2
Fundamentos

Este capítulo descreve os três principais fundamentos utilizados pela abordagem: tecnologia de agentes, *frameworks* de *middleware* e arquiteturas orientadas a serviços. A Seção 2.1 apresenta os conceitos básicos de agentes de software e apresenta tipos de SMA's, propriedades de agência, modelos de estrutura organizacional e perspectivas e classificação de agentes. A Seção 2.2 fornece uma visão geral da tecnologia de *middleware*, mostrando os elementos, as características e as propriedades presentes em *frameworks* de *middleware*. A Seção 2.3 ilustra os fundamentos básicos utilizados em arquiteturas orientadas a serviços, explorando a arquitetura de referência WSA e serviços Web.

2.1
Tecnologia de Agentes

A utilização da tecnologia de agentes em sistemas distribuídos deriva da idéia de que agilidade, flexibilidade, inteligência e desempenho podem ser sensivelmente melhorados quando aplicados à: (i) construção de sistemas distribuídos ao invés de sistemas centralizados; (ii) adoção de soluções emergentes ao invés de soluções totalmente planejadas, e (iii) utilização de sistemas de execução concorrente, ao invés de execução seqüencial. Um sistema baseado em agentes é formado por entidades computacionais autônomas que possuem capacidades e objetivos individuais, e podem ser agrupados para atuar cooperativamente visando atingir os objetivos de um sistema.

2.1.1
Agentes de Software

Não existe uma definição universalmente aceita para o termo agente de software. Wooldridge [114] explica que parte da dificuldade de definir agente surge do fato que, para diferentes domínios de aplicação, as propriedades associadas ao conceito de agente assumem diferentes níveis de importância. Desta

forma, podem existir muitos tipos de agentes de software com características diferentes tais como mobilidade, autonomia, colaboração, persistência e inteligência. O comportamento dos agentes depende e é afetado pelas suas propriedades. Baseado em estudos anteriores realizados por Kendall [66], OMG (Object Management Group) [88], Garcia et al. [49] descrevem as seguintes propriedades para agentes de software:

1. Interação: um agente comunica-se com o ambiente e com outros agentes por meio de *sensors* e *effectors*.
2. Adaptação: um agente deve adaptar seu estado e comportamento de acordo com as condições ambientais.
3. Autonomia: um agente tem seu próprio *thread* de controle e pode aceitar ou recusar uma solicitação. Entende-se por autonomia a capacidade do agente executar suas atividades sem intervenção humana.
4. Aprendizado: um agente pode aprender baseado em experiências anteriores, quando interagindo com o seu ambiente.
5. Mobilidade: um agente deve estar habilitado a poder transferir-se de um ambiente para outro para atingir suas metas.
6. Colaboração: um agente pode cooperar com outros agentes de forma a atingir suas metas e as metas do sistema.

De acordo com a OMG [88], autonomia, interação e adaptação podem ser consideradas propriedades fundamentais dos agentes de software, enquanto que aprendizado, mobilidade e colaboração não são condições necessárias nem suficientes para caracterizar agentes. Existe um consenso na literatura de que a propriedade chave de um agente é a autonomia. Os agentes devem ser, pelo menos em alguma extensão, independentes. Eles não são inteiramente pré-programados, mas podem tomar decisões baseadas em informações de outros agentes e do ambiente.

Na visão de Ferber [38], um agente pode ser definido a partir das propriedades de agenciamento como uma entidade física ou virtual que possui as seguintes propriedades e habilidades:

- é capaz de atuar em um ambiente;
- pode comunicar-se com outros agentes;
- é dirigido por um conjunto de tendências na forma de objetivos individuais;
- possui seus próprios recursos;
- é capaz de perceber seu ambiente (com uma extensão limitada);
- tem somente uma representação parcial do ambiente;
- possui habilidades e pode oferecer serviços;
- pode estar habilitado a se replicar;
- o comportamento é dirigido de forma a satisfazer seus objetivos, considerando a quantidade de recursos e as habilidades que ele tem disponível, e depende da sua percepção e das mensagens que recebe.

Pela definição, os agentes podem atuar – não somente raciocinar – e o efeito das ações no ambiente pode afetar decisões futuras.

2.1.2 Perspectivas de Agentes

Existem diferentes caminhos e formas de organizar SMA's. A maneira mais adequada depende dos propósitos e objetivos do sistema. Conseqüentemente, existem muitos tipos de SMAs, cada qual com suas particularidades, como capacidade social, raciocínio, interoperabilidade e assim por diante. Jennings [64] propôs uma estrutura para analisar e classificar atividades de sistemas multi-agentes de acordo com duas perspectivas diferentes: perspectiva do agente e perspectiva do grupo. A perspectiva do agente focaliza nas características do agente, tais como arquitetura interna, estrutura e manutenção do conhecimento e habilidades de raciocínio e aprendizado do agente. A perspectiva do grupo reúne os aspectos do grupo, tais como organização, coordenação, cooperação, interação e negociação, que serão detalhados na seção 2.1.4.

Na visão de Ferber [38], SMA's podem ser definidos a partir de três pontos de vista: (i) o do ponto de vista da IAD (Inteligência Artificial Distribuída); (ii) do

ponto de vista da Internet e (iii) do ponto de vista de Vida Artificial[1]. Pesquisadores da área de IAD consideram que inteligência não pode existir isoladamente; eles acreditam que a inteligência é, de alguma forma, fundamentalmente social. Agentes são considerados 'criaturas' individuais vivendo em um ambiente com o qual interagem de alguma maneira. Os sistemas de IAD usualmente consistem de um pequeno grupo de agentes especialistas, que cooperam para resolver problemas, o que é similar à forma como especialistas humanos trabalham juntos. Tais agentes são autônomos em maior ou menor extensão, e supõe-se que eles tenham pelo menos alguma espécie de inteligência.

O exemplo clássico de arquitetura IAD [38, 97] é o chamado sistema Blackboard, desenvolvido na década de 1970 em um sistema chamado Hearsay, que foi utilizado para o reconhecimento de voz. No modelo *blackboard*, os agentes são chamados fontes de conhecimento e o *blackboard* é uma memória global, acessível a todos os agentes. Ele contém o estado corrente do problema e as ações executadas pelos agentes gradualmente modificam a estrutura de dados no *blackboard*. O *blackboard* é uma metáfora: agentes se comunicam uns com os outros "escrevendo e lendo no *blackboard*", como os humanos fariam em uma seção de *brainstorming*.

