UNIVERSIDADE DO EXTREMO SUL CATARINENSE – UNESC

CURSO DE CIÊNCIA DA COMPUTAÇÃO

FELIPE RONCHI BRIGIDO

TENORIUM: UM *MIDDLEWARE* PARA DESENVOLVIMENTO EM DISPOSITIVOS MÓVEIS COM PYTHON PARA SYMBIAN SERIES 60

CRICIÚMA, JULHO DE 2009

FELIPE RONCHI BRIGIDO

TENORIUM: UM *MIDDLEWARE* PARA DESENVOLVIMENTO EM DISPOSITIVOS MÓVEIS COM PYTHON PARA SYMBIAN SERIES 60

Projeto de Pesquisa do Trabalho de Conclusão de Curso apresentado ao Curso de Ciência da Computação da Universidade do Extremo Sul Catarinense.

Orientador: Prof. Msc. Paulo João Martins

CRICIÚMA, JULHO DE 2009

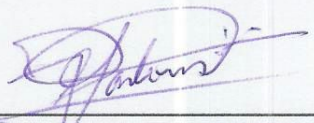
FELIPE RONCHI BRIGIDO

Tenorium: Um middleware para desenvolvimento em dispositivos móveis com Python para Symbian Series 60.

Submetido ao corpo docente do Curso de Ciência da Computação da Universidade do Extremo Sul Catarinense como um dos requisitos para obtenção do grau de Bacharel em Ciência da Computação.

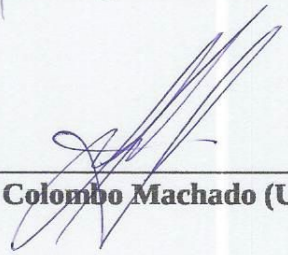
Prof. MSc. Rogério Antônio Casagrande
Coordenador Adjunto do Curso de Ciência da Computação

Banca Examinadora:



Prof. MSc. Paulo João Martins (UNESC)
Orientador

Profa. MSc. Cristiane Raquel Woszezenki (UNESC)



Esp. Fabrizio Colombo Machado (UNESC)

AGRADECIMENTOS

Primeiramente meus agradecimentos aos meus pais que sempre me apoiaram em tudo. Em segundo lugar, aos meus amigos que conheci na faculdade. Entre tantos merece destaque: José Márcio Cassetari Jr. por ser um grande amigo e estar ao meu lado em boas e difíceis horas. Fernando Gonçalves, também um amigão, sempre prestando ajuda a mim e ao Márcio. Rodrigo Spillere, grande companheiro de congressos. Não posso deixar de mencionar Dirceu Pereira Tiegs, por ter me apresentado a linguagem Python, base de todo este trabalho.

Sou muito grato também a Gabriele da Silveira do Nascimento, por aturar minhas ausências durante a redação deste trabalho com muita paciência e sempre estar por perto para me dar forças.

Agradecimentos seguem ao Sr. Manoel Mendes, nosso educador e a todos que estiveram em contato comigo durante minha passagem nesta universidade.

Por fim, contudo não menos importante, ficam meus agradecimentos a Deus, por me proporcionar tantas bênçãos e este maravilhoso dom de lidar com tecnologia que tenho.

NÃO CONSEGUIMOS ENCONTRAR A RESPOSTA
PARA TODOS OS NOSSOS PROBLEMAS,
AS QUE ENCONTRAMOS NOS LEVARAM
A FORMULAR NOVAS QUESTÕES.

SENTIMO-NOS HOJE TÃO CONFUSOS
COMO ANTES. ACREDITAMOS, ENTRETANTO,
QUE ESTAMOS CONFUSOS EM UM NÍVEL
MAIS ALTO E SOBRE COISAS MAIS IMPORTANTES.

Cartaz de alerta aos visitantes, encontrado
na entrada do departamento de matemática
de uma universidade escandinava.

RESUMO

A otimização do processo de desenvolvimento de um aplicativo é sempre uma peça chave para acompanhar a velocidade de evolução das tecnologias. Antigamente, o esforço era voltado para desenvolver um único código para várias arquiteturas de computadores, depois voltou-se para ferramentas que geravam e testavam código. Com o passar do tempo, dispositivos portáteis vem sendo introduzidos no mercado e há uma necessidade de ferramentas que auxiliem o desenvolvedor na tarefa de produzir aplicativos adequados a estes dispositivos. Este documento mostra que através de uma conexão de rede é possível a existência de uma ferramenta capaz de unir a praticidade de programar em um computador e a utilidade de depurar diretamente no dispositivo móvel de maneira instantânea.

ABSTRACT

The optimization of the application's developing process is always a key to follow the technology change speed. Previously, the effort was directed to develop a single code for multiple architectures of computers, then turned to tools that generate and test code. Over time, portable devices have been introduced to the market and there is a need for tools that help the developer in the task of producing appropriate applications for these devices. This paper shows that through a network connection is possible that there is a tool capable of uniting the practical to use a computer and use to directly and instantaneously debug on the mobile device.

LISTA DE ILUSTRAÇÕES

LISTA DE SIGLAS

ADI	Ambiente de Desenvolvimento Integrado
PyS60	Python para series 60
TCP/IP	Protocolo de comunicação da Internet
Pydoc	Documentação embutida em scripts python
ISO	*International Organization for Standardization*
OSI	*Open Systems Interconnection*
L2CAP	*Logical Link Control and Adaptation Protocol*
RFCOMM	*Radio Frequency Communication*
SMTP	*Simple Mail Transfer Protocol*
FTP	*File Transfer Protocol*
DNS	*Domain Names Service*
SSH	*Secure Shell*
HTTPS	*HyperText Transfer Protocol Secure*
IP	*Internet Protocol*
TCP	*Transmission Control Protocol*
MAC	*Media Access Control*
RS-232	Protocolo de comunicação serial
RPC	*Remote Procedure Call*
OO	Orientação a Objetos
PDA	*Personal digital assistants*
LCD	*Liquid Cristal Display*

SUMÁRIO

1 INTRODUÇÃO

Na construção de aplicativos para dispositivos móveis, tem-se um contra-tempo quando se trata de depuração. Isso deve-se ao trabalho de escrever o aplicativo, transferi-lo para o dispositivo, para depois verificar os erros, e caso estes existam, reescrever, transferir e verificar.

Em alguns casos, para facilitar este processo, são utilizados emuladores. Estes criam um dispositivo virtual para testar tais programas. Entretanto, pelo fato dos emuladores serem programas equivalentes aos dispositivos móveis, por mais que a depuração neste esteja completa, ainda corre-se o risco de haver problemas quando executar o aplicativo diretamente no dispositivo.

Em certos casos, como o do Python, tem-se a vantagem da linguagem multi-plataforma, onde os módulos para funcionalidades básicas, como os que tratam estruturas de dados, entrada e saída padrão, entre outros, são os mesmos para qualquer plataforma, tornando um script escrito para a plataforma Linux potencialmente executável na plataforma Symbian. Ou seja, grande parte de um *script* Python desenvolvido para Symbian pode ser depurado em outras plataformas. Contudo existem módulos que não são portáveis (Ex: módulo de interface gráfica, módulo de acesso as mensagens SMS, entre outros.), onde módulos genéricos precisam ser desenvolvidos para uso em depuração, dificultando a eficácia do processo de desenvolvimento para dispositivos móveis.

Focando no caso do Python para Series 60, verifica-se que este possui um console remoto, chamado Bluetooth Console. Este permite que o dispositivo móvel conecte-se a um computador e forneça um console *shell* para testes. Com esta ferramenta é possível testar linhas de código antes de adicioná-las ao aplicativo, porém, testar um aplicativo digitando-o linha a linha no console vai tornando-se difícil a medida que o código do aplicativo aumenta.

Como saída viável para todos os entraves, surge a idéia de um aplicativo que conectado ao celular, transfira o *script* desenvolvido para o mesmo e o execute, permitindo a depuração em tempo de desenvolvimento.

1.1 OBJETIVO GERAL

Construir um *middleware* para Ambientes de Desenvolvimento Integrado (ADI) que conecte-se ao dispositivo móvel e execute *scripts* em linguagem Python para dispositivos móveis sobre o sistema operacional Symbian Series 60, com propósito de depuração em tempo de implementação.

1.2 OBJETIVOS ESPECÍFICOS

a) Identificar ferramentas de controle Bluetooth e TCP/IP que sejam multi-plataforma, para estabelecer a conexão entre plugin e dispositivo;
b) Entender o funcionamento do Console interativo Bluetooth do PyS60;
c) Avaliar a necessidade de construção de um Console Interativo voltado apenas para o middleware;
d) Implementar o middleware que servirá de base a criação de plugins específicos para cada ADI;

1.3 JUSTIFICATIVA

Baseando-se na definição do problema, implementar usando apenas um editor de textos para escrita do *script*, com um transmissor de arquivos via bluetooth torna-se uma tarefa árdua, pois neste caso a depuração do aplicativo é feita diretamente no dispositivo móvel, este que em sua maioria apresenta pouca usabilidade, como telas minúsculas e teclados complicados. Contando também que os resultados produzidos pela aplicação no dispositivo móvel não são tão legíveis quanto no computador.

Com o uso de emuladores há uma melhora no tempo de produção devido a não haver necessidade de enviar o *script* manualmente para o dispositivo, bastando apenas uma chamada pelo emulador. Os resultados produzidos durante a depuração do aplicativo desenvolvido podem ser avaliados com mais clareza, já que o software está no mesmo ambiente no qual ocorre a implementação. Contudo, pelo fato de o emulador ser um programa equivalente ao dispositivo, há o risco de uma depuração feita no mesmo apresentar resultados diferentes de uma depuração feita diretamente no dispositivo móvel.

O Bluetooth Console nos permite executar linhas de código digitadas no computador instantaneamente no dispositivo móvel, permitindo pequenos testes mas inviabilizando uma depuração completa de um *script*. A idéia central deste projeto consiste em construir uma ferramenta que utilize a interface do Bluetooth Console para transferir o script a ser depurado e executá-lo em tempo real. Esta deve enviar ao dispositivo móvel comandos em tempo real, tendo em mente que o Bluetooth Console foi desenvolvido para operar com um comunicador serial (como o HyperTerminal®) e portanto precise de alguns ajustes ou até mesmo uma re-implementação para que se obtenha máximas velocidade e compatibilidade.

Também há de se verificar que embora este projeto desenvolva uma ferramenta escrita em e para Python, usando conexão Bluetooth e para dispositivos Symbian, não significa que não possam ser desenvolvidas outras ferramentas similares para outras linguagens de programação, utilizando-se de outros tipos de conexões e voltados para outros dispositivos móveis.

2 REDES DE COMPUTADORES

Formada por dispositivos com capacidade de processamento, uma rede de computadores interligada através de uma tecnologia em comum, (TANEMBAUN, 2003) trocando e armazenando informações e compartilhando recursos. O sistema de comunicação interliga os dispositivos por meio de uma topologia, utilizando meios de transmissão e protocolos que organizam essa comunicação.

2.1 PADRÕES DE COMUNICAÇÃO

Uma grande dificuldade durante um período da evolução dos sistemas de computadores foi percebida, uma vez que, os fabricantes de computadores dispunham de protocolos e/ou arquiteturas proprietárias. Isso forçava os usuários a optar por computadores de um mesmo fabricante, caso precisassem interconectar computadores em uma única rede. Devido a essa grande dificuldade, os usuários incentivaram o processo de padronização. O pretendido, foi definir uma arquitetura padrão que fosse aberta e pública. Com esta padronização, teriam a vantagem de permitir com que máquinas de fabricantes diferentes se comunicassem, e além disso, novos fabricantes poderiam introduzir seus produtos aderindo a essas regras.

Considerando isto, os projetos de rede foram organizados em camadas hierárquicas, onde cada camada oferece funções e serviços para as superiores. Cada camada de uma máquina se comunica com a camada de mesmo nível em outra máquina, utilizando um protocolo inerente à camada deste nível. Isto é definido como comunicação virtual. Estes protocolos são regras e convenções pertinentes a esta camada. Fisicamente a comunicação ocorre na vertical entre camadas adjacentes até chegar ao meio físico, onde realmente ocorre a

comunicação horizontal, que interliga duas máquinas distintas. Pode-se então dizer que, a arquitetura da rede é um conjunto de camadas de protocolos.

