UNIVERSIDADE FEDERAL DE SANTA CATARINA PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIAS DA COMPUTAÇÃO

TIAGO MAZZUTTI

UM MODELO PARA DESENVOLVIMENTO DE SISTEMAS MULTIAGENTE PLENAMENTE DISTRIBUÍDOS

Orientador: Profº Ricardo Azambuja Silveira, Dr.

Florianópolis, 2011

Catalogação na fonte pela Biblioteca Universitária
da
Universidade Federal de Santa Catarina

M478m Mazzutti, Tiago
Um Modelo para Desenvolvimento de Sistemas Multiagente Plenamente Distribuídos [Dissertação] / Tiago Mazzutti; orientador, Prof° Ricardo Azambuja Silveira, Dr. - Florianópolis, SC, 2011.

75 f.: il., tabs.

Dissertação (Mestrado) - Universidade Federal de Santa Catarina, Centro Tecnológico. Programa de Pós-Graduação em Ciências da Computação.

Inclui referências

1. Ciência da computação. 2. Sistemas multiagentes. 3. World Wide Web (Sistema de recuperação da informação). 4. Java (Linguagem de programação de computador). 5. JavaScript (Linguagem de programação de computador). I. Silveira, Ricardo Azambuja. II. Universidade Federal de Santa Catarina. Programa de Pós-Graduação em Ciência da Computação. III. Título.

CDU 681

TIAGO MAZZUTTI

UM MODELO PARA DESENVOLVIMENTO DE SISTEMAS MULTIAGENTE PLENAMENTE DISTRIBUÍDOS

Esta dissertação foi julgada adequada para obtenção do Título de "Mestre em Ciências da Computação", e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciências da Computação da Universidade Federal de Santa Catarina.

Florianópolis, 28 de novembro de 2011

_____

Profº Mário Antônio Ribeiro Dantas, Dr.
Coordenador do Curso

Banca Examinadora:

_____

Profº Ricardo Azambuja Silveira, Dr.
Orientador

_____

Profº João Carlos Gluz, Dr.
Universidade do Vale do Rio dos Sinos

_____

Profº Mauro Roisenberg, Dr.
Universidade Federal de Santa Catarina

_____

Profª Sílvia Modesto Nassar, Dra.
Universidade Federal de Santa Catarina

Dedico este trabalho à minha família, em especial a minha esposa Samantha e minha filha Clara que foram compreensivas e estiverem sempre presentes me apoiando e me amparando no momentos difíceis.

AGRADECIMENTOS

Agradeço primeiramente a Deus, pela família que me deu, pelas oportunidades que me ofereceu, pelas pessoas que colocou no meu caminho, por estar sempre me concedendo graças e me dar a força necessária nos momentos difíceis, especialmente quando achei que não conseguiria terminar este trabalho. Graças a Ele posso considerar-me uma pessoa privilegiada.

Agradeço à minha esposa, Samantha Lemke, meu porto seguro, pelo seu constante apoio, amor e dedicação; a minha filha Clara Lemke Mazzutti, por sua compreensão, carinho e afeto; à minha mãe, Vilde Inês Mazzutti, e as minhas irmãs, Tainara Mazzutti e Taise Mazzutti, por sempre me apoiar e amar. O que posso dizer-lhes é: "Amo muito vocês. Tenho muita sorte por tê-las na minha vida".

Agradeço ao Professor Doutor Ricardo Azambuja Silveira, pelo seu incentivo, orientação competente e relevantes ensinamentos que me proporcionou no decorrer desta pesquisa. Agradeço-lhe, também, pelo seu exemplo de profissionalismo, dedicação, disponibilidade e amizade.

Agradeço à Universidade Federal de Santa Catarina, pela oportunidade de realizar o Mestrado.

Agradeço a Coordenação, ao corpo docente, colegas do Programa de Mestrado em Ciências Computação e aos colegas do laboratório Inteligência Artificial e Tecnologia Educacional, da Universidade Universidade Federal de Santa Catarina.

Agradeço a CAPES e ao REUNI-MEC pelo apoio financeiro.

Agradeço a todas as pessoas que, direta ou indiretamente, contribuíram para a realização deste trabalho.

RESUMO

MAZZUTTI, Tiago. Um Modelo para Desenvolvimento de Sistemas Multiagente Plenamente Distribuídos. Florianópolis, 2011. 75 f. Dissertação (Mestrado) Programa de Pós-Graduação em Ciências da Computação, UFSC, Florianópolis – SC.

Para obtermos sistemas multiagente plenamente distribuídos na *Web* é necessário a possibilidade de os agentes serem executados na máquina do cliente, de forma a satisfazer plenamente as necessidades da aplicação, contornando os problemas usualmente causados por questões como segurança e concorrência, comuns nos sistemas atualmente disponíveis. As soluções atuais envolvem principalmente a execução através de *applets*, que dependem da extensão, instalação ou configuração de *plugins* por parte do usuário, ou por meio de aplicações completamente dependentes de um servidor, que contam apenas com a camada *thin*, no lado do cliente. Este trabalho apresenta, com base em estudos das soluções existentes, um modelo para o desenvolvimento de sistemas multiagente plenamente distribuídos. Na solução apresentada o sistema multiagente é desenvolvido utilizando-se da linguagem *Java* e compilado através de compilação cruzada para *JavaScript*, onde o código final gerado é compatível com a maioria dos navegadores modernos, deixando o desenvolvedor em um nível mais abstrato de desenvolvimento. Nessa solução o desenvolvedor pode também utilizar outras bibliotecas ou recursos da linguagem *Java*, bastando que essas sejam compatibilizadas com a ferramenta utilizada. A solução apresentada resulta em um ambiente que permite a criação de sistemas multiagente plenamente distribuídos de forma que aplicações possam ser executadas através de agentes que atuam utilizando recursos do lado dos clientes e servidores distribuídos em diversas máquinas, procurando equilibrar a distribuição de processamento e permitindo aplicações mais robustas.

PALAVRAS-CHAVE: sistemas multiagente, *client-side*, *web toolkits*.

ABSTRACT

MAZZUTTI, Tiago. *A Framework for Developing Multiagent Systems with the Web as Platform*. Florianópolis, *2011*. 75 f. *Thesis (Masters) Post-Graduate in Computer Scienci*, UFSC, Florianópolis – SC.

To get fully distributed multi-agent systems on the Web we need the ability to run agents on the client machine, in order to satisfy the needs of the application, and bypassing the problems usually caused by issues such as security and competition in systems currently available. Current solutions involve either the execution of applets - which depends on software extensions, plugin installation or configuration by the user - or applications completely dependent on a server, with only a thin layer running on the client side. This work presents, based on studies of existing solutions, a proposal to implement an execution environment for fully distributed agents. The proposed multi-agent system is developed using the Java language and then cross-compiled to JavaScript. Thus, the generated code is compatible with most modern browsers, while leaving the developer at a more abstract level of development. In this solution the developer can also use other libraries or features of the Java language, provided that these are aligned to the tool used. The solution presented results in an environment that allows the creation of fully distributed multi-agent systems. In this way, applications can be implemented by using agents that utilize client-side resources and distributed servers on multiple machines, therefore improving load-balancing and enabling more robust solutions.

KEYWORDS: *multi-agent systems, fully distributed, client-side, web toolkits.*

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

AID	Identificador do Agente (*Agent Identifier*)
AJAX	*Asynchronous Javascript and XML*
AMS	*Agent Management System*
API	*Application Programming Interface*
CP	Compatibilidade
CSS	*Cascading Style Sheets*
DF	*Directory Facilitator*
DOM	*Document Object Model*
ET	Extensibilidade
FC	FIPA-Compatível
FD	Ferramenta de Depuração (*Debugger*)
FIPA	*Fondation for Intelligent Physical Agents*
GWT	*Google Web Toolkit*
HTML	*HyperText Markup Language*
HTTP	*Hypertext Transfer Protocol*
IA	Inteligência Artificial
IDE	Ambiente de desenvolvimento Integrado (*Integrated Development Environment*)
LPA	Linguagem para Programação do Agente
MB	Mobilidade
MD	Metodologia de Desenvolvimento
MTS	*Message Transport Service*
NPAPI	*Netscape Plugin Application Programming Interface*
PDF	*Portable Document Format*
PMA	Plataforma Multiagente
POA	Programação Orientada a Agentes
POO	Programação Orientada a Objetos
RDF	*Resource Description Framework*
SaaS	Software como um serviço (*Software as a Service*)
SG	Segurança
SMA	Sistemas Multiagente
SQL	*Structured Query Language*
SVG	*Scalable Vector Graphics*
URI	*Uniform Resource Identifier*
W3C	*World Web Consortium*
WANs	*Wide Area Networks*
WHATWG	*Web Hypertext Application Working Group*
WWW	*World Wide Web*
XHTML	*Extensible HyperText Markup Language*

SUMÁRIO

1 INTRODUÇÃO

Desde o seu surgimento, a *Web* e seus serviços seguem uma linha evolucionária onde suas características estão indo cada vez mais ao encontro às já bem definidas e nativamente presentes no paradigma de Agentes e Sistemas Multiagente (SMA).

O conceito de serviço virtual está amplamente disseminado na *Web* atual. Colaboração, cooperação e interação são palavras-chave na descrição de tais serviços. Aplicativos, geralmente distribuídos, tiram proveito da inteligência coletiva e se tornam melhores a medida que são utilizados. A informação disponível na rede é apresentada de forma organizada e personalizada aos internautas.

Neste contexto, o conhecimento de clientes, usuários e consumidores com suas respectivas necessidades, diferentemente da forma tradicional de desenvolvimento de aplicações para *Web*, pode ser representado em mais alto nível e de maneira relativamente fácil na definição de um agente e em sua aprendizagem contínua. Agentes são projetados para conciliar ontologias facilmente, ao contrário do que se vê nos serviços *web*. Agentes são intrinsicamente pró-ativos e comunicativos, enquanto os serviços *web* são passivos, até serem chamados. Um serviço *web*, tal como é definido e usado atualmente, não é autônomo. A autonomia é uma característica de agentes, e é também uma característica em muitas circunstâncias nas aplicações baseadas na *Internet*.

Com a Programação Orientada a Agentes (POA) como paradigma para o desenvolvimento de aplicativos *web*, além das vantagens acima citadas, promove-se uma alternativa a atual arquitetura cliente-servidor, principalmente no lado do cliente, reduzindo-se a sobrecarga da rede.

Este trabalho apresenta um modelo para o desenvolvimento de sistemas multiagente plenamente distribuídos. Na solução proposta o sistema multiagente é desenvolvido utilizando-se a linguagem *Java* e é compilado através de compilação cruzada para *JavaScript*, sendo que o código final gerado é compatível com a maioria dos navegadores modernos, deixando o desenvolvedor em um nível mais abstrato de desenvolvimento. Nessa solução o desenvolvedor pode também utilizar outras bibliotecas ou recursos da linguagem *Java*, bastando para tal, que essas sejam compatibilizadas com a ferramenta utilizada. Este trabalho resulta em um ambiente que permite a criação de sistemas multiagente plenamente distribuídos de forma que aplicações possam ser executadas através de agentes que atuam utilizando recursos nativamente disponíveis no lado dos clientes e em servidores distribuídos em diversas máquinas, procurando equilibrar a distribuição de processamento e permitindo desta forma, a crição de aplicações mais robustas.

1.1.1 Sistemas Multiagente e o Futuro da *Web*

De acordo com Hendler (2001), Jennings (2003) e Huhns (2010), a *Web* está evoluindo de um ambiente onde as pessoas obtêm informação para um ambiente onde os computadores executam tarefas em nome de seus usuários.

A seguir, de acordo com autores acima citados, apresentam-se alguns desafios para a implementação de *Web* corrente e seus padrões:

Informação semântica: prestadores e utilizadores de serviços *web* devem concordar quanto a semântica da informação que está sendo trocada.

Colaboração: mesmo que se trate de protocolos simples, para que estes executem de forma confiável, prestadores de serviços devem garantir que as partes, em uma iteração, cheguem a um acordo quanto sua iteração atual e onde querem chegar. Isto requer elementos de trabalho em equipe como: persistência das decisões, capacidade de gerir o contexto e repetir operações.

Interesses autônomos: os serviços devem ser capazes de participar de forma automatizada em processos de mercado, onde, vários mecanismos são necessários para uma participação efetiva. Esta habilidade requer: fixar os preços, dar lances, aceitar ou rejeitar lances, e acomodar riscos.

Personalização: a utilização efetiva dos serviços providos pela *Web*, muitas vezes exige personalizações e composições de uma forma que seja sensível ao contexto, especialmente com relação às necessidades do usuário. Isso requer aprendizagem das preferências do cliente, interações de iniciativa mista, oferecendo orientação para o cliente (melhor se não for intrusiva), permitindo que um usuário interrompa o serviço, e garantir que a autonomia do usuário não seja violada.

Condições de exceção: para que sejam criadas empresas virtuais dinamicamente a fim de proporcionar bens e serviços mais apropriados a clientes em comum, é necessária a capacidade de construção de equipes, entrar em acordos multipartidários, gerenciar autorizações e comprometimentos , e acomodar exceções.

Localização de serviços: recomendações devem ser fornecidas para ajudar os usuários a encontrar serviços de qualidade, relevância e de confiança. Isso requer um meio para obter e/ou acrescentar avaliações.

Tomada de decisão distribuída: a tomada de decisão pode ser distribuída entre vários serviços, o que exige decisões inteligentes sobre o que e como cada serviço pode contribuir adequadamente.

Ainda, segundo Huhns (2010), com a preocupações listadas acima sendo tratadas em esforços de pesquisa, a *Web* vai evoluir de passiva para ativa,

da arquitetura cliente - servidor para a *peer-to-peer* ou cooperativa, de serviços para processos, da semântica para a compreensão mútua, pragmática e cognitiva.

1.1.2 Agentes *versus* Serviços *web*

Seguindo Huhns (2010), as arquiteturas de agentes típicas têm as mesmas funcionalidades dos serviços *web*. Arquiteturas de agentes oferecem serviços de páginas amarelas e páginas brancas, onde agentes podem anunciar suas funcionalidades distintas e pesquisar por funcionalidades que outros agentes forneçam. Entretanto, o paradigma de agentes pode estender os serviços da *web* em vários aspectos importantes:

- Um serviço *web* só conhece a si mesmo, mas não sobre os usuários/-clientes/consumidores. Muitas vezes os agentes são auto-conscientes em um meta-nível e através de aprendizagem e construção de modelos podem ganhar consciência sobre outros agentes e suas capacidades de interações. Isso é importante, porque sem tal consciência, um serviço *web* não seria capaz de tirar proveito das novas capacidades em seu ambiente, e não poderia personalizar os serviços para um determinado usuário, bem como fornecer melhores serviços aos usuários mais frequentes.

- Serviços *web*, ao contrário de agentes, não são projetados para usar ou conciliar ontologias. Se o cliente e o prestador do serviço utilizarem ontologias diferentes, então o resultado ao se utilizar o serviço *web* será incompreensível para o cliente.

- Agentes são intrinsecamente comunicativos, enquanto os serviços *web* são passivos até serem chamados. Agentes podem fornecer alertas e atualizações quando novas informações ficam disponíveis. Os padrões e protocolos atuais não prevêem ainda a subscrição em um serviço para recebimento de atualizações periódicas.

- Um serviço *web*, tal como atualmente definido e usado, não é autônomo. A autonomia é uma característica de agentes, e é também uma característica em muitas circunstâncias nas aplicações baseadas na Internet. No meio de agentes, autonomia geralmente se refere à autonomia social, um agente tem conhecimento de seus colegas e é sociável, mas mesmo assim exerce sua independência em determinadas circunstâncias. A autonomia está em tensão natural com a coordenação ou com a noção de compromisso. Para estar coordenado com outros agentes ou para manter seus compromissos, um agente deve abandonar algumas de suas autonomias. No entanto, um agente que é sociável e responsável pode ainda continuar autônomo.

- Agentes são cooperativos, e formando equipes e coligações podem oferecer serviços de mais alto nível e mais abrangentes. Os padrões atuais

para serviços da *Web* não fornecem padrões para compor funcionalidades.

1.2 OBJETIVO GERAL

O objetivo geral deste trabalho é a proposição de um modelo que permita que sejam criados aplicativos de *software* plenamente distribuídos, seguindo-se o paradigma de agentes, tendo a *Web* como plataforma, aproveitando-se adequadamente os recursos disponíveis no lado do cliente.

1.2.1 Objetivos específicos

1. Elaborar um *framework* que utilize recursos de computação disponíveis no lado do cliente, reduzindo a dependência das aplicações *web* e a sobrecarga na rede experimentada pela arquitetura cliente-servidor tradicional.

2. Permitir que aplicações *web* evoluam e se adaptem as necessidades de seus usuários a medida que forem utilizadas pelos mesmos.

3. Prover maior autonomia e proatividade as aplicações *web* utilizando-se dos conceitos e abstrações em alto nível do paradigma de agentes.

4. Reduzir a complexidade do desenvolvimento de aplicações *web*, ditadas pelas tendências atuais.

5. Capturar e modelar nativamente o processamento concorrente entre tarefas nas aplicações *web* no lado do cliente.

1.3 ESTRUTURA

Este trabalho está organizado da seguinte maneira: no cápitulo 1 é apresentada a motivação e os objetivos deste trabalho. No capítulo 2, apresenta-se um relato sobre a evolução da *Web*, sobre o desenvolvimento de aplicativos *web* no lado do cliente e um estudo relatando o estado da arte para o paradigma de agentes. Seguindo, no capítulo 3 é apresentada a metodologia utilizada neste trabalho e a seguir o capítulo 4, trata sobre a proposição de uma solução para o problema tratado neste trabalho. No capítulo 5 são apresentados os resultados e para finalizar, no capítulo 6 são relatadas as conclusões e trabalhos futuros.

2 REFERENCIAL TEÓRICO & TECNOLÓGICO

2.1 A EVOLUÇÃO DA *WEB*

Nesta seção serão apresentadas as definições, as principais características, exemplos de aplicações, etc, seguindo-se a linha evolucionária da *Web* 1.0 à futura *Web* 4.0, que podem ser classificadas como eras da *Internet*.

A chamada *Web* 1.0 foi a primeira geração de *Internet* comercial. Seu grande trunfo era a quantidade de informações disponíveis. Segundo O'Reilly (2005), os *sites* da *Web* 1.0 eram estáticos, contendo informações que podiam ser úteis, mas não existia razão para que um possível visitante retornasse ao mesmo mais tarde. Um exemplo pode ser uma página pessoal que ofereça informações sobre o dono do *site*, mas que não mude nunca. Os visitantes podem acessá-la, mas não modificá-la e nem contribuir.

Na *Web* 1.0, a maioria das organizações possuiam páginas de perfis que visitantes podiam consultar mas não fazer alterações. Sob a filosofia da *Web* 1.0, as empresas desenvolviam aplicativos de *software* que os usuários podiam baixar, mas não eram autorizados a ver como o aplicativo funcionava e nem a alterá-lo. O *Netscape Navigator*, por exemplo, era um aplicativo fechado da era da *Web* 1.0.

Dessa forma, a *Web* 1.0 pode ser definida como a primeira fase da *Web*, que se estendeu, principalmente, durante a década de 90. Caracterizada por um "*read-only web*", em que o internauta participava, atuando como mero espectador, sem ter condições de desenvolver o conteúdo dos sites que visitava (LISBOA, 2009).

