4
Arquitetura da solução

Um dos principais problemas de engenharia enfrentados no projeto de sistemas auto–organizáveis é a análise e verificação do comportamento macroscópico desses sistemas. Uma abordagem prática e bem fundamentada para estudar o comportamento global do sistema é um pré–requisito para a sua implantação e aceitação (De Wolf, 2007).

Nesse contexto, este capítulo propõe um novo método para a verificação experimental de sistemas multiagentes auto–organizáveis. A seção 4.1 descreve as principais características desse método e estabelece, também, um breve comparativo entre ele, as técnicas formais e a abordagem de Kevrekidis (Kevrekidis, 2003). Então, a seção 4.2 detalha o *framework* que implementa o método de verificação proposto e descreve os passos básicos necessários para a sua instanciação.

4.1
Método

As técnicas de verificação de sistemas multiagentes auto–organizáveis podem ser divididas em duas categorias principais: **verificação formal** e **verificação empírica**.

Nas abordagens formais, o sistema alvo é modelado matematicamente, extraindo–se equações que descrevam o seu comportamento evolucionário – características macroscópicas – em função das suas características microscópicas. Nesses casos, a principal dificuldade é conseguir elaborar representações formais corretas e completas de sistemas altamente dinâmicos e complexos como os sistemas auto–organizáveis.

Já nas técnicas empíricas ou experimentais, a execução do sistema é simulada várias vezes em um ambiente controlado, a fim de se observar o seu comportamento. A princípio, essa abordagem é mais viável para a verificação de sistemas auto–organizáveis, pois exclui a necessidade de se modelar matematicamente o sistema em questão. Entretanto, ela geralmente exige várias simulações completas até que se consiga dados suficientes para ser possível deduzir tendências de comportamento desse sistema. Isso é um sério

problema, pois os sistemas auto–organizáveis são extremamente complexos e simular a sua execução várias vezes pode levar bastante tempo e consumir uma quantidade exagerada de recursos computacionais.

Recentemente, uma abordagem proposta por Kevrekidis (Kevrekidis, 2003) e avaliada por De Wolf em (De Wolf, 2007) buscou mesclar os pontos positivos das estratégias de verificação formal e empírica, pois, ao mesmo tempo em que procura eliminar a necessidade de se modelar matematicamente o sistema, também não obriga a execução completa e repetida de simulações experimentais.

Denominada "*equation-free macroscopic analysis*" – ou simplesmente "análise macroscópica livre de equações", em tradução livre –, essa técnica de verificação de sistemas multiagentes substitui a utilização dos modelos formais matemáticos por pequenas simulações experimentais no nível microscópico do sistema considerando diversas situações e parâmetros de entrada. Os resultados das simulações são, então, processados utilizando–se os mesmos algoritmos de análise macroscópica que são normalmente utilizados nas técnicas formais, mais especificamente na avaliação dos dados fornecidos pelas equações evolucionárias.

Apesar de ser uma abordagem de aplicação visivelmente mais simples (quando comparada aos métodos tradicionais), essa técnica ainda não resolve um problema: a avaliação dos resultados das simulações continua sendo realizada através de algoritmos de análise numérica. Além disso, existem restrições para que esse procedimento possa ser aplicado a um sistema, conforme destacado em 3.1.

Sendo assim, ainda existe a necessidade da elaboração de um novo método de verificação tão eficiente como os citados, mas mais facilmente aplicável nos domínios de problemas envolvendo sistemas multiagentes. O ideal é que ele não demande conhecimentos avançados de programação matemática para a análise dos resultados. Este trabalho apresenta um método que se enquadra exatamente nesse contexto.

4.1.1 Descrição geral

O método de verificação de sistemas multiagentes aqui proposto é uma derivação da abordagem apresentada por Kevrekidis (Kevrekidis, 2003). Ele adota o mesmo princípio de verificação experimental, por ser mais simples que a elaboração de equações evolucionárias. No entanto, através de uma abordagem autonômica, ele substitui o ajuste manual das simulações (que era feito através da avaliação humana dos resultados obtidos pelos algoritmos de

análise numérica) por um ajuste autonômico. Esse ajuste é realizado por meio de planejadores e algoritmos de busca heurística em tempo real que avaliam o estado da simulação frente a um estado objetivo pré-definido. Com isso, apenas uma simulação automatizada completa é suficiente para a verificação do sistema, que ocorre sem a necessidade da supervisão humana (ao contrário da maioria das outras técnicas de verificação aplicáveis a sistemas multiagentes).

A decisão de se incorporar planejadores ao processo de verificação experimental de sistemas multiagentes vem da constatação de que eles são excelentes ferramentas para verificar as propriedades globais de sistemas auto–organizáveis e indicar como decisões locais perturbam o sistema afetando o seu estado global (Gatti, 2006). Se um planejador tem conhecimento do estado inicial do ambiente e também dos objetivos que devem cumpridos, ele é capaz de encontrar uma sequência de ações que levem o ambiente à um dos vários estados finais possíveis.

Entretanto, para este domínio de problema específico, é inviável que o planejador conheça todos os estados possíveis do ambiente. Em sistemas não determinístico – como grande parte dos modelos de interação dos sistemas multiagentes auto–organizáveis – o número de estados é infinito. Por esse motivo, na solução proposta, optou–se pela utilização de planejadores em tempo real, onde os estados são conhecidos somente no momento em que são explorados. Sendo assim, apenas parte dos estados serão armazenados até se chegar a uma solução para o problema, o que é o ideal para o contexto abordado.

A figura 4.1 esboça a estrutura do método proposto, evidenciando as suas características autonômicas, no que diz respeito à sua capacidade de ajustar autonomicamente a simulação do sistema com a finalidade de fazê-la convergir para um determinado estado que verifique a viabilidade do sistema (estado objetivo). De forma geral, a execução desse método se resume às seguintes etapas:

1. O módulo **monitor**, por meio dos sensores, percebe a execução de uma ação por um determinado agente no nível microscópico do sistema;

2. Os impactos dessa ação sobre o ambiente em um nível macroscópico são, então, avaliados pelo módulo de **análise**;

3. Nesse ponto, o fluxo de execução se divide:

   (a) Caso os impactos da ação executada sejam positivos, ou seja, aproximem o sistema de seu estado objetivo, nada é feito e a iteração se encerra;