2.2 MODELO DE REFERÊNCIA OSI

Partindo do princípio de camadas, um modelo de referência para a interconexão de sistemas abertos foi definido em 1978 pela *International Organization for Standardization* (ISO). Neste projeto, o *Open Systems Interconnection* (OSI), apresentado na figura 1, foi proposto como sendo um modelo de 7 camadas (Físico, Enlace, Rede, Transporte, Sessão, Apresentação e Aplicação) de referência para a arquitetura dos protocolos de redes de computadores, com a finalidade de coordenar padrões de interconexão entre sistemas.

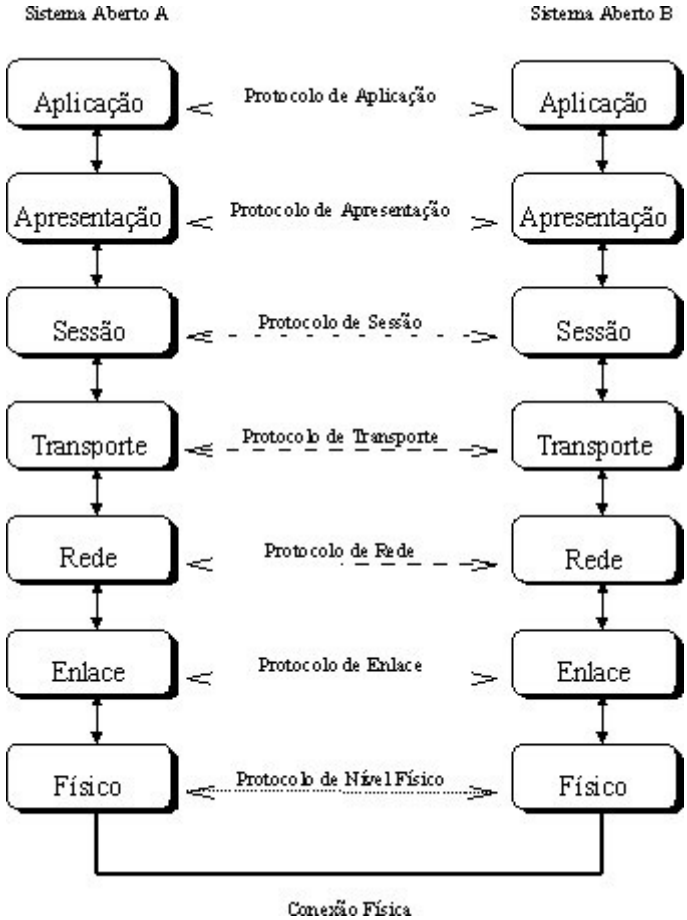


Figura 1: Modelo de referência OSI

2.2.1 Camada Física

É nesta camada, onde são definidas as características mecânicas, elétricas funcionais e de procedimentos de comando e gerência das conexões físicas. Esta camada tem como função, transmitir os dados (cadeia de *bits*) mediante o meio de transmissão, sem se qualquer relação com o significado dos mesmos. Está abaixo de todas as outras e é ligada diretamente como o meio de transmissão que interconecta os sistemas. (TANEMBAUN, 2003)

2.2.2 Camada de Enlace

Detecção e correção de erros ocorridos no nível físico, tornando os dados confiáveis para uso na camada de rede, é a principal função desta camada. Ela também assume a tarefa de criação e reconhecimento dos limites dos quadros, visto que a camada física transmite os dados brutos sem se preocupar com o significado e estrutura dos mesmos colocando *bits* especiais no início e fim dos quadros. (TANEMBAUN, 2003) Esta camada resolve também os problemas de quadros perdidos, danificados e repetidos. Também é responsável pelo controle de fluxo, balanceando a velocidade e o espaço de *buffer* entre o transmissor e o receptor.

2.2.3 Camada de Rede

Trata em especial, o chaveamento e roteamento de pacotes, através de sistemas intermediários, interligando diferentes endereços em nível de rede. Esta camada oferece dois

tipos de serviços: serviço orientado à conexão e não orientado à conexão. (TANEMBAUN, 2003)

Quanto ao serviço orientado à conexão, podemos distinguir três diferentes fases: estabelecimento de conexão, transferência de dados e encerramento de conexão. Isto torna as rotas previamente definidas por meio de tabelas estáticas na maioria das vezes, em outras palavras, a rede possui um mapeamento destes caminhos por onde trafegam os pacotes. No estabelecimento da conexão, um caminho é definido entre a origem e o destino, alocando os recursos de rede, virtualizando uma conexão direta. Uma vez estabelecida a conexão, os pacotes são transferidos em seqüencia através deste caminho. Os pacotes chegam ao destino, na mesma ordem em que são enviados pela origem, uma vez que o caminho é único. Após a informação ser completamente transferida, encerra-se a conexão.

Já o serviço não orientado à conexão não aloca recursos de rede e portanto o caminho não é previamente definido. Cada pacote tem seu destino especificado independentemente, podendo seguir caminhos distintos e não garantindo o seqüenciamento dos mesmos. Este serviço é mais tolerante em relação a ocorrência de falhas na rede, pois está focada na distribuição de dados em tempo real, descartando automaticamente dados perdidos e/ou atrasados.

2.2.4 Camada de Transporte

Permite a multiplexação (várias conexões de transporte utilizando uma mesma conexão de rede), garantindo que sobre uma mesma camada de rede sejam utilizadas várias aplicações simultaneamente. Para tal, o nível de transporte identifica em cada pacote o endereço da respectiva aplicação. Este endereço de aplicação é denominado porta de conexão.

Essa técnica de multiplexação é utilizada quando o tráfego gerado pela conexão de transporte é insuficiente para ocupar toda a capacidade de uma conexão de rede.

2.2.5 Camada de Sessão

Nesta camada são estabelecidas, gerenciadas e encerradas as sessões entre as aplicações e possui mecanismos para sincronização das tarefas. Esta camada permite o estabelecimento de sessões entre usuários de máquinas distintas. Também gerencia o controle de tráfego podendo ser *full-duplex* (nos dois sentidos e ao mesmo tempo) ou *half-duplex* (em um sentido de cada vez). Outro serviço desta camada é o gerenciamento de *token*, no qual determina que quem estiver com posse do mesmo, é que terá o direito de executar as operações. Este gerenciamento serve para evitar que os lados não executem operações ao mesmo tempo. Esta camada também controla a sincronização, que segmenta a transmissão de uma informação muito longa, para em caso de erros durante a transferência, seja possível rejeitar apenas o segmento corrompido e manter as partes corretas. Sem este, a cada erro durante uma longa transmissão tudo seria abortado e retransmitido desde o início. (TANEMBAUN, 2003)

2.2.6 Camada de Apresentação

Esta camada permite que um usuário tenha acesso a rede, utilizando de serviços como o *World Wide Web* para acesso a páginas, SMTP para e-mail, FTP, entre outros. (FOROUZAN, 2003)

2.3 MODELO TCP/IP

Sucessor da ARPANET, este modelo foi desenvolvido com o objetivo de conectar múltiplas redes sem que houvesse descontinuidades. O grande desafio era desenvolver uma rede em que a conexão entre dois pontos permanecesse estável até que um dos pontos fosse desativado, não importando se algum segmento da linha entre os mesmos falhasse. (TANEMBAUN, 2003)

Possui relação com o modelo OSI, porém é segmentado em 4 camadas apenas. No modelo TCP/IP, as camadas de aplicação, apresentação e sessão do modelo OSI são representadas pela camada de apresentação e camada de acesso a rede representa as camadas de enlace e física. As camadas de transporte e rede do modelo OSI são representadas pelas camadas de transporte e Internet respectivamente.

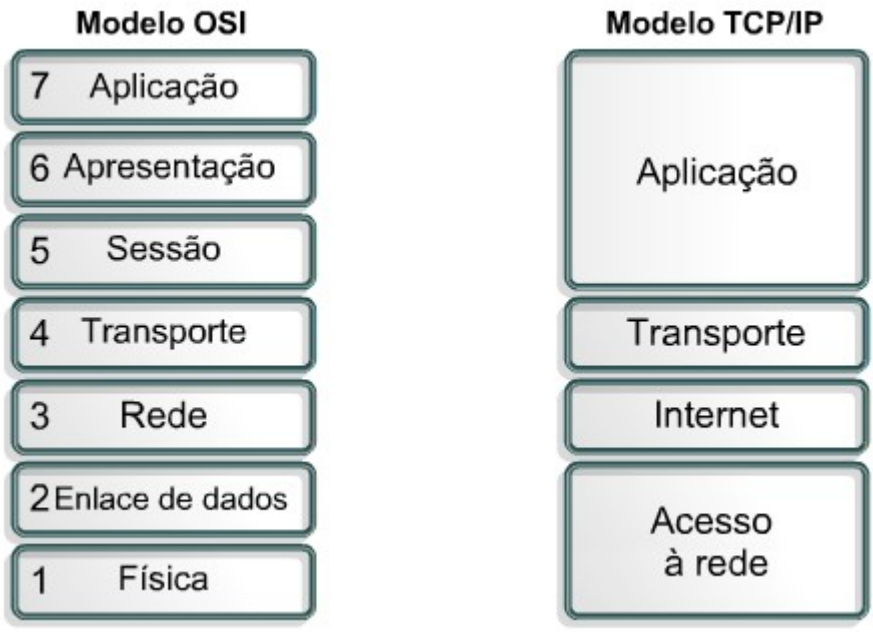


Figura 2: Modelo TCP/IP

2.3.1 Camada de Aplicação

É a camada de nível mais alto deste modelo, compreendendo os protocolos de alto nível, que tem contato direto com o usuário. Exemplos de protocolos seguem os do modelo OSI, são eles: FTP (*File Transfer Protocol*; Protocolo de Transferência de Arquivos), SMTP (*Simple Mail Transfer Protocol*; Protocolo de Transferência de Correio Simples), DNS (*Domain Names Service*; Serviço de Nomes de Domínio). (TANEMBAUN, 2003)

Neste modelo, as camadas de sessão e apresentação são quase que excluídas, pois sua necessidade é pouco percebida. Apenas algumas de suas funcionalidades são utilizadas, como a criptografia, por exemplo, e estas aparecem implementadas nesta camada, como o SSH e o HTTPS.

2.3.2 Camada de Acesso à Rede

Como dito anteriormente, engloba as camadas física e de enlace do modelo OSI, englobando as tarefas de transmissão e controle dos dados. Ex: Ethernet, Wifi, entre outros.

2.3.3 Camada de Internet

Tem por objetivo, segmentar os dados em pacotes, garantindo a sua entrega ao destinatário sem a perda de nenhuma das partes. A ordem que estes pacotes chegam ao destino e o caminho que eles percorrem é indiferente, desde que toda a informação saia de um ponto e seja entregue completamente em outro. O protocolo IP (*Internet Protocol*; Protocolo de Internet) é responsável por essa tarefa. (TANEMBAUN, 2003)

2.3.4 Camada de Transporte

Assim como no modelo OSI, esta camada é responsável pela ligação direta entre duas entidades, criando um canal onde estas podem trocar dados. Esta camada define dois principais protocolos, o TCP (*Transmission Control Protocol*; Protocolo de Controle de Transmissão) que permite uma conexão confiável. Para tal, este parte os dados em trechos e repassa para a camada abaixo (Internet) uma a uma, aguardando a resposta positiva para o envio do próximo trecho. Na ponta que recebe, o TCP trata de reunir os trechos na ordem correta e efetuar a entrega para o processo correspondente. Esta camada também é responsável por manter o fluxo na velocidade correta, para evitar que um receptor lento fique sobrecarregado. (TANEMBAUN, 2003)

O UDP (*User Datagram Protocol*) permite uma entrega de dados sem estabelecer uma conexão fixa. Recomendado por processos onde o desempenho é mais importante que a ordem ou a garantia de entrega, tais como serviços de vídeo e voz. (TANEMBAUN, 2003)

2.4 BLUETOOTH

Iniciado em 1994 pela Ericsson, trata-se de um consórcio entre várias empresas de comunicação (Nokia, IBM, Intel e Toshiba), com o propósito de desenvolver um padrão de comunicação sem fio para dispositivos com baixa capacidade de processamento. (FOROUZAN, 2003)

Seu modelo de camadas é representado por: Camada de Rádio, que implementa a parte física, como a camada de acesso a rede do modelo TCP/IP; *Baseband*, equivalente a sub-camada MAC; e a camada L2CAP que controla o fluxo de dados Bluetooth. (TANEMBAUN, 2003)

Sobre o protocolo L2CAP funciona o protocolo RFCOMM, responsável por definir uma porta serial virtual, transmitindo dados binários e emulando os sinais de controle do padrão RS-232.