O termo *Web* 2.0 é utilizado para descrever a segunda geração da *World Wide Web* (WWW), que possui uma tendência de reforçar o conceito de troca de informações e colaboração dos internautas com *sites* e serviços virtuais. Sua essência é permitir que os usuários sejam mais do que meros espectadores, ou seja, que eles sejam parte do espetáculo. Os melhores *sites* são os que possuem ferramentas para que os internautas gerem conteúdo, criem comunidades e interajam. Alguns, como a *Wikipédia*, possibilitam a construção coletiva do conhecimento (CORMODE; KRISHNAMURTHY, 2008; JIMENEZ; BARRADAS, 2010).

Segundo O'Reilly (2005), a *Web* 2.0 tem como principal conceito a *Web* como plataforma, possuindo como competências centrais: serviços ao invés de *softwares* empacotados; uma arquitetura de participação, onde os próprios usuários participam da construção de suas ferramentas; escalabilidade de custo eficiente, ou seja, a medida que o tamanho do aplicativo/serviço aumenta, seu custo total diminui proporcionalmente; fonte e transformação de dados manipuláveis; *softwares* distribuídos em mais de um dispositivo e emprego da inteligência coletiva, através das possíveis interações entre os usuários.

Posteriormente, O'Reilly (2006), redefiniu a *Web* 2.0 como uma revolução na indústria de computadores causada pelo ponto de vista da *Internet*

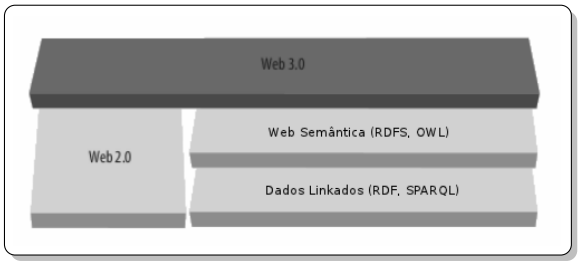

Figura 2.1: *Web* 3.0 estendendo a *Web* 2.0 e a *Web* Semântica (HENDLER, 2009)

como uma plataforma, e uma tentativa de entender as regras para obter sucesso nesta nova plataforma. A principal regra é construir aplicações que aproveitem os efeitos da *Internet* para se tornarem melhores assim que as pessoas as utilizem.

Dessa forma, a *Web* 2.0 representa a segunda década da *Web* (2000-2009), que foi caracterizada por uma mudança no uso da *Web*, permitindo que os usuários conectem-se aos *sites* e acrescente-lhes novas informações, através de mecanismos de interação. Isto pode ser chamado de *"read-write web"* (LISBOA, 2009).

Hendler (2009), relata que existe uma confusão sobre como se chamar a onda crescente de inovação, que é realizada em parte pela *Web* semântica, expressada por Berners-Lee, Hendler e Lassila (2001) em "*The semantic web: Scientific american*". Devido ao fato dessas tecnologias serem largamente baseadas em uso de conteúdo de mais de uma fonte para criar um novo serviço completo, chamados de *mashups*, que ocorrem no nível dos dados, ao invés de no nível de aplicação, e frequentemente envolvem a natureza de leitura/escrita dos aplicativos da *Web* 2.0, existe uma tendência em se chamar esse novo estágio evolucionário da *Web* de *Web* 3.0. Dessa forma, a *Web* 3.0 pode ser vista, essencialmente, como a *Web* Semântica integrada em aplicações *Web* 2.0 de larga escala.

Sendo assim, a *Web* 3.0 representa a década atual da *Web* (2010-2019). Ela é caracterizada por um *"read-write-execute web"*, onde se vê uma proliferação dos chamados *Softwares as a Service* (SaaS). De acordo com Lisboa (2009), a *Web* 3.0 representa um conjunto de tecnologias que possuem formas eficientes para ajudar os computadores a organizar as informações disponíveis na rede. Analisa-se muito mais informações com percentagem mínima de esforço e com resultados mais precisos.

Para entender a *Web* 3.0 é precisa explicar o que é a *Web* Semântica. O objectivo *Web* Semântica é estender os princípios da *Web* baseada em documentos para uma *Web* baseada em documentos e dados. Os dados devem ser acessados usando a arquitetura geral da *Web*, por exemplo, usando URIs (*Uniform Resource Identifier*) , e devem ser relacionados uns aos outros assim como os

documentos (ou partes de documentos). Isso significa também que é necessária a criação de uma estrutura comum que permita que dados sejam compartilhados e reutilizados além dos limites da aplicação, das empresas e comunidades, processados automaticamente por ferramentas, bem como manualmente, inclusive levantando possíveis novas relações existentes entre os mesmos ou parte deles (HERMAN, 2004; BERNERS-LEE; HENDLER; LASSILA, 2001).

A figura 2.1 mostra a aplicação destas tecnologias integradas com *frameworks* que potencializam as já bem conhecidas aplicações da *Web* 2.0, e estão, de modo geral, sendo definidas como geração *Web* 3.0. A base das aplicações *Web* 3.0 reside no *Resource Description Framework* (RDF) que disponibiliza meios para se ligar dados advindos de múltiplos *websites* ou diferentes bancos de dados, e na linguagem de consulta SPARQL (HOMMEAUX; SEABORNE et al., 2006), que segue o padrão SQL (*Structured Query Language*) e permite que sejam feitas consultas nos dados RDF. Mais informações podem ser encontradas em Hendler (2009).

A *Web* 4.0 será baseada em sistemas operacionais na *Web*. Será um sistema inteligente que englobará várias tecnologias. Lisboa (2009) relata que ocorrerá uma transformação global de natureza social, indústrial e política. A *Web* 4.0 atingirá uma massa crítica de participação em redes *online* que oferecerão transparência global, governança, distribuição, participação, colaboração na indústria, redes políticas e sociais e outros esforços importantes da comunidade.

A *Web* 4.0, de forma simplificada, será como um gigantesco sistema operacional inteligente e dinâmico, que irá suportar as iterações dos indivíduos, utilizando os dados disponíveis, instantâneos ou históricos, para propor ou suportar a tomada de decisão. A grande diferença entre tudo o que existe hoje e nos próximos anos (2020-2030) é que isso acontecerá automaticamente, com base num complexo sistema de inteligência artificial (GODIN, 2007).

Para finalizar esta seção, a figura 2.2 permite visualizar a trajetória, evolução das versões da *Web* e, bem como, as tecnologias empregadas em cada versão. Como pode ser observado no gráfico, existe um tendência de aumento da complexidade e necessidade de recursos para suportar a constante evolução da *Web*. Sendo assim, pesquisadores da área procuram constantemente novas metodologias e técnicas que tornem essa evolução possível e é nesse contexto que a Inteligência Artificial (IA) está inserida.

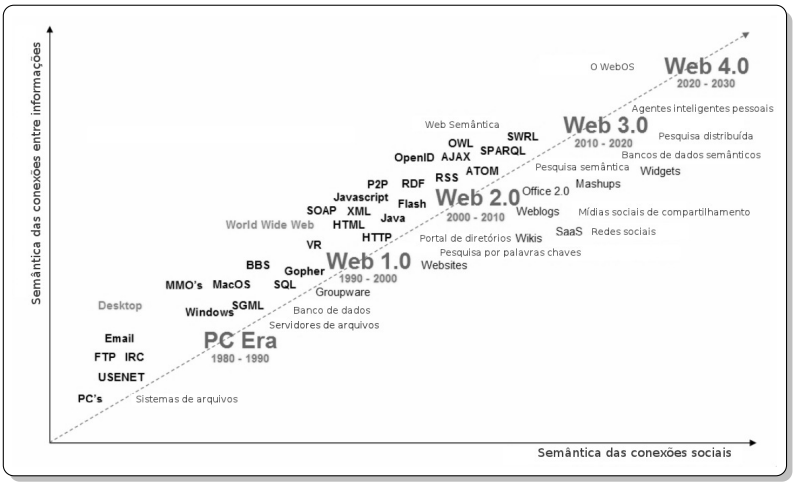

Figura 2.2: A evolução da *Web*
Fonte: Radar Networks & Nova Spivack, 2010 - www.radarnetworks.com

2.2 APLICAÇÕES WEB NO LADO DO CLIENTE

Aplicações *web* no lado do cliente são tipicamente pequenas e independentes, sendo utilizadas para renderizar e atualizar dados remotos. Essas aplicações são formatadas de maneira que um único *download* e uma única "instalação" seja feita na máquina do cliente (W3C, 2011a). Como os criadores de páginas HTML (*HyperText Markup Language*), os autores de aplicações *web* no lado do cliente dependem de muitos formatos de arquivos e especificações para estruturar, implementar e empacotar suas aplicações. A imagem na figura 2.3 apresenta uma estrutura típica para uma aplicação *web* no lado do cliente.

A camada mais abaixo da Figura 2.3, representa o ambiente de execução no hospedeiro (*host runtime environment*). Esta camada é diretamente construída, em um navegador de internet, ou provê suporte similar. A maioria dos ambientes de execução fornecem suporte para o protocolo de transferência de hipertexto HTTP (*Hypertext Transfer Protocol*), URI, *Unicode*, uma linguagem de programação baseada em *scripts* chamada de *ECMAScript*, que é usada de base para a criação da linguagem *JavaScript* (ASSEMBLY, 1999) , CSS (*Cascading Style Sheets*), DOM (*Document Object Model*), que oferece uma maneira dinâmica para se alterar o conteúdo de documentos eletrônicos (uma página HTML, por exemplo), e algum mecanismo para renderizar multimídia e sons. Com o advento do HTLM5, esta camada também inclui uma API que provê funcionalidades como: banco de dados, acesso controlado de leitura/escrita a uma porção do sistema de arquivos local, *XMLHttpRequest*

(KESTEREN; JACKSON, 2007), *websockets*, *web workers* (LUBBERS; ALBERS; SALIM, 2010), acesso ao *hardware*, etc, (W3C, 2011c).

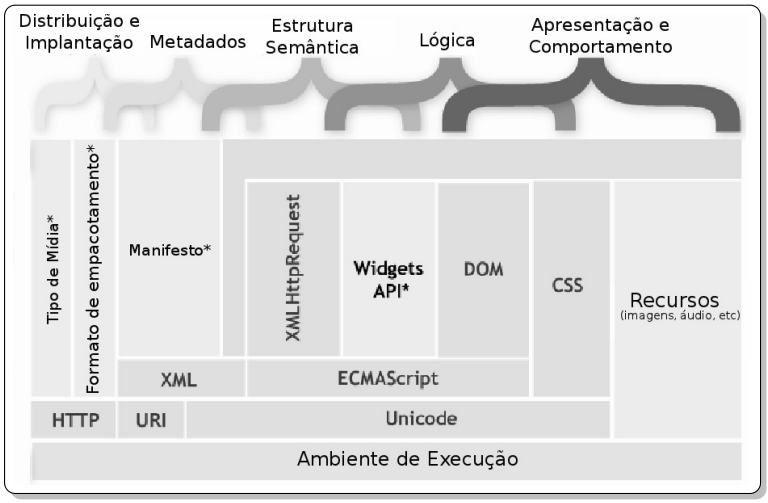

Figura 2.3: Arquitetura típica de uma aplicação *Web* no lado do cliente (*Fonte*: W3C (2011a)

2.2.1 HTML5

A primeira versão de HTML, segundo Lubbers, Albers e Salim (2010) foi publicada em 1993. Durante os anos 90 foram publicadas as versões 2.0, 3.2 e 4.0. Em 1999 foi publicada a versão 4.01 e a W3C (*World Web Consortium*) assumiu o controle sobre as especificações.

Com o intuito de elevar a plataforma *web* para um novo nível, um pequeno grupo denominado *Web Hypertext Application Working Group* (WHATWG) foi criado em 2004. Este grupo foi responsável pela especificação do HTML5 e também iniciou trabalhos para agregar novas características à áreas da *Web* consideradas pouco avançadas. Foi nesta época que o termo *Web* 2.0 foi cunhado (LUBBERS; ALBERS; SALIM, 2010).

Em 2008, a W3C publicou a primeira versão (*working draft*) do HTML5. Pelo fato do HTML5 resolver problemas realcionados a: acesso a arquivos e bancos de dados locais, acesso ao hardware, comunicação em tempo real, CSS3, acesso *offline*, *web* semântica, gráficos e multimídia, etc, os criadores e mantenedores de navegadores de internet estão implementando-a rapidamente (LUBBERS; ALBERS; SALIM, 2010).

2.2.2 Navegadores *Web*

Um navegador de internet (*web browser*) é uma aplicação de *software* com a funcão de recuperar, apresentar e navegar através da *World Wide Web*. Um recurso de informação é identificado através de uma URI e pode ser uma página *web*, imagem, vídeo, etc. *Hyperlinks* presentes, permitem que os usuários naveguem facilmente através dos conteúdos. Um navegador de internet pode ser definido também como um programa que habilita seus usuários a acessar, recuperar e visualizar documentos, entre outros recursos na Internet. Embora os navegadores estão primeiramente voltados ao acesso de conteúdo da WWW, eles também podem ser usados para acessar informações providas por servidores *web* em redes privadas ou em sistemas de arquivos (ROSENFELD; MORVILLE, 1998; JACOBS; WALSH, 2004).

Em navegadores *web* mais modernos como por exemplo *Google Chrome*, *Firefox*, *Safari* estão presentes algumas características que seguem:

- Renderização de páginas *web* consistente e de acordo com o padrão estalecido pela W3C;
- Interfaces receptivas e intuitivas;
- Suporte avançado para HTML5 e CSS3; e
- *JavaScript* com alta performance.

As páginas *web* tem se tornado cada vez mais avançadas, apoiando-se em AJAX (*Asynchronous Javascript and XML*) e *frameworks JavaScript* para proporcionar iteratividade e receptividade. Navegadores modernos, melhoraram muito a performance de execução de *JavaScript* em relação aos navegadores do passado, permitindo aos desenvolvedores *web* utilizar *JavaScript* cada vez mais avançado, assegurando uma boa experiência de uso (NEMETH, 2009).

2.2.3 *Web browser engine*

Todo navegador de internet pussui internamente uma *web browser engine* que provê recursos para o acesso e renderização de documentos HTML, XHTML (*Extensible HyperText Markup Language*), SVG (*Scalable Vector Graphics*), e interpretação de *JavaScript*, etc. Uma *web browser engine* pode ser incorporada em um aplicativo *desktop*, possibilitando que o mesmo tenha acesso e apresente conteúdos da *Web*. Uma ponte entre o ambiente de execução do *JavaScript* e a linguagem de programação utilizada na engine, possibilita a extensão do ambiente *JavaScript*. Por meio de módulos de conexão com a internet é possível carregar páginas de servidores *Web* ou do sistema de arquivos local (NOKIA, 2011).

Uma *web browser engine* pode ser estendida para se acrescentar novas funcionalidades a *interface Javascript* disponível. Essas extensões podem ser feitas tanto a nível de núcleo, pelo fato de existirem implemetações *open*

source, exigindo desta forma que todo código seja recompilado, ou atraves de *plugins* fazendo-se uso da API (*Application Programming Interface*) NPAPI (*Netscape Plugin Application Programming Interface*) ou *ActiveX*, descritos a seguir.

2.2.4 *Plugins*

Plugins são bibliotecas que os usuários podem instalar em seus navegadores de internet que permitem acessar conteúdos que o navegador por si só não suporta. Por exemplo, o *plugin* da *Abobe Reader* permite que os usuários abram arquivos PDF (*Portable Document Format*) dentro do seu navegador.

Os *plugins* são escritos usando-se a API NPAPI , que se trata de uma API *cross-browser*, permitindo a criação de plugins para todos os navegadores mais comuns hoje em dia. O Navegador *Internet Explorer* suportou esta API até sua versão 5.5 (MOZILLA, 2011). Após esta versão é necessário utilizar o *ActiveX* para o desenvolvimento de *plugins* para o *Internet Explorer* (CHAPPELL, 1996).

2.2.5 *Web Toolkits*

Atualmente, criar aplicativos para a *web* é um processo tedioso e com alta incidência de erros. Os desenvolvedores podem passar 90% do tempo trabalhando para contornar peculiaridades do navegador. Além disso, a criação, a reutilização e a manutenção de grandes bases de código *JavaScript* e componentes AJAX pode ser difícil e delicada (DEWSBURY, 2007; BARRY, 2008; LEIGHTON, 2010).

Um *Web Toolkit* facilita esse processo, permitindo que os desenvolvedores criem rapidamente e mantenham aplicativos *front end JavaScript* complexos e de alto desempenho em linguagens de programação como *Java* e *Python*, por exemplo, onde o *Web Toolkit* faz a compilação cruzada para o *JavaScript* otimizado que funciona automaticamente com todos os principais navegadores de internet.

Como exemplos de *Web Toolkits* pode-se citar o Gooogle Web Toolkit (GWT) (DEWSBURY, 2007) e o *Pyjamas* desenvolvido por Leighton (2010) em *Python*.

2.3 O PARADIGMA DE AGENTES

2.3.1 Agentes

O termo “agente”, ou agente de *software*, tem sido utilizado em uma série de tecnologias, por exemplo, em inteligência artificial, bancos de dados, sistemas operacionais e redes de computadores. Embora não exista um definição única (RUSSEL; NORVIG, 2003; BRENNER; WITTIG; ZARNEKOW, 1998a; GENESERETH; KETCHPEL, 1994; BRADSHAW, 1997), pra-

ticamente todas as definições apresentam um agente como uma entidade computacional com comportamento autônomo que lhe permite decidir suas próprias ações. Um agente de *software* é conceituado como uma entidade que funciona de forma contínua e autônoma em um ambiente em particular, geralmente habitado por outros agentes, que seja capaz de intervir no mesmo, de forma flexível e inteligente.

Seguindo Bellifemine et al. (2007), um agente é autônomo porque opera sem a intervenção de humanos ou outros e têm controle sobre suas ações e estados internos. Um agente é social porque coopera com humanos e/ou outros agentes na tentativa de realizar suas tarefas. Um agente é reativo porque percebe seu ambiente e responde em um tempo adequado a mudanças ocorridas no mesmo e, um agente é proativo porque não só atua respondendo a mudanças no ambiente, mas também é capaz de exibir um comportamento direcionado a objetivos, tomando a iniciativa da ação.

Além disso, se for necessário, um agente é capaz de se mover em seu ambiente, como a habilidade de viajar entre diferentes nodos em uma rede de computadores, por exemplo. Pode ser benevolente, sempre tentando executar aquilo que é pedido a ele.

Frequentemente são feitas distinções entre agentes que são racionais e os que não são. Um agente é dito ser racional se escolhe executar ações que vão de encontro aos seus interesses, podendo aprender, adaptando-se e ajustando-se ao seu ambiente e/ou aos desejos de seus usuários (WOOLDRIDGE; JENNINGS, 1995; BELLIFEMINE et al., 2007).

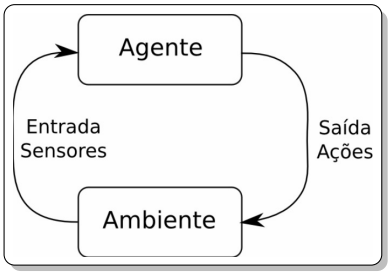

Figura 2.4: Agente e seu Ambiente. Baseada em Wooldridge e Jennings (1995)

A figura 2.4 representa o agente e seu ambiente. O agente recebe dados do ambiente através de seus sensores, e produz como saída ações que afetam o ambiente, através de seus atuadores.