A outra comunidade cujos membros estão interessados em agentes é a comunidade da Internet. Em aplicações para Internet, os agentes existem em ambientes de software mantidos por servidores em *hosts;* a comunicação entre agentes é de vital importância para o compartilhamento de serviços distribuídos. A maioria dos agentes para Internet tem mecanismos para troca de mensagens, com maior ou menor grau de sofisticação. A propriedade de mobilidade é interessante para que os agentes possam se locomover de *host* para *host*.

A outra espécie de SMA deriva do mundo da Artificial Life (Alife). Alife é o nome dado a uma nova disciplina que estuda a vida natural para recriar fenômenos biológicos dentro de computadores e outros meios artificiais. Alife trata não somente da construção e simulação de sistemas vivos, mas também da pesquisa e construção de robôs adaptativos/autônomos e construção de estruturas sociais imitando organizações humanas adaptativas [Adam2004]. Os agentes

[1] O conceito de Artificial Life (alive) visa entender e modelar sistemas que possuem vida, isto é, capazes de sobreviver, adaptar-se e reproduzir em situações algumas vezes hostis [Ferber1999].

interagem com o ambiente e aprendem através desta interação, levando a uma forma emergente de comportamento adaptativo e autônomo [62, 63].

2.1.3 Classificação de Agentes

Existe na literatura um número de diferentes critérios para classificar os agentes. Segundo Jennings [64], para que possam agir de maneira autônoma, agentes devem ter várias habilidades: percepção, raciocínio baseado em crenças, tomada de decisão, planejamento, e habilidade para executar planos, incluindo passagem de mensagens. Jennings categoriza os agentes quanto ao nível de capacidade de resolução de problemas nos seguintes tipos:

1. *Reativos*: reagem a alterações no ambiente ou a mensagens de outros agentes. Não têm capacidade de raciocínio sobre suas intenções, reagindo tão somente sobre regras e planos estereotipados. Suas ações podem ser atualizar a base de fatos e enviar mensagens para outros agentes ou para o ambiente.
2. *Intencionais*[2]: têm habilidade de raciocínio sobre suas intenções e crenças, e criar e executar planos de ações. Eles têm capacidade de raciocinar sobre os planos, criar planos agendando ações, detectar conflitos entre planos e inclusive revisar planos.
3. *Sociais*: agentes intencionais são considerados sociais quando possuem modelos de outros agentes sobre os quais raciocinam para tomar decisões. Deriva do conceito de que sistemas multi-agentes podem ser organizados para simular o comportamento de estruturas sociais e organizacionais usadas no mundo real.

Os tipos de agente também podem ser classificados com base em sua capacidade cognitiva [51]. Um *agente cognitivo* [38] reage no ambiente usando o conhecimento com base no histórico do agente e incorpora estratégias de adaptação sofisticadas (Seção 3.4.2) ou técnicas de aprendizagem automáticas. Um *agente reativo* [38] é um agente que reage no ambiente sem qualquer

[2] Intencional também é apresentado com outras denominações, tais como Cognitivo, Racional e Deliberativo.

conhecimento prévio de sua história. Se dois eventos do mesmo tipo forem recebidos pelo agente reativo em momentos diferentes, será executado um comportamento idêntico. Os agentes reativos só têm autonomia de execução; eles não têm autonomia de decisão ou autonomia pró-ativa. Um *agente híbrido* é um agente reativo e cognitivo [38].

Existem ainda vários outros tipos de taxonomias e critérios para a classificação de agentes, tais como Nwana [86] que propôs uma topologia de agentes[3] composta por quatro dimensões: mobilidade (agentes estáticos ou móveis); comportamento (agentes deliberativos ou reativos); atributos primários (agentes que possuem características como autonomia, aprendizado e colaboração) e papéis (agentes de controle e agentes de informação). Na visão de Franklin e Gaesser [47], agentes autônomos podem ser divididos em três grandes grupos: agentes biológicos, agentes robôs e agentes computacionais. A classificação de Franklin e Gaesser tem sido utilizada como base para taxonomias de tipos de agentes em muitas abordagens. Klusch [67] criou uma taxonomia para *Information Agents* que estende o sub-grupo *Task Specific Agents* de agentes computacionais da classificação de Franklin e Gaesser.

2.1.4 Modelos e Estruturas Organizacionais

Jennings et al. [64] define o termo sistema multi-agente como “uma rede fracamente acoplada de resolvedores de problemas que trabalham juntos para resolver problemas que estão além das capacidades individuais ou conhecimento de cada um dos resolvedores de problemas”. Para grande parte das abordagens existentes, sistemas multi-agentes são formados por agentes que se agrupam em organizações.

A metodologia Gaia [114] foi uma das primeiras a definir um modelo geral e completo de organização especialmente concebido para análise e projeto de SMAs. Na visão de Wooldridge, sistemas baseados em agentes são vistos como “sociedades artificiais” ou organizações similares às existentes no mundo real. Assim, SMAs são compostos de múltiplas entidades autônomas (os agentes) interagindo e executando um papéis específicos. Da mesma forma que no mundo

[3] Termo utilizado pelo autor.

real existem empresas com funcionários possuidores de diferentes habilidades, e que, utilizando estas habilidades desenvolvem parte das atividades necessárias ao processo organizacional, é possível compor sociedades de agentes onde cada agente executa um determinado papel. Um papel define um conjunto de responsabilidades, representadas por um conjunto de serviços requeridos pela estratégia da solução e os direitos que são os recursos que o agente têm disponível para cumprir suas responsabilidades.

Seguindo as características da metodologia Gaia, o modelo organizacional proposto por Zambonelli [119] leva à caracterização de um SMA conforme apresentado na Figura 2 [119]. Embora um SMA mais simples possa ser visto como uma organização única, em grande parte dos casos a complexidade pode levar a fragmentação do sistema em um conjunto de organizações. Note que os agentes não se comunicam diretamente uns com os outros. As conexões são estabelecidas em um meio de interação que define, para um alto nível de abstração, como os agentes influenciam uns aos outros negociando através de protocolos de interação serviços providos e serviços requeridos.