2.5 CONCLUSÃO

Fica evidente, segundo os estudos sobre comunicações, que a ferramenta proposta neste documento, trabalha com a camada de transporte do modelo TCP/IP, mais especificamente com o protocolo TCP e com o Protocolo RFCOMM da tecnologia Bluetooth.

3 MIDDLEWARE

Ferramenta que garante interoperabilidade, ou seja, inter-operação. É uma camada de software que liga dois ou mais softwares. De grande utilidade em sistemas distribuídos, principalmente quando heterogêneos, ou seja, formada por dispositivos com diferentes arquiteturas. Inicialmente utilizado com simples componentes distribuídos, definindo seu padrões de interoperação. Mas com o passar do tempo, estes aplicativos começaram a integrar sistemas inteiros.(MACIEL; ASSIS, 2004) Middlewares tem grande facilidade em interagir com sistemas implementados em diferentes linguagens, tornando-os ótimas ferramentas para a migração de tecnologias, sendo esta total mas principalmente em migrações parciais. Também torna possível o intercâmbio de informações entre grandes empresas e por isso é peça fundamental em BI (*Busines Inteligence*).

Em seus requerimentos básicos, está a conexão de rede, já que se trata de interligar diferentes dispositivos computacionais. Coordenação é sempre requerida como funcionalidade para estes tipos de software, principalmente quando trabalha com comunicação assíncrona. O contexto de segurança, além de evitar o vazamento de informação, a correta transmissão da mesma deve ser observada, visto que middlewares em sua maioria, trabalham em camadas de rede que não tratam este problema. Escalabilidade também deve ser estudada neste tipo de software, pois esta é muito suscetível ao crescimento. Como citado anteriormente, middlewares garantem a interação entre sistemas distribuídos e estes são de diferentes hardwares/softwares, suporte a heterogeneidade é vital para estes aplicativos. (EMMERICH, 2005) Além disso é importante lembrar que devem ser aplicações com altos níveis de desempenho e principalmente flexibilidade.

Os tipos segundo Emmerich (EMMERICH, 2005) são:

Transacional – Suporta transações entre pontos distribuídos. Estes tipos de middleware possuem uma orientação cliente-servidor. Caracterizam-se por ser tolerante a falhas, mas tem um ponto negativo com relação ao desempenho, pois ficam à mercê de respostas entre os nós. Encontrados em sistemas com replicação de bancos de dados.

Orientados a mensagem – Efetua troca de mensagens entre cliente/servidor. Suporta comunicação assíncrona com mais naturalidade. Desempenho melhor em relação ao Transacional pois suporta outros processamentos enquanto o servidor manipula sua requisição. Pode ser tolerante a falhas, a maneira clássica de garantir isso, é montando uma fila de mensagens, para caso haja uma falha de comunicação, possa-se re-estabelecer o estado original da troca de mensagens.

Procedurais – Conhecidos como RPCs, dão suporte a chamadas de procedimentos pela rede. Levam a idéia da Programação Procedural ao nível distribuído, onde uma cadeia de rotinas estruturadas em um nó servidor está a serviço, recebendo parâmetros processando-os e devolvendo os resultados aos nós clientes. É de grande utilidade quando se trata de diferentes linguagens, pois sua assinatura de métodos permanecesse inalterada diante dos mesmos.

Orientados a Objetos – Derivam dos Procedurais e trazem todo o suporte de OO para o ambiente distribuído. Oferecendo além de procedimentos, instâncias inteiras de objetos encapsulando informações e tornando mais natural a integração entre pontos.

4 DISPOSITIVOS MÓVEIS

Os telefones celulares evoluíram bastantes desde a sua criação e deixaram de ser artigos de luxo e simples mecanismos de comunicação por voz. Transformaram-se em aparelhos inteligentes, possuindo uma nova forma de comunicação de dados, que provêem acesso à Internet, agenda eletrônica e entretenimento em geral.

A necessidade de disponibilizar aplicações e informações em qualquer momento e lugar fez surgir uma linha de dispositivos móveis conhecidos com *PDA's* ou *Handhelds*. Esses dispositivos são considerados computadores de dimensões reduzidas, porém com uma grande capacidade computacional, considerando-se que se trata de um dispositivo móvel, e que oferecem uma grande variedade de funcionalidades. Possuem uma significativa quantidade de memória, telas com pontos sensíveis a toques e coloridas, mouse simulado por uma caneta (conhecida como *stylus*), modem, rede sem fio embutida. São, realmente, computadores que auxiliam nas mais diversas atividades que podem ser realizadas no dia-a-dia. (SMARTPHONE, 2008)

Há muitas similaridades e diferenças entre um dispositivo móvel e um PC. O importante requisito de mobilidade inerente aos dispositivos móveis faz com que estes sejam menores em todos os aspectos em comparação com um PC. Além disso, há aspectos da tecnologia de *hardware* que diferem muito das tecnologias de um PC. Um exemplo é o fato de operar com baterias, onde é preciso estar sempre atento ao consumo de energia por parte do *hardware* para obter mais tempo de funcionamento. As telas de LDC (*Liquid Cristal Display*) são aproximadamente 30 vezes menores, comparados a tela de um computador comum, diminuindo a quantidade de informação visualizada nas mesmas. Os teclados também são menores e possuem menos teclas, dificultando a entrada de dados no dispositivo.

Para reunir os recursos oferecidos pelos celulares e pelos PDA's em um único dispositivo, foram criados os *SmartPhones* ("telefones inteligentes" numa tradução livre do termo em inglês) (SMARTPHONE, 2008). O *SmartPhone* pode ser descrito como um aparelho celular com diversas funcionalidades extras tais como agenda telefônica, lista de tarefas, multimídia e sincronização com computador através de bluetooth e infravermelho. A idéia deste produto é misturar as funções do celular com os aplicativos de um PDA

4.1 COMPUTAÇÃO MÓVEL

A Computação Móvel representa um novo paradigma tecnológico que tem como objetivo principal prover ao usuário final acesso permanente a uma rede fixa ou móvel independente de sua posição física. É a capacidade de acessar informações em qualquer lugar e a qualquer momento (LOUREIRO et.al., 2003).

Este novo paradigma surge como uma quarta revolução computacional, antecedida pelos grandes centros de processamento de dados da década de 60, seguido do surgimento dos terminais nos anos 70, e as redes de computadores na década de 80 (MATEUS; LOUREIRO, 1998).

Segundo Loureiro et al. (2003), a Computação Móvel está se tornando uma área madura e parece destinada a se tornar uma tecnologia dominante no futuro. O mercado de dispositivos móveis, genericamente chamados de *handhelds*, que englobam telefones celulares, *palms*, PDAs, entre outros, cresce continuamente, sendo usados em aplicações que envolvem negócios, indústrias, escolas, hospitais, lazer. Enfim, é uma tecnologia já bastante difundida atualmente.

Uma característica importante deste paradigma, é a interação entre o mesmo e as diversas áreas da Computação como Sistemas Digitais, Arquitetura de Computadores,

Linguagens de Programação, Engenharia de Software, Interface Homem-Máquina, Compiladores, Banco de Dados, que possuem o papel importante de definir novas formas de uso da tecnologia de processamento e comunicação de dados (LOUREIRO et.al., 2003).

4.2 DESENVOLVIMENTO DE APLICAÇÕES PARA DISPOSITIVOS MÓVEIS

O desenvolvimento de aplicações para dispositivos móveis precisa levar em consideração as limitações de memória, a capacidade de processamento, o consumo de energia das baterias, a alta latência das redes de comunicações sem fios entre outros fatores limitantes dos dispositivos móveis abordados em tópicos anteriores. Isto se torna um conjunto de requisitos não funcionais que as aplicações desenvolvidas para estes dispositivos precisam levar em consideração ao serem projetadas. O fato de existir uma variedade muito grande de modelos, com processadores, *hardware*, arquiteturas e tecnologias diferentes é outro fator que precisa ser levado em consideração, pois limita o suporte que o aparelho pode dar a aplicação. (LEE et.al., 2005)

Uma das saídas para tais implicações de desenvolvimento são as linguagens de *scripts*, que possuem uma alta portabilidade em sua maioria.

4.3 LINGUAGENS DE *SCRIPT*

As linguagens de *script* são linguagens de computação executadas em um interpretador, onde o código fonte, *script*, é interpretado comando a comando cada vez que são executados. *SmartPhones* que possuem o sistema operacional *Symbian* podem adquirir suporte à linguagem de script Python através do projeto PyS60 (Python para series 60). Esta tecnologia oferece a oportunidade de desenvolver aplicativos para o *SmartPhone* sem a

necessidade de qualquer ferramenta adicional instalado no computador. Apenas é necessário o interpretador do Python instalado no *SmartPhone*.

4.4 PYTHON

Python é uma linguagem de programação de alto nível, interpretada, interativa, orientada a objetos e de tipagem dinâmica e forte, lançada por Guido van Rossum em 1991. Python também é multi-plataforma, pode operar em vários sistemas operacionais. (LUTZ; ASCHER, 2007)

Esta linguagem traz como ferramenta o console interativo. Trata-se de um console shell, que recebe linhas de comando em linguagem Python, as interpreta e imprime o resultado no próprio console. De grande utilidade para testes e desenvolvimento do aprendizado nesta linguagem, esta ferramenta também proporciona acesso ao chamado Pydoc, que é uma documentação embutida nos scripts Python. Os desenvolvedores de Python podem construir Consoles Interativos personalizados fazendo uso do módulo `code`, que possui vários métodos e classes específicos para trabalhar com cada aspecto do Interpretador Python.

4.4.1 PyS60

Para funcionar com eficiência em um ambiente com menos recursos de hardware, como é o caso dos SmartPhones com Symbian, foram feitas algumas modificações no interpretador. Isso deu inicio ao projeto PyS60 (Python para *Symbian Series 60*). Neste projeto há uma ferramenta chamada *bluetooth-console*, que permite ao programador conectar-se ao *SmartPhone* e através do aplicativo de comunicações como o *HyperTerminal*® é possível ter no computador um Console Interativo do PyS60. (SHEIBLE; TUULOS, 2007)

5 ENGENHARIA DE SOFTWARE

Para escrever programas, poder mante-los e evoluí-los é necessário uma certa disciplina, por mais que esse desenvolvimento seja efetuado por apenas um programador. Este programador terá em mente o problema a ser resolvido, o que fazer para tal e como fazer. Isto por mais simples que seja já segue o raciocínio de engenharia de software.

Engenharia de software não é limitada apenas a isso. ela também define toda a metodologia usada para o desenvolvimento, a documentação do mesmo, estudos de custo, entre outros aspectos da produção de software. Podemos entender a engenharia de software como sendo a área da ciência da computação que trata a construção de grandes sistemas de software, onde os componentes criados por um podem ser combinados com componentes de outros criadores para constituir um sistema(GHEZZI et.al., 1991). Ou "O estabelecimento e uso de sólidos princípios de engenharia para que se possa obter economicamente um software que seja confiável e que funcione eficientemente em máquinas reais."(NAUR; RANDELL, 1968).

Foram fixados alguns modelos de ciclo de vida de software, dentre eles: O modelo linear seqüencial e o modelo e espirais iterativas e incrementais.

5.1 O MODELO LINEAR SEQÜENCIAL

Este faz uma abordagem sistemática e seqüencial do projeto de software (veja figura) onde o projeto segue para a próxima fase apenas quando a fase em questão está totalmente concluída.

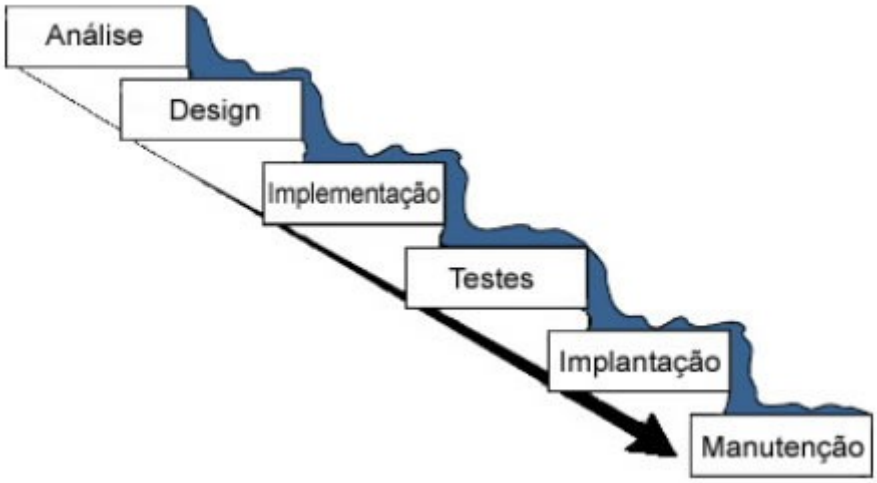


Figura 3: Etapas do Modelo Linear Sequencial

Este modelo tem uma deficiência por cada fase depender da total conclusão da fase anterior. Esse modelo é um tanto inflexível, e na pratica é difícil segui-lo a risca, pois aspectos impasses existentes em uma determinada fase podem requirir uma remodelagem da fase anterior. Por exemplo, se na fase de testes, algum erro for encontrado no sistema, haverá a necessidade de retornar a fase de codificação.