Segundo Wooldridge e Jennings (1995), de acordo com o tipo de problema com o qual está habilitado a tratar, um agente pode possuir cem maior ou menor grau, os seguintes atributos:

- *Reatividade*: a habilidade de perceber o ambiente de modo seletivo e manifestar um comportamento como resposta a um estímulo externo;

- *Autonomia*: comportamento dirigido a objetivos, pró-ativo e auto-iniciado;
- *Comportamento cooperativo*: trabalhar com outros agentes para atingir um objetivo comum;
- *Habilidade de comunicação ao nível de conhecimento*: capacidade de comunicar-se com pessoas ou outros agentes em uma linguagem de mais alto nível que um simples protocolo de comunicação programa a programa;
- *Capacidade de inferência*: capacidade de agir a partir de especificações abstratas de tarefas, usando conhecimentos prévios;
- *Continuidade temporal*: persistência de identidade por longos períodos de tempo;
- *Personalidade*: capacidade de demonstrar atributos de um personagem;
- *Adaptabilidade*: habilidade de aprender com a experiência; e
- *Mobilidade*: habilidade de migrar de uma plataforma para outra.

De acordo com Russell et al. (1995), Nwana (1996), agentes podem ser classificados como segue:

Agentes Colaborativos/Baseados em Objetivos: as características chave deste tipo de agente incluem autonomia, habilidade social, reatividade e proatividade. Assim, podem ser capazes de atuar racionalmente e com autonomia em um sistema multiagente aberto ou com restrições. Eles podem ser benevolentes, racionais, confiáveis ou uma combinação destas características. Da mesma forma que precisam de uma descrição do estado atual de seu ambiente, também precisam de alguma espécie de informação sobre objetivos que descrevam situações desejáveis. Essa informação sobre objetivos, torna este tipo de agente mais flexível, porque o conhecimento que apoia suas decisões é representado de maneira explícita e pode ser modificado.

Agentes de *Interface*: agentes de *interface* enfatizam a autonomia e a aprendizagem para realizar tarefas pelos seus usuários. Agentes de *interface* colaboram com seus usuários no mesmo ambiente de trabalho. A diferença está em colaborar com o usuário, no caso de agentes de *interface*, e colaborar com outros agentes no caso de agentes colaborativos.

Agentes Móveis: agentes móveis são processos computacionais capazes de se deslocar através da rede, uma *Wide Area Networks*(WANs) tal como a *World Wide Web* (WWW) , por exemplo, e interagir com *hosts* desconhecidos, reunir informações e retornar após realizar a tarefa a eles atribuída pelo usuário.

Agentes de Informação/*Internet*: agentes de informação surgiram devido à enorme procura por ferramentas que auxiliem os usuários no gerenciamento de informações encontradas na *Web*, geradas pelo crescimento explosivo sendo experimentado atualmente. Agentes de informação executam o papel de gerenciar, manipular ou comparar criticamente informações oriundas de muitas fontes distribuídas.

Agentes de Software Reativos: agentes reativos representam uma categoria especial de agentes que não possuem modelo simbólico interno representativo de seu ambiente. Eles atuam/respondem da maneira estímulo-reposta ao estado presente do ambiente em que estão inseridos. O ponto mais importante de agentes reativos é que são relativamente simples e podem interagir com outros agentes de maneira básica.

Agentes Híbridos: agentes híbridos possuem algumas das características mais fortes dos paradigmas deliberativos e reativos. Assim, agentes híbridos, são aqueles na qual sua constituição é uma combinação de duas ou mais filosofias em um único agente.

2.3.2 Sistemas Multiagente (SMA)

Embora um agente possa atuar solitariamente dentro de um ambiente, geralmente é necessário interagir com outros agentes e/ou com seus usuários. Sistemas em que existem mais de um agente são chamados de Sistemas Multi-Agente (*Multiagent Systems*)(SMA). Um SMA pode modelar sistemas complexos e introduz a possibilidade de os agentes terem objetivos conflitantes. Cada agente pode interagir com os demais indiretamente, atuando no ambiente, ou diretamente, via comunicação e negociação. Os agentes podem decidir cooperar para o benefício mútuo ou podem competir para servir seus próprios interesses (WOOLDRIDGE; JENNINGS, 1995).

2.3.2.1 Características de Sistemas Multiagente

Segundo Vlassis (2007) um SMA podem ser diferenciado pelas seguintes características:

Design do Agente: frequentemente existem agentes diferentes em várias maneiras. Essas diferenças podem envolver *hardware* e/ou *software*. Tais agentes implementados com base em diferentes *softwares* ou *hardwares* são conhecidos como agentes heterogêneos, em contraste com agentes homogêneos que possuem um design idêntico e a priori as mesmas habilidades. A heterogeneidade pode afetar todos os aspectos funcionais de um agente, da percepção a tomada de decisão.

Ambiente: o ambiente, no qual os agentes estão inseridos, podem ser estáticos ou dinâmicos (mudam com o tempo). Em um SMA, a mera presença de múltiplos agentes torna o ambiente dinâmico do ponto de vista de cada agente.

Percepção: a informação que chega aos sensores dos agentes coletivamente em um SMA é tipicamente distribuída, os agentes podem observar dados que diferem espacialmente (de diferentes lugares), temporalmente (chegam em tempo diferente) ou semanticamente (requerem diferentes interpretações). O fato de os agentes poderem observar coisas diferentes torna o "mundo" parcialmente observável por cada agente, o que tem várias conseqüências na tomada de decisão dos agentes. Por exemplo, planejamento em tempo ótimo pode se tornar intratável sob a observação parcial do ambiente.

Controle: diferentemente de um sistema com um único agente, o controle em um SMA é tipicamente descentralizado. Isso significa que a tomada de decisão reside dentro de cada agente. Prefere-se um controle descentralizado do que um controle centralizado (que envolve um centro) por razões de robustez e tolerância a falhas.

Conhecimento: em sistemas com um único agente, tipicamente assume-se que o agente conhece suas próprias ações, mas não necessariamente como o mundo é afetado por elas. Em um SMA, o nível de conhecimento de cada agente sobre o estado corrente do mundo pode ser substancialmente diferente. Por exemplo, em uma equipe de SMA envolvendo agentes homogêneos, cada agente pode conhecer o conjunto de ações disponíveis de algum outro agente, ambos agentes podem conhecer (via comunicação) suas percepções correntes ou podem inferir as intenções dos demais baseando-se em conhecimento anteriormente compartilhado. Em geral, em um SMA cada agente pode considerar o conhecimento dos outros agentes do sistema na sua tomada de decisão .

Comunicação: interação está frequentemente associada a alguma forma de comunicação. Tipicamente comunicação em SMA é vista como um processo de duas vias, onde todos os agentes potencialmente podem ser enviadores (*senders*) e/ou receptores (*receivers*) de mensagens. Comunicação pode ser usada em muitos casos, por exemplo, para coordenação entre agentes cooperativos.

2.3.3 Plataforma Multiagente (PMA) - Modelo de referência FIPA

Para permitir a interoperabilidade entre agentes/sistemas multiagente desenvolvidos com o uso de diferentes plataformas é necessário que cada plataforma atenda uma série de requisitos. Estes requisitos são estabelecidos, principalmente, através de padrões para a comunicação entre agentes e suas aplicações.

Os trabalhos para a especificação de padrões, iniciaram-se no início dos anos 90 (GLUZ; VICCARI, 2003), através da iniciativa KSE (Knowledge Sharing Effort), fomentada pela agência de pesquisas norte-americana DARPA (INTERCHANGE, 1998). Essa iniciativa tinha como objetivo criar uma série de padrões de comunicação que permitissem que sistemas inteligentes compartilhassem seus conhecimentos de forma padronizada. O principal resultado da KSE foi a linguagem KQML (*Knowledge Query and Manipulation Language*) utilizada para a comunicação entre agentes de forma padronizada através da troca de mensagens (FININ et al., 1994).

Posteriormente, os resultados obtidos pela KSE foram amplificados e estabilizados pelo modelo de referência FIPA (2002), estabelece uma normativa para a criação de *frameworks*, dentro dos quais os agentes podem operar. Ele estabelece um modelo lógico para a criação, registro, locação, comunicação, migração e finalização de agentes.

O modelo de referência para gerenciamento de agentes, apresentado na Figura 2.5, é composto dos seguintes componentes lógicos:

Agente: segundo o conceito apresentado em FIPA (2002), que complementa o que foi apresentado em **2.3.1**, um agente é um processo computacional que implementa de forma autônoma a funcionalidade de comunicação em um aplicativo. Agentes se comunicam através de uma Linguagem de Comunicação de Agentes. O agente é um ator fundamental em uma PMA e combina uma ou mais capacidades de serviço − conforme publicado na descrição de seus serviços − em um modelo de execução unificado e integrado. Um agente deve ter pelo menos um dono, por exemplo, com base na afiliação organizacional ou de propriedade do usuário humano, e deve suportar pelo menos uma noção de identidade. Essa noção de identidade é o Identificador do Agente (*Agent Identifier*) (AID) que rotula um agente para que ele possa ser distinguido claramente dentro do universo de agentes. Um agente pode ser registrado em um número de endereços de transporte, nos quais em que ele pode ser contatado.

***Directory Facilitator* (DF):** é um componente opcional em uma PMA. O DF oferece serviços de páginas amarelas a outros agentes. Agentes podem registrar seus serviços ou consultar serviços oferecidos por outros agentes através do DF. Podem existir vários DFs dentro de uma PMA.

***Agent Management System* (AMS):** é um dos componentes obrigatórios em uma PMA. O AMS exerce supervisão sobre o acesso e utilização da PMA. Apenas um AMS vai existir em uma PMA. O AMS mantém um diretório de AIDs que contêm os endereços de transporte (entre outras coisas) para todos os agentes registrados na PMA . O AMS oferece serviços de páginas brancas para os agentes. Cada agente deve se registrar no AMS a fim de obter uma ajuda válida.

***Message Transport Service* (MTS):** é o método de comunicação padrão entre agentes de diferentes PMAs.

***Software*:** descreve uma coleção de instruções acessíveis a um agente. Os agentes podem acessar *softwares* para adicionar novos serviços, adquirir novos protocolos de comunicação, adquirir novos protocolos/algoritmos de segurança, acessar ferramentas de migração, etc.

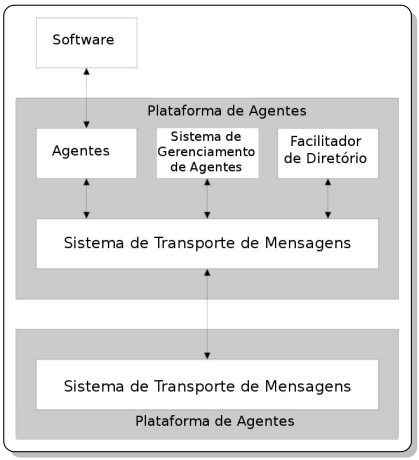

Figura 2.5: Modelo de referência para gerenciamento de agentes
Fonte: FIPA (2002)

2.3.3.1 Ciclo de vida do agente FIPA

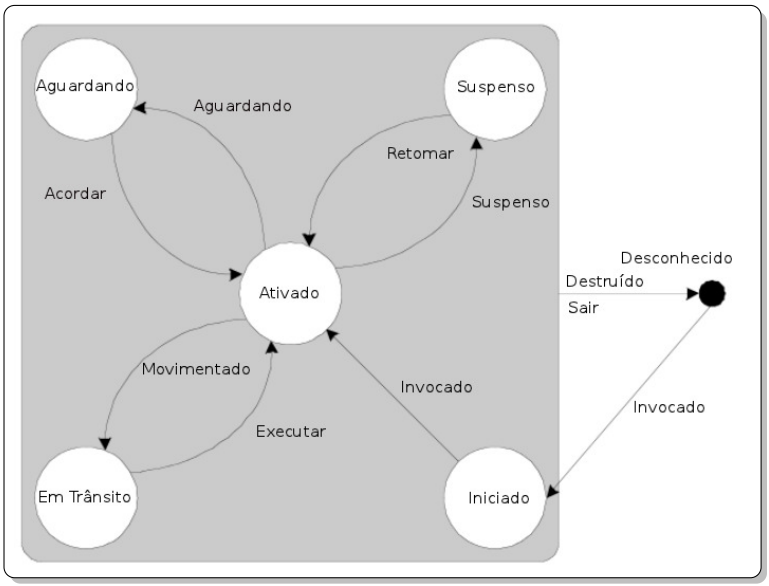

Figura 2.6: Ciclo de vida do agente
Fonte: FIPA (2002)

Segundo FIPA (2002), agentes existem fisicamente em uma PMA e utilizam as facilidades oferecidas pela mesma para realizar suas tarefas. Neste contexto, um agente, como um processo físico de *software*, tem um ciclo de vida físico que é gerido pela PMA. A seguir apresentamos o ciclo de vida que pode ser usado para descrever o estados de um agente e as responsabilidades do AMS nesses estados.

A figura 2.6 apresenta o ciclo de vida para um agente FIPA-compatível, que é definido pelas seguintes características:

PMA delimitado: um agente é fisicamente gerenciado dentro de uma PMA e o ciclo de vida deste agente é estático e, portanto, sempre ligado a uma PMA específica.

Independente de aplicação: o modelo de ciclo de vida é independente de qualquer aplicação e define somente os estados e as transições dos serviços do agente em seu ciclo de vida.

Orientado a instância: o agente descrito no modelo de ciclo de vida é considerado uma instância, ou seja, um agente que tem o nome original e é executado de forma independente.

Único: cada agente está em apenas um estado do ciclo de vida em qualquer momento e em apenas uma PMA.

A seguir são apresentadas as responsabilidades que um agente AMS, em nome da PMA, tem em relação à entrega de mensagens em cada estado do ciclo de vida de um agente:

Ativo (*Active*): o *Message Transport Service* (MTS) entrega as mensagens para o agente normalmente.

Iniciado/Esperando/Suspenso (*Initiated/Waiting/Suspended*): o MTS armazena as mensagens em um *buffer* até que o agente retorne ao estado ativo ou encaminhe mensagens para uma nova localização (se um encaminhamento estiver definido para o agente).

Trânsito (*Transit*): o MTS bufferiza as mensagens até que o agente se torne ativo, (ou seja, se a função de movimento falhar, na PMA original, ou na PMA destino, em caso de sucesso na função de movimento) ou encaminha mensagens para um novo local (se um encaminhamento estiver definido para o agente). Somente os agentes móveis podem entrar no Estado de trânsito. Isso garante que um agente estacionário executa todas as suas instruções no nó onde ele foi chamado.

Desconhecido (*Unknown*): o MTS adiciona as mensagens ao *buffer* ou rejeita-as, dependendo da política do MTS e as necessidades de transporte da mensagem.

Ainda apresentando uma descrição da figura 2.6, as transições de estado do agente podem ser descritas da seguinte forma:

Criar (*Create*): a instalação ou criação de um novo agente.

Invocar (*Invoke*): a invocação de um novo agente

Destruir (*Destroy*): a destruição forçada do agente. Isso só pode ser iniciado pelo AMS e não pode ser ignorado pelo agente.

Terminar (*Quit*): a finalização não forçada de um agente.

Suspender (*Suspend*): colocar o agente no estado de suspenso. Isso pode ser iniciado pelo agente ou pelo AMS.

Retornar (*Resume*): tirar o agente do estado suspenso. Isso só pode ser feito pelo AMS.

Esperar (*Wait*): coloca o agente no estado esperando. Isso só pode ser feito pelo próprio agente.

Acordar (*Wake Up*): tirar o agente do estado esperando. Isso só pode ser feito pelo AMS.

Mover (*Move*)*: colocar o agente em um estado de transição. Isso só pode ser feito pelo agente.

Executar (*Execute*)*: tirar o agente de um estado de transição. Isso só pode ser feito pelo AMS.

Em Bellifemine, Poggi e Rimassa (1999), Poslad, Buckle e Hadingham (2000), Collier et al. (2003), Bordini, Hubner e Wooldridge (2007), podem ser encontrados exemplos de PMAs, que implementam de forma básica as características apresentadas nesta seção.

2.3.3.2 PMA Distribuída

Seguindo a definição apresentada por Coulouris, Dollimore e Kindberg (2005), um sistema distribuído é um sistema no qual os componentes localizados em diversos computadores conectados em rede podem se comunicar e coordenar suas ações apenas por meio de troca de mensagens. O compartilhamento de recursos é a principal motivação para a construção de sistemas distribuídos.

Bellifemine, Poggi e Rimassa (1999) relatam que o crescimento dos recursos e informações em rede requer sistemas que possam ser distribuídos na mesma e inter-operar. Tais sistemas apresentam dificuldades para serem desenvolvidos com tecnologias de software tradicionais devido suas limitações para lidar com a distribuição e interoperabilidade. Entretanto, tecnologias baseadas em agentes, apresentam uma resposta promissora para facilitar a construção de tais sistemas, e sendo assim uma PMA pode ser distribuída em vários computadores em rede, para tal fim.

* - Transição usada apenas por agentes móveis.

2.4 TRABALHOS CORRELATOS

Ainda não existem muitos trabalhos voltados a aproveitar os recursos nativamente disponíveis no lado do cliente para o desenvolvimento de sistemas multiagente *web*. Procurou-se relacionar alguns trabalhos que possuem alguma relação com a idéia aqui proposta. Nestes trabalhos, o sistema multiagente é executado quase que totalmente no servidor, e no lado do cliente apenas são mostrados os resultados aos usuários através do navegador de internet. As propostas que executam alguma lógica no lado do cliente, fazem uso de *applets* Java.

O Framework Jadex (POKAHR; BRAUBACH; LAMERSDORF, 2005) permite que sejam criados agentes racionais, escritos em XML e na linguagem de programação Java. É possível fazer a integração entre estes agentes e sistemas web através de um *add-on* chamado *Webbridge*, que permite a comunicação entre agentes e *Servlets*.

Em Nguyen e Kowalczyk (2007) pode-se encontrar uma descrição para o WS2JADE. Se trata de um kit de ferramentas para a integração de *Web Services* com a plataforma de agentes Jade. Jade também possui uma extensão chamada WSIG (*Jade Web Services Integration Gateway*) (BOARD, 2005), que fornece suporte para invocação de serviços de agentes Jade executando em um servidor web, a partir do lado do cliente.

Aglets (LANGE et al., 1997) é uma plataforma Java com bibliotecas para a construção de aplicações baseadas em agentes móveis. Um *aglet* é um agente de Java que pode de forma autônoma e espontânea se deslocar de um hospedeiro para outro com um pedaço de código com ele. Ele pode ser programado para executar em um host remoto e mostrar comportamentos diferentes em diferentes hosts. Basicamente, um aglet é um applet que pode se mover de um hospedeiro para outro,levando consigo o código a ser executado.

Além dos trabalhos já citados, foram encontrados dois trabalhos na literatura, que são explicitamente voltados à execução de agentes em aplicativos web no lado do cliente. Um que utiliza o *Google Native Client* (NaCl) para disponibilizar uma *interface JavaScript* aos desenvolvedores, (HAENSCH M. O., 2011) e outra chamada *Jaca-Web* (MINOTTI; PIANCASTELLI; RICCI, 2010), que faz uso do estilo de programação orientada a eventos de *JavaScript* para desenvolver aplicativos concorrentes no lado cliente. Ambos trabalhos. estão descritos nas seções a seguir, procurado-se destacar suas qualidades e suas deficiências.

2.4.1 3APL-M + NaCl

A linguagem 3APL-M é uma variação para dispositivos móveis da linguagem 3APL (*Artificial Autonomous Agent Programming Language*) e se preocupa em ter um tamanho reduzido para possibilitar seu uso nesses dispositivos, em geral muito mais limitados em termos de memória e capacidade de processamento que computadores pessoais.

