsList>
                    <Component id="STRING_ITEM" type="STRING_ITEM">
                        <Properties>
                            <Label>Título do StringItem</Label>
                            <Text>Texto, texto, texto.</Text>
                            <Font>FACE_PROPORTIONAL,STYLE_BOLD,STYLE_UNDERLINED,
                                SIZE_MEDIUM</Font>
                            <Layout>LAYOUT_DEFAULT</Layout>
                            <AppearanceMode>PLAIN</AppearanceMode>
                        </Properties>
                    </Component>

                    <Component id="TEXT_FIELD" type="TEXT_FIELD">
                        <Properties>
                            <Label>Título do TextField:</Label>
                            <Text>escreva aqui um texto...</Text>
                            <MaxSize>100</MaxSize>
                            <Constraints>ANY,UNEDITABLE</Constraints>
                            <Layout>LAYOUT_DEFAULT</Layout>
                        </Properties>
                    </Component>

                    <Component id="IMAGE_ITEM" type="IMAGE_ITEM">
                        <Properties>
                            <Label>ImageItem</Label>
                            <Image>http://skynet-s.no-ip.org:6969/DynamicML/mac.gif</Image>
                            <Layout>LAYOUT_DEFAULT</Layout>
                            <AlternateText>image NOT loaded!</AlternateText>
                            <AppearanceMode>PLAIN</AppearanceMode>
                        </Properties>
                    </Component>

                    <Component id="DATE_FIELD" type="DATE_FIELD">
                        <Properties>
                            <Label>Data atual:</Label>
                            <InputMode>DATE_TIME</InputMode>
                            <DateTime>11:12:13,14/10/2015</DateTime>
                            <Layout>LAYOUT_DEFAULT</Layout>
                        </Properties>
                    </Component>
```

```
                    <Component id="CHOICE_GROUP" type="CHOICE_GROUP">
                        <Properties>
                            <Label>Escolha!</Label>
                            <ChoiceType>EXCLUSIVE</ChoiceType>
                            <Items>UM; ;DOIS; ;TRES; </Items>
                            <Selected>0,1,0</Selected>
                            <Font>FACE_SYSTEM,STYLE_PLAIN,SIZE_SMALL</Font>
                            <FitPolicy>TEXT_WRAP_DEFAULT</FitPolicy>
                            <Layout>LAYOUT_DEFAULT</Layout>
                        </Properties>
                    </Component>
                </ComponentsList>

                <Buttons>
                    <Command>ACTION_EXIT</Command>
                    <Command>ACTION_SENDDATA</Command>
                    <Command>ACTION_RECEIVEDATA</Command>
                    <Command ui="UI2">ACTION_GOTOUI</Command>
                </Buttons>
            </UI>
            <UI id="UI2">
                <ComponentsList>
                    <Component id="STRING_ITEM" type="STRING_ITEM">
                        <Properties>
                            <Label>TÃ-tulo do StringItem</Label>
                            <Text>Texto, texto, texto.</Text>
                            <Font>FACE_PROPORTIONAL,STYLE_BOLD,STYLE_UNDERLINED,
                                    SIZE_MEDIUM</Font>
                            <Layout>LAYOUT_DEFAULT</Layout>
                        </Properties>
                    </Component>
                </ComponentsList>

