Pontifícia Universidade Católica do Rio Grande do Sul
Faculdade de Informática
Pós-Graduação em Ciência da Computação

Planejamento Proposicional em Agentes BDI

Felipe Rech Meneguzzi

Dissertação apresentada como requisito parcial à obtenção do grau de mestre em Ciência da Computação

Orientador: Dr. Avelino Francisco Zorzo

Porto Alegre, setembro de 2005

"Success is never final. Failure is never fatal. It is courage that counts."

- Sir Winston Churchill

Agradecimentos

Ao Prof. Dr. Avelino Francisco Zorzo por ter me aceito como orientando e tornado possível a conclusão deste trabalho na forma como ele se encontra.

Ao Prof. Dr. Michael da Costa Móra, por ter dedicado parte do seu exíguo tempo durante o curso deste trabalho.

Aos meus pais Nelso e Maria e irmãos Cristiano e Clarissa por me fornecerem a estrutura familiar que me possibilitou a dedicação necessária à conclusão deste trabalho.

À Dra. Clotilde Druck Garcia e todos os membros da comunidade médica responsáveis pela minha existência.

À memória de Bernardo Rafael Konrad Richetti, um grande amigo que nos deixou este ano, lutando bravamente contra o câncer.

À Profa. Dra. Lúcia Maria Martins Giraffa, pelo seu incentivo desde o início e por ter zelado abnegadamente pela conclusão deste trabalho.

À Karina Benato, por sempre acreditar em mim e dar ouvidos aos meus devaneios.

Ao amigo Paulo Henrique Schneider pela ajuda no desenho dos grafos.

À Hewlett-Packard do Brasil e ao projeto CPSE, por terem financiado meu curso de mestrado e me proporcionado inúmeras experiências de aprendizado.

Abstract

The majority of BDI agent implementations perform means-end reasoning using statically defined plan libraries, thus limiting their flexibility at runtime. One of the reasons for this limiting choice in the implementation is the fact that deciding whether an agent is able to accomplish its goal or not is proven to be a very complex problem. On the other hand, advances in the field of propositional planning have enabled a larger class of problems to be tractable. Therefore, this dissertation aims to describe the relationship between BDI agents and propositional planning. In order to accomplish such goal, we describe a mapping between the mental states used in the BDI model and propositional planning formalisms. The mapping thus defined is implemented in an extension to the X-BDI agent model that includes Graphplan as its propositional planning algorithm. Finally, the implemented prototype is used to model and execute a number of case studies.

Resumo

A maioria das implementações de agentes BDI realiza o processo de raciocínio meios-fim utilizando bibliotecas de planos definidas de maneira estática, limitando, portanto, a sua flexibilidade em tempo de execução. Uma das razões para esta escolha limitadora de implementação é a comprovada complexidade computacional de decidir se um agente é ou não capaz de atingir seus objetivos. Em contrapartida, avanços na área de planejamento proposicional permitiram que uma classe maior de problemas possa ser tratada. Desta forma, esta dissertação tem como objetivo descrever a relação entre agentes BDI e planejamento proposicional. A fim de atingir tal objetivo, descreve-se um mapeamento entre os estados mentais usados no modelo BDI e formalismos de planejamento proposicionais. O mapeamento assim definido é implementado em uma extensão do modelo X-BDI de agentes que utiliza como algoritmo de planejamento proposicional o Graphplan. Finalmente, o protótipo implementado é utilizado para modelar e executar alguns estudos de caso.

Sumário

ABSTRACT ix

RESUMO xi

LISTA DE FIGURAS xvii

LISTA DE TABELAS xix

LISTA DE SÍMBOLOS E ABREVIATURAS xxi

Capítulo 1: Introdução 1

I Embasamento Teórico 5

Capítulo 2: Agentes BDI 7

2.1 Agentes de Software 7

2.2 O Modelo BDI 8

2.2.1 Máquinas, Intenções e Estados Mentais 9

2.2.2 Estados Mentais e Raciocínio Prático 10

2.2.3 Principais componentes do modelo BDI 15

2.3 Teorias BDI 17

2.3.1 Modelo de Cohen e Levesque 17

2.3.2 Lógica BDI de Rao e Georgeff 20

2.4 Arquiteturas BDI 24

2.4.1 IRMA 25

2.4.2 PRS 27

2.4.3 dMARS 33

2.4.4 AgentSpeak 34
2.4.5 X-BDI 35
2.5 Considerações 36

Capítulo 3: Planejamento Proposicional 39
3.1 Formalismos de Planejamento Proposicional 40
3.2 Compilação para SAT 43
3.3 Graphplan 44
3.3.1 Expansão do Grafo 46
3.3.2 Extração da Solução 49
3.3.3 Condições de Finalização 51
3.4 Considerações 54

Capítulo 4: X-BDI 55
4.1 Framework Lógico 55
4.1.1 Semântica WFSX 56
4.1.2 Procedimento SLX 56
4.1.3 Revisão de Cláusulas na SLX 56
4.2 Modelo do Agente 57
4.3 Processo de Deliberação no X-BDI 63
4.3.1 Seleção dos Desejos 63
4.3.2 Formação das Intenções 66
4.3.3 Revisão de Intenções 67
4.4 Considerações 68

II Descrição do Trabalho 71

Capítulo 5: X-BDI Estendido 73
5.1 Estrutura Cognitiva do Agente 73
5.2 Processo de Raciocínio do Agente 75
5.3 Revisão de Intenções 77
5.4 Considerações 78

Capítulo 6: Implementação do Protótipo **81**
6.1 Arquitetura da Implementação 81
6.2 Implementação do Graphplan 83
6.3 AgentViewer 85
6.3.1 Controle do X-BDI 86
6.3.2 Comunicação com o X-BDI 87
6.3.3 Modelo de Mundo 87

Capítulo 7: Estudos de Caso **91**
7.1 Célula de Produção 91
7.2 Injeção de Falhas utilizando o *AgentViewer* 97

Capítulo 8: Considerações Finais **103**
8.1 Trabalhos Futuros 104

REFERÊNCIAS BIBLIOGRÁFICAS **105**

Lista de Figuras

2.1 Raciocínio prático segundo Davidson. 11
2.2 Raciocínio Prático segundo Bratman. 13
2.3 Problema do Pacote, $i1$ e $i2$ representam crenças irrelevantes. 15
2.4 Estrutura interna da arquitetura IRMA. 25
2.5 Estrutura da Arquitetura do PRS. 28
2.6 Interpretador do PRS 32

3.1 Estrutura de um planejador por SAT. 44
3.2 Visão Geral do Algoritmo Graphplan. 45
3.3 (1) Efeitos Inconsistentes - (2) Interferência - (3) Necessidades Concorrentes - (4) Suporte Inconsistente 47
3.4 Grafo de planejamento para problema do Exemplo 3.2. 49
3.5 Extração de solução. 50
3.6 Extração de solução utilizando memoização. 52

4.1 Esquema de funcionamento do X-BDI 57
4.2 Relações entre os elementos de $\mathbb{P}(D')$ 64

5.1 Esquema modificado do X-BDI. 77

6.1 Arquitetura da Solução 82
6.2 Tempos de execução para o domínio *Rockets*. 84
6.3 A ferramenta AgentViewer 85
6.4 Funcionalidades do AgentViewer 86

7.1 Uma Célula de Produção BDI. 92
7.2 Tempos de execução para variações do problema. 97
7.3 Célula de produção. 99

Lista de Tabelas

2.1 Classificações de Estados Mentais [MÓR 99a] 10

Lista de Símbolos e Abreviaturas

AC	Áreas de Conhecimento	v
ACL	*Agent Communication Language* - Linguagem de Comunicação de Agentes	v
BDI	*Belief, Desire, Intention* - Crença, Desejo, Intenção	v
CNF	*Conjunctive Normal Form* - Forma Normal Conjuntiva	v
CTL	*Computation Tree Logics* - Lógicas de Árvores Computacionais	v
dMARS	*distributed Multi-Agent Reasoning System* - Sistema de Raciocínio Multi-Agente distribuído	v
ELP	*Extended Logic Programming* - Programação em Lógica Estendida	v
FIPA	*Foundation for Intelligent Physical Agents* - Fundação para Agentes Físicos Inteligentes	v
GPS	*General Problem Solver* - Resolvedor de Problemas Genérico	v
IA	Inteligência Artificial	v
KA	*Knowledge Areas* - ACs	v
PRS	*Procedural Reasoning System* - Sistema de Raciocínio Procedural	v
SLX	*Selected Linear derivation EXtended with explicit negation*- Derivação por Seleção Linear estendida com a negação explícita	v
STRIPS	*Stanford Research Institute Problem Solver* - Resolvedor de Problemas do Instituto de Pesquisas de Stanford	v
X-BDI	*EXecutable BDI* - BDI Executável	v

WFSX	*Well Founded Semantics EXtended with the explicit negation*- Semântica Bem Fundada Estendida com a negação explícita	v

Capítulo 1

Introdução

O aumento da complexidade e da necessidade de distribuição dos sistemas desenvolvidos atualmente têm motivado a utilização de uma modelagem em termos de agentes autônomos. Este tipo de modelagem aproxima tais sistemas do desenvolvimento de sistemas multi-agentes [JEN 99, JEN 00]. A utilização deste tipo de abordagem é o resultado de uma melhor compreensão do comportamento de sistemas complexos em termos de interações entre entidades independentes [WOO 00c]. O aumento da utilização de agentes como mecanismo de abstração motivou uma série de trabalhos voltados ao desenvolvimento de sistemas que utilizam agentes abstraindo seu mecanismo de funcionamento [CM 98, JEN 00, ASH 02, KNU 02], também conhecido como *nível macro* [BUS 00, WOO 00c, ZAM 00, ET 01, FAT 01]. Em contrapartida, diversas questões relativas ao funcionamento interno dos agentes, também conhecido como *nível micro* [BRA 88, GEO 89a, GEO 99, RAO 91a, RAO 96, BOR 02, NID 02], receberam menos atenção nos últimos anos, tendo seu interesse sido revigorado recentemente.

O desenvolvimento de agentes racionais, isto é, agentes capazes de realizar ações úteis considerando a sua percepção de mundo [RUS 94], tem sido um dos principais objetivos de pesquisas em Ciência da Computação desde o seu início [TUR 48]. Tal desenvolvimento deu origem a diversas abordagens para a implementação de raciocínio computacional, iniciando com o *General Problem Solver* (GPS) e os algoritmos genéricos de resolução de problemas, e evoluindo nos Algoritmos de Planejamento [RUS 94]. Apesar destes sistemas serem capazes de definir como os objetivos do agente serão atingidos, eles não lidam com o problema de quais objetivos devem ser perseguidos. Ou seja, eles são capazes de realizar raciocínio meios-fim mas não lidam com o problema do raciocínio prático [SCH 01]. Sistemas que realizam planejamento levaram à criação do modelo de raciocínio baseado em agentes deliberativos como meio de tratar o raciocínio prático. Estes sistemas utilizavam, inicialmente, uma série de mecanismos de tomada de decisão, como teoria da decisão. Apesar destes mecanismos serem bem-definidos teoricamente, eles se mostraram inadequados para implementação, uma vez que assumem agentes com recursos com-

putacionais e de tempo ilimitados [SCH 01]. A fim de resolver a questão dos recursos limitados, um modelo filosófico de raciocínio prático foi formalmente definido e implementado [BRA 88], que, em teoria, permite a um agente limitar o tempo gasto em deliberação. Não obstante, as implementações destes agentes tendem a evitar a complexidade do raciocínio meios-fim através do uso de bibliotecas de planos, definidas para o agente antes de sua execução. Conseqüentemente, esta abordagem resolve o problema de limitação de recursos de um agente, entretanto, ela delega a responsabilidade da construção dos planos para o seu desenvolvedor.

