UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

UTILIZAÇÃO DO TERMINAL NOKIA 12 EM APLICAÇÕES M2M

DIOGO DUARTE D'ALESSANDRO

RELATÓRIO FINAL DE GRADUAÇÃO

ÁREA DE CONCENTRAÇÃO: ENGENHARIA ELÉTRICA
LINHA DE PESQUISA: TELECOMUNICAÇÕES
ORIENTADOR: PROF. DR. LEONARDO R. A. X. DE MENEZES
BRASÍLIA, DEZEMBRO DE 2003.

Dedico este trabalho aos meus pais, Ana e Nicolau,
A meus irmãos, André e Marcela e a minha avó Celina (In Memoriam).

AGRADECIMENTOS

Gostaria de agradecer primeiramente a Deus por todas as oportunidades que tive, à minha família que sempre esteve olhando por mim e me incentivando a cada novo desafio, ao professor Dr. Leonardo R.A.X. de Menezes e, finalmente, a todos os colegas que estiveram presentes durante minha formação acadêmica.

RESUMO

Este relatório é sobre comunicação M2M, que significa comunicação Máquina a Máquina. M2M é considerada a etapa seguinte na evolução das comunicações porque estende a conectividade além dos seres humanos. Permite a comunicação entre indivíduos, dispositivos e sistemas.

A intenção do M2M é criar soluções visando o melhoramento de negócios existentes, criar oportunidades totalmente novas, e essencialmente facilitar a vida diária. Por meio de soluções M2M, que são criadas freqüentemente para se encaixar às exigências específicas de cada situação, as companhias podem automatizar seus processos e integrar seus dispositivos remotos com seus sistemas de informação. O objetivo é aumentar o desempenho e competitividade da companhia através do aumento da eficiência, da economia de gastos, aumentar rendimentos ou melhorar níveis de serviço.

Esta tecnologia recente ainda não está disponível no Brasil, embora pareça uma maneira promissora de lucro para as operadoras de telecomunicações com sua rede atual de 2,5G e a rede 3G futura. Com um uso maciço de tecnologias como esta, sua rede de dados estaria em uso por muito mais tempo, consolidando-a como uma nova forma de rendimento, já que o mercado de voz está cada vez mais competitivo e saturado.

Nesse relatório será apresentado um exemplo de aplicação de gerenciamento de aparelhos domésticos como um Ar condicionado e um sistema de iluminação. Através da Plataforma M2M implementada um usuário comandará remotamente serviços em sua própria residência. Com isso será mostrado um pouco da potencialidade dessa tecnologia.

ABSTRACT

This report is about M2M communication, which stands for Machine-to Machine communication. M2M is considered the next step in the evolution of communications because extends connectivity beyond human beings. It allows communication between individuals, devices and systems.

The M2M business is about creating solutions aimed at improving existing business, creating totally new opportunities, and essentially making daily life easier. By means of M2M solutions, which are often tailored to meet the specific demands of each situation, companies can automate their processes and integrate their assets with their IT systems. The goal is to increase the performance and competitiveness of the company through increased efficiency, cost savings, additional revenues or better service levels.

This recent technology is yet to be available in Brazil, although it sounds like a promising way for the telecommunications operators profit with their 2.5 Generation network and future 3G network. With a massive use of technology like this, their Data network would be in use for much more time, consolidating it as a new revenue stream, since the voice market is so competitive and increasingly saturated.

In this report will be demonstrated, as a sample, a Home device management service, to manage devices such as an air conditioning system and an illumination system. A user will be able to remotely command devices at his home, not being there, through Nokia M2M Platform. After this simulation people will have an idea of all the possibilities offered by this technology.

ÍNDICE

LISTA DE FIGURAS

LISTA DE ABREVIAÇÕES

Abreviação	Significado
8-PSK	8 level Phase Shift Keying
ANSI	American National Standards Institute
API	Applications Programming Interface
AWT	Application Window Toolkit
BSC	Base Station Controller
BTS	Base Transceiver Station
CDC	Connected Device Configuration
CEPT	Conference Europeene des Pôstes et Telecommunications
CLDC	Connected Limited Device Configuration
CORBA	Common Object Request Broker Architecture- Arquitetura do Mediador de Requisições de Objetos Comuns
CSD	Circuit Switched Data
EDGE	Enhanced Data Rates for GSM Evolution
ETSI	European Telecommunications Standards Institute
GGSN	Gateway GPRS Support Node
GIOP	General Inter-ORB Protocol
GMSK	Gaussian Minimum Shift Keying
GPRS	General Packet Radio Service
GPS	Global Positioning System
GSM	Global System for Mobile communication
GSN	GPRS support Node
GUI	Graphical User Interface
HSCSD	High Speed Circuit Switched Data
HTTP	Hypertext Transfer Protocol
IDL	Interface Definition Language
IIOP	Internet Inter-ORB Protocol
IMlet	Software da Aplicação
IP	Internet Protocol
ISDN	Integrated Services Digital Network
JAD	Java Archive Descriptor
JAR	Java Archive
JFC	Java Foundation Classes
J2ME	Java 2 Micro Edition
J2SE	Java 2 Standard Edition
JRE	Java Runtime Environment
JVM	Java Virtual Machine
M2M	Machine-to-Machine
MAC	Medium Access control
MS	Mobile Station
MSC	Mobile Switching Centre
MSID	Mobile Station Identifier

MSISDN	Mobile Station International ISDN Number
ORB	Object Request Broker - Mediador de Requisições de Objetos
OTA	Over The Air
PDN	Packet Data Network
PLMN	Public Land Mobile Network
RLC	Radio Link Control
RMI	Remote Method Invocation
SDK	Software Development Kit
SGSN	Serving GPRS Support Node
SIM	Subscriber Identification Module
SMS	Short Messaging Service
TCP	Transfer Control Protocol
UMTS	Universal Mobile Telecom. System
URL	Uniform Resource Locator
UWC CONSORTIUM	Universe Wireless Communication
WAP	Wireless Application Protocol
WTK	Wireless Tool Kit
WTP	Wireless Transaction Protocol

1. Introdução

Essa monografia nasceu da vontade de se criar soluções de pagamento automático via celular. Ao estudar as diversas possibilidades de implementação para tal serviço, chegou-se a existência de um conceito muito mais abrangente e excitante que a simples transferência de dinheiro, o M2M.

M2M se trata de um conceito de comunicação entre máquinas, celulares e sistemas.Através de comunicação Wireless, dispositivos remotos passam a integrar a rede de um usuário. Com isso toda a dificuldade de operação deste terminal (manutenção, programação, controle de estoque, reinicialização) passa a ser realizada remotamente via rede comum de celular. [5]

Essas novas possibilidades são extremamente interessantes para empresas. Utilizando um terminal M2M, como o Nokia 12, podem aumentar seu desempenho e competitividade através do aumento da eficiência, da economia de gastos, e ainda, aumentar rendimentos ou melhorar níveis de serviço. Como exemplo, pode-se analisar uma empresa que tenha máquinas de vendas. Utilizando um terminal M2M, não seria mais necessária a visita de funcionários para controlar o estoque e a quantia de dinheiro acumulada, para alterar preços e até reiniciar a maquia em caso de falta de luz. [5],[12]

Para compreender a fundo esse trabalho, faz-se necessária uma base teórica apresentada nos primeiros capítulos. Serão apresentados conceitos básicos a respeito das tecnologias Java, GSM, M2M e a respeito do terminal Nokia 12 e seu simulador, bem como a justificativa da escolha por cada um desses elementos componentes. Ao final, será apresentada uma simulação totalmente baseada em software demonstrando esses conceitos na prática.

2. Programação Orientada a Objeto [1]

Visando um entendimento amplo do leitor, inicia-se esse relatório pelo conceito de programação utilizado em Java. Isso devido ao emprego dessa linguagem tanto nos dispositivos simulados quanto nos reais.

Programação orientada a objetos (POO) é uma metodologia de programação adequada ao desenvolvimento de sistemas de grande porte, provendo modularidade e reusabilidade. A POO introduz uma abordagem na qual o programador visualiza seu programa em execução como uma coleção de objetos cooperativos que se comunicam através de mensagens. Cada um dos objetos é instância de uma classe e todas as classes formam uma hierarquia de classes unidas via relacionamento de herança. Existem alguns aspectos importantes na definição de POO:

- Usa *objetos*, e não funções ou procedimentos como seu bloco lógico fundamental de construção de programas;
- Objetos comunicam-se através de *mensagens;*
- Cada objeto é instância de uma *classe*;
- Classes estão relacionadas umas as outras via mecanismos de *herança*.

Programação orientada a objetos dá ênfase à estrutura de dados, adicionando funcionalidade e capacidade de processamento a estas estruturas. Em linguagens tradicionais, a importância maior é atribuída a processos e sua implementação em subprogramas. Em linguagens orientadas a objetos, ao invés de passar dados a procedimentos, requisita-se que objetos realizem operações neles próprios.

Alguns dos aspectos fundamentais na definição de programação orientada a objetos são apresentados a seguir.

2.1 Objetos

Na visão de uma linguagem imperativa tradicional (estruturada), os *objetos* aparecem como uma única entidade autônoma que combina a representação da informação (estruturas de dados) e sua manipulação (procedimentos), uma vez que possuem capacidade de processamento e armazenam um estado local. Pode-se dizer que um objeto é composto de:

- Propriedades - são informações que representam o estado interno do objeto. Em geral, não são acessíveis aos demais objetos.

- Comportamento - conjunto de operações (métodos), que agem sobre as propriedades. Os métodos são ativados (disparados) quando o objeto recebe uma mensagem solicitando sua execução. Embora não seja obrigatório, em geral, uma mensagem recebe o mesmo nome do método que ela dispara. O conjunto de mensagens que um objeto está apto a receber está definido na sua interface.

- Identidade - é uma propriedade que diferencia um objeto de outro; ou seja, seu nome.

Enquanto conceitos de dados e procedimentos são tratados separadamente nas linguagens de programação tradicionais, em POO eles são reunidos em uma única entidade: o objeto. A Figura 01 representa um objeto com seus métodos e atributos.

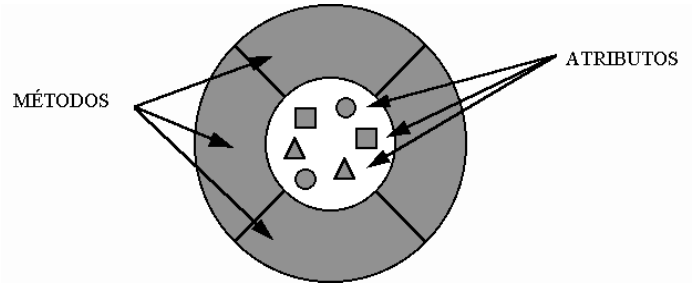

Figura 01: Representação de um objeto

No mundo real não é difícil a identificação de objetos (em termos de sistemas, objetos são todas as entidades que podem ser modeladas, não se restringindo apenas a objetos inanimados).

Uma vez que objetos utilizam o princípio da abstração de dados, o encapsulamento de informação proporciona dois benefícios principais para o desenvolvimento de sistemas:

- Modularidade: o código fonte de um objeto pode ser escrito e mantido independentemente do código fonte de outros objetos. Além disso, um objeto pode ser facilmente migrado para outros sistemas.

- Ocultamento de informação: um objeto tem uma interface pública para comunicar-se com outros objetos. Mas também mantém informações e métodos privados que podem ser alterados a qualquer hora, sem afetar os outros objetos que dele dependem. Ou seja, não é necessário saber como o objeto é implementado para poder utilizá-lo.

2.2 Mensagens

Um objeto sozinho não é muito útil e, geralmente, aparece como um componente de um grande programa composto de muitos outros objetos. Através da interação destes objetos pode-se obter uma grande funcionalidade e comportamentos mais complexos.

Objetos de software interagem entre si através de mensagens. Quando o objeto A deseja a execução de um método do objeto B, ele lhe envia uma mensagem. Caso o objeto B utilize alguma informação adicional do programa, esta é transmitida juntamente com a mensagem através de *parâmetros.*

Uma mensagem é formada por três componentes básicos:

- O objeto a quem a mensagem é endereçada (receptor);

- O nome do método que se deseja executar;

- Os parâmetros (se existirem) necessários ao método.

2.3 Classe

É a definição dos atributos e funções de um tipo de objeto. Cada objeto individual é então criado com base no que está definido na classe. Por exemplo, *homo sapiens* é uma classe de mamífero; cada ser humano é um objeto dessa classe.

Cada objeto criado a partir de uma classe é denominado de *instância* dessa classe. Uma classe provê toda a informação necessária para construir e utilizar (um ou mais) objetos de um tipo. Devido ao fato de todas as instâncias de uma classe compartilharem as mesmas operações, qualquer diferença de resposta a mensagens aceitas por elas é determinada pelos valores das variáveis da instância.

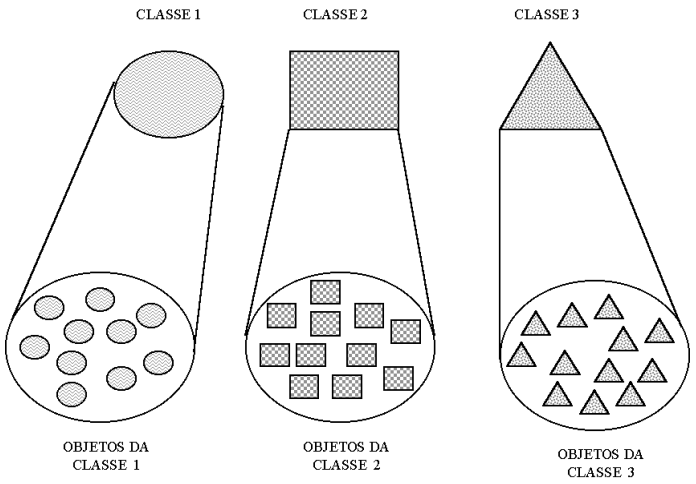

Figura 02: Relacionamento entre classes e objetos

A Figura 02 ilustra o relacionamento entre classes e objetos. Cada objeto instanciado a partir de uma classe possui propriedades e comportamentos definidos na classe, da mesma maneira que uma variável incorpora as características do seu tipo. A existência de classes proporciona um ganho em reusabilidade. Pois cada vez que um novo objeto é instanciado ou que uma mensagem é enviada, a definição da classe é reutilizada. Caso não existissem classes, para cada novo objeto criado, seria preciso uma definição completa do objeto.

2.4 Metaclasses

Uma metaclasse é uma classe de classes. Pode-se julgar conveniente que, em uma linguagem ou ambiente, classes também possam ser manipuladas como objetos. Por exemplo, uma classe pode conter variáveis contendo informações úteis, como:

- O número de objetos que tenham sido instanciados da classe até certo instante;

- Um valor médio de determinada propriedade, calculado sobre os valores específicos desta propriedade nas instâncias (por exemplo, média de idade de empregados).

2.5 Métodos

Um método implementa algum aspecto do comportamento do objeto. Comportamento é a forma como um objeto age e reage, em termos das suas trocas de estado e de mensagens.

Um método é uma função, ou procedimento, definido na classe. Tipicamente, pode acessar o estado interno de um objeto da classe para realizar alguma operação. Pode ser pensado como sendo um procedimento cujo primeiro parâmetro é o objeto no qual deve trabalhar. Este objeto é chamado receptor.

Construtores são usados para criar e inicializar objetos novos. Tipicamente, a inicialização é baseada em valores passados como parâmetros para o construtor. Destruidores são usados para destruir objetos. Quando um destruidor é invocado, as ações definidas pelo usuário são executadas, e então a área de memória alocada para o objeto é liberada. Em algumas linguagens, como C++, o construtor é chamado automaticamente quando um objeto é declarado. Em outras, como Object Pascal, é necessário chamar explicitamente o construtor antes de poder utilizá-lo.

Um exemplo de utilização de construtores e destruidores seria gerenciar a quantidade de objetos de uma determinada classe que já foram criados até o momento. No construtor pode-se colocar código para incrementar uma variável e no destruidor o código para decrementá-la.

2.6 Herança

O conceito de herança é fundamental na técnica de orientação a objetos. A herança permite criar um novo tipo de objeto - uma nova classe - a partir de outra já existente.

A nova classe mantém os atributos e a funcionalidade da classe da qual deriva; por isso, diz-se que ela "herda" as características daquela classe. Ao mesmo tempo, ela pode receber atributos e funções especiais não encontrados na classe original.

Uma das vantagens da herança é a facilidade de localizar erros de programação. Por exemplo, caso um objeto derivado de outro apresente um erro de funcionamento; se o objeto original funcionava corretamente, é claro que o erro está na parte do código que implementa as novas características do objeto derivado. A herança permite, também, reaproveitar o código escrito anteriormente, adaptando-o às novas necessidades.

Isso é muito importante porque os custos de desenvolvimento de software são muitos elevados. A mão-de-obra altamente especializada é cara; o processo é demorado e sujeito a ocorrências inesperadas.

2.7 Polimorfismo

Polimorfismo refere-se à capacidade de dois ou mais objetos responderem à mesma mensagem, cada um a seu próprio modo. A utilização da herança torna-se fácil com o polimorfismo. Desde que não seja necessário escrever um método com nome diferente para responder a cada mensagem, o código é mais fácil de entender.

Outra forma simples de polimorfismo permite a existência de vários métodos com o mesmo nome, definidos na mesma classe, que se diferenciam pelo tipo ou número de parâmetros suportados. Isto é conhecido como *polimorfismo paramétrico*, ou *sobrecarga de operadores* ("overloading"). Neste caso, uma mensagem poderia ser enviada a um objeto com parâmetros de tipos diferentes (uma vez inteiro, outra real, por exemplo), ou com número variável de parâmetros. O nome da mensagem seria o mesmo, porém, o método invocado seria escolhido de acordo com os parâmetros enviados.

Alguns benefícios proporcionados pelo polimorfismo:

- Legibilidade do código: a utilização do mesmo nome de método para vários objetos torna o código de mais fácil leitura e assimilação, facilitando muito a expansão e manutenção dos sistemas.

- Código de menor tamanho: o código mais claro torna-se também mais enxuto e elegante. Pode-se resolver os mesmos problemas da programação convencional com um código de tamanho reduzido.

Vantagens da POO

A POO tem alcançado muita popularidade, devido às vantagens que ela traz. Dessas, a reusabilidade de código é, sem dúvida, a maior vantagem, pois permite que programas sejam escritos mais rapidamente. Com isso, empresas conseguem melhorar seus serviços e aumentar sua agilidade, herdando códigos já existentes para desenvolver novos sistemas rapidamente.

Escalabilidade é a capacidade de uma aplicação executar mais funções sem aumentar demasiadamente a sua complexidade ou comprometer o seu desempenho. A POO é adequada ao desenvolvimento de sistemas. Afinal, um sistema é composto por objetos agrupados interagindo entre si.

O encapsulamento proporciona ocultamento e proteção da informação. Acessos a objetos somente podem ser realizados através das mensagens que ele está habilitado a receber. Nenhum objeto pode manipular diretamente o estado interno de outro. Assim, caso haja necessidade de se alterar as propriedades de um objeto ou sua implementação, o restante do sistema não será afetado, desde que a interface permaneça idêntica. Isto diminui em grande parte os esforços despendidos em manutenção. Além disso, para utilizar um objeto, o programador não necessita conhecer a fundo sua implementação.

O polimorfismo torna o programa mais enxuto, claro e fácil de compreender. Sem ele, seriam necessárias listas enormes de métodos com nomes

diferentes, mas comportamento similar. Em termos de manutenção, isto significa que o programa será mais facilmente entendido e alterado.

A herança também torna a manutenção mais fácil. Se uma aplicação precisa de alguma funcionalidade adicional, não é necessário alterar o código atual. Simplesmente cria-se uma nova geração da classe, herdando o comportamento antigo, e adiciona-se a nova característica.