                <Buttons>
                    <Command ui="TESTE_GUI">ACTION_GOTOUI</Command>
                </Buttons>
            </UI>
        </UIList>
    </Service>
</DynamicML>
```

Como visto na listagem exemplo, o arquivo descrevendo a aplicação inicia com a *tag* padrão XML, e logo após, uma *tag* informando que este é um documento no formato DynamicML. Em seguida vem o corpo que descreve realmente a aplicação que se quer gerar. Como principais *tags* para se entender a linguagem temos:

1. **Service**: identifica unicamente um serviço internamente no gerenciador de serviços (conta com o nome do serviço em formato ID e do seu número de versão).
2. **Description**: um breve texto para descrever os propósitos do serviço.
3. **URL**: indica os endereços para envio e recebimento de dados que serão acessados (a *tag* URL pode ser do tipo *upload* ou *download*).
4. **UIList**: *tag* que indica o início do bloco de UIs (interface de usuário, do inglês *user interface*).
5. **UI**: identifica uma UI unicamente dentro de um serviço.
6. **ComponentList**: *tag* para indicar o início do bloco de componentes de uma tela (UI), ou seja, a lista de itens de interface.
7. **Component**: identifica o componente, de forma única, dentro de uma UI (tela) e o tipo de componente (o tipo de objeto a ser instanciado será definido com base neste último atributo).

8. **Properties**: *tag* com número de *tags* internas variável, e indica o início do bloco de propriedades de configuração de um componente. A quantidade e a natureza destas *tags* internas variam de acordo com cada componente, alguns itens comuns de configuração são: fonte usada no componente, *label* ou título do componente, texto do componente, etc.
9. **Buttons**: indica o bloco de comandos para uma UI, ou seja, define os comandos para uma tela, como: ir para outra tela, sair do programa, etc.
10. **Command**: *tag* que define uma ação para uma dada UI. Atualmente existem somente quatro ações possíveis: sair do serviço, ir para outra tela, enviar dados da tela atual, receber dados e atualizar os campos da tela atual.

Para referência completa sobre a linguagem, consultar o manual da linguagem DynamicML.

4.4 DynamicML *framework*

O *framework* é composto pelos seguintes módulos principais: *Core*, *Net* e UI; e dois módulos menores: *Exceptions* e *Test*.

O módulo *Core* é o coração do sistema, e concentra todo o código de processamento dos arquivos XMLs, criação dos objetos e componentes da interface, gerenciamento das telas, processamento de comandos, e centralizando o acoplamento com os outros módulos.

O módulo *Net* centraliza todo o código de envio e recebimento de dados através de meios de comunicação (*Bluetooth*, internet, WiFi). É o responsável por preparar os dados antes de enviar ou após recebê-los via rede, realizando qualquer tratamento necessário (compactação, criptografia, etc.). Atualmente as interfaces de comunicação suportadas são redes TCP/IP via GPRS (rede de dados da telefonia móvel) ou WiFi; sendo que para o envio/recebimento de dados de um serviço e inclusive do arquivo XML do serviço, atualmente, só existe suporte através do protocolo HTTP (tipicamente *servlets*), já para o recebimento dos arquivos XMLs dos descritores e de atualizações automáticas dos serviços (dados e/ou interface) os dados são transmitidos via protocolo UDP.

O módulo *UI* foi divido em dois grupos de código: uma parte geral, com interfaces que devem ser implementadas por componentes a serem usados pelos serviços, e um módulo configurador das propriedades dos componentes (parte específica do componente na tradução XML); e a outra parte é o conjunto de componentes disponíveis atualmente para trabalhar com o *framework* proposto.

Já o módulo *exceptions* é para controle e tratamento de erros durante a execução, e o *test* é um conjunto de arquivos de teste de várias partes, usadas durante o desenvolvimento da ferramenta.

Na sequência será apresentada uma visão ampla da arquitetura do *framework*, detalhando a estrutura e os relacionamentos entre as classes; e em seguida cada módulo será apresentado individualmente para explicitar os detalhes internos de funcionando destes e de suas respectivas classes.

Para facilitar a visualização, o diagrama de classes foi dividido em duas etapas: na primeira serão ressaltados os relacionamentos de acordo com a arquitetura do *framework*, e por isso escondendo o conteúdo das classes (campos e métodos); na

segunda serão mostrados os diagramas exibindo o conteúdo das classes, mas sem os relacionamentos. Essa divisão foi feita para ter imagens mais claras e assim facilitar a compreensão do trabalho.

4.4.1 Visão Geral

O *framework* pode ser dividido em cinco grupos de relacionamentos:

- Gerenciador de serviços;
- Serviço;
- Interface;
- Tratamento de erros;
- Teste e pesquisa.

4.4.1.1 Gerenciador de Serviços

Este grupo de classes é o coração do *framework*, pois gerencia a instalação, atualização, desinstalação, os serviços de rede referentes à comunicação com o servidor do ambiente, além do motor que faz a análise do arquivo DynamicML para traduzi-lo em componentes do Java ME.

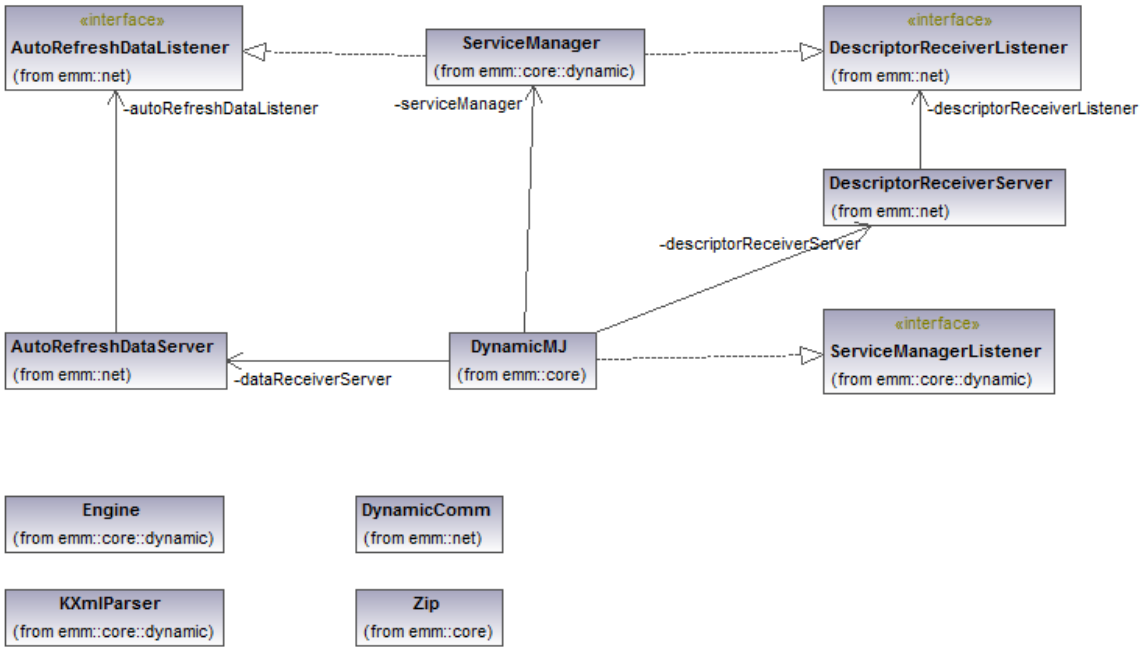

Figura 10 – Detalhamento dos relacionamentos de classes do Gerenciador de Serviços.

A interface do protótipo com o usuário é feita na classe DynamicMJ, que por sua vez contém um gerenciador de serviços, através da classe *ServiceManager*, e esta por sua vez implementa duas interfaces de servidores: *AutoRefreshDataListener* e *DescriptorReceiverListener*; a primeira é para o gerenciador ser alertado quando o *framework* receber dados para atualizar dados de algum serviço; a segunda é a interface que o gerenciador usa para receber um descritor de serviço (novo serviço descoberto ou um já existente tem uma nova versão de aplicação).

A classe *Engine* é uma *thread* usada para realizar a análise sintática de arquivos DynamicML, e a *KXmlParser* é usada internamente pelo *Engine* para *debugging*.

DynamicComm também é uma *thread*, e é usada para realizar a comunicação HTTP com os servidores mediadores dos serviços disponíveis no ambiente.

A classe *Zip*, é uma classe utilitária usada para compactação de dados, útil para economia de banda durante a transmissão de dados.

4.4.1.2 Serviço

Uma aplicação é representada pela classe *Service*, e esta por sua vez é derivada do descritor da aplicação, que contém os dados básicos de identificação de uma aplicação. A classe *Service* concentra todos os dados referentes a um serviço, como a listas UIs (*ServiceUI*); e cada UI, ou tela, tem uma classe *Form* correspondente, que é efetivamente a tela gerada pelo arquivo DynamicML após a análise e instanciação feita pelo *Engine*. Na Figura 11, a seguir, a representação da estrutura descrita.

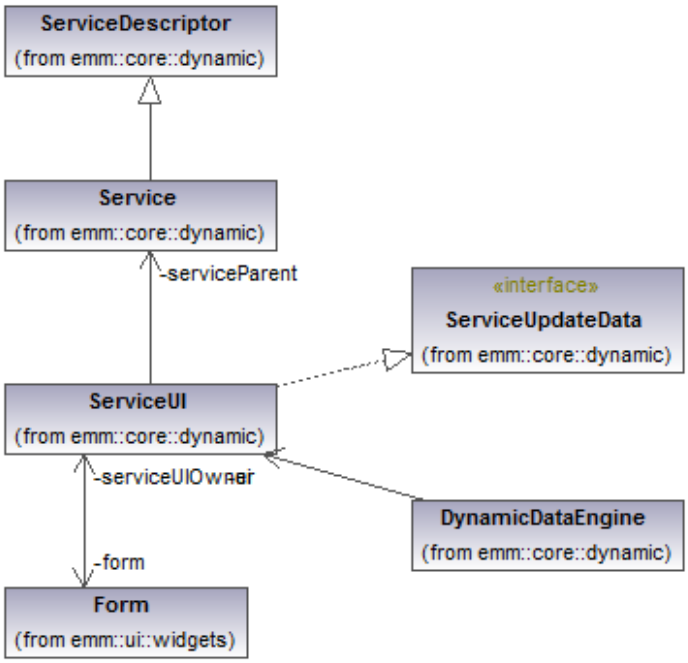

Figura 11 – Detalhamento dos relacionamentos de classes do Serviço.

A classe que, internamente, representa uma tela, a *ServiceUI* implementa a interface *ServiceUpdateData* que é usada para as chamadas de envio e recebimento de dados pela aplicação, um diferencial aqui é que essas são chamadas que devem ser executadas explicitamente pelo usuário, diferentemente da atualização automática de dados (quando disponível) feita pela interface *AutoRefreshDataListener* no gerenciador de serviços.