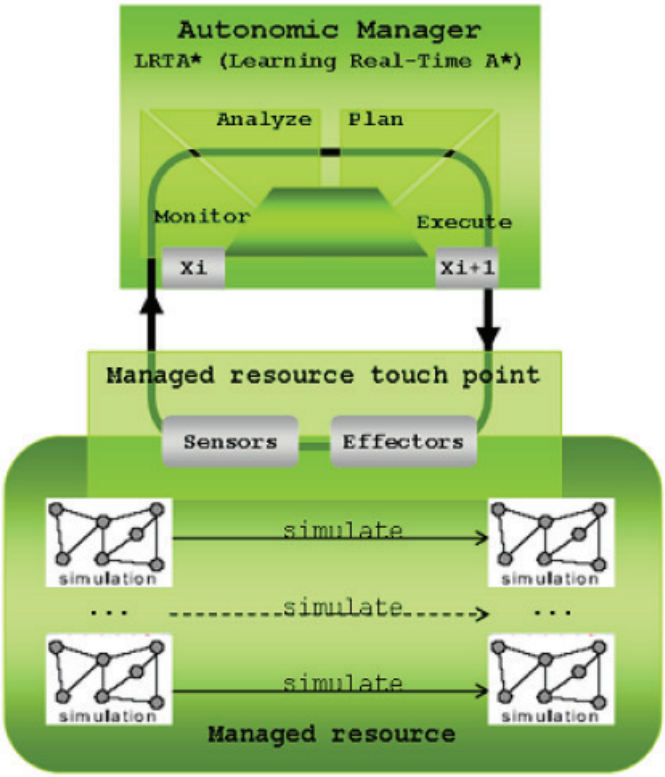

Figura 4.1: Esboço do método de verificação autonômico

(b) Caso contrário, o módulo **planejador** é acionado e elabora um plano de ações que será aplicado pelo módulo **executor**, por meio dos efetuadores, com o objetivo de reverter o ambiente ao seu estado anterior.

Em todas as iterações durante o seu funcionamento o gerenciador mantém atualizada uma base de conhecimento que ajuda no processo de tomada de decisão, isto é, na constatação de quando uma ação contribui ou não para convergir o sistema para o seu estado ótimo.

O papel dos algoritmos de busca heurística em tempo real (como o LRTA* (Korf, 1990)) no cumprimento dessas etapas é essencial. É através deles que o módulo de análise verifica se o estado do ambiente após a execução da ação percebida é realmente um estado ótimo local ou se existem outras ações mais apropriadas para serem executadas naquele momento específico.

É importante ressaltar que, para o correto funcionamento do método proposto, é imprescindível que cada ação monitorada possua uma ou mais ações reversas que possam ser executadas quando for necessário. Caso essas ações não sejam simétricas o método poderá **não funcionar** corretamente, uma vez que a tentativa de reverter o ambiente a um estado anterior estará sujeita a falhas.

4.2 Framework

O método de verificação experimental aqui proposto foi implementado através de um *framework* que exerce o papel do gerenciador autonômico do processo de verificação, conforme descrito no inicio deste capítulo. Como um dos objetivos deste trabalho era obter um ambiente unificado para observar e verificar a execução de sistemas multiagentes, esse *framework* foi desenvolvido através da modificação e melhoria do MASOF (*Multi-Environment Self-Organizing Framework*) (Gatti, 2009).

O MASOF é, na verdade, uma evolução do MASON (Luke, 2005), um ambiente para simulação de sistemas auto–organizáveis desenvolvido em Java. O propósito do MASOF é adicionar ao MASON o conceito de hierarquia de ambientes em sistemas auto–organizáveis, isto é, possibilitar a existência de múltiplos ambientes em uma única simulação.

Esse ambiente conjunto possibilita a simulação discreta de sistemas multiagentes, oferecendo recursos auxiliares que apoiam o processo de observação dessas simulações, tais como visualização em duas e três dimensões, gravação de vídeos, construção de gráficos e relatórios, entre outros. Seu funcionamento é bastante simples: a cada passo da simulação, percorre-se a lista dos agentes (e subambientes) existentes no ambiente principal da simulação, fazendo com que cada um deles execute um passo do seu comportamento projetado.

Entretanto, durante a proposta e execução inicial deste trabalho, verificou–se que, apesar de o MASOF introduzir um conceito útil e necessário para a engenharia de sistemas multiagentes auto–organizáveis, a sua modelagem e principalmente a sua implementação continham falhas e inconsistências. Por esse motivo, ele foi completamente refatorado, mantendo os princípios originais de sua proposta, e codificado obedecendo aos principais padrões de qualidade de software.

Essa nova versão do *framework* MASOF, no contexto deste trabalho, foi denominada **subsistema central** de verificação. Já a tarefa de verificação propriamente dita é realizada pelo **subsistema de verificação**. Juntos, esses subsistemas formam uma solução completa voltada para a observar e verificar a execução de sistemas multiagentes auto–organizáveis.

4.2.1 Subsistema central

O subsistema central (ou núcleo), derivado originalmente do *framework* MASOF (Gatti, 2009), é a parte do sistema de verificação responsável por possibilitar o projeto e o desenvolvimento de sistemas multiagentes auto–

organizáveis de forma clara e direta, através da definição explícita dos conceitos de **agente**, **ambiente**, **evento**, etc. Além disso, conforme proposta original do MASOF, esse subsistema implementa a possibilidade da existência de múltiplos ambientes em uma única simulação, o que é extremamente útil para se modelar processos complexos onde muitas vezes ambientes internos possuem uma importância significativa (por exemplo, sistemas celulares, onde cada célula é um sub–ambiente complexo de uma colônia celular (Gatti, 2006, Soares, 2008)).

A figura 4.2 detalha o relacionamento entre as classes que compõem o subsistema central do *framework* de verificação. As classes destacadas em vermelho são pontos flexíveis (classes abstratas ou interfaces) que devem ser obrigatoriamente estendidos durante a instanciação do *framework*:

- **Entidade:** A classe abstrata *Entity* representa um elemento que existe dentro de um ambiente, mas não necessariamente atua sobre ele. Ou seja, uma entidade é um elemento inativo, que possui uma localização dentro do ambiente e que é incapaz de perceber ou produzir alterações (eventos) no ambiente. Na maioria dos casos não será necessário instanciar essa classe diretamente, mas através da criação de agentes e ambientes;
- **Agente:** A classe abstrata *Agent* implementa um agente conforme definição conceitual da seção 2.2.1 deste trabalho. Um agente é também uma entidade, mas possui um grande diferencial: ele observa e atua sobre o ambiente (captando e produzindo eventos), sempre com o propósito de atingir os seus objetivos de projeto. Ele ainda é capaz de se comunicar com outros agentes, seja direta (enviando mensagens) ou indiretamente (modificando o ambiente). Além disso, um agente considerado móvel também pode se locomover dentro do ambiente ou entre ambientes diferentes, alterando a sua localização;
- **Ambiente e Localização:** Um ambiente (classe abstrata *Environment*) é um agente que representa um espaço dividido em localizações (interface *Location*). Cada localização pode armazenar uma outra entidade – inclusive um outro ambiente, o que possibilita a construção de subambientes em vários níveis. Um ambiente ainda controla a propagação de eventos e ações no seu espaço, notificando todos os agentes, os agentes em uma dada localização ou apenas um agente específico, conforme necessário;
- **Simulação:** A classe abstrata *Simulation*, como o próprio nome diz, representa a simulação propriamente dita e centraliza todo o controle sobre a execução da mesma. Ela encapsula um ambiente principal, sendo capaz de dizer em qual estado ele se encontra. Também faz parte das

suas atribuições fornecer um identificar único sobre o estado atual da simulação, requisito indispensável para que o verificador possa controlar o espaço de busca de estados da simulação;