Tanto a linguagem original quanto a sua variação *Mobile* possuem uma

estrutura de agente baseada no conceito de raciocínio prático de Michael Bratman de crenças, desejos e intenções, ou BDI (*Belief, Desire, Intention*). O modelo de agente BDI utilizado pela linguagem 3APL-M e sua respectiva implementação são bastante simples, ajudando a diminuir a carga de processamento envolvida na execução dos agentes, melhorando os resultados na execução via *browser*.

O *Google Native Client* é uma plataforma de código-aberto de *sandboxing* para execução segura de código nativo não confiável através de um *browser*. Essa tecnologia permite que aplicações *Web* com uma grande demanda de processamento ou de interatividade, como jogos, aplicações multimídia e análise e visualização de dados, sejam executadas na máquina do cliente com acesso restrito, formando um ambiente seguro. Mais detalhes sobre o funcionamento da tecnologia pode ser encontrados na página do projeto, assim como o kit desenvolvimento disponibilizado.

Um dos objetivos da plataforma era permitir a adição de funções mais complexas e de acordo com as necessidades do usuário da plataforma, através de código na linguagem *JavaScript*. Porém, a tecnologia *Google NaCl* na versão atual (0.3) não permite que sejam feitas chamadas de funções durante a execução da plataforma, impossibilitando essa funcionalidade no momento.

Como boas características deste *framework* pode-se citar sua execução nativa no lado do cliente, o que pode aumentar o desempenho, e a facilidade de uso dos aplicativos por parte dos usuários, uma vez que não é necessário instação e/ou configuração. Como limitação pode-se citar que o NaCl somente está disponível para o navegador *Google Chrome*, e desta forma, somente usuários que utilizem este navegador podem acessar os aplicativos desenvolvidos com este *framework*.

2.4.2 *Jaca-Web*

Jaca-Web usa o estilo de programação orientada a eventos de JavaScript para desenvolver aplicativos concorrentes e iterativos no lado do cliente. Segundo Minotti, Piancastelli e Ricci (2010) a *Web* 2.0 está mostrando deficiências em termos de propriedades de engenharia, tais como reusabilidade e manutenibilidade. Bibliotecas adicionais, estruturas e técnicas de AJAX não ajudam a reduzir a lacuna entre o modelo *JavaScript single-threaded* e as necessidades de simultaneidade de aplicações.

Jaca-Web foi uma primeira tentativa de explorar um modelo de programação diferente, com base em uma camada de abstração no modelo orientado a agentes, onde entidades de primeira classe ou seja, agentes e artefatos podem ser usados, respectivamente, paracaptar a simultaneidade de atividades e suas interações, e para representar ferramentas e recursos utilizados pelos agentes durante suas atividades.

Jaca-Web é uma implementação parcial de um modelo mais geral para aplicações *web* no lado do cliente, descrito em Minotti, Piancastelli e Ricci (2010) focado em superar a maioria dos problemas de compatibilidade entre os navegadores *web* e as tecnologias orientadas a agentes disponíveis. *Jaca-Web*

usa *Jason* (*agentspeak*) como linguagem de programação – para programação de agentes inteligentes – e *Cartago* – para a programação do ambiente onde os agentes são inseridos.

Como boa característica desta solução, pode-se citar a abstração disponível para acomodar imcompatibilidades dos navegadores de internet. Embora *Jaca-Web* possa ser utilizado na mairia dos navegadores de internet modernos, ele executa dentro de *applets Java*, o que exige a instalação e configuração do *plugin Java* no navegador do cliente, o que geralmente é fonte de muitos problemas para os usuários comuns.

3 PROCEDIMENTOS METODOLÓGICOS

Este trabalho de pesquisa se enquadra na categoria de pesquisa tecnológica, pois visa alcançar a inovação em um produto ou processo, frente a uma demanda ou necessidade preestabelecida.

Seguindo a classificação por estilos apresentada por Wazlawick (2009), este trabalho se enquadra no estilo de apresentação de uma forma diferente para resolver um problema, sendo que este tipo de pesquisa é característico de áreas emergentes, onde os resultados são apresentados como uma simples comparação de técnicas.

Para o desenvolvimento deste trabalho, decidiu-se dividir em quatro etapas principais, que estão listadas a seguir:

1. Avaliação dos recursos e restrições existentes no lado do cliente, propondo um modelo de SMA com base nos mesmos;

2. Avaliação de *frameworks* para desenvolvimento de SMA existentes, levando em consideração as restrições levantadas no item 1 bem como as características descritas na literatura como essenciais para a POA.

3. Proposição do *framework* a que se refere este trabalho, partindo-se do *framework* selecionado na etapa 2 e do modelo proposto na etapa 1.

4. Implementação e validação, a partir de um estudo de caso.

A imagem da figura 3.1 apresenta de forma gráfica as etapas listadas acima. Estas etapas são descritas nas seções a seguir.

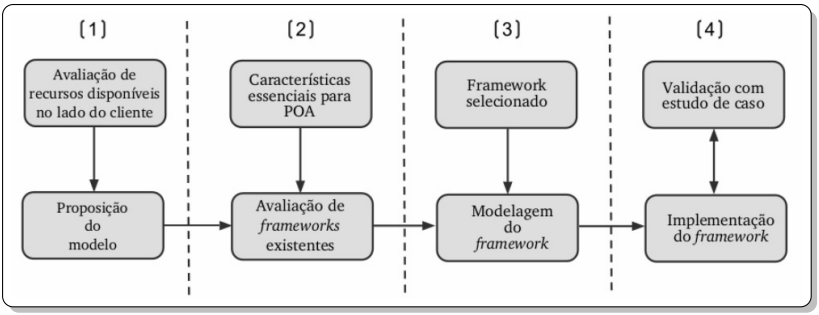

Figura 3.1: Metodologia utilizada

Durante a etapa de avaliação dos recursos e restrições existentes **(1)**, foi feito um levantamento sobre os requisitos para a execução de aplicativos *Web* baseados no paradigma de agentes, no lado do cliente. Para tal, as seguintes questões foram levantadas:

1. Como e onde executar estes aplicativos?

2. Quais são os recursos de *software* e *hardware* existentes, em geral?

3. Quais são as principais restrições?

Com base na resposta para as perguntas acima, um modelo para o desenvolvimento de sistemas multiagente *web*, que execute no lado do cliente foi proposto.

Com respostas para as questões acima e com o modelo proposto, partiu-se para a etapa seguinte, **(2)** na Figura 3.1. Nesta etapa, foram levantadas as principais propriedades relativas a agentes, SMAs e PMAs. Como "principais propriedades", serão consideradas aquelas que se enquadrem nos requisitos levantados durante a etapa **(1)** e também as que são descritas na literatura como mais importantes. Cada propriedade foi definida a partir do estado da arte encontrado na literatura.

Tendo-se as propriedades relacionadas, iniciou-se então a etapa de avaliação de *frameworks*, onde foram considerados os que possuem pelo menos algumas das propriedades definidas em **(2)**, relacionadas aos requisitos levantados em **(1)**. Trata-se de uma avaliação empírica, realizada pelo autor deste trabalho, onde, dada uma propriedade e um *framework* sendo avaliado, tal propriedade foi dada como presente, parcialmente presente ou ausente no *framework* em questão. Obviamente, a lista de ferramentas e *frameworks* para a POA é bastante extensa atualmente, o que torna praticamente impossível a avaliação de todas.

Ao final da segunda etapa selecionou-se um *framework*. Este *framework* serviu como base para o desenvolvimento do *framework* desenvolvido, atendendo ao modelo proposto em **(1)**. A idéia por traz da etapa **(3)**, foi modelar um *framework* baseando-se principalmente no *framework* selecionado em **(2)**, mas não descartando a possibilidade de se usar as "partes bem sucedidas"dos demais, de acordo com a avaliação em **(2)** (entenda-se "partes bem sucedidas"como a propriedade presente no *framework*), que atenda aos requisitos levantados em **(1)** e **(2)**.

Como resultado, na etapa **(3)** tem-se a proposição de um *framework* voltado para o desenvolvimento de aplicativos *Web*, com ênfase na utilização de recursos disponíveis no lado do cliente, utilizando-se a POA para tal.

Na etapa final, um protótipo inicial do *framework* foi desenvolvido e avaliado através da proposição e implementação de um estudo de caso.

4 WEBMAS *TOOLKIT*

Conforme apresentado na introdução, este trabalho apresenta um modelo para o desenvolvimento de sistemas multiagente plenamente distribuídos de forma que aplicações possam ser executadas através de agentes que atuam utilizando-se de recursos nativamente disponíveis no lado dos clientes e em servidores distribuídos em diversas máquinas, procurando equilibrar a distribuição de processamento e permitindo desta forma, a crição de aplicações mais robustas.

Para chegar a solução proposta foram realizadas algumas etapas, conforme descrito no capítulo sobre procedimentos metodológicos. Estas estapas compreendem um levatamento de características desejáveis ao *framework* sendo proposto, avaliação de recursos e restrições existentes no lado do cliente, proposição de um modelo para o desenvolvimento de sistemas multiagente plenamente distribuídos, análise e seleção de framework base e proposição do framework WebMAS Toolkit. Cada etapa está descrita na sequência deste capítulo.

Primeiramente, estabeleceu-se algumas características desejáveis a um modelo para o desenvolvimento de SMAs *web*. Essas características serviram de base para a avaliação dos recursos e restrições existentes no lado do cliente, sendo elas:

- **Portabilidade:** a plataforma de execução do SMA precisa ser compatível com o maior número possível de tecnologias, proporcionado ao usuário final a mesma experiência durante a utilização de aplicativos, que venham a ser desenvolvidos usando-se o modelo proposto. Considerando-se o ponto de vista do desenvolvedor, é importante que a quantidade de código necessária para atingir a portabilidade seja a mínima possível, não tendo que "reinventar a roda" para que um aplicativo possa executar em uma nova tecnologia alvo.

- ***Standalone* (sem necessidade de extensões):** não obrigar o usuário final a realizar instalações e/ou configurações em seu dispositivo para poder acesser um aplicativo desenvolvido a partir do modelo. A plataforma, que implementa o modelo, deve ser executada usando-se apenas recursos disponíveis no lado do cliente.

- ***Open Source*:** a tecnologia empregada deve ser de código aberto (*open source*), possibilitando que assim que o *framework* que implemente o modelo, seja aprimorado e extendido pela comunidade acadêmica em geral, seguindo-se as regras da licença de *software open source* empregada.

- **Curva de Aprendizagem**: o desenvolvimento de SMAs para a *Web* não deve exigir que o desenvolvedor tenha que aprender a utilizar muitas ferramentas e/ou muitas linguagens de programação ao mesmo tempo,

ou seja, seguir o modelo deve ser relativamente fácil, com uma curva de aprendizagem curta.

4.1 AVALIAÇÃO DOS RECURSOS E RESTRIÇÕES EXISTENTES NO LADO DO CLIENTE

Visando responder às perguntas *Como e onde executar estes aplicativos?*, *Quais são os recursos de software e hardware existentes, em geral?* e *Quais são as principais restrições?* apresentadas no capítulo 3 e usando as características listadas acima como base, buscou-se fazer um estudo sobre onde, como e sob que condições um sistema mutiagente *web* pode ser executado no lado do cliente.

Para apresentar tal estudo, as subseções a seguir relatam o *onde*, *como* e *sob que condições* para cada possível tecnologia levantada pelo autor.

4.1.1 Recursos Disponíveis

Inicialmente, constatou-se que só existe uma possível resposta para o *onde* executar um sistema multiagente *web* no lado do clinte: em uma *web browser engine* (motor de execução dos navegadores de internet), sendo que existem duas possíveis variações para a utilização de uma *web browser engine*: em aplicações híbridas (*Web/Desktop*), onde a própria engine é incorporada ao aplicativo; na *web browser engine* do navegador de internet do usuário, aqui chamada de aplicação *web standalone*.

Aplicações híbridas *Web/Desktop*: como descrito em **2.2.3**, uma *Web browser engine* possibilita que sejam criados aplicativos híbridos, isto é, aplicativos que utilizam recursos *Desktop*, que também podem acessar conteúdos diretamente da *Web*. Como exemplos de *engines* podemos citar: *Webkit* (WEBKIT, 2011), *QTWebKit* (NOKIA, 2011), *Gecko* (NETWORK, 2011), etc (em Wikipedia (2011), pode ser encontrada uma lista comparativa sobre *web browser engines*).

Algumas *web browser engines* possuem bibliotecas com funcionalidades implementadas para facilitar sua utilização. *QTWebKit*, por exemplo, disponibiliza uma vasta biblioteca, incluindo recursos (NPAPI) que possibilitam extender as funcinalidades presentes em sua *interface JavaScript*.

Mesmo que com o uso de *web browser engine* tenhamos: integração e utilização de bibliotecas destinadas a aplicativos *desktop*, execução nativa e disponibilidade de implementações *open source*, existem alguns problemas relacionados a portabilidade e necessidade de instalação/configuração no hospedeiro. Outro problema a ser destacado é o fato da necessidade de se incorporar a *engine* ao aplicativo, o que aumenta muito o tamanho final do mesmo. Vale também lembrar que utilizar uma *web browser engine* pode exigir do desenvolvedor conhecimentos em mais de uma linguagem de programação, tendo que criar o aplicativo com partes em uma linguagem e partes em outra e fazer a integração destas partes, o que pode levá-lo a cometer erros. A curva de apren-

dizagem para a utilização da tecnologia também é maior, pelo emprego de mais de uma linguagem.

Aplicações *web standalone*: em uma aplicação *web standalone* a idéia é que os aplicativos executem diretamente no navegador de internet do hospedeiro, não havendo necessidade de instalação e/ou configuração do aplicativo. Neste caso, os aplicativos seriam desenvolvidos a partir de recursos nativamente disponíveis nos navegadores (HTML + *JavaScript*). Cada vez que o usuário acessar o aplicativo via *web*, estará acessando sempre a versão mais atualizada do mesmo, o que pode ser considerada uma grande vantagem desta solução. Outra vantagem importante relaciona-se a portabilidade. Hoje em dia existem modernas e eficientes bibliotecas *JavaScript*, compatíveis com a maioria das versões dos navegadores de internet mais modernos disponíveis.

Por outro lado, existem problemas ao se considerar a curva de aprendizagem da linguagem *JavaScript* e algumas de suas limitações como: linguagem *single-threaded*, execução interpretada (o que pode degradar o desempenho), restrições de acesso a arquivos locais (por medida de segurança), etc, que no fim das contas exigem mais conhecimentos por parte do desenvolver, embora, como será apresentado mais adiante, existem soluções práticas para o desenvolvimento, que praticamente anulam essas limitações e problemas.

Aplicações *Web standalone + plugins*: muitos aplicativos *web* não são suportados nativamente nos navegadores de internet. Para poder executá-los é necessário estender o navegador através da instalação de *plugins*. *Java*, *Flash* e *QuickTime* são exemplos de *plugin*s, criados a partir da NPAPI, descrita em **2.2.4**, utilizados para acrescentar novas funcionalidades aos navegadores de internet.

Como vantagens desta tecnologia podemos citar: existência de implementações *open source*, integração com bibliotecas *desktop* relativamente fácil através do *plugin* e execução parcialmente nativa, sendo que parte do aplicativo é executava dentro do *plugin* e parte dentro do navegador. A parte executada dentro do *plugin*, pode ter acesso ao *hardware* do cliente e executar instruções nativas, como é o caso do *plugin NaCl* da *Google* (YEE et al., 2009).

Para esta tecnologia os problemas são praticamente os mesmos em se usar uma uma *web browser engine*, ou seja, existem problemas de portabilidade (no nível do *plugin*), necessidade de instalação e configuração no computador do cliente, necessidade de que o desenvolvedor tenha conhecimentos da linguagem em que o *plugin* foi desenvolvido e possivelmente de *JavaScript*, já que parte do aplicativo irá executar no navegador e parte no *plugin*, o que também pode ser uma grande fonte de erros de programação, pela dificuldade de depuração do código. Outro grande problema, tanto para essa tecnologia, quanto se usar uma *web browser engine*, é a necessidade do usuário reinstalar o aplicativo a cada nova versão ou atualização, o que nem sempre é fácil.

A Tabela 4.1 resume os prós e contras em relação as tecnologias avaliadas.

Tabela 4.1: Prós e Contras das tecnologias avaliadas

Tecnologia	Prós	Contras
web browser engine	implementações *open source* facilidade de integração execução nativa	portabilidade prejudicada exige instalação tamanho de aplicativo curva de aprendizagem atualização manual
web standalone	implementações *open source* atualização automática portabilidade sem instalação	curva de aprendizagem execução interpretada tamanho de aplicativo
web standalone + plugins	implementações *open source* integracão relativamente fácil execução parcialmente nativa	curva de aprendizagem portabilidade prejudicada exige instalação (plugin) atualização manual tamanho de aplicativo

4.1.2 A solução escolhida

Observando-se a tabela 4.1, chegou-se a conclusão de que utilizar os recursos de uma aplicação *web standalone*, onde existem implementações e recursos *open source*, atualizações automatizadas, portabilidade e não se exige instalação e/ou extensão de recursos para sua utilização, é o melhor caminho a se seguir. Assim, os sistemas multiagente *web* a serem desenvolvidos, com base no modelo proposto neste capítulo, resultarão em aplicações *web* que executam e utilizam recursos do navegador de internet, no lado do cliente.

Com a resposta para a pergunta *Como e onde executar estes aplicativos?*, podemos agora fazer uma análise mais detalhada sobre quais são os recursos de *software* em geral e quais são as principais restrições para a soulução escolhida.

Recursos disponíveis em aplicações *web standalone*: como recursos disponíveis em aplicações *web standalone*, conforme descrito na seção 2.2, pode-se listar:

- Programação através de scripts (*JavaScript*);
- Renderização de HTML, XHTML, SVG, etc;
- Definição de *layout* através de CSS;
- Recursos do HTML5 (*webworkers*, *websockets*, bancos de dados local, etc);
- Ajax;
- Bibliotecas *JavaScript* compatíveis com a maioria dos navegadores de internet modernos; e

- *Web Toolkits*.

Restrições existentes em aplicações *web standalone*: algumas das restrições existentes em aplicações *web standalone* estão colocadas na tabela 4.1. Entre elas estão, tamanho da aplicação reduzido e execução interpretada. Além destas, outras podem ser citadas, seguindo-se o que foi apresentado na seção 2.2:

- Muitos formatos de arquivos e especificações para estruturar a aplicação;
- Limitações do *JavaScript* como: linguagem *single-threaded*, restrições de acesso a arquivos locais, etc;
- Peculiaridades de cada navegador de internet; e
- Dificuldades de criação, reutilização e manutenção de grandes bases de código.

Com respostas para as perguntas feitas no capítulo 3, pôde-se propor o modelo para o desenvolvimento de sistemas multiagente *web*, utilizando-se os recursos levantados, e seguindo-se as restrições acima citadas. As seções a seguir apresentam o modelo e sua respectiva descrição.

4.2 MODELO PARA DESENVOLVIMENTO E UTILIZAÇÃO DE SISTEMAS MULTIAGENTE *WEB* NO LADO DO CLIENTE

Para o desenvolvimento de SMAs *web*, usando-se apenas recursos nativamente disponíveis no lado do cliente (no navegador de internet) e observando-se as restrições encontradas, propôs-se o modelo apresentado na figura 4.1. Trata-se de um modelo bastante abstrato, que embora simples, apresenta a idéia global da proposta.