Um dos mais importantes problemas em nível micro atualmente é a prova da existência de um agente capaz de realizar uma determinada tarefa em um determinado ambiente. Este problema é denominado *Agent Design Problem* [WOO 00c]. Se considerarmos que a descrição de um ambiente consiste de um conjunto de estados válidos, um estado inicial e um conjunto de ações e seus significados, e que o resultado de uma tarefa é representado por um estado alvo [WOO 01], podemos facilmente relacionar a verificação da capacidade de um agente de cumprir uma tarefa com um problema de planejamento.

O objetivo deste trabalho é demonstrar como incorporar planejamento proposicional em um modelo de agentes BDI de modo a provê-lo com a habilidade de realizar tanto raciocínio prático como raciocínio meios-fim em tempo de execução. A abordagem utilizada neste trabalho consiste em, tomando um modelo de agentes BDI específico como base, estendê-lo com um algoritmo de planejamento proposicional [NEB 00]. O modelo de agentes utilizado neste trabalho é o X-BDI [MÓR 99b], que irá utilizar o Graphplan [BLU 97] como mecanismo de raciocínio meios-fim. Considerando o fato de que o X-BDI já realiza uma forma de planejamento no seu raciocínio meios-fim, um dos resultados esperados da sua extensão é o aumento da classe de problemas tratáveis pelo X-BDI. Este aumento deve ser na direção da classe de problemas para os quais o Graphplan apresenta bom desempenho.

Este trabalho é dividido em duas partes: a primeira parte contém o embasamento teórico necessário ao desenvolvimento deste trabalho. A segunda parte contém a descrição do trabalho desenvolvido. Na primeira parte deste trabalho, os Capítulos 2 e 3 contêm, respectivamente, o material teórico sobre agentes BDI e sobre planejamento proposicional; no final desta parte, o Capítulo 4 descreve o modelo de agentes X-BDI utilizado na implementação descrita no Capítulo 6. Ao leitor com familiaridade nos assuntos desenvolvidos nestes capítulos é recomendada a leitura das respectivas seções de considerações a respeito do material apresentado, sendo o restante dos capítulos de leitura opcional. Na segunda parte do trabalho, o Capítulo 5 descreve as modificações realizadas no X-BDI para que este comporte a utilização de algoritmos de planejamento proposicional como processo de raciocínio meios-fim; o Capítulo 6 descreve a implementação realizada para verificar a aplicabilidade do trabalho apresentado; a seguir, o Capítulo 7 descreve um conjunto de estudos de caso desenvolvidos utilizando a implementação descrita no Capítulo 6 a fim

de observar o funcionamento dos agentes desenvolvidos nela. Finalmente, o Capítulo 8 contém as considerações finais a respeito deste trabalho.

Parte I

Embasamento Teórico

Capítulo 2

Agentes BDI

Este capítulo visa fornecer os subsídios teóricos relacionados a agentes computacionais. A Seção 2.1 contém a definição informal de agentes que irá caracterizar tais entidades ao longo deste trabalho. A Seção 2.2 descreve o modelo de agentes deliberativos utilizados neste trabalho. A Seção 2.3 reúne os principais modelos formais criados para caracterizar o modelo BDI. A Seção 2.4 descreve alguns dos sistemas mais importantes que implementam o modelo BDI. No final do capítulo são apresentadas considerações a respeito das teorias estudadas e sua relação com o restante do trabalho.

2.1 Agentes de Software

Um número crescente de sistemas têm utilizado a noção de agentes autônomos como abstração na descrição de seus componentes, aproximando o desenvolvimento de sistemas computacionais complexos ao desenvolvimento de sistemas multi-agentes [JEN 99, JEN 00]. Esta abordagem pode ser atribuída à melhor compreensão do comportamento de um sistema quando este é descrito em termos de interações entre agentes autônomos [WOO 00c]. Apesar do aumento da sua utilização como mecanismo de abstração, ainda existem questões fundamentais em relação à construção de sistemas orientados a agentes que permanecem sem resposta [JEN 00]. Em particular, questões relacionadas a metodologias de desenvolvimento de sistemas multi-agentes. Além disto, a comunidade de pesquisa em sistemas multi-agentes não atingiu um consenso em relação a uma definição precisa para o que é um agente. Apesar da possibilidade de que esta questão nunca seja respondida de forma definitiva, alguns autores utilizam a Definição 2.1 [WOO 97, JEN 99, JEN 00], que será utilizada ao longo deste trabalho.

Definição 2.1 (Agente) *Um agente é um sistema computacional encapsulado situado em um ambiente onde é capaz de agir de maneira flexível e autônoma a fim de atingir seus objetivos de*

projeto.

Esta definição deixa implícitas uma série de características de agentes, que são [JEN 00]:

- Agentes são entidades de resolução de problemas claramente identificáveis e com interfaces e limites bem definidos;
- Agentes estão situados em um ambiente em particular e recebem entradas relacionadas ao estado deste ambiente e agem sobre ele através de atuadores; Agentes são projetados com um propósito específico, de forma que eles possuam objetivos próprios a alcançar;
- Agentes são autônomos na medida que eles têm controle sobre o seu estado interno e seu comportamento;
- Agentes são capazes de exibir comportamento flexível na resolução de problemas quando estão tentando satisfazer seus objetivos. Eles devem ser reativos no sentido de responder a mudanças no ambiente de maneira temporalmente aceitável e pró-ativos no sentido de agir na expectativa de alcançar objetivos futuros.

2.2 O Modelo BDI

Durante a evolução de sistemas baseados em agentes, um dos mais importantes modelos a ter sido criado é o modelo baseado na interação dos estados mentais Crenças, Desejos e Intenções (BDI[1]). O modelo BDI se baseia na noção de estados mentais para descrever o comportamento de um agente. Esta noção foi descrita inicialmente nos trabalhos de Searle [SEA 82] e Dennet [DEN 87]. Juntamente com a teoria de ações intencionais de Davidson [DAV 88, apud[BRA 99a]], a utilização de estados mentais foi refinada por Bratman [BRA 99a, BRA 84, BRA 90, BRA 99b], resultando no modelo BDI.

O modelo BDI é um dos mais conhecidos e estudados modelos de agentes deliberativos [GEO 99], com o qual foram realizados diversos estudos teóricos [BRA 99a, COH 90, RAO 95b] e implementações visando a prova de sua utilidade na resolução de diversos problemas [BRA 88, GEO 89a, RAO 96, POL 90]. A evolução do modelo BDI no contexto de ciência da computação pode ser traçada através da análise destes trabalhos, uma vez que cada um deles contribuiu para reduzir a distância entre o modelo filosófico que o originou [BRA 84] até um modelo de raciocínio prático concreto. Alguns destes trabalhos são descritos nas Seções 2.3 e 2.4.

A validade do modelo BDI já foi questionada em relação a abordagens mais recentes de descrição de sistemas [GEO 99]. Entretanto, diversos autores continuam utilizando o modelo

[1]Da sigla em inglês *Beliefs, Desires and Intentions*.

BDI como modelo base de agentes deliberativos [SCH 01, HOE 03, WOO 99, NID 02, BOR 02, BOR 03], buscando o aprimoramento do modelo através da construção de novas teorias que embasem o modelo BDI em um sistema unificado [HOE 03, WOO 00b] ou da extensão de teorias pré-existentes [MÓR 99a, NID 02, BOR 02, NAI 03].

2.2.1 Máquinas, Intenções e Estados Mentais

A utilização de estados mentais na descrição de sistemas computacionais remete aos trabalhos de Dennet. Estes trabalhos descrevem *sistemas intencionais* como entidades cuja conduta pode ser prevista através da atribuição de Crenças, Desejos e perspicácia racional. Estes sistemas se dividem em níveis onde um sistema intencional de primeira ordem é um sistema onde as entidades têm crenças e desejos a respeito de qualquer objeto que não outras crenças e desejos. Um sistema de segunda ordem pode ter crenças e desejos a respeito de outras crenças e desejos, e assim por diante, expandido a hierarquia *ad infinitum* [DEN 78, DEN 87, SHO 93, MÓR 99a, WOO 00b].

Descrevendo máquinas como estados mentais

A motivação para a atribuição de estados mentais a sistemas computacionais pode ser analisada do ponto de vista da sua utilidade e legitimidade. É legítimo atribuir estados mentais a máquinas quando tal atribuição possibilita que seja compreendida a mesma informação a respeito da máquina que seria compreendida de uma pessoa. Esta atribuição é útil quando facilita o entendimento da estrutura da máquina, seu comportamento passado e futuro, e maneiras de consertá-la ou melhorá-la. Esta atribuição talvez não seja fundamental nem para descrever seres humanos, mas a utilização de algum tipo de qualidade mental possibilita que uma descrição sucinta de um estado em particular de determinada máquina seja feita [MCC 79]. Desta forma é possível e correto descrever sistemas extremamente simples com estados mentais, porém é mais útil fazê-lo para descrever sistemas cuja estrutura interna não é completamente conhecida.

Tipos de Estados Mentais

Diferentes classificações para estados mentais foram criadas como resultado da sua utilização no estudo do raciocínio prático e na descrição de sistemas. Neste trabalho enumera-se as classificações de três autores extraídas de [MÓR 99a], especificamente as classificações de Searle [SEA 82], Shoham & Cousins [SHO 94] e Kiss [KIS 92]. Estas classificações estão resumidas na Tabela 2.1, e ordenadas por aproximação ao modelo de raciocínio prático de Bratman descrito na Seção 2.2.2.

Tabela 2.1: Classificações de Estados Mentais [MÓR 99a]

Autor	**Grupos**	**Nome do Estado**
Searle	Informacionais	Crenças, Conhecimento
	Pró-Ativos	Desejos, Preferências, Intenções, Obrigações
Shoam & Cousins	Informacionais	Crenças, Conhecimento
	Motivacionais	Desejos, Preferências, Intenções, Planos
	Sociais	Permissões, Obrigações
Kiss	Informacionais	Crenças, Conhecimento
	Conativos	Comprometimentos, Intenções, Planos
	Affetivos	Desejos, Permissões, Obrigações

2.2.2 Estados Mentais e Raciocínio Prático

Esta seção irá detalhar as teorias relacionadas ao uso de estados mentais para descrever comportamento, inicialmente descrevendo a teoria de Davidson de Crenças e Desejos, seguindo para a descrição da teoria consolidada de Bratman de Desejos, Crenças e Intenções.

A teoria de Crenças e Desejos de Davidson

No trabalho "Actions Reasons and Causes" [DAV 63, apud [BRA 99a]], Davidson apresenta as bases de uma teoria sobre o raciocínio prático. Tal trabalho define *ações intencionais* como sendo ações que são explicáveis através das razões do agente para tê-las realizado, razões estas que podem ser traçadas até pares de crenças e desejos. Assim sendo, o autor pressupõe que é possível realizar uma avaliação total (*all-out evaluation*) das crenças e desejos para se gerar cada ação. Esta avaliação total é baseada em uma função de utilidade máxima que, levando em consideração as crenças do agente, determina a cada momento qual é a ação mais eficiente a ser realizada de modo a satisfazer seus desejos. Um esquema básico deste funcionamento pode ser visto na Figura 2.1.