E finalmente, a classe *DynamicDataEngine* é instanciada pela *ServiceUI* sempre que for feito o recebimento ou envio de dados para o servidor; isso porque essa classe é a responsável pela conversão dos dados da aplicação para um arquivo DynamicML e vice-versa. Assim, existe um formato padrão: o mesmo usado no arquivo DynamicML para descrição de serviços, só que mais simples, pois somente os valores de cada campo são necessários, e não as suas propriedades de configuração.

4.4.1.3 Interface

Cada componente de interface é representado por duas classes: uma abstrata, usada para incluir um campo ID para identificar o componente unicamente dentro da estrutura do *framework* e para trabalhar generalizações de componentes (útil na expansão do

conjunto atual); a segunda, concreta, é a efetiva instância de um componente, responsável pelo seu desenho e funcionalidades.

E, como visto na Figura 12, todos os componentes devem implementar duas interfaces: *DynamicWidget* e *WidgetXMLParser*. A primeira é usada na manipulação dos componentes internamente pelo *framework* para identificar o componente e seu tipo, além de ler e gravar dados neste. A segunda é a responsável pela implementação da conversão do formato de dados do componente para DynamicML e vice-versa.

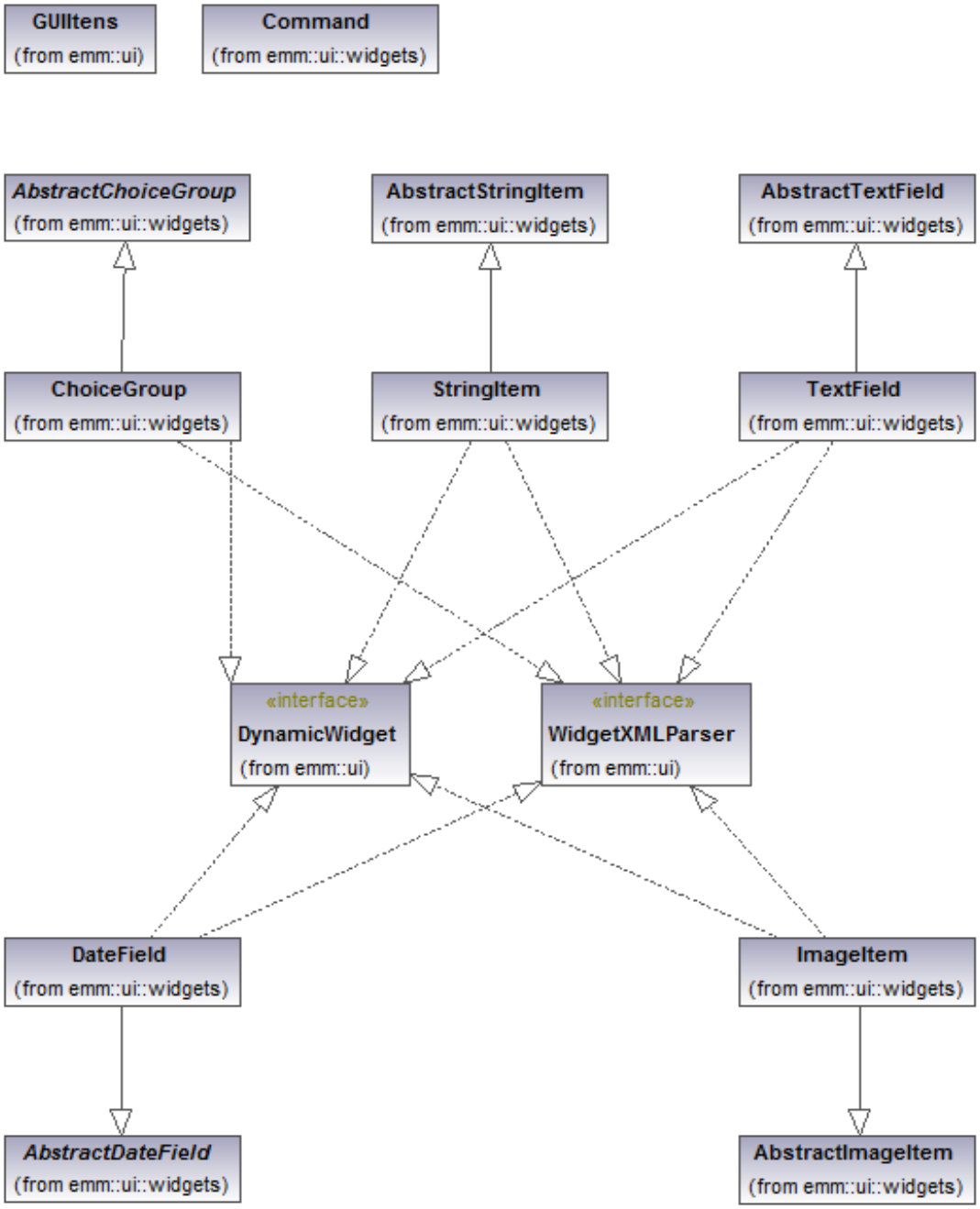

Figura 12 – Detalhamento dos relacionamentos de classes da Interface.

A classe *Command* responsável pela instanciação de botões de ação e navegação nas telas da aplicação.

E finalmente, a classe utilitária *GUIItens* que é onde se concentram todos os métodos de interpretação de propriedades dos componentes na linguagem DynamicML. Essa separação foi feita para manter as classes de componentes limpas e todos os métodos de processamento num só lugar.

4.4.1.4 Tratamento de Erros

As classes de erros são muito importantes no *framework*, pois como todo o processamento deve ser feito em tempo de execução, toda e qualquer anomalia deve ser capturada e tratada corretamente da forma mais transparente possível para o usuário, e sempre que possível o sistema deve ser capaz de recuperar seu estado funcional.

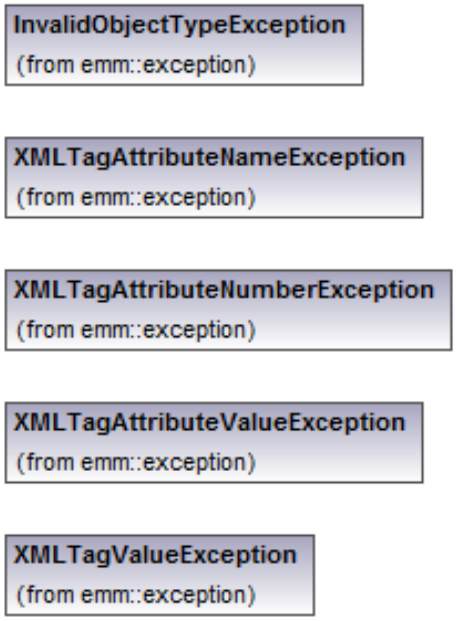

Figura 13 – Classes de tratamento de erros.

A maioria das classes de erros criadas foram para o processamento da linguagem DynamicML, sendo apenas uma (a primeira, de cima para baixo, da Figura 13) para tratar tentativa de manipulação de componente com tipo diferente do seu.

4.4.1.5 Teste e Pesquisa

As classes deste conjunto são classes de teste e de pesquisa, usadas ao longo do desenvolvimento do *framework*. Algumas são de recursos *beta* (não implementados no protótipo atual) e outras foram consideradas obsoletas, mas mantidas aqui como referência.

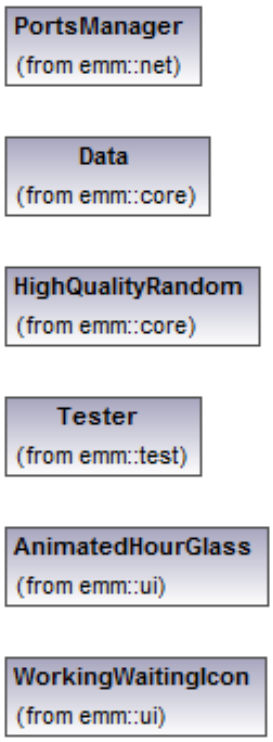

Figura 14 – Classes de testes, *debugging* e pesquisa.

4.4.2 Módulo *Core*

Na raiz do módulo, está a classe *Zip* que encapsula as chamadas à biblioteca de compactação, usada para diminuir o volume de dados trafegados na rede, uma exigência devido aos custos das redes móveis atuais; a classe *Data*, atualmente em estágio *beta* e, portanto não incluída na primeira versão do protótipo, seria usada para criar tipagem dinâmica de dados e assim aumentar a generalização do *framework*; a classe *DynamicMJ* que é na verdade uma aplicação exemplo usando este trabalho.

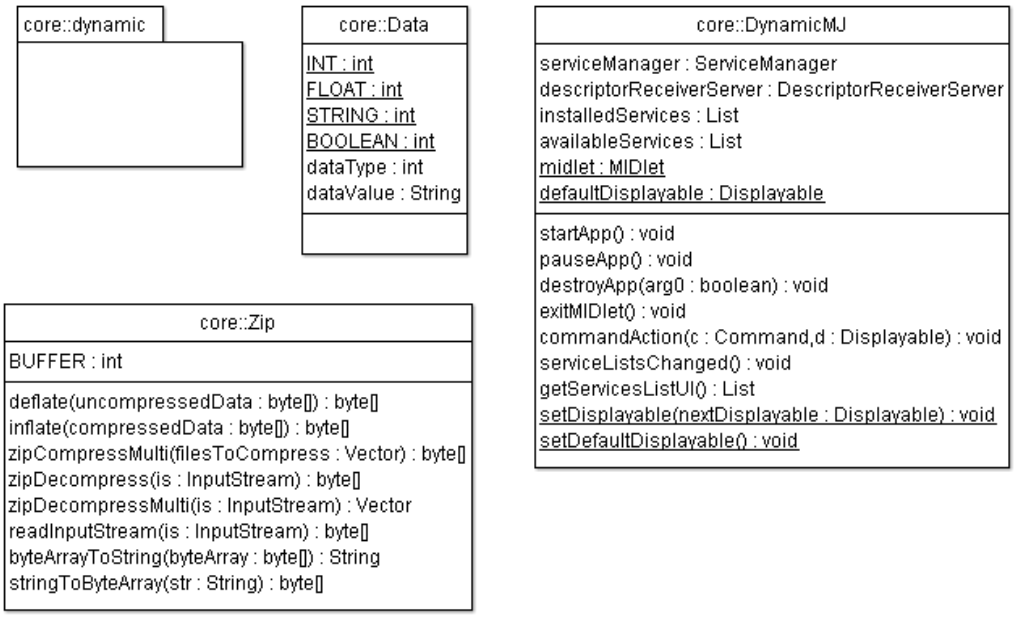

Figura 15 – Diagrama UML do módulo *Core*.

No subconjunto *dynamic* do módulo está a classe *ServiceManager*, que é a responsável por organizar os serviços disponíveis, instalados e fazer as chamadas de processamento dos XMLs; a interface *ServiceManagerListener* deve ser implementada por qualquer classe que deseje ser informada sobre mudanças nas listas de serviços no *ServiceManager* (novos serviços, serviços excluídos, etc.); a classe *Engine* é responsável por fazer a tradução do XML dos serviços e seus descritores, e assim gerar os dados necessários para instanciar os objetos correspondentes; a classe *DynamicDataEngine* é semelhante ao tradutor anterior, mas esta é a responsável somente pela tradução de atualizações dos dados de uma tela, ou seja, traduz os dados de um objeto na tela para XML e vice-versa, necessário na comunicação com o sistema externo; a classe *ServiceDescriptor* define os dados gerais de um serviço, usados principalmente para identificá-los unicamente; a classe *Service* extende a anterior e contém dados específicos do serviço, como sua URL de envio e\ou recebimento de dados, gerencia as telas da aplicação do serviço e o ponto de entrada desde (execução); a classe *ServiceUI* é a representação de uma tela, e um serviço pode ter de 1 a *n* telas, que são gerenciadas pela classe anterior, aqui é feito o gerenciamento dos componentes, e a implementação das interfaces para permitir os envio/recebimento de dados, e para atualização automática de dados; e como última classe de módulo está a *KXmlParser*, que é somente um encapsulamento para debug da biblioteca KXml usada.

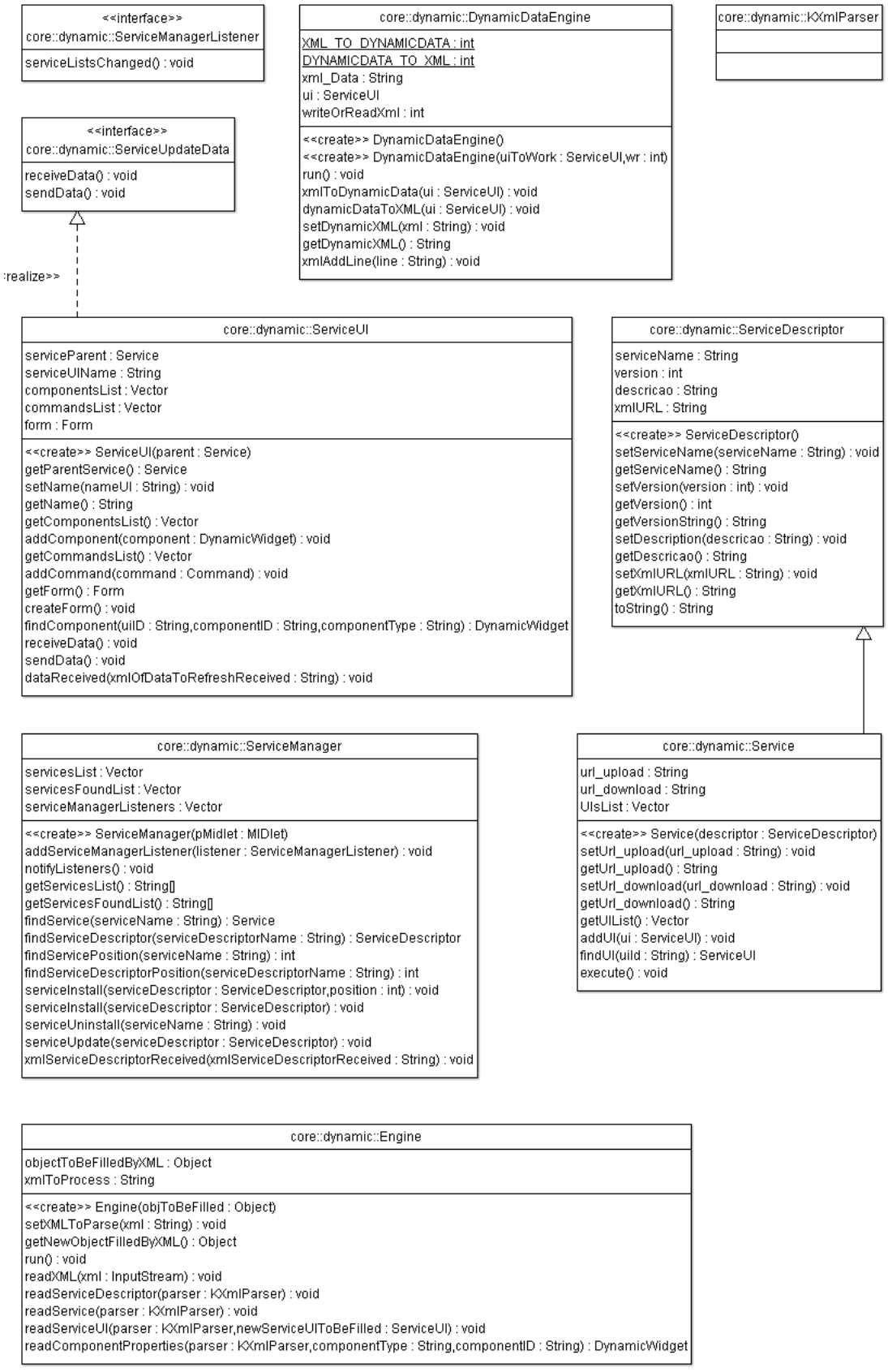

Figura 16 – Diagrama UML do módulo *Core.Dynamic*.

4.4.3 Módulo *Net*

Este módulo é o responsável por toda a comunicação do *framework*, atualmente existe parte da comunicação implementada através do UDP e HTTP (para demonstrar a possibilidade de uso em múltiplos protocolos).

A classe *DescriptorReceiverServer* é uma *Thread* que fica constantemente ativa e é responsável por receber os descritores de serviços encontrados, ou seja, ao entrar num ambiente com um novo serviço disponível, é essa classe que irá receber o descritor e repassá-lo ao *listener* da classe (no sistema exemplo, a classe *main* do projeto); a classe *AutoRefreshDataServer* também é uma *Thread* paralela, e tem como função receber atualizações de dados dos serviços, exemplo: um serviço que monitora uma dada temperatura e cada vez que a temperatura do sistema supervisionado mudar, o novo valor é enviado para o serviço, que recebe através desta classe. As duas classes receptoras de dados externos (*DescriptotReceiverServer* e *AutoRefreshDataServer*) recebem dados através do protocolo UDP; a classe *DynamicComm* é usada para enviar dados para o servidor do serviço, e também para requisitar dados dinâmicos como imagens e outros itens, essa comunicação é feita via HTTP; a classe *PortsManager*, que foi tornada obsoleta, tinha como objetivo apenas gerenciar o número das portas que cada serviço registrava para atualização automática de dados, ou seja, para cada serviço que desejasse receber dados automaticamente, este teria uma porta exclusiva (as portas iniciavam em 30.002 e iam até 30.011), contudo para contornar uma limitação de número de portas simultaneamente abertas, esta classe foi descontinuada e a *thread* que recebe atualização de dados usa a porta 30.002 para todos os serviços, e esta repassa os dados para o serviço destino de acordo com os IDs no XML de dados; por fim temos duas interfaces, *DescriptorReceiverListener* e *AutoRefreshDataListener*, a primeira deve ser implementada pela classe que quiser ser notificada do recebimento de novos descritores, no *framework* essa classe é o gerenciador de serviços (*ServiceManager*), e a segunda interface será implementada pela classe de interface (*ServiceUI*) que queira fazer uso do recurso de atualização de dados automática.

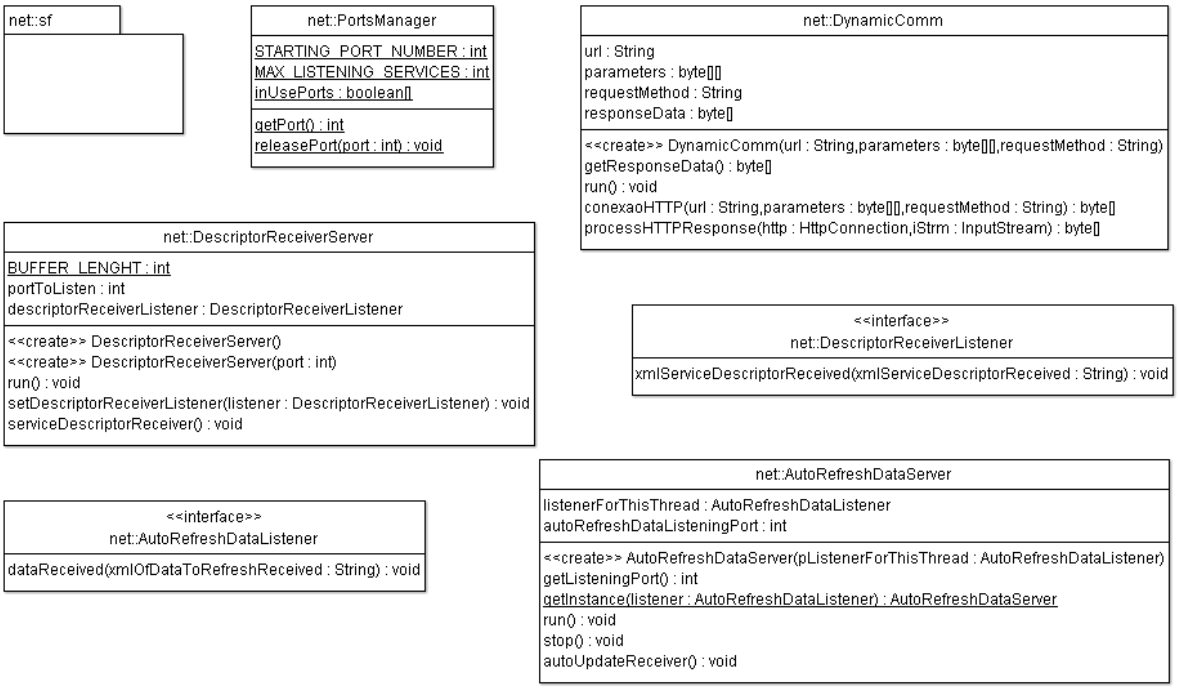

Figura 17 – Diagrama UML do módulo *Net*.

4.4.4 Módulo *UI*

Este módulo contém as classes usadas para componentizar e tornar a interface flexível e permitir a configuração de conjuntos dinâmicos de componentes durante a execução.