No modelo existe uma intencional distinção entre o lado do cliente e o lado do servidor, isto porque, de acordo com a proposta deste trabalho, "Um Modelo para Desenvolvimento de Sistemas Multiagente Plenamente Distribuídos", a intenção é de se executar os aplicativos *web*, seguindo-se o paradigma de agentes, no lado do cliente, utilizando-se de recursos locais e proporcionando-se maior autonomia e proatividade aos aplicativos, permitindo-se que os mesmos evoluam e se adaptem as necessidades de seus usuários, a medida que estes os utilizem.

Ao se observar a porção mais a esquerda da imagem na figura 4.1, identificada como *client-side*, verifica-se, conforme proposto, que os aplicativos executam dentro dos navegadores de internet, não utilizando-se de outros recursos, além daqueles providos pelos mesmos (*JavaScript* + HTML).

As linhas tracejadas bidirecionais, enumeradas de (1) a (4), indicam o fluxo que as informações podem seguir. Note-se que o modelo não restringe comunicação entre dois aplicativos que estejam executando no mesmo navegador, em navegadores diferentes no mesmo hospedeiro ou navegadores em

hospedeiros diferentes, ou seja, deve ser possível, em aplicações *web* seguindo o modelo, realizar troca de informações entre duas aplicacões executando em clientes diferentes, não passando necessariamente pelo servidor.

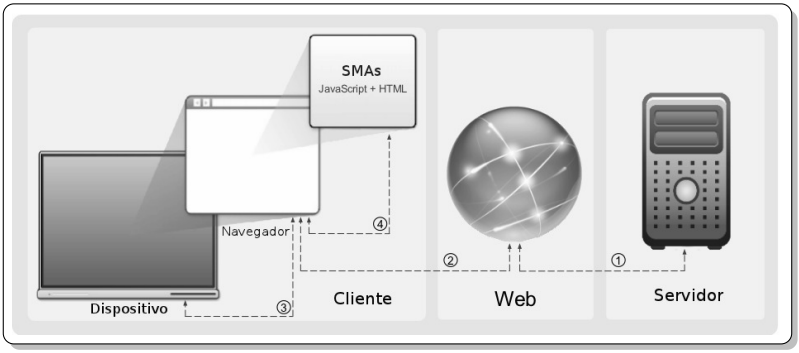

Figura 4.1: Modelo para desenvolvimento e utilização de sistemas multiagente plenamente distribuídos

Outro ponto importante a se destacar é que o dipositivo (*device*) representado no modelo, pode ser qualquer dispositivo que comporte a execução de um navegador de internet, ou seja, desde *desktops*, *tablets*, *smartphones*, etc. Isso atende a característica desejável sobre portabilidade, definida no início deste capítulo.

Neste modelo, o servidor possui o papel de hospedar os fontes dos aplicativos e fornecer serviços globais como: autenticação, tabela de endereços, suporte a migração, banco de dados, etc, atuando de certa forma como coordenador das subplataformas que executam no lado do cliente.

Com base neste modelo, na seção a seguir será elaborado um estudo sobre *frameworks* para o desenvolvimento SMAs. O objetivo é selecionar um destes *frameworks* que servirá de base para a proposição de um *framework* que possibilite o desenvolvimento de sistemas multiagente plenamente distribuídos, de acordo com o modelo proposto.

4.3 AVALIAÇÃO DE *FRAMEWORKS* PARA DESENVOLVIMENTO DE SMA

Uma aplicação *web* que execute no lado do cliente está sujeita a uma série de restrições. Além das citadas em **4.1.2**, pode-se destacar: capacidade de processamento reduzido, restrições de segurança, tempo de resposta limitado, etc. Desta forma, um *framework* que se destine ao desenvolvimento de SMAs voltados a execução no lado do cliente, deve estar atento a estas restrições.

Pelo fato de já existirem muitas propostas de *frameworks* para o desenvolvimento de SMAs, mas nenhum voltado especificamente para a *web*,

com capacidade de executar *standalone* no lado do cliente, procurou-se avaliar aqueles *frameworks*, cujas características se aproximam ao modelo proposto, levando em consideração as características desejáveis, recursos e restrições existentes e, conforme será descrito a seguir, aqueles que possuem um mínimo de propriedades relevantes ao desenvolvimento de SMAs, de acordo com o que foi encontrado na literatura.

Ambientes restritos: do ponto de vista de uma PMA (Plataforma MultiAgente), um ambiente restrito pode ser considerado: um ambiente onde os recursos de *hardware* são limitados. Espaço, velocidade de memória e capacidade de processamento servem como exemplo; um ambiente onde o tempo de resposta do software é crítico (*real-time*), por exemplo, em um sistema de monitoramento de segurança, onde ocorre um evento relativo a uma situação de perigo, uma resposta a este evento deve ser dada em um tempo adequado; um ambiente onde a largura de banda é reduzida, rede celular, por exemplo; um ambiente onde os recursos de *softwares* são limitados, um aplicativo embutido em um navegador de internet, por exemplo; para informações adicionais pode-se consultar Kfir-Dahav, Monderer e Tennenholtz (2000), DiPippo et al. (2001), Soler et al. (2002), Gasmelseid (2006).

o mapa conceitual representado na figura 4.2 apresenta as propriedades consideradas na avaliação de PMAs realizado neste trabalho. Procurou-se avaliar plataformas com potencial para serem portadas a execução em navegadores de internet, ou seja, plataformas voltadas a execução em ambientes restritos. Para que as referências a estas propriedades sejam simplificadas, utilizou-se a seguinte notação: `Nome_da_propriedade(X)`, onde `X` passa a representar a propriedade na sequência deste trabalho, podendo ser utilizada em lugar da propriedade por extenso.

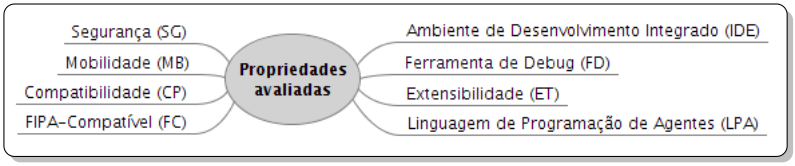

Figura 4.2: Mapa conceitual de propriedades avaliadas

Caso ocorra alguma dúvida quanto a notação, pode-se encontrar uma referência a ela na LISTA DE SIGLAS na forma `X Nome_da_propriedade`.

4.3.1 Propriedades consideradas

SG → Segurança:

Em uma aplicação desenvolvida seguindo-se o paradigma de programação orientada a agentes, assim como em qualquer outro paradigma, a segurança e privacidade dos dados é fator crucial a ser observado. Uma plataforma multiagente deve garantir confiabilidade, confidenciabilidade, privacidade, integridade e autenticação para atender minimamente as necessidades de seu usuário, (BORSELIUS, 2002; MOURATIDIS; GIORGINI; MANSON, 2003; RAMCHURN; HUYNH; JENNINGS, 2005; LU; HUANG, 2006; XIAO et al., 2007), e por este motivo, a presença desta propriedade está sendo avaliada.

MB → Mobilidade:

Uma característica inerente ao conceito de agentes é a mobilidade. Um agente deve poder se movimentar dentro de seu ambiente, seja ele físico e/ou virtual (BORSELIUS, 2002; MALIK et al., 2005; ABECK; KOPPEL; SEITZ, 1998). A mobilidade torna aparente vantagens significantes de SMAs, como por exemplo, a redução de carga na rede, porque um agente pode ir de encontro ao computador e/ou agentes com as informações necessárias para a execução de sua tarefa e executá-la toda localmente no computador alvo. Do ponto de vista técnico o processo de migração constitui uma das tarefas mais complexas no uso de agentes móveis. Uma PMA deve fornecer meios para o transporte de agentes entre dois computadores e também determinar quais componentes devem ser transferidos durante a migração de um agente(BRENNER; WITTIG; ZARNEKOW, 1998b). Dada a importância da mobilidade em SMAs e a necessidade de suporte para tal nas PMAs, é crucial a presença de tal propriedade em uma proposta de plataforma para SMA.

CP → Compatibilidade:

No desenvolvimento de novos aplicativos é importante manter a compatibilidade com o software já existente. No desenvolvimento de aplicações orientadas a agentes isso não é diferente e uma proposta para tal deve contemplar esse critério e apresentar meios que tornem possível a integração e uso por parte dos desenvolvedores de *software* legado (SYCARA, 1998; JENNINGS, 2001; NWANA, 2009; ZHAO et al., 2008).

FC → FIPA-Compatível:	Para proporcionar interoperabilidade entre diferentes plataformas, deve-se seguir alguma especificação pré-estabelecida, que seja aceita pela maioria. Na comunidade de SMAs essa especificação foi criada pela FIPA (*Fondation for Intelligent Physical Agents*) (SPECIFICATION, 2000). Como exemplo, a especificação "FIPA *Abstract Architecture Specification*" (GENEVA, 2002) trata da definição dos elementos arquiteturais presentes em um SMA e das relações entre eles. Sendo assim, a propriedade representada neste item, procura avaliar se a plataforma está de acordo com as especificações FIPA, permitindo a inter-operação de agentes heterogêneos e de serviços representados pelos mesmos.
IDE → Ambiente de desenvolvimento Integrado (*Integrated Development Environment*) :	De acordo com Bordini et al. (2009), uma IDE combina diferentes ferramentas de desenvolvimento de *software* sob uma *interface* unificada. O principal foco da IDE é auxiliar o desenvolvedor nos processos de modelagem, desenvolvimento, teste e implantação do sistema (BELLIFEMINE; POGGI; RIMASSA, 2001; BORDINI; HUBNER; WOOLDRIDGE, 2007; BRAUBACH; POKAHR; LAMERSDORF, 2005). Sendo assim, pode-se considerar que uma IDE, do ponto de vista de uma PMA, integra linguagem de programação de agentes, metodologia de desenvolvimento, ferramenta de depuração, etc, sob uma *interface* única, facilitando e aumentando a produtividade do desenvolvedor do SMA, e deste modo, consideramos um importante característica a ser avaliada.
FD → Ferramenta de Depuração (*Debugger*):	De acordo com Bordini et al. (2009) uma ferramenta de depuração permite controlar e monitorar um programa em execução com o objetivo de encontrar erros de programação. Dada a importância de se criar aplicativos que estejam livres de erros, a presença de uma ferramenta de depuração tem grande relevância em uma PMA e devido a isto, considerou-se este critério (LIEDEKERKE; AVOURIS, 1995; NDUMU et al., 1999; POUTAKIDIS; PADGHAM; WINIKOFF, 2002).

ET → Extensibilidade:	Em geral, extensibilidade pode ser definida como uma facilidade com a qual um software pode ser modificado para se adaptar a novos requisitos e/ou mudanças nos requisitos existentes. Um SMA aberto, isto é, que possui a propriedade de aceitar a entrada de novos agentes com o passar do tempo (*openness*) (SYCARA, 1998; VERCOUTER, 2002; ZAMBONELLI; PARUNAK, 2004; TAO; HUANG, 2009; SHAH et al., 2005), pode ser considerado como um sistema extensível dinamicamente. Decomposição, modularização e abstração são métodos que, do ponto de vista do desenvolvedor, possibilitam a criação de softwares extensíveis e reusáveis. Para que desenvolvedores criem SMAs abertos é necessário que as PMAs forneçam os recursos necessários. Sendo assim, a extensibilidade se torna um importante critério a se avaliar.
LPA → Linguagem para Programação do Agente:	Em Shoham (1993) a Programação Orientada a Agentes (*Agent-Oriented Programming*) (POA) é apresentada como uma especialização da Programação Orientada a Objetos (*Object-Oriented Programming*) (POO). O estado de um agente consiste em componentes como crenças, desejos, intenções, decisões, capacidades, obrigações, etc. Por esta razão o estado de um agente é chamado de *estado mental* (WOOLDRIDGE, 2000). Os estados mentais dos agentes são representados por operadores de crença e a POA introduz operadores para obrigações, capacidades e decisões, por exemplo. Agentes são controlados por seus respectivos programas e a linguagem utilizada para a criação destes programas precisa dar suporte a descrição dos estados mentais do agente. Desta forma, uma PMA precisa suportar uma linguagem de programação de agentes e, por este motivo, avaliou-se a presença desta propriedade (BORDINI; DASTANI, 2005; BORDINI et al., 2009).

Dentre os *frameworks* avaliados, procurou-se priorizar aqueles que possuíam uma versão para a execução em ambientes restritos, conforme descrito na seção 4.3.

Neste trabalho, foram avaliados os seguintes *frameworks*:

- **JADE** (Bellifemine, Poggi e Rimassa (1999), Caire e Pieri (2006)) : JADE é um framework para o desenvolvimento de SMAs implementado em linguagem *Java*. Está em conformidade com as especificações FIPA e através de um conjunto de ferramentas gráficas que suporta a implementação, depuração e fases de implantação. A plataforma de agentes pode ser distribuída através de várias máquinas, mas a mobilidade é possível somente dentro da mesma plataforma. JADE não possui uma linguagem de programação de agentes, uma interface para desenvolve-

dores e uma metodologia de desenvolvimento explicitamente definidas. JADE possui uma versão leve chamada JADE-LEAP que pode executar em dispositivos que vão desde celulares até *workstations*.

- **FIPA-OS** (Poslad, Buckle e Hadingham (2000), Tarkoma e Laukkanen (2002)) : FIPA-OS é um conjunto de ferramentas de desenvolvimento de agentes e uma plataforma de execução compatível com o FIPA. FIPA-OS oferece a comunicação de alto nível básica e funcionalidades para o gerenciamento da comunicação, a capacidade de criar tarefas e subtarefas, os serviços de plataforma básica especificada pela FIPA (AMS, DF), e sistemas de transporte de mensagens, tais como RMI e CORBA. Possui uma versão leve chamada de MicroFIPA-OS que executa em dispositivos móveis. Este *framework* não possui ferramenta de suporte ao desenvolvedor, suporte a mobilidade e não possui suporte a nenhuma metodologia de desenvolvimento de agentes.

- **3APL** (Hindriks et al. (1999), Koch et al. (2006)) : 3APL não se trata exatamente de um *framework* e sim uma linguagem de programação para a implementação de agentes cognitivos. Fornece construções de programação para implementar crenças, metas, capacidades básicas (tais como atualizações de crenças, ações externas, ou ações de comunicação) e um conjunto de regras de raciocínio prático através do qual agentes agentes metas podem ser atualizados ou revistos. Os programas são executados 3APL na plataforma 3APL. Cada programa 3APL é executado por meio de um interpretador que delibera sobre as atitudes cognitivas de cada agente. Sua versão leve é chamada 3APL-M executa em dispositivos móveis. 3APL não é compatível com FIPA, não possui suporte a migração e não dá suporte ao desenvolvimento de aplicativos seguros.

- ***Agent Factory* (AF)** (Collier (2002), Muldoon et al. (2006)) : *Agent Factory* é uma coleção de ferramentas de código aberto que suportam o desenvolvimento e implantação de sistemas multiagentes. Possui suporte para a implantação de sistemas multiagente em celulares, *laptops*, *desktops* e servidores. É um *framework* compatível com os padrões FIPA e possui suporte a linguagens de desenvolvimento de agents como AFAPL (*Agent Factory agent programming language*) e *AgentSpeak(L)*. Sua interface para desenvolvedores é modular o que facilita sua extensão. Sua versão leve (AFME) possui suporte a migração fraca, exigindo que o destino, para o qual agente deseja de mover, já possua o seu código fonte. Em termos de segurança, não foi encontrado uma documentação clara na literatura.

- **Jason** (Bordini, Hubner e Wooldridge (2007)) : Uma das melhores abordagens conhecidas para o desenvolvimento de agentes cognitivos é a arquitetura BDI (*Beliefs-Desires-Intentions*)(BRATMAN, 1987; WOOLDRIDGE, 2000). *Jason* interpreta uma versão melhorada do *AgentSpeak*, incluindo também comunicação entre agentes através de atos de

fala (*speak acts*). Existem várias implementações de sistemas BDI *ad-hoc*, mas uma característica importante da linguagem *AgentSpeak* é a sua base teórica. Outra característica forte de *Jason* em comparação com outros sistemas BDI é que ele é implementado em *Java*, e portanto é multiplataforma, sendo se código *open-source* sob a licença GNU LGPL. *Jason* não provê suporte explícito para segurança e não é compatível com FIPA, mas provê recursos para desenvolvimento como ferremantas de *debug* e IDE. *Jason* não possui uma versão leve, mas considerando-se o tamanho de seu código, é possível portá-lo para a execução em ambientes mais restritos.

Tabela 4.2: *Frameworks* avaliados

Plataforma	CP	LPA	FD	MB	IDE	ET	FC	SG
AF	●	●	●	●	●	●	●	○
Jason	●	●	●	●	●	●	○	○
JADE	●	○	●	◐	●	○	●	●
3APL	●	●	●	○	●	◐	○	○
FIPA-OS	●	●	○	○	○	◐	●	○

A tabela 4.2 agrupa as propriedade avaliadas para os *frameworks* avaliados. As propriedades foram classificadas como presentes, ausentes ou parcialmente presentes. As propriedades, representadas pelas siglas estão definidas na seção **4.3.1**. Colocando-se uma ordem de *framework* com mais propriedades presentes para framework com menos propriedades presentes temos: 1º *Agent Factory*(AFME), 2º *Jason*, 3º JADE (LEAP), 4º 3APL (3APL-M) e 5º FIPA-OS (MicroFIPA-OS).

Levando em consideração somente esta avaliação, a escolha natural seria pelo *Agent Factory*, mas, conforme descrito no início deste capítulo o objetivo deste trabalho é propor um *framework* para o desenvolvimento de sistemas multiagente plenamente distribuídos, seguindo-se um modelo que tenha portabilidade, seja *standalone*, *open source* e com uma curva de aprendizagem curta. Desta forma, o autor deste trabalho optou pelo *Jason*, visto que embora *Agent Factory* seja *open source* e possua uma versão leve, ao observar sua implementação o autor observou que o mesmo faz uso pesado de execução concorrente, o que dificultaria muito portar seu código para execução *standalone*.

4.4 O *FRAMEWORK JASON*

Além de interpretar a linguagem AgentSpeak(L) original, segundo Bordini, Hubner e Wooldridge (2007), o Jason possui os seguintes recursos:

- negação forte (*strong negation*), portanto tanto sistemas que consideram mundo-fechado (*closed-world*) quanto mundo-aberto (*open-world*) são possíveis;
- tratamento de falhas em planos;
- comunicação baseada em atos de fala (incluindo informações de fontes como anotações de crenças);
- anotações em identificadores de planos, que podem ser utilizadas na elaboração de funções personalizadas para a seleção de planos;
- suporte para o desenvolvimento de ambientes (que normalmente não é programada em AgentSpeak; no Jason o ambiente é programado em Java);
- possibilidade de especializar (em Java) as funções de seleção, de planos, as funções de confiança e toda a arquitetura do agente (percepção, revisão de crenças, comunicação e atuação);
- possuir uma biblioteca básica de "ações internas";
- possibilitar a extensão da biblioteca de ações internas.

Além de programar o código AgentSpeak dos agentes e prover o arquivo de configuração do SMA, para viabilizar a execução do SMA é necessário criar o ambiente que os agentes perceberão e atuarão. Isto é feito na forma de um classe Java. Esta classe deve ter pelo menos o seguinte código:

hello

```
1  import java.util.*;
2  import jason.*
3
4  public class <EnvironmentName> extends Environment {
5          public boolean executeAction(String ag, Term act) {
6          ...
7          }
8  }
```

onde *<EnvironmentName>* é o nome da classe ambiente que será colocada no arquivo de configuração do SMA.