Desvantagens da POO

Apesar das inúmeras vantagens, a POO também tem algumas desvantagens. A apropriação é apresentada tanto como uma vantagem como uma desvantagem, porque nem sempre soluciona os problemas elegantemente. Além disso, A POO requer definições precisas de classes; definições flexíveis e imprecisas não são suportadas. Na mente humana, essas classificações podem mudar com o tempo. Os critérios para classificar objetos podem mudar significativamente. A apropriação utilizada na POO torna-a muito rígida para trabalhar com situações dinâmicas e imprecisas.

Além disso, algumas vezes não é possível decompor problemas do mundo real em uma hierarquia de classes. Negócios e pessoas têm freqüentemente regras de operações sobre objetos que desafiam uma hierarquia limpa e uma decomposição orientada a objetos. O paradigma de objetos não trata bem de problemas que requerem limites nebulosos e regras dinâmicas para a classificação de objetos.

Isto leva ao próximo problema com POO: fragilidade. Desde que uma hierarquia orientada a objetos requer definições precisas, se os relacionamentos fundamentais entre as classes-chave mudam, o projeto original orientado a objetos é perdido. Torna-se necessário reanalisar os relacionamentos entre os objetos principais e reprojetar uma nova hierarquia de classes. Se existir uma falha fundamental na hierarquia de classes, o problema não é facilmente consertado.

3. Linguagem Java [1]

3.1 Java

Após o primeiro contato com POO, o leitor está apto a ir mais a fundo nos aspectos que caracterizam a linguagem Java. Aqui serão apresentados alguns dos aspectos que serão utilizados na programação dos terminais, que tornam Java a linguagem mais apropriada para a Plataforma M2M.

Java é uma linguagem orientada a objeto recente. Ela foi idealizada no ano de 1991 e a primeira versão pública foi lançada em março de 1995. Isso tornou possível analisar os defeitos das outras linguagens anteriores e assimilar as vantagens das mesmas.Por causa da grande quantidade de programadores C e C++, todos os operadores lógicos, aritméticos foram mantidos da mesma forma em Java. Mas as declarações de baixo nível, como ponteiros, foram abandonadas por causar vários problemas de desenvolvimento. Outra causa de dificuldades no desenvolvimento em C era o gerenciamento manual da memória com os comandos `malloc`, e `free`. Em Java esse problema foi resolvido com a figura do *garbage collection* que desaloca a memória de objeto quando ele não é mais necessário.O modelo de gerenciamento de memória do Java é baseado em referências aos objetos.Quando não existe mais nenhuma referência a um objeto no gerenciador de memória, automaticamente o *garbage collection* desaloca a memória do objeto.

3.2 Plataforma Java

No cenário atual, existem várias plataformas de desenvolvimento como Windows, Unix e Macintosh. Um software desenvolvido para um desses sistemas operacionais (SO) exige adaptações e recompilação do código fonte para trocar para outro SO. Isso é um grave problema dentro do cenário heterogêneo que é a Internet, onde essas várias plataformas coexistem. A plataforma Java desenvolveu uma forma de resolver esse problema. Basicamente, ela interpreta os códigos binários feitos para uma máquina virtual executada por estes sistemas. Isso torna a plataforma neutra, executável em qualquer

sistema operacional, sem necessidade de modificação no código. Esse novo ambiente pode ser dividido em duas partes principais.

- Máquina Virtual Java (JVM) – Uma máquina virtual que é emulada nos processadores atuais ou implementada em um hardware específico.

- Interface de programação de aplicação Java (Java API)- Interface padrão para o desenvolvimento de aplicações para esta linguagem.

A Figura 03 mostra como essas camadas se organizam e alguns aplicativos que podem ser feitos em Java.

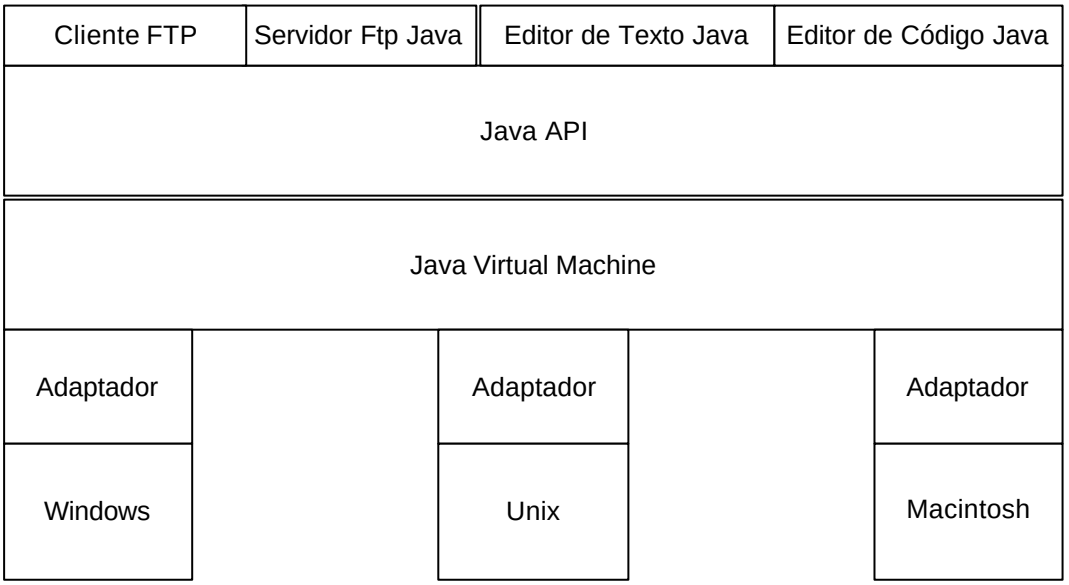

Figura 03: Plataforma Java

3.3 Máquina Virtual Java

A máquina virtual Java é a chave para a independência dos códigos binários. Nessa linguagem eles têm a extensão class. As instruções são transformadas em códigos binários e executados numa plataforma virtual. Os métodos dentro de uma classe são indexados por nome. Isso torna possível, em tempo de execução, que um novo método possa ser inserido ou cancelado.

3.4 Java APIs

A interface de programação de aplicativos (API, *Application Programming Interface*) Java apresenta inúmeras classes já implementadas, como as JFCs. Isso facilita ao programador na construção de seu aplicativo. JFC corresponde à abreviatura de *Java Foundation Classes*, que abrange uma série de atributos para ajudar na construção de interfaces gráficas, as GUIs (*graphical user interfaces*). O JFC foi lançado em 1997 na conferência de desenvolvedores *JavaOne* e contém os seguintes pacotes:

- Componentes *Swing*: Inclui todos os componentes visuais desde botões a janelas.

- Suporte a *Look and Feel*: Oferece a qualquer programa que utilize componentes Swing à escolha de sua aparência.

- API acessíveis: Permite opções de acessibilidade para leitores viva-voz e dispositivos de impressão em Braile para fornecer informações da interface do usuário.

- Java 2D (Java 2 somente): Permite a desenvolvedores a incorporarem facilmente gráficos 2D de alta qualidade, textos e imagens em aplicações.

- Suporte a *Drag and Drop* (Java 2 somente): Fornece a operação de “arrastar e soltar” em aplicações Java e entre estas e outras aplicações nativas.

Nesta monografia serão abordados os componentes *Swing* e algumas APIs. Antes, porém, serão abordados temas relativos à manipulação de eventos e tratamento de exceções, que são se importância fundamental em programas com GUI. E, para finalizar a abordagem relativa à linguagem Java, apresenta-se o conceito de fluxo de dados e arquivos.

3.5 Manipulação de eventos

Um sistema operacional precisa, constantemente, monitorar eventos como teclas pressionadas ou cliques do mouse. Ao receber um comando, ele informa aos programas que estão em execução, e então cada programa decide o que fazer em resposta a

esses eventos. A linguagem Java adota uma metodologia em termos de recursos e, conseqüentemente, na complexidade resultante. Dentro dos limites dos eventos que o pacote AWT (*Abstract Window Toolkit* – kit de ferramentas de janelas abstratas) conhece, pode-se controlar totalmente a maneira como esses são transmitidos, desde suas origens (como botões e barras de rolagem) até a determinação de quem serão os ouvintes dos eventos.

Pode-se designar qualquer objeto para ser um ouvinte de evento. Na prática, escolhe-se um objeto que possa efetuar convenientemente a resposta desejada. As origens dos eventos têm métodos que permitem registrar ouvintes de eventos neles. Quando um evento ocorre na origem, esta envia uma notificação do mesmo para todos os objetos ouvintes registrados no seu método. Então, a informação sobre o evento é encapsulada em um objeto *evento*. Em Java, todos os objetos *evento* derivam da classe `Java.util.EventObject`. Abaixo estão alguns exemplos de classes ouvintes e suas respectivas ações que resultam num evento:

- *ActionListener*: Usuário clica num botão, pressiona a tecla *Return* enquanto está digitando em um campo de texto, ou seleciona algum item de menu.

- *WindowListener*: Usuário fecha uma janela principal.

- *MouseListener*: Usuário pressiona um botão de mouse enquanto o cursor está sobre um componente.

- *MouseMotionListener*: Usuário move o mouse sobre um componente.

- *ComponentListener*: Componente se torna visível.

- *FocusListener*: Componente recebe foco do teclado.

- *ListSelectionListener*: Tabela ou lista alteram seu conteúdo.

3.6 Tratamento de Exceções

Examinam-se agora os mecanismos Java para lidar com dados incorretos e programas com erros. Quando o programa encontra erros durante sua execução, o ideal seria notificar o usuário do erro e permitir que fosse salvo todo trabalho, além de permitir a saída do programa de forma adequada. Para tanto, Java usa uma forma de captura de erros chamada, apropriadamente, de tratamento de exceções.

A reação tradicional a um erro num método é retornar um código de erro especial, que o método chamador possa analisar. Infelizmente, nem sempre isso é possível. Às vezes não há uma maneira óbvia de distinguir entre dados válidos e inválidos. A linguagem Java permite que todo método tenha uma saída alternativa, caso seja incapaz de finalizar sua tarefa normalmente. Nessa situação, o método não retorna um valor. Em vez disso, ele *lança* um objeto que encapsula a informação do erro. Então, o mecanismo de tratamento de exceções começa sua busca por um manipulador de exceção que possa lidar com essa condição de erro particular. Em Java, um objeto exceção é sempre instância de uma classe derivada de *Throwable*.

4. CORBA [1]

4.1 Introdução

Visando o emprego do M2M na indústria, faz-se necessário que essa tecnologia possa se adequar às plataformas já existentes nelas. Para tanto, a comunicação do terminal M2M com os terminais encarregados de comandá-lo se dá via CORBA. Como será detalhado adiante, CORBA é uma linguagem capaz de prover ao sistema a neutralidade que ele necessita, tornando a plataforma M2M extremamente flexível.

CORBA é a sigla para Arquitetura do Mediador de Requisições de Objetos Comuns (*Common Object Request Broker Architecture*), um padrão aberto feito pela OMG para trabalhar com objetos remotos através das redes. Para transmissão dos dados é utilizado o protocolo IIOP. Esse padrão é útil para integração de sistemas heterogêneos, pois é independente da linguagem de programação, processador ou mesmo do sistema operacional. A base da interoperabilidade e neutralidade do CORBA está baseada em três conceitos. O primeiro é que as interfaces usadas pelos programas são escritas em uma linguagem neutra denominada IDL, o segundo é que a implementação é separada da interface e o terceiro conceito é que o canal para comunicar a interface com a implementação denominado de Mediador de Requisições de Objetos (ORB) é padronizado.

4.2 IDL

Essa linguagem é usada para descrever as interfaces dos objetos que são chamados pelos clientes e implementados pelos servidores. A interface se limita a definir o nome do método, a entrada e saída do mesmo. A linguagem de descrição de interface é denominada IDL Para implementar os métodos, o código deve ser mapeado para alguma linguagem de programação. Atualmente, existe norma para as linguagens: C, C++, Java, Smalltalk, COBOL, Ada, Lisp, PL/1, Python, e IDLscript. A grande vantagem é que uma interface implementada em C pode ser lida por um cliente em COBOL, pois a interface é a mesma. Genericamente, nem o servidor nem o cliente precisam saber em que linguagem está escrita o outro lado da comunicação.

Implementação separada da interface

Essa separação permite que a implementação fique em um lugar e possa ser chamada em vários lugares pelas interfaces locais. Isso é a essência da interoperabilidade do sistema. Além disso, como o cliente só tem acesso à interface, o código da implementação fica protegido.Essa divisão é implementada por duas entidades (STUB e Esqueletos):

Stub – Código que o cliente usa para instanciar os objetos remotos e fazer a comunicação com o canal ORB.

Esqueleto - Código do servidor, implementa uma interface definida no stub e faz a comunicação com o canal ORB.

4.3 Canal ORB

O canal ORB é responsável em direcionar uma requisição do cliente para o objeto e encaminhar a resposta para o destino. No lado do cliente, o canal fornece as definições das interfaces. No lado do servidor, o canal é responsável por ativar os objetos requisitados e desativar os objetos que não estão sendo usado para poupar os recursos do servidor.

4.4 IIOP

O protocolo IIOP é uma implementação em TCP/IP do protocolo GIOP. Esse protocolo destina traduzir as mensagens entre os elementos da estrutura CORBA para a camada 4 e inferiores.

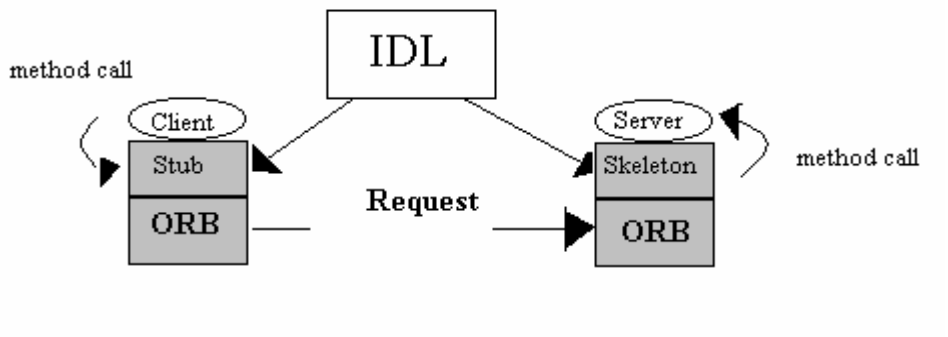

Figura 04: Arquitetura CORBA [21]

Na figura 4, os elementos em branco são definidos e implementados pelo desenvolvedor da aplicação. Após a definição da interface IDL o código do esqueleto e do Stub devem ser gerados, automaticamente ao se compilar a IDL. Os canais ORB são produtos disponibilizados gratuitamente ou comercialmente.

As setas nos blocos mostram que o código do cliente chamando um método no servidor somente enxerga uma chamada de método local, como seria o caso de um sistema não distribuído. O canal ORB junto com o Stub e o Esqueleto tornam essa distribuição transparente ao usuário comum.

Vantagens do CORBA

Sendo uma solução muito flexível e com poucas linhas de código, é possível enviar os objetos entre o cliente e o servidor. Aceita bem as evoluções de novas funcionalidades, pois exige apenas a criação de uma nova interface, e toda a estrutura de interconexão é aproveitada. Facilmente os algoritmos de simulação poderiam trocar de linguagem de programação que não exigiria nenhuma mudança do código.

5. Introdução à tecnologia GSM

Agora, tendo conhecimento do funcionamento lógico da rede, passa-se a estudar os princípios físicos da comunicação. A plataforma M2M é essencialmente GSM. Por tanto, a transmissão de dados se dá através da mesma rede utilizada para celulares, usando, inclusive, os recursos SMS, GPRS, HSCSD e EDGE. Nesse capítulo, serão explicados de maneira razoavelmente profunda o GSM, desde o seu surgimento até sua evolução a Geração 2,5, juntamente com os recursos que permitiram isso, e as tendências dessa tecnologia para o futuro.

5.1 Visão Geral Do GSM[19]

Em 1982, o principal corpo organizador das operadoras européias de telecomunicações, conhecido como CEPT (Conference Europeene dês Postes et Télécommunications), criou o comitê Groupe Spécial Móbile (GSM) e o encarregou de especificar um sistema de rádio celular Pan-europeu que operasse na banda de 900 MHz. O sistema foi concebido para superar as limitações de capacidade dos sistemas analógicos desenvolvidos em vários países europeus. O padrão celular Pan-europeu teria suporte ao roaming internacional e daria impulsão à indústria européia de telecomunicações.

Após discussões iniciais, três frentes de trabalho foram criadas para lidar com aspectos específicos da definição do sistema. Foi requerida a essas frentes que definissem interfaces de sistema que permitissem a uma unidade móvel circular pelos países que adotassem o novo sistema com acesso pleno aos serviços. Comparando com o sistema analógico existente, o novo sistema teria uma maior capacidade, custos operacionais inferiores e uma melhor qualidade de voz. Também foi exigido que esse novo sistema co-existisse com o sistema anterior. Concordou-se com a alocação da banda na faixa de 890-915 MHz e 935-950 MHz para o novo sistema, em todos esses países. Porém, em alguns deles parte dessa banda já estava alocada para sistemas analógicos , como no Reino Unido, limitando a banda inicial do GSM.

Embora estudos em vários países europeus tivessem concluído que sistemas digitais seriam preferidos a sistemas analógicos, a escolha do esquema de

múltiplo acesso não foi tão simples. Houve oito propostas distintas de sistema. O vencedor foi o sistema com maior eficiência espectral proposto pela ELAB. Por volta de junho de 1987 houve completo acordo de que o sistema deveria empregar TDMA de banda estreita e técnicas propostas pela ELAB. O sistema inicialmente suportaria oito canais por portadora com eventual evolução a 16 canais por portadora. A codificação escolhida foi GMSK (Gaussian Minimum Shift Keying) pela eficiência espectral demonstrada. A pedido da Inglaterra também foi desenvolvido um sistema GSM em 1800 MHz. Com a propagação da tecnologia, mais grupos de estudo surgiram. Dentre eles, um grupo destinado a especificar o UMTS (Universal Mobile Telecommunications System), sucessor do GSM. O termo GSM passou a significar "The Global System for Mobile Communication".

5.2.Evolução do GSM

5.2.1 Introdução[19]

O sistema GSM foi inicialmente projetado para suportar voz, assim como dados em baixa velocidade. A taxa de dados por usuário usando um único canal, ou seja, um único Timeslot por Frame TDMA, era inicialmente de 9.6 kbps. A máxima taxa disponível, foi aumentada a 14.4 kbps pela diminuição da potência da codificação do canal. Mas ainda assim era bastante lento. Outros meios de se aumentar mais significativamente essa taxa seriam a liberação de mais timeslots por usuário e a elevação do esquema de modulação.

Dois novos serviços para esse fim foram desenvolvidos, GPRS e HSCSD. O HSCSD permite que seja alocado para uma estação móvel (MS) um número de timeslots por Frame TDMA. Isso é, o terminal tem uso exclusivo dos recursos alocados pela duração de uma ligação. Em contraste, o GPRS usa conexões orientadas a pacotes na interface do rádio e dentro da rede. Ao usuário é concedido um, ou um número de canais de tráfego quando uma transferência de informação é requisitada. O canal é liberado uma vez que a transmissão se completa.

Uma segunda aproximação ao aumento da taxa de transferência de dados pelo emprego de esquemas de modulação de maior nível está atualmente sendo estudada

no projeto Enhanced Data Rates for GSM Evolution (EDGE). O princípio básico por trás do EDGE é que o esquema de modulação usado na interface do rádio GSM é escolhido com base na qualidade do link de rádio estabelecido. Uma modulação de maior nível é preferível quando o link é "bom". O sistema reverteria para um esquema de modulação de menor nível quando a qualidade do link caísse para "fraca". Em canais de baixa qualidade o esquema adotado é o GMSK enquanto que para os canais de boa qualidade é usado o 8-PSK. O EDGE também incluirá funções de adaptação de link que permitem ao terminal e a BS acessar a qualidade do link e chavear entre os diferentes tipos de modulação necessária.