Inicialmente temos duas interfaces usadas na técnica criada para tornar os componentes dinâmicos, podendo ser configurados e alterados em tempo de execução: *WidgetXMLParser* e *DynamicWidget*. O primeiro exige apenas que o componente implemente um método para interpretar o XML específico para sua configuração, ou seja, que leia os campos do XML e instancie e configure um componente conforme as instruções lidas; o segundo exige a implementação de métodos ligados ao gerenciamento do componente: método para retornar o seu identificador único, seu tipo interno no *framework*, e métodos para visualizar e marcar o valor do componente. Existem duas classes de interface, que não estão sendo usadas ainda pois estão em estágio *beta*, que servem para criar um ícone animado no estilo "aguarde, processando..."; que são as classes *AnimatedHourGlass* e *WorkingWaitingIcon*.

Na raiz do módulo, por fim, temos a classe *GUIItens*, que funciona como um registro para todos os componentes usados no *framework*, tem uma enumeração dos componentes suportados e vários métodos auxiliares para o processo de interpretação do XML. Essa separação de códigos utilitários de interpretação de propriedades foi feita, pois existem várias propriedades comuns entre os componentes; assim centralizou-se todos os códigos referentes à interpretação de propriedades num só lugar, o que permitiu grande reuso de código, e ainda permitiu que as classes dos componentes ficassem mais "limpas", facilitando inclusive sua compreensão.

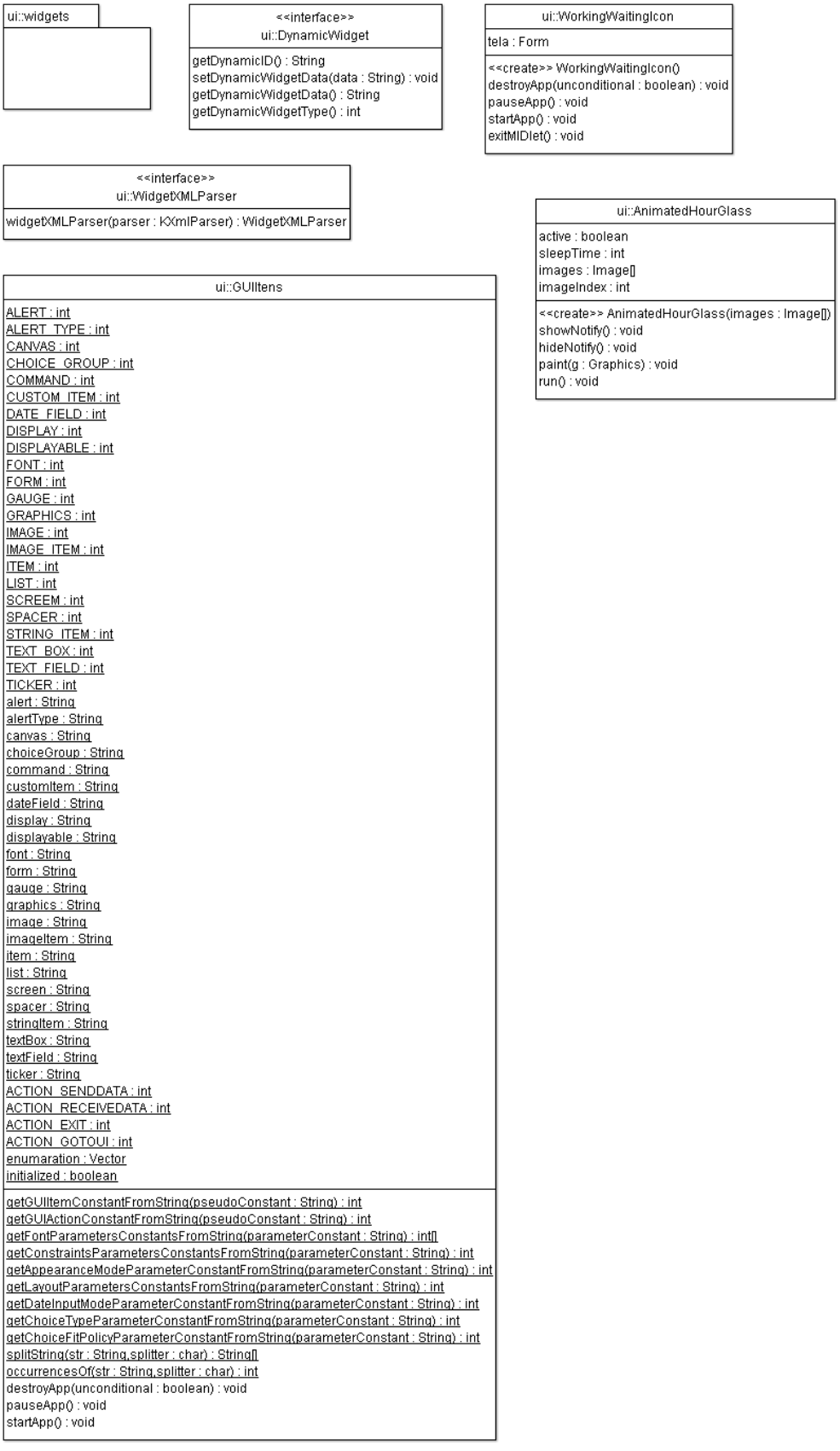

Figura 18 – Diagrama UML do módulo *UI*.

No subconjunto *Widgets* do módulo *UI* estão as classes de todos os componentes disponíveis atualmente no *framework*. As classes *Form* e *Command* extendem as respectivas classes da biblioteca Java e adicionam recursos para sua integração com o sistema proposto neste trabalho. Todos os outros componentes da biblioteca Java foram adaptados usando dois passos: primeiro foi criada uma classe abstrata, por exemplo, um *AbstractChoiceGroup*, para tratar a instanciação do objeto, identificação única interna, e conter qualquer código específico necessário do componente; num segundo passo foi criado a classe do componente propriamente dito, por exemplo um *ChoiceGroup*, que irá implementar as interfaces descritas no início deste módulo, além de extender a sua classe abstrata, citada acima. Neste segundo passo sim teremos componentes concretos, que podem ser instanciados e configurados em tempo de execução (obtendo assim aplicações dinâmicas).

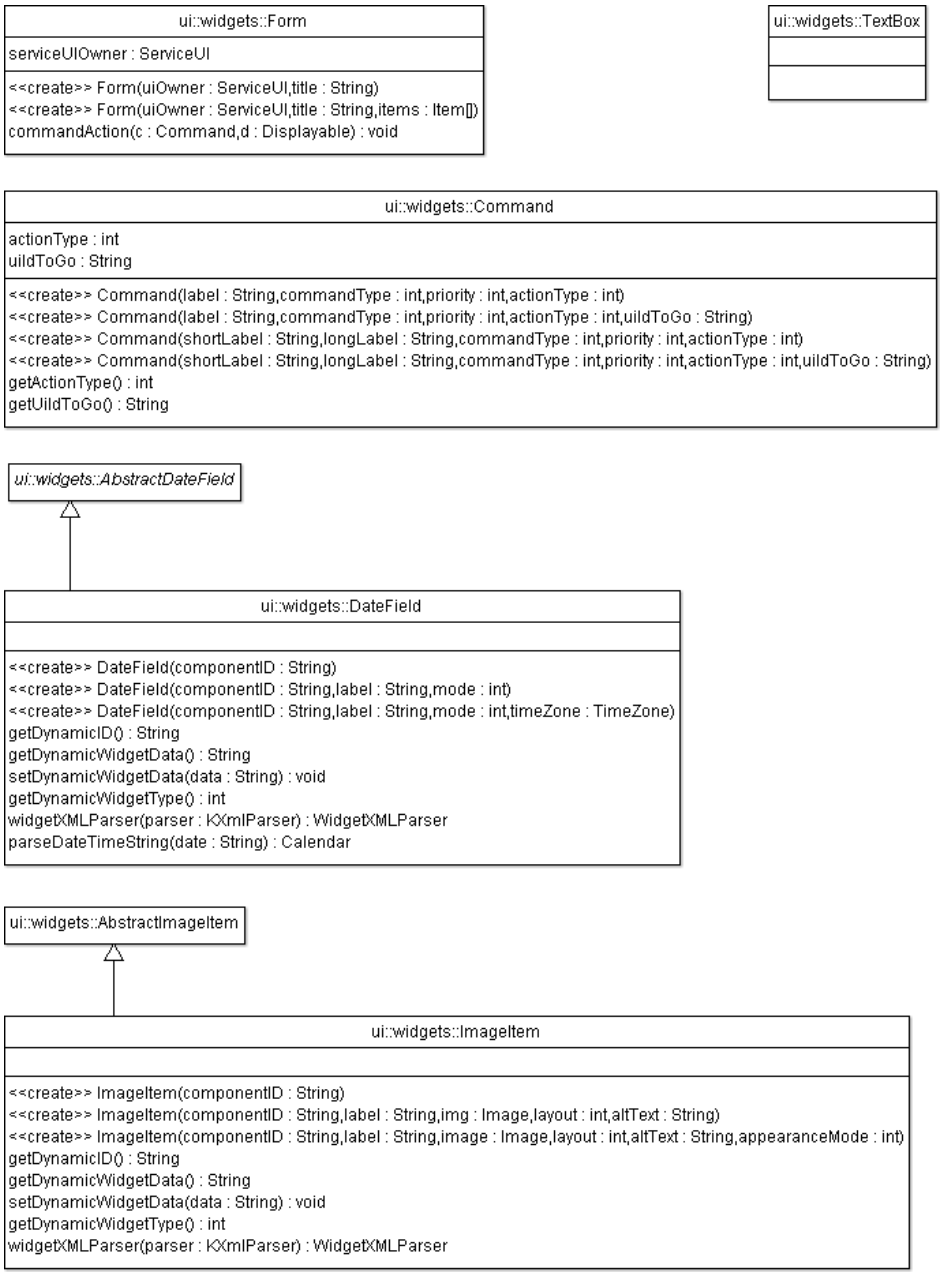

Figura 19 – Diagrama UML do módulo *UI.Widget*, parte um.

Atualmente, os componentes disponíveis são: *ChoiceGroup*, *DateFIeld*, *ImageItem*, *StringItem* e *TextField*; tendo também as classes *Form*, que faz o correspondente visual para classe *ServiceUI* do *framework*, e a classe *Command* que é usada para adaptar ações e navegação aos componentes e à interface da aplicação do serviço.

Nas Figuras 19 e 20 temos os diagramas UML das classes dos componentes disponíveis, e na Figura 18, o diagrama das classes da raiz do módulo.

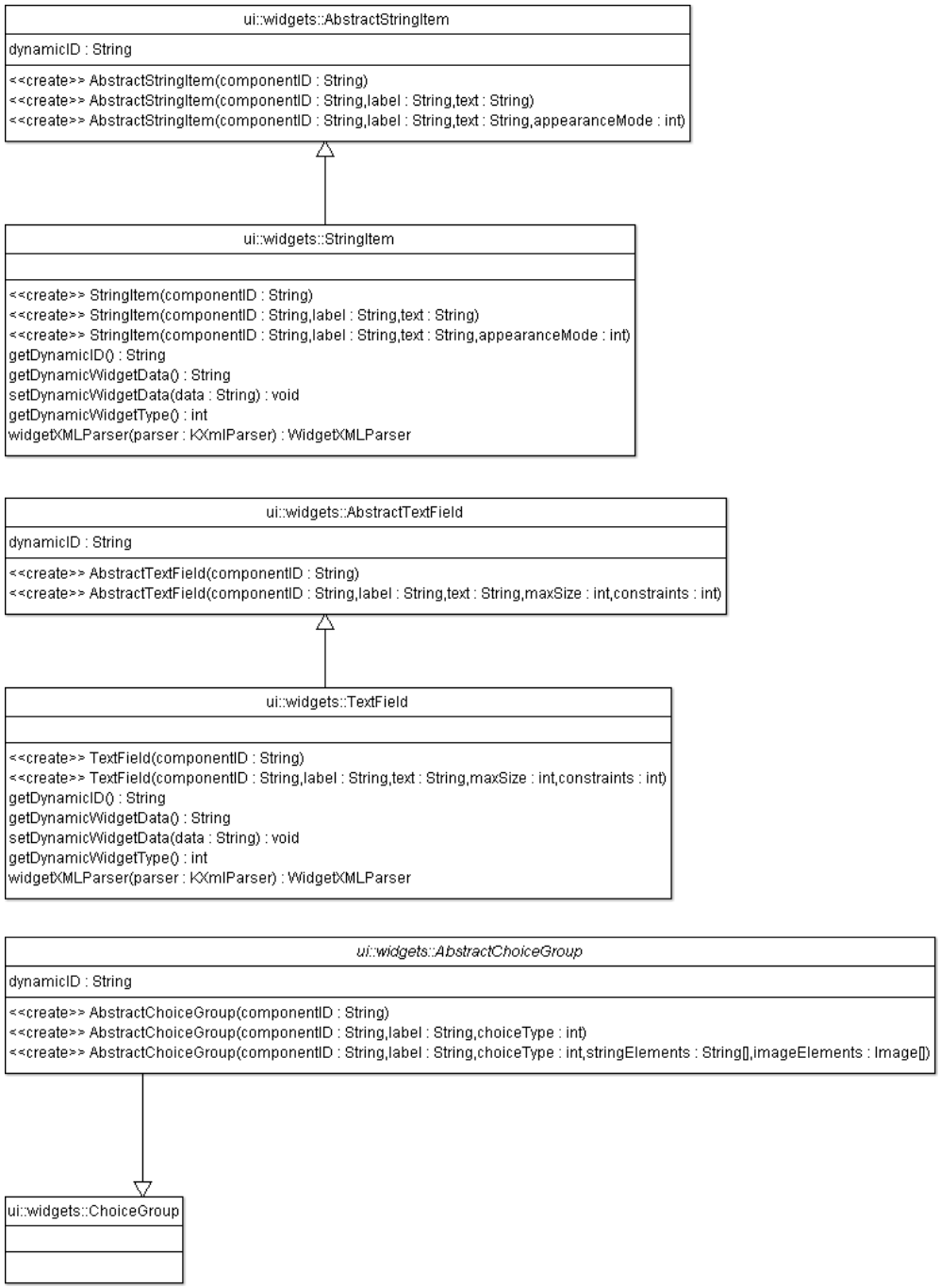

Figura 20 – Diagrama UML do módulo *UI.Widget*, parte dois.

4.5 Infraestrutura

Para integrar o *framework* DynamicML com sistemas reais é necessário uma camada de *software* intermediária, um tradutor, capaz de codificar os dados recebidos pelo aplicativo do serviço no dispositivo móvel para o sistema externo real, como um televisor, um motor, etc.

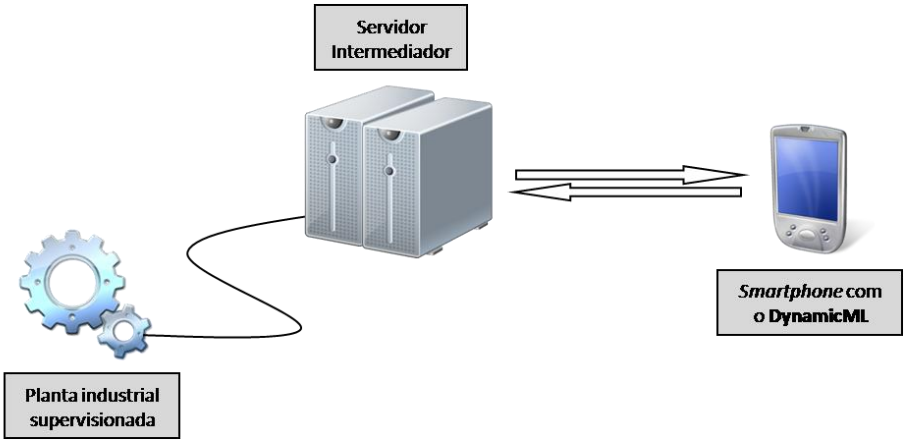

Figura 21 – Diagrama de funcionamento do DynamicML.

O servidor intermediador é necessário nos casos em que o sistema a ser supervisionado não possua interface de comunicação de alto nível (internet), ou use algum tipo de protocolo proprietário. Neste caso o papel do servidor é apenas ser um tradutor.

Caso o sistema supervisionado utilize os protocolos necessários, mas ainda não tenha capacidade computacional para formatar os dados no padrão funcional do DynamicML, também será necessário o uso do servidor intermediador. E este pode servir também como um ponto centralizador de serviços Em uma casa, por exemplo, todos os serviços disponíveis podem ser centralizados neste servidor e acessados por dispositivos móveis através deste ponto. Neste caso retira-se inclusive o problema de descoberta de serviço porque existirá uma central conhecida.

5 ESTUDO DE CASO

Para validar o trabalho desta dissertação e avaliar a eficiência do *framework* DynamicML, foram feitos três estudos de caso: dois na área de automação residencial, e um na área de automação industrial. Os estudos foram feitos com sistemas simulados, com o intuito apenas de validar o *framework* desenvolvido.

5.1 Automação Residencial

Esta é uma área que apresenta grande potencial de crescimento, pois a cada dia existem mais e mais itens eletrônicos nas residências, sendo necessário lidar com uma série de manuais, controles, fora as situações onde seria desejável a integração de mais de um aparelho.

Por isso o trabalho proposto tem um de seus testes de validação elaborado para atender a esse tema: automação residencial. A idéia é poder centralizar no celular o máximo de controles e serviços disponíveis por um lar, estando o usuário dentro ou fora deste (a política de segurança não é discutida aqui).

5.1.1 Controle Remoto

A aplicação de Controle Remoto consiste exatamente nisso, um controle remoto virtual para TV's ligados a um sistema computacional capaz de receber sinais externos. A idéia é concentrar o controle de aparelhos de TV em um único aparelho que esteja sempre à mão da pessoa, e a resposta é certamente um aparelho celular. Então, a partir deste aparelho, poder controlar qualquer aparelho de TV que se encontra no ambiente onde o usuário estiver no momento.

O funcionamento segue como o explicado no capítulo anterior, que descreve o funcionamento do *framework* em si: ao entrar num ambiente onde exista algum aparelho que forneça serviços a clientes, o dispositivo móvel é informado da disponibilidade de um novo serviço; ao instalar o aplicativo o usuário passa a interagir com o sistema do ambiente. No exemplo, o usuário passa a interagir com a TV, podendo ver o canal e volume atual, além de poder alterá-los.

Ao trocar de ambiente, ou num ambiente com vários aparelhos, o sistema não precisa ser re-instalado, é preciso apenas diferenciar a TV alvo dos comandos (usando um ID, por exemplo).

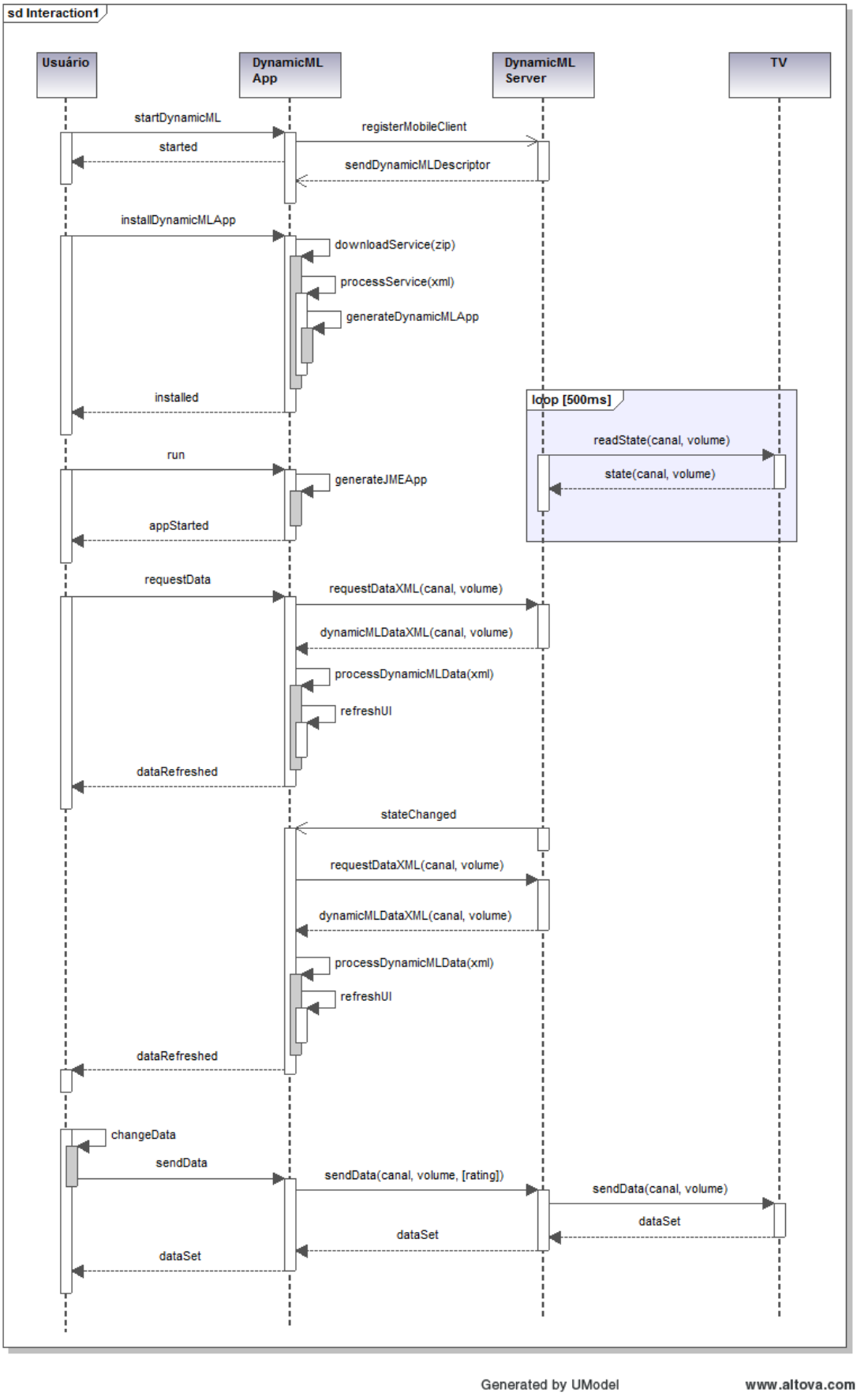

Figura 22 – Diagrama de sequência do aplicativo TVCommander.

Na Figura 23 é mostrada a aplicação exemplo executando em um emulador J2ME, e como recurso interessante acrescentado está a avaliação do programa atual, o que permitiria às emissoras de TVs auferir não só a audiência mas também a avaliação do espectador quanto a programação; com esse tipo de interação seria possível inclusive criar um sistema de recomendação da programação pelos próprios usuários.

Figura 23 – Aplicativo de controle remoto (com avaliação).

Já na Figura 24 vemos outros recursos que podem ser adicionados ao sistema de controle remoto: seleção de sinal da TV e apresentação da grade de programação. Outro exemplo bastante útil seria a possibilidade de supervisionar o que crianças assistem em casa mesmo seus pais estando fora (inclusive tempo de uso, etc), pois o acesso ao sistema não é restrito ao ambiente da TV.

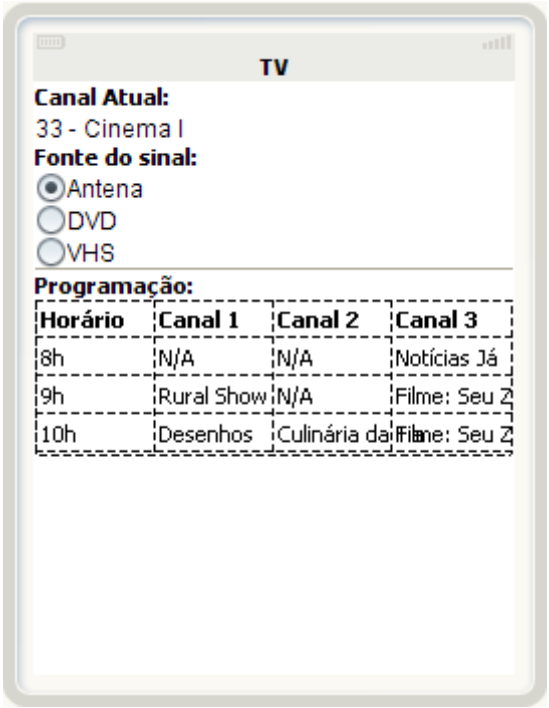

Figura 24 – Aplicativo de controle remoto.

Abaixo segue a listagem mostrando o arquivo fonte da aplicação TVCommander.