– **Evento:** Um evento (classe *Event*) é um acontecimento disparado por um agente, podendo ser fruto da execução do seu comportamento ou em resposta à um outro evento recebido. Os tipos de evento (enumeração *EventType*) mais comuns são:

  – **Emissão (*EMISSION*):** Este tipo define todo e qualquer evento enviado em *broadcast*, seja para todos os agentes do ambiente ou apenas para os agentes em uma determinada localização;
  – **Gatilho (*TRIGGER*):** São os eventos enviados em resposta à um outro evento com o propósito de informar ao remetente do evento original que o mesmo foi recebido e processado;
  – **Movimentação (*MOVEMENT*):** Eventos enviados para notificar a locomoção (mudança de localização) do agente remetente;
  – **Reação (*REACTION*):** Este tipo de evento é utilizado para informar que o agente remetente reagiu à um outro evento;
  – **Comunicação (*COMMUNICATION*):** Define todos os eventos de comunicação direta entre agentes.

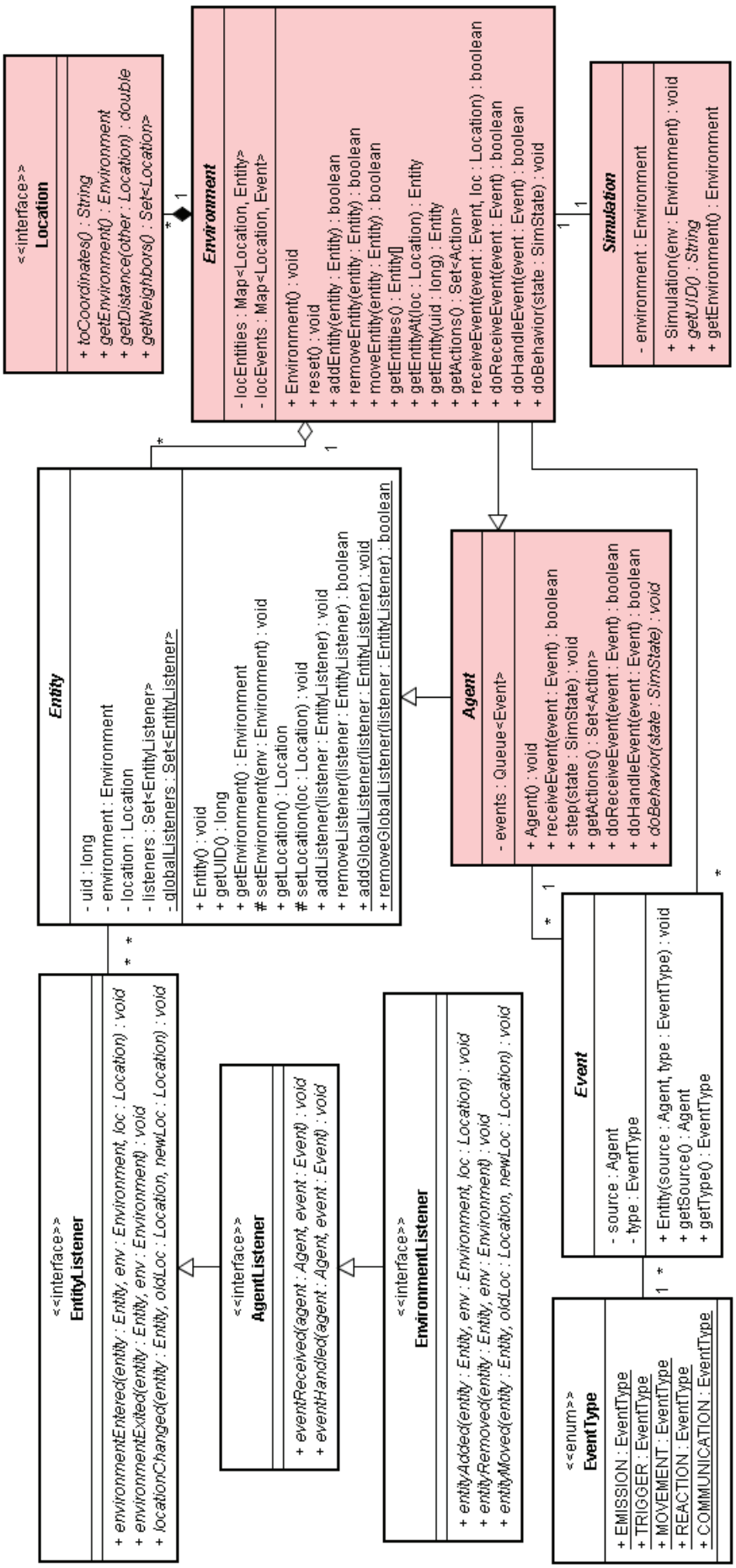

Figura 4.2: Diagrama de classes: Subsistema central

4.2.2 Subsistema de verificação

O subsistema de verificação (ou verificador) estende o subsistema central e implementa o método de verificação experimental descrito na seção 4.1, definindo os conceitos de **gerenciador autonômico**, **ação**, **objetivo** e todos os demais recursos necessários para a observação do espaço de busca dos estados da simulação.