A versão inicial do EDGE aumentará o alcance de serviços oferecidos pelo GPRS e HSCSD, elevando-os a EGPRS e ECSD.

Serão descritas a seguir, as tecnologias de transmissão de dados utilizadas pelo GSM.

5.2.2 Short Message Service [3]

O SMS é um serviço integrado de mensagem que permite aos usuários do GSM enviar e receber dados usando sua estação móvel. Uma mensagem SMS pode conter até 160 caracteres em seu corpo, podendo conter qualquer tipo de caractere. Mensagens SMS também podem ser baseadas em código binário.

O SMS é um serviço baseado em armazenamento/encaminhamento. Isso significa que as mensagens não são enviadas diretamente ao destinatário, mas via uma central SMS da rede. Isso possibilita que mensagens sejam entregues ao destinatário mesmo que sua estação móvel esteja desligada ou fora da área de cobertura no momento que a mensagem foi enviada. Ou seja, se trata de um serviço assíncrono como o e-mail. Outro aspecto desse serviço é a confirmação de recebimento, que seria uma mensagem retorno, notificando ao usuário remetente o recebimento ou não de sua mensagem. Em algumas circunstâncias múltiplas mensagens podem ser concatenadas.

Mensagens SMS podem ser enviadas e recebidas em qualquer aparelho TDMA. Outra utilidade do SMS é o recebimento das notícias mais atuais sobre esporte, loteria, política. Também pode ser usado para comunicação crítica e confidencial. Para a utilização desse serviço em aplicações M2M reais, deve haver garantias de QoS.

5.2.3 High Speed Circuit Switched Data [19]

O serviço HSCSD é uma extensão natural do serviço Circuit Switched Data (CSD) que era suportado nas versões anteriores do GSM. Na evolução para o HSCSD não houve necessidade de mudanças nas interfaces da camada física entre os diferentes elementos da rede. Nas camadas mais altas, a estação móvel e a rede suportam funcionalidade adicional para multiplexar e demultiplexar dados do usuário em um número de canais de tráfego para transmissão sobre a interface Abis e a interface do rádio. Funcionalidades adicionais também estão incluídas no nível de administração de recursos de rádio para suportar a nova situação onde mais de um canal está associado com a mesma conexão. Por exemplo, quando um usuário HSCSD faz o Handover entre das células, deve haver um mecanismo que garanta a existência de canais suficientes disponíveis na nova célula antes de efetuar o Handover. Uma conexão HSCSD é limitada a 64kbps na interface aérea.

Ao fazer o set up da ligação a estação móvel manda informação para a rede definindo a natureza da conexão HSCSD, como, por exemplo, o máximo número de Timeslots que podem ser acessados por ela, e a quantidade de tempo que deve ser permitida entre os timeslots, para o propósito de dimensionamento das células vizinhas. Mais ainda, são enviados também, a taxa de dados que a estação móvel gostaria de obter na rede, os esquemas de codificação de canal suportados por ela e o número máximo de canais de tráfego a serem usado durante a conexão. Esse último parâmetro permite ao usuário controlar o custo da ligação, ao limitar o número de canais de tráfego que serão ocupados Essa informação é usada para definir as capacidades da estação móvel em ambos HSCSD e GPRS.

Há dois tipos de estações móveis multislots. Elas podem ser tipo 1 ou tipo 2. Estações móveis tipo 2 são capazes de transmitir e receber dados simultaneamente, enquanto as estações tipo 1 só o fazem alternadamente.

Para a utilização desse serviço em aplicações M2M reais, deve haver garantias de QoS.

5.2.4 General Packet Radio Service [19]

Muitos serviços não requerem um fluxo bi-direcional contínuo de dados pela interface de rádio. Por exemplo, o download de uma página web tem um período de amplo uso até ser completada a transmissão da página inicial. Depois de concluído o download, a taxa de demanda de dados cairá enquanto a pessoa estiver lendo a pagina. Só depois serão requisitados dados de outra página. Usar conexões permanentes como CSD para serviços do tipo rajada, representa um uso ineficiente dos recursos do rádio. Afinal, um usuário estaria ocupando um canal pelo período de uma ligação, mesmo usando o canal por apenas frações desse tempo. Isso pode ser superado usando conexões orientadas a pacote.

Na segunda fase do sistema GSM foram incluídas provisões para o uso de serviços orientados a pacote, conhecido como General Packet Radio Service (GPRS). Essa tecnologia tenta otimizar os recursos da rede e do rádio. A alocação de canais GPRS é flexível, variando de um a oito timeslots em um frame TDMA, sendo que os timeslots de up-link e down-link são alocados separadamente. Os recursos da interface de rádio são capazes de ser compartilhados dinamicamente entre circuitos comutados e serviços a pacote, dependendo da preferência da operadora. A taxa de bits pode variar de 9kbps até 50 kbps por usuário. GPRS pode interagir com redes IP e X.25. Serviços ponto a ponto e multiponto são suportados, bem como o SMS (short message service). O GPRS é capaz de acomodar transmissões em rajada assim como transmissões contínuas.

Arquitetura Lógica do GPRS

Serviços GPRS requerem dois componentes adicionais na rede GSM, a saber, o Gateway GPRS Support Node(GGSN), e o Serving GPRS Support Node (SGSN). Como sugere seu nome o GGSN atua como Gateway entre redes externas de dados a pacote (PDN) e a rede GSM. O GGSN contém informações suficientes para rotear pacotes de dados de um SGSN que esteja servindo a uma estação móvel (MS) particular, e está conectada a redes externas pelo ponto de referência Gi (interface de rede estabelecida) (fig 05). O protocolo de rede Backbone do GPRS é o IP, que é usado para roteamento de dados de usuários e sinalização.

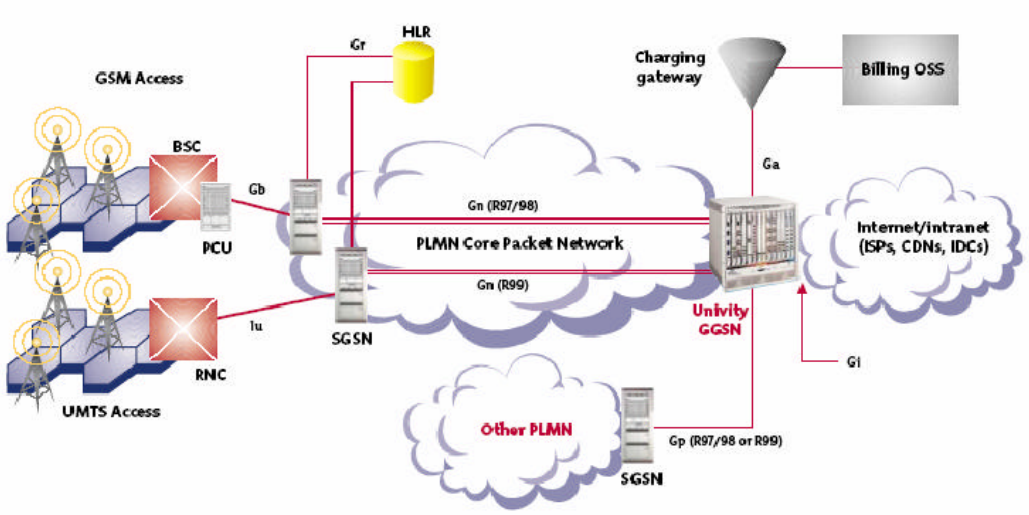

Figura 05: Implementação do GPRS na rede GSM, adicionando-se dois nós, GGSN e SGSN

Funções de Alto nível necessárias para o GPRS

Função de Controle de acesso à rede.

Um protocolo de acesso tem uma lista de procedimentos, como registro, autenticação e autorização, controle de admissão, cobrança, que permite a um usuário utilizar os serviços que a rede proporciona.

Acesso Anônimo

A idéia de acesso anônimo também foi introduzida no GPRS pelo qual uma estação móvel (MS) pode acessar a rede sem ser autenticada e sem exigir encriptação. Também não há exigências quanto a fornecer sua IMSI ou IMEI. Um exemplo de aplicação que poderia usar essa função é o pagamento automático de pedágio, onde um motorista usaria um cartão pré-pago, inserido em seu terminal, e automaticamente teria a cancela do pedágio liberada.

Roteamento de pacotes e funções de transferência

Uma rota consiste de um nó de rede originador, um nó de destino e,se necessário, outros grupos de nós. Roteamento é a transmissão de mensagens dentro e entre a PLMN (rede pública terrestre móvel). Um nó encaminha dados recebidos ao próximo nó usando a função de grupo. A função roteamento determina o nó suporte da rede GPRS (GSN) onde a mensagem é enviada usando o endereço do destinatário na mensagem. A função roteamento também seleciona o caminho a ser percorrido até o próximo GSN ao longo da rota.

Gerenciamento de mobilidade GPRS e estados da estação móvel (MS)

Gerenciamento de mobilidade descreve o processo no sistema móvel de rádio que estão associados ao rastreamento de movimentos de assinantes quando eles circulam pela rede. Os sistemas não GPRS utilizam área de localização e atualização dessas áreas para garantir que eles sempre saibam o paradeiro das estações. O sistema GPRS utiliza um conceito semelhante, porém usando áreas de roteamento. Se a MS detectar que entrou em uma nova área de roteamento, ela vai enviar uma atualização de área.

Os estados possíveis para a estação móvel são Idle, standby e ready. No idle o sistema está indisponível e, portanto, não faz atualização de Área de Roteamento(RA). No estado standby, a estação móvel está pronta para dispor de qualquer uma das suas funções. E quando está ativo o serviço a MS está no estado Ready.

Funções de gerenciamento de recursos de rádio

Essas funções provêm alocação e manutenção de canais de comunicação de rádio. Os recursos de rádio GSM são dinamicamente compartilhados entre CSD e GPRS. O gerenciamento de recursos de rádio GPRS se preocupa com a alocação e liberação de timeslots, monitoramento da utilização de canais GPRS, controle de congestionamento, e distribuição de informações de configuração de canais GPRS.

Funções de baixo nível para GPRS

As funções de interface de rádio para serviços GPRS são o Controle de Acesso de meio (MAC) e controle de link de rádio (RLC) que operam sobre a camada física. A função MAC arbitra entre as MSs tentando transmitir ao mesmo tempo. Portanto a MAC trabalha evitando colisões, detectando-as e com a recuperação apo colisão. Ela também é quem permite que uma estação móvel (MS) utilize vários canais simultaneamente.

Para a utilização desse serviço em aplicações M2M reais, deve haver garantias de QoS.

5.2.5 EDGE [19]

O objetivo do EDGE é aumentar as taxas de dados do GSM por meio de melhoramentos dos métodos de modulação; especificamente, aumentar a taxa de transmissão de dados por timeslot se comparada à modulação GMSK. Tipos diferentes de métodos avançados de modulação foram considerados como, por exemplo, o BOQAM e o 8-PSK. Embora os objetivos iniciais fossem aumentar a taxa de bit dos usuários e com isso aumentar o alcance dos serviços, o EDGE foi promovido a sistema 3G. O novo papel do EDGE tem muito a ver com uma estratégia evolucionária do IS 136, o TDMA americano

A evolução de GSM e do IS 136 para EDGE está sob encargo da ETSI e do consórcio UWC. Conseqüentemente o EDGE será compatível com ambos. O plano é empregar o GPRS, passando para enhanced GPRS (EGPRS) e ECSD. Então haverá um desdobramento de modulações de alto nível para realizar serviços 3G EDGE.

Interface de Rádio EDGE

A interface de rádio continuará tendo o GMSK disponível, mas será capaz de utilizar o 8-PSK que tem 3 bits por símbolo ao invés de um bit por símbolo usado no GMSK. Como a taxa de símbolos é 271 ksimb/s então a taxa bruta de bit por timeslot é 22,8 e 69,2 para o GMSK e 8-PSK respectivamente.

Transmissão Comutada de Pacotes

O conceito do EDGE inclui tanto a comutação por pacotes quanto a comutação por circuitos. Na verdade GPRS e HSCSD. O EGPRS difere do GPRS porque ao usar modulação multinível a codificação do canal tem que ser melhorada por estar mais vulnerável a interferência e ruído. Assim, um esquema de adaptação de link estima regularmente a qualidade do link e seleciona entre GMSK e 8-PSK e também a codificação de canal apropriada para proporcionar a taxa de bit por usuário mais alta possível.

O modo enhanced CSD (ECSD) têm os dados entrelaçados sobre 22 frames TDMA. Para modulação GMSK a taxa por timeslot é 14,5 kbps para uma taxa de código de 0,64; enquanto que para 8-PSK essa taxa pode chegar a. 38,8 kbps para uma taxa de código de 0,56.

Para o EGPRS a máxima taxa suportada deve ser de 384 kbps. Uma taxa como essa requer que o usuário faça uso de múltiplos timeslots por frame. Para o 8psk esse numero de timeslots é reduzida. Há a possibilidade de se usar o GMSK para realizar as transmissões de uplink e o 8-PSK para as transmissões de downlink.

6. TECNOLOGIA M2M

Finalmente, depois de entender as bases do M2M, pode-se facilmente entender seu conceito e funcionamento.

6.1 Introdução ao Conceito M2M [5],[20]

Recentemente presenciou-se uma série de mudanças fundamentais nas tecnologias e cultura de comunicação. Há aproximadamente 20 anos, a introdução da telefonia móvel tornou a transmissão de voz independente de local. Sucessivamente, novos conceitos foram inseridos para satisfazer as novas necessidades dos usuários como a transmissão de dados via celular e a internet móvel.

O próximo passo na evolução é estender essa conectividade além dos humanos. Em um Mundo Móvel, independente de localização, será necessária a comunicação entre indivíduos, aparelhos e sistemas. M2M significa *machine to machine, mobile to machine, e machine to mobile,* e normalmente se relaciona aos conceitos de telemetria e telemática, que podem ser considerados sub aspectos do contexto geral do M2M. No Mundo Móvel, o M2M trata da combinação de telecomunicações e tecnologia da informação. Dados Wireless são usados para estabelecer um link entre aparelhos remotos, sistemas e pessoas, aumentando a visão de telecomunicação que se tinha em mente. Agora, não só pessoas utilizam tecnologias de internet e telecomunicações para se comunicar, mas as máquinas ao nosso redor também.

A idéia do M2M é criar soluções para melhorar serviços que já existem, criando novas oportunidades e facilitando o dia a dia. Utilizando-se do M2M, que são produzidos sob demanda, empresas podem automatizar seus processos, integrando seus serviços com seus sistemas de informação. O objetivo é aumentar a performance e a competitividade da empresa através do aumento na eficiência, diminuição de custos, aumento de receita, ou um melhor nível de serviço. Exemplos disso são o emprego de M2M em substituição a visitas físicas para fins de medida de gasto de energia e de água, e a diminuição de custos pela utilização dessa informação em tempo real gerada por essa conexão remota de dados.

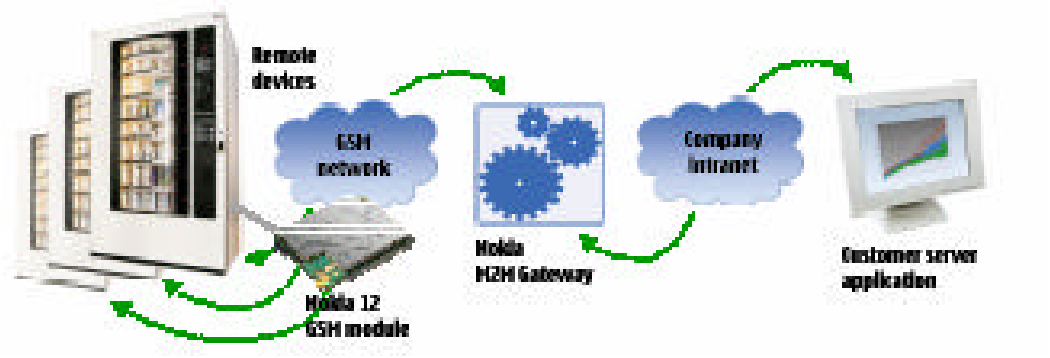

Figura 06: Sistema M2M em funcionamento.

Soluções M2M funcionam tipicamente coletando informações, configurando parâmetros, enviando informações de situações atípicas ou se encabendo de uma transação on-line por meio de conexão remota de dados (fig 06). Não há limites para o M2M podendo servir a qualquer tipo de negócio ou ambiente físico. Para dar uma idéia do potencial dessa tecnologia, algumas das aplicações já reconhecidas se encontram na figura 07 abaixo.

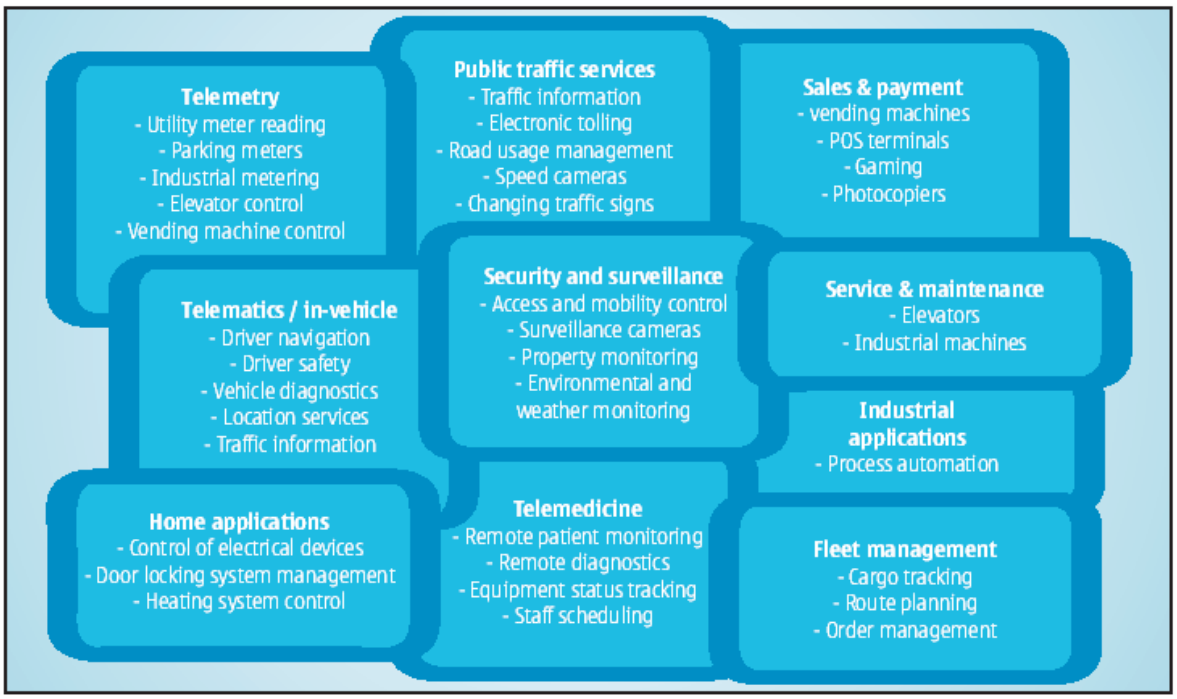

Figura 07: Aplicações típicas M2M

Atualmente, a maior utilização de M2M é dentro das empresas, acelerando processos internos. Entretanto, cada vez mais há aplicações M2M voltadas aos consumidores das empresas. Afinal, virtualmente todas as máquinas podem ser conectadas

à internet. Posteriormente, essas máquinas fornecerão alguns serviços e informações locais para usuários de aparelhos móveis, como no exemplo de uma máquina de vendas a seguir.