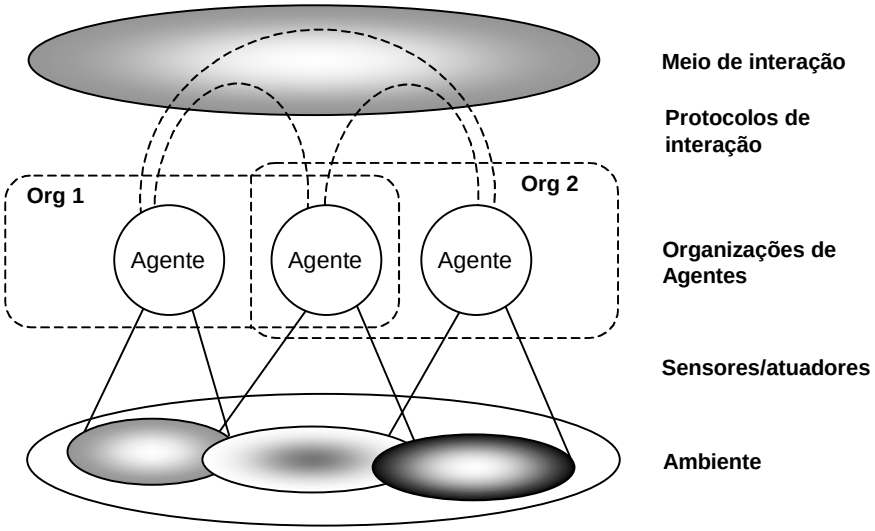

Figura 2 - Visão de um Sistema Multi-Agente

Na visão de Yu [118], um SMA pode ser descrito como uma rede fracamente acoplada de “resolvedores” de problemas que trabalham juntos para solucionar problemas que estão além das capacidades individuais ou conhecimento de cada resolvedor. De maneira similar à metodologia Gaia, Yu propõe modelar o SMA como um conjunto de papéis relacionados, definidos a

partir da teoria dos papéis [11]. O modelo de papéis tem sido utilizado por inúmeras abordagens [114, 35, 118, 100] para fornecer uma descrição abstrata de agentes de software. Um papel possui um conjunto de serviços e atividades que são disponibilizados através de uma interface externa e um conjunto de colaboradores, que são os outros papéis que interagem com ele. De acordo com a teoria dos papéis, o modelo é descrito através de dois elementos básicos: (i) *obrigações* são as responsabilidades do papel, e indicam a funcionalidade; e (ii) *permissões* são os direitos associados com o papel, que indicam os recursos que podem ser explorados.

Na metodologia MESSAGE [35], a arquitetura do SMA é definida através de um modelo organizacional, focada na estrutura da organização e no relacionamento entre os agentes que ela contém. O modelo organizacional também descreve mecanismos de resolução de conflitos e normas que habilitam o grupo de agentes a funcionar como uma unidade servindo a um propósito comum. Os agentes são identificados a partir de um modelo orientado a metas, onde as metas da organização são decompostas e associadas a tarefas. A decomposição de metas é feita de forma recursiva, até que a tarefa associada à meta possa ser executada integralmente por um agente isolado ou em colaboração com outros agentes. A metodologia utiliza a abordagem BDI (Believes, Desires, Intentions) para a modelagem da estrutura de aprendizado e conhecimento.

De acordo com Ferber [38], os vários elementos de uma organização artificial podem ser obtidos através da análise funcional e análise estrutural. O estágio de análise funcional ocorre após o contexto ser definido, ou seja, o que o sistema faz e os objetivos. Nesta fase, é preciso declarar a opção e fazer a escolha entre agentes cognitivos e reativos. Outra questão na análise funcional é determinar as restrições impostas, assim como os recursos disponíveis para os agentes poderem executar as tarefas. É durante o estágio de análise estrutural que a forma desejada para a organização é selecionada. Ferber considera os seguintes tipos de estruturas organizacionais:

1. *Organização fixa, hierárquica, com estrutura pré-definida*: utilizada em sistemas onde não existe grande número de agentes e normalmente os agentes são especialistas, sabem onde obter informações e para onde

enviar os resultados. Os papéis e relacionamentos são estáticos e podem ser definidos antecipadamente.

2. *Organização variável, equalitária, com estrutura emergente*: são características de sistemas populados por agentes reativos que se auto-organizam. Um agente pode se beneficiar das habilidades que encontra em outros agentes. O problema neste tipo de organização consiste em tornar os agentes capazes de cooperar.
3. *Organização variável, equalitária, com estrutura pré-definida*: esta estrutura é típica de organizações baseadas em requisições para provedores caracterizada por relacionamentos de conhecimento com alto grau de variabilidade.
4. *Organização evolucionária*: organizações evolucionárias não são objeto de muita atenção no contexto de sistemas multi-agentes.

Na visão de Silva et al. [100], uma sociedade de agente é composta não somente de agentes, mas de organizações, papéis, ambientes e objetos. Objetos são considerados elementos passivos que tem controle sobre seus estados e podem modificar seus estados durante o ciclo de vida, mas não tem controle sobre o seu comportamento. Agentes são entidades orientadas a *metas*, que executam planos e ações de forma a atingirem suas *metas*. Da mesma forma que os agentes, organizações são entidades autônomas e adaptativas que possuem *metas*, crenças, planos e ações. O termo organização é usado para representar grupos de entidades, tais como departamentos, comunidades e sociedades. Uma organização é considerada uma entidade de primeira ordem, e também define os seus papéis, que são desempenhados por sub-organizações e agentes.

2.1.5 Comunicação

Comunicação é definida por Russel e Norvig [97] como a troca intencional de informações, oriunda da produção e percepção de sinais emitidos através de um sistema de sinais convencionados. Ferber [58] distingue categorias de comunicação para SMAs baseado no perfil clássico, que classifica os meios de comunicação de acordo com o relacionamento expresso entre os vários agentes

envolvidos em um ato de comunicação. Estes relacionamentos se reportam às seguintes categorias de comunicação: (i) *sender-addressee link*; (ii) *nature of medium* e (iii) *intention to communicate*.

A categoria de comunicação *sender-addressee* (remetente-destinatário) pode ser explicada a partir da questão “qual é o relacionamento que liga o recipiente de uma comunicação ao seu remetente?". Quando o endereço é conhecido pelo remetente, é dito que o modelo de comunicação é feito no modo ponto-a-ponto. Nesta forma de comunicação, é possível enviar mensagens na forma *broadcasting* (para todos os agentes), utilizando um *mediator*.

Na categoria *nature of medium* podem ser identificados três diferentes mecanismos de roteamento de mensagens:

1. *direct routing*: é o modo mais simples. Quando um agente quer enviar uma mensagem, a mensagem é capturada pelo canal de comunicação, que encaminha a mensagem diretamente ao destinatário, ou para todos os potenciais destinatários, no caso de comunicação *broadcasting*.
2. *routing by signal propagation*: é característica do modo de comunicação de agentes reativos. Um agente envia um sinal, que é *broadcast* dentro do ambiente e cuja intensidade decresce à medida que a distância aumenta.
3. *public notice routing*: é tipicamente um mecanismo de comunicação para *´small ads*’ [38]. Quando um agente deseja se comunicar, ele coloca sua mensagem em um espaço comum chamado *noticeboard* que é visível para todos os agentes ou para aqueles de um grupo específico. Esta modalidade também é muito utilizada em arquiteturas do tipo *blackboard*, onde fontes de conhecimento são ativadas por eventos acontecendo no *blackboard*.

