2 Conceitos Básicos

Neste capítulo serão apresentados, de maneira sucinta, alguns conceitos básicos relevantes a este trabalho. Em primeiro lugar, apresentar-se-á a tecnologia de agentes de software distinguindo-se as diferentes propriedades incorporadas por ela. Em seguida será discutido o conceito de espaços de tuplas. Finalmente será apresentada a tecnologia de reflexão computacional e sua utilização em arquiteturas em meta-nível.

2.1 Agentes de Software

A tecnologia de agentes de software pode ser vista como uma abordagem complementar ao paradigma de objetos no desenvolvimento de sistemas de software complexos. Esta nova tecnologia propõe um nível de abstração mais alto que aquele proposto pela tecnologia de objetos uma vez que o comportamento dos agentes situa-se mais próximo do comportamento de seres humanos cujo trabalho em geral é substituído ou suportado por sistemas de software [63]. Além disso, os sistemas atuais têm como objetivo lidar com ambientes complexos e de comportamento imprevisível como, por exemplo, a Internet. Agentes de software são por definição adaptativos [42] e mostram-se mais preparados que objetos para lidar com os sistemas atuais.

A definição precisa do que é realmente um agente de software é um dos temas mais debatidos e controversos na área de ciência da computação [63]. Um agente de software pode ser definido de uma maneira simples como um objeto complexo com atitude [5]. Mais especificamente, um agente pode ser encarado como uma entidade autônoma que possui uma série de capacidades e objetivos definidos, que interage com um ambiente e com outros agentes e é capaz, através destas interações, de adaptar seu estado e comportamento [42,61].

O estado de um agente pode ser definido como seu conhecimento [42] e é composto por componentes mentais como crenças, objetivos e preferências. As crenças de um agente modelam o ambiente externo com o qual o agente interage, enquanto seus objetivos podem ser atingidos através de diferentes planos. Para definir o plano a ser utilizado para atingir um determinado objetivo, um agente de software baseia-se em suas preferências e em suas crenças. Durante sua vida, ou período de execução, o agente é capaz de atualizar seu estado baseado em interações com o ambiente ou com outros agentes [6]. Um agente percebe mudanças em seu ambiente através de seus sensores e realiza alterações neste através de seus efetuadores.

O comportamento de agentes de software é composto e afetado pelas *propriedades intra-agente* que ele incorpora. Propriedades intra-agente são características que um agente pode ter e utilizar com o propósito de atingir seus objetivos. A Tabela 1 resume algumas das principais propriedades que um agente pode incorporar, baseando-se nas definições propostas em [42]. Em geral autonomia, interação e adaptação são consideradas como propriedades fundamentais para um agente. Entretanto as propriedades de aprendizagem, mobilidade e colaboração não são nem necessárias, nem suficientes para que uma entidade de software seja considerada um agente.

Existem diferentes tipos de agentes de software. Cada tipo de agente incorpora um conjunto de propriedades intra-agentes além de características específicas á aplicação da qual ele faz parte. Dentre os diferentes tipos de agentes podemos citar agentes móveis, agentes de interface e agentes de busca. Uma descrição mais detalhada sobre estes e outros tipos de agentes pode ser encontrada em [42,24,25].

Outro ponto importante, no que se refere a agentes de software, é a distinção entre agentes cognitivos e agentes reativos [13]. Os agentes cognitivos são os agentes que incorporam a propriedade de aprendizagem e que são capazes de aprender a partir de suas próprias experiências, a partir daí tomar suas decisões. Por outro lado, os agentes reativos são aqueles agentes que são programados para tomar decisões instintivas a partir de mudanças no ambiente, percebidas através de seus sensores. Estes agentes, por sua vez, possuem um grau de inteligência inferior ao dos agentes

Propriedade	Definição
Autonomia	Um agente é capaz de agir sem intervenção externa direta e é capaz de aceitar ou recusar uma requisição.
Interação	Um agente comunica-se com seu ambiente e com outros agentes através de sensores e efetuadores.
Adaptação	Um agente deve ser capaz de se adaptar e modificar seu estado de acordo com o estabelecimento de novas condições em seu ambiente.
Mobilidade	Um agente é capaz de se transportar de um ambiente para outro a fim de atingir seus objetivos.
Aprendizagem	Um agente é capaz de aprender baseado em experiências prévias adquiridas ao interagir com outros agentes e ao reagir a mudanças em seu ambiente.
Colaboração	Um agente é capaz de colaborar e negociar com outros agentes para atingir seus próprios objetivos e os objetivos de seu sistema.