Alguns problemas em relação a esta teoria puderam ser verificados, em particular, relacionados ao funcionamento deste mecanismo de raciocínio em um contexto com tempo e recursos limitados. Isto motivou a criação de um modelo de raciocínio que contemplasse o tratamento destas limitações [BRA 99a, BRA 99b]. Este problemas serão descritos a seguir:

Problema 2.1 (Problema de Buridan) *Se um asno racional (i.e. com a habilidade de sempre tomar a melhor decisão a respeito de como lidar com uma situação) for colocado em uma posição*

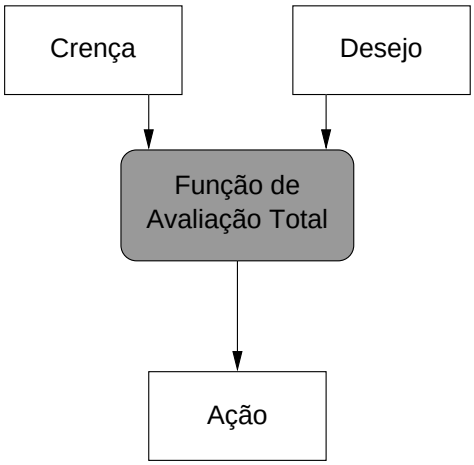

Figura 2.1: Raciocínio prático segundo Davidson.

eqüidistante de duas pilhas iguais de feno, ele irá morrer de fome pois não irá ser capaz de decidir qual é a melhor ação a ser tomada

O primeiro problema relacionado à função de avaliação total é chamado de Problema de Buridan[2] [BRA 99a, BRA 99b]. Considerando que todas as opções do asno são idênticas, ele não será capaz de estabelecer qual delas é a melhor. Este problema ilustra o fato de que a teoria de pensamento e ação de Davidson enfrentaria sérios problemas em situações com opções igualmente desejáveis, situações estas que seres humanos são eficazes em solucionar.

Problema 2.2 (Problema do Raciocínio Parcial) *Considere que o Agente Smith tem o desejo de comprar um jogo de computador em particular. O Agente também tem a crença de que isto pode ser alcançado indo para a loja X, mas o agente não sabe se a loja X tem ou não uma cópia do jogo no seu estoque.*

Também deve haver uma maneira de se gerar estados mentais (desejos/intenções) parciais a fim de se avançar na direção do cumprimento de objetivos cuja viabilidade teórica ainda não é completamente conhecida. Se a ação necessária para se cumprir o desejo do Agente Smith depender somente de suas crenças e desejos, não fica claro se o agente deve ir à loja para verificar a existência do jogo ou não. A fim de solucionar este problema, é necessário um estado mental intermediário que permita ao agente iniciar o movimento em direção à satisfação do seu desejo. Em contrapartida, este estado mental deve permitir sua revisão durante a sua execução de modo que ele possa refletir os dados que vão se completando à medida que a informação completa é preenchida.

[2]Por ser atribuído ao filósofo Jean Buridan, mesmo que tal problema nunca tenha sido encontrado entre seus escritos.

Problema 2.3 (Problema da Desconsideração do Tempo) *Se o tempo que o Agente Smith leva para considerar todas as suas opções é relevante em relação ao tempo necessário para que o mundo mude de estado, então, quando o agente terminar seu processo de raciocínio, suas decisões podem estar desatualizadas.*

Ao realizar uma avaliação total das crenças e desejos do agente, o modelo de Davidson assume que o tempo necessário para realizar tal operação é irrelevante em relação à taxa de mudança do estado de mundo. Desta forma o principal problema em relação a este modelo é que ele desconsidera o tempo decorrido entre a deliberação e a ação. Este problema se torna mais claro quando se consideram agentes com recursos limitados, tais como seres humanos e programas de computador, em um mundo dinâmico. Neste caso, o tempo gasto no processo de deliberação deve ser o menor possível, dado que o estado atual do mundo pode mudar durante este processo, caso contrário as decisões tomadas durante a deliberação podem não mais ser válidas no final do processo. Desta forma, mesmo que o modelo de Davidson seja interessante do ponto de vista teórico, ele falha em diversos aspectos quando utilizado para descrever o funcionamento de agentes no mundo real.

A importância das intenções

Apesar de ser uma base interessante para uma teoria completa de raciocínio prático, a teoria de desejos e crenças descrita por Davidson possui limitações. Estas limitações estão ligadas ao uso de apenas dois estados mentais para lidar com cursos de ação conflitantes [BRA 90]. Não obstante, seria possível realizar o processo de deliberação apenas com estes estados mentais [BRA 99b], mesmo que este processo seja extremamente limitado, em particular em relação a limitações de tempo e recursos. A deliberação, neste caso, sofreria de um problema onde o agente reconsideraria seus cursos de ação constantemente levando-o a possivelmente nunca levar seus desejos longe o suficiente para satisfazê-los, visto que cada ação seria o resultado de uma avaliação completa dos desejos e crenças [RAO 96]. Em contrapartida, se o agente gastar algum tempo planejando um curso de ações consistente que leve a um dado objetivo e se comprometer com a sua execução, o agente apenas re-consideraria suas ações quando algo for alcançado.

Considerando que o planejamento é uma peça fundamental da inteligência, três habilidades são fundamentais para agentes que planejam: agir intencionalmente, criar e executar planos [BRA 99b]. A primeira habilidade é um requisito para a segunda, visto que é necessário um esforço ativo do agente para que ele crie um plano. Este esforço descreve de maneira apropriada o modo como a mente humana funciona, dado que estamos constantemente refletindo em relação a nossas ações e intenções, no passado e no futuro. É este mecanismo de raciocínio que nos permite ponderar sobre cursos de ação complexos *a priori*, evitando a necessidade de decidir todas

as ações no momento em que são executadas.

Desta forma, Bratman propõe o uso de intenções como estados mentais distintos e com a mesma importância dos desejos e crenças no processo deliberativo [BRA 99b]. Este modelo é esquematizado na Figura 2.2.

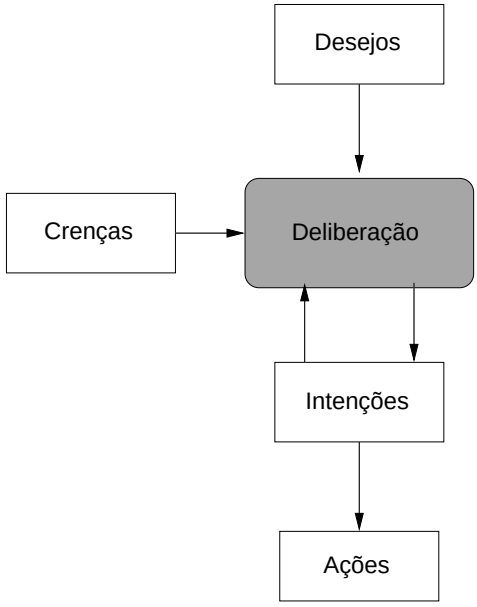

Figura 2.2: Raciocínio Prático segundo Bratman.

O papel das intenções não é apenas complementar o modelo de crenças e desejos como outro elemento que associa racionalidade aos atos de um agente, mas prover ao agente a habilidade de estruturar seu planejamento, participando do processo do raciocínio meios-fim do agente. O planejamento ao qual Bratman se refere não é necessariamente o mesmo que ocorre em sistemas como o STRIPS [FIK 71], mas um processo de mais alto nível onde as intenções guiam o planejamento em si. As intenções se distinguem dos desejos na medida em que são pró-atitudes que controlam a ação do agente enquanto os desejos apenas a influenciam. Intenções constituem planos de alto nível, que serão refinadas e modificadas quando necessário. Estes planos permitem ao agente iniciar o movimento em direção aos seus objetivos cuja viabilidade ainda não está definida.

No contexto desta teoria, são definidas três características fundamentais no processo de raciocínio. Intenções têm um efeito mais direto nas ações do agente no sentido de que elas controlam a sua conduta. Além disto, elas possuem uma característica denominada inércia, que significa que, apesar de revogáveis, as intenções resistem à reconsideração. Finalmente, intenções servem de entrada na geração de novas intenções [BRA 99b].

A possibilidade de revogação de intenções durante o processo de raciocínio suscita uma das principais questões do modelo BDI, que é o balanço entre a estabilidade das intenções e a sua

reconsideração. Este problema foi identificado na criação da arquitetura IRMA. Um agente que reconsidera suas intenções muito freqüentemente é chamado de *cauteloso*, e um agente que se apega demais às suas intenções é chamado de *audacioso* [BRA 88]. Diversos autores estudaram este problema e suas soluções potenciais [KIN 90, KIN 91, POL 94].

Quando utilizadas como entrada na geração de novas intenções, as intenções restringem quais estados serão considerados no raciocínio futuro, *i.e.* elas estabelecem padrões de relevância para as opções que serão consideradas na deliberação futura. Estes padrões criam um filtro de admissibilidade para as soluções de um dado problema. Conseqüentemente, esta pré-deliberação que limita o espaço de busca por soluções provê uma maneira de superar as limitações de tempo e recursos mencionada anteriormente. Neste sentido, estas características das intenções as tornam uma solução mais apropriada do que um função de utilidade máxima. Além disto, este filtro de admissibilidade resolve o Problema de Buridan (Problema 2.1) na medida em que fornece uma justificativa para desconsiderar certas ações, quando um curso de ação for estabelecido.

A utilização de intenções permite a coordenação do agente tanto interna quanto externamente. Do ponto de vista interno, o agente saberá que ao se comprometer com a realização de uma intenção, quando o momento chegar, ele irá, pelo menos, tentar fazer aquilo a que se propunha. Além disto, o esforço despendido no planejamento de como cumprir o que se pretende, definindo os passos preliminares, irá deixar o agente numa posição apropriada para realizar as ações necessárias e ter melhores condições de sucesso. Do ponto de vista externo, uma vez que os demais agentes de um sistema tenham sido informados das intenções de um determinado agente, eles poderão planejar suas ações baseados na expectativa de que este irá cumprir as suas intenções.

A possibilidade de realimentação das intenções no processo de raciocínio (*i.e.* intenções podem gerar outras intenções) também acarreta dificuldades. A principal delas é chamada de Problema do Pacote (*Problem of the Package Deal*) [BRA 90]. Este problema consiste em, inicialmente, escolher uma intenção para satisfazer um dado objetivo, e posteriormente durante a utilização desta intenção na geração de novas intenções, esta gerar uma nova intenção no sentido de fazer algo que não representa os desejos do agente (Problema 2.4).

Problema 2.4 (Problema do Pacote) *Se o Agente Smith tem o desejo de obter o grau de Mestre com um bom trabalho (m na Figura 2.3), ele terá a intenção de trabalhar muito como conseqüência (w na Figura 2.3). Esta intenção de trabalhar muito pode resultar na intenção de não dormir algumas noites a fim de aumentar o seu tempo de trabalho ($\neg s$ na Figura 2.3), mesmo que seus desejos afirmem que ele gosta de dormir bem (s na Figura 2.3).*

Neste caso, a intenção de trabalhar muito gerou uma intenção conflitante com os seus desejos pois ela "veio com o pacote". Conseqüentemente, a intenção de não dormir algumas noites não é utilizada como entrada para outras intenções, visto que ela não representa os desejos do Agente

Smith. Saber como diferenciar intenções que podem gerar novas intenções daquelas que não podem é uma das dificuldades inerentes a este problema.