O quesito *intention to communicate* procura responder à questão “a ação de comunicar-se está vinculada a uma intenção por parte do remetente ou é uma ação incidental, um processo independente do desejo do remetente”? Estas são as duas formas básicas de comunicação que existe na natureza.

2.2 Middleware

Middleware é uma tecnologia de software projetada para ajudar a gerenciar a complexidade e heterogeneidade inerente a sistemas distribuídos. Um *middleware* se situa entre vários sistemas operacionais e uma plataforma de programação distribuída, e provê abstrações de alto nível que facilitam o entendimento da estrutura. *Frameworks* de *middleware* provêem serviços de infraestrutura, tornando possível aos desenvolvedores abstrair funcionalidades complexas, tais como gerenciamento, concorrência, heterogeneidade de plataforma, de linguagem e de rede. Alguns conceitos e classificação de *middleware* são apresentados a seguir.

2.2.1 Definição de Middleware

Existem variadas definições sobre o termo *middleware* [19, 105, 4, 82, 27], induzidas pelas variações de serviços que um *middleware* pode prover. A definição mais simples poderia seguir a acepção da palavra, ou seja, "*middleware* é alguma coisa que está no meio". Entretanto, este meio pode variar, e muito, dependendo do que é contextualizado. Tentando especializar um pouco mais a definição do que está no meio, Bakken [4] define *middleware* como "uma classe de tecnologia de software projetada para ajudar a gerenciar a complexidade e heterogeneidade inerente em sistemas distribuídos". Um *middleware* atua como uma camada de software acima do sistema operacional, mas abaixo do programa de aplicação que provê a API (Application Programming Interface) de programação comum.

A tecnologia de *middleware* tornou-se disponível no início de 1990, e desde então aconteceram três grandes mudanças [19]: a introdução e o surgimento de Java, a introdução de *middlewares* de componentes transacionais (TCM), e a mais importante de todas, o surgimento da Internet. Todas as três mudaram a forma de projeto e implementação de *frameworks* de *middleware*. Produtos de *middleware* isolados, tais como *message queuing systems*, *transaction processing monitor*, e *concentrators* estão aos poucos desaparecendo. Ao invés disto, estão emergindo

servidores especializados que combinam funções tais como gerenciamento da aplicação, de transações, de serviços e outras funções [105].

Tecnologias tradicionais de computação distribuída tais como CORBA (Common Object Request Broker Architecture), RMI (Remote Method Invocation) e DCOM (Distributed Component Object Model) diferem em sua arquitetura básica e abordagem, mas utilizam comumente um modelo de comunicação síncrono baseado em um mecanismo fortemente acoplado. Todas estas tecnologias estão baseadas em protocolos de comunicação ponto-a-ponto. De acordo com Nagappan [82], estas tecnologias têm sido aplicadas com sucesso para integrar aplicações dentro de um contexto de rede local que pode operar remotamente. Como a Internet tornou-se a solução lógica que espalha e conecta os limites de negócios, é preciso haver interoperabilidade de aplicações sobre a rede. Entretanto, estas tecnologias não são muito amigáveis e apropriadas para serem acessadas sobre a Internet.

De acordo com Curry [26], MOM pode ser definido como “qualquer infraestrutura de *middleware* que provê capacidades para manipular mensagens”. Um MOM é uma arquitetura de computação distribuída baseada em modelo de comunicação fracamente acoplado, onde a aplicação cliente não precisa conhecer os seus recipientes de aplicação ou os seus argumentos de métodos. MOM habilita aplicações a se comunicarem indiretamente, usando filas de mensagens. A aplicação cliente envia mensagens para a fila de mensagens e a aplicação destino captura a mensagem da fila. Neste modelo de operação, a aplicação que envia mensagens para a outra aplicação pode continuar a operar sem esperar a resposta da outra aplicação. Os principais benefícios do MOM emanam do seu modelo de comunicação assíncrono, o uso de filas de mensagens, capacidade transacional e QoS (Quality-of-service) superior aos tradicionais modelos RPC [82].

SOA (*Service Oriented Architecture*) é uma das mais recentes evoluções da computação distribuída que habilita componentes de software, objetos e processos de diferentes sistemas serem expostos como serviços. Recentemente, todas as grandes companhias tais como IBM, Microsoft, Oracle, Sun e SAP estão direcionando fortemente o desenvolvimento de software para SOA [108]. De acordo com Booth [14], um SOA “é um conjunto de componentes fracamente acoplados que podem ser invocados e cuja descrição de interface pode ser publicada e descoberta”.

2.2.2
Frameworks do tipo Middleware

Frameworks do tipo *middleware* são projetados para abstrair parte da heterogeneidade (como ambientes operacionais, linguagens de programação, componentes e ferramentas) com a qual programadores de sistemas distribuídos precisam lidar. De acordo com Fayad [37], *frameworks* de integração de *middleware* são comumente usados para integrar aplicações distribuídas e componentes. Eles são projetados para melhorar a capacidade de desenvolvedores de software de modularizar, reusar e estender sua infraestrutura de software para trabalhar mais facilmente em ambientes distribuídos. Em princípio, os objetos no sistema podem ser implementados com o uso de diferentes linguagens de programação, podem ser executados em diferentes plataformas e seus nomes não precisam ser conhecidos por todos os outros objetos do sistema.

Frameworks do tipo *middleware* sempre encobrem heterogeneidade de rede e de hardware, e a maioria também abstrai heterogeneidade de sistema operacional e de linguagem de programação. Além das duas básicas, de uma maneira geral abstrações de programação oferecidas por *middlewares* podem prover transparência com respeito à distribuição de uma ou mais das seguintes dimensões [4]: localização, concorrência, replicações, falhas e mobilidade. O uso de *middlewares* permite desacoplar componentes que implementam funcionalidades de mais alto nível daqueles componentes que tratam com o nível mais baixo das aplicações, como, por exemplo, protocolos e APIs, que são complexos e cujo ciclo de vida é menor quando comparado ao tempo de vida de muitas aplicações distribuídas.

O conceito de *container* leve ou *lighweight container* é relativamente novo e tem sido utilizado na literatura [81, 9] para caracterizar elementos de *frameworks* de *middleware* voltados para o desenvolvimento de aplicações de negócios e sistemas multi-agentes. De acordo com Machado [72], muito se aprendeu utilizando os *containers* mais pesados como o EJB (Enteprise Java Beans). O acúmulo de funcionalidades nem sempre necessárias para a aplicação é sem dúvida o maior problema dos *containers* pesados. Nestes, o *deployment* é complexo, o processo de configuração mais ainda e o desempenho nem sempre é adequado.