A figura 4.3 detalha o relacionamento entre as classes que compõem o subsistema de verificação do *framework*. A linha tracejada indica a fronteira do sistema. As dependências para com o subsistema central (seção 4.2.1) foram omitidas para tornar o diagrama mais legível. As classes destacadas em vermelho são pontos flexíveis (classes abstratas ou interfaces) que devem ser obrigatoriamente estendidos durante a instanciação do *framework*:

- **Ação:** Uma ação (classe *Action*) é um evento específico que possui um agente fonte (executor da ação) e um agente de destino (receptor da ação). Toda ação deve, obrigatoriamente, possuir um conjunto de ações simétricas reversas que serão executadas caso seja necessário desfazê-la. Opcionalmente, pode–se também definir um custo para cada ação. Neste caso, o verificador buscará o estado final que tenha o menor custo envolvido;
- **Objetivo:** A interface *Goal* define um objetivo, isto é, algo que deve ser satisfeito para que o sistema seja válido. Um conjunto de objetivos (classe *GoalSet*) é, também, um único objetivo, e pode ser utilizado quando é necessário definir mais de um objetivo na instanciação do *framework*. Os objetivos formam o principal pilar do processo de verificação, pois o sistema só será considerado válido se todos os seus objetivos forem atingidos;
- **Estratégia de busca:** Quando uma ação é executada, é necessário verificar se ela é realmente a melhor ação para aquele momento específico. Simplificando, é preciso responder à seguinte questão: existe uma ação mais apropriada para ser executada neste momento? Essa resposta é obtida através da estratégia de busca que implementa a interface *SearchStrategy* (padrão de projeto *Strategy*) em conjunto com o gerenciador autonômico. Apesar de ser um ponto flexível do *framework*, sua instanciação não é obrigatória. Por padrão, é utilizada uma estratégia de busca derivada do algoritmo LRTA* (*Learning Real-Time A**) (Korf, 1990).
- **Espaço de busca:** O espaço de busca pesquisado é controlado através de um dicionário mantido no gerenciador autonômico. Esse dicionário in-

dexa identificadores únicos (*UID*) dos estados da simulação às instâncias da classe *SearchNode*, onde cada instância representa um estado explorado. A criação de estados de busca é realizada através de uma fábrica (interface *SearchNodeFactory*, padrão *Factory*), o que adiciona suporte à criação de estados personalizados. O subsistema de verificação já inclui duas classes de estado de busca que se diferem pela forma de armazenamento: *MemorySearchNode*, cuja informação do estado é armazenada na memória principal e *PersistableSearchNode*, onde essa informação é guardada em disco (arquivo);

– **Gerenciador autonômico:** O gerenciador autonômico (classe *Manager*) é o principal componente do processo de verificação. É ele quem unifica todos os recursos auxiliares detalhados acima para controlar e verificar o ambiente. É importante ressaltar que ele também mantém um rígido controle sobre todo o espaço de busca pesquisado para que o processo de verificação nunca entre em *looping*;

O gerenciador autonômico irá coexistir com os outros agentes dentro do ambiente, de tal forma que ele seja capaz de perceber ações internas e externas através do seu módulo monitor. Essas ações devem ser definidas durante o processo de instanciação do *framework*. Além disso, somente as ações registradas junto ao gerenciador autonômico serão monitoradas. Isso é especialmente útil para situações onde deseja–se que ações simples sejam ignoradas no processo de verificação. Entretanto, deve–se considerar registrar toda e qualquer ação que tenha o potencial para modificar o ambiente.

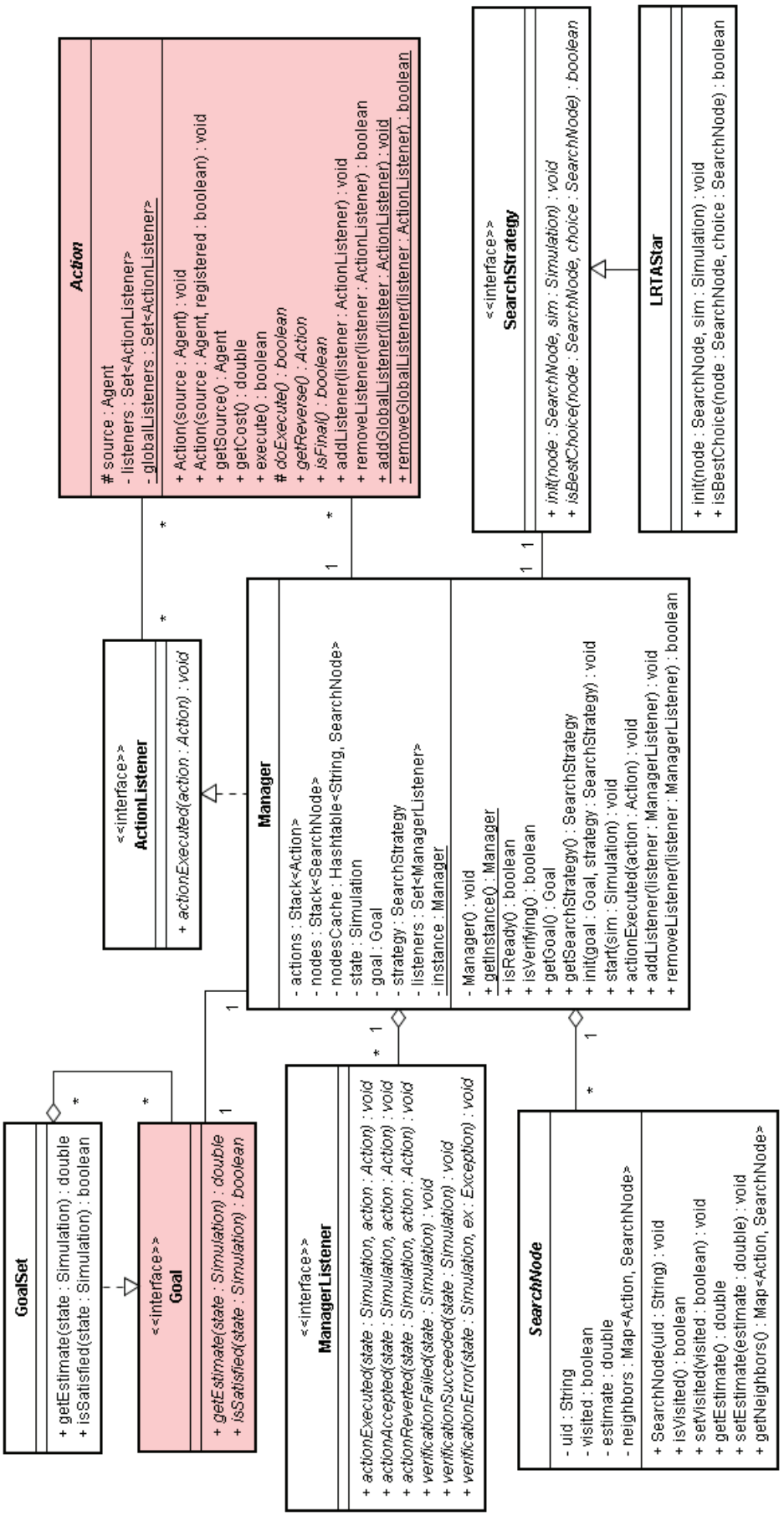

Figura 4.3: Diagrama de classes: Subsistema de verificação

4.2.3 Algoritmo de verificação

A figura 4.4 descreve, em alto–nível, o algoritmo executado em cada etapa do processo de verificação. Já as figuras 4.5 e 4.6 detalham os passos internos da implementação e execução desse algoritmo através de dois diagrama de sequência.

