3
Fundamentos Teóricos

Neste capítulo são apresentados os fundamentos teóricos contemplados no desenvolvimento deste trabalho. Para propiciar um melhor entendimento, inicia-se com uma apresentação do contexto de aplicação do trabalho, aprofundando-se na descrição de serviços web e o ambiente de computação móvel, sendo abordado em particular o sistema operacional Android (GOOGLE, 2010a). Em seguida, abordam-se os conceitos de computação autonômica e sistema multiagentes, discorrendo-se sobre as razões para sua aplicação neste trabalho. Por fim, relaciona-se este trabalho com abordagens que foram encontrados na literatura.

O atual cenário dos serviços de telecomunicação nos oferece a flexibilidade de acesso à internet praticamente em qualquer momento e lugar. Esta tecnologia estimula o desenvolvimento e a disponibilização de serviços aos usuários via web. Serviços esses que podem ser acessados através de dispositivos móveis, uma vez que a tecnologia de computação móvel evoluiu de tal modo que atualmente a maioria dos dispositivos é capaz de conectar a internet e, portanto, utilizar esses serviços.

No entanto, a comunicação entre sistemas computacionais, como a de aplicações de dispositivos móveis e serviços web, está suscetível a falhas. A primeira dificuldade a ser transposta é a indisponibilidade de um serviço web, que pode ser superada através da escolha de serviços alternativos com a mesma finalidade. Contudo, pode-se incorrer em outra dificuldade, a interoperabilidade dos dados, caso o novo serviço escolhido utilize um protocolo de dados diferente do anterior, o que ocorre frequentemente. Além disso, o usuário espera que estas questões sejam tratadas preferencialmente pelas aplicações de forma automática durante sua execução, configurando-se assim a necessidade de um comportamento autonômico do sistema executado pelo dispositivo móvel. Para auxiliar no desenvolvimento de aplicações de modo a atender essa expectativa dos usuários, é proposto um *framework* cujos fundamentos considerados na sua construção são apresentados a seguir.

3.1. Serviços Web

Benslimane *et al* (BENSLIMANE, DUSTDAR e SHETH, 2008) relata que a internet e tecnologias relacionadas criaram um mundo interconectado na qual se pode trocar informações facilmente, processar tarefas colaborativamente e formar redes sociais, com o objetivo de alcançar a eficiência e melhorar o desempenho. Os autores dizem ainda que os serviços web estão emergindo como a principal tecnologia para desenvolvimento de sistemas automatizados no que tange questões de interação entre sistemas distribuídos e heterogêneos.

A W3C (W3C, 2004) define serviço como um recurso abstrato que representa uma capacidade de realizar tarefas que formam uma funcionalidade coerente do ponto de vista das entidades fornecedoras e solicitantes. Para ser usado, um serviço deve ser realizado por um agente provedor. Adicionalmente, a W3C define serviço web como um sistema de *software* projetado para suportar interação máquina-máquina sobre uma rede, no caso a web.

Assim, para que um sistema se comunique com os serviços web é necessário que ele esteja conectado à internet. Além dessa conexão, deve haver um meio para que o sistema encontre o serviço web desejado. Após superar esses dois requisitos inicias, para que a troca de informações entre o sistema e o serviço web se efetive ainda é necessária a utilização de protocolos adequados para que a troca de dados faça sentido entre os sistemas. Neste contexto, nota-se a existência de três atores: provedor de serviço, registrador de serviços e consumidor de serviços. Provedor de serviço é aquele que efetivamente executa a atividade e publica seus serviços ofertados no registrador de serviços. O registrador de serviços funciona como um repositório em que estão descritos os serviços ofertados e procurados. Já o consumidor de serviços é aquele que procura no registro de serviços o serviço que melhor lhe atende e também publica no registro de serviços a necessidade de um serviço. A Figura 3.1 apresenta uma ilustração simplificada da arquitetura elaborada pela W3C (W3C, 2004), em que se observam os três atores envolvidos na utilização de serviços web e a indicação de protocolos utilizados na comunicação, que são descritos a seguir.

Figura 3.1 – Ilustração simplificada da arquitetura dos serviços web.

Na definição W3C (W3C, 2004), os serviços web se comunicam através de uma interface WSDL, de maneira prescrita utilizando do protocolo de comunicação SOAP, normalmente transmitida através de HTTP com uma serialização XML em conjunto com outros padrões relacionados à Web.

O *Universal Description, Discovery and Integration* (UDDI) é responsável pela descrição e descoberta de Web Services. O UDDI oferece informações básicas sobre organizações (*White Pages*), categoriza serviços segundo diversas categorias (*Yellow Pages*) e providencia informações técnicas sobre o serviço (*Green Pages*).

O *Web Service Description Language* (WSDL) é a linguagem utilizada para descrever os serviços web. Os WSDL definem os tipos de dados usados pelos serviços web, descritos através da estrutura XML.

O *Simple Object Access Protocol* (SOAP) é um protocolo para definição e estruturação das mensagens que serão trocadas entre um sistema qualquer e o serviço web. A mensagem SOAP é organizada em um envelope contendo um cabeçalho e um corpo, descritos através da estrutura XML.