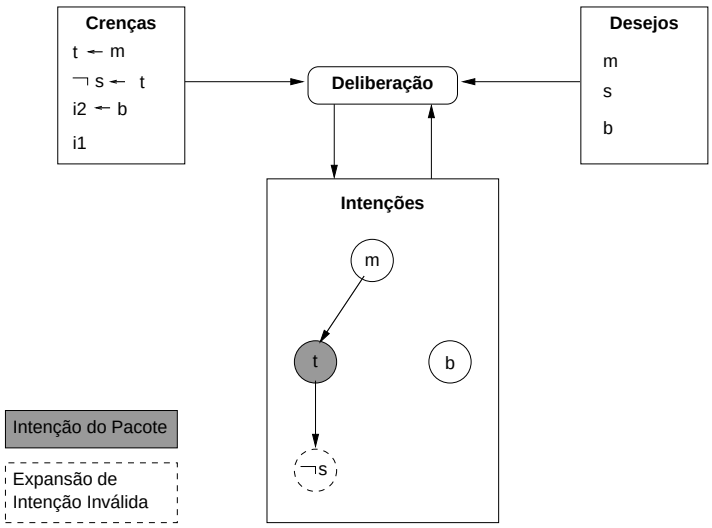

Figura 2.3: Problema do Pacote, $i1$ e $i2$ representam crenças irrelevantes.

A teoria de Bratman fornece uma alternativa de solução não apenas para os problemas do modelo de Davidson, mas também introduz um modelo onde é possível que as intenções de um agente, e em última instância suas ações, possam ser motivadas por outros elementos que não pares de desejos e crenças. Esta característica possibilita a geração de intenções sem uma relação direta com os desejos, o que permite ao agente se aproveitar de oportunidades surgidas no ambiente, o que é uma característica importante do comportamento inteligente [WOO 00b].

2.2.3 Principais componentes do modelo BDI

O modelo BDI é, essencialmente, uma transposição computacional do modelo de raciocínio proposto por Bratman [BRA 84, BRA 87, BRA 90]. Este modelo é baseado na utilização de crenças, desejos e intenções como estados mentais fundamentais e necessários na descrição do raciocínio prático. Apesar do modelo BDI ter sido originado por uma teoria em comum, existem várias interpretações para ele, de modo que atualmente nenhuma arquitetura ou modelo formal o representa em sua totalidade. Desta forma, a operação do modelo BDI pode ser descrita de modo abrangente como o balanço racional de crenças, desejos, intenções e demais componentes auxiliares como planos, comprometimentos e ações [COH 90]. Os componentes do modelo BDI podem ser descritos da seguinte forma [MÜL 96]:

Crenças representam a expectativa do agente em relação ao estado atual do mundo e a possibilidade de um dado curso de ações levar a outro estado de mundo. Tal estado é normalmente modelado através de uma semântica de mundos possíveis, porém, nem todas as instâncias do modelo BDI utilizam uma semântica tão complexa. Crenças podem ser representadas de maneira mais simples [GEO 99], como por exemplo, proposições ou variáveis em um programa [D'I 98a, MÓR 99a].

Desejos são uma abstração que especifica preferências em relação a estados de mundo ou cursos de ação. Um ponto em comum de vários trabalhos é o fato de que desejos não são necessariamente consistentes [BRA 90, COH 90, MÓR 99a, WOO 99]. Por exemplo, um agente pode ter o desejo de avançar contra um inimigo para atacá-lo e ao mesmo tempo fugir dele a fim de garantir a sua sobrevivência. O que determinará qual dos dois desejos será adotado pelo agente será o estado de mundo onde o agente se encontra.

Desejos são facilmente representáveis em programas de computador, tal como as crenças. Eles podem simplesmente representar estados de mundo, de maneira semelhante a sistemas de planejamento [GEO 99]. A utilização de desejos torna um sistema orientado a objetivos, o que é vantajoso na medida em que o sistema possui um senso de propósito em suas ações. Isto permite que o agente, ao descobrir que suas ações não tiveram o resultado desejado, possa tentar algum tipo de ação corretiva.

Intenções são utilizadas para superar a limitação de recursos do agente no processo de raciocínio focando o processo de deliberação em um subconjunto dos seus desejos, com cuja realização o agente se compromete. O processo de seleção dos desejos que irão constituir intenções é chamado de *formação das intenções*.

A função das intenções não se limita a um conjunto de comprometimentos, podendo também fazer parte do processo de raciocínio meios-fim [BRA 90]. Neste sentido, intenções podem funcionar como planos parciais, de modo que, caso o mundo mude de maneira sutil, seja possível adaptar o plano ao invés de se fazer um re-planejamento total.

Outros estados mentais: Alguns autores sugerem a utilização de outros estados mentais além de crenças, desejos e intenções. Dentre estes diferentes estados mentais, alguns são propostos visando facilitar a compreensão e otimizar o funcionamento dos modelos para os quais foram definidos. Exemplos disto são os objetivos e planos do InterRAP [MÜL 96], e comprometimentos e capacidades da programação orientada a agentes [SHO 93], entre outros.

Devido ao fato de que o conjunto de desejos de um agente é potencialmente inconsistente, alguns autores sugerem a utilização de um estado mental auxiliar entre desejos e intenções. Este

estado intermediário poderia, por exemplo, representar um subconjunto internamente consistente dos desejos do agente (*i.e.* cada desejo no conjunto é consistente com os demais) e possivelmente externamente consistente (*i.e.* os desejos são consistentes com as crenças do agente).

A habilidade de um agente em decidir como atingir seus objetivos é essencial no processo deliberativo [BRA 88]. Conseqüentemente, alguns autores [RAO 91b] modelam os planos de um agente como uma seqüência de intenções adotadas em um determinado momento. A origem destas intenções geralmente é uma biblioteca de planos, porém, nada impede que o planejamento das intenções ocorra em tempo de execução dando mais flexibilidade ao agente.

2.3 Teorias BDI

Esta seção resume as principais teorias utilizadas no embasamento de modelos computacionais de agentes BDI. Especificamente, descreve-se a teoria de intencionalidade de Cohen e Levesque [COH 90] e a Lógica BDI de Árvores Computacionais de Rao e Georgeff [RAO 95b], em ordem evolucionária.

2.3.1 Modelo de Cohen e Levesque

Bratman definiu conceitos filosóficos que buscam explicar por que os seres humanos se comportam de maneira eficiente no mundo real através de três estados mentais, definindo informalmente as relações entre estes estados e enumerando diversas características que tais estados devem possuir. Estes conceitos foram utilizados por pesquisadores em Ciência da Computação na tentativa de emular este processo no sentido contrário, isto é, definir processos centrados em Crenças, Desejos e Intenções que permitiriam que um agente operasse no mundo real de modo satisfatório. A fim de utilizar de modo preciso este modelo de raciocínio em Ciência da Computação, uma definição formal de como funcionam estes conceitos filosóficos foi criada por Cohen e Levesque [COH 90]. O objetivo principal dos autores foi definir formalmente intenções que satisfizessem um conjunto de papéis funcionais tal como definido por Bratman e tratar alguns dos possíveis efeitos colaterais previstos pelo mesmo autor [BRA 99a, BRA 99b]. Além disto, os autores buscam estabelecer os fundamentos de uma teoria de interação entre agentes baseada em atos de fala.

Definições Básicas

A teoria descrita pelos autores é fundamentada em uma forma de lógica modal que utiliza semântica de mundos possíveis [HAL 92], com quatro operadores modais principais, os conectivos da lógica clássica ($\neg$, $\wedge$, $\vee$, $\rightarrow$), modalidades temporais ($\Diamond$ e $\Box$) e o quantificador existencial ($\exists$). Os principais operadores modais são descritos abaixo:

- **BEL** é utilizado para descrever crenças, onde $(BEL\ Agt\ P)$ significa que a fórmula P é conseqüência das crenças do agente Agt.
- **GOAL** é utilizada para representar objetivos, que por sua vez são utilizados para representar desejos, onde $(GOAL\ Agt\ P)$ significa que a fórmula P é conseqüência dos objetivos do agente Agt. Alternativamente, considerando a semântica de mundos possíveis, isto significa que a fórmula P é verdadeira em todos os mundos acessíveis a partir do mundo atual que são compatíveis com os objetivos do agente;
- **HAPPENS** é utilizado em conjunto com **DONE** no raciocínio temporal, onde $(HAPPENS\ ActExpression)$ significa que $ActExpression$[3] acontece a seguir;
- **DONE** complementa **HAPPENS** no raciocínio temporal, onde $(DONE\ ActExpression)$ significa que $ActExpression$ acabou de acontecer.

Além dos operadores básicos, a noção de tempo é representada por números que denotam o momento onde uma determinada fórmula é verdadeira para o agente, por exemplo, a proposição $(At\ Smith\ Brazil) \wedge 3$ significa que o "agente $Smith$ estar no $Brazil$" somente é verdadeira no tempo 3. Outra característica do raciocínio temporal nesta teoria é que são permitidos eventos primitivos. Ações podem ser eventos primitivos, representados por uma variável de ação $(a, b, \dots)$, ou uma expressão de ação, que pode ser:

- Uma variável de ação;
- A ação seqüencial, $ActExpression_1; ActExpression_2$, que significa que a primeira expressão de ação ocorre e a segunda expressão de ação ocorre logo a seguir $(HAPPENS)$;
- A ação de escolha não determinística, $ActExpression_1 | ActExpression_2$, que significa que ou a primeira ou a segunda expressão de ação ocorrerá a seguir $(HAPPENS)$;
- A ação de teste, $P?$, que é utilizada para restringir o raciocínio de um agente aos mundos possíveis onde a fórmula P é verdadeira;
- A ação de iteração, $ActExpression*$, que significa que $ActExpression$ ocorre múltiplas vezes como se diversas ocorrências da ação seqüencial ";" tivessem sido utilizadas, $ActExpression$ irá ocorrer pelo menos uma vez.

Estas construções relativas a ações são utilizadas no raciocínio sobre uma variedade de fenômenos do mundo real relacionados aos atos do agente e a eventos ocorrendo fora da esfera de influência do próprio agente.

[3] Onde $ActExpression$ pode ser qualquer expressão de ação.

Crenças

As crenças do agente caracterizam aquilo em que o agente acredita implicitamente [COH 90], isto significa que se tudo em que o agente acredita é verdadeiro, então as crenças do agente representariam completamente o estado do mundo em um dado momento. Outra característica da natureza implícita das crenças nesta teoria é que o agente acredita em todas as ramificações das suas crenças, sem a necessidade de raciocinar explicitamente a fim de inferir as conseqüências de suas crenças. Esta definição de crenças é utilizada pelos autores para definir as noções de conhecimento e competência, representadas pelos operadores modais **KNOW** e **COMPETENT**. Diz-se que um agente A conhece a fórmula P (representado na forma $(KNOW\ A\ P)$), se o agente acredita que P é verdadeira e P é, de fato, verdadeira. Além disto, diz-se que um agente A é competente a respeito de uma fórmula P (representado na forma $(COMPETENT\ A\ P)$), se sempre que o agente acredita que P é verdadeiro, P é realmente verdadeiro.