5.2 O MODELO DE ESPIRAIS ITERATIVAS E INCREMENTAIS

Esse modelo segue alguns dos passos do modelo de cascata, porém de uma maneira cíclica (iterativa), onde as fases de projeto, codificação, teste e reavaliação são repetidos até que o software tenha a qualidade esperada. Cada ciclo do modelo identifica uma versão do software, e a cada versão o software torna-se melhor. Este modelo tem o beneficio de ser um modelo rápido de construção, já que não é necessário ater-se a detalhes profundos de cada fase, pois há a possibilidade de reavaliar a mesma. Outro ponto é que em pouco tempo existirá uma versão pronta para análise por parte do usuário, que torna o desenvolvimento em conjunto com o mesmo mais preciso. (GHEZZI et.al., 1991)

5.2.1 A Etapa de Codificação e Teste

Dentro dos modelos apresentados anteriormente, a etapa de codificação e a etapa de testes podem ser muito próximas, pois dependendo do projeto é dispendioso ter um profissional para cada etapa. Mesmo em casos onde há equipes de codificação e testes distintas existe a necessidade de testes por parte dos profissionais de codificação, testes estes em menor escala, geralmente para verificar pequenos erros na parte sintática e léxica do código fonte.

5.2.2 Codificação e Testes em Dispositivos Móveis

Quando tratamos de dispositivos móveis, em muitos casos não é possível ou é muito dispendioso escrever o código fonte diretamente no mesmo, pois estes dispositivos dispõem de certa limitação de hardware (contem teclados e telas reduzidos, não possuem recursos de processamento suficientes para suportar uma ferramenta de programação). O software então é feito em um computador e depois inserido no dispositivo para passar a fase de testes. Se for seguido o modelo de cascata, o fato de codificar em um dispositivo e testar em outro traz um mínimo impacto na produtividade, porem no modelo espiral descrito anteriormente este impacto é bem maior. É despendido um certo tempo para interligar os dispositivos em questão (de codificação e de testes) e transferir o software, tempo este que é multiplicado pela quantidade de iterações do modelo de espirais iterativas e seqüenciais.

5.3 AMBIENTE DE DESENVOLVIMENTO INTEGRADO

Um ambiente de desenvolvimento integrado é conjunto de softwares criado para auxiliar na tarefa de desenvolver softwares, tornando-a menos dispendiosa. Fazem parte desses sistemas vários plugins, que trazem funcionalidades como: Edição de texto, compilador, depurador, ferramentas CASE para geração de código, entre outros. Existem Ambientes de Desenvolvimento Integrado voltados para o desenvolvimento de aplicações para dispositivos móveis, como o NetBeans que é utilizado para a linguagem Java e outras, e o Carbibe.c++ utilizado para a linguagem C++. Cada uma dessas ferramentas possui seu plugin para depuração de código. No caso do Python, os Ambientes de Desenvolvimento Integrado que o atendem, não possuem suporte para dispositivos móveis

5.3.1 Emuladores

Os emuladores oferecem ao programador um dispositivo móvel virtual dentro de seu computador para a realização de testes. Assim permitindo codificação e teste em um dispositivo apenas. Esta alternativa apenas minimiza a quantidade de testes diretamente no dispositivo móvel, pois o não é garantido que o emulador se comporte exatamente como o dispositivo móvel, e portanto, não é garantido que um software testado em um funcione corretamente no outro (LEE et.al., 2005).

6 TRABALHOS RELACIONADOS

Seguindo a idéia deste projeto, encontra-se uma ferramenta que permite a depuração de scripts Python em dispositivos com PyS60 em tempo de execução, chamado

Putools. Este software pode ser acessado através do endereço http://people.csail.mit.edu/kapu/symbian/python.html e oferece ao desenvolvedor o console interativo e execução de scripts remotos, sincronização de arquivos entre computador e dispositivo e a possibilidade de obter retratos do que é exibido na tela do celular (conhecidos *screenshots*). Possui interface gráfica com um pequeno editor de texto. Contudo, ao que pude observar sobre esta ferramenta, ela não foi projetada com suporte a acoplagens, portanto dificultando sua utilização em conjunto de ADI's. Outro detalhe importante é que também não apresenta uma divisão entre aplicativo e meio de comunicação evidente, dificultando a implementação de outros meios (wireless por exemplo). Utiliza comunicação bluetooth, porém, no computador, é necessário que o usuário efetue a conexão e com este, crie uma porta serial virtual no computador para que o putools possa efetuar a leitura e escrita de informações.

Ainda no assunto de desenvolvimento para dispositivos móveis, porém fora do conceito de *middleware*, há de se ressaltar o PED. Este é um pequeno editor de textos para o celular. Permite testes, edições, criação e exclusão de arquivos e pastas, além do console interativo. Pode ser encontrado em http://code.google.com/p/ped-s60/ .

7 TRABALHO DESENVOLVIDO

Este projeto consiste na implementação de um middleware que permita ao ADI conectar-se ao dispositivo móvel e assim oferecer testes no mesmo em tempo de implementação.

Este middleware será um modelo base, com uma interface genérica. A partir deste modelo, poderá ser acoplado a ADI em questão. Possuirá duas partes, uma no computador fazendo a conexão entre o ADI e a interface de comunicação (Bluetooth ou Wireless) e outra no dispositivo móvel, fazendo a conexão entre a interface de comunicação e o Interpretador da linguagem.

7.1 ESTRUTURA DO PROTÓTIPO

Este projeto foi segmentado em três partes principais. O módulo Tenoriumcel, instalado no dispositivo, trata de servir a interface do celular, respondendo aos comandos enviados pelo computador. O Tenoriumpc, é encarregado de prover ao desenvolvedor uma interface que permita enviar e receber scripts, executá-los (recebendo retornos e alertas de erros), executar trechos de código e manipular arquivos e diretórios no dispositivo móvel. Também oferece referências sobre o funcionamento do protótipo. O módulo Conexão, define as regras de comunicação entre o Tenoriumpc e o Tenoriumcel, oferecendo um canal estável de fluxo e dados.

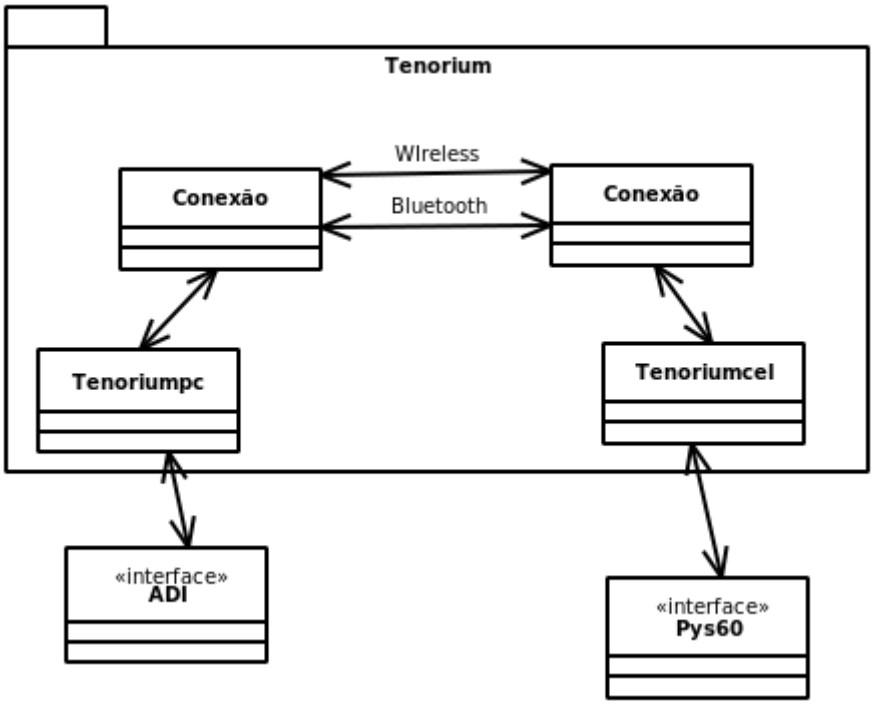


Figura 4: Modelagem do Tenorium.

7.1.1 Padrão de Comunicação

Como o módulo conexão apenas concede um canal de comunicação de texto, se fez necessário a criação de regras, para que possamos distinguir fluxos de comandos, retornos, erros e arquivos. Para a distinção dos mesmos, o fluxo é dividido em pacotes identificado através de cabeçalhos, definido como: quatro caracteres definindo o comando em questão, seguidos por n caracteres numéricos que definem o tamanho do texto enviado, seguidos por um caractere de espaço, que define o fim do cabeçalho. Após isso é enviado a porção de texto definida no cabeçalho.

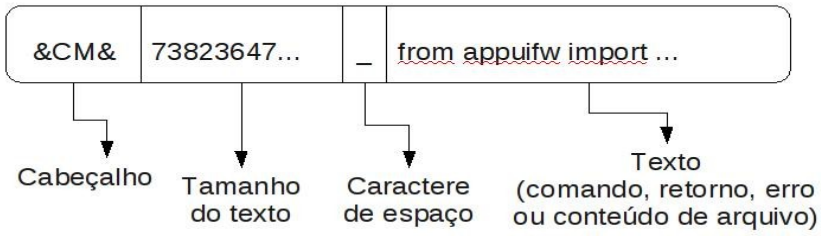


Figura 5: Segmento de fluxo de texto do Tenorium

O cabeçalho tem tamanho fixo, seguido pelo tamanho do texto a receber que tem tamanho variável, sendo delimitado pelo caractere de espaço. O texto vem em seguida, seu tamanho é definido anteriormente no próprio pacote.

O cabeçalho informa a ação, conforme descrito abaixo:

- Do computador para o celular:

&CM&	Envio de comandos para o interpretador interativo.
&DC&	Envio de arquivos do computador para o celular.
&CD&	Solicitação de envio de arquivos do computador.
&EX&	Iniciar a execução de determinado script Python.
&DL&	Solicita uma lista com as unidades de disco do dispositivo.
&LD&	Solicita uma lista com o conteúdo de um diretório.
&IF&	Questiona se um nome é um arquivo.
&ID&	Questiona se um nome é um diretório.
&MD&	Solicita a criação de um diretório no celular.
&RD&	Solicita a remoção de um diretório no celular.

Tabela 1: Cabeçalhos para comunicação Computador - Celular

- Do celular para o computador.

&CM&	Recebimento de retorno do interpretador interativo.
&CD&	Recebimento de um arquivo solicitado.
&RT&	Recebimento de um questionamento ou solicitação.

Tabela 2: Cabeçalhos para comunicação Celular - Computador

7.2 O MÓDULO TENORIUMCEL

O método `interact()` do modulo `code` é a principal parte deste módulo. Este cria um ambiente interativo do Python, lendo entradas de texto da entrada padrão e devolvendo os resultados na saída padrão. O Tenoriumcel apenas desvia a entrada e saída padrão para o socket estabelecido pelo módulo conexão. A partir de então, ao invés de escrever a entrada no teclado e ler o retorno na tela do celular, o usuário pode controlar de onde vem os comandos e para onde vão os resultados.

Para que se possa executar outras tarefas além de enviar comandos para o interpretador interativo, foi necessário controlar o fluxo de texto antes que ele atinja o interpretador. Sabe-se que a entrada e saída padrão do Python é tratada como um objeto do tipo arquivo. Para extrair dados desse objeto, o aplicativo faz uma chamada ao método `read()` do objeto arquivo. O Tenoriumcel tem uma classe interna que imita este objeto arquivo e oferece um método `read()` modificado, que através do cabeçalho da informação, executa a ação requerida, ou retorna o texto para ser interpretado, como mostrado na figura abaixo.