Benslimane *et al* (BENSLIMANE, DUSTDAR e SHETH, 2008) destacam a utilização de vários padrões que suportam o desenvolvimento dos serviços web como WSDL, UDDI e SOAP, quando se aborda o conceito de computação autonômica aplicada a sistemas web. No entanto, o uso de padrões mais leves para implantação de serviços, especialmente serviços web, vem aumentando. Desta forma, surgem novas alternativas para anotação e serialização dos dados como

JSON e YAML que apresentam sintaxe mais limpa, fácil de ler e editar, e boa estruturação de dados (FONSECA e SIMOES, 2007).

Embora se observe o esforço em estabelecer padrões mais simples para construção dos serviços web, vide Fonseca e Simões (FONSECA e SIMOES, 2007), os desafios da indisponibilidade e interoperabilidade na utilização dos serviços web permanecem. Tais desafios proporcionam oportunidades para o desenvolvimento de serviços similares e, por conseguinte, propicia a construção de sistemas capazes de se adaptar a este intercâmbio de serviços. Essa necessidade de construção de sistemas autônomos impulsionou ainda mais a pesquisa e desenvolvimentos dos conceitos de engenharia de software com ênfase em computação autonômica. Observando que dispositivos móveis é o foco deste trabalho, aplicaremos os conceitos de autonomia no desenvolvimento de aplicações no ambiente de computação móvel.

3.2. Computação Móvel

Heilmeier (HEILMEIER, 1992) descreve o cenário da computação móvel em que pessoas e máquinas devam ser capazes de acessar informações e se comunicarem com facilidade e segurança, em qualquer meio ou combinação de meios de comunicação - voz, dados, imagem, vídeo ou multimídia – a qualquer momento, em qualquer lugar, no momento oportuno, preservando a relação custo-benefício. Vale ressaltar que esses meios de comunicação estão disponíveis na internet através de serviços web, por exemplo, como os fornecidos pelos populares sites de redes sociais (Facebook[1], Twitter[2], Orkut[3]), e são acessíveis por dispositivos móveis.

O crescente mercado de *smartphones* contribuiu para uma acirrada disputa no campo de sistema operacional para dispositivos móveis. No momento da escrita desta dissertação, o mercado apresenta dois grande concorrente, o sistema operacional da Google chamado Android (GOOGLE, 2010a) e o sistema da Apple chamado iOS (APPLE, 2012). Além de um crescimento vertiginoso na utilização

[1] Fonte: https://www.facebook.com/

[2] Fonte: http://twitter.com/

[3] Fonte: http://www.orkut.com.br/

do sistema Android, vale ressaltar que sua licença de distribuição é gratuita, diferente de seu concorrente iOS. Além disso, diversos fabricantes de *smartphones* (Samsung, Motorola, Sony Ericsson, HTC, entre outros) desenvolvem aparelhos para esta plataforma, enquanto o iOS é de uso exclusivo da Apple. Segundo pesquisa da empresa Nielsen (NIELSEN, 2011) o sistema Android já domina o mercado de sistemas operacionais para dispositivos móveis e desta forma ao desenvolver aplicações para este sistema obtém-se acesso a uma maior quantidade de usuários.

Por ser um sistema operacional de distribuição livre, líder do mercado de sistemas operacionais para dispositivos móveis, e que possui uma boa API de desenvolvimento baseada na linguagem de programação Java, consideramos que o sistema Android reuniu qualidades suficientes para que o trabalho fosse desenvolvido para sua plataforma.

3.2.1. Android

O sistema operacional Android (GOOGLE, 2010a) foi lançado pela Google em novembro de 2007 e foi liberada uma API para desenvolvedores junto com seu código fonte em outubro de 2008. Seu código fonte é aberto para os desenvolvedores[4]. Desde seu lançamento, o Android obteve um vertiginoso crescimento em seu uso, chegando a um patamar de aproximadamente 43% de participação no mercado mundial sistemas operacionais para *smartphones* em Agosto de 2011 (KELLOGG, 2011).

A Figura 3.2 apresenta a estrutura elaborada pela Google para o sistema operacional Android. Observa-se que o Android é estruturado em camadas em que a camada mais superior é formada pelas aplicações, a intermediária é formada por *middleware* composto por "Application Framework", "Libraries" e "Android Runtime", e um núcleo Linux compõe a camada mais inferior, que são descritos a seguir.

[4] Exceto a versão em desenvolvimento a partir Março de 2011 - versão Honeycomb (3.0).