```
<?xml version="1.0" encoding="UTF-8"?>
<DynamicML>
    <Service Name="TVCommander" Version="200000">

        <Description>Controle remoto virtual para tv.</Description>

        <URL type="upload">http://skynet-s.no-ip.org:6969/TVCommander/SendData</URL>
        <URL type="download">http://skynet-s.no-ip.org:6969/TVCommander/GetData</URL>

        <UIList>
            <UI id="TV Commander">
                <ComponentsList>
                    <Component id="CHANNEL_LOGO" type="IMAGE_ITEM">
                        <Properties>
                            <Label></Label>
                            <Image>http://skynet-s.no-
ip.org:6969/DynamicML/TVCommander/tve.jpg</Image>
                            <Layout>LAYOUT_DEFAULT</Layout>
                            <AlternateText>Band logo!</AlternateText>
                            <AppearanceMode>PLAIN</AppearanceMode>
                        </Properties>
                    </Component>

                    <Component id="CHANNEL_NAME" type="STRING_ITEM">
                        <Properties>
                            <Label>Canal: </Label>
                            <Text>TVE</Text>
                            <Font>FACE_PROPORTIONAL,STYLE_BOLD,SIZE_MEDIUM</Font>
                            <Layout>LAYOUT_DEFAULT</Layout>
                            <AppearanceMode>PLAIN</AppearanceMode>
                        </Properties>
                    </Component>

                    <Component id="CHANNEL_NUMBER" type="TEXT_FIELD">
                        <Properties>
                            <Label>No: </Label>
                            <Text>10</Text>
                            <MaxSize>2</MaxSize>
                            <Constraints>NUMERIC</Constraints>
                            <Layout>LAYOUT_DEFAULT</Layout>
                        </Properties>
                    </Component>

                    <Component id="VOLUME" type="TEXT_FIELD">
                        <Properties>
                            <Label>Vol: </Label>
                            <Text>25</Text>
                            <MaxSize>3</MaxSize>
                            <Constraints>NUMERIC</Constraints>
                            <Layout>LAYOUT_DEFAULT</Layout>
                        </Properties>
                    </Component>

                    <Component id="VOTE" type="CHOICE_GROUP">
                        <Properties>
                            <Label>Avaliação do programa:</Label>
                            <ChoiceType>EXCLUSIVE</ChoiceType>
                            <Items>Ruim; ;Bom; ;Ótimo; </Items>
                            <Selected>0,1,0</Selected>
                            <Font>FACE_SYSTEM,STYLE_PLAIN,SIZE_SMALL</Font>
                            <FitPolicy>TEXT_WRAP_DEFAULT</FitPolicy>
                            <Layout>LAYOUT_DEFAULT</Layout>
                        </Properties>
                    </Component>
                </ComponentsList>
```

```
                    <Buttons>
                    <Command>ACTION_EXIT</Command>
                    <Command>ACTION_SENDDATA</Command>
                    <Command>ACTION_RECEIVEDATA</Command>
                </Buttons>
            </UI>
        </UIList>
    </Service>
</DynamicML>
```

5.1.2 Ambiente Inteligente

Uma aplicação exemplo para o contexto de ambientes inteligentes é o controle de uma sala e seus dispositivos de ambiente: luzes, ar, tela de projeção, etc.

Ao entrar em um ambiente com um dispositivo móvel usando o *framework* DynamicML, este é alertado pelo ambiente de que este segundo provê um serviço de controle. E caso o usuário instale o serviço, será possível controlar e inspecionar o estado da sala.

A partir deste ponto, é possível controlar cada conjunto de ambiente que a sala disponibiliza em seu serviço, como: sistema de iluminação, sistema de temperatura, sistema de projeção, etc. No aplicativo é possível ver o estado atual de cada um dos conjuntos e alterá-los.

Neste tipo de aplicação também tem-se que lidar com questões de segurança: será permitido ver/alterar as configurações do ambiente remotamente (distante do local alvo)? Recursos que requerem dados de localização não foram implementados, mas podem ser facilmente agregados ao trabalho, bastaria adicionar um módulo e um trecho de código no *Core* para tratar da localização nos trechos que tratam as ações nos aplicativos dos serviços.

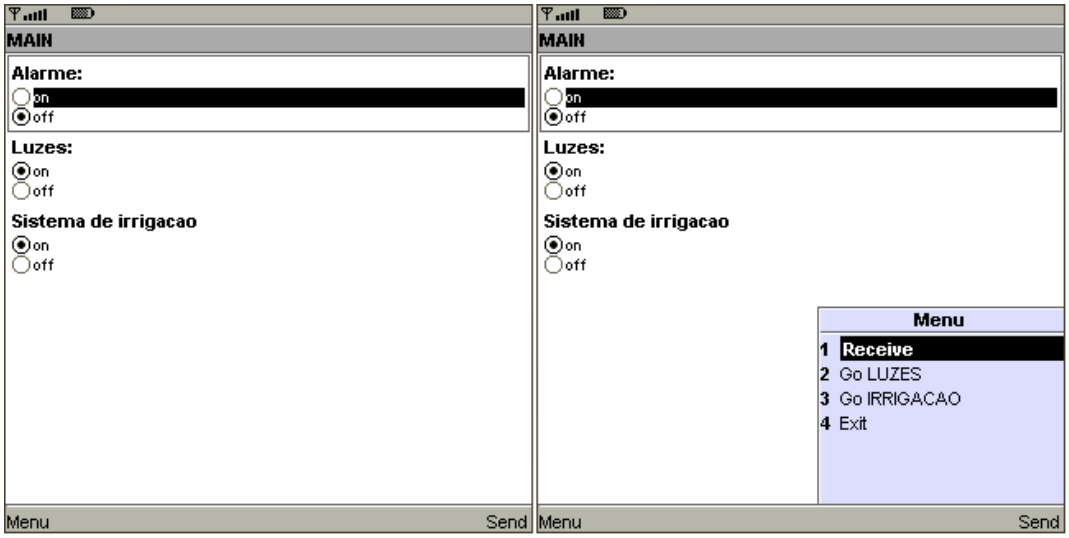

Figura 25 – Tela principal do sistema HouseMaster (automação residencial).

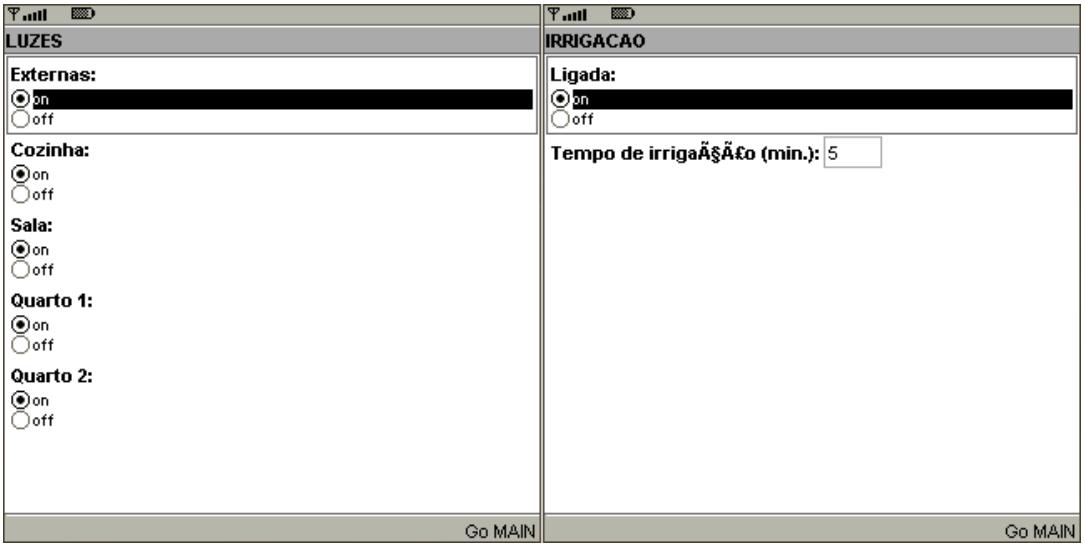

Figura 26 – Telas secundárias do HouseMaster (controle de iluminação e irrigação).

5.2 Automação Industrial

Aqui é onde temos a automação, de forma geral, mais difundida; pois melhoras no sistema de produção aumentam a produtividade (produtos de maior qualidade e custos competitivos), o que se reflete em maiores lucros, assim a automação tem um forte argumento para sua implantação na área industrial. E justamente por isso sistemas supervisórios são importantes aqui, pois permitem o acompanhamento e supervisão dos processos automatizados, e assim aferir e certificar do seu adequado funcionamento.

Então, um caso de teste simulando o uso do *framework* neste ambiente é fundamental para validar a importância e relevância deste trabalho.

5.2.1 Supervisório industrial

É muito comum na indústria que alguns supervisores regionais tenham de visitar constantemente plantas distantes, e nem sempre estas evoluem igualmente ao mesmo tempo, assim é comum terem disparidades tecnológicas, e por consequência a forma e meios para que o supervisor possa aferir cada uma pode variar. O que não é ideal.

A partir da proposta deste trabalho, cada planta industrial teria que ter apenas os sistemas que desejarem ser monitorados interligados a um servidor capaz de receber e enviar dados destes, e capaz de se comunicar via HTTP e/ou UDP.

No contexto do DynamicML, o supervisor teria o gerenciador de serviços pré-instalado em seu celular, por exemplo, e assim que ele entrasse em cada planta industrial, automaticamente seria notificado dos serviços disponíveis, naquele momento, para monitoração. Essa disponibilidade pode variar de acordo com localização e credencial (ambos recursos não foram projetados nesta versão inicial do trabalho).

Assim, uma aplicação exemplo neste contexto da automação industrial seriam sistemas de supervisão de componentes de uma industria química, por exemplo, onde eu pode-se acompanhar temperaturas, pressões e fazer alguns ajustes no processamento.

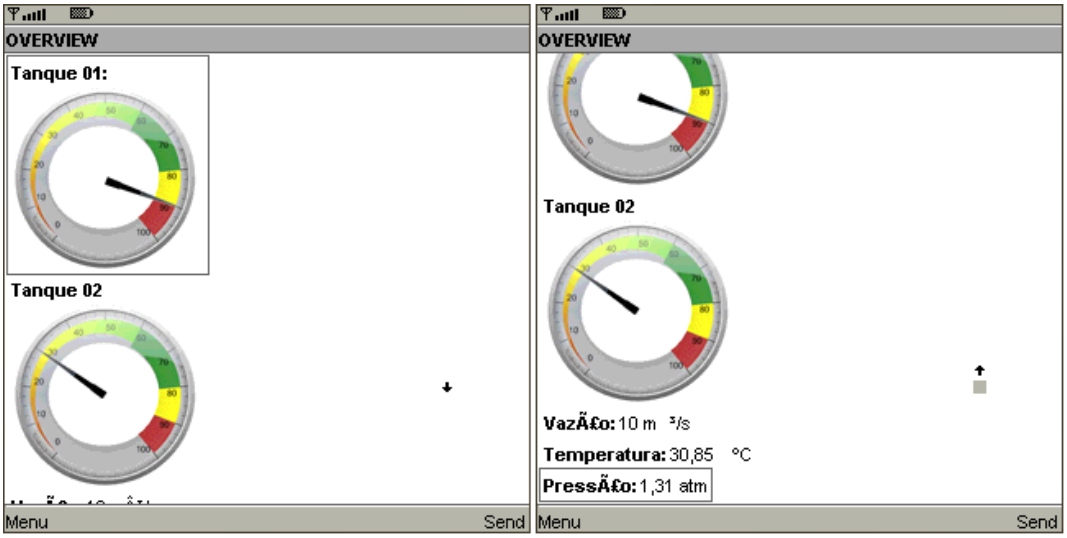

Figura 27 – Tela do sistema de supervisão industrial (Supervisory).

Abaixo segue a listagem mostrando o arquivo fonte da aplicação Supervisory.