O processo inicia–se com o gerenciador autonômico sendo notificado sobre a execução de uma determinada ação monitorada. Em seguida, com o auxílio da estratégia de busca definida anteriormente, ele identifica se a ação executada é realmente a melhor escolha para o estado atual da simulação. Para isso, ele dispara a estratégia de busca que, por sua vez, analisa o estado atual da simulação (atingido após a execução da ação em foco) frente a todos os outros estados possíveis, considerando–se o conjunto de todas as ações disponíveis. Neste momento, o processo se divide em dois fluxos de execução:

1. No primeiro caso (figura 4.5), o gerenciador autonômico conclui que a ação executada é, realmente, a melhor opção dentre todas as ações possíveis. Então, ele verifica se o objetivo da simulação foi atingido e, novamente, surgem dois novos fluxos de execução:

   1.1. Caso o objetivo estabelecido na inicialização do gerenciador autonômico tenha sido satisfeito, isso significa que a simulação é viável, ou seja, existe um determinado conjunto de ações que, quando executado, leva o sistema do seu estado inicial à um estado ótimo. Portanto, o processo de verificação termina com **sucesso**;

   1.2. Caso contrário, o gerenciador verifica se o estado atual da simulação possui estados vizinhos ainda inexplorados. Se não houver, ele reverte as últimas ações executadas, recursivamente, até que se retorne à um estado que possua um ou mais estados vizinhos ainda inexplorados. Se todas as ações forem revertidas e ainda assim não houver nenhum estado inexplorado, o processo de verificação termina com **falha**, ou seja, o sistema não é viável para o objetivo definido;

2. No segundo caso (figura 4.6), o gerenciador autonômico verifica que, no conjunto de todas as ações possíveis nesta etapa da simulação, existe uma outra ação que, segundo uma estimativa heurística, levaria o sistema à um estado mais próximo do objetivo. Portanto, é necessário reverter a ação executada para possibilitar que esta outra ação mais adequada seja executada posteriormente:

2.1. Se a ação executada não for reversível (ou seja, se ela for uma ação final), a tentativa de reversão do estado da simulação é abortada, mas isso não interrompe o processo de verificação;

2.2. Entretanto, se essa ação for reversível, o gerenciador autonômico executa a ação simétrica reversa, esperando que isso retorne o ambiente ao seu estado anterior. Se essa tentativa de reversão fracassar, tanto pela falha na execução da ação reversa, quanto pela constatação de que o estado da simulação não foi revertido conforme o esperado, o processo de verificação termina com **falha**. Senão, ele continua normalmente.

```
1   verify(action, curr_state) : STATUS
2     prev_state = peek(states)
3     push(states, state)
4     push(actions, action)
5     IF is_best_choice(action, prev_state) THEN
6       IF is_goal_satisfied(curr_state) THEN
7         RETURN VERIFICATION_SUCCEEDED
8       ELSE
9         set_visited(curr_state)
10        FOR ALL a IN available_actions(curr_state) DO
11          next_state = execute(a, curr_state)
12          IF NOT has_path(curr_state, next_state) THEN
13            add_path(curr_state, next_state, a)
14          END
15        END
16        WHILE all_paths_visited(curr_state) DO
17          IF count(actions) == 0 THEN
18            RETURN VERIFICATION_FAILED
19          ELSE
20            action = pop(actions)
21            IF NOT revert(action) THEN
22              RETURN VERIFICATION_ERROR
23            END
24          END
25        END
26      END
27    ELSE
28      IF NOT revert(action) THEN
29        RETURN VERIFICATION_ERROR
30      ELSE
31        RETURN VERIFICATION_NOT_FINISHED
32      END
33    END
34  END
```