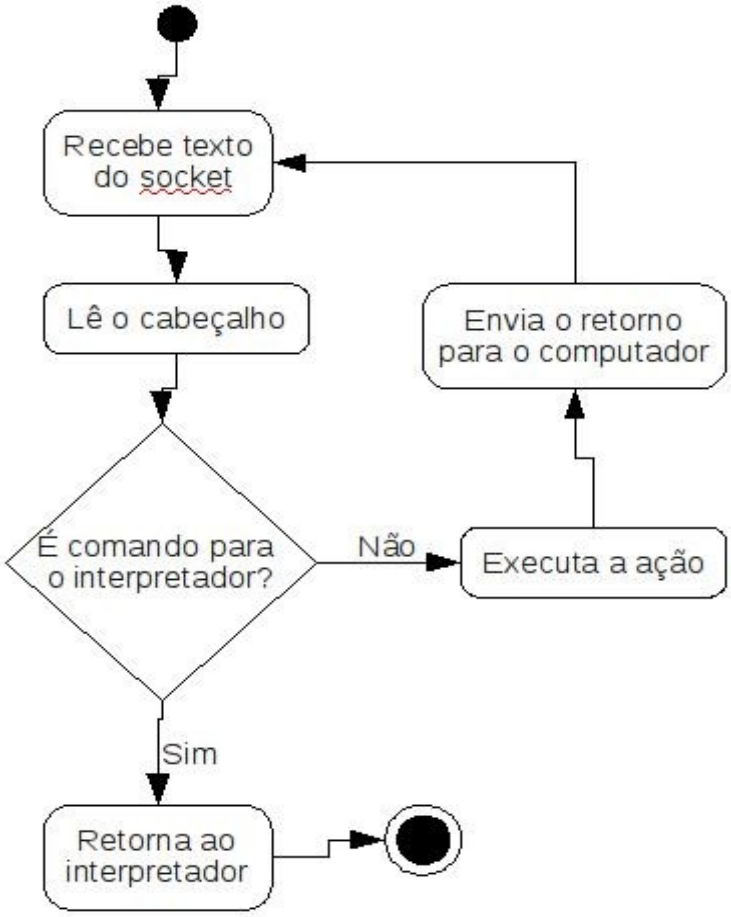


Figura 6: Diagrama de atividades do método read()

Caso o cabeçalho indique envio de comandos (&CM&) este método retorna o texto contido no segmento para o interpretador e finaliza. Caso contrário, ele executa a ação determinada no cabeçalho, passando argumentos contidos no segmento quando necessário, e envia a resposta ao computador, voltando a esperar pelo próximo segmento de texto.

7.3 O MÓDULO CONEXÃO

Tem a simples tarefa de estabelecer uma conexão entre o celular e o computador. É dividido em scripts, um para cada tipo de comunicação e o script padrão (o __init__.py),

que gerencia o módulo. O módulo é mesmo utilizado em ambos dispositivos, porém, quando utilizado pelo tenoriumpc, este módulo carrega todos os scripts de comunicação para o pc, enquanto que quando utilizado no celular, ele questiona ao usuário qual método utilizar. É possível definir novos métodos de comunicação. Para tal, apenas é necessário produzir um script que retorne um socket, já conectado e editar o __init__.py para inserir uma nova opção de escolha para o caso o módulo seja carregado pelo tenoriumcel e uma linha que o repasse caso seja carregado pelo tenoriumpc.

```
#CASO SEJA UTILIZADO NO CELULAR:

if sys.platform == 'symbian_s60':
    tipos = [u'Tcp/Ip',
             u'Bluetooth',
             #insira aqui a descrição do novo tipo
            ]
    import appuifw
    while True:
        id = appuifw.popup_menu(tipos, u'Escolha o tipo de conexao')
        if id == None:
            sys.exit()
        elif id == 0:
            from tcps60 import conectar
            break
        elif id == 1:
            from bluetooths60 import conectar
            break
        #INSIRA AQUI O CARREGAMENTO DO NOVO TIPO COMO ABAIXO:
        #elif id == 2:
            #from <OUTRO_MÓDULO> import <MÉTODO_PARA_CONECTAR>
            #break
#-------------------------------------------------------------

#CASO SEJA UTILIZADO NO COMPUTADOR:
else:
    from bluetoothpc import conectar as conectarbt
    from tcppc import conectar as conectartcp
    #INSIRA AQUI O CARREGAMENTO DO NOVO TIPO COMO ABAIXO:
    #FROM <NOVO_MODULO> IMPORT <MÉTODO_PARA_CONECTAR>
```

Figura 7: Código fonte do __init__.py.

7.4 O MÓDULO TENORIUMPC

Responsável por toda a interação com o usuário. Carrega os meios de conexão do módulo respectivo e possui a classe Tenorium.

Esta classe é executada em duas *threads*; uma principal, onde ocorre interação com o usuário e a segunda, que trata de recolher os dados enviados pelo celular e guardá-los em um *buffer* para posteriormente ser acessado pela *thread* principal.

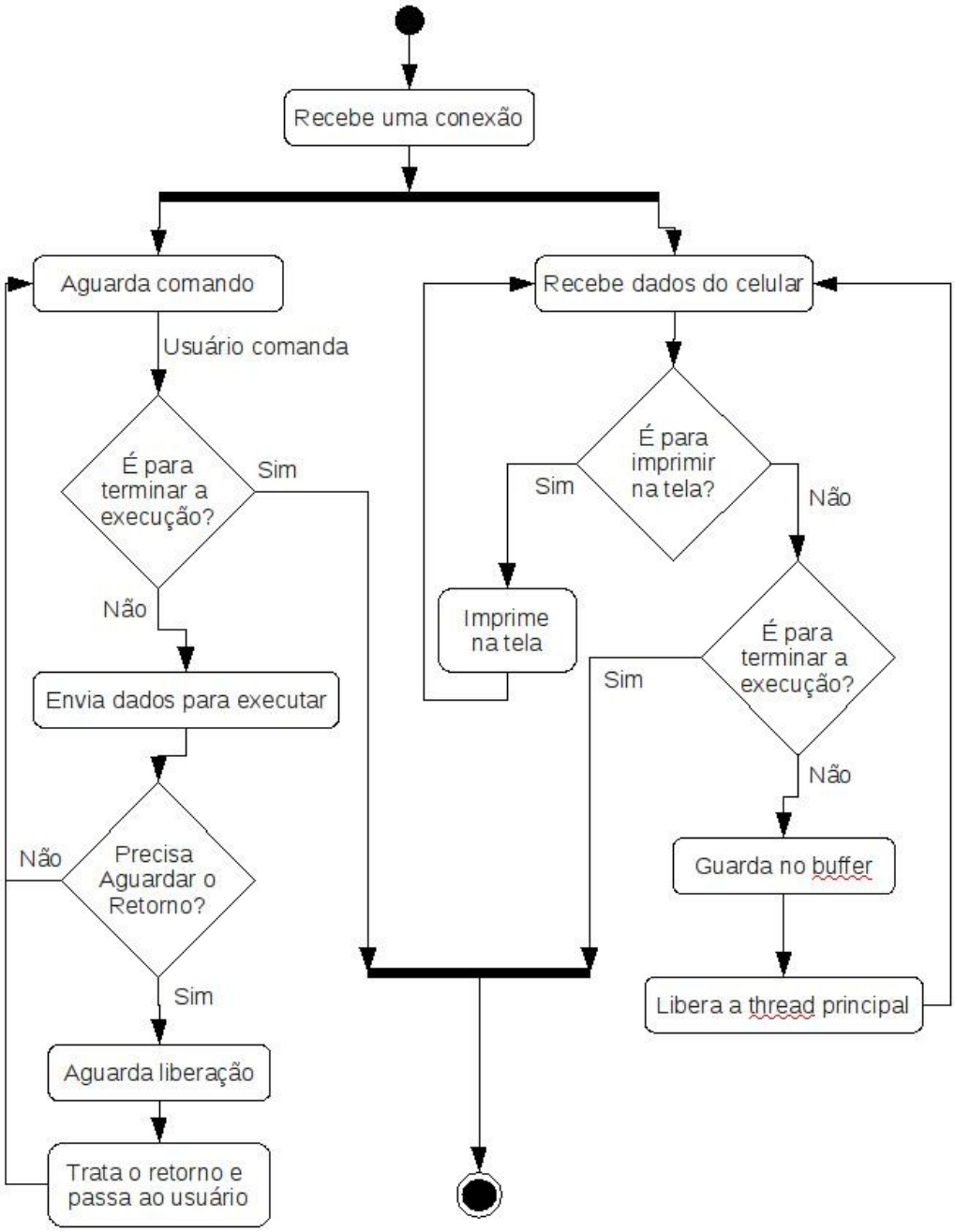


Figura 8: Diagrama de atividade da classe Tenorium.

Após receber uma conexão, a classe Tenorium inicia uma *thread* secundaria que fica aguardando por dados enviados pelo celular e continua interagindo com o usuário na *thread* principal. Nesta, o Tenorium fica aguardando as chamadas de seus métodos por parte do usuário. Quando executados, estes métodos enviam um segmento de dados para o celular, informando a ação e os argumentos necessários e dependendo do caso, faz uma pausa para aguardar o retorno que será recebido pela *thread* secundária. Esta, atua conforme o cabeçalho da informação recebida. Caso seja o retorno de algum método do Tenorium, guarda as informações em um *buffer* para serem acessadas pela *thread* principal e dispara um sinal para que a mesma continue sua execução quando necessário. Senão, imprime o resultado do processamento do interpretador interativo utilizando seu método de escrita definido quando o mesmo foi instanciado.

7.4.1 Funcionamento da classe Tenorium

Classe Tenorium: `Tenorium(socket[, metodo_de_escrita])` Cria uma instância da classe Tenorium, passando como argumentos uma instância de socket já conectado ao celular e opcionalmente um método de escrita, que será executado pela classe sempre que o celular enviar algum texto do interpretador interativo. O argumento opcional `metodo_de_escrita` deve ser um método no formato `metodo(texto)` onde `texto` é o retorno enviado pelo dispositivo móvel. Caso não seja informado, escreve o retorno na saída padrão. Sua instância possui os seguintes métodos:

- `comando(texto)` → envia um trecho de código para o interpretador interativo do celular. O resultado será impresso através do método de escrita definido na criação da instância. O argumento `texto` é do tipo string.

- `envia(arquivo[, local[, executar]])` → envia um arquivo para o celular e o executa caso solicitado. O argumento `arquivo` deve ser uma string com o caminho completo do arquivo. O `local` também deve ser do tipo string e conter o diretório destino para o arquivo (caso não seja atribuido, o Tenorium envia para "E:\Python\"). A variável `executar` deve ser do tipo booleano. O Tenorium executará o arquivo como script python caso esta contenha o valor verdadeiro.
- `recebe(arquivo[, local])` → recebe um arquivo do celular. O argumento `arquivo` deve ser uma string com o caminho completo do arquivo no celular. O `local` também deve ser do tipo string e conter o diretório destino para o arquivo.
- `executefile(arquivo)` → executa um arquivo como script python. O argumento `arquivo` deve ser uma string com o caminho completo do arquivo no celular.
- `drivelist()` → retorna uma lista com as unidades de disco do celular.
- `listdir(diretorio)` → retorna uma lista com o conteúdo do diretório especificado pela variável `diretorio`.
- `mkdir(diretorio)` → cria um diretório. O argumento `diretorio` contém o caminho completo do novo diretório e é do tipo string.
- `rmdir(diretorio)` → remove um diretório. O argumento `diretorio` contém o caminho completo do diretório a ser excluído e é do tipo string.
- `isdir(caminho)` → retorna booleano verdadeiro caso o caminho especificado no argumento `caminho` exista e seja um diretório. Caso contrário retorna booleano falso.
- `isfile(caminho)` → retorna booleano verdadeiro caso o caminho especificado no argumento `caminho` exista e seja um arquivo. Caso contrário retorna booleano falso.
- `close()` → encerra a conexão com o celular e termina a *thread* secundária.

7.5 REQUISITOS BÁSICOS

Para o funcionamento do Tenorium, é necessário um dispositivo móvel compatível com PyS60 (PyS60 e Python Script Shell, para Symbian Series 60 3ª Ed.), para a instalação do tenoriumcel. Também é preciso um computador, com o interpretador do Python previamente instalado. Os requisitos de hardware / software para a comunicação dependem do meio escolhido, caso seja via Bluetooth, ambos dispositivos precisam ter a interface instalada e o pacote PyBluez (encontrado em: http://org.csail.mit.edu/pybluez) precisa estar instalado no computador. Por outro lado, se for mais cômodo utilizar via TCP/IP, nenhum pacote adicional precisa ser instalado, apenas sendo necessária uma interface de rede wireless (caso o celular suporte) ou conexão a Internet para ambos dispositivos. Apesar de ter sido apenas testado em ambiente GNU/Linux (Ubuntu 9.04 mais precisamente) todo o projeto (com exceção da conexão bluetooth) foi desenvolvido com pacotes padrões do Python, que por sua vez são portáveis, tornando teoricamente possível sua utilização em outros ambientes. O pacote PyBluez é compatível com GNU/Linux e Windows XP, segundo seus desenvolvedores.