O primeiro objetivo de se usar a M2M em uma máquina desse tipo é diminuição dos custos de operação pelo uso de monitoramento remoto, coletagem de dados e configuração das máquinas. Essa aplicação M2M permite o mantimento de um estoque otimizado, realização de manutenção ativa, concerto imediato de defeitos e alteramento de preços.

Uma aplicação futura poderia ser a possibilidade de pagamento via celular, o que permite que a empresa deixe de se preocupar com o manejamento de dinheiro, e obtenha aumento de vendas. Criar um link direto com o consumidor através de seu aparelho móvel tornará possível construir perfis de usuário e personalizar serviços e ofertas de acordo com as necessidades daquele usuário.

6.2 Requisitos para soluções M2M de sucesso [5]

Para se ter sucesso com aplicações M2M, uma solução dever ser confiável, ser capaz de ser melhorada, e ter um custo acessível. A maior dificuldade nesse processo é a criação de aplicações, devido à demanda específica de cada usuário, e a integração de sistemas. Algumas aplicações, como o pagamento via MS, requerem muito mais segurança que outras atividades, como a de monitoramento.

6.3 Uma parte essencial do mercado de dados [5]

Nos dias de hoje, vê-se uma acirrada disputa pelo mercado de transmissão de voz. Há pelo menos quatro operadoras na região, disputando os mesmos usuários, a procura de novos meios de obter receita, através do tráfego de dados. O uso do SMS passou de um meio de comunicação pessoal a serviço popular, gerando boas receitas. Há três tipos de segmentos de serviços de dados: mensagem, navegação e M2M. Cada um deles necessita de estratégias e desenvolvimentos únicos e específicos. Aplicações M2M podem ser combinadas com aplicações de mensagem e navegação, com o intuito de criar soluções que satisfaçam as necessidades dos consumidores.Afinal, conteúdo é o que realmente importa no mercado de transmissão de dados. Serviços M2M serão um setor significativo na transmissão de dados, por permitir conectividade remota e acesso à internet de qualquer localidade. Uma forma de lucrar para as operadoras seria oferecer um

serviço de gateway, o que significa hospedar e gerenciar links de dados entre servidores e clientes M2M.

A tendência mundial de negócios em rede, de trabalhar com parcerias e focando as próprias competências também se aplica ao M2M. Redes de empresas são necessárias na criação de serviços M2M. Conhecimento específico do ambiente de negócios, tecnologias de comunicação, desenvolvimento de software, integração total de sistema, serviços de atendimento ao cliente e cobrança, são alguns dos parâmetros necessários para um serviço M2M funcionar. Ou seja, é necessária a união de profissionais de várias áreas. Na figura abaixo, estão ilustradas as várias atividades integradas no processo.

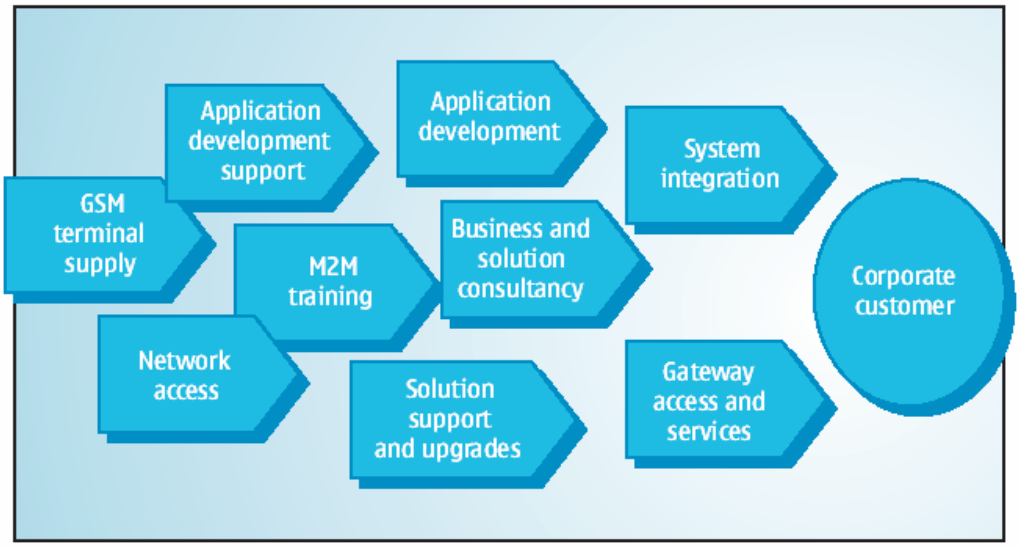

Figura 08: Atividades integradas em um serviço M2M

Um dos grandes desafios do M2M parece ser o gerenciamento das complexidades causadas pela presença de tantas empresas envolvidas para prover um serviço. Afinal, usuários não querem comprar peça por peça de diferentes empresas para conseguir utilizar uma aplicação. Eles querem uma aplicação *End-To-End*. Os vencedores dessa corrida serão as empresas que conseguirem estabelecer e gerenciar parcerias com competência para criar soluções *End-to-End.*

Os maiores beneficiários de aplicações M2M serão clientes corporativos. Isso porque empresas estão sempre buscando algo que aumente sua eficiência, corte custos e melhore sua relação com os clientes. Essas são qualidades intrínsecas do M2M.

Terminais desse tipo já estão instalados em várias indústrias. Por meio do M2M, as empresas podem mais facilmente analisar seus negócios e processos, o que contribue para seu desenvolvimento.

Uma solução M2M integra todas os terminais de uma empresa em um único sistema, esteja ele nas ruas ou em qualquer local do mundo. Custos de operação podem ser reduzidos quando se substituem visitas, manutenção, e atualização in loco pelos mesmos serviços feitos a distância graças a conexões on-line com todas as máquinas.

6.4 GSM - A tecnologia sob o M2M

O que realmente importa para clientes M2M é a funcionalidade do serviço e a facilidade de uso; de forma transparente. Porém, para projetar uma aplicação dessa natureza, deve-se considerar com cuidado qual a melhor forma de fazê-lo. Uma decisão dessas pode ser o sucesso ou o fracasso de uma tecnologia. Entretanto, nesse caso é evidente que o GSM seria a melhor tecnologia para subsidiar o M2M. Isso, por se tratar de uma tecnologia global, presente em mais de duzentos países, de padrão aberto e padronizado, o que torna as soluções M2M também globais. Outra vantagem é o alto nível de serviços de dados como o GPRS, HSCSD e SMS que contém aplicações de segurança já funcionando na rede GSM.

O mercado GSM vivencia um enorme crescimento desde a última década, e a tendência é que continue a crescer. Conseqüentemente, os preços dessa tecnologia tendem a ser reduzir, beneficiando o M2M, tanto pelo barateamento da tecnologia, como ampliando as possibilidades dessa tecnologia, tornando cada novo usuário conectado um cliente M2M em potencial. [5]

7. O Terminal Nokia 12

Há na plataforma M2M vários terminais distintos capazes de realizar ações parecidas. Entretanto, o Nokia 12 parece ser o mais versátil e promissor devido à utilização de Java. Ao utilizar Java este terminal se torna capaz de realizar processamento, coisa só possível em outros terminais na presença de processadores externos. Isso devido a Máquina Virtual Java, estudada no capítulo 2. As características principais do Nokia 12 são descritas nesse capítulo.

7.1-Introdução [12]

O terminal Nokia 12 é um módulo GSM inteligente desenvolvido para aplicações M2M e para outras soluções Wireless. É uma parte da Plataforma M2M da Nokia, mas que também pode ser usada como um modem celular por comandos AT. Por suportar controle de I/O baseados em dispositivos móveis e Java, o desenvolvimento de aplicações nele é barato e rápido.

Seu público alvo são empresas de produção que integrem o Nokia 12 com seus produtos e os venda sob sua própria marca para seus clientes. O Nokia 12 é parte da plataforma M2M da Nokia (há outros tipos de terminal, porém não utilizam Java), que é uma plataforma aberta, end-to-end, para o desenvolvimento de operações e serviços. Além de terminais como o Nokia 12, Terminais de conectividade GSM e o Gateway M2M da Nokia compõem a plataforma, que se baseia em CORBA e GSM, que são padrões amplamente aceitos da indústria.

A solução Nokia conecta uma aplicação servidor acessível por internet a aparatos remotos sem fio, como portas, bombas e etc. Os terminais GSM conectam os aparelhos a rede GSM. O Nokia Gateway faz a ponte entre a rede GSM e a internet para conectar os aparelhos remotos das empresas com seus sistemas de informação. A plataforma utiliza vários métodos de comunicação, porém esconde toda a complexidade das redes Wireless do software que vai ser utilizado pelo usuário.

7.2-Especificação

7.2.1-Introdução

O módulo GSM Nokia 12 foi projetado para aplicações M2M e outros serviços Wireless. É um aparelho que existe em duas versões:

- RX-2 aparelho GSM de banda dupla com suporte a EDGE, GPRS, (HS)CSD, e SMS em EGSM 900/GSM 1800 MHz.

- RX-9 aparelho GSM de banda dupla com suporte a EDGE, GPRS, (HS)CSD, e SMS em GSM 850/GSM 1900 MHz.

Aqui ambos são referenciados como Nokia 12.

O Nokia 12 pode ser usado em diversas aplicações devido aos seus três modos de operação. Aplicações simples de I/O podem ser implementadas no modo de Controle do usuário, que oferece personalização de mensagem, mensagem segura, e funcionalidades baseadas em tempo para aplicações I/O controladas por SMS.

No modo de comando AT, o Nokia 12 pode ser usado como um modem GSM com suporte a Java, para o desenvolvimento de inteligência extra.

No modo M2M system, o terminal se comunica com um servidor de aplicação através do Gateway M2M da Nokia. Nesse perfil, todas as funções estão disponíveis para se desenvolver aplicações de amplo alcance.

Além desses modos, o terminal possui uma pilha TCP/IP integrada, que permite conexões diretas GSM e GPRS de dados entre uma aplicação final remota e um servidor de aplicação. Devido à presença dessa função, APIs de Soquete e HTTP estão disponíveis para o desenvolvimento de aplicações.

Para terminar, o Nokia 12 suporta diversas APIs Java, serviços de localização para integração de módulos GPS, aspectos de confiança como AutoPin,

encriptação GSM e códigos de segurança, mecanismo de reinicialização e autenticação da plataforma Nokia M2M. O suporte a tecnologia Java permite atualizar o IMlet (software da aplicação) à distância.

A funcionalidade GSM do Nokia 12 está conforme as especificações de 97-98-99 e suportam o Java 2 Micro Edition (J2ME), e as especificações IMP1.0 e CLDC 1.0. Ele é compatível com outros terminais Nokia como o Nokia 30 e Nokia 31.

7.3-Componentes do Terminal

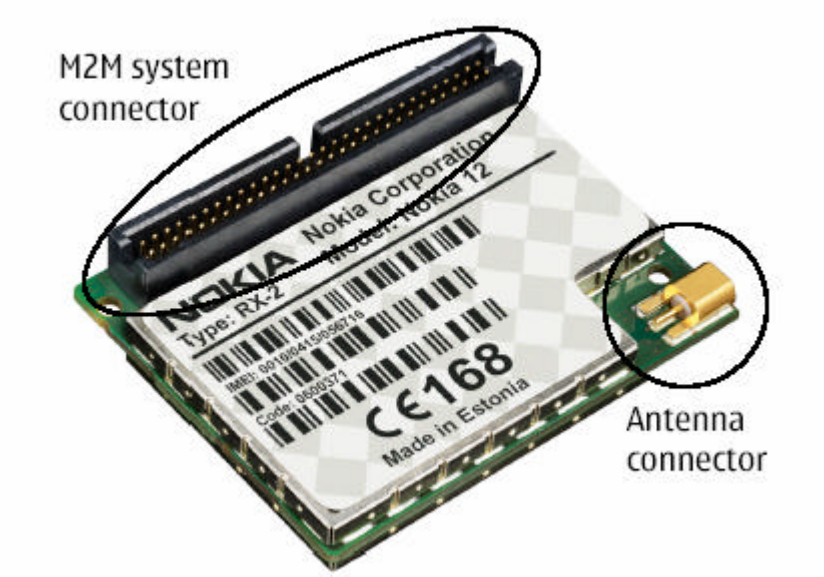

Figura 09: Terminal Nokia 12

O terminal tem entradas e saídas para os seguintes sinais (fig 09):

- Power input (Alimentação)
- Grounding (terra)
- Analog audio (audio analógico)
- A/D converters (3)
- Digital audio
- Reset input

- Reset output
- Asynchronous serial channels (3) (Canais serias assíncronos)
- Digital inputs (max 9)
- Digital outputs (max 8)
- I/O voltage level shifter

Há ainda um conector Microax em miniatura para a integração de uma antena externa. Dependendo da rede, há a possibilidade de se ter relógio em tempo real atualizado pela própria rede GSM.

7.4- Modos de Operação

No terminal Nokia 12 há três modos de operação para diferentes tipos de aplicação.Entretanto esses modos de operação são puramente lógicos e podem ser utilizados simultaneamente.

7.4.1 Modo M2M system[12]

Este modo permite operações na Plataforma Nokia M2M através do Gateway M2M Nokia.

A Plataforma Nokia M2M é uma plataforma de comunicação, composta por um Gateway e um terminal, como o Nokia 12. A plataforma oferece serviços de aplicação assim como soluções de *middleware* para tornar o desenvolvimento de aplicações M2M o mais fácil possível.

Ela conecta aplicações de uma intranet de uma corporação aos seus componentes remotos utilizando algum dos suportes GSM (GPRS, HSCSD, etc) e a internet. A comunicação se dá através do Gateway e do terminal. O gateway é ligado a elementos de rede da operadora GSM como SMSC ou GGSN.

Além de estabelecer uma conexão e ativar o GPRS, a plataforma fornece mecanismos de autenticação e controle, além de mecanismos de confiabilidade e endereçamento especificamente desenvolvidos para comunicação Wireless. Para

aplicações, a plataforma oferece a transparência da rede de comunicação, deixando a cargo do desenvolvedor apenas a lógica da aplicação.

O uso de interfaces abertas e acesso a múltiplos suportes GSM (e até a troca dinâmica deles) torna possível o ajuste a qualquer mudança na rede, nas tarifas e também das exigências de novas aplicações.

7.4.2- Modo de comando AT[6]

O Nokia 12 pode ser usado como um modem GSM tradicional. São compatíveis comandos das seguintes especificações: ITUV.25ter, ETSI GSM 07.07 e ETSI 07.05 AT.

Neste modo, uma porta serial do Nokia 12 pode ser usada como I/O digital ou conectada a um aparelho externo como um módulo GPS, que podem ser controladas por uma aplicação Java integrada.

7.4.3- Modo Controle do Usuário [13]

Este modo ativa aplicações I/O simples que são controladas por aparelhos móveis. Podem ser usadas informações de ligado/desligado ou de sinal contínuo.

Pelo uso de mensagens de texto enviadas ao terminal, o Nokia 12 passa a monitorar e controlar o aparelho agregado a ele através dos conectores do sistema M2M. O terminal processa a mensagem de controle recebida, e retorna uma mensagem de texto de volta ao celular do usuário. Mensagens de controle recebidas e reconhecidas não são armazenadas na memória do terminal. Porém, se a mensagem não for reconhecida como sendo de controle, ela é armazenada no cartão SIM ou na memória como uma mensagem padrão de texto.

Segurança

O controle de acesso no modo controle do usuário possui vários níveis de autenticação que podem ser implementados: um nome de usuário e senha especificados pelo usuário, comandos definidos por usuário e MSISDN.

Deve-se usar um identificador de mensagem, ou seja, deve-se dar um nome ao seu terminal para poder usá-lo nesse modo. Cada mensagem de controle é iniciada com um identificador, o que possibilita ao terminal reconhecê-la como uma mensagem de controle do modo Controle do Usuário. Somente usuários que conhecem o identificador de mensagens correto podem comandar o Nokia 12.

Para aumentar ainda mais a segurança da comunicação entre o celular e o terminal, senhas podem ser usadas. A senha deve ser inserida nas mensagens de controle enviadas ao terminal.

No caso de mais de um usuário necessitar usar o terminal, há a possibilidade de registrar um grupo de até 16 números de celular de usuários com privilégios de controle do terminal.

Para finalizar, os comandos de fábrica do terminal podem ser substituídos por comandos definidos por usuário. Após a substituição, um comando anterior não é mais reconhecido. Somente o usuário que definiu o comando pode controlar e monitorar o aparelho.

7.5- Java ä

A tecnologia Java permite a criação de aplicações para aparelhos M2M. Essas aplicações podem ser baixadas pelo ar (OTA) para aparelhos M2M para serem executados, o que permite um rápido desenvolvimento e aprimoramento.

Tradicionalmente, o desenvolvimento de aplicações M2M era feito em linguagem C. Porém isso tornava a atualização e o *download* complicados. Também não havia como desenvolver a plataforma C embarcada sem ferramentas especiais. As aplicações desenvolvidas não podiam rodar diretamente dentro do terminal M2M, necessitando de processadores externos que as executassem.

No caso de Java, aplicações podem ser desenvolvidas e testadas com ferramentas disponíveis a qualquer pessoa. Devido a sua natureza portátil e dinâmica, aplicações Java podem ser baixadas e atualizadas dinamicamente. Também, como o Nokia

12 inclui uma máquina virtual, as aplicações desenvolvidas nesta plataforma podem ser executadas diretamente no terminal.

A última novidade da plataforma Java é conhecida como Plataforma Java 2. Um de seus aspectos mais poderosos é sua escalabilidade para se adaptar a uma grande variedade de plataforma de aparelhos. Como nem todas essas plataformas são capazes de oferecer as mesmas funcionalidades e recursos, a plataforma Java foi dividida em três edições diferentes, a saber , Java 2 Enterprise Edition, Java 2Standard Edition e Java 2 Micro Edition. A primeira é usada em desenvolvimento de soluções de servidor sob demanda. A segunda é utilizada em computadores comuns, e a última é especialmente desenhada para aparatos eletrônicos de consumidores, como telefones móveis e aparelhos M2M.

O J2ME não é uma simples especificação ou componente de software. Ao invés disso, ele é uma coleção de tecnologias e especificações que são designadas a partes distintas do mercado de aparelhos pequenos. A parte central da plataforma J2ME é formada por duas configurações distintas, nomeadas Connected Device Configuration (CDC) e Connected Limited Device Configuration (CLDC). Uma configuração define bibliotecas centrais Java e as capacidades da Máquina Virtual. O CDC é projetado para aparelhos portáteis de alta capacidade como comunicadores; já o CLDC é projetado para aparelhos de baixo custo como celulares mais populares. Sobre as configurações existem perfis (*profiles*) que definem a funcionalidade em uma categoria específica de aparelho.

O Perfil de Informação do Módulo (IMP), utilizado no Nokia 12, é baseado no Perfil de Informação de Aparelhos Móveis (*Mobile Information Device Profile*-MIDP), que é amplamente usado e conhecido. O IMP é designado especificamente para dispositivos controlados remotamente. A diferença entre eles é que o IMP não requer uma interface gráfica do usuário. As aplicações Java que aparelhos baseados em IMP podem executar são denominadas IMlets. Tecnicamente, um IMlet é uma classe Java derivada da classe javax.microedition.midlet.MIDlet. Esta classe possui métodos que o aparelho IMP chamará durante o ciclo de vida do IMlet. Por exemplo, quando o IMlet é inicializado, seu método startApp() é chamado, similarmente a função Main em programas C.

7.5.1-Interfaces de aplicação Java

O Nokia 12 suporta diversas APIs para aplicações Java. Além das APIs do perfil IMP, APIs para conectar aparelhos externos usando as capacidades do terminal são oferecidas assim como APIs para conectividade Wireless. O IMP também oferece funcionalidades para uso de memória e modelos de aplicações avançadas para produzir soluções M2M inteligentes. Algumas das principais APIs são descritas a seguir.