Figura 4.4: Algoritmo implementado pelo subsistema de verificação

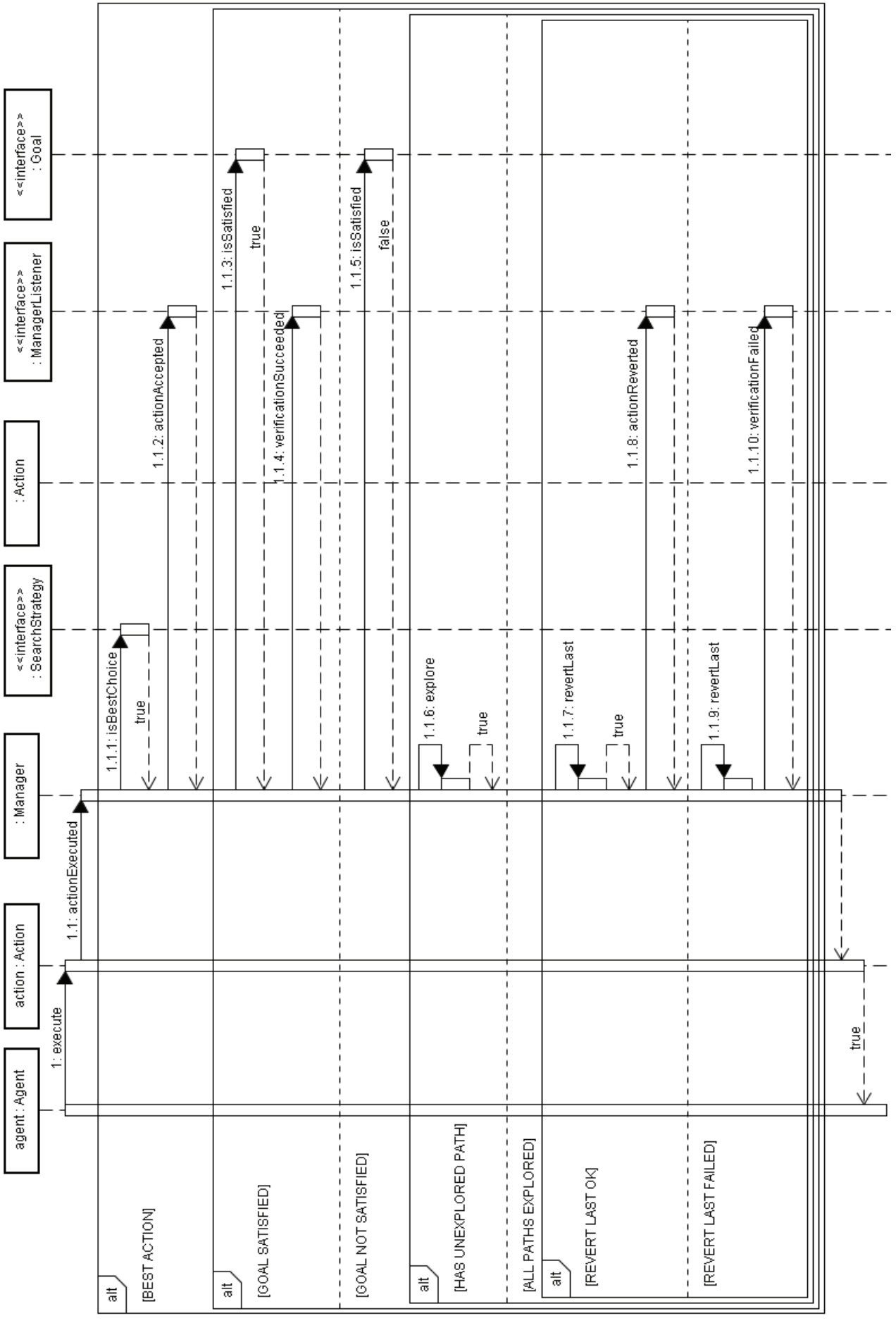

Figura 4.5: Diagrama de sequência: Processo de verificação (fluxo 1)

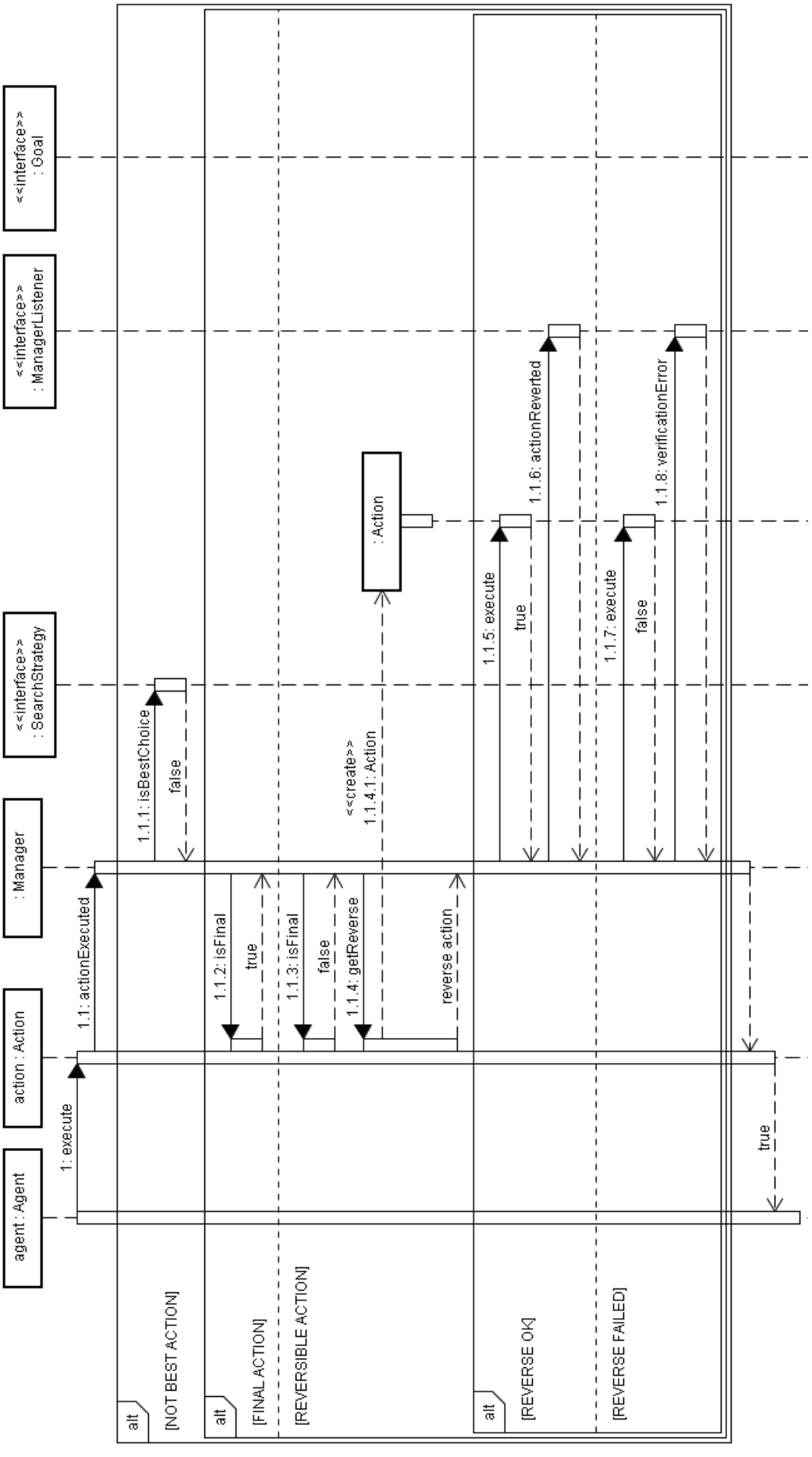

Figura 4.6: Diagrama de sequência: Processo de verificação (fluxo 2)

4.3
Aplicação do método (instanciação do framework)

Com a finalidade de facilitar a compreensão do processo de aplicação do método de verificação aqui proposto, optou–se por exemplificar cada uma das etapas através da utilização de um problema muito conhecido no contexto de inteligência artificial e sistemas multiagentes: colônia de formigas. Esse problema modela o processo de coleta de comida por um conjunto de formigas autônomas pertencentes à uma mesma colônia. O principal objetivo é conseguir efetuar a coleta percorrendo o menor caminho entre as fontes de comida e o ninho (base da colônia), considerando–se a possibilidade de existirem obstáculos no percurso.

4.3.1
Colônia de formigas

Neste problema, os agentes (formigas) cooperam indiretamente modificando o ambiente através da construção e manutenção de caminhos de feromônio[1] que, juntos, formam uma rede de trilhas ligando o ninho às fontes de comida. Quando uma formiga encontra comida ela retorna ao ninho marcando o caminho através do feromônio excretado. Outras formigas da mesma colônia estão sempre monitorando a intensidade e o gradiente local de feromônio. Dessa forma, elas podem alterar o seu comportamento e seguir um determinado caminho demarcado. Caso também encontrem comida, elas voltam ao ninho reforçando o mesmo caminho de feromônio. Ou seja, quanto mais comida, maior a quantidade de formigas e maior a intensidade do feromônio nessa trilha. Como o feromônio é uma substância altamente volátil, à medida que a fonte de comida vai se esgotando e a quantidade de formigas reduz, a trilha de feromônio evapora gradativamente, desaparecendo quando não há mais comida.