Tabela 1 – Resumo das principais propriedades intra-agentes

cognitivos.

Um agente de software, em geral não é encontrado sozinho em uma aplicação ou sistema, mas em conjunto com outros agentes, de tipos iguais ou diferentes, formando uma sociedade ou organização. A esta sociedade dá-se o nome de sistema multi-agente. Em um sistema multi-agente, os agentes cooperam, negociam ou competem de forma a alcançar seus próprios objetivos, o objetivo de um determinado grupo ou o objetivo da sociedade como um todo.

Um sistema multi-agente pode então ser visto como um conjunto de agentes de software que trabalham até certo ponto de certa forma independente tentando atingir seus objetivos [63]. Mesmo trabalhando de forma independente, estes agentes terão a necessidade de interagir com outros agentes e com o seu ambiente a fim de adquirir novas informações, solicitar que serviços sejam executados ou coordenar suas atividades com as atividades de outros agentes. Desta forma um sistema multi-agente deve prover uma série de funcionalidades que possibilitem a interação dos agentes entre si e com o seu ambiente. A estas funcionalidades incorporadas pelo sistema multi-agente dá-se o nome de *propriedades inter-agentes* [63,18,57].

Dentre as principais propriedades inter-agentes podem ser citadas dentre outras a coordenação, a comunicação, a persistência e a mobilidade dos agentes[50]. Embora algumas destas propriedades sejam também consideradas propriedades intra-agentes, como por exemplo, a mobilidade, o próprio sistema deve incorporá-las de forma a simplificar o projeto e desenvolvimento de aplicações complexas baseadas em agentes. A seguir apresentaremos estas propriedades inter-agentes com maiores detalhes.

Coordenação. Alguns tipos de agentes colaboram, competem ou negociam a fim de atingir seus objetivos. De forma a garantir este comportamento cooperativo ou competitivo, o sistema multi-agente deve incorporar algum nível de coordenação entre as atividades dos agentes que o compõem. Coordenação é definida como o processo de gerenciar as dependências entre diferentes atividades [39]. Quando aplicado a sistemas multi-agentes, o conceito de coordenação está ligado a impedir trabalho redobrado dos agentes e a garantir que as ações dos diferentes agentes sejam sincronizadas [28]. Entretanto, é muito difícil garantir a coordenação global de um sistema multi-agente sem a implementação de um controle global explícito [28]. Desta

forma torna-se necessária a incorporação do suporte à coordenação no próprio sistema multi-agente.

Comunicação. Uma vez que os agentes de software interagem entre si, o sistema multi-agente deve prover uma interface para entre eles, incorporando a propriedade inter-agente de comunicação. Esta interface funciona como um protocolo e de comunicação entre os agentes, o que facilita a modelagem e construção de aplicações baseadas em agentes. Este protocolo de comunicação pode ser implementado através de diversos métodos, como por exemplo tratadores de mensagens [34], blackboards [28,45,47], espaços de tuplas [50,51,62,63] ou até chamada direta de métodos [46]. Qualquer que seja o método utilizado para a implementação da propriedade inter-agente de comunicação, deve estar disponível aos agentes uma interface de acesso a ela através, por exemplo, de uma API.