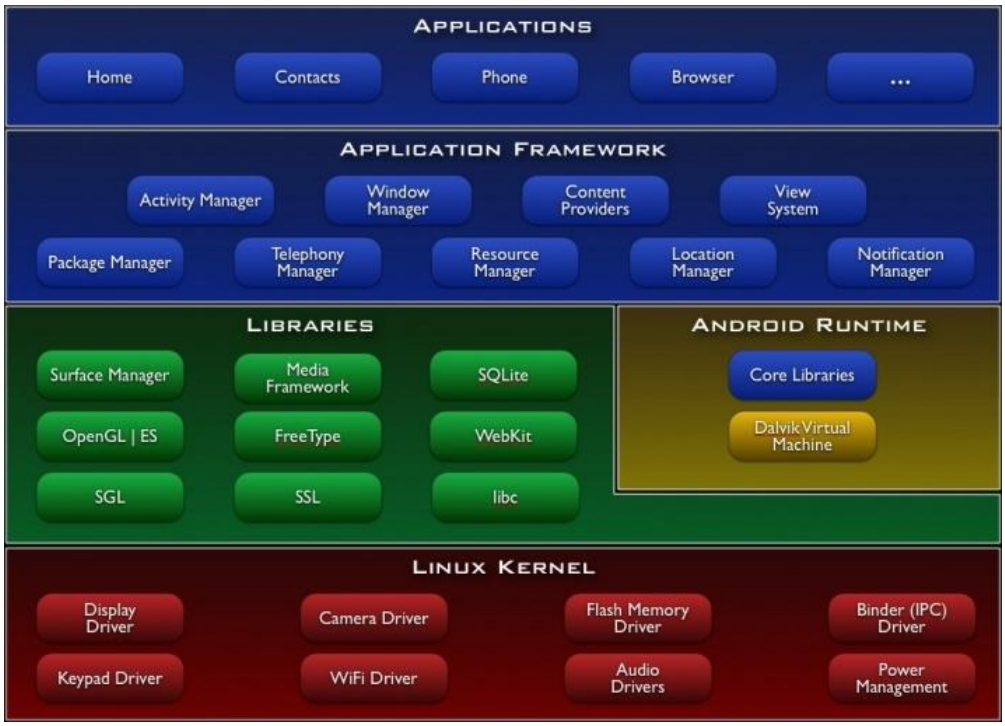

Figura 3.2 – Arquitetura do sistema operacional Android (GOOGLE, 2010b).

A camada mais superior, chamada de *Applications*, provê uma série de aplicações pré-definidas como calendário, mapas, agenda, dentre outras. Esta camada abrigará todas as aplicações desenvolvidas para o sistema Android. Já o *Application Framework* é um *framework* de desenvolvimento de aplicações para o Android que permite acesso a todas as funcionalidades do *hardware* bem como permite a conexão de uma aplicação a recursos de outras aplicações. Esta arquitetura foi desenhada de forma a facilitar o reuso de componentes. O Android fornece também diversas bibliotecas em C/C++ que podem ser acessadas através do *framework* de desenvolvimento, bibliotecas essas disponibilizadas pela camada *Libraries*.

O *Android Runtime* é o responsável por controlar a execução da aplicação visto que, o Android possui uma série de bibliotecas de sistema que possuem funcionalidades similares às bibliotecas core do Java. Cada aplicação possui seu próprio processo e é executada em uma máquina virtual Dalvik. A máquina virtual Dalvik foi otimizada para o sistema Android permitindo a execução de múltiplas instâncias da máquina virtual consumindo pouca memória e energia de processamento. Por fim, temos o núcleo do sistema, representado pela camada *Linux Kernel* que é baseado no sistema Linux, utilizando de forma eficiente seus serviços de segurança, gerenciamento de memória, processos, redes e *drivers* o

tornado assim viável de ser utilizado em ambientes de recursos limitados como dos dispositivos móveis.

A Google provê uma API simples e de fácil manipulação, baseada na linguagem de programação Java, como meio de incentivar o desenvolvimento de aplicações para seu sistema operacional. Esta API pode ser obtida também para o ambiente Eclipse, cuja Google é parceira, através do plug-in *Android Development Tools* (ADT) que inclui um emulador de *smartphone* para testar seus aplicativos. A API utilizada no desenvolvimento deste trabalho é sua décima terceira versão, lançada em Julho de 2011 com nome de Android 3.2.

Os aplicativos desenvolvidos para o Android podem ser publicados na loja virtual Android Market, que ultrapassou a marca de duzentos e cinquenta mil aplicativos disponíveis em Julho de 2011 e continua num forte crescimento. Tal crescimento é suportado por uma média de mil e quinhentos novos aplicativos lançados a cada dia.

3.3. Computação Autonômica

Kephart e Chess (KEPHART e CHESS, 2003) resumem o desafio da computação autonômica com a seguinte frase: "Os sistemas devem se gerenciar de acordo com as metas do administrador. Novos componentes devem se integrar facilmente assim como uma nova célula se estabelece no corpo humano".

A partir da comparação feita por Kephart e Chess, é possível notar que um sistema autônomo deve estar propenso a realizar as autoadaptações necessárias para reconfigurar seus serviços em resposta às alterações ocorridas no ambiente de rede de forma a se manter acessível aos usuários. Como as alterações necessárias a uma aplicação que deseja recuperar-se da indisponibilidade de um serviço para atender a meta do usuário, que no caso é poder usar o serviço desejado.

A IBM (IBM, 2003) apresenta uma série de estudos sobre computação autonômica dentre eles, "The Vision of Autonomic Computing" e "Autonomic Computing Manifesto", exibindo a visão de uma rede de informações organizada, com componentes computacionais "inteligentes" que nos fornece o que precisamos, quando precisarmos, sem esforço mental consciente ou físico, já que, segundo a mesma, estamos chegando rapidamente ao limite da evolução industrial

que compreende a computação associada à infraestrutura de comunicação, componentes intermediários e serviços. A crescente complexidade do sistema está chegando ao nível além da capacidade humana de garantir uma boa gerência de manutenção e segurança. Ao mesmo tempo, a computação muda de foco, sai do processamento e armazenamento de informações para redes de larga escala suportada por sistema de autogerenciamento e autodiagnóstico a fim de manter um serviço transparente ao usuário.