Quando obstáculos são introduzidos no ambiente, o problema torna–se ainda mais interessante. A figura 4.7 descreve em três etapas o comportamento das formigas frente à introdução de um obstáculo no percurso do ninho (ponto A) até a fonte de comida (ponto E).

a) Inicialmente, as formigas seguem a trilha do ponto A ao ponto E, sem obstáculos;

b) Então, um obstáculo é introduzido no percurso, interrompendo a trilha de feromônio existente. Cada formiga pode escolher por qual lado seguir

[1]Os feromônios são substâncias químicas altamente voláteis que fornecem um meio de comunicação indireta através de marcações efetuadas no ambiente.

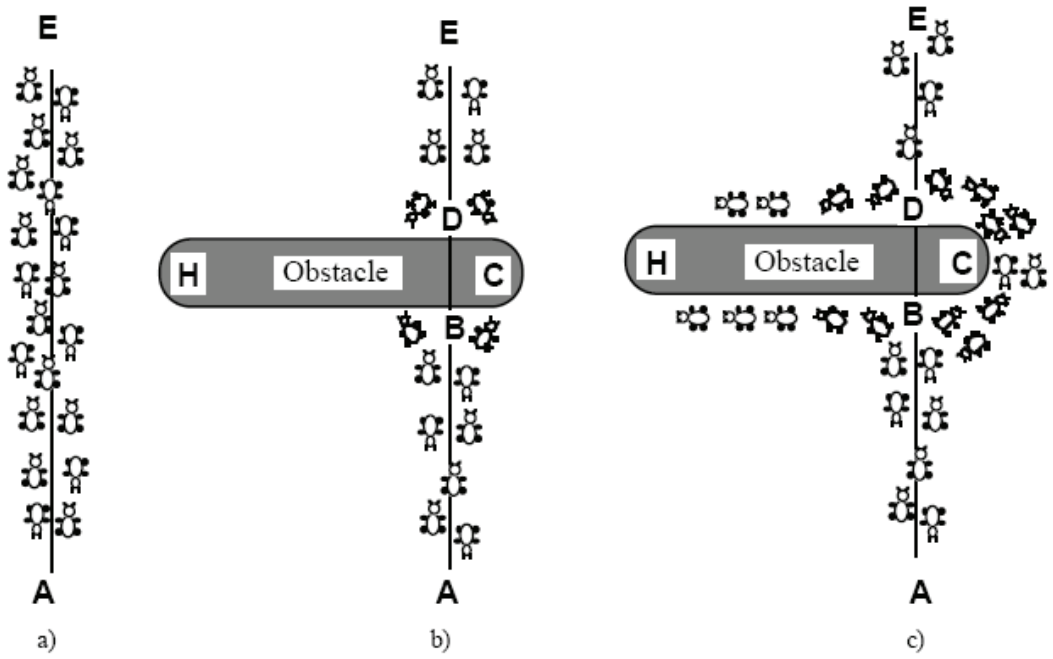

Figura 4.7: Comportamento das formigas frente à introdução de obstáculos no percurso (Dorigo, 1996)

(em direção ao ponto H ou ao ponto C), sendo que ambas as escolhas possuem a mesma probabilidade;

c) Pelo caminho mais longo (via ponto H), as formigas demoram mais tempo para retornar do ponto E ao ponto A. Isso faz com que a trilha de feromônio se evapore mais rápido do que a trilha formada pelo caminho mais curto (via ponto C). Consequentemente, as formigas passarão a seguir o caminho mais curto, pois a intensidade do feromônio será maior. Esse comportamento reforçará cada vez mais a intensidade do feromônio nessa trilha. Dessa forma, o caminho mais longo será gradativamente abandonado, até desaparecer por completo.

É importante ressaltar que a colônia adquire esse comportamento sem que haja qualquer controle centralizado. Cada formiga é autônoma no que diz respeito às ações e decisões que adota buscando cumprir um único objetivo: coletar comida. Os caminhos de feromônio são comportamentos emergentes que surgem através das atividades e interações dos agentes do sistema – as formigas (De Wolf, 2007).

4.3.2 Etapas da aplicação

A figura 4.8 detalha a modelagem multiagente do problema "colônia de formigas" (conforme seção 4.3.1) através da instanciação do subsistema

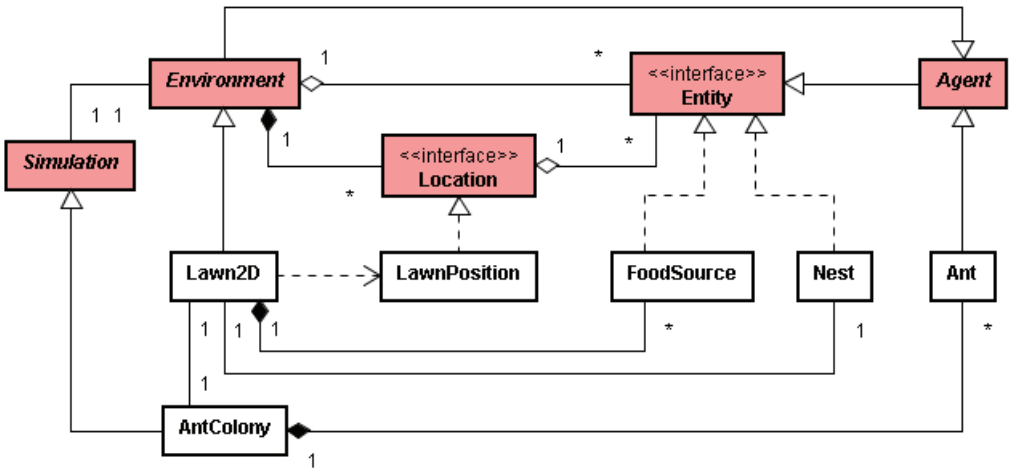

Figura 4.8: Modelagem multiagente de uma colônia de formigas

central descrito na seção 4.2.1. A simulação de uma colônia de formigas (classe *AntColony*) possui um ambiente principal, neste caso chamado de gramado (classe *Lawn2D*). O gramado é, na verdade, um *grid* de duas dimensões com *largura* e *altura* pré–definidas. Cada posição do gramado (classe *LawnPosition*) é dada pelas coordenadas (x, y), onde $0 \leq x < largura$ e $0 \leq y < altura$. As fontes de comida (classe *FoodSource*) e o ninho (classe *Nest*) são entidades inativas e possuem uma posição fixa no gramado. Já as formigas (classe *Ant*) são agentes móveis e podem se locomover por todas as posições do gramado que não estejam ocupadas por outras entidades ou obstruídas.