Mobilidade. Para sistemas multi-agentes cujos agentes necessitem de mobilidade (agentes móveis), prover um protocolo transparente no que se refere a esta propriedade intra-agente também deve ser uma funcionalidade disponível nestes sistemas, tornando-se também uma propriedade inter-agente. Embora sejam os próprios agentes que tomam a decisão de se transferir de um ambiente (contexto de execução) para outro, deve ser uma responsabilidade do sistema multi-agente garantir sua transferência, incluindo seu código e estado, de forma correta. A propriedade inter-agente de mobilidade também deve oferecer suporte ao registro de um agente recém chegado a um novo ambiente bem como o controle de acesso aos recursos disponibilizados pelo ambiente.

Persistência. Em um sistema multi-agente, um agente de software pode existir durante longos períodos de tempo. Durante este tempo ele pode estar ativo ou inativo. Enquanto inativo é desejado que o agente seja “colocado para dormir” tendo seu estado gravado de forma persistente [42]. Desta forma o estado do agente pode ser conservado e recuperado tão logo ele deseje tornar-se ativo novamente. Na verdade o agente deve ter autonomia para decidir quando deve se tornar persistente ou não. No entanto, o sistema multi-agente, incorporando a propriedade inter-agente de persistência, deve oferecer suporte ao armazenamento e recuperação do agente persistente, bem como a gerência deste estado.

2.2 Espaços de Tuplas

Um *blackboard* [7] pode ser visto como uma área de memória compartilhada que pode ser acessada por diferentes agentes. Os agentes são capazes de escrever, ler e apagar dados desta área, que pode então ser utilizada como uma interface de comunicação entre pos agentes. Além da comunicação explícita entre os agentes um blackboard pode ser utilizado para armazenar informações relativas ao ambiente onde o agente executa. O blackboard pode ser utilizado ainda como área de armazenamento de código e estado de agentes persistentes.

Existem implementações específicas de blackboards denominadas espaços de tuplas à la Linda [16], também denominados *blackboards associativos* ou simplesmente *espaços de tuplas*. Existem dois elementos básicos em um espaço de tuplas: as *Tuplas* e o próprio *Espaço*. Ao contrário de mensagens tradicionais, uma tupla é um objeto propriamente dito, composta de uma série de outros objetos. O espaço funciona como um repositório de tuplas e nele podem ser executadas diferentes operações como, por exemplo, leitura e escrita de tuplas.

Tuplas são lidas de um espaço através da utilização de buscas associativas onde coringas[2] podem ser utilizados. Por exemplo, a tupla (“mensagem” , “agente1” , “Olá!”) pode ser lida através de uma busca utilizando padrão (“mensagem” , “agente1” , **String**), onde o **String** representa um coringa para qualquer conjunto de caracteres. Espaços de tuplas devem no mínimo implementar as operações básicas de leitura, escrita e retirada de tuplas, mas dependendo da implementação outras operações derivadas destas podem existir, como por exemplo, leitura de um conjunto de tuplas ou o bloqueio do agente que deseja ler uma tupla que ainda não se encontra espaço.

Ao contrário da troca de mensagens tradicional entre agentes, a utilização de espaços de tuplas oferece um estilo de programação desacoplado. Isto pode ser dito porque uma vez que um agente cria e escreve uma tupla em um espaço ele não precisa saber especificamente qual outro agente a lerá nem em que momento isto irá acontecer [9]. Na verdade a existência de uma tupla em um espaço é independente do agente que a criou e dos agentes que possam vir a realizar alguma operação sobre ela.

Existem diversas implementações comerciais de espaços de tuplas. Dentre estas as duas mais conhecidas são IBM TSpaces [35] e Java Spaces [14]. Estas implementações oferecem suporte para as operações básicas de espaços de tuplas (leitura

[2] do inglês *wildcard*

escrita e retirada de tuplas), além de outras derivadas destas. Também é disponibilizado o controle de transações no acesso aos espaços, e a opção de manter espaços persistentes. Os espaços utilizados nestas implementações também podem ser acessados a partir de clientes remotos.