Por ser um paradigma novo e em evolução, a IBM (IBM, 2003) definiu oito elementos que permitem um melhor entendimento de sistemas autonômicos, conforme é apresentado a seguir:

1 - Um sistema autonômico deve conhecer a si mesmo. Uma vez que o sistema pode existir em diversos níveis, um sistema autonômico necessitará de conhecer seus componentes, estado atual, capacidade e todas suas conexões com outros sistemas;

2 - Um sistema autonômico deve se configurar e reconfigurar de acordo com variações do ambiente. As configurações devem ocorrer de forma automática, assim como os ajustes para se adequar as variações do ambiente;

3 - Um sistema autonômico nunca deve se estabilizar no estado atual. O sistema deve sempre procurar otimizar seu trabalho, monitorando e ajustando seu processo para atingir seus objetivos;

4 - Um sistema autonômico deve realizar algo semelhante à cura. O sistema deverá ser capaz de se recuperar de possíveis falhas de execução. Deve ser capaz de descobrir problemas ou possíveis problemas e encontrar uma alternativa para reconfigurar o sistema de modo a mantê-lo funcionando;

5 - Um sistema autonômico deve ser capaz de se autoproteger. O sistema deve detectar, identificar e agir de forma a se proteger de ataques de outros sistemas mantendo a segurança e integridade de seu sistema;

6 - Um sistema autonômico deve ter conhecimento do ambiente em que está inserido e também do contexto a sua volta. O sistema deverá identificar e gerar regras de forma a interagir e negociar com seus vizinhos para atingir seus objetivos;

7 - Um sistema autonômico não existe em ambientes herméticos. O sistema deve funcionar em ambientes heterogêneos e de implementação de código aberto, ou seja, não poderá ser de tecnologia proprietária;

8 - Um sistema autonômico deve otimizar os recursos necessários enquanto mantém a complexidade escondida. O sistema fará o intermédio entre negócios e usuários de forma a atingir os objetivos estabelecidos pelos usuários, mas sem a intervenção do mesmo.

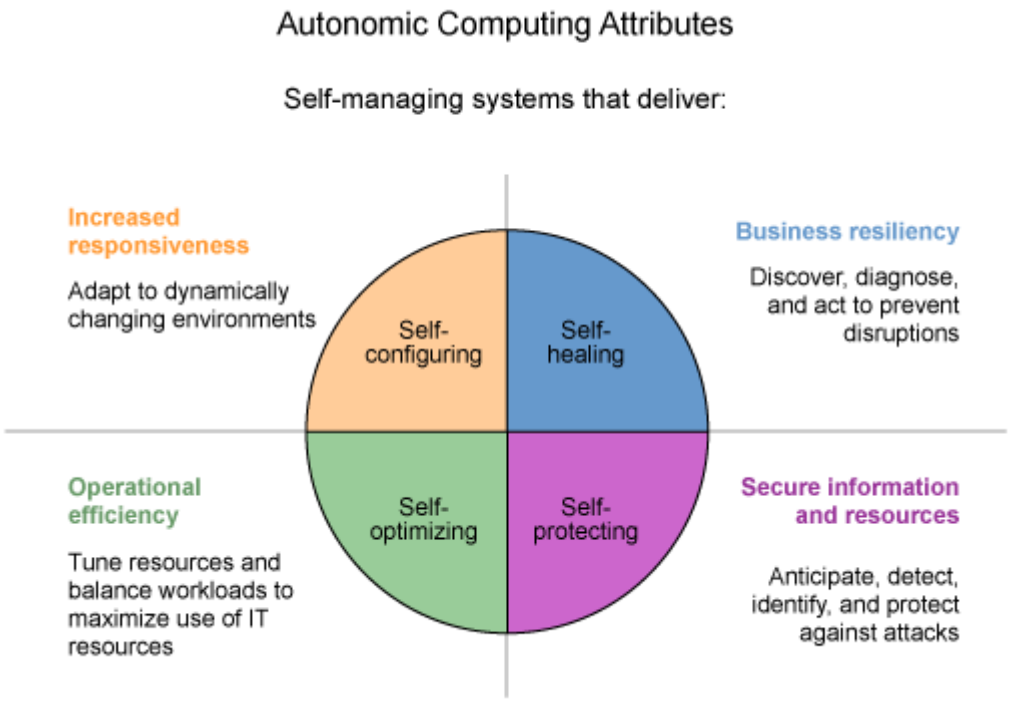

Figura 3.3 – Atributos da computação autonômica (MILLER, 2005).

Tendo em vista os oito elementos, o conceito de autogerenciamento foi detalhado pela IBM (JACOB, LANYON-HOGG, *et al.*, 2004) almejando obter melhor entendimento do conceito, como pode ser observado na Figura 3.3. O autogerenciamento compreende os atributos: autoconfiguração, auto-otimização, autocura e autoproteção. Abaixo temos a descrição dos atributos elaborados:

A Autoconfiguração (*Self-Configuring*) é a capacidade de se configurar dinamicamente, ou seja, um sistema poderá adaptar-se imediatamente – com um mínimo de intervenção – para incorporar novos componentes ou a alterações ocorridas no ambiente.

A Autocura (*Self-Healing*) é a capacidade de detectar problemas de operação e iniciar ações corretivas evitando interrupções de execução do sistema.

Ações corretivas podem ser alterações em seu próprio estado ou provocar mudanças nos outros elementos do ambiente.