Objetivos

Os autores constroem as noções de objetivos e intenções de Bratman refinando o conceito básico de objetivo (**GOAL**). Este conceito é definido de maneira similar às crenças na medida em que o operador **GOAL** define o que é implícito em relação aos objetivos do agente. De maneira semelhante às crenças, um agente tem como objetivo todas as conseqüências de seus objetivos explícitos. Considerando a natureza implícita de ambos os operadores **BEL** e **GOAL**, e o fato de que um agente escolhe o mundo onde ele se encontra, então quando o agente acredita que uma propriedade P é verdadeira ele também escolheu P como sendo verdadeira.

Utilizando esta definição de objetivo, os autores refinam a noção de objetivo definindo um *objetivo de realização* $(A - GOAL\ A\ P)$ como sendo uma propriedade P que atualmente é falsa, mas que eventualmente será verdadeira. O objetivo de realização captura uma propriedade importante dos desejos definidos por Bratman [BRA 99a, BRA 99b] que diz que um agente não pode desejar algo que ele já saiba que é verdadeiro ou que nunca irá se tornar verdadeiro. Os autores prosseguem na definição de um objetivo que implica comprometimento, que é chamado de objetivo persistente, onde $(P - GOAL\ A\ P)$ significa que o agente A tem P como um objetivo persistente até que o agente satisfaça P ou acredite que P é impossível. Um aspecto importante da definição de objetivos persistentes é que eles são fechados em relação à equivalência lógica, o que será importante quando os autores tiverem que tratar do Problema do Pacote (Problema 2.4). Esta propriedade é importante, na medida que, se os objetivos de um agente fossem fechados em relação à implicação ou outro conectivo lógico que pudesse ser utilizado na construção da implicação, então seria possível que, um agente que possui um objetivo com conseqüências indesejadas tivesse como objetivo, por implicação lógica, as conseqüências indesejadas. Sendo os

objetivos persistentes fechados apenas em relação à equivalência lógica, não é possível que novos objetivos sejam derivados apenas por propriedades lógicas.

Intenções

Tendo formalizado diversas noções de objetivos e analisando suas conseqüências lógicas, os autores definem dois tipos de intenções:

- $(INTEND_1\ A\ P)$ significa que o agente A está comprometido, através de um objetivo persistente, a acreditar que ele está prestes a realizar P, e então irá realizá-lo;
- $(INTEND_2\ A\ P)$ significa que o agente A está comprometido, novamente através de um objetivo persistente, a acreditar que ele irá fazer algo (ele pode não ter nenhum conhecimento do que ele irá fazer além do primeiro passo) que irá resultar em P sendo verdadeiro como conseqüência.

Estas duas definições capturam os dois propósitos das intenções previstos por Bratman. $INTEND_2$ representa as intenções como componentes de planos de alto-nível, que serão utilizados pelo agente para iniciar a sua ação na direção de um determinado objetivo mesmo que a maneira exata de como alcançar tal objetivo não seja completamente conhecida. $INTEND_1$ representa os componentes dos planos de baixo nível, que estão mais próximos de ações concretas.

Até a definição básica das intenções ($INTEND_1$ e $INTEND_2$), objetivos e intenções são definidos de uma forma dita *fanática*, isto é, o agente irá querer realizar algo até que ele tenha conseguido realizá-lo ou até que ele acredite que é impossível torná-lo verdadeiro. Relativizando os objetivos (**GOAL**) através da dependência por uma condição, os autores especificam um critério além da impossibilidade para que um agente desista de um objetivo. Desta forma, estas definições capturam a maioria, senão todas, as propriedades antevistas por Bratman, entretanto, os autores não especificam o processo pelo qual um agente decide quais desejos ele irá perseguir, e como ele lida com desejos conflitantes.

2.3.2 Lógica BDI de Rao e Georgeff

Diversos autores propuseram uma série de sistemas lógicos visando a construção de agentes racionais. Entretanto a maioria destes sistemas captura apenas um subconjunto das propriedades de racionalidade necessárias para um agente, carecendo, portanto, de uma axiomatização completa. Dentre estes sistemas, aqueles definidos a fim de descrever o funcionamento de agentes BDI também partilham destas limitações. A fim de enfrentar a falta de uma axiomatização correta e completa para agentes BDI, Rao e Georgeff [RAO 91b, RAO 95a] definiram uma família de lógicas e procedimentos de decisão para descrever agentes BDI. Estas lógicas são chamadas de

Lógicas de Árvores Computacionais BDI (*BDI Computation Tree Logics*), sendo suas instâncias mais importantes chamadas de BDI_{CTL} e BDI_{CTL*}. Este conjunto de lógicas foi largamente utilizado na pesquisa sobre agentes BDI [RAO 95b, D'I 98b, WOO 00b, SCH 01, BOR 03] tendo recebido o nome de *Lógicas BDI*.

De Estruturas Temporais Ramificadas a Mundos Possíveis

Os sistemas lógicos assim definidos tinham o objetivo de relacionar métodos clássicos de definição de racionalidade como teoria da decisão. Desta forma, os autores criaram um mapeamento a partir da Lógica Temporal Ramificada (*Branching Temporal Logic*) [EME 90] e de Árvores de Decisão [JON 77] em um modelo representando crenças, desejos e intenções na forma de relações de acessibilidade em um conjunto de mundos possíveis.

Informalmente, Árvores de Decisão são compostas de vértices representando estados de mundo, arcos nesta árvore representam caminhos de execução alternativos. Transições de estado podem ser ações realizadas pelo sistema ou eventos primitivos ocorrendo no ambiente, ou mesmo ambos. Vértices resultantes de ações são chamados de *Vértices de Escolha* e vértices resultantes de eventos são chamados de *Vértices de Acaso*. As folhas na árvore são chamadas de vértices terminais. Vértices de Acaso podem ser rotulados com probabilidades representadas com valores em $\mathbb{R}$. Uma função de recompensa atribui valores de recompensa em $\mathbb{R}$ a vértices terminais. Uma função de deliberação escolhe o caminho a partir da raiz até Vértices terminais com recompensa maior.

A relação entre Árvores de Decisão e Lógicas BDI é definida utilizando vértices de acaso como ponto de referência. A Árvore de Decisão é dividida em várias árvores, que não mais possuem transições de acaso, cada uma representando um mundo possível, onde a probabilidade de o agente estar em um dado mundo é a probabilidade da transição de acaso que gerou aquela árvore. Estas árvores são então separadas em uma relação de acessibilidade de crenças utilizando as probabilidades da árvore inicial, e uma relação de acessibilidade de desejos utilizando as recompensas da árvore inicial. Diz-se que os caminhos gerados pela função de deliberação são a relação de acessibilidade das intenções.

Sintaxe e Semântica

Considerando que as Lógicas de Árvores Computacionais BDI são extensões da Lógica Temporal Ramificada de Emerson [EME 90], elas incluem um conjunto de proposições primitivas $\Phi \neq \varnothing$ e um conjunto de conectivos básicos e operadores dos quais outros podem ser definidos. Os conectivos proposicionais utilizados nesta lógica são $\vee$ e $\neg$, dos quais são definidos $\wedge$, $\supset$ e $\equiv$. Os Operadores Temporais Lineares utilizados nesta lógica são $\mathbf{X}$ (a seguir), $\mathbf{U}$ (até) e $\mathbf{F}$ (em algum momento no futuro (*eventually*), similar ao operador $\Diamond$). A partir destes operadores temporais

são definidos $\mathbf{G}$ (em todos os momentos no futuro ou sempre, equivalente ao operador $\Box$) e $\mathbf{B}$ (antes de). Um quantificador de caminho utilizado nesta lógica é $\mathbf{E}$ (algum caminho no futuro ou opcionalmente), do qual é definido $\mathbf{A}$ (todos os caminhos no futuro ou inevitavelmente). Finalmente, a lógica é estendida com os operadores modais $\mathbf{BEL}$ (o agente acredita), $\mathbf{DES}$ (o agente deseja) e $\mathbf{INTEND}$ (o agente tem a intenção).

Fórmulas bem formadas nesta lógica são definidas utilizando proposições atômicas e os conectivos e operadores definidos anteriormente. Estas fórmulas são divididas em dois tipos: *fórmulas de estado*, que expressam a verdade em um estado ou mundo em particular, e *fórmulas de caminho*, que expressam a verdade em um mundo em particular ou ao longo de um caminho em árvores de mundos possíveis. Um conjunto de regras para a definição da classe de fórmulas válidas na linguagem é definido abaixo:

S1 cada proposição atômica ϕ é uma fórmula de estado;

S2 se ϕ e ψ são fórmulas de estado, então $\neg\phi$ e $\phi \wedge \psi$ também o são;

S3 se ϕ é uma fórmula de caminho então $\mathbf{A}\phi$ e $\mathbf{E}\phi$ são fórmulas de estado;

S4 se ϕ é uma fórmula de estado então $\mathbf{BEL}(\phi)$, $\mathbf{DES}(\phi)$ e $\mathbf{INTEND}(\phi)$ são fórmulas de estado;

P0 se ϕ e ψ são fórmulas de estado então $\mathbf{X}\psi$ e $\phi\mathbf{U}\psi$ são fórmulas de caminho;

P1 cada fórmula de estado é também uma fórmula de caminho;

P2 se ϕ e ψ são fórmulas de caminho então $\neg\phi$ e $\phi \wedge \psi$ também o são;

P3 se ϕ e ψ são fórmulas de caminho então $\mathbf{X}\phi$ e $\phi\mathbf{U}\psi$ também o são.

A classe de fórmulas válidas para a BDI_{CTL} é definida utilizando as regras S1-S4 e P0, limitando portanto a composição de fórmulas de caminho a declarações em relação a fórmulas de estado (utilizando especificamente os operadores $\mathbf{X}$ e $\mathbf{U}$). A classe de fórmulas válidas para BDI_{CTL*} é definida utilizando as regras S1-S4 e P1-P3.

A atribuição de valores verdade nesta lógica é dada por uma semântica de mundos possíveis onde cada mundo possível é uma estrutura de árvores com um passado linear infinito e um futuro ramificado. O futuro ramificado representa os cursos de evento os quais o agente pode escolher quando em um mundo em particular. A relação de acessibilidade das crenças mencionada anteriormente mapeia um estado de mundo possível em outros mundos possíveis. As relações de acessibilidade de desejos e intenções também fazem um mapeamento semelhante. Além disto, os autores provêem uma série de procedimentos de decisão para a verificação da validade e satisfação

de fórmulas nesta lógica. Estes procedimentos são baseados na prova da existência de uma propriedade chamada de propriedade de modelo pequeno (*small model property*) para uma dada fórmula.