2.3 Reflexão Computacional

Na linguagem natural, o termo reflexão pode ser definido através de dois significados distintos. O primeiro está ligado à introspecção, isto é, o ato de examinar a si próprio. Já o segundo, está ligado ao redirecionamento da luz, por exemplo, por um espelho. No domínio da informática, o termo reflexão computacional está ligado à capacidade de um programa conhecer e examinar sua própria estrutura e de alterar seu comportamento através do redirecionamento ou interceptação de operações efetuadas. [8]. Na verdade, ambos os conceitos de introspecção e redirecionamento são somados na definição de reflexão computacional.

Os conceitos de introspecção e redirecionamento estão intimamente ligados uma vez que a capacidade de alteração do comportamento de um sistema depende da sua possibilidade de adquirir informações sobre este. Por outro lado, qualquer modificação no sistema é capaz de alterar sua estrutura e conseqüentemente deve refletir nas informações adquiridas sobre o sistema através da introspecção. Esta ligação é denominada conexão causal [38]. Um sistema é dito causalmente conectado à sua estrutura se qualquer modificação em um dos dois (sistema ou domínio) corresponder a modificações no outro. Desta forma um sistema causalmente conectado sempre possui uma representação correta de sua estrutura e pode causar modificações nesta através dos efeitos de sua execução.

Um outro conceito fundamental no que se refere à reflexão computacional é o de meta-informação. Este conceito pode ser encarado como a representação de toda informação contida e manipulável por um sistema computacional que seja referente a ele mesmo [48]. Neste contexto, uma arquitetura reflexiva é em geral definida como uma arquitetura de meta-nível [37]. Nestas arquiteturas o sistema é dividido em dois níveis, o nível base e o meta-nível. Enquanto o nível base contém os componentes responsáveis pelas funcionalidades básicas do sistema, conforme definido em seus requisitos funcionais, o meta-nível provê uma auto-representação do sistema de forma a prover conhecimento sobre sua estrutura e comportamento [7].

O protocolo de meta-objeto[3] ou MOP (do inglês *Meta-Object Protocol*) estabelece o relacionamento entre o nível base e o meta-nível. O MOP provê uma interface de acesso ao meta-nível acoplando informações a meta-informações. Esta meta-informação pode ser do tipo estrutural ou dinâmica. A meta-informação estrutural diz respeito aos dados que não são alterados durante a execução do programa, como por exemplo, classes, hierarquias de classes, tipos e estruturas de dados. Por outro lado a meta informação dinâmica se refere aos aspectos comportamentais do sistema, isto é, aqueles que podem ser alterados em tempo de execução. Dentre estes podemos citar valores de objetos ou dados e parâmetros passados em chamadas de métodos.

De maneira geral os MOPs possuem quatro aspectos básicos como apresentado abaixo [37]:

- **Ligação com o nível base**: este aspecto possui o propósito de estabelecer o vínculo entre componentes do nível base com entidades do meta-nível, de forma dinâmica ou estática. Um ponto importante é a cardinalidade deste vínculo, que pode ser estabelecido de forma que um meta-objeto controle um objeto de nível base, de forma que vários meta-objetos controlem um objeto de nível base ou de que um meta-objeto controle um grupo de objetos de nível base.
- **Reificação**: este aspecto é representado pela materialização de informações do nível base no meta-nível. Esta materialização está relacionada a características dinâmicas e estáticas do sistema como, por exemplo, o estado das entidades de nível base e mensagens trocadas entre elas.
- **Execução**: este aspecto está ligado à execução de operações no meta-nível e à possibilidade de requisição de serviços a entidades do nível base.

[3] Na verdade a utilização do termo protocolo de meta-objeto vem do fato da maioria das linguagens que incorporam reflexão computacional serem orientadas a objetos. O termo protocolo de meta-informação também poderia ser utilizado, mas para manter a coerência com trabalhos relacionados, neste trabalho será utilizado o conceito “protocolo de meta-objeto”.

- **Modificação**: este aspecto está ligado à capacidade do meta-nível controlar e até alterar a estrutura, comportamento e estado dos entidades do nível base.