A super classe Environment possui o método *getPercepts* que retorna uma lista onde o programador pode incluir e remover as percepçõees que os agentes terão. Normalmente, a maior parte do código do ambiente é escrita no método *executeAction*. Sempre que um agente tenta executar uma ação básica, o nome do agente e um objeto *Term* representando a ação escolhida pelo agente são passados como parâmetros para este método. Portanto, o código deste método deve verificar se a ação é válida e então realizar o que for necessário para que a a ação seja de fato executada. Possivelmente a execução da ação irá alterar as percepções dos agentes. Se o retorno do método for verdadeiro (*true*) significa que a ação executou com sucesso.

Feita a avaliação de *frameworks*, de acordo com as propriedades apresentadas e definidas neste capítulo, especialmente na seção 4.3, selecionou-se então o *framework Jason* para servir de base ao *framework* proposto por este trabalho, para o desenvolvimento de sistemas multiagente plenamente distribuídos.

Conforme descrito na seção **2.2.5**, um *Web Toolkit* facilita o processo de desenvolvimento de aplicativos *web* no lado do cliente, permitindo que os desenvolvedores criem rapidamente e mantenham aplicativos *front end JavaScript* complexos e de alto desempenho em linguagens de programação como *Java*. Por este motivo, o autor deste trabalho decidiu fazer uso do *Google Web Toolkit* (GWT) (DEWSBURY, 2008).

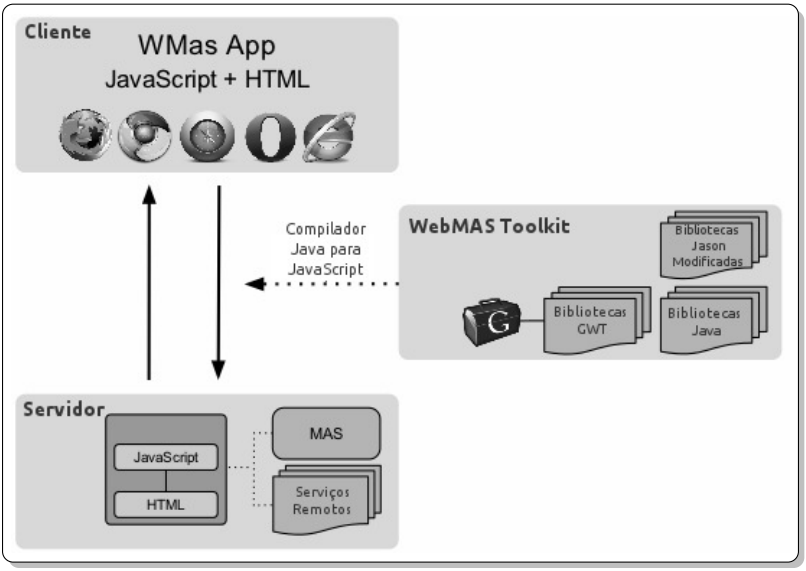

Figura 4.3: *Framework* para desenvolvimento de sistemas multiagente web plenamente distribuídos - WebMAS Toolkit

A figura 4.3 apresenta a arquitetura abstrata do WebMAS Toolkit. Na parte superior da imagem é representada a execução do lado do cliente do sistema multiagentes. Com o uso de GWT é possível compilar o *Java* empregado nos sistema multiagentes desenvolvidos, gerando um arquivo executável (*JavaScript* + HTML) compatível e otimizado para cada navegador *web*.

Do ponto de vista do desenvolvedor, é possível usar praticamente qualquer biblioteca *Java* com WebMAS Toolkit, bastando para tal, que a mesma seja compatibilizada com o GWT e por consequência com o WebMAS Toolkit.

Dentre as principais vantagens em usar o GWT no modelo proposto, pode-se citar:

- Uma única linguagem para desenvolver tanto para cliente, quanto para o

servidor;

- GWT cuida das diferenças entre os navegadores;
- Minimiza o tamanho do código;
- Erros são encontrados em tempo de compilação;
- Integração completa com o *framework* de testes *JUnit* (incluindo suporte ao teste unitário);
- Disponível para *Windows*, *Linux* e *Mac*;
- Economia de espaço (além de reduzir o tamanho do código, os trechos não utilizados são eliminados);
- *Plugin* com suporte para desenvolvimento em IDEs como eclipse, netbeans, etc..

4.5 IMPLEMENTAÇÃO E VALIDAÇÃO

4.5.1 Conceitos e ferrametas envolvidas

Google Web Toolkit (GWT): Com o uso do GWT é possível programar rapidamente como no *JavaScript*, no mesmo ciclo "editar - atualizar - exibir" com o qual a maioria dos desenvolvedores está acostumado, com a vantagem adicional de poder depurar e percorrer o código *Java* linha por linha. O compilador do GWT compila o código fonte *Java* para arquivos *JavaScript* independentes e otimizados. Ao contrário dos *minifiers JavaScript* que funcionam somente em nível textual, reduzindo o tamanho dos arquivos através de renomeação de variáveis e métodos, o compilador GWT executa análises estáticas abrangentes e otimizações em toda a base de códigos do GWT, produzindo frequentemente *scripts* que carregam e executam mais rapidamente do que um *script* equivalente criado por um desenvolvedor. Por exemplo, o compilador GWT elimina código sem função com segurança, cortando agressivamente classes, métodos, campos e mesmo parâmetros de métodos não utilizados, para garantir que o *script* compilado seja o menor possível. Outro exemplo: o compilador GWT incorpora métodos de forma seletiva, eliminando os excessos das chamadas do método.

A compilação cruzada permite que sejam mantidas as abstrações e a modularidade necessárias para o desenvolvimento, sem prejudicar o desempenho do tempo de execução (DEWSBURY, 2008).

JavaScript Native Interface (JSNI): muitas vezes, ao se utilizar um *web toolkit*, é necessário integrar a API sendo desenvolvida com APIs já existentes (neste caso APIs *JavaScript*) e/ou é necessário ter acesso de baixo nível as funcionalidades do navegador internet. Em ambos os casos a JSNI resolve o problema, permitindo integrar o *JavaScript* diretamente no aplicativo escrito em *Java* (DEWSBURY, 2008).

Overlay Types: com a JSNI é possível chamar pedaços de código *JavaScript* a partir de código *Java*. Embora a JSNI funcione bem, só funciona no nível dos métodos individuais. Alguns cenários de integração requerem que o desenvolvedor entrelace mais profundamente objetos *JavaScript* e objetos *Java* e desta forma é necessária uma maneira de interagir diretamente com objetos *JavaScript* dentro do código-fonte *Java*. Com *overlay types* é possível integrar famílias inteiras de objetos *JavaScript* dentro do código *Java*, resolvendo este problema (DEWSBURY, 2008).

Web workers: são tarefas em segundo plano que podem ser facilmente criadas, receber e enviar mensagens de volta para os seus criadores. A criação de um *worker* é tão simples como chamar o construtor *Worker()* construtor, especificando um *script* para ser executado no escopo do *worker* (W3C, 2011c).

Com *Web Workers*, é possível executar *scripts* em threads distintas da thread de UI, o que permite muito mais eficiência na execução e fluidez na interface, principalmente por se tratar de um modelo que utiliza os núcleos (*cores*) do processador disponível.

Uma das características mais interessantes das especificações do W3C é a simplicidade de utilização dos recursos, e isso se aplica a *Web Workers*, o que torna o trabalho muito simples para o desenvolvedor.

A maneira mais simples para se utilizar um *Web Worker* é utilizar um arquivo de script separado. Poderia também ser utilizado um *script* inline e outras modalidades como *DedicatedWorkers* e *SharedWorkers* (W3C, 2011c).

4.5.2 Execução concorrente no lado cliente

Para possibilitar a execução de porções do código de forma concorrente no lado do cliente foram utilizados *web workers*. Uma API *Java* foi desenvolvida, utilizando-se do conceito do *JavaScript Native Interface* (JSNI) e do conceito de *overlay types* (tipos sobrepostos) presentes no GWT. Basicamente, a implementação *JavaScript* para *web workers* disponibilizada pelos navegadores de internet, foi encapsulada e extendida por uma API implementada em *Java*.

O diagrama de classes apresentado na figura 4.4 contêm as principais classes presentes na API desenvolvida. A classe *Worker*, por exemplo, pode ser utilizada para criar um novo *web worker* e executar código em segundo plano. A classe *Timer* tem a mesma finalidade que *java.util.Timer* servindo para se criar e agendar temporizadores. As classes *WorkerController*, *MessageHandler*, *ErrorHandler* e *WorkerManager* são utilizadas para a criação, controle e gerenciamento de eventos relativos aos *workers* dentro da plataforma.

Para que porções de código (um agente, por exemplo) possam ser executadas concorrentemente em um *web worker*, é necessário que seu código escrito em *Java* seja compilado e linkado como um módulo independente. No WebMAS Toolkit isso foi resolvido através da implementação e utilização de um linkador especial para gerar código para os workers.

De acordo com a especificação e, consequentemente, implementação

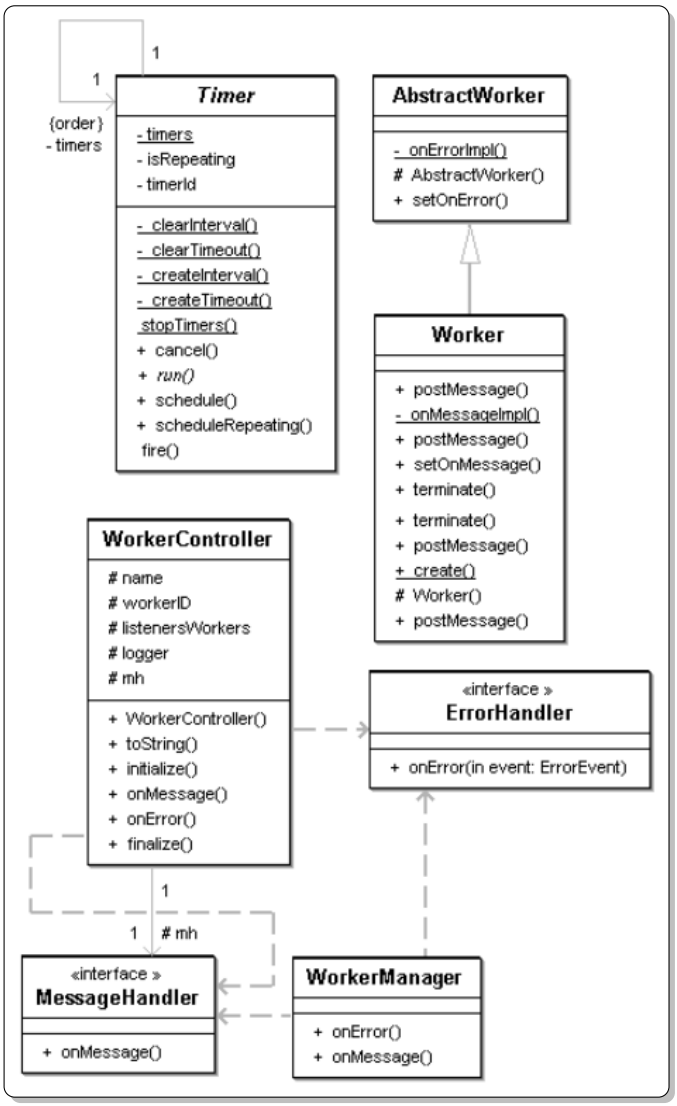

Figura 4.4: Diagrama de classe para WebMAS Toolkit *worker*

dos *web workers*, os mesmos só podem se comunicar com o mundo externo através do envio e do recebimento de mensagens de texto. Portanto, não é possível que métodos implementados dentro de um *web worker* seja chamado diretamente. Não existe compartilhamento de memória.

No WebMAS Toolkit, decidiu-se utilizar um padrão de projeto para implementar uma camada de software que possibilita efetuar chamadas de métodos entre os *workers* e/ou navegador indiretamente. O desenvolvedor efetua a chamada de método a partir de um *worker* como se fosse um método local, mas na verdade uma camada de software intermediária, que é gerada durante o processo de compilação cruzada para JavaScript, captura esta chamada e executa todos os procedimentos de serialização e deserialização dos argumentos, envio para o worker ou thread principal onde o método está implementado (em forma de mensagem), e caso exista retorno, serialização e deserialização do resultado (após a efetiva execução do método). Ao final retorna o resultado para o worker que efetuou a chamada do procedimento. Para os *workers*,tudo se passa como se esta camada intermediária não existisse. O padrão de projeto utilizado para tal é o *command patern* (GAMMA et al., 2002) e a camada implementada foi nomeada como *Worker Procedure Call* (WPC).

4.5.3 Worker Procedure Call (WPC)

Um WPC é uma chamada de procedimento a um método externo a um *worker*, podendo ele estar implementado na *thread* principal do navegador ou em outro *worker*. O código que é invocado a partir do *worker* também é conhecido como uma *action*, o ato de fazer uma chamada de procedimento externo é referido como invocar uma *action*.

O objetivo do WPC é tornar mais fácil a tarefa do desenvolvedor para implementar código que efetue troca de objetos *Java* (e por consequência *JavaScript*), durante uma chamada de procedimento entre *workers*, através do mecanismo de envio e recebimento de mensagens dos mesmos. O worker e/ou thread principal do navegador em que o procedimento é efetivamente executado é chamado de *Receiver*. O worker que invoca o procedimento é chamado de *Invoker*. O ato de executar uma chamada de procedimento pode ser referida como uma *action*. A implementação do WPC no WebMAS Toolkit é baseado no padrão de projeto *Java* chamado *command*. Dentro do código do *Invoker*, uma classe *proxy* gerada automaticamente vai ser usada para fazer chamadas para a *action* desejada.

Dentro do WebMAS Toolkit, e por conseguinte no WPC, a serialização de objetos é suportada pela injeção de código durante o processo de compilação com o uso do GWT. Para tornar os objetos de uma classe serializáveis, basta acrescentar linhas de código equivalentes as linhas 3, 4, 6 e 7 na listagem 4.1, a qualquer classe que se deseje tornar suas instâncias serializáveis. O objeto estático *WRITER* é usado para serializar e o objeto stático *READER* para deserializar. Os objetos são serializados para JSON *strings* e deserializados a partir delas.

```
1 public class Book {
```

```
2
3       public interface BookReader extends JsonReader<Book> {}
4       public static final BookReader READER = GWT.create(BookReader.class);
5
6       public interface BookWriter extends JsonWriter<Book> {}
7       public static final BookWriter WRITER = GWT.create(BookWriter.class);
8
9       String isbn;
10      int pages;
11      String title;
12      Author author;
13      List<String> reviews;
14
15
16      ...
17
18
19  }
```

Listagem 4.1: Mecanismo de Serialização do WebMAS Toolkit

Essa automatização do processo de serialização foi possível graças a módulo open source do GWT chamado *Piriti* que está disponível para *download* no *Google Project Hosting*.

O diagrama apresentado na figura 4.5, apresenta de forma gráfica como está estruturado o WPC dentro do WebMAS Toolkit.

No diagrama pode-se observar que *Invoker* e *Receiver* compartilham bibliotecas que definem *actions* e *results*, definidas para cada procedimento WPC. Durante o processo de compilação, o gerador (*generator*) WPC injeta código para o processo de serialização, cria as classes *proxy* e demais artefatos necessários.

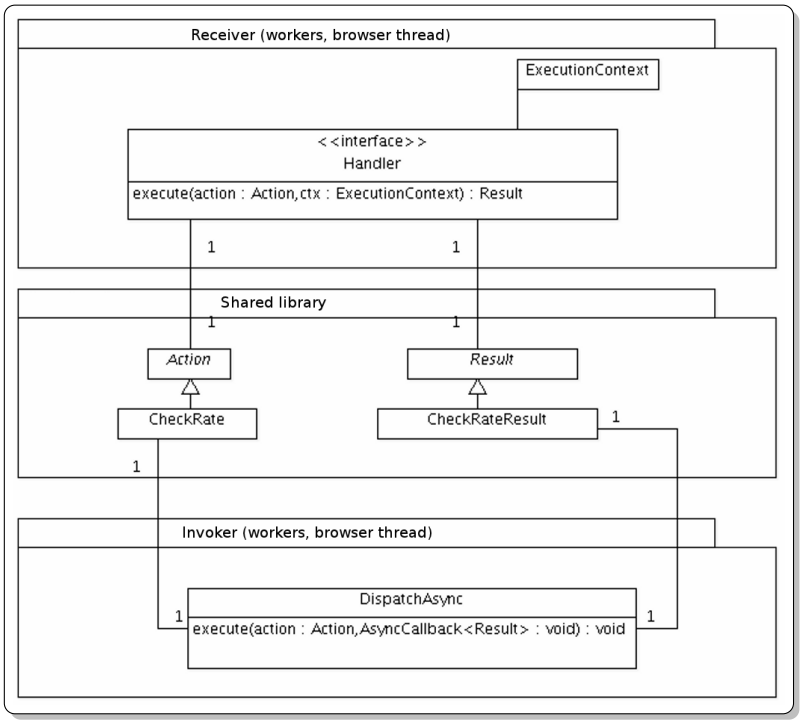

Figura 4.5: Modelo WPC (*Worker Procedure Call*)

5 RESULTADOS

Pelo fato de *Jason* ser implementado na linguagem *Java*, foi possível através da compilação cruzada com o uso do GWT, criar uma versão *JavaScript* do *framework*, compatível com a maioria dos navegadores de internet modernos. Para tornar possível a compilação cruzada, toda a computação que executava em *threads Java* teve de ser reestruturada e rearranjada para executar dentro dos *web workers*, encapsulados pela API desenvolvida no WebMAS Toolkit. Por outro lado, algumas classes como *Enviroment*, *InfraTier*, etc não sofreram mudanças significativas e, sendo assim, decidiu-se ater-se àquelas mudanças realizadas no *framework Jason* e que foram importantes para o seu correto funcionamento no lado do cliente. Essas classes que não foram significativamente alteradas já estão bem documentadas no *framework Jason* e não serão descritas neste trabalho.

O Google Web Toolkit inclui uma biblioteca que emula um subconjunto da biblioteca runtime Java. Esse conjunto pode ser traduzido automaticamente para JavaScript pelo GWT. Apenas um subconjunto da runtime Java é suportado. Esse recurso foi largamente utilizado no desenvolvimento do WebMAS Toolkit. Desta forma, classes Java, utilizadas pelo Jason, e não diponíveis no conjunto emulado pelo GWT, foram criadas e adicionadas a este conjunto.

Acredita-se que a grande contribuição do WebMAS Toolkit, é a possibilidade de se desevolver o sistema multiagente em uma linguagem de alto nível como Java, compilar para JavaScript para executar no lado do cliente ou, se for o caso, executá-lo no servidor ou em um desktop como uma aplicação Jason normal. Uma aplicação pode também, ser dividida e executar parcialmente no servidor, rodando em uma JVM e, parcialmente no navegador de internet, no lado do cliente, rodando na forma de JavaScript. O GWT fornece recursos que facilitam essa divisão e a comunicação client/servidor, a qual foi incorporada e ajustada para utilização no WebMAS Toolkit.