```
<?xml version="1.0" encoding="UTF-8"?>
<DynamicML>
    <Service Name="Supervisory" Version="100000">

        <Description>Sistema de supervisão industrial.</Description>

        <URL type="upload">http://skynet-s.no-ip.org:6969/Supervisory/SendData</URL>
        <URL type="download">http://skynet-s.no-ip.org:6969/Supervisory/GetData</URL>

        <UIList>
            <UI id="OVERVIEW">
                <ComponentsList>
                    <Component id="RESERVOIR_01" type="IMAGE_ITEM">
                        <Properties>
                            <Label>Tanque 01:</Label>
                            <Image>http://skynet-s.no-
ip.org:6969/DynamicML/Supervisory/imgs/90.jpg</Image>
                            <Layout>LAYOUT_DEFAULT</Layout>
                            <AlternateText>reservoir load.</AlternateText>
                            <AppearanceMode>PLAIN</AppearanceMode>
                        </Properties>
                    </Component>

                    <Component id="RESERVOIR_02" type="IMAGE_ITEM">
                        <Properties>
                            <Label>Tanque 02</Label>
                            <Image>http://skynet-s.no-
ip.org:6969/DynamicML/Supervisory/imgs/30.jpg</Image>
                            <Layout>LAYOUT_DEFAULT</Layout>
                            <AlternateText>reservoir load.</AlternateText>
                            <AppearanceMode>PLAIN</AppearanceMode>
                        </Properties>
                    </Component>

                    <Component id="VAZAO" type="STRING_ITEM">
                        <Properties>
                            <Label>Vazão:</Label>
                            <Text>10 m³/s</Text>
                            <Font>FACE_PROPORTIONAL,STYLE_PLAIN,SIZE_MEDIUM</Font>
                            <Layout>LAYOUT_DEFAULT</Layout>
                            <AppearanceMode>PLAIN</AppearanceMode>
                        </Properties>
                    </Component>
```

```
                    <Component id="TEMPERATURA" type="STRING_ITEM">
                        <Properties>
                            <Label>Temperatura:</Label>
                            <Text>30,85 ºC</Text>
                            <Font>FACE_PROPORTIONAL,STYLE_PLAIN,SIZE_MEDIUM</Font>
                            <Layout>LAYOUT_DEFAULT</Layout>
                            <AppearanceMode>PLAIN</AppearanceMode>
                        </Properties>
                    </Component>

                    <Component id="PRESSAO" type="STRING_ITEM">
                        <Properties>
                            <Label>Pressão:</Label>
                            <Text>1,31 atm</Text>
                            <Font>FACE_PROPORTIONAL,STYLE_PLAIN,SIZE_MEDIUM</Font>
                            <Layout>LAYOUT_DEFAULT</Layout>
                            <AppearanceMode>PLAIN</AppearanceMode>
                        </Properties>
                    </Component>
                </ComponentsList>