Wireless Messaging API

A API WMS dá acesso a fontes padrão de comunicação SMS da rede GSM. Essa API permite desenvolver aplicações M2M baseadas em J2ME que sejam conectadas bidirecionalmente a terminais GSM ou qualquer outro serviço baseado em SMS na infraestrutura de aplicação.

Serial API

A API serial é uma simples interface baseada na estrutura conectora IMP1.0 para comunicação com aparelhos externos de linha serial ativados como módulos GPS ou aparatos de mensuração.

ORB API

A ORB API é uma estrutura para a criação de aplicações cliente/servidor reais em CORBA baseado em J2ME via Nokia M2M Gateway. O Gateway Nokia M2M facilita a confecção de aplicações cliente/servidor bidirecionais com as atuais redes GSM e GPRS.

IO API

Ao utilizar-se dessa API, aplicações J2ME podem controlar e monitorar os pinos I/O analógicos e digitais do terminal.

HTTP API (inclusa no IMP)

A API HTTP oferece a clientes HTTP capacidades de conectividade a uma aplicação J2ME. Permite buscar informação em serviços WEB na internet via serviços GPRS públicos. Os protocolos usados por essa API são o TCP/IP ou UDP/IP.

Socket API

Essa API permite a utilização da funcionalidades dos protocolos TCP/IP e UDP/IP em aplicações J2ME via interface do terminal. Com isso é possível abrir soquetes de clientes aos servidores na internet ou intranet. Essa funcionalidade funciona bem na atual rede GPRS devido à direção de comunicação do cliente. Também se pode usar (HS)CSD.

RMS API

Essa API torna possível salvar a aplicação na memória do terminal.

7.5.2- TCP/IP e UDP/IP

Na comunicação de dados, sempre há a necessidade da utilização de protocolos de nível baixo para assegurar uma transmissão de dados confiável. As pilhas de protocolos são de implementação difícil e cara, portanto o Nokia 12 fornece pilhas de protocolo TCP/IP e UDP/IP na camada de transporte. A interface padrão de soquetes também é implementada no Nokia 12 para possibilitar o uso dessas pilhas. O protocolo HTTP também é suportado para buscar páginas www de um servidor.

TCP é um protocolo orientado a conexão que fornece um fluxo confiável entre duas aplicações em redes que suportam o protocolo IP. Atualmente, TCP/IP pode ser utilizado em ligações de dados GSM e GPRS. A implementação deste protocolo no Nokia12 é otimizado para redes Wireless. A versatilidade desse protocolo o torna ideal para uma diversidade de aplicações muito grande, desde FTP a navegadores WEB.

A natureza confiável do TCP/IP minimiza a necessidade das aplicações checarem se a transmissão foi realizada com sucesso. O mecanismo que garante essa confiabilidade é um Checksum (soma dos dados transmitidos). Se houver perda de pacotes na rede, estes são retransmitidos.

7.5.3- Aspectos de Confiança e Segurança

O terminal Nokia 12 suporta encriptação GSM normal, códigos de segurança GSM, AUTOPIN e checagem ativa com mecanismos de reinicialização, para garantir a confiança e segurança da aplicação M2M.

Métodos efetivos e confiáveis fornecem supervisionamento de controle do terminal e da aplicação. A conexão entre o terminal remoto e o Gateway é mutuamente autenticada para prevenir intervenções. Métodos adicionais podem ser implementados dentro das aplicações.

Encriptação GSM

O terminal suporta encriptação GSM e GPRS

Códigos GSM de segurança

Os seguintes códigos são suportados pelo terminal:

- PIN
- PIN2
- PUK
- PUK2
- SECURITY CODE
- CALL BARRING CODE

AutoPIN

O recurso AutoPIN permite que o aparelho se recupere de uma situação de falta energia sem intervenção no local, ajudando a prevenir fraudes. O código PIN é programado na memória do terminal de onde é retransmitida em situações atípicas. Quando o AutoPin está em uso o cartão SIM é inútil a qualquer agente exterior.

Checagem ativa e mecanismo de reinicialização

A conexão entre o Nokia 12 e o módulo de aplicação é conferida periodicamente pela checagem ativa.Caso essa conexão seja quebrada, o módulo de aplicação e o terminal são reinicializados. Esse recurso só existe no modo M2M system.

Há também pinos de reinicialização em ambos os módulos que podem ser controlados por aplicação ou usuário.

Autenticação da plataforma Nokia M2M

O acesso do terminal Nokia 12 via Nokia M2M Gateway ao servidor de aplicação é restrito por controle de acesso. Para (HS)CSD e GPRS, a autenticação é feita através de RADIUS/CHAP. Nesse processo, o Gateway funciona como um servidor RADIUS. Para SMS, a autenticação é feita no terminal MSISDN recebida da rede da operadora de celular.

7.5.4- Serviços suplementares

- Auto redial
- CLIP, CLIR (GSM)
- COLP, COLR
- Call forwarding
- Call waiting
- Call hold
- Conference call (Multiparty service)
- Call barring (GSM)
- Unstructured SS data
- Explicit call transfer

7.5-Simulador Conceitual Nokia 12 IMP 1.0 e Nokia M2M Gateway Trial Version [9],[12]

7.5.1 Simulador Conceitual Nokia 12 IMP 1.0

O simulador Nokia 12 é uma ferramenta que simula funções do produto real. Pode ser usado para o desenvolvimento e verificação de aplicações M2M implementadas em J2ME em um ambiente simulado. A vantagem desse simulador é a possibilidade de se desenvolver software sem a presença do terminal Nokia 12. Esta

ferramenta oferece várias opções da configuração de ambientes compatíveis com as necessidades para desenvolvimento de software.

As principais características do simulador são:

- Simulação de recebimento e envio de dados seriais.
- Uso de conexão física seria para recebimento e envio de dados
- Simulação digital e analógica de estados I/O
- Simulação de envio e recebimento de mensagens SMS binárias e texto.
- Controle das chamadas de input e output da aplicação pelo arquivo de log da interface
- Ativação e desativação de restrições de ambiente do simulador
- Suporte ao modo M2M system

7.5.2 Nokia M2M Gateway [15],[21]

O Nokia M2M Gateway Trial Version tem a intenção de testar a tecnologia da Plataforma Nokia M2M e testar softwares em fase de desenvolvimento. A versão Trial suporta dados em SMS e CSD através de um terminal GSM acoplado ao computador hospedeiro do gateway ou o simulador Nokia 12 nas mesmas condições.A comunicação é gerenciada pelo uso de uma conexão TCP/IP local disponível no computador ao invés de GPRS, por exemplo.

O Gateway fornece acesso Wireless CORBA a internet e garante interoperabilidade. Ele faz a ponte entre a rede GSM e a internet estabelecendo conexões wireless e traduzindo protocolos entre as aplicações Web e os aparelhos remotos. O middleware CORBA é otimizado para as conexões Wireless, e o Gateway M2M traduz o protocolo em um padrão CORBA Internet Inter-ORB protocol (IIOP).

Quando integrado ao simulador o Nokia M2M Gateway oferece a possibilidade do desenvolvimento de aplicações M2M completas em um ambiente simulado. Nesse cenário, Imlets e aplicações por trás do Gateway se comunicarão usando mensagens CORBA.

7.5.3 Ambiente do Simulador

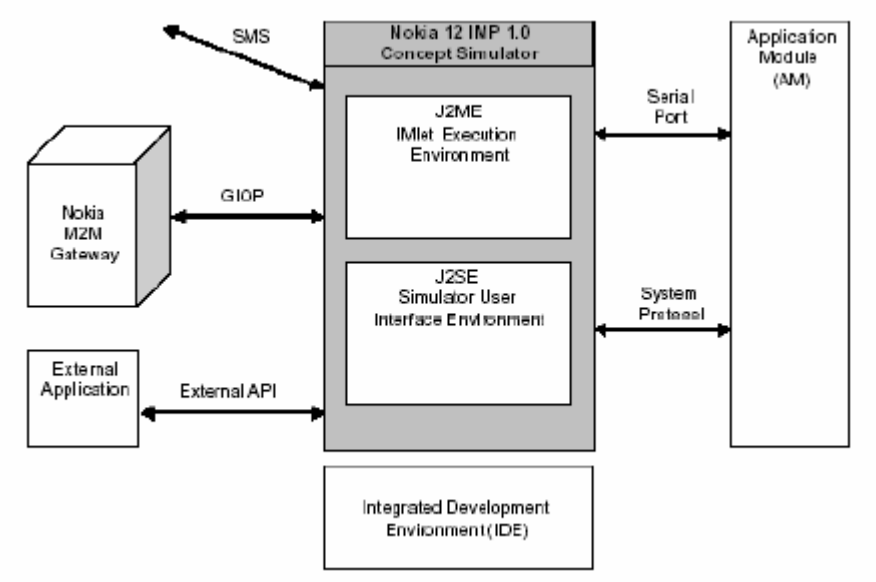

Figura 10: Ambiente do simulador Nokia 12

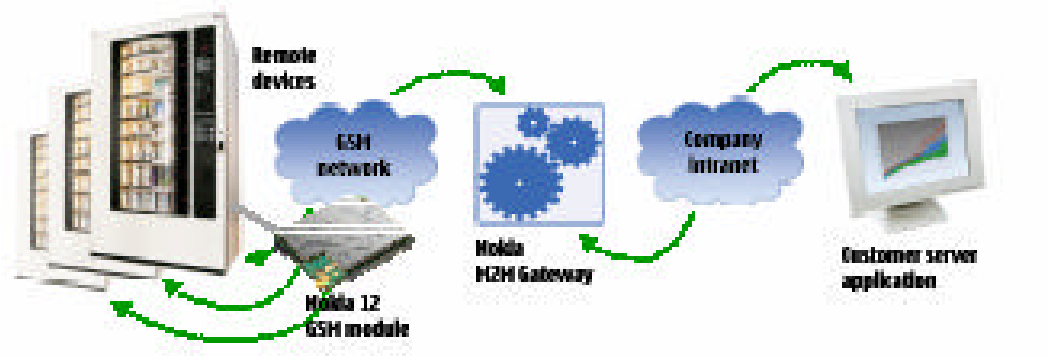

Figura 11: Ambiente real.

Arquitetura Geral

O ambiente do simulador Nokia 12 consiste do Gateway Nokia M2M, do próprio simulador, aplicações externas, de um IDE (ambiente de desenvolvimento integrado, no caso WTK2.0) , um módulo de aplicação e uma rede unindo todos esses elementos (fig 10). Juntos esses elementos constituem um ambiente completamente funcional para o desenvolvimento de aplicações M2M reais (fig 11).

Dos elementos citados, o Gateway e a Aplicação Externa são aplicações à parte do simulador. Porém podem ser usados com o simulador para testar Imlets e as interfaces públicas, como a API serial e as outras descritas anteriormente. Aplicações Externas podem receber notificações específicas de ciclos de vida, e ter algum nível de controle sobre pinos I/O simulados, mensagens SMS simuladas e sobre a porta J2ME simulada. Esses ambientes simulados podem ser modificados pelo IMlet, Aplicação Externa ou pelo usuário na própria interface do simulador.

Aplicações Externas devem ser executadas no mesmo processo J2SE que o simulador. Programadores de Aplicações Externas devem saber que a API Aplicação Externa só pode ser executada junto com um IMlet em um ambiente simulado. Aplicações Externas não podem ser executadas em ambientes reais de desenvolvimento M2M.

O simulador Nokia12 consiste de dois Ambientes Java de Execução(JRE): J2ME, que executa o IMlet em desenvolvimento, e J2SE, que executa a interface do simulador e torna a entrada de dados no sistema possível.

O módulo de aplicação é um aparelho agregado ao terminal M2M. O simulador se comunica com o Módulo de Aplicação via interface serial ou por uma interface de protocolo de sistema.

8. Implantação da Rede e de Serviços M2M [6],[7],[8],[9],[10]

Essa sessão é dedicada à simulação dos conceitos descritos em todo relatório. Idealmente, a simulação utilizaria a Plataforma Nokia M2M completa, em uma situação de monitoramento e controle de um aparelho conectado ao Nokia 12. Porém, vários dos componentes da Plataforma não estavam disponíveis, a começar pelo próprio terminal Nokia 12. Esse dispositivo está com o lançamento previsto para o final do ano de 2003. Apesar disto, o simulador seria, teoricamente, capaz de emular todas as funcionalidades do terminal físico. Os terminais Nokia 30 e 31 já são mais antigos e estão disponíveis para compra, porém foram preteridos por não possuírem uma máquina virtual capaz de adicionar processamento ao terminal.

A ausência de um terminal físico trouxe à tona um outro problema. O Gateway M2M Trial Version, atualmente, ainda não é capaz de funcionar com suporte a SMS nessas condições, como seu manual informa. Contactou-se o suporte M2M da Nokia via e-mail e foi confirmada essa informação. Com o funcionamento parcial do Gateway, não foi possível executar sequer os exemplos do próprio manual.

Restou, então, a possibilidade de criar um IMlet comandando uma aplicação externa rodando no próprio terminal. Para não fugir do contexto do que foi descrito, o IMlet criado comanda uma série de aparelho domésticos, como ar condicionado e iluminação, simulados na aplicação externa. O código está no anexo 1.No anexo 2será descrito algumas informações para pessoas que se interessem em dar continuidade a esse trabalho. Seguem Imagens da simulação.