5.1 DEFINIÇÃO E EXECUÇÃO DE UM SMA COM O USO DO WEBMAS TOOLKIT

Esta seção descreve o ciclo programar-compilar-executar para se desenvolver um SMA utilizando-se o WebMAS Toolkit. No WebMAS Toolkit, a definição de um SMA deve incluir um conjunto de agentes definidos na linguagem *AgentSpeak* e um ambiente onde todos estes agentes atuarão, programado na linguagem Java. Estas informações devem ser inseridas em um arquivo texto com extensão *.json* e em conformidade com a sintaxe definida pela gramática JSON definida em Crockford (2006), para que o WebMAS Toolkit saiba onde encontrá-las. Com isso, não foram necessárias alterações estruturais no Jason, e assim, aplicações já desenvolvidas para o Jason padrão, podem ser facilmente adaptadas a execução no WebMAS Toolkit.

Conforme será apresentado adiante, a listagem 5.3 representa um arquivo de definição de um SMA. Na definição, a chave "*MAS*" representa o nome dado ao sistema multiagente em questão. "*infrastructure*", representa um objeto JavaScript que contem atributos relacionados a classe Java responsável pela inicialização do sistema multiagente. A chave "*environment*" da acesso ao objeto JavaScript que contêm os dados necessários a criação e inicialização do ambiente no qual os agentes serão inseridos e a chave "*agents*" refere-se a um objeto JavaScript que contêm a definição de todos os agentes que serão inseridos no SMA. O código AgentSpeak de cada agente, por exemplo, poder ser acessado através da chave "aslBundleClass".

Conforme descrito em 4.4, é necessário criar o ambiente em que os agentes serão inseridos para que executem suas tarefas. Isso é feito através da definição de uma classe Java no WebMAS toolkit.

O WebMAS Toolkit contêm um conjunto básico de funcionalidades para facilitar a criação de sistemas multiagente. No entanto, os desenvolvedores geralmente têm uma variedade de requisitos específicos ao desenvolver os seus sistemas, como, por exemplo, acesso a banco de dados, sistemas legados, interfaces gráficas, necessidade de percepções customizadas para os agentes, manipulação de intenções diferenciado, etc. O WebMAS Toolkit seguiu a abordagem já presente no Jason, que oferece suporte a extensões e customizações. Tais customizações e extensões envolvem programação Java. Com isso, durante o desenvolvimento do sistema multiagente é possível estender as bibliotecas já presentes, conforme necessário.

Uma vez que estejam definidos o código AgentSpeak para os agentes, a classe Java que representa o ambiente e as demais classes Java estendidas, conforme necessidade, basta compilar o código com o uso do compilador cruzado Java para JavaScrit do GWT e a aplicação SMA estará pronta para executar em qualquer navegador de internet com capacidade de renderização de HTML e execução de JavaScript, o que é o caso da maioria dos navegadores modernos.

Para realizar a depuração do código, é possível executar o aplicativo em desenvolvimento diretamente no navegador de internet. Isso é possível graças ao uso do GWT, que possui um plugin que auxilia o desenvolvimento de aplicativos na IDE eclipse, que permite visualizar alterações realizadas no código fonte, sem a necessidade de recompilação de todo o código, ou seja, a alteração pode ser visualizada no navegador, bastando para tal, que a página HTML em que o aplicativo está incorporado, seja atualizada.

5.2 ESTUDO DE CASO - *GOLD MINERS*

Para realização de um estudo de caso, o autor selecionou um dos muitos exemplos de sistemas multiagente desenvolvidos utilizando-se *Jason* e fez as adaptações necessárias para sua execução no WebMAS Toolkit. O exemplo escolhido foi o *Gold Miners* (HÜBNER; BORDINI, 2008), descrito a seguir.

Existem dois tipos de agentes na equipe: os mineiros (*miners*) e o líder (*leader*). Mineiros são os agentes que interagem dentro do ambiente de

simulação e ajudam o líder na coordenação de algumas atividades. O líder ajuda os mineiros a coordenarem-se em duas situações. Inicialmente divide o *grid* que representa o ambiente em quatro quadrantes e aloca mineiros em cada um deles; os mineiros, desta forma, procuram por ouro em lugares diferentes. Como temos seis agentes e apenas quatro quadrantes, os dois agentes sem um quadrante específico irão procurar ouro em qualquer lugar do *grid*, preferindo os lugares menos visitados pelos outros. A segunda situação de coordenação é o processo de negociação que é iniciado quando um mineiro vê um pedaço de ouro e não é capaz de coletá-lo. Este mineiro transmite a localização do ouro para outros mineiros que, em seguida, enviam propostas para o líder. O líder escolhe a melhor oferta e aloca o agente correspondente para recolher aquele pedaço de ouro. O protocolo também estabelece que sempre que algum agente decide ir para algum lugar de ouro, deve anunciá-lo aos outros (para que possam reconsiderar suas intenções). Da mesma forma, eles devem anunciar quando eles coletam um pedaço de ouro.

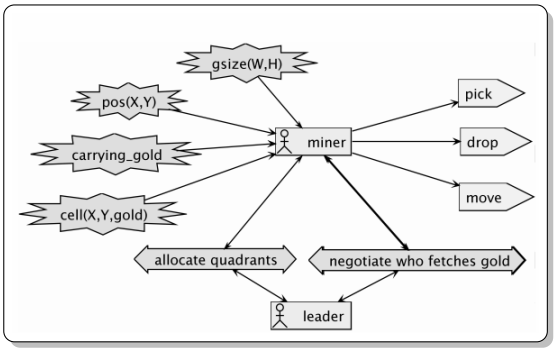

Figura 5.1: Sistema multiagente *Gold Miners*, baseado em Hübner e Bordini (2008)

Para tornar a descrição mais clara, as figuras 5.1 e 5.2 apresentam a modelagem do sistema multiagente e do agente mineiro respectivamente.

Descrito o exemplo utilizado, partimos para como ele foi implementado no WebMAS Toolkit. Para melhor entender a dinâmica de desenvolvimento, os paragrafos a seguir decrevem como utilizar o WebMAS Toolkit, para depois realmente fazê-lo através do exemplo.

No WebMAS Toolkit, por consequência do uso do *Jason*, os agentes são definidos usando-se a linguagem *AgentSpeak*. A listagem 5.1 apresenta parcialmente o código agentspeak para o agente *leader*, no estudo de caso utilizando o WebMas Toolkit. Já o ambiente, conforme descrito em 4.4, precisa ser programado em Java. WebMAS Toolkit provê uma vasta API *Java* para a definição, configuração e execução de sistemas multiagente. Classes *Java*, que implementam os principais recursos e funções dentro do *framework* podem ser extendidas e customizadas para atender as necessidades do desenvolvedor.

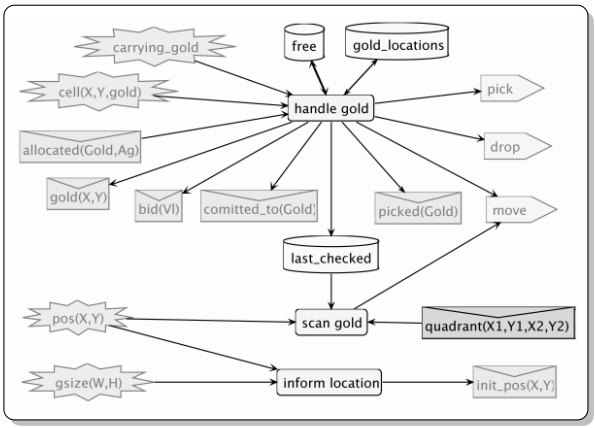

Figura 5.2: Agent mineiro (*miner*), baseado em Hübner e Bordini (2008)

Além da API do *Jason*, outras APIs para o desenvolvimento de agentes podem ser utilizadas em conjunto, bastando alguns ajustes para torná-la compatível com o GWT.

```
1  // leader agent
2  /* quadrant allocation */
3
4  @quads[atomic]
5  +gsize(S,W,H) : true
6    <- // calculates the area of each quadrant and remembers them
7       .print("Defining quadrants for simulation ",S," : ",W,"x",H);
8
9       CellH = H div 2;
10      +quad(S,1, 0,       0,       W div 2 - 1, CellH - 1);
11      +quad(S,2, W div 2, 0,       W - 1,       CellH - 1);
12      +quad(S,3, 0,       CellH,   W div 2 - 1, (CellH * 2)-1);
13      +quad(S,4, W div 2, CellH,   W - 1,       (CellH * 2)-1);
14
15      !inform_quad(S,miner1,1);
16      !inform_quad(S,miner2,2);
17      !inform_quad(S,miner3,3);
18      !inform_quad(S,miner4,4).
19 ...
20
21 /* negotiation for found gold */
22
23 +bid(Gold,D,Ag)
24    : .count(bid(Gold,_,_),5)  // five bids were received
25   <- //.print("bid from ",Ag," for ",Gold," is ",D);
26      !allocate_miner(Gold);
27      .abolish(bid(Gold,_,_)).
28 //+bid(Gold,D,Ag)
29 //  <- .print("bid from ",Ag," for ",Gold," is ",D).
30
31 +!allocate_miner(Gold)
32   <- .findall(op(Dist,A),bid(Gold,Dist,A),LD);
33      .min(LD,op(DistCloser,Closer));
34      DistCloser < 10000;
```

```
35        .print("Gold ",Gold," was allocated to ",Closer, " options ware ",LD);
36        .broadcast(tell,allocated(Gold,Closer)).
37        //-Gold[source(_)].
38    -!allocate_miner(Gold)
39      <- .print("could not allocate gold ",Gold).
```

Listagem 5.1: Código *agentspeak* para o agent *leader*

Durante o desenvolvimento, com o uso do plugin do GWT para a IDE eclipse, alterações no código podem ser imediatamente visualizadas diretamente no navegador. Não há necessidade de recompilar para *JavaScript*. Basta fazer as alterações e clicar em "Atualizar" no navegador.

Durante a produção, o código é compilado para *JavaScript*, mas durante o desenvolvimento, ele é executado como *bytecode* no *Java Virtual Machine*. Isso significa que, quando o código executa uma ação, como por exemplo um evento do mouse, ocorre uma depuração *Java* completa. Tudo o que o depurador Java pode fazer se aplica ao código GWT também e consequente ao WebMAS, portanto, coisas como break points e passo único, estão naturalmente disponíveis. No WebMAS o GWT compilará o código *Java* nos arquivos independentes simples JavaScript que poderão ser hospedados em qualquer servidor web. Além disso, os aplicativos GWT suportam automaticamente *IE*, *Firefox*, *Mozilla*, *Safari* e *Opera*, sem necessidade de detectar o navegador ou criar casos especiais no programa, o código é criado uma vez e o compilador GWT o transforma no *JavaScript* mais eficiente para o navegador específico de cada usuário.

A listagem 5.3 representa o arquivo de configuração que define a estrutura geral para o sistema multiagente *goldminers* implementado com o uso do WebMAS Toolkit. Da linha 2 até a linha 15, está definido a classe principal do SMA *GoldMiners_Runner*, que é responsável por inicializar todo SMA dentro da plataforma. Da linha 16 até a linha 58, são definidas as configurações para os agentes. Atributos como nome do agente, classe controladora, classe que representa a base de crenças de cada agente, etc, são definidos neste arquivo. Vale lembrar que por limitações de espaço este arquivo de denição do SMA em questão, não está sendo apresentado em sua forma completa.

Através do pacote *mas2js* disponível no WebMAS Toolkit é possível acessar estes dados de configuração de uma maneira prática e intuitiva. O método apresentado na listagem 5.2, por exemplo, retorna um objeto do tipo *ClassDef*, que representa a classe que implementa a infraestrutura utilizada pelo SMA do *Gold Miners*.

```
1         public native ClassDef getInfraDef() /*-{
2                 return this.infrastructure;
3         }-*/;
```

Listagem 5.2: Example de método presente no pacote *mas2js*

A primeira vista, isso parece ser relativamente banal, mas, pelo fato de permitir que toda a configuração do SMA se concentre em um arquivo único, que é lido uma única vez e passa a ser acessado programaticamente através do pacote *mas2js*, esta solução se constitui em uma ferramenta poderosa. Isso é

possível graças ao uso da syntax JSON no arquivo de configuração, a JSNI e aos *overlay types*. No WebMAS Toolkit, isso ainda vai um pouco além. O autor faz uso de um recurso muito interessante, disponível no GWT e incorporado no WebMAS, chamado de *Resource Bundle*. Com ele é possível, por exemplo, garantir que o arquivo de configuração correto está sendo acessado, uma vez que com este recurso, o arquivo de configuração é incorpado ao código gerado pelo processo de compilação cruzada. *Resource Bundle* é utilizado em outros locais do WebMAS, nas imagens que representam os agentes dentro do SMA *Gold Miners*, por exemplo.

```
{   "MAS": "goldminers",
    "infrastructure": {
        "fullQualifiedName": "... .infra.GoldMiners_Runner"
    },
    "environment": {
        "fullQualifiedName": "... .MiningEnvironment",
        "params": [
                {
                        "type"  : "int",
                        "value" : 5
                },
            ...
        ]
    },
    "agents": [
        {
            "agentName": "dummy",
            "aslBundleClass": "... .DummyASLBundle",
            "controllerClass": {
                "fullQualifiedName": "... .DummyController"
            },
            "numberOfAg": 6
        },
        {
            "agentName": "leader",
            "aslBundleClass": "... .LeaderASLBundle",
            "controllerClass": {
                "fullQualifiedName": "... .LeaderController"
            },
            "beliefBaseClass": {
                "fullQualifiedName": "... .DiscardBelsBB",
                "params": [ ... ]
            }
        },
        {
            "agentName": "miner",
            "aslBundleClass": "... .MinerASLBundle",
            "controllerClass": {
                "fullQualifiedName": "... .MinerController"
            },
            "agentWorkerPath":  {
                "path": "../goldmineragents/goldmineragents.nocache.js"
            },
            "numberOfAg": 6
        }
    ]
}
```

Listagem 5.3: Arquivo de configuração Json

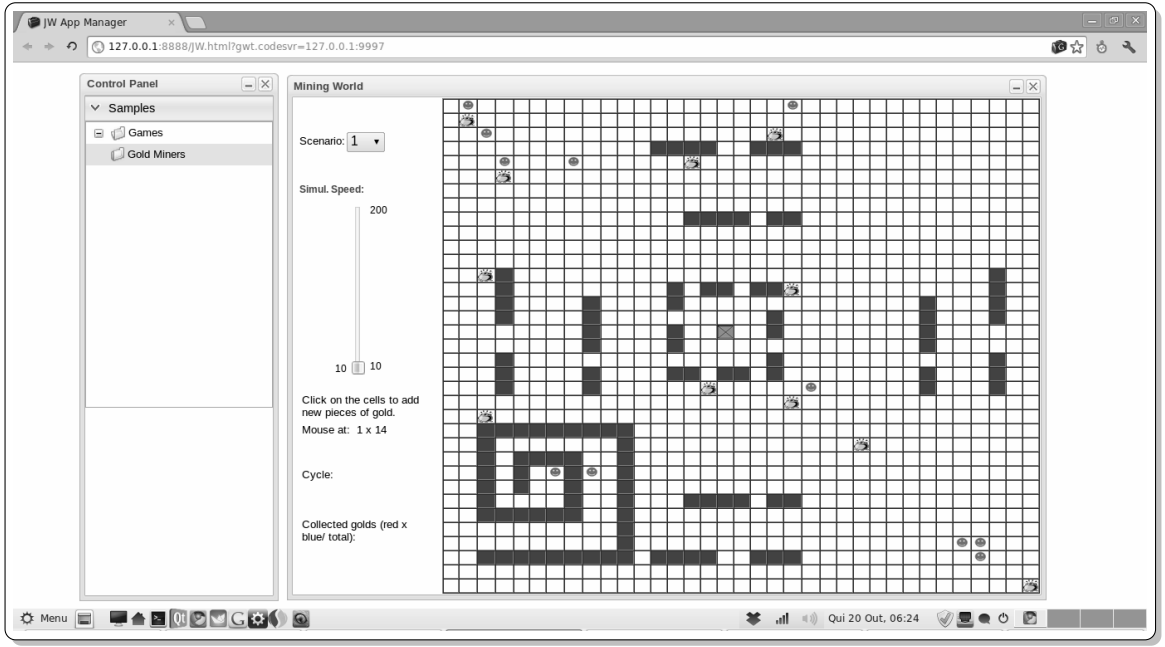

Figura 5.3: *Gold Miner* rodando no navegador de internet

É importante lembrar que todo esse processo de inicialização do SMA, criação e execução dos agentes ocorre no lado do cliente. Neste exemplo, o servidor é meramente um hospedeiro e fornecedor dos arquivos fontes.

Na figura 5.3 é apresentado o sistema multiagente *Gold Miners* rodando no navegador de internet Google Chrome, versão 15, para sistema operacional Linux. O sistema foi desenvolvido usando-se o WebMAS Toolkit e a interface gráfica foi desenvolvida com o uso das APIs gráficas disponíveis no GWT.

Através da realização do estudo de caso exposto acima, foi possível observar que as características desejáveis ao framework, relatadas no capítulo 4, foram atendidas. Conseguiu-se propor um framework totalmente *opensource*, que trata a maioria dos problemas relacionados a portabilidade e os aplicativos desenvolvidos com a ferramenta, executam *standalone* no lado do cliente. A característica *Curva de Aprendizagem* foi parcialmente atendida, pela razão exposta no paragrafo a seguir.

As limitações do WebMAS Toolkit são as mesmas presentes no GWT. Dentre as principais pode-se citar o o tempo longo necessário para um ciclo de desenvolvimento (implantar, testar, ajustar, reimplantar) e a curva de aprendizagem. O processo de compilação é relativamente demorado e nem todas as alterações no código, podem ser visulizadas atualizando a página do aplicativo no navegador. Para usar o GWT corretamente, o desenvolvedor é forçado a usar padrões de projeto. Devido a isso a curva de aprendizado é bastante íngreme para desenvolvedores iniciantes.

Embora o estudo de caso não trate sobre a comunicação entre agentes rodandoo em diferentes plataformas, ou de maneira mais simples em diferentes navegadores, é possível que agentes localizados em direntes plataformas ou mesmo servidores se comuniquem e migrem dentro da mesma. Atualmente o mecanismo utilizado para essa comunicação é o chamado *Server Push* (GRAVELLE, 2010). Nesse mecanismo a aplicação rodando no cliente faz uma requisição HTTP de longa data que permite o servidor web envie dados para o navegador, sem que o navegador explicitamente solicite. Cometa (*Commet*) é um termo genérico, abrangendo várias técnicas para a realização desta interação. Todos estes métodos dependem de recursos incluídos por padrão em navegadores, como *JavaScript*. Através desse mecanismo e com a ajuda do servidor é possível criar uma conexão "virtual" entre plataformas/sub-plataformas rodando em clientes diferentes, permitindo assim que os agentes efetuem a trocas de informações ou mesmo migrem dentro da plataforma.

Atualmente ainda não é possível a comunicação direta (P2P), via web, entre dois aplicativos, rodando em navegadores diferentes. Por isso a utilização do mecanismo de *Server Push*. Porém, existe na W3C um trabalho em andamento para a especificação (ALVESTRAND, 2011; W3C, 2011b) de uma API que permitirá a utilização de conexões P2P para a comunicação entre aplicativos web rodando em diferentes navegadores. Essa tecnologia poderá ser utilizada para a implementação de um módulo dentro do WebMAS Toolkit para comunicação de agentes rodando em diferentes navegadores sem a necessidade da intervenção de um servidor web.