                <Buttons>
                    <Command>ACTION_EXIT</Command>
                    <Command>ACTION_SENDDATA</Command>
                    <Command>ACTION_RECEIVEDATA</Command>
                </Buttons>
            </UI>
        </UIList>
    </Service>
</DynamicML>
```

6 CONCLUSÃO

Este trabalho propôs um *framework* para criação de aplicativos dinâmico em Java ME na configuração CLDC. A proposta é comparada a várias técnicas e vários trabalhos que abordam cada uma das técnicas, principalmente para salientar a importância da nossa contribuição. Este *framework* usa a ideia de parametrizar trechos dos MIDlet usando uma biblioteca (também resultado da proposta deste trabalho) e um arquivo XML de configuração. Isto torna possível criar aplicativos dinâmicos em dispositivos móveis (mesmo com hardware muito limitado).

As vantagens do *framework* proposto sobre as atuais soluções são: não é limitado a dispositivos específicos ou *hardwares* sofisticados, e também não necessita de *softwares* auxiliares implementados especificamente para cada aparelho alvo. O resultado deste trabalho deverá atender a qualquer dispositivo móvel que seja compatível com a CLDC 1.1.

Neste momento o *framework* ainda está sendo desenvolvido, e a especificação do formato do arquivo de configuração XML ainda está sujeita a alterações. Por enquanto só é possível adicionar serviços de entrada e saída de dados (sem interface gráfica). São 3 tarefas em andamento: código do gerente de serviços (responsável por receber notificações de serviços e comandar as adaptações do software em tempo de execução), as classes da biblioteca para permitir a configuração dinâmica e a especificação do arquivo XML de configurações.

Trabalhos futuros envolvem o término de um protótipo do *framework* para ter alguns resultados mais palpáveis e poder submeter artigos da proposta a algumas conferências sobre o a tema.

6.1 Trabalhos futuros

O *framework* DynamicML, no atual estágio de desenvolvimento, mostrou atender de forma satisfatória ao propósito de permitir a criação de sistemas dinâmicos para plataformas móveis, no entanto ainda tem um longo caminho de possibilidades a serem exploradas.

Como sugestão para novos trabalhos ficam os seguintes itens:

- Comparativo da linguagem DynamicML com as novas linguagens para desenho de interfaces para aplicações móveis: QML da linguagem QT (disponível a partir da versão 4.7 e usada para Symbiam e MeeGo) e a linguagem XML usada no Android.
- Criação e/ou integração com um sistema de descoberta automática de serviços, ou seja, o dispositivo ser capaz de determinar o local em que se encontra e detectar os serviços disponíveis neste.

- Ampliação do módulo *Core* para que seja possível interpretar inclusive trechos de algoritmos no dispositivo móvel, através do arquivo de configuração.
- Estudo do impacto de novas tecnologias, exemplo: plataformas Android e Apple IPhone, na aplicabilidade deste trabalho.
- Desenvolvimento de uma camada de segurança para aplicações no contexto de sistemas supervisórios em ambientes industriais críticos.
- Criação de uma plataforma de desenvolvimento (IDE, do inglês *Integrated Development Environment*) para facilitar a construção de aplicações usando a linguagem DynamicML.
- Aprimoramento e extensão da biblioteca de componentes, e melhorias correspondentes na linguagem DynamicML.

Alguns dos itens acima não foram desenvolvidos neste trabalho de forma intencional, a fim de não tornar o trabalho muito extenso e evitar que o tema fosse excessivamente abrangente e acabando por não ter uma conclusão.

O primeiro item, a comparação com as novas linguagens QML (do Nokia Qt *framework*) e Android, não foi realizada pois as linguagens são novas (o Android foi lançado em 2009; e a QML surgiu em 2009, sendo somente liberada em versão estável em 2010). E estas linguagens têm, hoje, objetivos diferentes da DynamicML, que é uma metalinguagem para geração de aplicações inteiras em tempo de execução; enquanto as primeiras são apenas linguagens descritivas de interface, usadas para melhorar o desenvolvimento de UI's. Ainda assim, o interessante de uma futura análise comparativa é poder inspirar novos recursos e funcionalidades para a DynamicML.

O segundo e terceiro itens foram deixados em aberto intencionalmente, pois são temas extensos, dando assunto para um trabalho inteiro. Considerando ainda que suas implementações não são simples, o que estenderia por demais o tempo de desenvolvimento do protótipo inicial.

Os demais itens sugeridos são estudos complementares, interessantes e vitais para a continuidade do projeto DynamicML, contudo, não eram essenciais para o desenvolvimento de uma primeira versão.

Apesar dos itens dois e três, mencionados acima, terem ficado de fora deste trabalho inicial do *framework*, a inclusão destes em trabalhos futuros são as propostas de maior relevância, pois sem dúvida nenhuma são as que podem acrescentar maior importância ao trabalho, além de poder expandir seu uso para novas aplicações. Assim, fica inclusive uma sugestão para a sequência de novos trabalhos inspirados aqui.

REFERÊNCIAS

/K/ EMBEDDED JAVA SOLUTIONS. Mika VM. **/K/ Embedded Java Solutions**, 2010. Disponivel em: <http://www.k-embedded-java.com>. Acesso em: 2 Dezembro 2010.

ABOWD, G. D. et al. **Towards a Better Understanding of Context and Context-Awareness**. In: INTERNATIONAL SYMPOSIUM ON HANDHELD AND UBIQUITOUS COMPUTING, HUC '99, 1999, Karlsruhe, Germany. **Proceedings...** [s.l.]: [s.n.]. 1999.

AMOR, D. **Internet Future Strategies:** How Pervasive Computing Will Change the World. 1ª Edição. ed. [s.l.]: Prentice Hall PTR, 2001. 320 p. ISBN 978-0130418036.

ANDROID (operating system). **Wikipedia**, 2010. Disponivel em: <http://en.wikipedia.org/wiki/Android_%28operating_system%29>. Acesso em: 7 Dezembro 2010.

APACHE FELIX. Apache Felix. **Apache Felix**, 2010. Disponivel em: <http://felix.apache.org/>. Acesso em: 6 Dezembro 2010.

BEERS, D. Is OSGi the Solution for Mobile Java? **InfoQ**, 2007. Disponivel em: <http://www.infoq.com/news/2007/05/osgi-javame>. Acesso em: 2 Dezembro 2010.

BITPIPE.COM. Wireless Computing. **Bitpipe.com**, 2010. Disponivel em: <http://www.bitpipe.com/tlist/Wireless-Computing.html>. Acesso em: 4 Dezembro 2010.

BOWEN, C. L.; BUENNEMEYER, T. K.; THOMAS, R. W. **Next Generation SCADA Security:** Best Practices and Client Puzzles. In: INFORMATION ASSURANCE WORKSHOP, IAW '05, 2005, Waikoloa, Hawaii, USA. **Proceedings from the Sixth Annual IEEE SMC.** [s.n.]. p. 426-427. 2005.

CONCIERGE GROUP. Concierge OSGi. **Concierge OSGi**, 2009. Disponivel em: <http://concierge.sourceforge.net>. Acesso em: 2 Dezembro 2010.

DAY, B. What is preverification? **jGuru**, 2000. Disponivel em: <http://www.jguru.com/faq/view.jsp?EID=201507>. Acesso em: 2 Dezembro 2010.

DUCATEL, K. et al. **Scenarios for ambient intelligence in 2010**. Sevilha: Office for official publications of the European Communities, 2001.

DYNAMIC Definition. **Webopedia**. Disponivel em: <http://www.webopedia.com/term/d/dynamic.html>. Acesso em: 24 Novembro 2010.

ECLIPSE EQUINOX. Eclipse Equinox. **Eclipse Equinox**, 2010. Disponivel em: <http://www.eclipse.org/equinox/>. Acesso em: 6 Dezembro 2010.

FREI, A.; ALONSO, G. A dynamic Architecture for Pervasive Computing. **CiteSeerX**, 2004. Disponivel em: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.90.2303&rep=rep1&type=pdf>. Acesso em: 2 Dezembro 2010.

FREI, A.; ALONSO, G. **A dynamic lightweight Platform for Ad-hoc Infrastructures**. Pervasive Computing and Communications, 2005. PerCom 2005. Third IEEE International Conference. [S.l.]: [s.n.]. 2005. p. 373-382.

GIGUÈRE, E. The Connected Limited Device Configuration (CLDC). **Developer.com**, 2002. Disponivel em: <http://www.developer.com/java/j2me/article.php/1436051>. Acesso em: 2 Dezembro 2010.

HAIGES, S. OSGi Tutorial, A Step by Step Introduction to OSGi Programming Based on the Open Source Knopflerfish OSGi Framework. **Knopflerfish**, 2004. Disponivel em: <http://www.knopflerfish.org/tutorials/osgi_tutorial.pdf>. Acesso em: 2 Dezembro 2010.

J2ME Programming. [s.l.]: WikiBooks, 2010. ISBN http://en.wikibooks.org/wiki/J2ME_Programming.

JADABS GROUP. Jadabs-CLDC. **Jadabs-CLDC**, 2004. Disponivel em: <http://jadabs.berlios.de/jadabs-cldc/>. Acesso em: 2 Dezembro 2010.

JAMVM GROUP. JamVM. **JamVM Project**, 2010. Disponivel em: <http://jamvm.sourceforge.net>. Acesso em: 2 Dezembro 2010.

JAVA Platform, Micro Edition. **Wikipedia**. Disponivel em: <http://en.wikipedia.org/wiki/Java_Platform,_Micro_Edition>. Acesso em: 24 Novembro 2010.

JCP. **JSR 30: J2ME CLDC (Connected, Limited Device Configuration)**. Java Community Process. Palo Alto. 2000. (http://jcp.org/en/jsr/summary?id=30).

JCP. **JSR 66: RMI Optional Package Specification 1.0**. Java Community Process. Palo Alto. 2002. (http://jcp.org/en/jsr/detail?id=66).

JCP. **JSR 36: Connected Device Configuration**. Java Community Process. Palo Alto. 2005. (http://jcp.org/en/jsr/detail?id=36).

JCP. **JSR 218: Connected Device Configuration (CDC) 1.1**. Java Community Process. Palo Alto. 2006. (http://jcp.org/en/jsr/detail?id=218).

JCP. **JSR 139: Connected Limited Device Configuration 1.1**. Java Community Process. Palo Alto. 2007. (http://jcp.org/en/jsr/summary?id=139).

JCP. **JSR 232: Mobile Operational Management**. Java Community Process. Palo Alto. 2008. (http://jcp.org/en/jsr/detail?id=232).

KLEINROCK, L. **Nomadic Computing**. Keynote at International Conf. on Mobile Computing and Networking (MOBICOM '95). [s.l.]: [s.n.]. 1995.

KNOPFLERFISH GROUP. Knopflerfish OSGi. **Knopflerfish**, 2010. Disponivel em: <http://www.knopflerfish.org>. Acesso em: 2 Dezembro 2010.

KOSTAKOS, V. J2ME vs.Net Compact. **Cityware**. Disponivel em: <http://www.cityware.org.uk/index.php?option=com_content&task=view&id=21&Itemid=41>. Acesso em: 24 Novembro 2010.

KRIENS, P. Interesting Times. **OSGi Alliance**, 2006. Disponivel em: <http://www.osgi.org/blog/2006/07/interesting-times.html>. Acesso em: 2 Dezembro 2010.

LISTA de Máquinas Virtuais Java. **Wikipedia**. Disponivel em: <http://en.wikipedia.org/wiki/List_of_Java_virtual_machines>. Acesso em: 24 Novembro 2010.

LISTA de Sistemas Operacionais Embarcados. **Wikipedia**. Disponivel em: <http://en.wikipedia.org/wiki/List_of_operating_systems#Embedded>. Acesso em: 24 Novembro 2010.

MAREJKA, R. Java ME Technology: Everything a Developer Needs for the Mobile Market. **Oracle - Sun Developer Network**, Julho 2008. Disponivel em: <http://java.sun.com/developer/technicalArticles/javame/mobilemarket/>. Acesso em: 24 Novembro 2010.

MARQUES, G. Economia e Negócios: Número de celulares no Brasil chega a 175,6 milhões. **O Estado de São Paulo**, 2010. Disponivel em: <http://economia.estadao.com.br/noticias/not_5884.htm>. Acesso em: 7 Dezembro 2010.

MINGO, S. A Breif History of Automation: The Past, Present and Future of the Industry. **Insurance Journal**, 15 Maio 2000. Disponivel em: <http://www.insurancejournal.com/magazines/west/2000/05/15/coverstory/21633.htm>. Acesso em: 24 Novembro 2010.

MOBILE Agent. **Wikipedia**, 2010. Disponivel em: <http://en.wikipedia.org/wiki/Mobile_agent>. Acesso em: 4 Dezembro 2010.

MOBILE and Embedded Community. **Java.Net**. Disponivel em: <http://community.java.net/mobileandembedded/>. Acesso em: 24 Novembro 2010.

NOKIA OS. **Wikipedia**. Disponivel em: <http://en.wikipedia.org/wiki/Nokia_OS>. Acesso em: 24 Novembro 2010.

ORACLE DEVELOPER NETWORK. Java ME Products. **Oracle Developer Network**, 2010. Disponivel em: <http://java.sun.com/javame/overview/products.jsp>. Acesso em: 5 Dezembro 2010.

ORACLE TECHNETWORK. Java ME. **Oracle TechNetwork**. Disponivel em: <http://www.oracle.com/technetwork/java/javame/overview/index.html>. Acesso em: 24 Novembro 2010.

OSGI ALLIANCE. OSGi Alliance. **OSGi Alliance**, 2010. Disponivel em: <www.osgi.org>. Acesso em: 2 Dezembro 2010.

PALAZZO M. DE OLIVEIRA, J. et al. **Sistemas de Informação Distribuídos**. Porto Alegre: WikiBooks, 2006. Disponivel em: <http://pt.wikibooks.org/wiki/Sistemas_de_Informa%C3%A7%C3%A3o_Distribu%C3%ADdos>. Acesso em: 24 Novembro 2010.

PEROZZO, R. F.; PEREIRA, C. E. **Arquitetura para Construção de Sistemas Supervisórios em Dispositivos Móveis**. In: IBERIAN LATIN AMERICAN CONGRESS ON COMPUTATIONAL METHODS IN ENGINEERING, XXVII CILAMCE, 2006, Belém, Pará, Brazil. **Proceedings of the XXVII CILAMCE**. [s.n.]. 2006.

PEROZZO, R. F.; PEREIRA, C. E. **Framework for Building Supervisory Systems in Mobile Devices**. In: 11th IEEE INTERNATIONAL CONFERENCE ON EMERGING TECHNOLOGIES AND FACTORY AUTOMATION, ETFA '06, 2006, Praga, República Checa. **Proceedings...** [s.n.]. p. 167-172. 2006.

PETREA, L. L.; GRIGORAS, D. **Dynamic Class Provisioning on Mobile Devices**. Parallel and Distributed Computing, International Symposium, Proceedings of The Fifth International Symposium on Parallel and Distributed Computing (ISPDC'06). [s.l.]: [s.n.]. 2006. p. 140-147.

PINTO, J. A short history of Automation growth. **Automation.com**, 24 Abril 2007. Disponivel em: <http://www.automation.com/sitepages/pid2900.php>. Acesso em: 24 Novembro 2010.

PROSYST. mBedded Server CLDC Edition - Data Sheet. **ProSyst**, 2006. Disponivel em: <http://dz.prosyst.com/datasheets/FWandExtensions/CLDC_OSGi_Framework.pdf>. Acesso em: 2 Dezembro 2010.

PROSYST. ProSyst. **ProSyst**, 2010. Disponivel em: <http://www.prosyst.com>. Acesso em: 2 Dezembro 2010.

QML. **Wikipedia**, 2010. Disponivel em: <http://en.wikipedia.org/wiki/QML>. Acesso em: 7 Dezembro 2010.

RELLERMEYER, J. S.; ALONSO, G. **Concierge:** A Service Platform for Resource-Constrained Devices. The Proceedings of the EuroSys 2007 Conference and ACM SIGOPS Operating Systems Review. [S.l.]: [s.n.]. 2007.

SEARCHMOBILECOMPUTING.COM. SearchMobileComputing.com Definitions: Nomadic Computing. **SearchMobileComputing.com**, 2002. Disponivel em: <http://searchmobilecomputing.techtarget.com/definition/nomadic-computing>. Acesso em: 4 Dezembro 2010.

SUN MICROSYSTEMS. Java 2 Platform - Micro Edition Datasheet. **UFBA - Disciplina de Programação em Java**, [s.l.]. Disponivel em: <https://disciplinas.dcc.ufba.br/pastas/MAT169/cdJava/j2me-ds.pdf>. Acesso em: 24 Novembro 2010.

UBIQUITOUS Computing. **Wikipedia**. Disponivel em: <http://en.wikipedia.org/wiki/Ubiquitous_computing>. Acesso em: 24 Novembro 2010.

WEISER, M. The computer for the 21st century. **Scientific American**, Setembro 1991. ISSN http://dx.doi.org/10.1145/329124.329126.

WEISER, M. Ubiquitous Computing. **Computer**, p. 71-72, Outubro 1993.

WILLIG, A.; MATHEUS, K.; WOLISZ, A. Wireless Technology in Industrial Networks. Proceedings of the **IEEE**. [s.l.]: [s.n.]. 2005. p. 1130-1151.

YORK, J.; PENDHARKAR, P. C. Human–computer interaction issues for mobile computing in a variable work context. **International Journal of Human-Computer Studies**, HCI Issues in Mobile Computing. [S.l.]: [s.n.]. 2004. p. 771-797.

YOURDICTIONARY.COM. Computer Dictionary Definition: Nomadic Computing. **YourDictionary.com**, 2010. Disponivel em: <http://computer.yourdictionary.com/nomadic-computing>. Acesso em: 4 Dezembro 2010.

APÊNDICE

7 MANUAL DA LINGUAGEM DYNAMICML

A construção de aplicativos usando a linguagem pode ser comparada à montagem de blocos em um formato de árvore, ou seja, a partir de um bloco raiz vai-se adicionando novos blocos, e cada bloco a partir daí pode ter novo blocos filhos e/ou irmãos (dependo do tipo do nó).

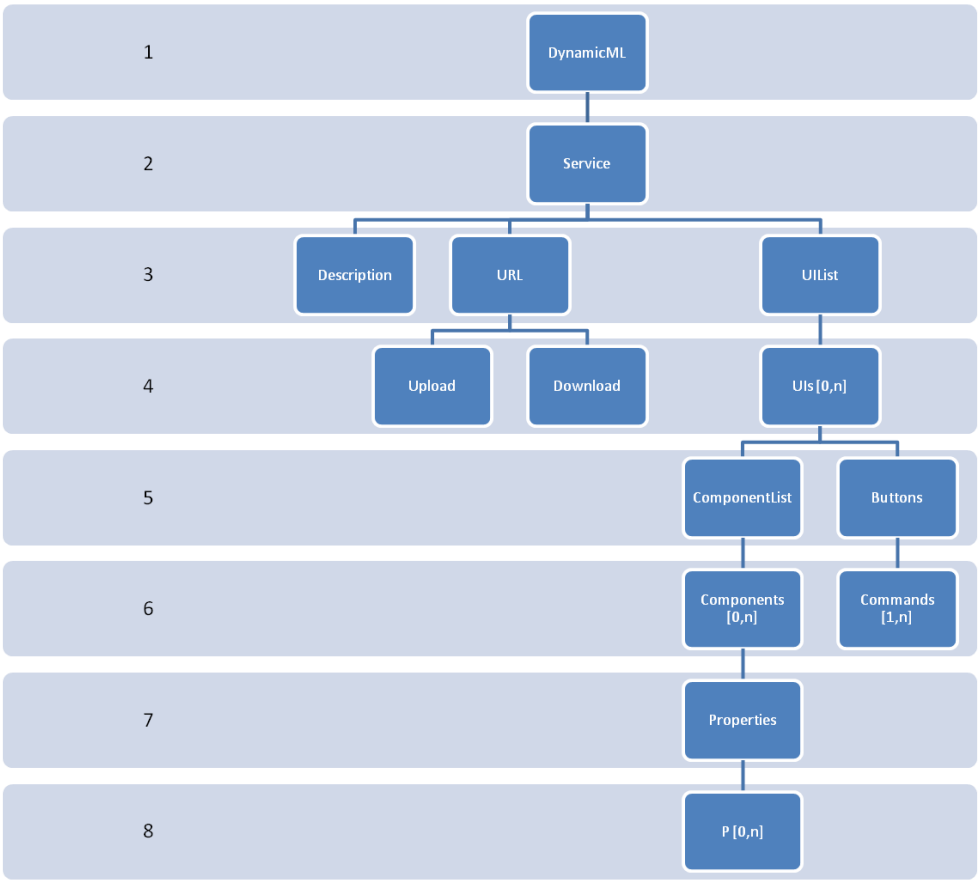

Figura 28 – Arquitetura de um documento DynamicML.

O nó raiz **<DynamicML>** indica o início do arquivo, e este pode conter de 1 a *n* aplicações, aqui chamadas de serviços. O *framework* foi projetado para ser capaz de processar mais de um serviço por arquivo para otimizar o tráfego de rede; porém para simplificar o desenvolvimento do protótipo, atualmente só é processado um serviço por arquivo.

7.1 Corpo de uma Aplicação (um serviço)

Um serviço (*tag* **<Service name="*serviceID*" version="*serviceVersion*">**), como visto na Figura 28, é composto por:

- sua descrição em **<Description>;**
- dois endereços *web* (um indica a URL para envio de dados pelo aplicativo, e o outro indica a URL para receber dados do servidor). Ambos usam a *tag* **<URL type="*(upload|download)*">**; e,
- uma lista de telas, ou simplesmente interfaces de usuário, na *tag* **<UIList>**.

Assim, tem-se como layout básico de um aplicativo DynamicML a listagem abaixo:

```
<?xml version="1.0" encoding="UTF-8"?>
<DynamicML>
    <Service Name="DynamicMJ" Version="100001">

        <Description>Descrição deste Serviço.</Description>

        <URL type="upload">http://127.0.0.1:6969/DynamicMJServer/SendData</URL>
        <URL type="download">http://127.0.0.1:6969/DynamicMJServer/GetData</URL>

        <UIList>
            .
            .
            .
        </UIList>
    </Service>
</DynamicML>
```

Todas as tags nesse bloco básico são obrigatórias, inclusive seus respectivos atributos.

7.2 Uma Interface de Usuário, ou tela (UI)

Dentro da lista de interfaces, podemos ter de 1 a *n* telas numa aplicação, cada uma destas delimitada por uma *tag* **<UI id="uiID">**. Único cuidado aqui é indicar um ID válido (alfanumérico e sem espaços) e único.

Uma UI é contém somente dois nós filhos: uma lista de componentes, que irão compor a interface e uma lista de comandos.