Uma nova estratégia, a partir da qual um container de *middleware* deve resolver apenas os problemas básicos, tem sido utilizada no mercado. Os problemas básicos compreendem (i) o gerenciamento do ciclo de vida dos objetos manipulados; (ii) mecanismos para localização de componentes e (iii) inicialização rápida. Antes de incorporar funcionalidades pesadas na sua estrutura, tais como persistência e segurança, os containers leves provêem meios para "colar" nele o código específico de uma aplicação e integrar centenas de *frameworks open-source* que podem efetuar outras funcionalidades, tais como acesso a dados, camada de apresentação, segurança e outras.

A Figura 3 ilustra o papel de um *framework* de *middleware* baseado no conceito de container leve. Nesta configuração, cada *framework* especialista fica responsável por uma camada do sistema. O *middleware* caracteriza a camada intermediária onde são desenvolvidos os agentes e outras entidades que compõem a camada lógica de negócios.

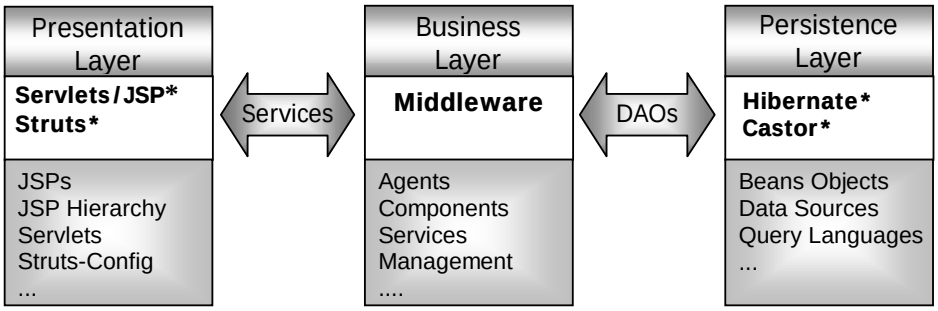

Figura 3 - Camadas representando o domínio

Frameworks tais como Struts[4] da Apache – que utiliza a tecnologia de JSP-Servlets da Sun Microsystems – podem ser utilizados para implementar as funcionalidades de interface gráfica. Da mesma forma, existem frameworks especializados em persistência, tais como o Hibernate[5] e o Castor[6], que podem ser utilizados para efetuar mapeamento objeto-relacional. Nesta estrutura, a comunicação entre as camadas é feita através de interfaces, como serviços (Services) e objetos de acesso a dados (DAOs).

[4] Struts é uma ferramenta da Apache para camada de apresentação que utiliza tecnologia padrão de Servlets e implementa uma arquitetura baseada no modelo MVC, provendo o Model e View.
[5] Hibernate é um *framework* para mapeamento objeto-relacional – http://www.hibernate.org/
[6] Castor é uma ferramenta para mapeamento objeto-relacional – http://www.castor.org/

Um dos mais conhecidos *containers leves* é o *framework* Spring [111]. Ele faz parte do pacote de soluções da Sun Microsystems e é composto por um conjunto de módulos tais como O/R Mapping, JDBC, Application Context, Web Context, dentre outros, para facilitar o desenvolvimento de aplicações e serviços. No contexto de sistemas multi-agentes, existe um número de *frameworks* de *middleware*, tais como JADE [9], AgentScape [89], Cougaar [115], JAF [110], CAG [43], dentre outros, que provêem infraestrutura para suportar o desenvolvimento de SMAs[7].

Juntamente com *frameworks*, agentes de *middleware* têm sido citados na literatura [103, 44, 121] como entidades que auxiliam as tarefas de gerenciamento, descoberta de serviços e atualização de serviços em *frameworks* de *middleware*. Flores-Mendez classifica os agentes de *middleware* nos seguintes tipos: (i) *brokers*, que recebem requisições e executam ações usando serviços de outros agentes; (ii) *blackboards*, que provêem um repositório compartilhado de dados e um meio de comunicação assíncrona entre agentes e (iii) *facilitators*, que são agentes encarregados de descobrir serviços e entidades responsáveis pela execução do serviço.

2.3 Arquitetura de Referência WSA

WSA (*Web Services Architecture*), a serem descritos nas seções a seguir, são instâncias de SOA, cujos serviços estão em servidores Web.WSA tem por função prover uma definição comum de serviço Web e definir seu lugar dentro de um *framework* de espectro mais amplo para guiar implementadores de serviços, autores de especificação de serviços e desenvolvedores de aplicações Web [14]. Enquanto serviços Web provêem um meio padrão para interoperabilidade entre aplicações distribuídas que podem ser executadas em uma variedade de plataformas, WSA provê um modelo para o entendimento de serviços Web e promove interoperabilidade através da definição de protocolos compatíveis.

A arquitetura de referência é composta por cinco modelos [13], que definem diferentes visões de serviços Web, como mostra a Figura 4 [121].

[7] Alguns destes frameworks para desenvolvimento de SMAs serão apresentados em detalhe no capítulo 4

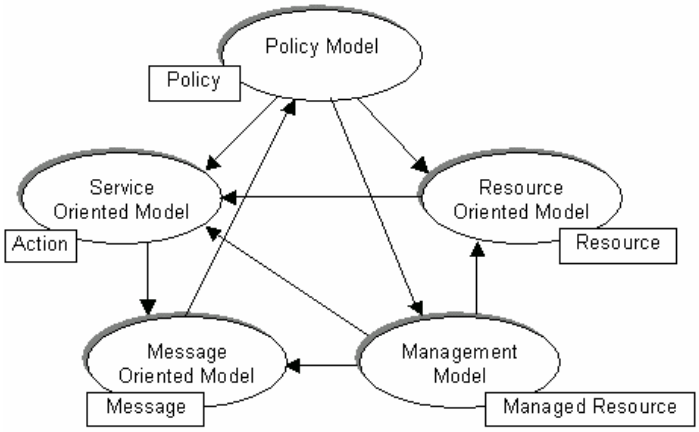

Figura 4 - Modelos de Referência de WSA

WSA é uma arquitetura que se baseia em uma estrutura fracamente acoplada, para interoperar em um ambiente de computação distribuída. Os modelos podem ser descritos como segue:

(a) *Message-Oriented Model* (MOM): define como agentes de serviços Web podem interagir com outros usando um modelo de comunicação orientado a mensagens;

(b) *Service-Oriented Model* (SOM): construído sobre o MOM para incluir conceitos de serviços e ações que são executadas por requisitantes e provedores de serviços;

(c) *Resource-Oriented Model* (ROM): construído sobre SOM para incluir aspectos relacionados aos recursos e serviços associados à manipulação de recursos;

(d) *Management Model* (MGM): focaliza o gerenciamento e ciclo de vida de recursos;

(e) *Policy Model*: define o núcleo dos conceitos para descrever as políticas de qualidade e segurança dos serviços Web.