6 CONCLUSÕES E TRABALHOS FUTUROS

Acredita-se, com este trabalho, ter-se alcançado seu objetivo de propor um modelo para o desenvolvimento de sistemas multiagente plenamente distribuídos, permitindo um melhor aproveitamento dos recursos diponíveis no lapo do cliente por padrão. Para a utilização destes recursos de computação, foi proposto um *framework* que possibilita que sejam criados sistemas multiagente, que podem executar total ou parcialmente no lado do cliente, reduzindo-se assim a dependência e a sobrecarga na rede experimentada pela arquitetura cliente-servidor tradicional.

Com o *framework* desenvolvido é possível que sejam criados aplicativos *web* que incorporam características naturalmente presentes no paradigma de agentes, como por exemplo, permitir que aplicativo evolua e se adapte as necessidades do usuário a medida que este o for utilizando. Também, tornou-se possível prover maior autonomia e proatividade as aplicações *web* utilizando-se dos conceitos e abstrações em alto nível do paradigma de agentes.

Pelo fato do *framework* proposto ter sido implementado com o uso do GWT, a complexidade do desenvolvimento e manutenção de aplicações *web*, ditadas pelas tendências atuais, foram amenizadas. A utilização da mesma linguagem de programação tanto no lado do cliente, quanto no servidor e abstração das peculiaridades de cada navegador de internet pelo GWT são os fatores chave na solução deste problema.

O processamento concorrente entre tarefas no lado do cliente foi possibilitado pelo desenvolvimento de uma API que encapsula os *web workers* especificados no HTML5 e implementados pela maioria dos navegadores modernos. Também foi desenvolvida uma API (WPC) para viabilizar a chamada de métodos entre *web workers*, uma vez que como o uso destes, só é possível troca de informações através de messagens de texto.

Levando-se em conta as características desejáveis ao *framework*, levantadas no capítulo 4, Conseguiu-se propor um *framework* totalmente *open-source*, que trata a maioria dos problemas relacionados portabilidade, onde os aplicativos desenvolvidos executam no lado do cliente utilizando-se somente de recursos disponíveis por padrão.

Em comparação com as propostas relatadas nos trabalhos correlatos, o WebMAS Toolkit apresenta soluções para as limitações de portabilidade e necessidade de realizar instalação e a configuração de extensões no lado do cliente para ter acesso aos aplicativos desenvolvidos com aquelas soluções. Também possibilita que se tire vantagem da execução concorrente no lado do cliente, criando-se aplicativos mais robustos e com menor dependência de um servidor.

Como trabalhos futuros fica a implementação e validação de um módulo de segurança para o WebMAS Toolkit, a modelagem, implementação e validação de um módulo para a comunicação P2P entre clientes e a adequação do *framework* aos padrões internacionais estabelecidos pela FIPA para permitir a interoperabilidade com outras plataformas.

REFERÊNCIAS

ABECK, S.; KOPPEL, A.; SEITZ, J. A management architecture for multi-agent systems. In: . [S.l.: s.n.], 1998. p. 133 –138.

ALVESTRAND, H.T. Overview: Real time protocols for brower-based applications. 2011.

ASSEMBLY, E.G. Ecmascript language specification. *Standard ECMA-262,*, 1999.

BARRY, P. Introducing openlaszlo 4. *Linux Journal*, Specialized Systems Consultants, Inc., v. 2008, n. 171, p. 4, 2008.

BELLIFEMINE, F.L. et al. *Developing Multi-agent Systems with JADE*. [S.l.]: Springer, 2007.

BELLIFEMINE, F.; POGGI, A.; RIMASSA, G. JADE–A FIPA-compliant agent framework. In: CITESEER. *Proceedings of PAAM*. [S.l.], 1999. v. 99, p. 97–108.

BELLIFEMINE, F.; POGGI, A.; RIMASSA, G. Developing multi-agent systems with JADE. *Intelligent Agents VII Agent Theories Architectures and Languages*, Springer, p. 42–47, 2001.

BERNERS-LEE, T.; HENDLER, J.; LASSILA, O. The semantic web: Scientific american. *Scientific American*, v. 284, n. 5, p. 34–43, 2001.

BOARD, J. Jade web services integration gateway (wsig) guide. *Whitestein Technologies AG*, 2005.

BORDINI, R.H.; DASTANI, M. *Multi-agent programming: languages, platforms, and applications*. [S.l.]: Springer-Verlag New York Inc, 2005.

BORDINI, R.H. et al. Multi-Agent Programming: Languages, Tools and Applications. *Multiagent Systems, Artificial Societies, And Simulated Organizations*, p. 419, 2009.

BORDINI, R.H.; HUBNER, J.F.; WOOLDRIDGE, M.J. *Programming multi-agent systems in AgentSpeak using Jason*. [S.l.]: Wiley-Interscience, 2007.

BORSELIUS, N. Security in multi-agent systems. In: CITESEER. *Proceedings of the 2002 International Conference on Security and Management (SAM'02)*. [S.l.], 2002. p. 31–36.

BRADSHAW, J.M. *Software agents*. [S.l.]: MIT Press Cambridge, MA, USA, 1997.

BRATMAN, M.E. *Intention, plans, and practical reason*. [S.l.]: Harvard University Press Cambridge, MA, 1987.

BRAUBACH, L.; POKAHR, A.; LAMERSDORF, W. Jadex: A BDI-agent system combining middleware and reasoning. *Software agent-based applications, platforms and development kits*, Springer, p. 143–168, 2005.

BRENNER, W.; WITTIG, H.; ZARNEKOW, R. *Intelligent Software Agents: Foundations and Applications*. [S.l.]: Springer-Verlag New York, Inc. Secaucus, NJ, USA, 1998.

BRENNER, W.; WITTIG, H.; ZARNEKOW, R. *Intelligent software agents: foundations and applications*. [S.l.]: Springer-Verlag New York, Inc. Secaucus, NJ, USA, 1998.

CAIRE, G.; PIERI, F. Leap user guide. *TILab, Jan*, 2006.

CHAPPELL, D. *Understanding ActiveX and OLE: a guide for developers and managers*. [S.l.]: Microsoft Press, 1996.

COLLIER, R.W. Agent factory: A framework for the engineering of agent-oriented applications. 2002.

COLLIER, R. et al. Beyond prototyping in the factory of agents. *Multi-Agent Systems and Applications III*, Springer, p. 1068–1068, 2003.

CORMODE, G.; KRISHNAMURTHY, B. Key differences between Web 1.0 and Web 2.0. *First Monday*, Citeseer, v. 13, n. 6, p. 2, 2008.

COULOURIS, G.F.; DOLLIMORE, J.; KINDBERG, T. *Distributed systems: concepts and design*. [S.l.]: Addison-Wesley Longman, 2005. ISBN 0321263545.

CROCKFORD, D. The application/json media type for javascript object notation (json). 2006.

DEWSBURY, R. Google web toolkit applications. Addison-Wesley Professional, 2007.

DEWSBURY, R. *Google web toolkit applications*. [S.l.]: Prentice-Hall PTR, 2008.

DIPIPPO, L. Cingiser et al. A real-time multi-agent system architecture for e-commerce applications. In: . [S.l.: s.n.], 2001. p. 357 –364.

FININ, T. et al. Kqml—a language and protocol for knowledge and information exchange. In: *Proceedings of the 13th Intl. Distributed Artificial Intelligence Workshop*. [S.l.: s.n.], 1994. p. 127–136.

FIPA, T.C. FIPA Agent Management Specification. *FIPA T.C. B.*, 2002.

GAMMA, E. et al. *Design patterns*. [S.l.]: Addison-Wesley Reading, MA, 2002.

GASMELSEID, T.M. A multi agent negotiation framework in resource bounded environments. In: . [S.l.: s.n.], 2006. v. 1, p. 465 – 470.

GENESERETH, M.R.; KETCHPEL, S.P. Software Agent. *Communications of the ACM*, v. 37, n. 7, p. 48–53, 1994.

GENEVA, S. FIPA Abstract Architecture Specification. *Change*, 2002.

GLUZ, J.C.; VICCARI, R.M. Linguagens de comunicação entre agentes: Fundamentos padrões e perspectivas. *Jornada de Mini-Cursos de Inteligência Artificial*, v. 3, p. 53–102, 2003.

GODIN, S. *Web4*. Jan. 2007. Acessado em: http://sethgodin.typepad.com, Nov. 2010.

GRAVELLE, R. Comet programming: Using ajax to simulate server push. *Webreference, WebMediaBrands Inc., http://www. webreference. com/programming/javascript/rg28*, 2010.

HAENSCH M. O., Campos D. e Silveira R. A. Uma plataforma para desenvolvimento de sistemas multiagente bdi na web. *Segunda edição dos Anais do V Workshop-Escola de Sistemas de Agentes, seus Ambientes e apliCações*, 2011.

HENDLER, J. Agents and the semantic web. *Intelligent Systems, IEEE*, v. 16, n. 2, p. 30 – 37, mar-apr 2001. ISSN 1541-1672.

HENDLER, J. Web 3.0 emerging. *Computer*, v. 42, n. 1, p. 111 –113, 2009. ISSN 0018-9162.

HERMAN, I. *W3C Semantic Web Frequently Asked Questions*. 2004.

HINDRIKS, K.V. et al. Agent programming in 3apl. *Autonomous Agents and Multi-Agent Systems*, Springer, v. 2, n. 4, p. 357–401, 1999.

HOMMEAUX, E. Prud'; SEABORNE, A. et al. SPARQL query language for RDF. *W3C working draft*, v. 4, 2006.

HÜBNER, J.; BORDINI, R. Developing a team of gold miners using jason. *Programming Multi-Agent Systems*, Springer, p. 241–245, 2008.

HUHNS, M. Software agents: The future of web services. *Agent Technologies, Infrastructures, Tools, and Applications for E-Services*, Springer, p. 1--18, 2010.

INTERCHANGE, K. The darpa knowledge sharing effort: Progress report. *Readings in Agents*, Morgan Kaufmann Pub, p. 243, 1998.

JACOBS, I.; WALSH, N. Architecture of the world wide web. 2004.

JENNINGS, N.R. An agent-based approach for building complex software systems. *Communications of the ACM*, ACM, v. 44, n. 4, p. 35–41, 2001.

JENNINGS, Nick. Building automated negotiators. In: *Proceedings of the NODe 2002 agent-related conference on Agent technologies, infrastructures, tools, and applications for E-services*. Berlin, Heidelberg: Springer-Verlag, 2003. (NODe'02), p. 19–19. ISBN 3-540-00742-3. Disponível em: <http://dl.acm.org/citation.cfm?id=1756326.1756329>.

JIMENEZ, G.; BARRADAS, C. Knowledge Management System Based on Web 2.0 Technologies. *Web-based Support Systems*, Springer, p. 273–301, 2010.

KESTEREN, A. Van; JACKSON, D. The xmlhttprequest object. *World Wide Web Consortium, Working Draft WD-XMLHttpRequest-20070618*, 2007.

KFIR-DAHAV, N.E.; MONDERER, D.; TENNENHOLTZ, M. Mechanism design for resource bounded agents. In: . [S.l.: s.n.], 2000. p. 309 –315.

KOCH, F. et al. Programming deliberative agents for mobile services: the 3apl-m platform. *Programming Multi-Agent Systems*, Springer, p. 222–235, 2006.

LANGE, D. et al. Aglets: Programming mobile agents in java. *Worldwide Computing and Its Applications*, Springer, p. 253–266, 1997.

LEIGHTON, Luke Kenneth Casson. *Pyjamas Book*. Out. 2010. Disponível em: http://pyjs.org/book/output/Bookreader.html.

LIEDEKERKE, M.H. Van; AVOURIS, N.M. Debugging multi-agent systems. *Information and Software Technology*, Elsevier, v. 37, n. 2, p. 103–112, 1995.

LISBOA, N. *Breve comparação dos termos Web 1.0, 2.0, 3.0 e 4.0*. Out. 2009.

LU, Feng; HUANG, Mei. Research and design of security in multi-agent system. In: . [S.l.: s.n.], 2006. p. 1 –4. ISSN 0537-9989.

LUBBERS, P.; ALBERS, B.; SALIM, F. Overview of html5. *Pro HTML5 Programming*, Springer, p. 1–23, 2010.

MALIK, S.S. et al. Inter platform agent mobility in fipa compliant multi-agent systems. In: . [S.l.: s.n.], 2005. p. 393 – 396.

MINOTTI, Mattia; PIANCASTELLI, Giulio; RICCI, Alessandro. Agent-oriented programming for client-side concurrent web 2.0 applications. In: CORDEIRO, José et al. (Ed.). *Web Information Systems and Technologies*. [S.l.]: Springer Berlin Heidelberg, 2010, (Lecture Notes in Business Information Processing, v. 45). p. 17–29.

MOURATIDIS, H.; GIORGINI, P.; MANSON, G. Modelling secure multiagent systems. In: ACM. *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*. [S.l.], 2003. p. 866.

MOZILLA, Fondation. *Plugins*. Out. 2011. Https://developer.mozilla.org/en/Plugins. Https://developer.mozilla.org/en/Plugins.

MULDOON, C. et al. Agent factory micro edition: A framework for ambient applications. *Computational Science–ICCS 2006*, Springer, p. 727–734, 2006.

NDUMU, D.T. et al. Visualising and debugging distributed multi-agent systems. In: ACM. *Proceedings of the third annual conference on Autonomous Agents*. [S.l.], 1999. p. 326–333.

NEMETH, Justin. *JavaScript Improvements in Modern Web Browsers*. 2009. Online: acessado em 5-Julho-2011. Disponível em: <http://assets.webassist.com/roadmaps/roadmap_08.pdf>.

NETWORK, Mozilla Developer. *Gecko*. 2011. Online: acessado em 6-Julho-2011. Disponível em: <https://developer.mozilla.org/en/Gecko>.

NGUYEN, X.; KOWALCZYK, R. Ws2jade: Integrating web service with jade agents. *Service-Oriented computing: Agents, Semantics, and Engineering*, Springer, p. 147–159, 2007.

NOKIA, Corporation. *WebKit in Qt*. 2011. Online acessado em 5-Julho-2011. Disponível em: http://doc.qt.nokia.com/4.7-snapshot/qtwebkit.html#details.

NWANA, H.S. Software agents: An overview. *Knowledge Engineering Review*, v. 11, n. 3, p. 205–244, 1996.

NWANA, H.S. Software agents: An overview. *The Knowledge Engineering Review*, Cambridge Univ Press, v. 11, n. 03, p. 205–244, 2009.

O'REILLY, T. What is web 2.0. *Design patterns and business models for the next generation of software*, San Francisco:, v. 30, p. 2005, 2005.

O'REILLY, T. Web 2.0 compact definition: Trying again. Retrieved, 2006. Acessado em: http://radar.oreilly.com/, Nov. 2010.

POKAHR, A.; BRAUBACH, L.; LAMERSDORF, W. Jadex: A BDI reasoning engine. *Multi-Agent Programming*, Springer, p. 149–174, 2005.

POSLAD, S.; BUCKLE, P.; HADINGHAM, R. The FIPA-OS agent platform: Open source for open standards. In: CITESEER. *Proceedings of the 5th International Conference and Exhibition on the Practical Application of Intelligent Agents and Multi-Agents*. [S.l.], 2000. v. 355, p. 368.

POUTAKIDIS, D.; PADGHAM, L.; WINIKOFF, M. Debugging multi-agent systems using design artifacts: The case of interaction protocols. In: ACM. *Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 2*. [S.l.], 2002. p. 960–967.

RAMCHURN, S.D.; HUYNH, D.; JENNINGS, N.R. Trust in multi-agent systems. *The Knowledge Engineering Review*, Cambridge Univ Press, v. 19, n. 01, p. 1–25, 2005.

ROSENFELD, L.; MORVILLE, P. *Information architecture for the World Wide Web.* [S.l.: s.n.], 1998.

RUSSEL, S.J.; NORVIG, P. *Artificial intelligence*. [S.l.]: Prentice-Hall, 2003.

RUSSELL, S.J. et al. *Artificial intelligence: a modern approach*. [S.l.]: Prentice hall Englewood Cliffs, NJ, 1995.

SHAH, N. et al. Exception diagnosis in open multi-agent systems. In: . [S.l.: s.n.], 2005. p. 483 – 486.

SHOHAM, Y. Agent-oriented programming. *Artificial intelligence*, Elsevier, v. 60, n. 1, p. 51–92, 1993.

SOLER, J. et al. Towards a real-time multi-agent system architecture. *COAS, AAMAS*, Citeseer, v. 2002, 2002.

SPECIFICATION, F.I.C.A. *Foundation for Intelligent Physical Agents*. 2000.

SYCARA, K.P. Multiagent systems. *AI magazine*, v. 19, n. 2, p. 79, 1998.

TAO, Cheng; HUANG, Shengguo. An extensible multi-agent based traffic simulation system. In: . [S.l.: s.n.], 2009. v. 3, p. 713 –716.

TARKOMA, S.; LAUKKANEN, M. Supporting software agents on small devices. In: ACM. *Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 2*. [S.l.], 2002. p. 565–566.

VERCOUTER, L. A fault-tolerant open MAS. In: ACM. *Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 2*. [S.l.], 2002. p. 671.

VLASSIS, N. A Concise Introduction to Multiagent Systems and Distributed Artificial Intelligence. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, Morgan & Claypool Publishers, v. 1, n. 1, p. 1–71, 2007.

W3C. *Client-Side Web Applications Requirements*. 2011. Online: acessado em 4-Julho-2011. Disponível em: <http://www.w3.org/TR/WAPF-REQ/>.

W3C. *Real-time Communication Between Browsers*. 2011. Online: acessado em 15-Julho-2011. Disponível em: <http://dev.w3.org/2011/webrtc/editor/webrtc.html>.

W3C. *A vocabulary and associated APIs for HTML and XHTML*. 2011. Online: acessado em 4-Julho-2011. Disponível em: <http://www.w3.org/TR/html5/>.

WAZLAWICK, R.S. Metodologia de Pesquisa para Ciência da Computação. *Campus, primeira edição*, v. 1, 2009.

WEBKIT. *The WebKit Open Source Project*. 2011. Online: acessado em 6-Julho-2011. Disponível em: <http://www.webkit.org/>.

WIKIPEDIA. *Comparison of web browser engines — Wikipedia, The Free Encyclopedia*. 2011. Online: acessado em 6-Julho-2011. Disponível em: <http://en.wikipedia.org/w/index.php?title=Comparison_of_web_browser_engines&oldid=423880504>.

WOOLDRIDGE, M.J. *Reasoning about rational agents*. [S.l.]: The MIT Press, 2000.

WOOLDRIDGE, M.; JENNINGS, N.R. Intelligent agent: theory and practice. *KNOWLEDGE ENGINEERING REVIEW*, CAMBRIDGE UNIVERSITY PRESS, v. 10, p. 115–152, 1995.

XIAO, Liang et al. An adaptive security model for multi-agent systems and application to a clinical trials environment. In: . [S.l.: s.n.], 2007. v. 2, p. 261 –268. ISSN 0730-3157.

YEE, B. et al. Native client: A sandbox for portable, untrusted x86 native code. In: IEEE. *2009 30th IEEE Symposium on Security and Privacy*. [S.l.], 2009. p. 79–93.

ZAMBONELLI, F.; PARUNAK, H. Van Dyke. Towards a paradigm change in computer science and software engineering: a synthesis. *The Knowledge Engineering Review*, Cambridge Univ Press, v. 18, n. 04, p. 329–342, 2004.

ZHAO, Chenguang et al. An agent based wrapper mechanism used in system integration. In: . [S.l.: s.n.], 2008. p. 637 –640.