Propriedades Modais BDI

Além das definições semânticas, este framework lógico provê uma série de axiomatizações possíveis de modo que diferentes tipos de lógicas BDI possam ser definidas. Além dos axiomas herdados da Lógica Temporal Ramificada, diversos axiomas modais aplicados às modalidades BDI podem ser incluídos no sistema lógico resultando em diferentes propriedades para o sistema lógico correspondente. O conjunto mínimo de axiomas nesta lógica é resultado da aplicação do axioma K para cada uma das modalidades. O axioma K é similar à regra de implicação lógica, aplicado à um operador modal. Este axioma estipula que, se um operador modal aplicado a uma implicação é verdadeiro e o mesmo operador aplicado à condição desta implicação também é verdadeiro, então o operador modal aplicado à conseqüência da implicação é verdadeiro por implicação. Isto resulta nos seguintes axiomas [COH 90]:

B-K $BEL(\phi) \wedge BEL(\phi \supset \psi) \supset BEL(\psi)$

D-K $DES(\phi) \wedge DES(\phi \supset \psi) \supset DES(\psi)$

I-K $INTEND(\phi) \wedge INTEND(\phi \supset \psi) \supset INTEND(\psi)$

Além destes axiomas, os autores fizeram experiências com a inclusão do axioma D, 4, 5 e a regra da generalização, resultando em diversas propriedades do modelo BDI definido por Cohen e Levesque [COH 89]. Em particular, a regra da generalização estipula que qualquer fórmula válida é necessariamente acreditada, desejada e almejada, resultando em:

B-Gen If $\vdash \phi$ then $\vdash BEL(\phi)$

D-Gen If $\vdash \phi$ then $\vdash DES(\phi)$

I-Gen If $\vdash \phi$ then $\vdash INTEND(\phi)$

O sistema modal *weak-S5* (axiomas D, 4 e 5) é utilizado nas crenças para proporcionar ao agente respectivamente, consistência e introspecção positiva e negativa. O axioma D de consistência estipula que se um agente acredita que uma propriedade é verdadeira, então ele não acredita que esta propriedade é falsa. O axioma 4 de introspecção positiva estipula que, se um agente acredita em uma propriedade, ele está ciente de sua crença, *i.e.* ele acredita que acredita na propriedade. A introspecção negativa funciona de forma semelhante, *i.e.* se um agente não acredita em uma propriedade, ele acredita que não acredita na propriedade.

B-D $BEL(\phi) \supset \neg BEL(\neg\phi)$

B-4 $BEL(\phi) \supset BEL(BEL(\phi))$

B-5 $\neg BEL(\phi) \supset BEL(\neg BEL(\phi))$

Finalmente o axioma D para consistência também é utilizado em relação aos desejos e intenções.

D-D $DES(\phi) \supset \neg DES(\neg\phi)$

I-D $INTEND(\phi) \supset \neg INTEND(\neg\phi)$

BDI_{CTL} e $\mathcal{LORA}$

O *framework* lógico BDI_{CTL} foi expandido posteriormente em um novo sistema chamado de $\mathcal{LORA}$ [WOO 00b]. Este *framework* introduz diversas mudanças e extensões, não obstante, diversas características da lógica BDI_{CTL} foram mantidas. Em relação à representação do tempo, a $\mathcal{LORA}$ utiliza uma noção similar de tempo discreto, ilimitado e ramificado no futuro, e linear no passado, entretanto o tempo passado é limitado por um ponto inicial.

Em relação à expressividade, a $\mathcal{LORA}$ estende o conjunto de operadores temporais com o operador $\mathbf{W}$, onde $\phi\mathbf{W}\psi$ significa ϕ a não ser que ψ. Ela também acrescenta a noção de ações e sua execução de forma similar ao sistema lógico de Cohen e Levesque [RAO 97]. Especificamente, ela utiliza as construções de ação seqüencial, $(a; a')$, ação de escolha não determinística $(a|a')$, ação de iteração $(a*)$ e ação de teste $(a?)$, onde o seu significado é o mesmo descrito na Seção 2.3.1.

2.4 Arquiteturas BDI

Como visto na Seção 2.2, a teoria de raciocínio prático de Bratman levou ao desenvolvimento do modelo BDI de agentes. Este modelo foi utilizado na criação de um série de trabalhos visando a sua implementação em sistemas computacionais reais. Esta seção descreve duas das mais notórias implementações de agentes BDI. A primeira é uma proposta do autor do modelo filosófico de como um agente BDI deve ser implementado. A segunda implementação foi utilizada como referência no desenvolvimento de diversos outros trabalhos, em particular trabalhos que lidam com a utilização de lógicas modais na descrição de comportamento de um sistema. No final desta seção, descreve-se um terceiro modelo teórico de agente BDI que utiliza uma abordagem diferente dos anteriores ao implementar o raciocínio não-monotônico do agente utilizando Programação em Lógica Estendida.

2.4.1 IRMA

A *Intelligent Resource-bounded Machine Architecture* (IRMA) [BRA 88] foi definida com o objetivo de demonstrar a viabilidade do modelo de raciocínio prático de Bratman. O objetivo da IRMA é prover um mecanismo de raciocínio para um agente que leva em consideração seus recursos limitados. Outro aspecto desta arquitetura é ser uma das primeiras[4] a incorporar intenções como estado mental fundamental no processo de raciocínio prático. O papel das intenções nesta arquitetura é estabilizar o processo de cumprimento de objetivos de longo prazo, além disto as intenções mantém um registro do progresso das atividades do agente, permitindo a ele modificar passos de um plano em execução sem ter que recomeçar as suas atividades.

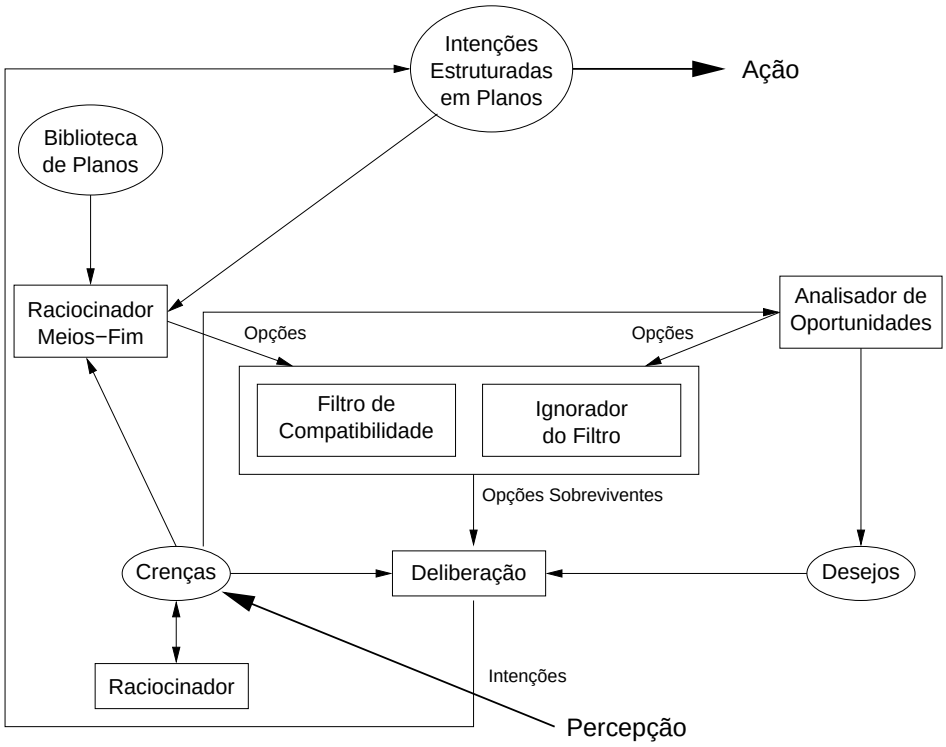

Figura 2.4: Estrutura interna da arquitetura IRMA.

A Figura 2.4 [BRA 88] representa a organização interna da IRMA e contém dois tipos básicos de entidades: processos (denotados por retângulos) e entidades de armazenamento (denotadas por elipses). As intenções de um agente IRMA são estruturadas em planos de alto nível que guiam a escolha dos planos propriamente ditos (ou planos como receitas), que estão armazenados em uma biblioteca de planos. O Analisador de Oportunidades reage a mudanças no ambiente criando opções de ação baseado em eventos não previstos pelo planejamento convencional, que

[4] Junto com o PRS (Seção 2.4.2).

é realizado pelo Raciocinador Meios-Fim. O Raciocinador Meios-Fim tem como entradas mais óbvias as Crenças do agente e os planos armazenados na Biblioteca de Planos. Além disto, de acordo com o modelo de raciocínio prático de Bratman, as Intenções têm o papel de restringir o espaço de busca por soluções no raciocínio meios-fim.

O Raciocinador Meios-fim e o Analisador de Oportunidades sugerem opções para o processo de filtragem, representado pelo Filtro de Compatibilidade e pelo Ignorador do Filtro. O Filtro de Compatibilidade verifica se as opções geradas são consistentes com as intenções adotadas atualmente, e as opções sobreviventes são passadas para o Processo de Deliberação, que pesa as novas opções entre si e as incorpora aos planos do agente. O Ignorador do Filtro foi incorporado ao processo de filtragem devido à possibilidade do agente ter conhecimento limitado, que cria situações onde certas alternativas seriam interessantes apesar da base de crenças indicar que há inconsistência nestas alternativas. Desta forma, mesmo que uma determinada opção seja eliminada pelo Filtro de Compatibilidade, é possível que ela ative uma regra do Ignorador do Filtro que a torne uma regra sobrevivente.

A arquitetura IRMA, apesar de ser bastante abstrata, serviu para delinear alguns dos problemas que as demais arquiteturas BDI deveriam resolver, tais como a falta de procedimentos para [BRA 88]:

- Propor novas opções em virtude da percepção de mudanças no ambiente;
- Avaliar opções conflitantes;
- Ignorar o filtro de compatibilidade.

Alguns dos conceitos definidos no trabalho original da IRMA foram testados no sistema *Tileworld* [POL 90], cujo objetivo principal era prover uma plataforma de testes de arquiteturas de agentes para estratégias de raciocínio em meta-nível. Essencialmente, o sistema *Tileworld* consiste de um agente robô simulado e um ambiente simulado, que é dinâmico e imprevisível. Tanto ambiente como agente foram projetados para serem altamente parametrizáveis de modo que diversas situações nas quais o agente poderia se encontrar pudessem ser testadas, e o comportamento de pares agente/ambiente pudesse ser avaliado. Os principais componentes examinados no agente *Tileworld* foram o Mecanismo de Filtragem (*Filtering Mechanism*), composto do Filtro de Compatibilidade e do Ignorador do Filtro (Figura 2.4), que é responsável por decidir se uma mudança no ambiente deve causar a reconsideração das intenções atuais do agente, e o processo de deliberação. Diversas estratégias de deliberação com complexidades crescentes foram testadas contra diferentes composições de ambientes, que variavam em diversos aspectos, a fim de testar a adequação do agente a diversas situações de ambiente. Os experimentos realizados na plataforma de testes *Tileworld* levaram à conclusão de que um Mecanismo de Filtragem que permite apenas

que oportunidades claras cheguem ao processo de deliberação é mais desejável tanto quanto o ambiente é mais dinâmico. Utilizando a noção de Cautela e Audácia [BRA 88], um agente Audacioso tende a apresentar um melhor desempenho que um agente Cauteloso em um ambiente dinâmico [POL 94]. Apesar de estas conclusões concordarem com as hipóteses apresentadas em trabalhos anteriores [BRA 88], os autores são cautelosos em não afirmar sua generalidade em relação a aplicações do mundo real, visto que o ambiente onde o agente foi embutido é altamente controlado [POL 94].