A Autoproteção (*Self-Protecting*) é a capacidade do sistema permitir que somente pessoas autorizadas tenham acesso aos dados corretos, no momento certo e tomem os devidos cuidados para tornar o ambiente menos vulnerável a ataques. Este ambiente também deve ser capaz de detectar comportamentos hostis e intrusivos e tomar as devidas ações para se preservar destas ameaças.

A Auto-otimização (*Self-Optimization*) é a capacidade de maximizar de forma eficiente a alocação e utilização dos recursos. Este ambiente deve ser capaz de aprender através de suas experiências e automaticamente e proativamente se otimizar procurando alcançar os objetivos do negócio.

Ainda existem enormes desafios a superar no desenvolvimento e implementação de sistemas autonômicos. Por exemplo, um desafio é a necessidade de agrupar profissionais com conhecimentos técnicos e científicos de diferentes linhas de negócio para que haja troca de experiências e alinhamento de entendimento em relação ao conceito de sistemas autonômicos. Outro desafio é a definição de padrão e tecnologia que dê suporte ao desenvolvimento de sistemas autonômicos capazes de interação eficiente, de agir de forma pró-ativa respeitando as regras de negócios e de se protegerem e curarem sem a intervenção de um ser humano.

A pesquisa em computação autonômica é incentivada por grandes entidades como IBM (IBM, 2003) e IEEE (IEEE, 2012) quando se trata de desenvolvimento de sistemas autoadaptativos. O desenvolvimento de aplicações segundo a disciplina de Engenharia de Software Multiagentes é um dos principais conceitos utilizados para introduzir a autonomia aos sistemas.

3.4. Sistemas Multiagentes

Sistemas multiagentes (SMA) surgiram como uma subárea da Inteligência Artificial Distribuída dedicada à pesquisa e desenvolvimento de soluções para problemas complexos, que não são de fácil resolução, através de algoritmos clássicos da programação (RUSSELL e NORVIG, 2009). Com o objetivo de melhor gerenciar a complexidade inerente à aplicação do conceito de sistemas

autônomos em ambiente de computação móvel, a Engenharia de Sistemas Multiagentes mostra-se mais apropriada visto sua habilidade em agregar "inteligência" ao sistema (LUCENA, 1987), (GARCIA, LUCENA, *et al.*, 2003), (SARDINHA, NOYA, *et al.*, 2006), (SILVA e LUCENA, 2007).

Um SMA é aquele que consiste em um número de agentes, que interagem uns com os outros. No caso mais geral, os agentes atuarão em nome de usuários com diferentes objetivos e motivações. Para interagirem com sucesso, serão necessárias as habilidades de cooperação, coordenação e negociação entre eles, assim como os humanos o fazem (WOOLDRIDGE, 2002).

O conceito chave utilizado nos SMA é uma abstração chamada agente. Um agente é uma entidade autônoma virtual (ou física) capaz de compreender o ambiente ao qual está inserido. Um agente possui a habilidade de se comunicar com outros agentes do sistema para atingir um objetivo comum, que um agente sozinho poderia não ser capaz de alcançar.

Existem duas principais características distintivas para agentes. Em primeiro lugar, tarefas relativamente de alto nível podem ser delegadas aos agentes que irão realizá-las de forma autônoma. Em segundo lugar, os agentes estão situados em um ambiente que pode dinamicamente afetar seu comportamento e estratégia na resolução do problema (JENNINGS e WOOLDRIDGE, 1996).

As propriedades fundamentais que caracterizam um agente são (JENNINGS e WOOLDRIDGE, 2000), (WOOLDRIDGE, 2002):

Autonomia refere-se à capacidade de realizar a maior parte de suas atividades sem a interferência direta de um humano e possuem certo nível de controle sobre suas ações e estados;

Habilidade Social compreende a capacidade de interagir com outros agentes e seres humanos a fim de atingir seus objetivos ou ajudar outros com suas atividades;

Reatividade considera a capacidade de perceber e responder a estímulos do ambiente e a qualquer mudança que nele ocorra;

Pró-atividade expressa a capacidade de responder aos estímulos do ambiente de forma oportuna, orientando-se por seus objetivos e tomando a iniciativa quando apropriado.

Por ser uma entidade com certo grau de complexidade, em 1996 foi criada uma organização responsável por produzir padrões que deveriam ser utilizados na manipulação dos agentes. A organização criada chama-se *Foundation for Intelligent Physical Agents* (FIPA), que definiu uma máquina de estados (FIPA, 2011) com cinco estados, um início e fim para melhor explicar o ciclo de vida de um agente.