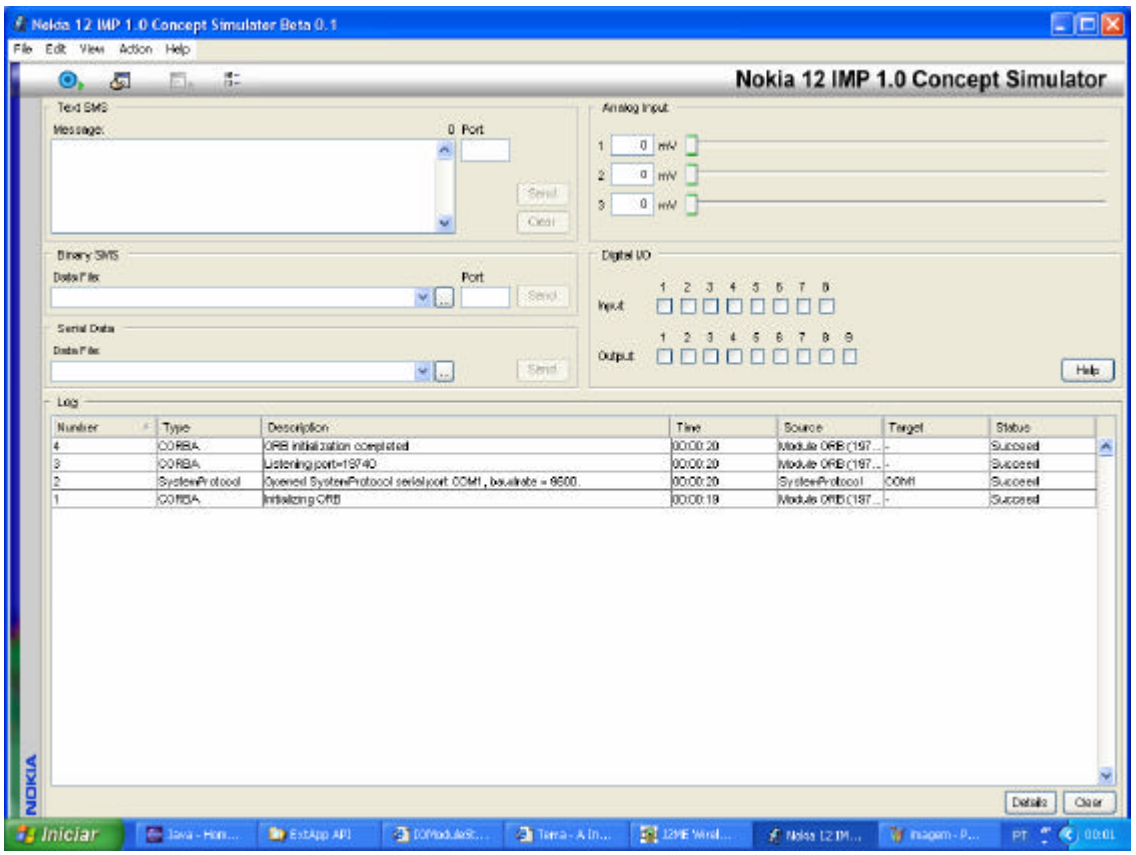

Figura 12: Inicializando Terminal

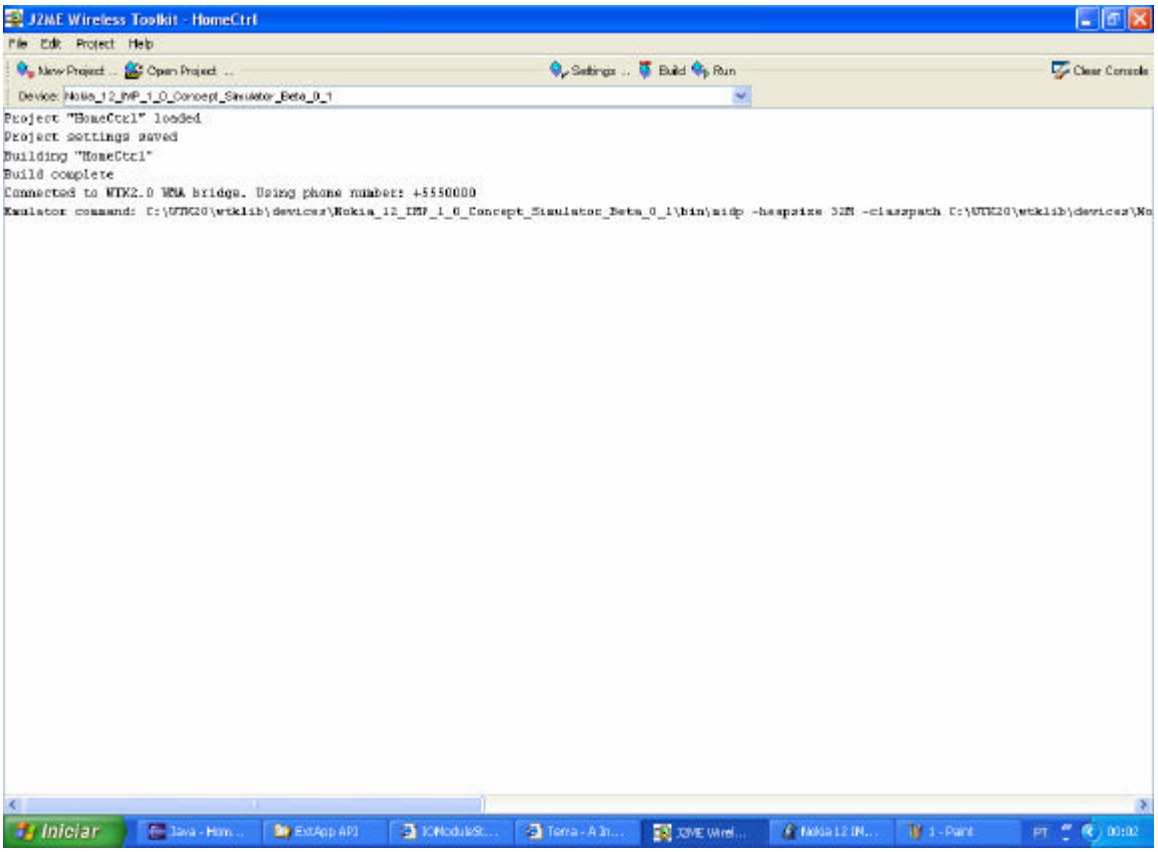

Figura 13: Arquivo de Log inicializado

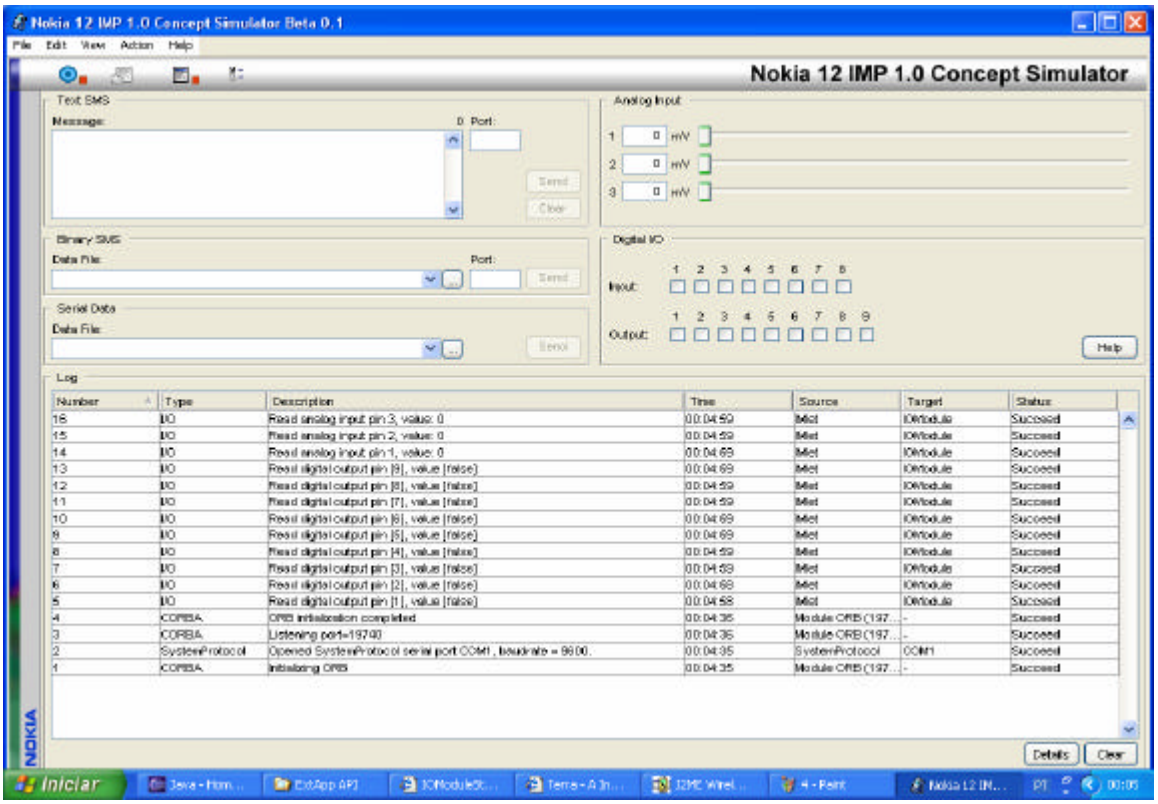

Figura 14: Imlet inicializado

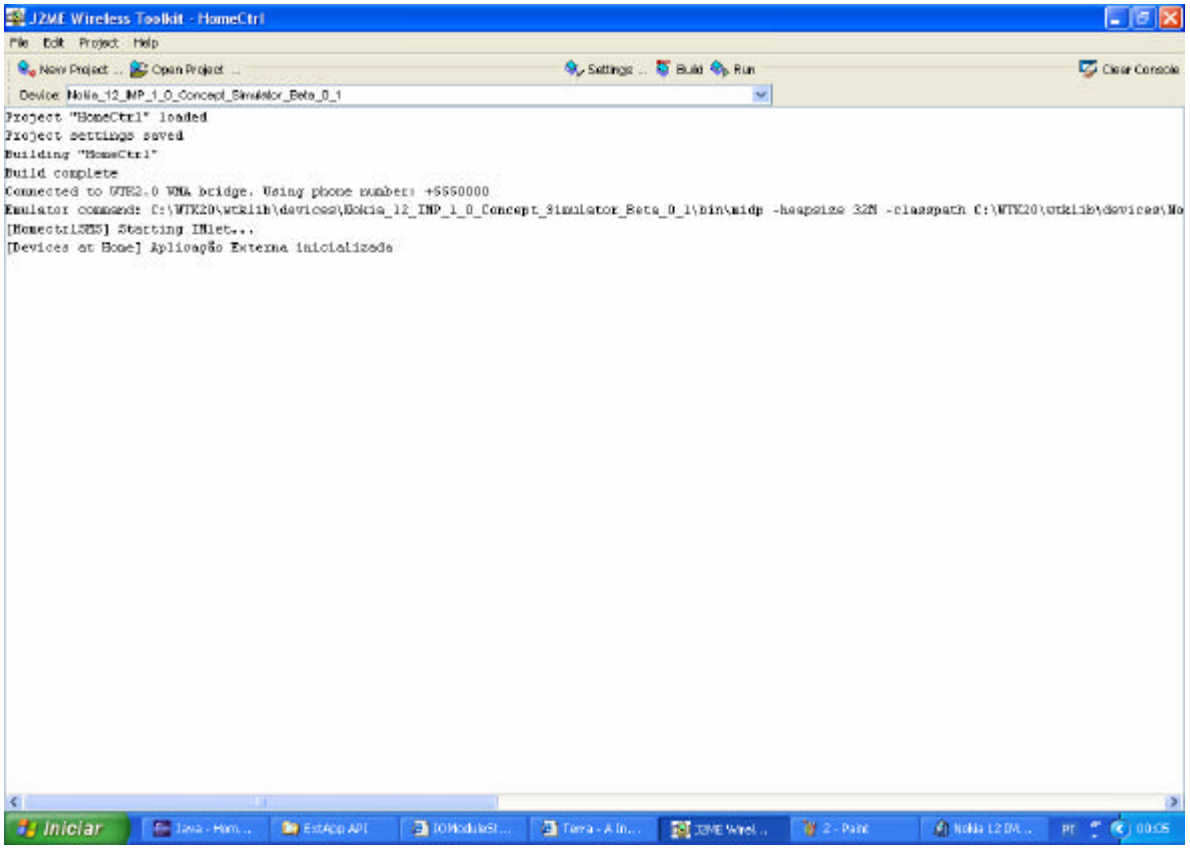

Figura 15: Arquivo de Log registrando inicialização

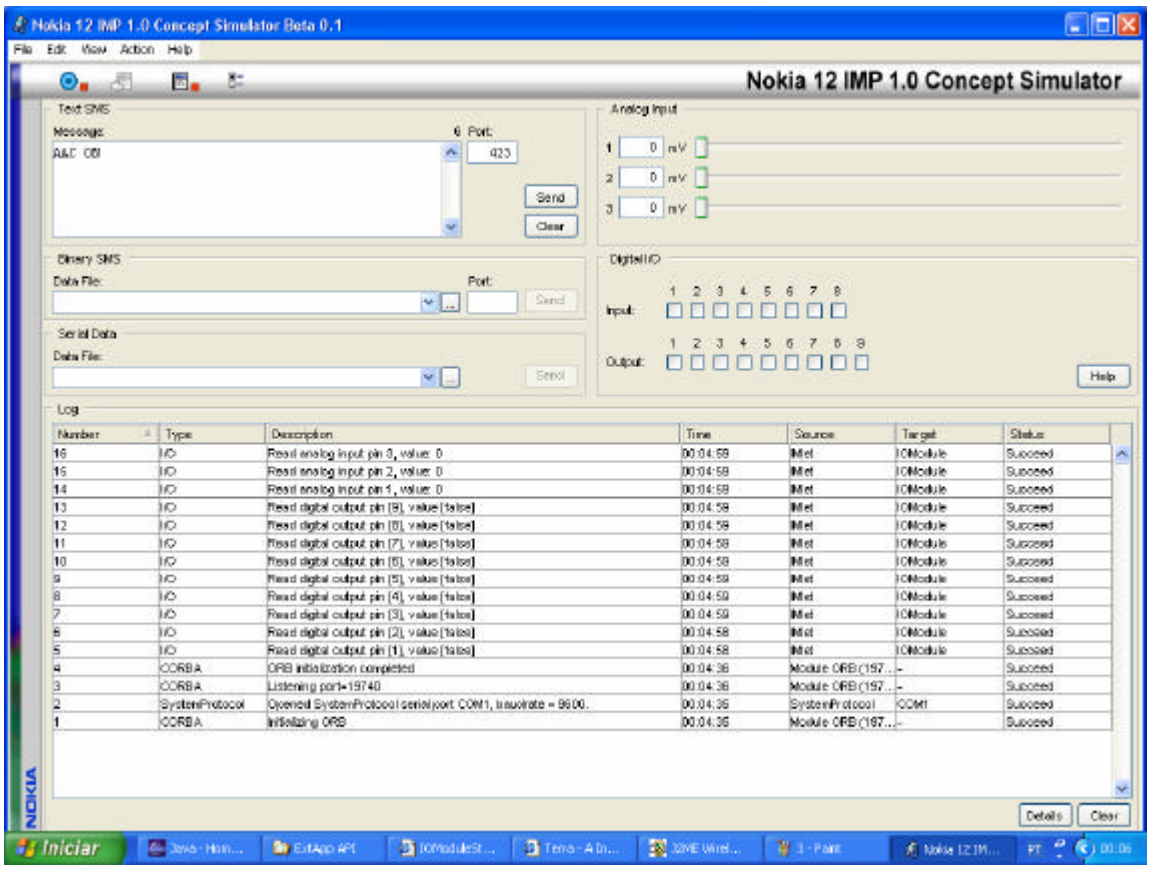

Figura 16: Comando SMS para ligar o Ar condicionado antes de ser enviado

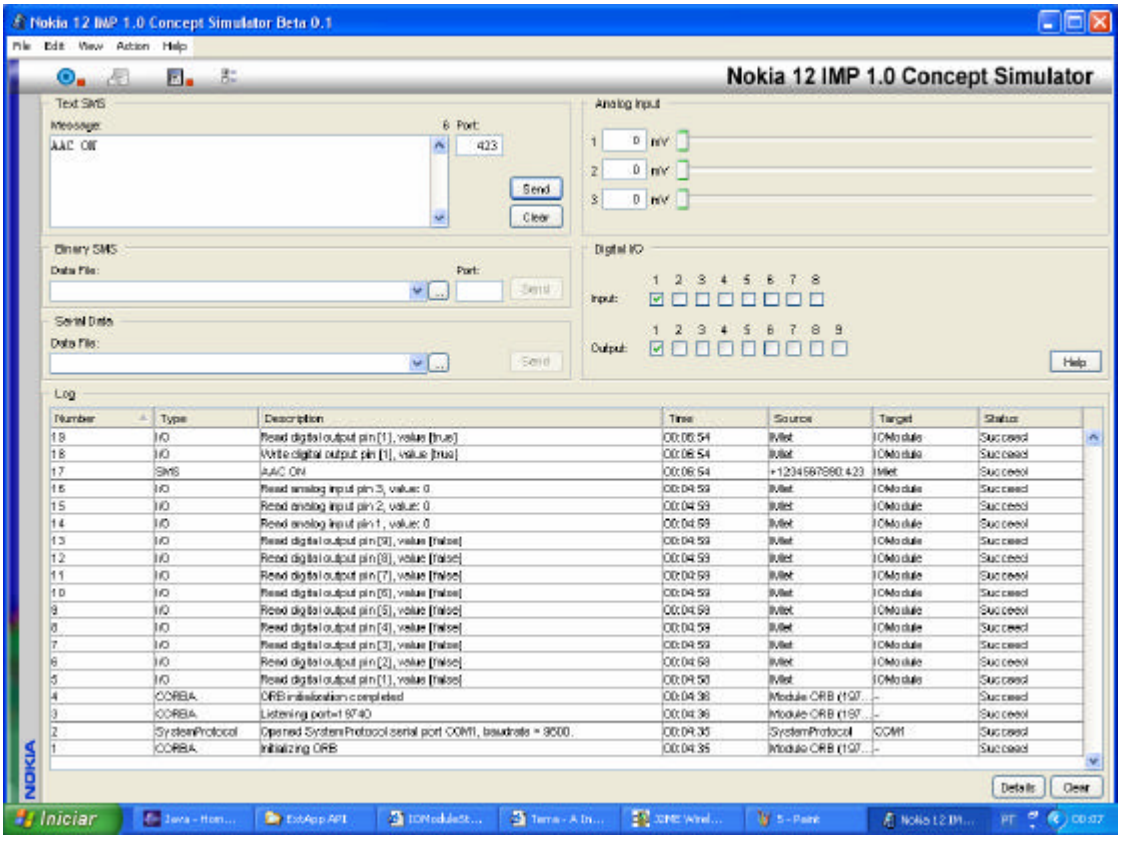

Figura 17: Comando do ar condicionado processado e mostrando seu funcionamento

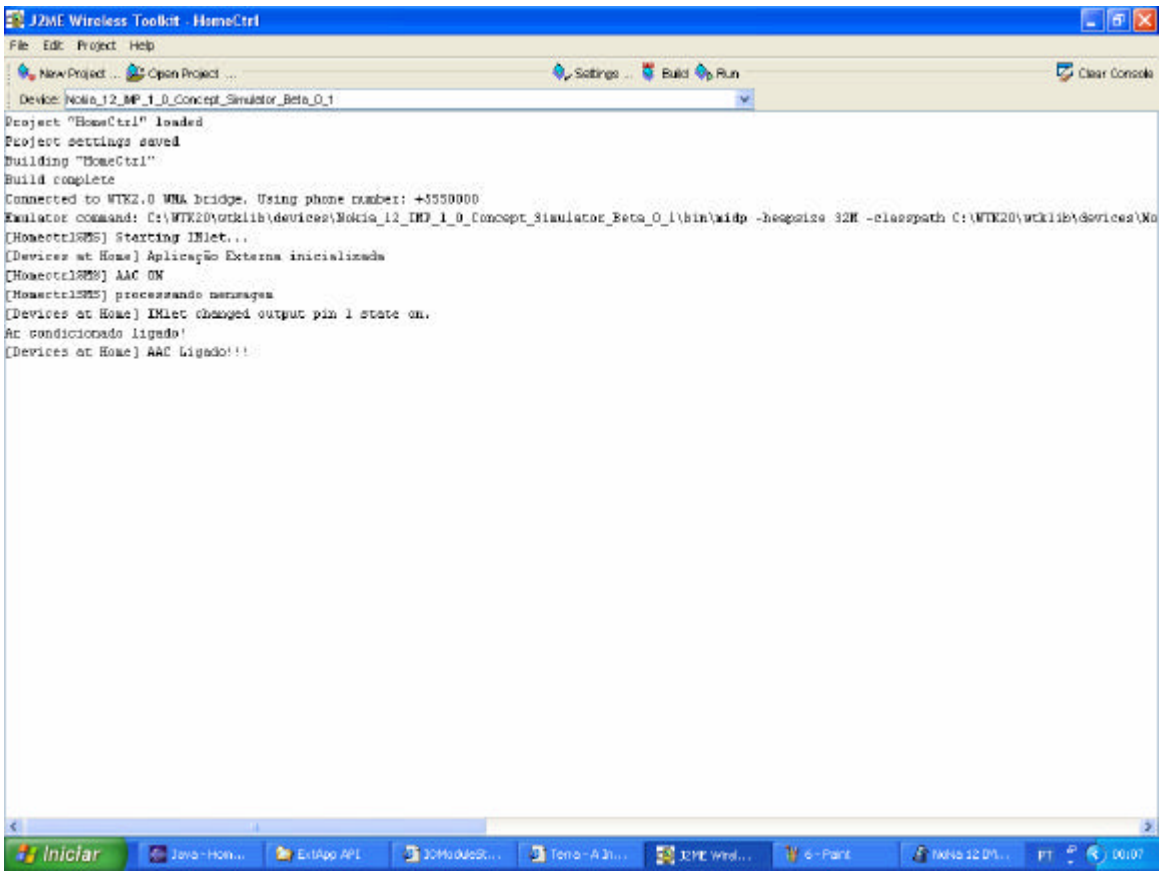

Figura 18: Arquivo de Log Registrando Operação e Confirmando Sucesso

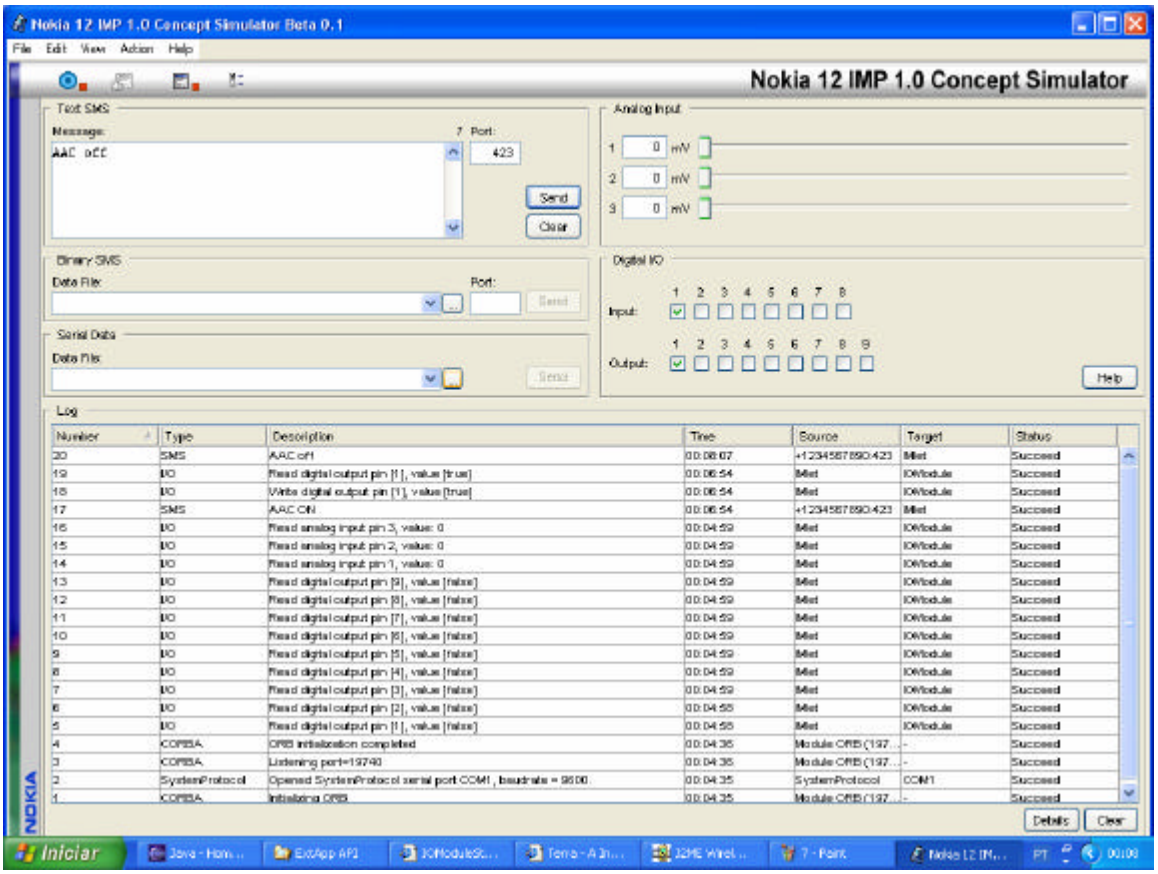

Figura 19: Comando Equivocado para desligar ar condicionado

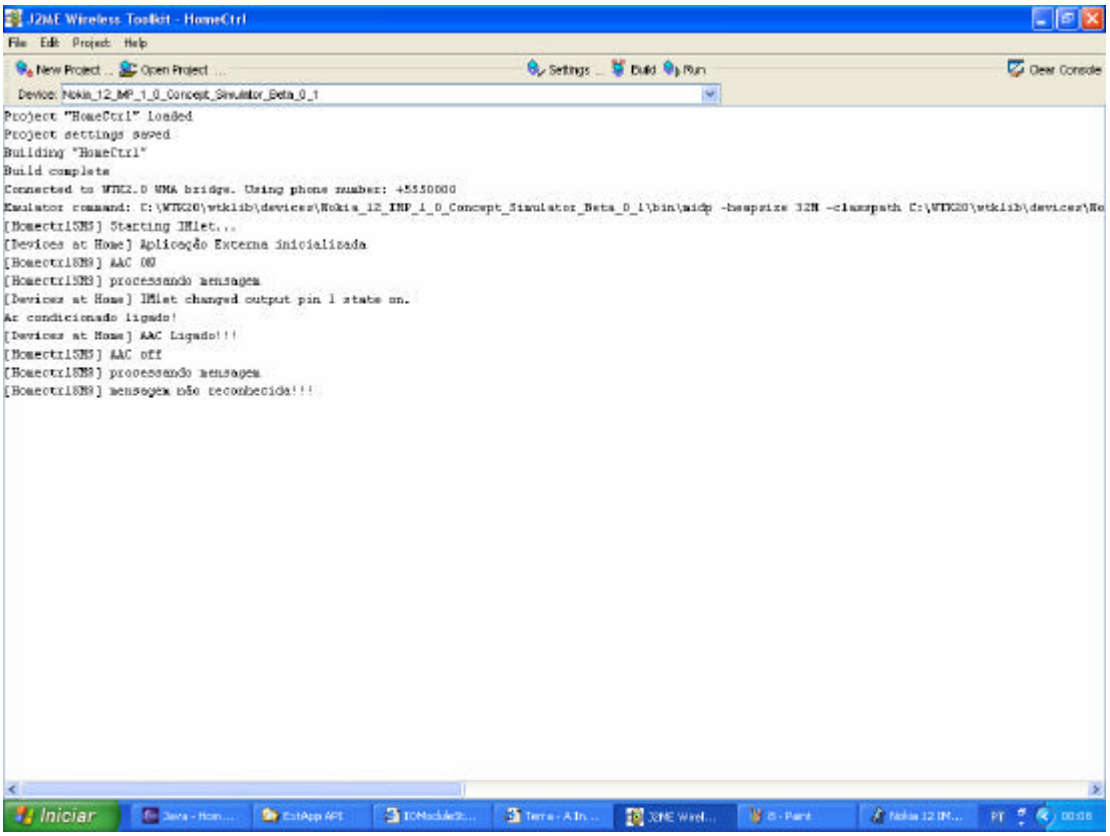

Figura 20: Comando não reconhecido (ar condicionado permanece ligado)