2.4.2 PRS

O *Procedural Reasoning System* (PRS) [GEO 87] foi criado visando uma arquitetura BDI que pudesse ser utilizada em aplicações do mundo real. Ele também objetivava suportar raciocínio tanto orientado a objetivos quanto reativo. O sistema foi utilizado inicialmente na implementação de um sistema de controle de tarefas em um simulador de espaçonaves da NASA. Esta seção irá descrever os componentes da arquitetura do PRS e o processo utilizado pelo interpretador do PRS nestes componentes.

Componentes do PRS

O PRS é organizado em uma série de componentes arquiteturais cujo controle é delegado ao interpretador, como pode ser visto na Figura 2.5 [GEO 89a]. Um agente ou módulo PRS consiste de um banco de dados contendo as crenças atuais do sistema em relação ao mundo, um conjunto de objetivos correntes, uma biblioteca de procedimentos ou planos cujos elementos descrevem seqüências de ações e testes que objetivam o cumprimento de determinados objetivos ou a reação a situações específicas [ING 92]. O PRS utiliza uma estrutura de intenções que consiste de um conjunto de planos escolhidos para execução, e um interpretador que funciona como mecanismo de inferência. O mecanismo de inferência manipula estes componentes e seleciona um plano adequado baseado nas crenças e objetivos do sistema, inserindo-o na estrutura de intenções e o executando.

A primeira implementação do PRS foi feita utilizando a linguagem LISP e executada em um processador específico para esta linguagem. Portanto, uma descrição de agente PRS é feita através de uma linguagem baseada em LISP. Desta forma, as formas sintáticas dos componentes PRS descritas nesta seção utilizarão a mesma sintaxe utilizada pelos autores nos seus trabalhos originais [GEO 89a].

Banco de Dados do Sistema (Crenças). O Banco de Dados do Sistema (*System Database*) pode ser visto como uma representação das crenças do sistema em relação ao ambiente, e é

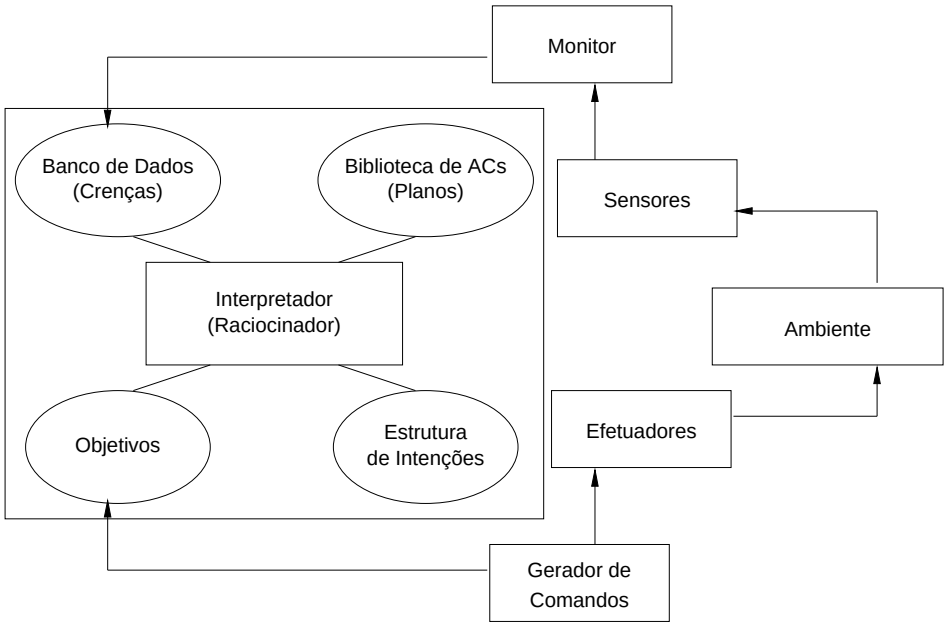

Figura 2.5: Estrutura da Arquitetura do PRS.

representado em predicados de lógica de primeira ordem [GEO 89a]. Estas crenças podem, inicialmente, estar relacionadas a propriedades constantes em relação ao mundo e ao domínio de aplicação de um sistema PRS. Entretanto elas podem evoluir ao longo da execução do sistema e incluir observações do agente sobre o mundo, ou mesmo conclusões derivadas pelo sistema utilizando o conhecimento contido no banco de dados, que pode mudar ao longo do tempo. O Banco de Dados do sistema é também o meio pelo qual o agente recebe informações sobre o mundo [ING 01]; o PRS assume que existe um processo automático que inclui no banco de dados novos fatos advindos do ambiente. Além de descrever o mundo, as crenças do agente podem se referir à estrutura do próprio agente e seus estados internos, incluindo crenças, objetivos e intenções. Este tipo de crença é chamado de expressão de *meta-nível*.

Objetivos (Desejos). Os Objetivos (*Goals*) de um agente PRS são expressos na forma de condições sobre um intervalo de tempo [GEO 89a], que no PRS é o equivalente a uma seqüência de estados de mundo, descritos como operações temporais sobre descrições de estado (ver estados de mundo e operações temporais em [RAO 91a]). Objetivos descrevem tarefas e comportamentos desejados, que podem ser expressos na lógica do PRS com os seguintes tipos de objetivo [GEO 89b, ING 92]:

- Atingir uma determinada condição `(! C)`;
- Testar uma determinada condição `(? C)`;

- Esperar até que um determinada condição seja verdadeira (`^C`);
- Manter uma condição verdadeira (`# C`);
- Declarar (*Assert*) uma condição como verdadeira (`→ C`);
- Cancelar (*Retract*) uma condição (`> C`);
- Concluir que uma determinada condição é verdadeira (`=> C`).

Objetivos são divididos em dois tipos: *intrínsecos* e *operacionais*. Objetivos intrínsecos são aqueles que advêm do processo de execução de planos na estrutura de intenções, isto é, não são objetivos intermediários de um objetivo principal. Em contrapartida, objetivos operacionais são aqueles que representam passos intermediários no cumprimento de um objetivo principal. Da mesma forma que as crenças, no PRS é possível se estabelecer objetivos de *meta-nível*, permitindo que se especifique objetivos em relação ao comportamento interno do sistema.

Áreas de Conhecimento (Planos). O conhecimento de como atingir um determinado objetivo no PRS é descrito por especificações de procedimento declarativas chamadas de Áreas de Conhecimento (*Knowledge Areas*) [GEO 89a, GEO 89b]. Estas especificações utilizam a noção de conhecimento procedural [GEO 86]. ACs são representadas por um *corpo* e uma condição de ativação. O corpo de uma AC pode ser visto como um plano ou um esquema de plano. Tal componente é representado por um grafo direcionado com um vértice de início e um ou mais vértices de final. Os arcos do grafo são rotulados com sub-objetivos a serem atingidos ao longo da execução do plano. A execução de uma AC é dita com sucesso quando os arcos que conectam um vértice de início a um vértice de final são percorridos e, no decorrer deste caminho, todos os sub-objetivos especificados foram satisfeitos. Desta forma, temos que o caminhamento do grafo de uma AC é multi-dimensional, uma vez que os sub-objetivos podem implicar na execução de outras ACs. É possível que algumas ACs não tenham um corpo, neste caso elas são chamadas de ACs primitivas, pois elas têm algum tipo de ação primitiva associada que é diretamente executável pelo sistema. O formalismo de construção de grafos utilizado pelos autores permite a criação de estruturas de controle do fluxo de execução, tais como desvios condicionais, iterações e recursões [GEO 89a, GEO 89b].

A condição de invocação de uma AC é dividida em duas partes, a Parte Gatilho (*Triggering Part*) e a Parte de Contexto (*Context Part*) [GEO 89a]. A Parte Gatilho de uma condição de invocação é uma expressão lógica que descreve os eventos que devem ocorrer para que a AC seja executada. Estes eventos podem consistir da aquisição de novos objetivos, caso em que o raciocínio utilizado é orientado a objetivos, ou da modificação das crenças, no caso em que o raciocínio utilizado é orientado a dados ou reativo. A parte de contexto da condição de invocação

especifica as condições que devem ser verdadeiras em relação ao estado atual do sistema para que a AC associada seja executada. ACs, da mesma forma que crenças e desejos, não se limitam a operar sobre o ambiente que cerca o agente, elas também podem ser utilizadas na manipulação de crenças, desejos e intenções do próprio PRS. Estas ACs são, portanto, chamadas de ACs de *Meta-Nível* [GEO 89a, ING 92]. Um dos objetivos de tais ACs é a modificação do comportamento padrão do interpretador do PRS no tratamento do raciocínio do agente. Isto pode incluir a modificação de planos durante a execução, o estabelecimento de novos objetivos ou mesmo a modificação das crenças durante a execução de uma AC de meta-nível.

Estrutura de Intenções. A estrutura de intenções contém todas as tarefas que o sistema escolheu para execução imediata ou posterior, estas tarefas são chamadas de intenções [GEO 89a, GEO 89b]. Uma intenção é composta de uma AC escolhida para cumprir um objetivo, junto com todas as sub-ACs que forem necessárias para que se complete a execução da AC inicial. As intenções na estrutura de intenções podem estar ativas, suspensas ou serem adiadas, esperando, por exemplo, que uma condição se torne verdadeira. Existe uma intenção propriamente dita para cada objetivo intrínseco que está sendo perseguido, e para cada intenção existe uma pilha de ACs a serem executadas de modo a atingi-la.

As ACs incluídas na estrutura de intenções estão parcialmente ordenadas com, possivelmente, mais de uma AC ocupando a raiz da estrutura de intenções. A ordem estabelecida para execução das ACs escolhidas é obedecida de forma que, para que ocorra a execução de uma AC subseqüente na estrutura, é necessário que as ACs anteriores sejam cumpridas ou retiradas. Não há diferenciação quanto à natureza das ACs dentro da estrutura de intenções, desta forma, ACs de meta-nível não são tratadas de modo diferente das ACs regulares no processamento. O comprometimento com as intenções [BRA 88] é implementado no PRS na medida em que, uma vez escolhida uma determinada AC para o cumprimento de um objetivo, não serão consideradas outras ACs cuja condição de ativação for satisfeita, a não ser que a AC atual se torne inviável [GEO 89a]. Ou seja, o PRS se compromete com o plano de ação determinado por uma AC escolhida, sendo que ele só irá considerar outras maneiras de cumprir com o objetivo caso a maneira inicial se torne impossível.

Interpretador do Sistema. O interpretador é o responsável pela interação dos componentes do PRS, sendo o seu processo de operação o mais simplificado possível de modo a garantir o menor tempo de reação possível no sistema [GEO 89a, GEO 89b, ING 01]. Considerando os objetivos e as crenças existentes em um dado momento, uma ou mais ACs podem se tornar passíveis de execução, sendo que uma ou mais destas serão escolhidas para serem adicionadas à estrutura de intenções (*i.e.* serem escolhidas para execução). Para verificar se as ACs serão executadas,

o interpretador utiliza apenas o processo de unificação das suas condições de execução com as crenças do sistema. Os autores salientam que, caso fosse utilizado qualquer outro processo de inferência mais complexo, não seria mais possível provar que o tempo de execução do processo de escolha de ACs é limitado [GEO 89a]. Para realizar processos de inferência mais complexos, *i.e.* que não consistam simplesmente da execução seqüencial dos passos das ACs selecionadas, devem ser utilizadas as ACs de meta-nível [GEO 89a]. A utilização destas ACs não viola a capacidade de reação do sistema, já que elas são tratadas da mesma forma que quaisquer outras ACs, sendo portanto possível que novas ACs tenham precedência sobre as ACs de meta-nível utilizadas para realizar inferências.