7.6 INSTALAÇÃO

- Celular: Copie o arquivo tenoriumcel.py para o diretório Python da memória interna ou do cartão de memória (c:\Python\ ou e:\Python), crie um diretório de nome “lib” dentro deste, e copie a pasta “conexao” para dentro dela.
- Computador: Copie a pasta conexão e o arquivo tenoriumpc.py para o diretório do pythonpath (geralmente /usr/local/lib/python em ambiente Linux).

7.7 UTILIZAÇÃO

Para utilizar o Tenorium, é preciso carregá-lo como um módulo, visto que ele foi projetado para ser um plugin. Neste caso, a título de exemplo, será utilizado o interpretador interativo do python para interagir com a biblioteca.

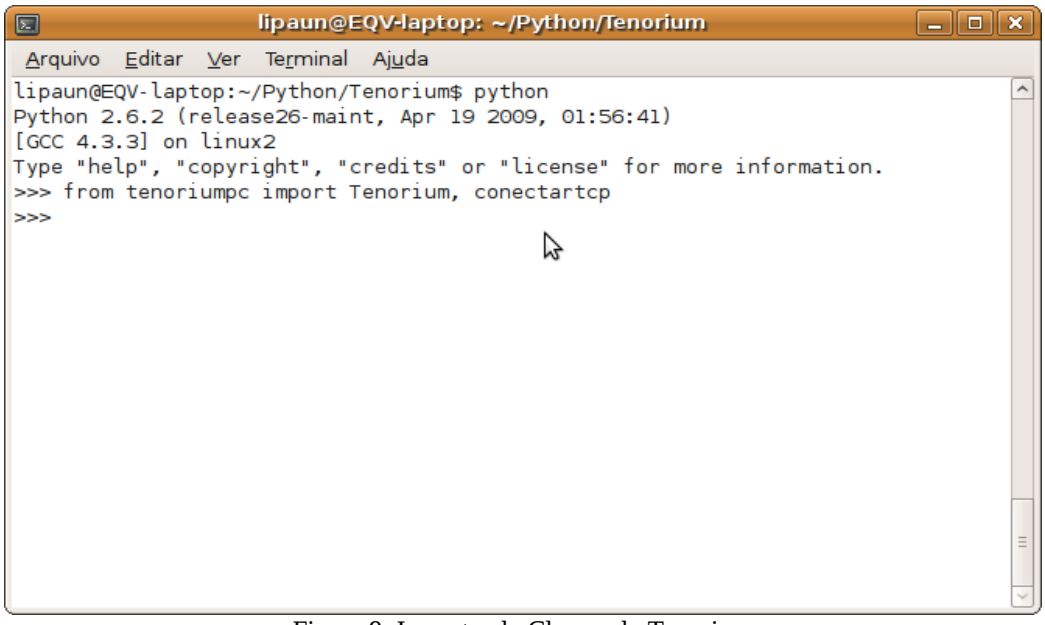
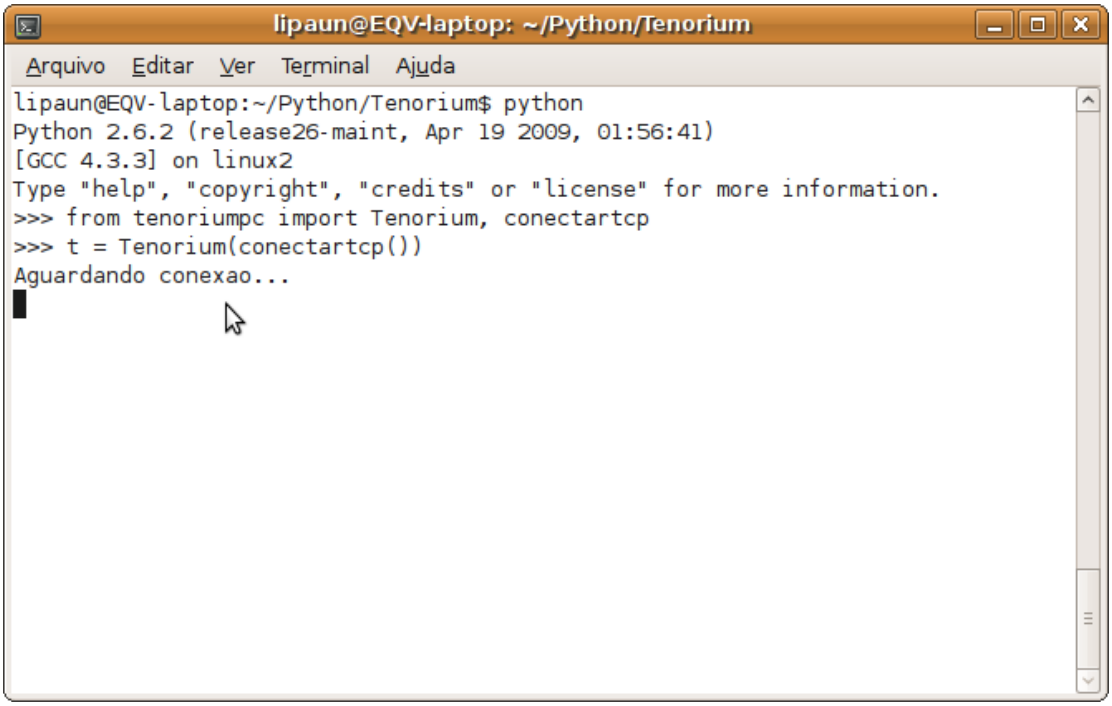


Figura 9: Importando Classes do Tenorium

Inicia-se a utilização carregando a classe `Tenorium` e um dos métodos de conexão do módulo `tenoriumpc` (neste exemplo, será utilizado conexão wireless), como visto na figura 9.

Logo após, cria-se uma instância da classe `Tenorium`, passando como argumentos o socket conectado com o celular (criado utilizando-se o método `conectartcp`). Segundo a regra definida no modulo conexao, o método `conectartcp` deve aguardar uma conexão na porta 12009.



É preciso agora, executar o script tenoriumcel.py no celular, escolher um dos métodos de conexão listado (neste caso TCP/IP). Como citado anteriormente, o computador aguarda uma conexão do celular, e para tal, quando utiliza-se conexão TCP/IP, deve-se inserir o *host* (número IP ou nome do domínio) do computador e a porta para conexão e logo após, escolher o ponto de acesso a ser utilizado, como mostrado na figura 10.

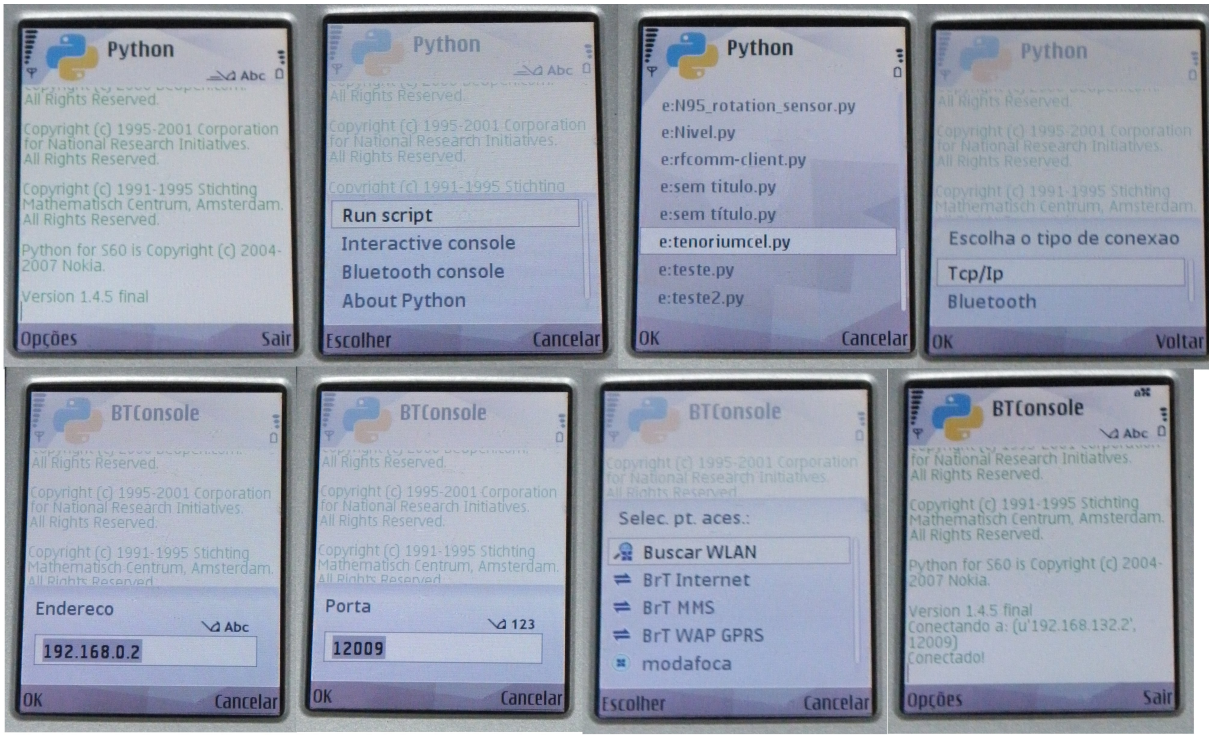


Figura 10: Conectando com o computador

Assim que a conexão é estabelecida, o celular envia o cabeçalho de seu interpretador interativo, que é impresso na tela do computador.

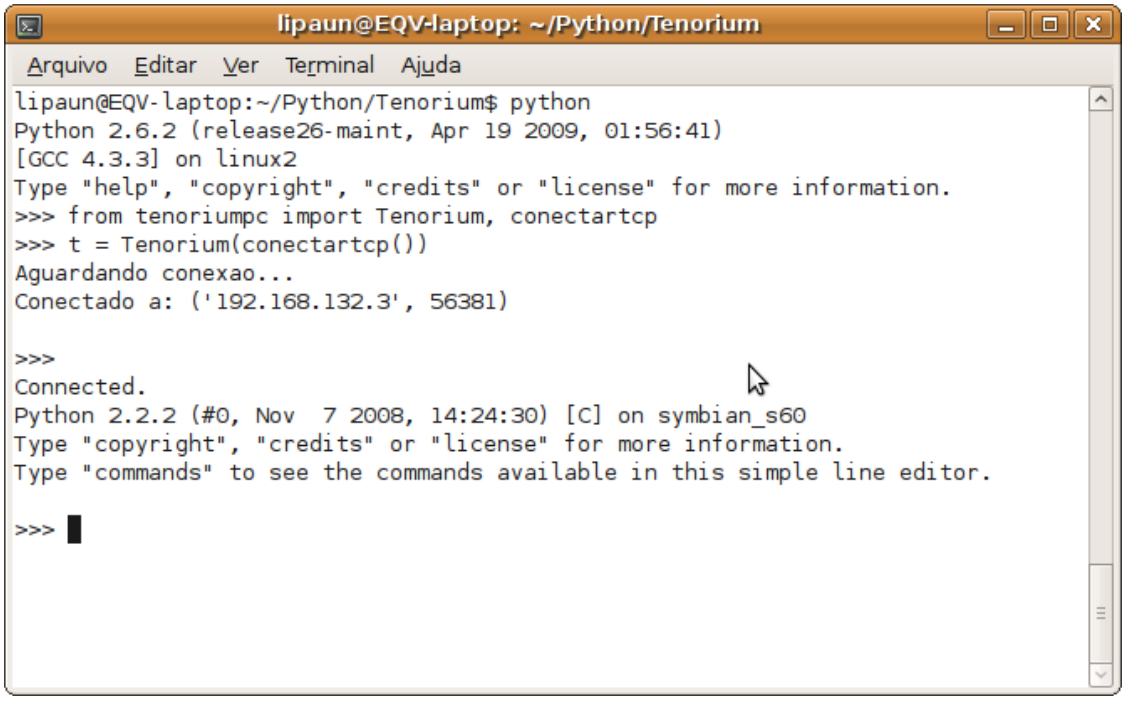


Figura 11: Dispositivo conectado.

A partir deste ponto, pode-se utilizar os métodos da instância `t` para executar trechos de código e scripts python, enviar e receber arquivos, criar e remover diretórios, entre outros. Neste exemplo, mostrado na figura 12, será carregado o modulo `appuifw`, que trata do acesso à interface gráfica do celular e após isso, usa-se o método `note` para exibir uma notificação na tela do celular.