WSA descreve ambas, as características mínimas que são comuns a todos os serviços Web e um número de características que são desejáveis para alguns, porém não necessárias para todos os serviços Web. Intuitivamente WSA pressupõe a utilização da tecnologia de agentes ao definir um serviço como “uma noção abstrata que precisa ser implementada por um agente concreto” [13]. O

agente é a entidade física (um pedaço do software) que envia e recebe mensagens, enquanto o serviço é um conjunto abstrato de funcionalidades providas pelo agente. Para ilustrar esta distinção, seria possível implementar um mesmo serviço utilizando um agente num determinado dia e um outro agente no dia seguinte escrito em uma linguagem de programação diferente. Apesar do agente ter mudado, o serviço permanece o mesmo. WSA provê aos desenvolvedores modelos de referência que são compreensíveis e úteis, e captura abstrações que representam diferentes perspectivas da arquitetura.

2.3.1
Modelo Orientado a Mensagens

O modelo orientado a mensagens (MOM) focaliza os aspectos da arquitetura relacionados à estrutura das mensagens e ao seu transporte. Neste modelo não existe qualquer interesse no significado semântico do conteúdo de uma mensagem ou nos relacionamentos com outras mensagens. O MOM focaliza a estrutura das mensagens, o relacionamento entre remetente e destinatário e também a forma como as mensagens são transmitidas.

A Figura 5 [14] ilustra os principais conceitos envolvidos no MOM e os seus relacionamentos. Um endereço (*Address*) é aquela informação requerida pelo mecanismo de transporte de mensagem de forma a entregar a mensagem apropriadamente. Tipicamente, a forma do endereço depende do transporte de mensagem (*Transport*) que está sendo utilizado. No caso de transporte de mensagens HTTP, a informação de endereço toma o formato de uma URL.

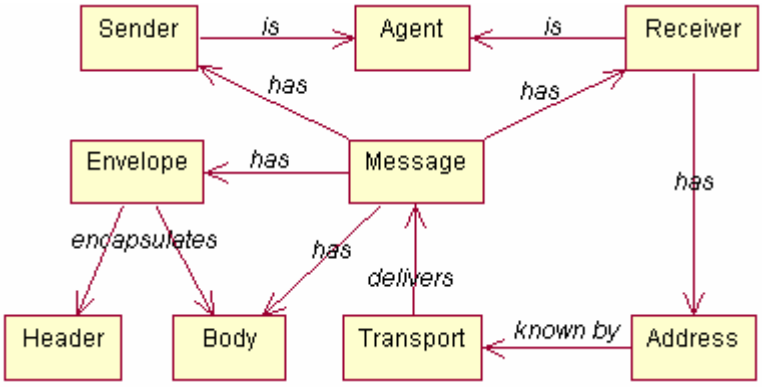

Figura 5 - Principais conceitos do MOM e os seus relacionamentos

O envelope (*Envelope*) é a estrutura que encapsula ar partes que compõem uma mensagem: o cabeçalho (*Header*) e o corpo (*Body*) da mensagem. O corpo da mensagem é a parte que contém os detalhes específicos da aplicação que o *sender* deseja endereçar e pode ser expressa como parte da descrição do serviço. O cabeçalho representa informações independentes do corpo da mensagem que podem ser processadas separadamente, como por exemplo, mecanismos de autenticação e segurança.

Outros conceitos são [14]: (i) *sender and receiver* são as possíveis entidades que tomam parte de uma mensagem; a mensagem é enviada de um remetente para um destinatário e entre um remetente e um destinatário pode existir zero ou mais intermediários; (ii) *message path* caracteriza uma seqüência de entidades (agentes ou componentes) que processam a mensagem; o caminho inicia do remetente original e termina no destinatário final (Seção 3.3).

2.3.2 Modelo Orientado a Serviços

Enquanto o MOM focaliza comunicação estritamente a partir de uma perspectiva de transporte de mensagens, não se preocupando em relacionar mensagens e serviços, o SOM estende o MOM de forma a explicitar os conceitos fundamentais envolvidos com serviços. Um serviço [14] é um recurso abstrato que representa uma capacidade de executar uma funcionalidade coerente do ponto de vista da entidade provedora e da entidade requisitante. Em uma arquitetura orientada a serviços, um serviço é formalmente definido em termos de provedores e requisitantes, e não em termos das propriedades de uma entidade em si.

Os principais conceitos e relacionamentos associados ao modelo de serviços podem ser vistos na Figura 6 [14]. Um serviço é descrito por um meta-dado processável por máquina e possui um conjunto de atributos relacionados. Os atributos representados através do relacionamento *has* caracterizam os atributos fundamentais de um serviço. Um serviço possui uma interface (*Interface*), um identificador, (*Identifier*) uma descrição e uma semântica (*Semantic*). Uma interface é uma fronteira abstrata que um serviço expõe; ela define os tipos de mensagens e os padrões para troca de mensagens que estão envolvidos na

interação com o serviço, junto com algumas condições implícitas pelas mensagens.

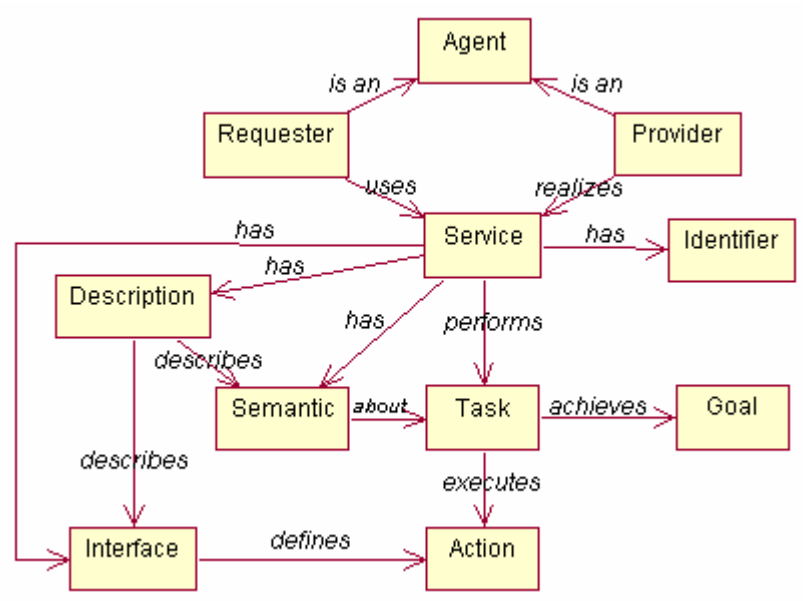

Figura 6 - Principais conceitos do SOM e seus relacionamentos

A descrição do serviço contém os detalhes da interface, e potencialmente o comportamento esperado do serviço. Estas interfaces são públicas, descritas e expostas em XML e são conhecidas por todos os agentes. Um serviço pode desempenhar uma ou mais tarefas (*Task*), e uma tarefa é executada por um conjunto de ações (*Action*), que podem ser representadas por funções ou operações em um programa.