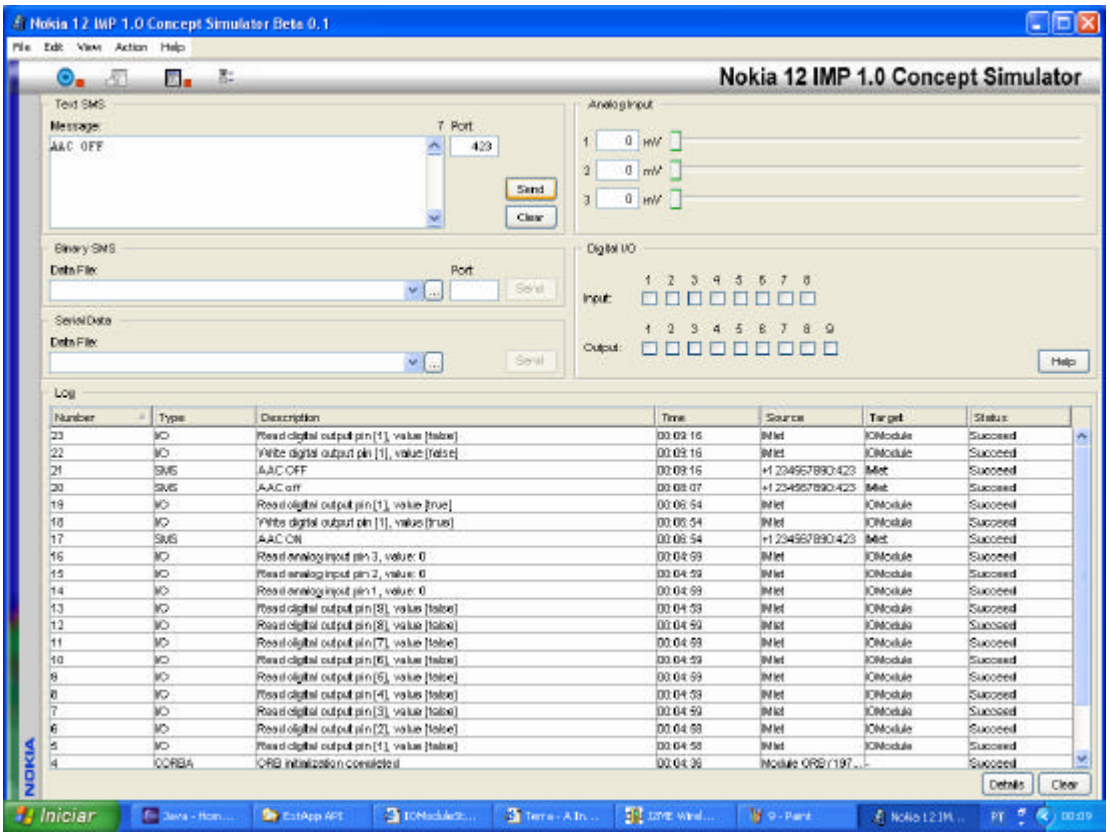

Figura 21: Comando Correto para desligar ar condicionado e ele é efetivamente desligado

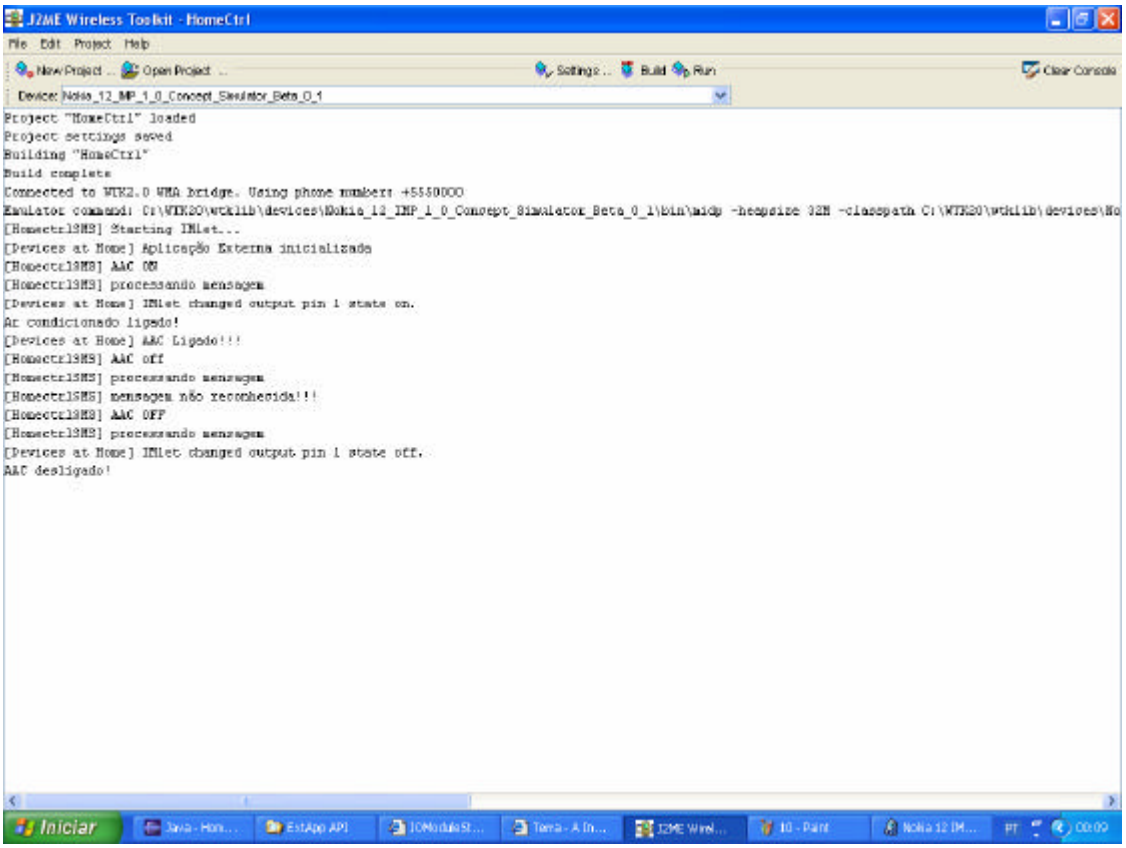

Figura 22: Arquivo de Log mostrando última ação

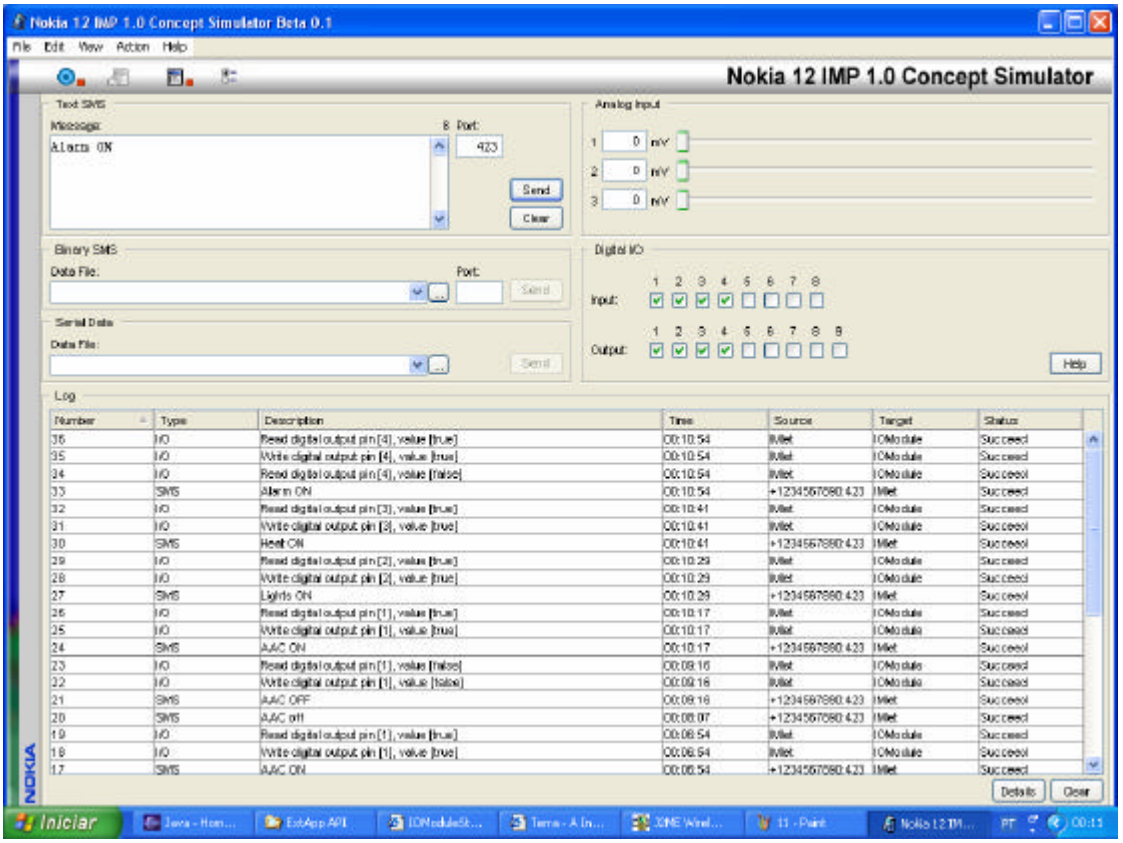

Figura 23: Todos os dispositivos Ligados

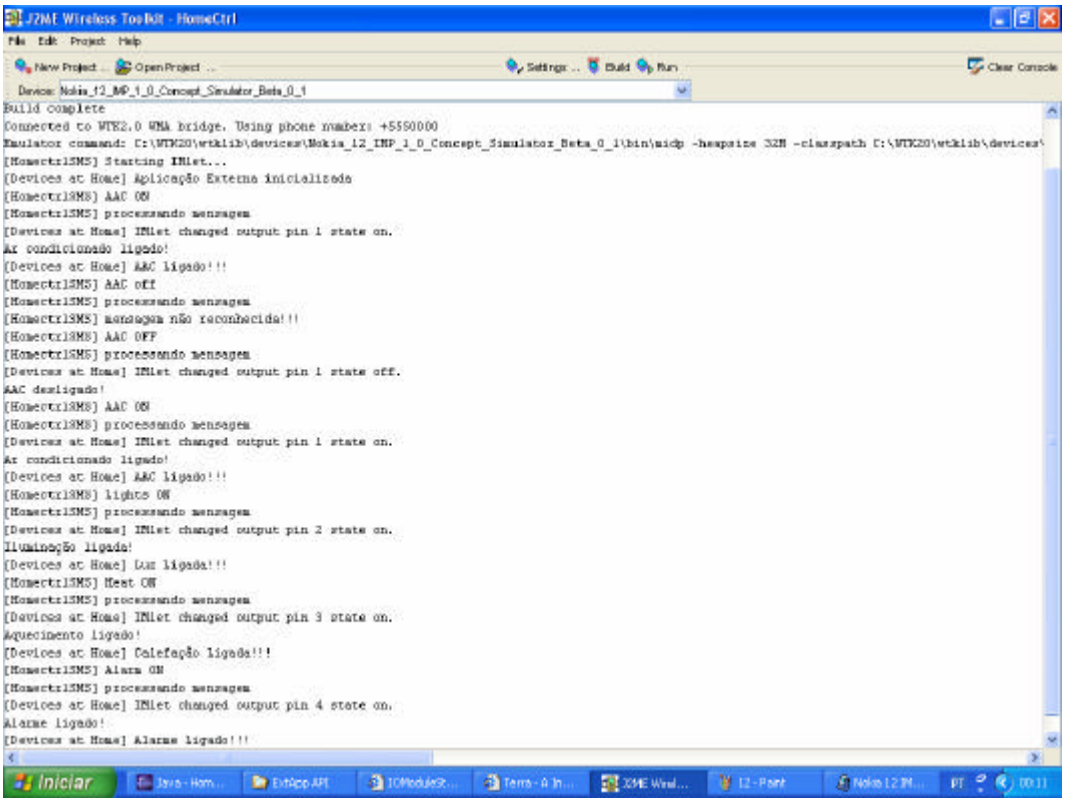

Figura 24: O histórico de Comandos usados.

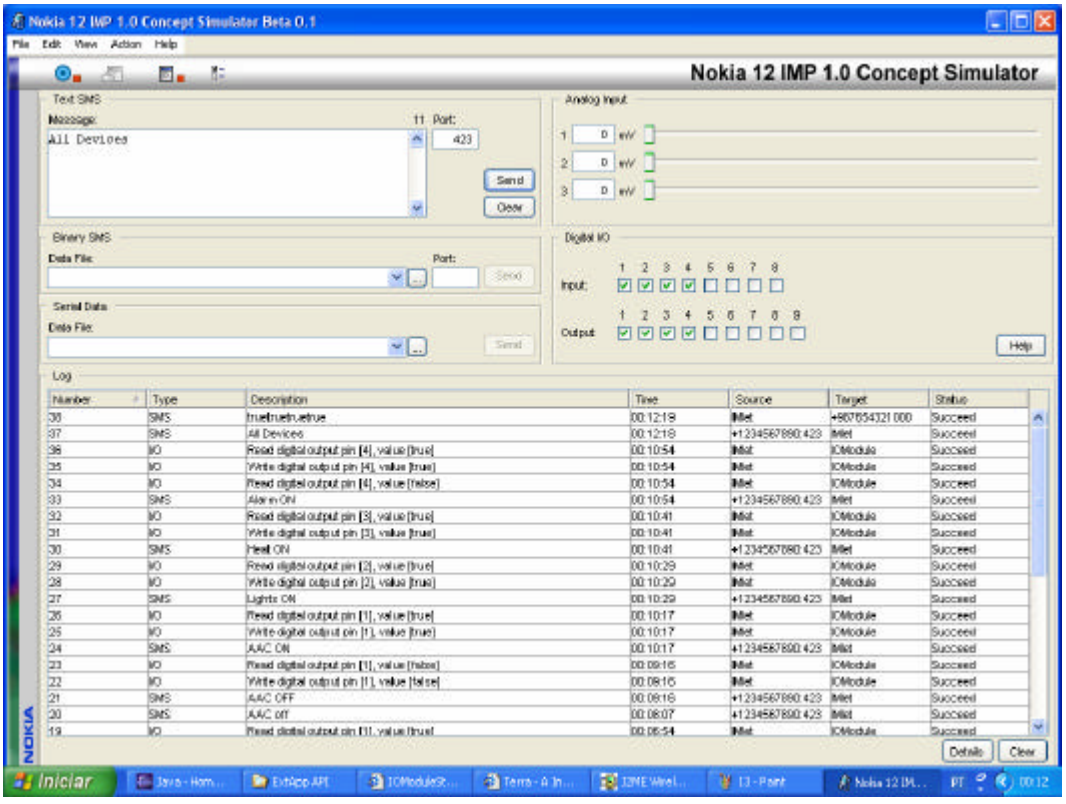

Figura 25: Comando All Devices pede informação sobre todos os aparelhos que se encontram ligados.