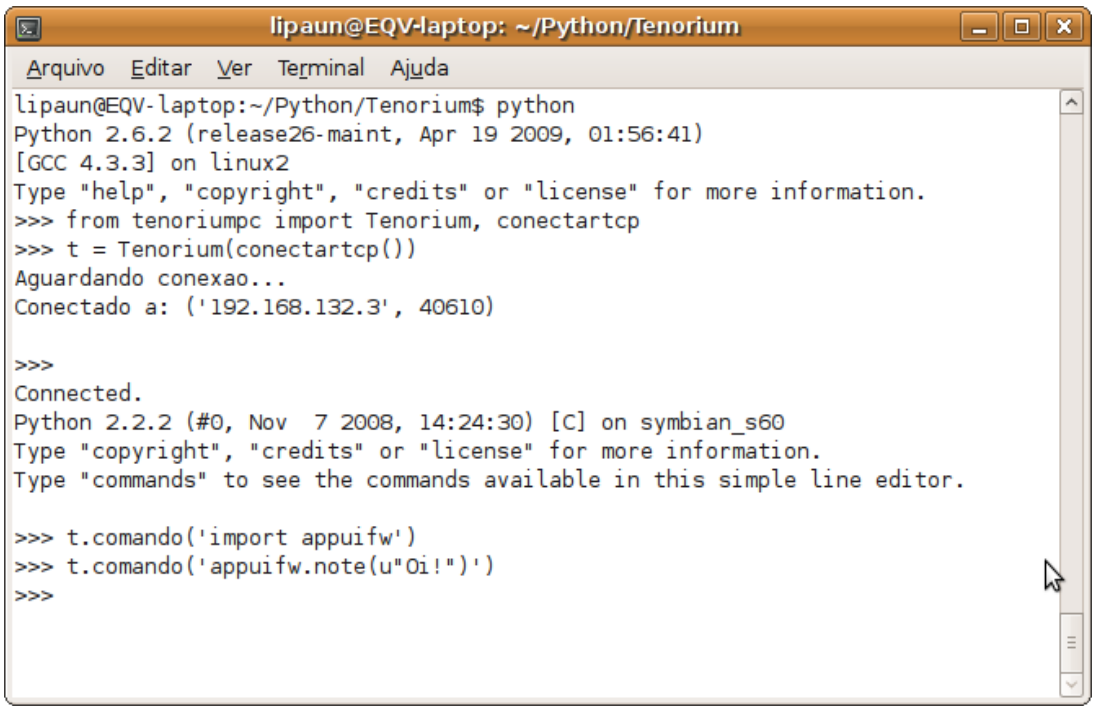


Figura 12: Enviando comando para celular.

E o resultado é:

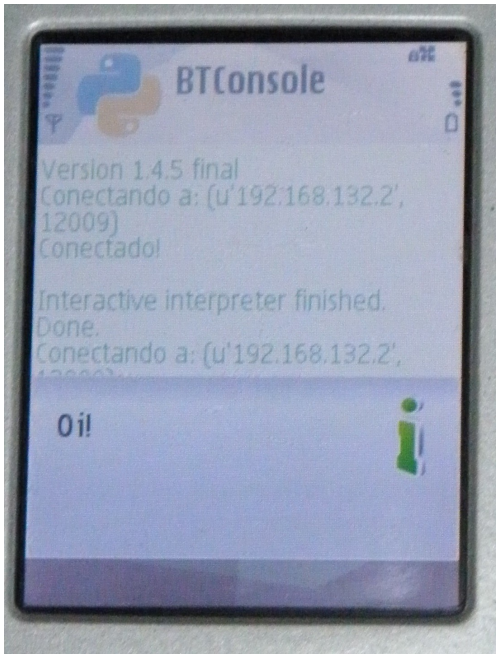


Figura 13: Nota lançada no celular.

Para executar um script no celular, deve-se primeiro salvá-lo em um arquivo de texto com a extensão “.py” e utilizar o método `enviar`, passando como argumentos o caminho do arquivo e o valor booleano verdadeiro para a variável `executar`.

Toma-se o exemplo da figura 14, este script carrega o módulo de interface gráfica do celular, usa o método `query` deste para lançar ao usuário um campo onde ele possa digitar seu nome e usa esta informação para saudar o usuário.

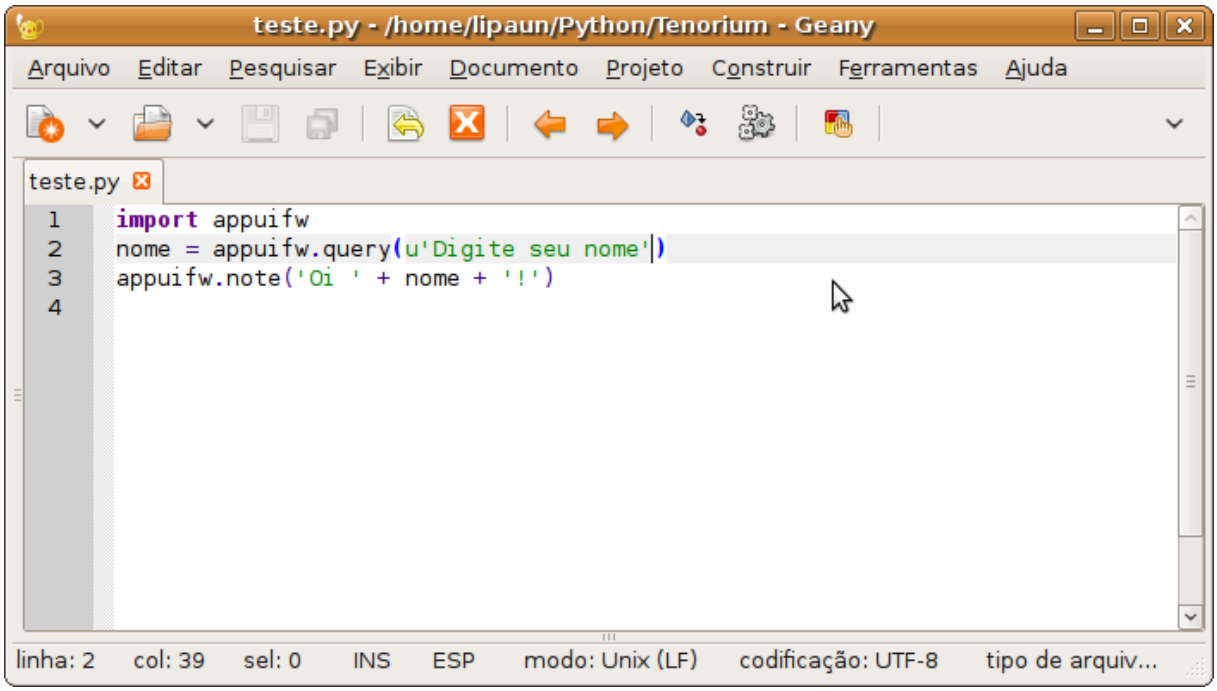


Figura 14: Exemplo de script Python

Chamando o método `envia` da instância `t` tem-se:

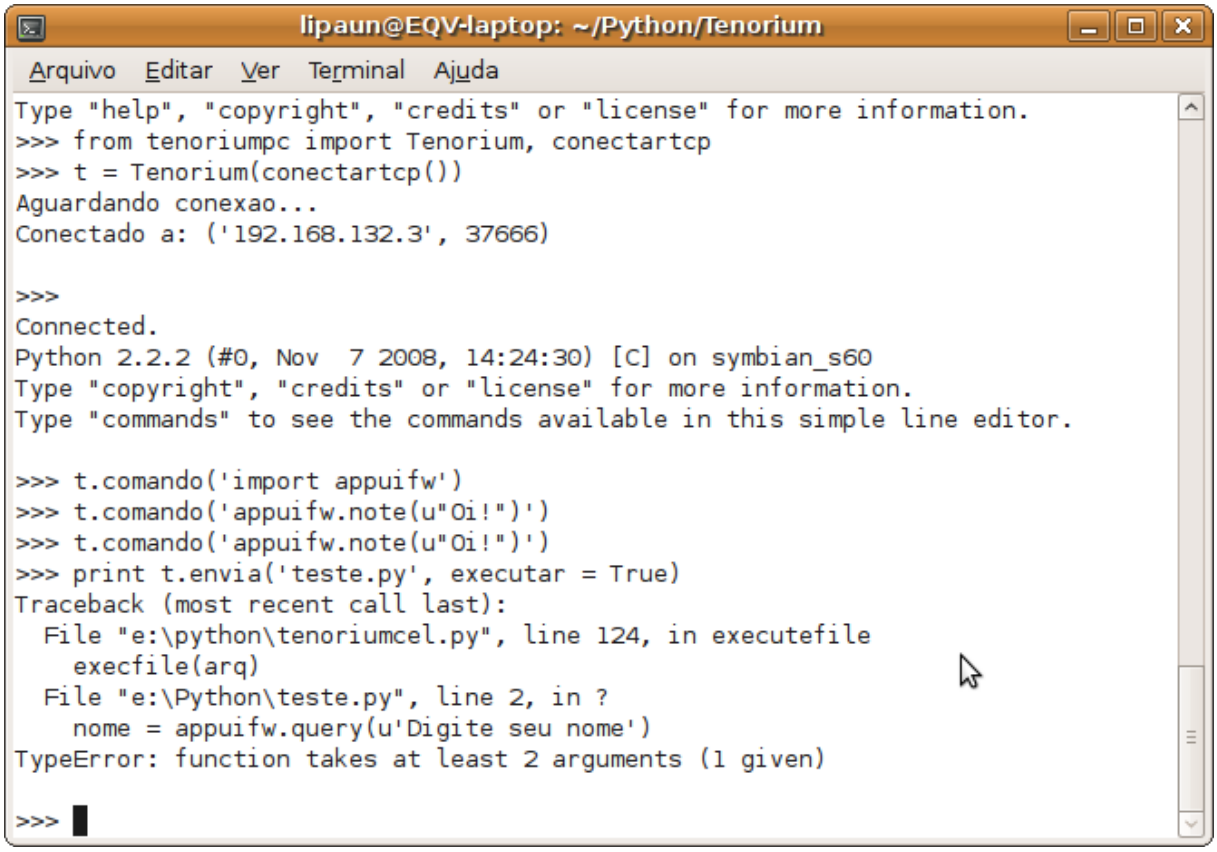


Figura 15: Enviando e executando script com defeito no celular

Note que o interpretador relatou um erro. Segundo o relatório, está faltando um argumento para a chamada de `query` dentro do script. Este método possui a seguinte assinatura: `query(texto, tipo_de_retorno)` onde `tipo_de_retorno` pode ser `“text”` para retornar strings, `“number”` para inteiros, entre outros. Portanto a correção da linha fica: `appuifw.query(u'Digite seu nome', 'text').`

Efetuando a devida correção e chamando `envia` novamente tem-se:

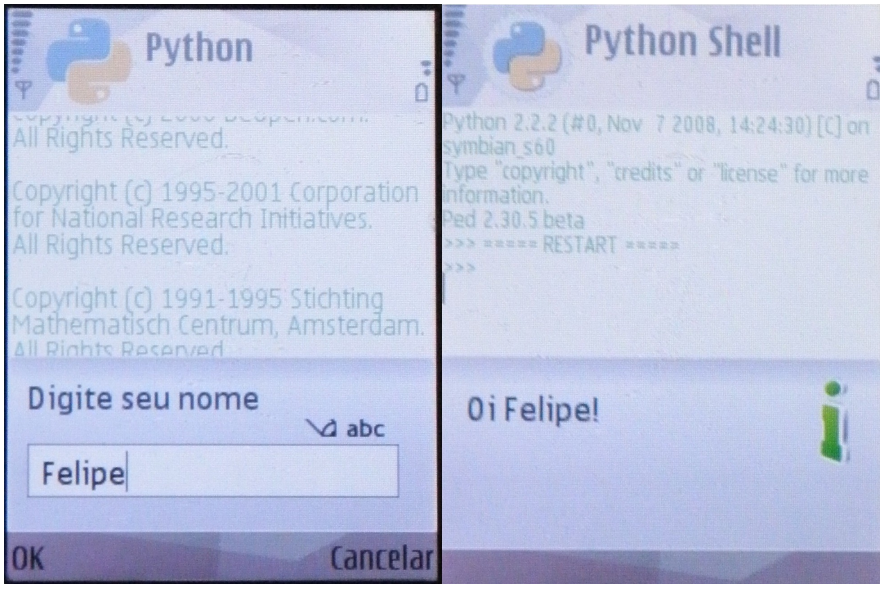


Figura 16: Script executando no celular.