2.3.3 Modelo Orientado a Recursos

O modelo orientado a recursos focaliza os aspectos da arquitetura relacionados à manipulação dos recursos existentes na plataforma. Recurso é o conceito fundamental por trás de serviços Web, já que um serviço é um recurso. Sendo um serviço uma representação abstrata, ele necessita de um modelo de recursos para fornecer uma representação concreta. O ROM focaliza as características chave que são relevantes para o conceito de recursos, independente do papel que o recurso desempenha no contexto de serviços Web. A Figura 7 [14] ilustra os conceitos básicos e relacionamentos no ROM.

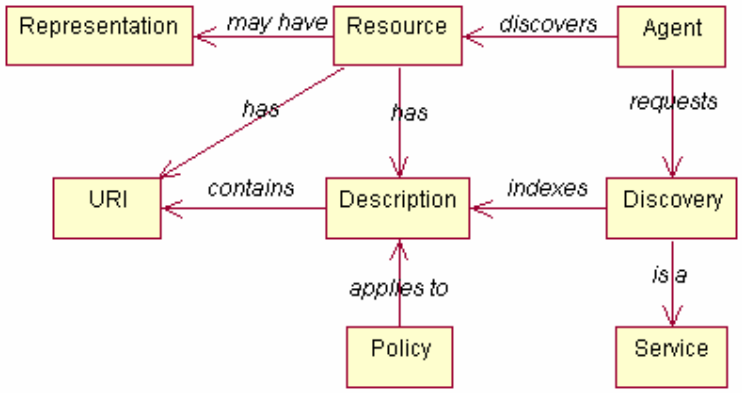

Figura 7 - Principais conceitos do ROM e seus relacionamentos

Um recurso possui um identificador no formato de uma URIs (*Uniform Resource Identifier*), que provê um meio simples e extensível de identificação. Um recurso possui uma representação (*Representation*), que é um conjunto de objetos de dados que refletem o estado do recurso. Os protocolos Web (HTTP, FTP, SOAP, SMTP) são baseados na troca de mensagens, e uma representação é usualmente recuperada executando um HTTP GET em uma URI. A descrição (*Description*) de um recurso é uma representação de meta-dados processável por máquina que permite que o recurso possa ser descoberto por um agente (*Agent*) de software ou agente humano. O ato de localizar uma descrição de um recurso relacionado a um serviço é denominado de descoberta (*Discovery*) e envolve combinar um conjunto de critérios funcionais, tais como a descrição dos recursos.

O atributo *Policy* representa as restrições aplicadas ao comportamento do agente quando ele executa uma ação ou acessa recursos. Políticas se aplicam para definir propriedades de segurança, propriedades de qualidade de serviço e de gerenciamento. Uma descrição de política é uma representação processável por máquina de uma política ou de um conjunto de políticas.

2.3.4
Modelo de Gerenciamento

O modelo de gerenciamento foi construído sobre a política de gerenciamento de serviços Web. As atividades de gerenciamento envolvem um conjunto de atividades que habilitam o gerenciamento do ciclo de vida da

plataforma e monitoramento e *reporting* de QoS. De acordo com Booth [14], um serviço Web torna-se gerenciável quando ele expõe um conjunto de operações que suportam capacidade de gerenciamento. A Figura 8 [14] mostra o modelo de políticas de gerenciamento, que define as capacidades de gerenciamento do ciclo de vida, gerenciamento através de monitoramento e gerenciamento das capacidades que definem a qualidade do serviço.

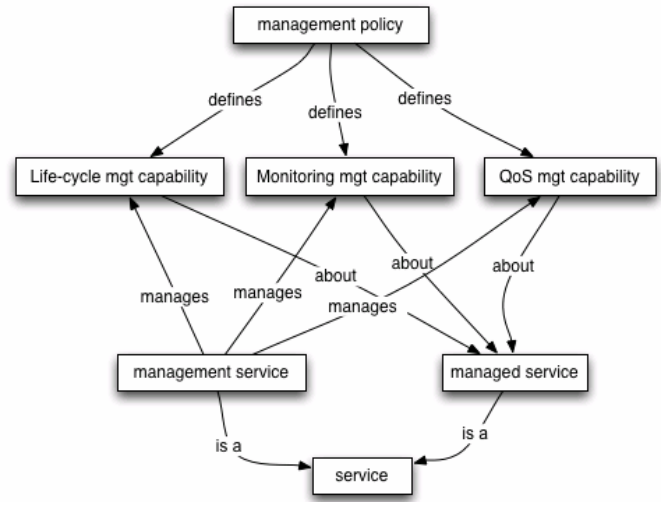

Figura 8 - Principais conceitos do modelo de gerenciamento de políticas

Conforme pode ser observado na figura, as atividades de gerenciamento derivam da política de gerenciamento e envolvem funções de gerenciamento do ciclo de vida (*Life-cycle*), monitoramento (*Monitoring*) e gerenciamento de QoS. Informações sobre a qualidade dos serviços incluem contadores de requisições e respostas, tempo de início e tempo fim de transações, estados do ciclo de vida de um serviço, quantidade de exceções e erros ocorridos por período de tempo.

2.4
Serviços Web e Interfaces Orientadas a Serviços

De acordo com Gartner [53], “serviços Web são componentes de software fracamente acoplados, atuando sobre tecnologias padrão de Internet”. Essencialmente, os componentes chave de uma SOA são as mensagens que são trocadas, os agentes que agem como requisitantes/provedores do serviço, e os mecanismos compartilhados de transporte que suportam o fluxo de mensagens. Um serviço é uma unidade de trabalho desempenhada por um provedor de

serviços para atingir os resultados desejados por potenciais consumidores do serviço. Em WSA ambos – provedor e consumidor – são papéis desempenhados por agentes de software em nome dos seus proprietários.