Para verificar a modelagem apresentada acima, é necessário aplicar o método de verificação experimental de sistemas multiagentes proposto neste trabalho, através da execução de quatro etapas principais realizadas por meio da instanciação do subsistema de verificação apresentado na seção 4.2.2:

1. **Definir ações internas e externas (e suas reversas):**

   Primeiramente, é necessário definir quais ações internas e externas serão monitoradas pelo gerenciador autonômico do método de verificação. De uma forma geral, toda ação que tem potencial para modificar o estado da simulação deve ser monitorada. Ações internas são aquelas executadas pelos agentes e pelo ambiente, enquanto ações externas são entradas recebidas pelo sistema.

Para simplificar o problema, foi assumido que o ambiente da colônia não sofre alteração por entidades externas ao sistema, como obstruções por galhos de árvore ou folhas que caem com o passar do tempo. A configuração inicial do ambiente, no que diz respeito aos obstáculos presentes, é mantida durante

toda a execução. Portanto, não há a necessidade de se considerar ações externas nesta etapa; apenas ações internas.

Sendo assim, a movimentação das formigas é a primeira ação que deverá ser monitorada. Isso porque espera–se que haja uma grande concentração de formigas nas redes formadas pelas trilhas de feromônio que levam às fontes ativas de comida. Caso as formigas estejam se locomovendo desordenadamente, mesmo na presença de comida, isso pode ser um forte indício de que o sistema atingiu um estado de desorganização. Além disso, também deve–se monitorar as ações de coleta e entrega de comida, pois a quantidade coletada é um fator que altera o estado do ambiente.

> *Então, define–se uma ou mais ações simétricas reversas para cada ação indicada. É imprescindível que as ações reversas sejam sempre capazes de reverter os efeitos da ação principal sem deixar qualquer vestígio da execução desta no ambiente. Ou seja, após a execução da ação reversa, a simulação deve retornar exatamente ao estado em que estava antes da execução da ação original.*

Para a ação de movimentação (locomoção) das formigas, a ação reversa imediata é o retorno da formiga à sua posição anterior. Por exemplo: se uma formiga se locomoveu do ponto A para o ponto B (ação principal) a ação reversa esperada é a sua locomoção do ponto B para o ponto A.

Já para a ação de coleta, a ação reversa é a devolução da comida na fonte de onde ela foi retirada. É necessário que o ambiente retorne ao estado exato do momento anterior à execução da ação, ou seja, a fonte deve voltar a possuir a comida retirada e a formiga executora não deve estar de posse de nenhuma comida.

Finalmente, para a ação de entrega, a ação reversa esperada é a retirada da comida do ninho, ou seja, a formiga executora volta a estar de posse da comida que antes tinha sido entregue e, consequentemente, o ninho deixa de conter essa comida.

2. **Definir as variáveis de estado:**

> *Em seguida, identifica–se quais variáveis macroscópicas definem o estado do ambiente, isto é, quais elementos caracterizam o comportamento macroscópico do sistema em questão. Essas variáveis serão, então, utilizadas para construir um identificador único (UID) para cada estado da simulação. Elas também contribuirão para auxiliar as duas próximas etapas da instanciação.*

A partir da descrição do problema e das ações definidas na etapa anterior, é possível identificar algumas variáveis de estado que devem ser observadas:

a) Localização de cada agente (formiga) no ambiente;

b) Quantidade de comida disponível (existente nas fontes de comida);

c) Quantidade de comida coletada, isto é, que foi efetivamente transportada das fontes até o ninho.

Então, é possível definir um identificador único para cada estado da simulação concatenando as informações fornecidas pelas variáveis identificadas. Por exemplo: *[F1=(0,0);F2=(0,0);F3=(0,0);C1=10;C2=5;N=0]* pode ser o identificador único do estado inicial de uma simulação contendo três formigas (*F1*, *F2* e *F3*), ainda localizadas dentro do ninho (posição *(0,0)*), e duas fontes de comida (*C1* e *C2*), com 10 e 5 unidades de comida, respectivamente. Como nenhuma unidade de comida foi transportada – uma vez que este é o estado inicial da simulação –, o ninho (*N*) ainda está vazio (não possui nenhuma unidade de comida).

3. **Definir os objetivos:**

   Em seguida, devem ser enumerados os objetivos do sistema em função das variáveis de estado identificadas na etapa anterior. Um objetivo é um comportamento que se espera que o sistema adquira durante ou ao final da sua execução. Um sistema será considerado viável apenas quando todos os objetivos estabelecidos forem satisfeitos. Portanto, esses objetivos devem obrigatoriamente representar um subconjunto válido de estados, caso contrário o sistema nunca será considerado viável.

Para o problema em questão, busca–se comprovar que o sistema modelado é capaz de completar o processo de coleta, ou seja, transportar toda a comida das fontes disponíveis até o ninho. Considerando–se o exemplo da etapa anterior, os estados finais da simulação são todos cujo identificador seja derivado de *[F1=(x1,y1);F2=(x2,y2);F3=(x3,y3);C1=0;C2=0;N=15]*, onde os valores de *(x1,y1)*, *(x2,y2)* e *(x3,y3)* podem conter quaisquer coordenadas, desde que válidas (uma vez que a posição das formigas não é um objetivo do sistema).

4. **Definir a função de avaliação de estados:**

   Finalmente, elabora–se a função heurística de avaliação de estados que será utilizada no processo de verificação desse sistema. Essa função é

responsável por decidir se o sistema está convergindo ou não para um estado objetivo, ou seja, ela analisa as chances de ele atingir um estado objetivo (que representa o seu comportamento esperado), retornando uma estimativa do custo a partir do estado atual.

São várias as possibilidades de funções que podem ser utilizadas para o problema analisado, cada qual utilizando uma heurística diferente para o cálculo do custo para se atingir um estado objetivo. Entretanto, para que um estado possa ser considerado ótimo, alguns fatores essenciais devem ser considerados:

- A partir do momento em que fontes de comida forem identificadas e caminhos de feromônio demarcados, não será permitido que mais de 50% das formigas que se locomovem por trilhas pré-existentes se movimentem aleatoriamente em busca de mais comida. Isso poderia caracterizar que as trilhas de feromônio não surtem o efeito esperado sobre todas as formigas, o que contribuiria para a desorganização do sistema;
- É esperado que o processo de coleta se mantenha de forma organizada até que todas as fontes de comida se esgotem ou a capacidade de estocagem do ninho seja atingida.