```
<UI id="UI2">
    <ComponentsList>
        <Component id="STRING_ITEM" type="STRING_ITEM">
            <Properties>
                <Label>Título do StringItem</Label>
                <Text>Texto, texto, texto.</Text>
                <Font>FACE_PROPORTIONAL,STYLE_BOLD,STYLE_UNDERLINED,SIZE_MEDIUM</Font>
                <Layout>LAYOUT_DEFAULT</Layout>
            </Properties>
        </Component>
    </ComponentsList>
    <Buttons>
        <Command ui="TESTE_GUI">ACTION_GOTOUI</Command>
    </Buttons>
</UI>
```

Na listagem anterior vimos um exemplo de uma UI, com sua lista de componentes e comandos. No caso havia apenas um componente na tela (um *STRING_ITEM*), configurado com suas propriedades; e também apenas um comando, que direcionava a aplicação a outra tela.

Nos capítulos seguintes serão detalhadas cada parte de uma UI: seus componentes, suas propriedades, e seus comandos.

Lembrando que este manual é apenas para os componentes básicos e que o *framework* pode ser expandido com novos componentes, propriedades e comandos.

7.2.1 Componentes de uma Tela.

A lista de componentes é representada pela *tag* **<componentlist>**, sem atributos, e dentro desta segue a listagem dos componentes, cada um identificado pela *tag* **<Component id="componentID" type="componentType">**. A regra para formação do ID segue o mesmo padrão mencionado acima (alfanumérico e sem espaços, com o cuidado de ser único em toda a aplicação). O tipo do componente depende da lista suportada; inicialmente foram desenvolvidos apenas cinco tipos básicos, mas a arquitetura do *framework* permite que esse número seja facilmente expandido. O tipos atualmente suportados são:

1. STRING_ITEM – campo de texto, sem edição (cabeçalho e texto).
2. TEXT_FIELD – campo de texto para edição (cabeçalho e texto).
3. IMAGE_ITEM – imagem (pode ter um cabeçalho).
4. DATE_FIELD – campo para edição de datas (abre uma tela especial para edição).
5. CHOICE_GROUP – lista de seleção (pode ser múltipla ou exclusiva).

7.2.1.1 STRING_ITEM

O componente de texto tem cinco campos de propriedades (*sub-tags*):

- Label – define o título inicial (quando o aplicativo inicia) do campo. [opcional, valor padrão: ***“”***]
- Text – define o texto inicial do campo. [opcional, valor padrão: ***“”***]
- Font – define a fonte do componente. [opcional, valor padrão: ***padrão do sistema***]
- Layout – define a disposição do componente na tela. [opcional, valor padrão: ***LAYOUT_DEFAULT***]
- AppearanceMode – define o estilo do componente. [opcional, valor padrão: ***PLAIN***]

7.2.1.2 TEXT_FIELD

O componente para edição de texto tem cinco propriedades (*sub-tags*):

- Label – define o título inicial (quando o aplicativo inicia) do campo. [opcional, valor padrão: ***“”***]
- Text – define o texto inicial do campo. [opcional, valor padrão: ***“”***]
- MaxSize – define o número máximo de caracteres do campo, quantidade maior o conteúdo será truncado. [opcional, valor padrão: ***100***]

- Constraints – define restrições de formato de caracteres. [opcional, valor padrão: ***ANY***]
- Layout – define a disposição do componente na tela. [opcional, valor padrão: ***LAYOUT_DEFAULT***]

7.2.1.3 IMAGE_ITEM

O componente de imagem também tem cinco propriedades:

- Label – define o cabeçalho inicial da imagem (quando o aplicativo inicia). [opcional, valor padrão: ***“”***]
- Image – define a URL da imagem. [opcional, valor padrão: ***“”, sem imagem***]
- Layout – define a disposição do componente na tela. [opcional, valor padrão: ***LAYOUT_DEFAULT***]
- AlternateText – define o texto alternativo, que será exibido caso a imagem não possa ser carregada ou exibida. [opcional, valor padrão: ***“”***]
- AppearanceMode – define o estilo do componente. [opcional, valor padrão: ***PLAIN***]

7.2.1.4 DATE_FIELD

O campo para edição de datas tem apenas quatro propriedades para configuração:

- Label – define o cabeçalho inicial da imagem (quando o aplicativo inicia). [opcional, valor padrão: ***“”***]
- InputMode – define o tipo de data que será editada (data-hora, data ou hora). [opcional, valor padrão: **DATE_TIME**]
- DateTime – define o valor inicial do campo de data. [opcional, valor padrão: **hora atual**]
- Layout – define a disposição do componente na tela. [opcional, valor padrão: ***LAYOUT_DEFAULT***]

7.2.1.5 CHOICE_GROUP

Este componente auxilia a edição/exibição de uma lista de itens para serem marcados ou não. Para sua configuração existem sete propriedades.

- Label – define o cabeçalho inicial da imagem (quando o aplicativo inicia). [opcional, valor padrão: ***“”***]
- ChoiceType – define o estilo da escolha: exclusiva, múltipla ou *popup*. [opcional, valor padrão: **EXCLUSIVE**]
- Items – define os itens para optar na escolha. [opcional, valor padrão: **“”**, sem itens]
- Selected – define os itens que estão selecionados inicialmente. [opcional, valor padrão: **“”**, nenhum item selecionado]
- Font – define a fonte do componente. [opcional, valor padrão: ***padrão do sistema***]

- FitPolicy – define se os itens serão divididos em *n* linhas ou truncados, caso o tamanho do texto seja maior que a largura da tela para exibição. [opcional, valor padrão: **TEXT_WRAP_DEFAULT**]
- Layout – define a disposição do componente na tela. [opcional, valor padrão: ***LAYOUT_DEFAULT***]

7.2.2 Propriedades dos Componentes

Neste capítulo serão descritas cada uma das propriedades dos componentes mencionados acima. Serão mostrados inclusive exemplos, e informações sobre comportamento, restrições e regras de formato; além de ressalvas que se façam pertinentes.

7.2.2.1 Fonte

A fonte é definada por **três** opções, na seguinte ordem: fonte, estilo e tamanho. Sendo **obrigatória** uma propriedade de cada. Sem deixar espaços entre as opções.

- Fontes: FACE_SYSTEM, FACE_MONOSPACE ou FACE_PROPORTIONAL.
- Estilos: STYLE_PLAIN, ou uma combinação de STYLE_BOLD, STYLE_ITALIC, STYLE_UNDERLINED.
- Tamanhos: SIZE_SMALL, SIZE_MEDIUM ou SIZE_LARGE.

Exemplos:

<Font>FACE_SYSTEM,STYLE_PLAIN,SIZE_SMALL**</Font>**

<Font>FACE_PROPORTIONAL,STYLE_BOLD,STYLE_UNDERLINED,SIZE_MEDIUM**</Font>**

7.2.2.2 Layout

O layout é definido da seguinte forma:

Na primeira posição deve ser escolhido entre: LAYOUT_DEFAULT ou LAYOUT_2. Se for optado pelo padrão, nenhuma opção posterior será considerada, caso contrário existem mais configurações obrigatórias para o segundo modo.

No caso do LAYOUT_2, ainda temos os seguintes campos exclusivos (deve-se ter apenas **uma** marcação):

- Disposição Horizontal: LAYOUT_LEFT, LAYOUT_CENTER, LAYOUT_RIGHT ou NONE.
- Disposição Vertical: LAYOUT_BOTTOM, LAYOUT_VCENTER, LAYOUT_TOP ou NONE.

E finalmente, um último campo de marcação múltipla:

- Espaçamento: LAYOUT_NEWLINE_BEFORE, LAYOUT_NEWLINE_AFTER, LAYOUT_SHRINK, LAYOUT_VSHRINK, LAYOUT_EXPAND e LAYOUT_VEXPAND.

Exemplos:

<Layout>LAYOUT_DEFAULT**</Layout>**

<Layout>LAYOUT_2,LAYOUT_CENTER,NONE,
LAYOUT_NEWLINE_BEFORE,LAYOUT_NEWLINE_AFTER,**</Layout>**

7.2.2.3 AppearanceMode

O estilo do componente diz respeito a como o componente se comportará, como um botão, um *hyperlink* ou seu formato padrão (sem ações). Deve-se escolher uma das três opções para esta propriedade.

- Aparência: PLAIN, HYPERLINK ou BUTTON.

Exemplos:

<AppearanceMode>PLAIN**</AppearanceMode>**

<AppearanceMode>BUTTON**</AppearanceMode>**

7.2.2.4 Constraints

A restrição de caracteres funciona como um filtro para facilitar a entrada de dados em campos onde estes dados devem respeitar alguma regra, por exemplo: um campo para CEP só deve aceitar números, um campo para nomes só deve aceitar caracteres, etc.

O formatado da restrição é dividido em dois grupos, o primeiro é obrigatório e deve-se optar por um valor; já o segundo grupo é opcional e pode-se optar por *n* valores, que serão somados e aplicados.

Grupo obrigatório (exclusivo):

- ANY, EMAILADDR, NUMERIC, PHONENUMBER, URL ou DECIMAL.

Grupo opcional (múltiplo):

- PASSWORD, UNEDITABLE, SENSITIVE, NON_PREDICTIVE, INITIAL_CAPS_SENTENCE, INITIAL_CAPS_WORD.

Todas as opções são autoexplicativas, exceto duas que vou explicitar abaixo:

- SENSITIVE – Indica que o texto do campo é uma informação sensível (de segurança) e a implementação interna do componente não deve armazenar este dado em nenhuma estrutura interna de predição de digitação, *auto-complete*, etc.
- NON_PREDICTIVE – Indica que o texto do campo não é uma palavra comum, que pode ser usualmente encontrada nos dicionários dos sistemas de predição de digitação.

Exemplos:

<**Constraints**>ANY</**Constraints**>

<**Constraints**>ANY,PASSWORD,SENSITIVE</**Constraints**>

<**Constraints**>NUMERIC</**Constraints**>

7.2.2.5 Image

A propriedade *image*, indica a URL para se buscar a imagem a ser exibida. Deve-se tomar o cuidado de certificar-se de sempre ter uma URL válida (atualmente não existe tratamento deste erro).

Exemplos:

<**Image**>http://127.0.0.1/imagens/sun.gif</**Image**>

<**Image**>http://www.google.com.br/intl/en_com/images/srpr/logo1w.png</**Image**>

<**Image**>http://127.0.0.1:8080/imagens/javalogo.jpg</**Image**>

7.2.2.6 AlternateText

Esta propriedade configura o texto alternativo a ser exibido enquanto o arquivo da imagem é baixado, e também no caso de a imagem não poder ser baixada ou exibida.

Exemplos:

<**AlternateText**>image NOT loaded!</**AlternateText**>

<**AlternateText**>Google logo.</**AlternateText**>

7.2.2.7 InputMode

Aqui é definido o tipo de data que será editada: data, hora, ou data e hora.

Os valores para a propriedade são: DATE, TIME ou DATE_TIME.

Exemplos:

<**InputMode**>DATE_TIME</**InputMode**>

<**InputMode**>DATE</**InputMode**>

7.2.2.8 DateTime

Define o valor inicial (e atual em caso de atualização) do campo de edição de data.

O formato do campo varia de acordo com a propriedade *InputMode* do componente.

Exemplos:

- InputMode = DATE_TIME:
  <DateTime>11:12:13,14/10/2015**</DateTime>**
- InputMode = DATE:
  <DateTime>14/10/2015**</DateTime>**
- InputMode = TIME:
  <DateTime>11:12:13**</DateTime>**

7.2.2.9 ChoiceType

Esta propriedade define o tipo de escolha que deverá ser feita no componente, como: exclusiva (somente uma opção possível), múltipla (várias escolhas são possíveis) e *popup* que é semelhante à primeira (a diferença é que aqui só fica visível a opção selecionada, e ao abrir o *popup* as outras são mostradas, enquanto na primeira todas as opções estão sempre visíveis).

As opções na linguagem são: EXCLUSIVE, MULTIPLE ou POPUP.

Exemplos:

<**ChoiceType**>EXCLUSIVE</**ChoiceType**>

<**ChoiceType**>MULTIPLE</**ChoiceType**>

7.2.2.10 Items

Aqui a propriedade configura os itens que serão mostrados como opções em um componente *ChoiceGroup*. Cada item é um par *string-imagem*, e podendo cada um destes ser vazio, ou seja, posso ter somente itens com texto, somente com imagem ou ambos.

Deve-se apenas tomar o cuidado de se não for usar a parte de imagem, deixar um espaço em branco, pois o sistema sempre espera a posição da imagem na lista.

A imagem é adicionada através de uma URL válida.

Exemplos:

<**Items**>UM; ;DOIS; ;TRES; </**Items**>

<**Items**>UM;http://www.server.com.br/um.jpg;

DOIS;http://www.server.com.br/dois.jpg</**Items**>

<**Items**>UM; ;DOIS; ; ;http://www.server.com.br/tres.png</**Items**>

No ultimo exemplo temos os dois primeiros itens com texto e sem imagem, e o último ao contrário, sem texto e apenas com imagem. Note os espaços nos itens.

7.2.2.11 Selected

Esta propriedade é responsável por marcar quais itens estarão marcados num componente *ChoiceGroup* quando o aplicativo é iniciado. Lembrando de respeitar o tipo de escolha, se exclusiva ou múltipla (vide a propriedade *ChoiceType*).

Os itens marcados são sinalizados por um **1**, e os não selecionados por um **0** na lista (separados por vírgulas).

Exemplos:

<**Selected**>0,1,0</**Selected**>

<**Selected**>0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0</**Selected**>

<**Selected**>0,1,0,0,0,0,1,1,0,0,0,1,0,0,0,0,1,0</**Selected**>

7.2.2.12 FitPolicy

A propriedade configura o modo preferencial que a aplicação vai arrumar os itens do componente *ChoiceGroup* na tela (caso do texto/imagem excederem o espaço disponível).

As opções são:

- TEXT_WRAP_DEFAULT – não define se o item deve ser dividido em *n* linhas ou truncado, deixando para o comportamento padrão da implementação da JVM usada.
- TEXT_WRAP_ON – define que o item deve ser dividido em *n* linhas.
- TEXT_WRAP_OFF – define que o item deve ser truncado.

Exemplos:

<**FitPolicy**>TEXT_WRAP_DEFAULT</**FitPolicy**>

<**FitPolicy**>TEXT_WRAP_ON</**FitPolicy**>

7.2.3 Comandos de uma Tela

Os comandos (*tag* <**Command**>) são *sub-tags* da *tag* <**Buttons**> e designam ações que a aplicação irá executar. Atualmente existem apenas quatro comandos disponíveis: sair da aplicação, enviar dados ao servidor do serviço, requisitar dados ao servidor do serviço e ir para outra tela da aplicação.

As *tags* para os comandos ficam assim:

- ACTION_EXIT – cria um comando para sair da aplicação DynamicML, retornando ao gerenciador de aplicativos.
- ACTION_SENDDATA – cria um comando para enviar dados ao servidor do serviço (atualizando o sistema).
- ACTION_RECEIVEDATA – cria um comando para requisitar dados do servidor, atualizando a visão da aplicação quanto ao estado do sistema servido.

- ACTION_GOTOUI – cria um comando para ativar outra tela do aplicativo. A tela a ser ativada é designada por uma

7.2.4 Notas importantes

Todos os componentes e propriedades da linguagem DynamicML apenas fazem uma adaptação de propriedades e componentes da linguagem Java, e que são dependentes da implementação usada.

Infelizmente houve certa liberdade de implementação por parte da *Sun* (hoje *Oracle*) no licenciamento da KVM, o que levou a certa fragmentação da base; ou seja, existem diferentes comportamentos possíveis para um mesmo componente ou ação. Por exemplo, a propriedade *FitPolicy* do componente *ChoiceGroup*, independente da opção marcada, a implementação pode ignorar e usar a configuração padrão da KVM.

Assim, existem outros tantos cuidados que devem ser tomados para que as aplicações fiquem realmente homogêneas independente da máquina virtual usada. O ideal é consultar esse tipo de informação da documentação Java oficial e/ou da KVM.