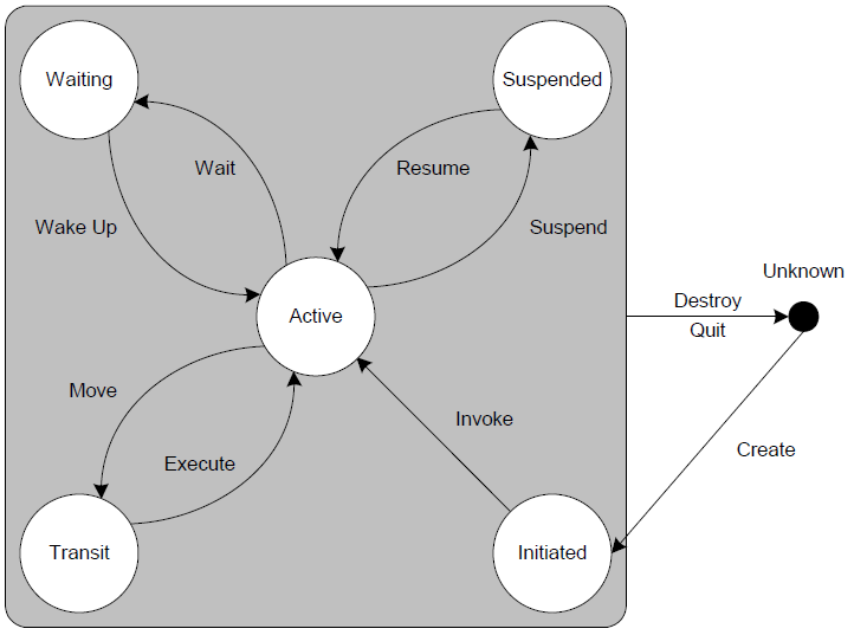

Figura 3.4 – Máquina de estados dos agentes FIPA (FIPA, 2002).

As especificações FIPA (FIPA, 2011) representam um conjunto de normas que se destinam a promover a interação dos agentes e os serviços ao qual eles podem se comunicar. A seguir temos a descrição dos estados dos agentes FIPA.

O estado inicial de todo agente é o *Initiated*, neste estado o agente é construído, mas ainda não está registrado no AMS, portanto não possui nome, nem AID e não pode se comunicar com outros agentes. A partir deste estado o agente pode realizar transições por quatro outros estados que são: *Active* - o agente está registrado no AMS e ativo. Possui um nome e um AID e poderá utilizar todos os serviços da plataforma JADE (JADE, 2011); *Suspended* - o agente está parado, ou seja, não está executando nenhum comportamento; *Waiting* - o agente está bloqueado esperando que alguma condição seja satisfeita para voltar a executar seu comportamento; e *Transit* - o agente entra neste estado quando está migrando de plataforma. Toda mensagem recebida pelo agente neste estado será redirecionada para o novo endereço do agente. O último estado do

agente é o *Deleted*, neste estado o agente está literalmente desativado e não possui mais registro do AMS.

Este trabalho foi elaborado considerando a engenharia de sistemas multiagentes para introduzir o conceito de autonomia nas aplicações para dispositivos móveis de modo que estas superem os desafios da disponibilidade e interoperabilidade na utilização de serviços web. A seguir é apresentado um *framework* elaborado pela Telecom Italia SpA (JADE, 2011), que tem como finalidade auxiliar o desenvolvimento de sistemas multiagentes em dispositivos móveis e foi utilizado no desenvolvimento do *framework* proposto nesta dissertação.

3.4.1. JADE-LEAP

JADE-LEAP (JADE Lightweight Extensible Agent Plataform) é um *framework* que auxilia o desenvolvimento de sistemas multiagentes para dispositivos móveis conforme as normas da FIPA (FIPA, 2011). Devido à necessidade de se construir um *framework* robusto e confiável, o JADE-LEAP utilizou-se das normas FIPA (FIPA, 2011), pois apesar de não ser o único padrão para desenvolvimento de agentes, possui um conjunto de normas bem aceito e que estabelece regras para linguagens de comunicação, protocolos de comunicação, dentre outras.

Este *framework* tem em sua essência a portabilidade em dispositivos móveis uma vez que, seu desenvolvimento foi dedicado a estes dispositivos. Para atingir este objetivo, o JADE-LEAP teve que obedecer às limitações do ambiente móvel e, portanto, podemos citar as seguintes limitações deste *framework*: somente um agente poderá ser executado no modo *split*; agentes executados no modo *split* não podem ser clonados e não possuem a propriedade de mobilidade; a biblioteca Jade *Runtime* será sempre um serviço local; o *container* dos agentes presentes nos dispositivos móveis será exclusivo para os mesmos.

Os agentes JADE-LEAP podem ser executados de duas formas:

Stand-alone : Modo de execução onde um container completo é criado no dispositivo móvel;

Split : Modo de execução onde um container é segregado em duas partes: *FrontEnd*(executado no dispositivo móvel) e *BackEnd*(executado em um servidor remoto), estes serviços são ligados permanentemente através de uma conexão.

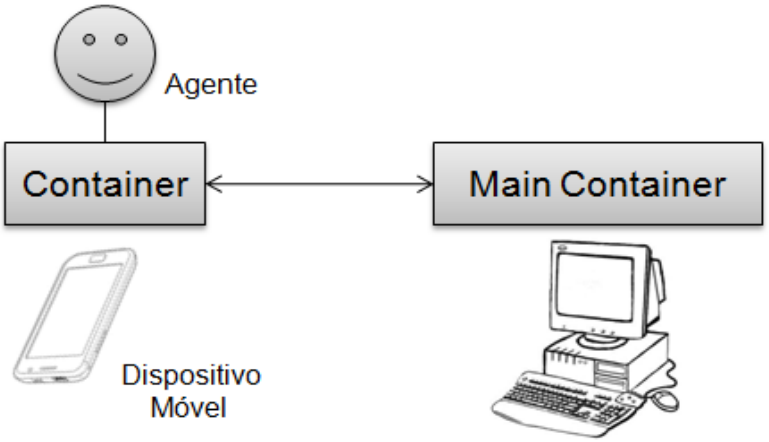

Figura 3.5 – Ilustração da execução no modo *Stand-alone*.