Baixo acoplamento é um dos princípios chave em SOA. De acordo com [13, 17], baixo acoplamento pode ser alcançado através de duas restrições:

(i) Um pequeno conjunto de relações simples e ubíquas à todos os agentes participando do sistema. Somente a semântica genérica é codificada nas relações.

(ii) Utilização de mensagens descritivas confinadas por um esquema extensível (normalmente XML) acessível através de interfaces. Nenhum, ou somente um comportamento mínimo do sistema é prescrito por mensagens. Um esquema extensível limita o vocabulário e a estrutura das mensagens permitindo que novas versões do serviço possam ser introduzidas sem quebrar os serviços existentes.

Considerando que em SOA existem apenas poucas interfaces disponíveis no sistema, é preciso expressar a semântica específica da aplicação em mensagens. Mas existem certas restrições inerentes a SOA quanto ao tipo de mensagem para a interface. A primeira delas é que as mensagens precisam ser descritivas, antes que instrutivas, já que o provedor do serviço deve ser responsável por resolver este problema. Segundo, os provedores dos serviços somente estarão habilitados a entender as mensagens se elas estiverem descritas em um formato, estrutura e vocabulário que seja compreensível por todas as partes. Limitar o vocabulário e estrutura de mensagens é uma necessidade para uma comunicação eficiente. Quanto mais restrita for uma mensagem, mais fácil será entender a mensagem. Terceiro, SOA precisa ter mecanismos que habilitem um consumidor a descobrir um provedor de serviços.

A literatura reconhece que WSDL possui uma limitação óbvia: a sua incapacidade de descrever serviços complexos de negócios, que tipicamente são formados por composição serviços de granularidade fina. Serviços Web normalmente possuem granularidade mais grossa do que objetos e componentes porque as funções que precisam ser expostas devem possuir muito mais lógica de aplicação do que aquelas apresentadas normalmente por um único objeto ou

componente [46]. A Figura 9 a seguir [46] ilustra a forma para alcançar integração entre serviços Web e serviços de negócios.

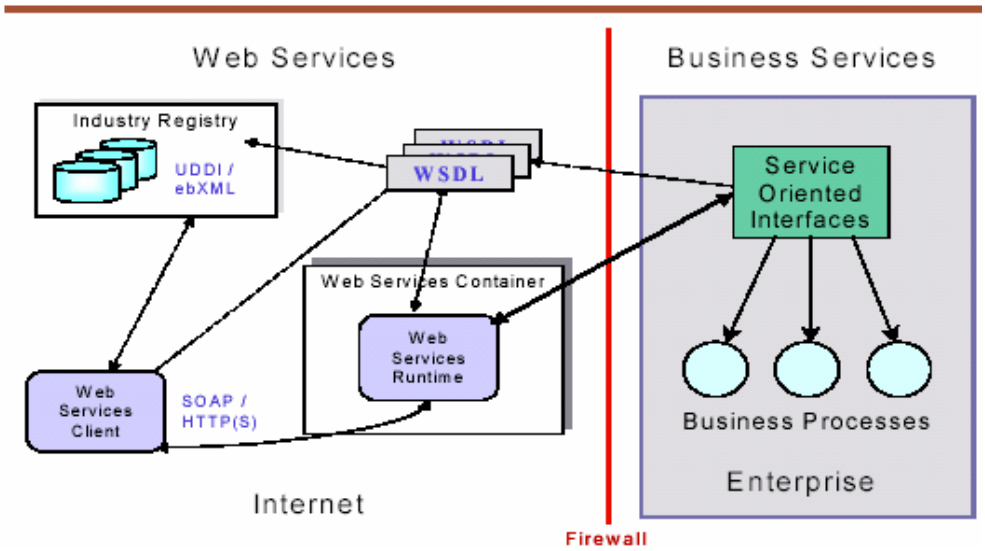

Figura 9 - Integração entre serviços Web e serviços de negócios

Na visão de Frankel, a maneira de alcançar integração entre serviços Web e serviços de negócios é utilizar WSDL apenas como uma ponte entre os requisitantes de serviços na Internet e as aplicações de negócios. A figura mostra como os serviços Web na partição *Web Services* são integrados com os serviços de negócios na partição *Business Services* via WSDL e convertidos para o formato de interfaces orientadas a serviços nas aplicações de negócios. No contexto das aplicações de negócios, os serviços são acessados através das interfaces orientadas a serviços, que obedecem a sintaxe estabelecida pelo protocolo nativo da aplicação.

Mesmo sendo uma tecnologia emergente, existem hoje inúmeras soluções e *frameworks* para sistemas multi-agentes que utilizam os conceitos SOA. Dentre estes, podem ser citados AgentScape [89], IASM [108] e Cougaar [115]. Apesar da arquitetura de JADE não ser orientada a serviços, ela utiliza fortemente os conceitos de serviços e prove mecanismos para registro e descoberta de serviços. Atualmente, JADE provê uma extensão à plataforma para a integração com serviços Web chamada JADE WSIG (Web Services Integration Gateway) [57].

2.5 Discussão

As propriedades de agenciamento (Seção 2.1) oferecem um meio importante para avaliar as potencialidades de um agente, tornando possível identificar as propriedades que eles estão aptos a atender, quais ele não atende e porque. As abstrações fornecidas por metodologias para SMA's, tais como o conceito de organização, modelo abstrato de papéis, *blackboard*, dentre outros, oferecem meios eficientes para reduzir a complexidade do desenvolvimento.

Frameworks do tipo *middleware* (Seção 2.2) são projetados para gerenciar a complexidade e as heterogeneidades inerentes a sistema distribuídos, provendo um conjunto de serviços de infraestrutura que podem reduzir drasticamente a complexidade e os esforços de programadores para trabalhar com sistemas distribuídos. Os serviços de infraestrutura são desempenhados por agentes de *middleware* dinâmicos e pró-ativos, capazes de adaptar de forma colaborativa e dinâmica o seu comportamento e o comportamento da arquitetura.

A arquitetura do MIDAS adere à arquitetura de referência WSA (Seção 2.3), que oferece um conjunto de modelos compreensíveis e úteis. Os modelos capturam abstrações que representam diferentes perspectivas da arquitetura, e facilitam o entendimento da arquitetura, diminuindo a curva de aprendizado do *framework*.

A integração de SMA's com os serviços Web (Seção 2.4) é uma necessidade frente aos novos desafios de aplicações para a Internet. Todavia, a integração pode não ser tão simples: muitas abordagens se fundamentam em modelos fortemente acoplados e unilateralmente dirigidos a mensagens.