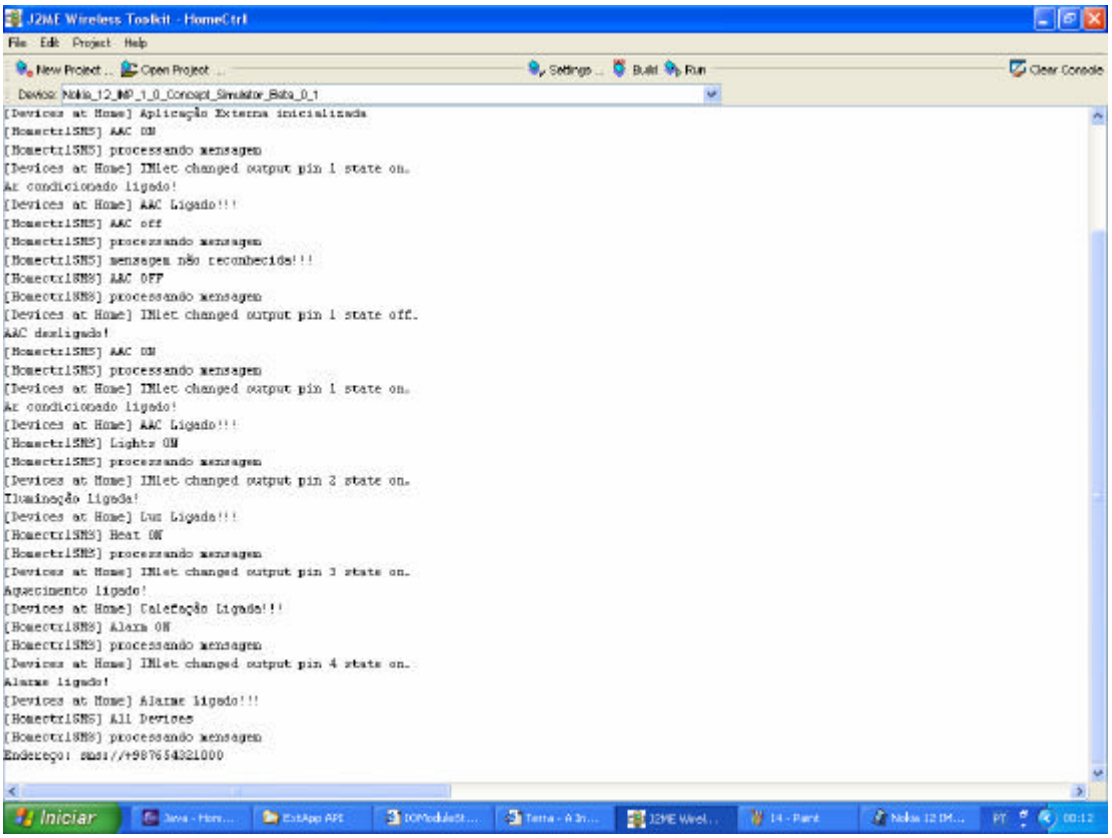

Figura 26: Arquivo de Log registrando pedido

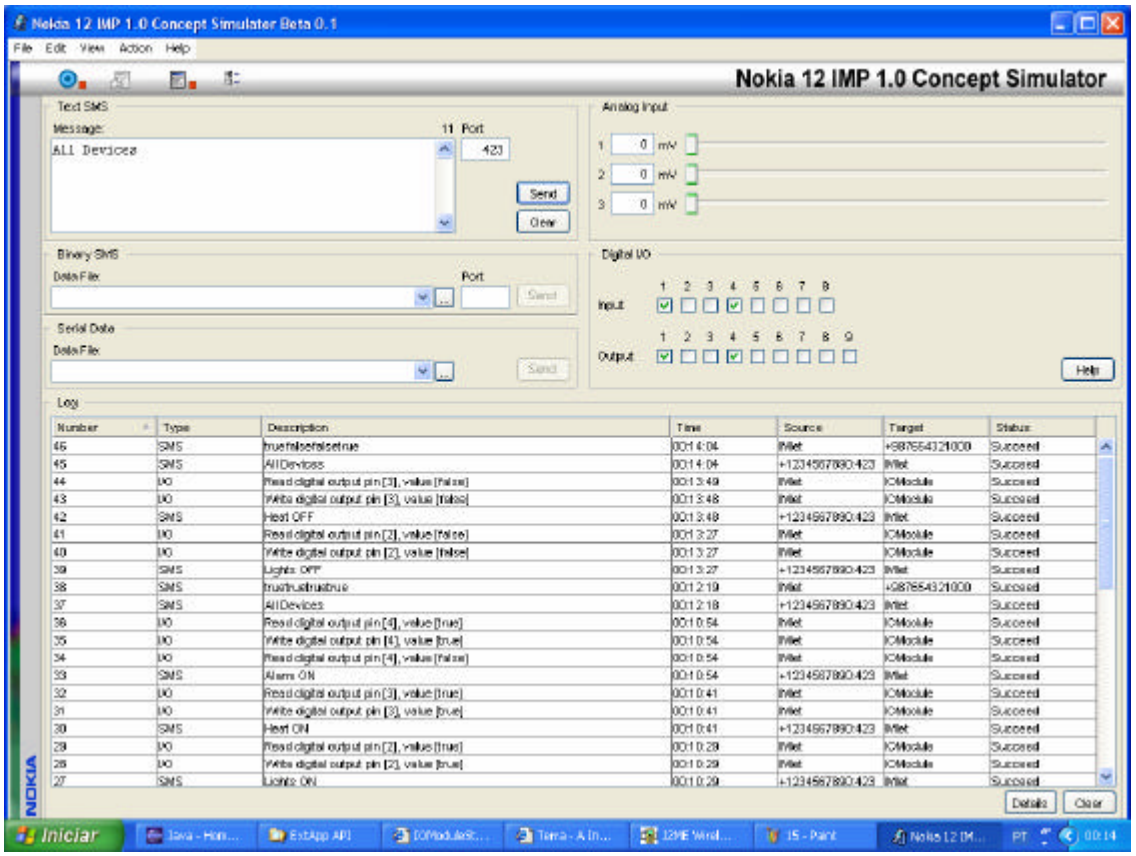

Figura 27: Resposta ao Usuário ao comando all devices

9. Conclusão

Nesse relatório apresentou-se o conceito geral do M2M, uma nova tecnologia de monitoramento e controle remoto. Para tanto, foram investigados um a um todos os aspectos transparentes a um usuário comum. Partindo desde a programação dos terminais remotos até o funcionamento completo da plataforma, passando por configurações de Gateway e da Aplicação Externa.

Devido ao atraso no lançamento do Terminal Nokia 12, não foi possível efetuar todas as simulações desejadas. Apenas conseguiu-se testar estruturas do Gateway e do simulador do terminal. No entanto, foi de grande valia esse estudo inicial, pois tão logo se consiga um exemplar deste Terminal para a Universidade, qualquer pessoa estará apta a criar aplicações diretamente, pois encontrará todo o restante já encaminhado.

É interessante investir no domínio dessa tecnologia, pois ela permite o desenvolvimento de um número ilimitado de aplicativos. Num mercado de montadoras, praticamente sem produção de tecnologia como o brasileiro, conhecer uma plataforma que promete revolucionar a indústria é estar aberto ao futuro e a parcerias em diversos campos, como a tele-medicina.

Esse sistema demonstrou ter potencialidade para ser amplamente empregado no dia a dia. Operadoras interessadas em conquistar o mercado de dados deveriam estar atentas a ela. Isso porque o maior público alvo do M2M são empresas de produção de manufaturados e empresas que queiram melhorar seus processos.[18] Ou seja, o potencial de vendagem é incomensurável.

Referência Bibliográfica

[1] – OLIVEIRA, A. H. C., CORREIA, H. B., IIDA R.F., *Implementação de uma interface neutra cliente-servidor dedicada à simulação e dos módulos de sistemas móveis e circuitos elétricos*, Dissertação de Projeto Final de Graduação da Universidade de Brasília, Departamento de Engenharia Elétrica, setembro de 2002.

[2] – Java, como programar / H.M. Deitel P.J. Deitel; trad. Carlos Arthur Lang Lisboa. – 4.ed. – Porto Alegre : Bookman,2003.

[3] – http://www.wirelessmessging.org/media/term1.02.html

[4] – http://www.forum.nokia.com

[5] – http://developer.java.sun.com/developer/onlineTraining/

[6] – Nokia_12_AT_command_guide_ver_1_1.pdf [4]

[7] – Nokia_12_IMP_InstallationGuide_v1_0.pdf [4]

[8] – Nokia_12_IMP_Tutorial_v10.pdf [4]

[9] – Nokia_12_IMP_UsersGuide_v10.pdf [4]

[10] – Nokia_12_IMP_ProgrammingGuide_v10.pdf [4]

[11] – Nokia_12_integrators_manual_v1_0.pdf [4]

[12] – Nokia_12_product_specification_ver_1_1.pdf [4]

[13] – Nokia_12_user_control_mode_guide_ver_1_0.pdf [4]

[14] – Nokia_GSM_Connectivity_Terminal_Version_v1_0.pdf [4]

[15] – Nokia_M2M_SDK_and_GW_Trial_Installation_Guide_Issue7_0.pdf [4]

[16] – M2M_System_Protocol_v4_0.pdf [4]

[17] – Terminal_IDL_Reference_Guide_Issue3_0.pdf [4]

[18] – m2m_whitepaper_v2.pdf [4]

[19] – STEELE R., CHIN-CHUN LEE , GOULD, PETER, *GSM, cdmaOne and 3G Systems*,Wiley, 2001

[20] – Nokia M2M FAQ [4]

[21] – NOKIA M2M GATEWAY TRIAL VERSION [4]

Anexo 1

Código do Imlet utilizado na simulação:

```
// Home control program
package com.nokia.m2m.examples.j2me.homectrl;
/**
 * IMlet for commanding home devices through I/O pins and SMS
 * UnB - Dezembro de 2003
 * Projeto Final De Graduação
 * Diogo Duarte D'Alessandro
 */
import java.io.IOException;

import javax.microedition.io.Connector;
import javax.microedition.midlet.MIDlet;
import javax.microedition.midlet.MIDletStateChangeException;
import javax.wireless.messaging.MessageConnection;
import javax.wireless.messaging.TextMessage;

import com.nokia.m2m.imp.iocontrol.IOControl;

public class HomeCtrl extends MIDlet implements Runnable
{
	private String targetAddress;
	private boolean isRunning;
	private MessageConnection sendConn;
	private MessageConnection readConn;

	public HomeCtrl()
	{
	}
	protected void pauseApp()
	{
	}
	protected void destroyApp(boolean parm1) throws
MIDletStateChangeException
	{
		isRunning = false;
		// close message connection if open
		try
		{
			readConn.close();
		}
		catch (Exception ex)
		{
		}
	}

	protected void startApp() throws MIDletStateChangeException
	{
		isRunning = true;
		new Thread(this).start();

		//start reading messages and then digital i/o pin values
		boolean[] pinValues = new boolean[9];
		int[] analogValues = new int[3];
		IOControl io = IOControl.getInstance();

		for (int i = 1; i < 10; i++)
```

```
            {
                  try
                  {
                        boolean val = io.getDigitalPin(i);
                        if (i > 0 && i < 10)
                        {
                              pinValues[i - 1] = val;
                        }
                  }
                  catch (Exception ex)
                  {
                        System.out.println(ex.getMessage());
                  }
            }

            //read analog I/O Pin values
            for (int i = 1; i < 4; i++)
            {
                  try
                  {
                        int val = io.getAnalogPin(i);
                        if (i > 0 && i < 4)
                        {
                              analogValues[i - 1] = val;
                        }
                  }
                  catch (Exception ex)
                  {
                        System.out.println(ex.getMessage());
                  }
            }
      }

      public void run()
      {
            writeLog("Starting IMlet...");

            readSMS();

      }

      private void sendTextSMS(String text, String address)
      {
            TextMessage msg;
            sendConn = null;

            System.out.println("Endereço: " + address);

            try
            {
                  if (address != null)
                  {
                        targetAddress = address;
                  }
                  else
                  {
                        targetAddress = "sms://+987654321000";
                  }
                  // creating MessageConnection to target address
                  sendConn = (MessageConnection)
Connector.open(targetAddress);
```

```
                    // creating TextMessage to send
                    msg =
                          (TextMessage) sendConn.newMessage(
                                MessageConnection.TEXT_MESSAGE);
                    msg.setPayloadText(text);
                    // send message
                    sendConn.send(msg);
              }
              catch (Exception e)
              {
                    e.printStackTrace();
              }
              finally
              {
                    // close MessageConnection if open
                    try
                    {
                          sendConn.close();
                    }
                    catch (java.io.IOException ex)
                    {
                    }
              }
        }

        private void readSMS()
        {
              String lista = " ";
              boolean[] pinValues = new boolean[9];
              TextMessage msg = null;
              readConn = null;
              IOControl io = IOControl.getInstance();

              try
              {
                    targetAddress = "sms://:423";
                    // create MessageConnection to targetAddress
                    readConn = (MessageConnection)
Connector.open(targetAddress);
                    // wait for incoming messages
                    while (isRunning)
                    {
                          msg = (TextMessage) readConn.receive();
                          writeLog(msg.getPayloadText());
                          String text = msg.getPayloadText();
                          writeLog("processando mensagem");

                          if ("AAC ON".equals(text))
                          {
                                try
                                {
                                      io.setDigitalPin(1,true);
                                      pinValues[1] = io.getDigitalPin(1);
                                }
                                catch (Exception ex){}

                                System.out.println("Ar condicionado
ligado!");
                          }

                          else if ("AAC OFF".equals(text))
```

```
                                                {
                                                      try
                                                      {
io.setDigitalPin(1,false);
                                                         pinValues[1] =
io.getDigitalPin(1);
                                                      }
                                                      catch (Exception
ex){}

      System.out.println("AAC desligado!");
                                                }
                        else if ("Lights ON".equals(text))
                        {
                              try
                              {
                                    io.setDigitalPin(2,true);
                                    pinValues[2] = io.getDigitalPin(2);
                              }
                              catch (Exception ex){}

                              System.out.println("Iluminação ligada!");
                        }

                        else if ("Lights OFF".equals(text))
                                                {
                                                      try
                                                      {
io.setDigitalPin(2,false);
                                                         pinValues[2] =
io.getDigitalPin(2);
                                                      }

                                                      catch (Exception
ex){}

      System.out.println("Luzes desligadas!");
                                                }

                        else if ("Heat ON".equals(text))
                        {
                              try
                              {
                                    io.setDigitalPin(3,true);
                                    pinValues[3] = io.getDigitalPin(3);
                              }
                              catch (Exception ex){}

                              System.out.println("Aquecimento ligado!");
                        }

                        else if ("Heat OFF".equals(text))
                                                {
                                                      try
                                                      {
```

```
io.setDigitalPin(3,false);
                                                    pinValues[3] =
io.getDigitalPin(3);
                                                }

                                                catch (Exception
ex){}


      System.out.println("Aquecimento desligado!");
                                                }

                        else if ("Alarm ON".equals(text))
                        {
                              try
                              {
                                    if (io.getDigitalPin(4))
                                    writeLog("alarm is already on");
                                    else io.setDigitalPin(4,true);
                                    pinValues[4] = io.getDigitalPin(4);
                              }

                              catch (Exception ex){}

                              System.out.println("Alarme ligado!");
                        }

                        else if ("Alarm OFF".equals(text))
                        {
                              try
                              {
                                 io.setDigitalPin(4,false);
                                 pinValues[4] = io.getDigitalPin(4);
                              }

                              catch (Exception ex){}

                              System.out.println("Alarme desligado!");
                        }

                        else if ("All Devices".equals(text))
                        {
                              lista = "";
                              for (int i=1;i<5;i++)
                              {
                                    lista = (pinValues[i]) + lista;
                              }
                              sendTextSMS(lista, "sms://+987654321000");
                        }

                        else writeLog("mensagem não reconhecida!!!");


                  }
            }
            catch (IOException e)
            {
                  // check if exception is other than 'connection closed'
                  if (!e.getMessage().equals("connection closed"))
                  {
```

```
                        e.printStackTrace();
                }
        }
        finally
        {
                // close MessageConnection if open
                try
                {
                        readConn.close();
                }
                catch (java.io.IOException ex)
                {
                }
        }
    }

    private void writeLog(String textParam)
    {
        System.out.println("[HomectrlSMS] " + textParam);
    }
}
```

Agora a Aplicação Externa:

```
package com.nokia.m2m.examples.homectrl;
/*
 * External Aplication representig domestic devices such as Heat
 * , air conditionning
 * UnB - Dezembro de 2003
 * Projeto Final De Graduação
 * Diogo Duarte D'Alessandro
 */
import com.nokia.m2m.simulator.externalapp.ExternalAppControl;
import com.nokia.m2m.simulator.externalapp.SimulatorControlFactory;
import com.nokia.m2m.simulator.externalapp.iomodule.IOModuleStateMachine;
import
com.nokia.m2m.simulator.externalapp.iomodule.IOPinValueChangeEvent;
import
com.nokia.m2m.simulator.externalapp.iomodule.IOPinValueChangeListener;
import
com.nokia.m2m.simulator.externalapp.iomodule.PinNumberOutOfRangeException
;
import
com.nokia.m2m.simulator.externalapp.iomodule.PinValueOutOfRangeException;


public class HomeCtrl implements ExternalAppControl,
IOPinValueChangeListener,
Runnable
{
   private IOModuleStateMachine io;
      private boolean         isRunning;

   public HomeCtrl()
   {
   }
```

```
    public void stopApplication()
     {
          io.removeIOPinValueChangeListener(this);
          io = null;
          writeLog("Stopped");
         isRunning = false;
     }

       public void startApplication()
     {
          writeLog("Aplicação Externa inicializada");
          // create IOModuleStateMachine
          io = SimulatorControlFactory.getIOModuleStateMachine();
          // start listening changes in pin values
          io.addIOPinValueChangeListener(this);
       }


     private void writeLog(String msg)
     {
          System.out.println("[Devices at Home] " + msg);
     }

            public void ioPinValueChanged(IOPinValueChangeEvent e)
       {
            writeLog(e.getDescription());
            //start thread which sets the digital input on for the device
            Thread t = new Thread (new ChangePinValue(this,
e.getType(),e.getPin(),
            e.getOldValue(),2000));
            t.start();
       }

       class ChangePinValue implements Runnable
       {
            public int type;
            public int pin;
            public long value;
            public long wTime;
            public boolean bVal;
            public IOPinValueChangeListener listener;

            public ChangePinValue(IOPinValueChangeListener listener, int
type,
             int pin, long value, long wTime )
             {
                  this.type = type;
                  this.pin = pin;
                  this.value = value;
                  bVal = false;
                  if (value > 0)
                  {
                        bVal = true;
                  }
                  this.wTime = wTime;
                  this.listener = listener;
             }
            /* (non-Javadoc)
             * @see java.lang.Runnable#run()
             */
```

```
            public void run()
            {
                  boolean[] set = new boolean[9];
                  long[] sensor = new long[3];
                  try
                  {
                        Thread.sleep(wTime);
                        if (io == null)
                        {
                              return;
                        }
                        io.removeIOPinValueChangeListener(listener);

                        switch (type)
                        {
                              case
IOPinValueChangeEvent.ANALOG_INPUT_PIN_VALUE_CHANGE:
                                    switch (pin)
                                    {
                                     case 1: writeLog("Temperatura
do AAC (°C): " + value/100);
                                     break;

                                     case 2: writeLog("Luminosidade
(W) : " + value/10);
                                     break;

                                     case 3: writeLog("Temperatura
Ambiente (°C): " + value/10);
                                     break;
                                    }

                              case
IOPinValueChangeEvent.DIGITAL_OUTPUT_PIN_VALUE_CHANGE:
                                    switch (pin)
                                        {
                                                case 1: set[pin] =
io.getDigitalOutput(pin);

io.setDigitalInput(pin,set[pin]);
                                                      if (set[pin])
                                                      writeLog("AAC
Ligado!!!");
                                                break;

                                                case 2: set[pin] =
io.getDigitalOutput(pin);

io.setDigitalInput(pin,set[pin]);
                                                      if (set[pin])
                                                      writeLog("Luz
Ligada!!!");
                                                break;

                                                case 3: set[pin] =
io.getDigitalOutput(pin);

io.setDigitalInput(pin,set[pin]);
                                                      if (set[pin])

writeLog("Calefação Ligada!!!");
```

```
                                                      break;
                                                case 4: set[pin] =
io.getDigitalOutput(pin);
io.setDigitalInput(pin,set[pin]);
                                                            if (set[pin])
writeLog("Alarme Ligado!!!");
                                                      break;
                                          }
                        }
                        io.addIOPinValueChangeListener(listener);
                        }
                  catch (Exception ex){}

                  }
                  // TODO Auto-generated method stub

            }
      /* (non-Javadoc)
       * @see java.lang.Runnable#run()
       */
      public void run()
      {
            // TODO Auto-generated method stub

      }
      public void setAnalogInput(int pin,long value) throws
PinNumberOutOfRangeException,
               PinValueOutOfRangeException
               {
               }
       }
```

Anexo 2

Configuração dos Softwares

Todos os softwares utilizados (simulador Nokia 12 e Nokia M2M Gateway Trial Version) estão disponíveis gratuitamente no Fórum Nokia, mediante registro. O cadastro é grátis.

Para instalá-los deve-se ter também instalado o J2SDK 1.4.2 ou superior. Para rodar o simulador é necessário ter também o Wireless Toolkit. Estes programas estão disponíveis no site da Sun http://java.sun.com.

Esses programas se integram automaticamente no computador.

O gateway na versão 1.0 só funciona com o simulador no modo CSD. A porta acessada pelo Gateway, originalmente 9999 deve ser substituída pela 900, para encontrar o servidor Corba que por default se conecta a essa porta.

Para criar um imlet, deve-se usar a guia new project no WTK, para criar as pastas onde serão salvos e compilados os arquivos Java que forem criados. Na guia settings deve-se configurar a guia required com MIDP 1.0, cldc 1.0 e na guia midlet deve-se configurar a guia classe, com o endereço do pacote e a classe do imlet, no caso com.nokia.m2m.examples.j2me.homectrl.HomeCtrl.

Os documentos relatando como programar o Nokia 12 não estão no fórum nokia e sim na pasta doc do simulador.