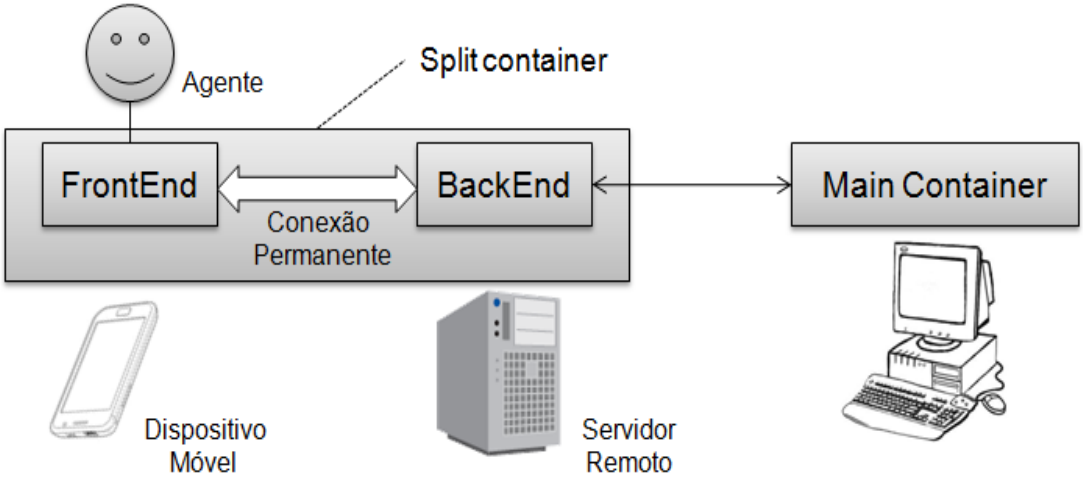

Figura 3.6 – Ilustração da execução no modo *Split*.

As execuções em modo *split* são as mais indicadas para dispositivos móveis e dispositivos que possuam limitações de processamento e memória. O *FrontEnd* é claramente mais leve que um *container* completo; a inicialização do agente no *FrontEnd* é mais rápida e o consumo da banda de internet é menor que no modo *Stand-alone* (WEISS, 1999).

O JADE-LEAP (MORENO, VALLS e VIEJO, 2003) como o próprio nome diz é uma extensão da plataforma JADE (BELLIFEMINE, CAIRE, *et al.*, 2003), (BELLIFEMINE, CAIRE, *et al.*, 2010) em formato mais leve de modo a ser

utilizado para desenvolvimento de sistemas multiagentes em uma variedade enorme de dispositivos como *desktop*, PDA, celulares, *smartphones*, *tablets* dentre outros. Destacamos como pontos positivos deste *framework* sua portabilidade, seu tamanho reduzido, sua adequação a um padrão bem aceito no mercado, o padrão FIPA, sua compatibilidade de estrutura com a plataforma JADE.

Um agente criado através da plataforma JADE-LEAP poderá utilizar dos mesmos mecanismos disponíveis aos agentes JADE, ou seja, o agente receberá um identificador (AID), poderá se registrar na página de serviços (DF) e utilizar o canal de comunicação (MTS). Os componentes da plataforma JADE (BELLIFEMINE, CAIRE, *et al.*, 2003), (BELLIFEMINE, CAIRE, *et al.*, 2010) são descritos a seguir.

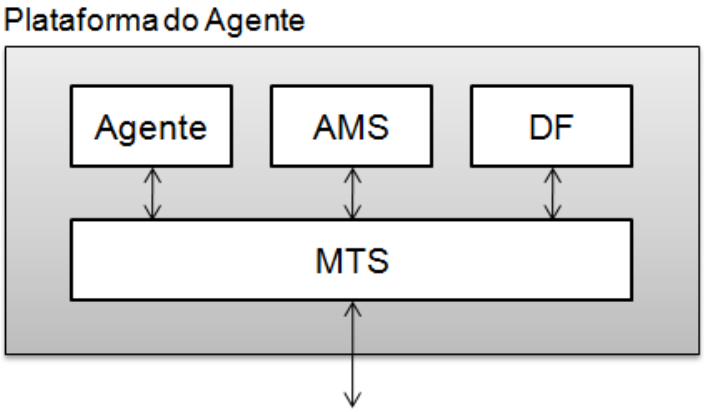

Figura 3.7 – Ilustração simplificada da plataforma do agente **(FIPA, 2002)**.

O *Agent Management System* (AMS) é o responsável pelo controle de acesso dos agentes à plataforma. Através do serviço *white page* o AMS atribui a cada agente um identificador único chamado Agent ID (AID). Todo agente deve se registrar em um AMS para obter um AID válido e assim se integrar à plataforma.

O *Directory Facilitator* (DF) é o responsável por fornecer um serviço chamado *yellow page* em que os agentes podem publicar seus serviços e também procurar por outros agentes que forneçam algum serviço que atenda sua necessidade.

O *Message Transport System* (MTS) é o responsável por controlar toda a troca de mensagens entre os agentes da plataforma e também a troca de informações com serviços externos à plataforma.