O desenvolvedor pode também consultar o Pydoc do Tenorium, apenas fazendo uma chamada de `help(classe_ou_instancia_Tenorium)` como apresentado nas figuras 17 e 18.

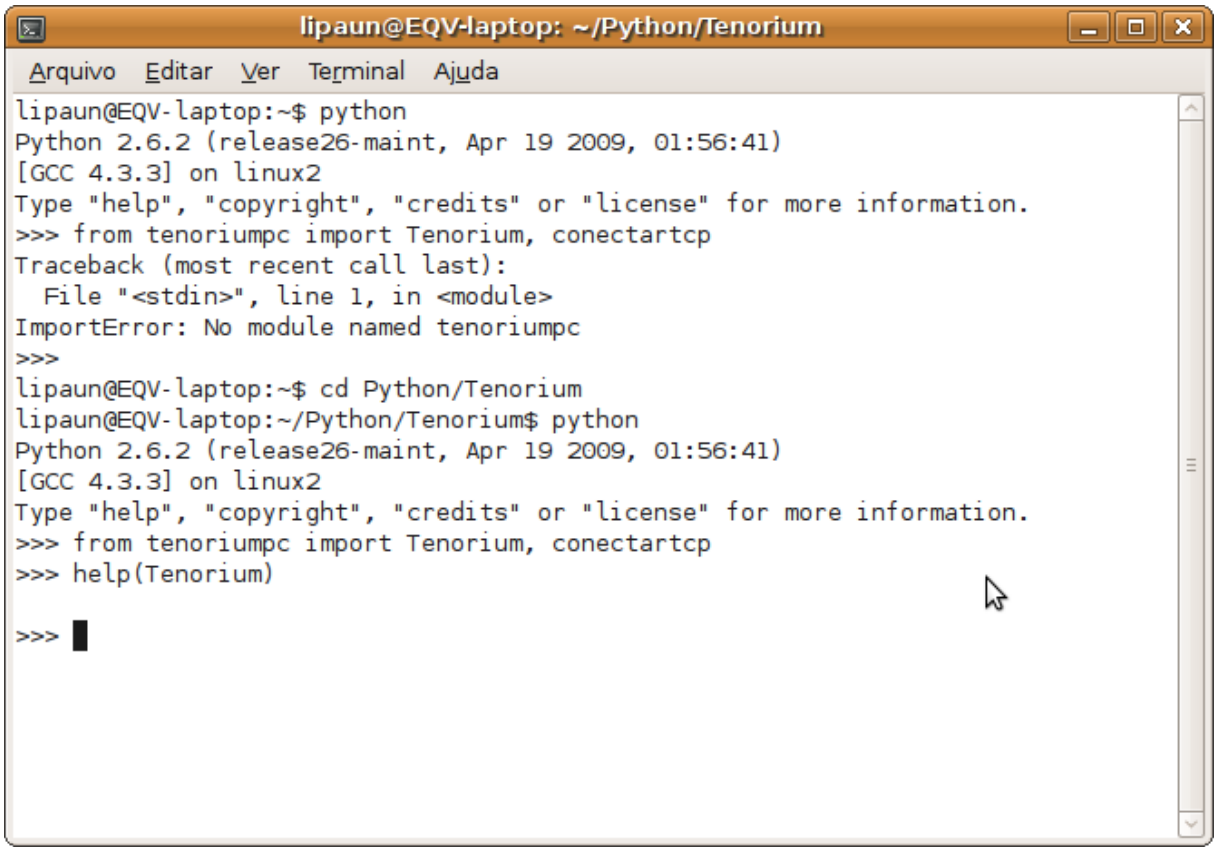


Figura 17: Chamada de acesso ao Pydoc do Tenorium.

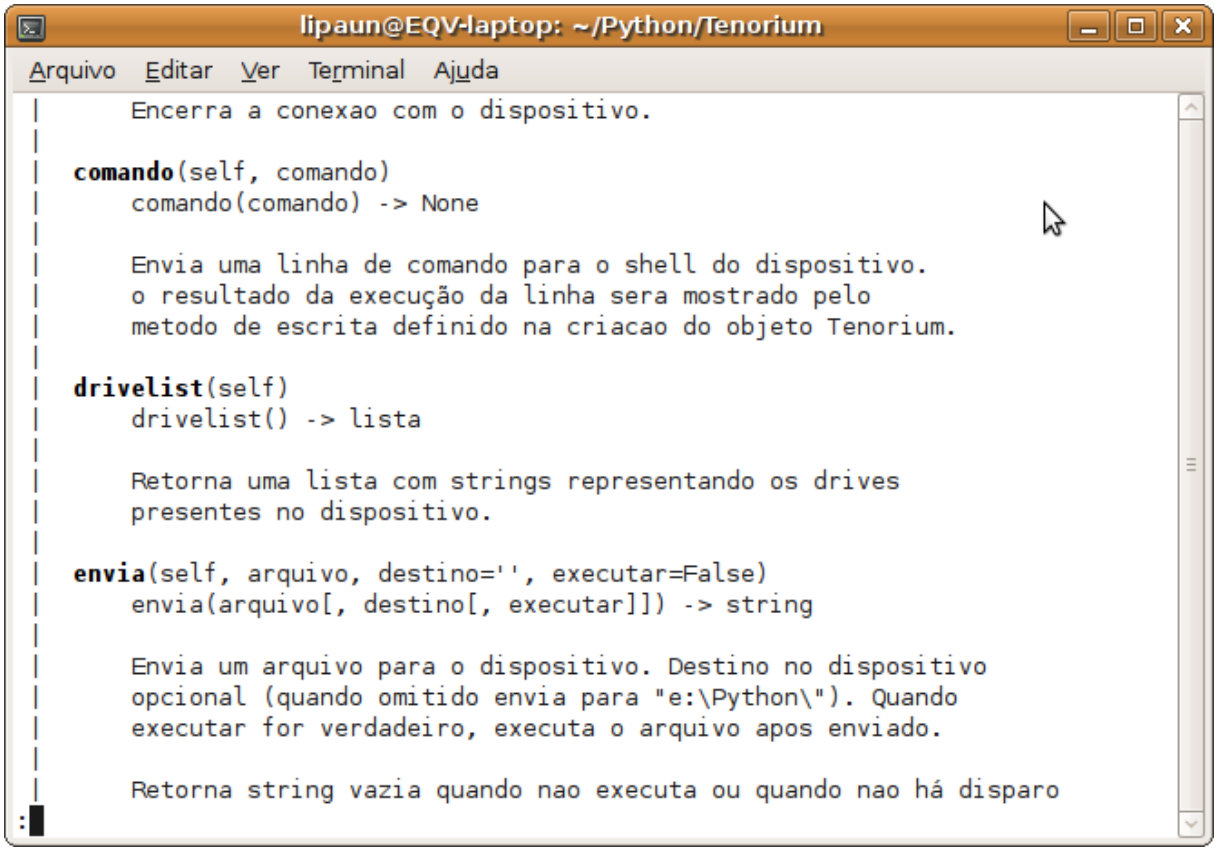


Figura 18: Pydoc do Tenorium.

7.8 RESULTADOS OBTIDOS

Com a utilização do Python e do PyS60, insere-se uma nova linguagem de programação no âmbito do desenvolvimento de um projeto dando suporte para futuros trabalhos junto aos acadêmicos do curso de Ciência da Computação da UNESC.

Também inserido neste meio, fica o desenvolvimento de um middleware para computação distribuída realizando a comunicação entre dispositivos móveis e computadores pessoais.

Entra em cena com o middleware, a comunicação utilizando as tecnologias Bluetooth e redes TCP/IP (tais como: wireless, 3G ou GPRS; dependendo das capacidades do dispositivo móvel) e além disso está preparado para outras tecnologias, como USB e Infra-Vermelho.

O resultado máximo deste projeto é um software que permita ao usuário efetuar depurações e testes em tempo de desenvolvimento. Otimizando assim a fase de desenvolvimento.

8 CONCLUSÃO

Atualmente dentro da área da computação temos a necessidade de produzir inúmeras aplicações para dispositivos móveis, sendo que isto pode ser feito por meio de um emulador ou desenvolvido em um ambiente de programação e transportado para o dispositivo assim verificando a situação do projeto. Nesta fase é acarretado um certo tempo de desenvolvimento do projeto e testes. Existe uma necessidade para os desenvolvedores minimizar este tempo, sendo assim este projeto vem contribuir na diminuição do tempo gasto

durante este processo, tornando isto mais simples e amigável. Desta forma o desenvolvedor passa a ter um beneficio evidente.

Por outro lado temos a área de sistemas distribuídos que vem aumentando consideravelmente no que tange o desenvolvimento de aplicações para melhorar o desempenho dos sistemas existentes. Este projeto desenvolveu um middleware de comunicação baseado nas características de sistemas distribuídos e comunicação entre dispositivos, por meio de várias tecnologias de acesso.

A partir do desenvolvimento deste aplicativo (middleware) espera-se que sejam agregadas novas funcionalidades a este modelo e que permita que mais aplicações seja idealizadas na área de sistemas distribuídos.

Os resultados esperados neste projeto atingiram os objetivos propostos, na forma de implementação e agregação de conhecimento na área de desta aplicação. Estes resultados foram obtidos mediante a análise e execução do software desenvolvido, e teve seu comportamento da forma esperada.

8.1 DIFICULDADES

Uma dificuldade encontrada foi com o módulo de threads do python, pesquisando sobre o mesmo, foram encontrados vários relatos sobre usuários com a mesma dificuldade. A biblioteca oferece uma classe que implementa a abertura de uma nova thread, porém não implementa um método forçar o seu interrompimento (mata-la), para tal, é necessário chamar um método do sistema operacional para interromper seu processo.

8.2 TRABALHOS PROPOSTOS

A implementação de mais meios de comunicação para o módulo conexão, bem como a criação de Plugins para que o Tenorium possa ser utilizado juntamente a ADI's são as propostas originais. Seguindo o ideal da ferramenta, sugere-se também a criação de middlewares similares para outros tipos de dispositivos que suportem Python, bem como para outras linguagens de programação.

Como escrito anteriormente, threads em Python representam uma certa dor de cabeça. Portanto é sugerido também a substituição da mesma por um módulo que permita a comunicação assíncrona entre os dispositivos.

REFERÊNCIAS

EMMERICH, Wolfgang. **Software Engineering and Middleware: A Roadmap**. London: University College London, 2005.

FOROUZAN, Behrouz A.. **Data Communications and Networking**. Columbus: McGraw-Hill Science/Engineering/Math, 2003.

GHEZZI, Carlo; JAZAYERI, Mehdi; MANDRIOLI, Dino. **Fundamentals of software engineering**. New Jersey: Prentice-Hall, 1991.

LEE, Valentino; SCHINEIDER, Heather; SCHELL, Robbie. **Aplicações móveis**: Arquitetura, projeto e desenvolvimento. Pearson/Makron Books, 2005.

LOUREIRO, A. A .F. et al.. **Comunicação sem fio e Computação Móvel: Tecnologias, Desafios e Oportunidades**. In.: CONGRESSO DA SOCIEDADE BRASILEIRA DA COMPUTAÇÃO, 2003, Campinas. **Anais**...Campinas, 2003

LUTZ, Mark; ASCHER, David. **Aprendendo Python**. Porto Alegre: Bookman, 2007.

MACIEL, Rita S. P.; ASSIS, Semírames R.. **Middleware: Uma solução para o desenvolvimento de aplicações distribuídas**. Salvador: CienteFico. Ano IV, 2004.

MATEUS,Geraldo R.; LOUREIRO, Antônio A. F.. **Introdução à Computação Móvel**. Rio de Janeiro: Imprinta Gráfica e Editora Ltda, 1998.

NAUR, Peter.; RANDELL, Brian. **Software Engineering: Report on a Conference**. In.: NATO CONFERENCE ON SOFTWARE ENGINEERING, 1968, Garmish. ...Garmish, 1969

SCHEIBLE, Jürgen; TUULOS, Ville. **Mobile Python: Rapid Prototyping of Applications on the Mobile Platform**. New Jersey: Wiley, 2007.

SYMBIAN. **Symbian OS**. Disponível em: <http://www.symbian.com/>. Acesso em: 30 Outubro 2008.

TANEMBAUN, Andrew S.. **Computer Networks**. New Jersey: Prentice Hall, 2003.