Todo agente é reconhecido como uma entidade autônoma, ou seja, capaz de realizar ações sem interferência direta humana. Para atingir tal abstração, os agentes possuem uma propriedade chamada *Behaviour*. *Behaviour* é uma classe estendida do *framework* JADE-LEAP para implementar o comportamento dos agentes. Os comportamentos podem ser variados e, portanto são implementados através da classe abstrata *Behaviour*. Os modelos de comportamentos suportados pelos agentes criados na plataforma JADE-LEAP são: *SimpleBehaviour*, que pode ser implementado através dos comportamentos *OneShotBehaviour, CyclicBehaviour, WakerBehaviour ou TickerBehaviour;* e *CompositeBehaviour* que pode ser implementado através dos comportamentos *SequentialBehaviour, ParallelBehaviour ou FSMBehaviour* (BELLIFEMINE, CAIRE, *et al.*, 2010), (MORENO, VALLS e VIEJO, 2003). Uma descrição mais detalhada de cada comportamento é apresentada a seguir.

O *SimpleBehaviour* é uma classe abstrata que modela comportamentos singulares, ou seja, permite apenas um tipo de comportamento. Este comportamento pode ser expresso através de quatro comportamentos: *OneShotBehaviour* – classe que implementa comportamentos que serão executados uma única vez; *CyclicBehaviour* – classe que implementa comportamentos que serão executados de forma cíclica, ou seja, repetidas vezes sem um fim definido; *WakerBehaviour* – classe que implementa comportamentos que serão executados uma única vez após um determinado período de tempo; e *TickerBehaviour* – classe que implementa comportamentos que serão executados periodicamente, ou seja, executado a cada período de tempo de forma cíclica.

O *CompositeBehaviour* também é uma classe abstrata, mas que modela comportamentos compostos, ou seja, permite que sejam implementados comportamentos compostos de vários outros comportamentos. Desta forma, este comportamento pode ser implementado em três aspectos: *SequentialBehaviour* – classe que executa seus sub-comportamentos de forma seqüencial. Este comportamento só termina após todos seus sub-comportamentos terminarem suas execuções; *ParallelBehaviour* – classe que executa seus sub-comportamentos de forma concorrente. Este comportamento termina sua execução quando algum sub-comportamento atinge uma condição considerada suficiente; *FSMBehaviour* – classe que executa seus sub-comportamentos de acordo com uma máquina de estados finita definida pelo usuário. Este comportamento termina sua execução

quando alcança o estado definido como final ou quando atinge uma condição considerada como suficiente.

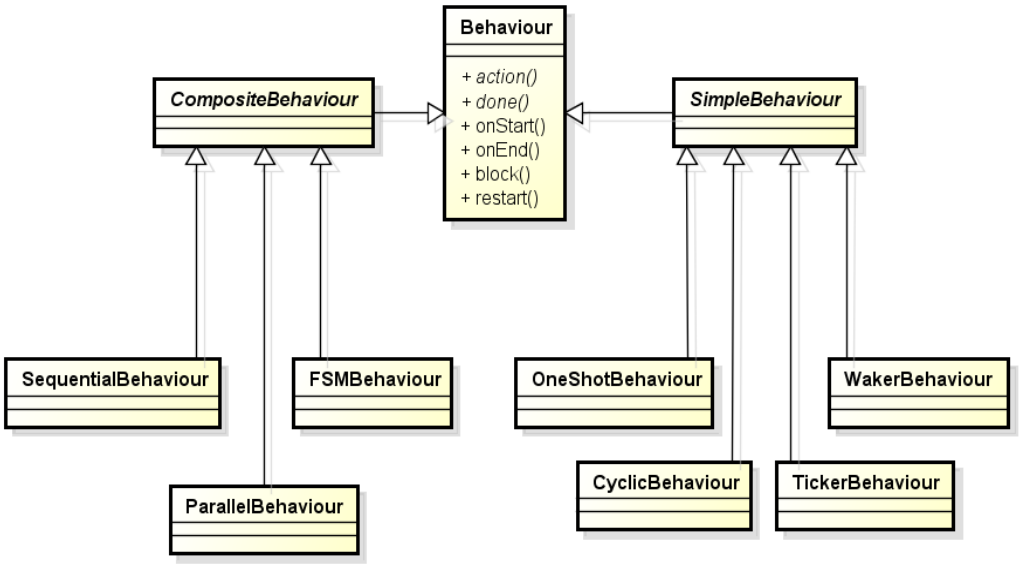

Figura 3.8 – Ilustração da hierarquia de comportamentos de um agente.

Para adicionar os agentes JADE-LEAP às aplicações do Android faz-se necessário a utilização da classe *JadeGateway*. Esta classe atua como ponto de conexão entre a aplicação e o sistema multiagentes, provendo uma comunicação simples e efetiva entre as partes. É através dela que se faz o gerenciamento de ativação e desativação dos agentes e também onde se inicializa o *split* container. Os comportamentos definidos para o sistema multiagentes da aplicação serão executados pelo método *execute* da classe *JadeGateway*.

A crescente presença dos dispositivos móveis no dia a dia expõe desafios interessantes em relação à troca de informações entre estes dispositivos e serviços web. A primeira dificuldade a ser superada são as limitações processamento, memória, armazenamento, energia, dentre outras dos dispositivos móveis. Outro desafio está relacionado à utilização de serviços web através dos dispositivos móveis. Neste caso, o foco deste trabalho, são os problemas de indisponibilidade dos serviços web e a interoperabilidade dos dados na troca de informações com serviços similares, que utilizam protocolos de dados distintos.