Funcionamento do PRS

O funcionamento do PRS foi projetado para ser o mais simples possível, permitindo um funcionamento pré-definido mesmo sem qualquer informação de alto-nível sobre o seu processo de raciocínio. Em contrapartida, é possível adicionar informações de meta-nível ao sistema a fim de refinar o comportamento do sistema, ao mesmo tempo em que se mantém o limite do tempo de reação do processo básico.

Ciclo do Interpretador. O principal processo que norteia o funcionamento do PRS é determinado pelo interpretador do sistema visto no centro da Figura 2.5. Este processo é responsável pela seleção de ACs de modo a reagir a mudanças no ambiente, cumprir com os objetivos estabelecidos pelo processo de aquisição e abandono de intenções, ou ainda para avançar na execução de uma AC previamente selecionada. Um esquema do processo de funcionamento do interpretador do PRS pode ser visto na Figura 2.6.

O interpretador do PRS inicia o seu ciclo de execução utilizando como base o conjunto de objetivos e crenças contidas na sua base de dados. As ACs conhecidas pelo sistema são analisadas tentando-se unificar as condições de invocação destas com as crenças e os objetivos estabelecidos; a partir desta unificação, tem-se um conjunto de ACs que podem ser executadas. Dentre estas ACs, algumas são selecionadas para serem de fato executadas. Estas ACs são inseridas na estrutura de intenções com base no seguinte critério: se a AC foi selecionada devido a aquisição de um novo objetivo intrínseco ou de uma nova crença, então esta AC será inserida na estrutura de intenções como uma nova intenção, caso contrário (*i.e.* a AC foi selecionada como resultado de um objetivo operacional), ela será inserida na pilha de ACs que compreendem a intenção correspondente. O passo seguinte é a escolha de uma intenção na raíz da estrutura de intenções para que seja executado mais um passo da mesma. Este passo pode ser o estabelecimento de um ou mais novos objetivos ou a execução de uma ação primitiva. A execução de uma ação primitiva pode resultar na mudança da base de crenças do sistema tanto por ser uma ação de meta-nível como

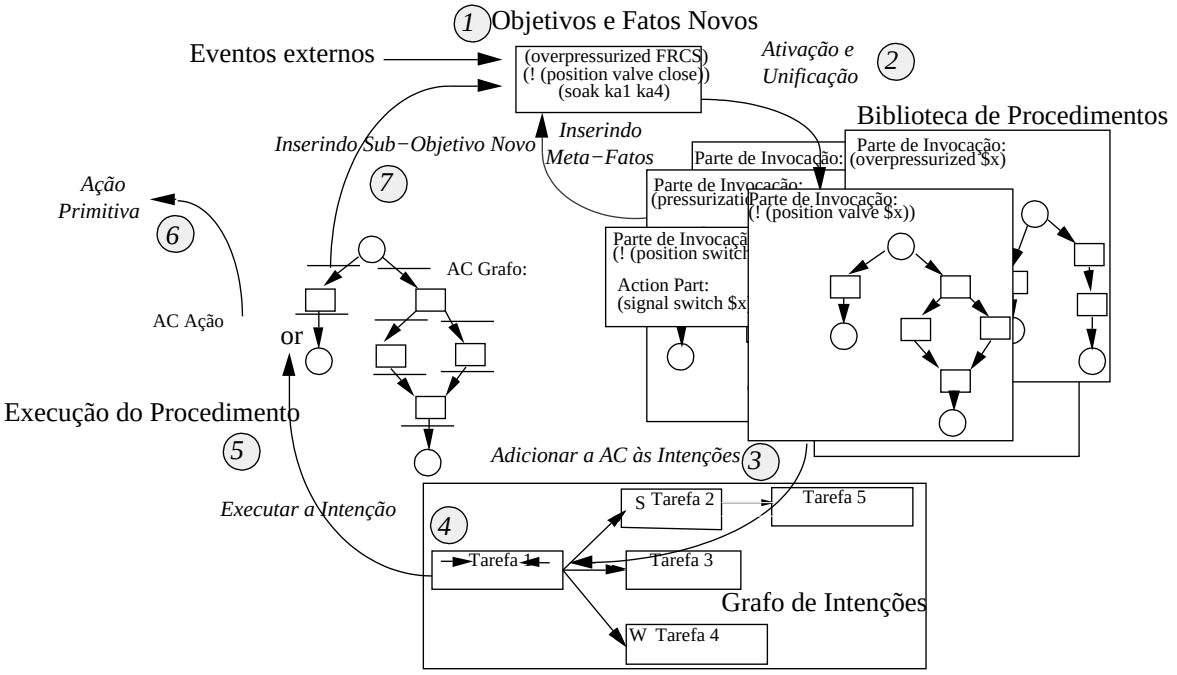

Figura 2.6: Interpretador do PRS

por modificar o mundo de forma que isto seja refletido nas crenças posteriormente. Ao final deste passo o ciclo se inicia novamente. A execução do interpretador do PRS pode ser resumida da seguinte maneira (Figura 2.6) [ING 96]:

1. O interpretador recebe novas crenças e objetivos;
2. Baseado nestes novos dados, o interpretador seleciona planos ou ACs apropriados;
3. As ACs selecionadas para execução serão inseridas na estrutura de intenções;
4. Dentre as raízes da estrutura de intenções, é selecionada uma intenção;
5. Um passo da AC selecionada no item anterior é executado;
6. Este passo pode ser a execução de uma ação primitiva;
7. Ou o estabelecimento de um novo objetivo.

Estados das Intenções. Uma intenção no PRS pode estar em três estados possíveis: ativa, suspensa ou suspensa condicionalmente [GEO 89a]. Uma intenção suspensa pode ser executada assim que ela se tornar uma raiz na estrutura de intenções. Uma intenção suspensa foi adotada pelo sistema, mas não há definição de quando ela deve ser executada, portanto ela deve ser

ativada explicitamente antes de ser executada. Uma intenção suspensa condicionalmente está suspensa temporariamente até que uma dada condição de ativação seja atingida. A suspensão de uma intenção pode ser realizada através de ACs de meta-nível. Quando uma intenção suspensa é reativada, é necessário que o sistema decida se a estrutura de intenções deve ser reorganizada ou não. Esta reorganização pode ser feita através de ACs de meta-nível, porém o comportamento padrão do interpretador do PRS é priorizar a execução das intenções reativadas a fim de minimizar o tempo de reação ao evento que causou tal reativação.

Estabelecimento e Abandono de Objetivos. Como descrito anteriormente, as intenções são inseridas na estrutura de intenções devido a mudanças nos objetivos ou nas crenças. Como resultado desta inserção, outros sub-objetivos também podem ser inseridos na estrutura de intenções. A possibilidade de falha destes objetivos deve ser tratada [GEO 89a]. Desta forma é necessário determinar como o sistema deve reagir caso uma falha ocorra, decidindo que curso de ação alternativo deve ser tomado a fim de se atingir o objetivo. Em contrapartida, deve-se determinar quando um objetivo que falhou se tornou impossível de ser atingido, *i.e.* que não existe outra maneira de atingi-lo. Dificilmente será possível provar em tempo de execução que um objetivo se tornou impossível [GEO 89a], portanto ACs de meta-nível devem ser incluídas no agente a fim de tratar da questão de caminhos de execução alternativos. Na falta destes, o PRS executará exatamente uma vez cada uma das ACs que poderiam cumprir um dado objetivo. Neste caso, ainda é possível incluir ACs de meta-nível para lidar com a falha de todos os planos disponíveis para cumprir um determinado objetivo, talvez re-considerando um plano específico cuja re-execução pareça mais promissora em uma segunda vez [GEO 89a].

O PRS foi utilizado como base para uma variedade de implementações do modelo BDI. Ele também evoluiu em diferentes trabalhos visando a resolução de suas limitações iniciais. Dois dos descendentes diretos mais notáveis do PRS são o dMARS e o AgentSpeak, cujos objetivos principais são, respectivamente, criar uma definição formal de um sistema baseado em agentes que fosse adequada para implementação, e definir formalmente uma linguagem de especificação de agentes e sua semântica. Estes trabalhos são descritos resumidamente nas Seções 2.4.3 e 2.4.4.

2.4.3 dMARS

O *distributed Multi-Agent Reasoning System* (dMARS) [D'I 98a] é uma implementação do PRS utilizada como referência para a criação de uma especificação formal do sistema na linguagem Z [POT 96]. Esta especificação modela uma implementação *ideal* do dMARS, declarando implicitamente que não existe garantia de equivalência entre a especificação e a implementação utilizada como base. O sistema descrito em [D'I 98a] utiliza a mesma noção de crenças do PRS, ou seja, a utilização de literais rasos (*ground*) da lógica de primeira ordem clássica. O dMARS reduz o

número de tipos possíveis de objetivos dos sete presentes no PRS para objetivos de realização (`!C`) e consulta (ou teste) (`?C`), redefinindo os objetivos de declaração e cancelamento na forma de ações internas (*internal actions*), chamadas, respectivamente, de ação de adição (`add`) e remoção (`remove`). No dMARS também é formalizada a interação com o ambiente através da noção de ações externas (*external actions*), que são utilizadas para realizar operações arbitrárias definidas pelo programador do sistema. Da mesma forma que o PRS original, as intenções no dMARS são representadas pelos planos adotados atualmente, *i.e.* as ACs escolhidas para execução. Os eventos que desencadeiam a adoção de uma nova intenção foram expandidos, além da adição e remoção de crenças e do estabelecimento de um novo objetivo o dMARS também considera a recepção de uma mensagem como evento de inclusão de intenções [D'I 98a]. A definição de planos no dMARS é realizada através de uma versão formalizada dos grafos de planos do PRS, sendo que os dois componentes da condição de invocação de um plano no PRS foram refinados em noções de relevância e aplicabilidade. Uma das maiores contribuições da formalização do dMARS é a definição sem ambigüidade de um algoritmo de interpretação.

2.4.4 AgentSpeak

A linguagem AgentSpeak (AgentSpeak(L)) [RAO 96] foi criada a fim de diminuir a distância entre a teoria e a prática de agentes BDI. Esta distância é resultante de diversos fatores [RAO 96, D'I 98b], como por exemplo o enfraquecimento da base teórica de um agente resultante da simplificação das implementações, ou a tênue relação entre as lógicas utilizadas nas teorias com problemas reais. Desta forma, o objetivo de diminuir esta distância deveria ser alcançado através de uma especificação formal dos agentes que seria utilizada para embasar uma implementação dos mesmos. Tal funcionamento deveria corresponder ao funcionamento das implementações do PRS em [GEO 89a] e do dMARS [D'I 98a]. AgentSpeak(L) é uma linguagem de programação baseada em um linguagem de primeira-ordem restrita que contém eventos e ações, onde os componentes do modelo BDI não são representados na forma de fórmulas modais explícitas [RAO 96]. Apesar de AgentSpeak(L) ter sido concebida com o intuito de prover um embasamento teórico mais forte para sistemas já implementados como o PRS e o dMARS, este objetivo não pode ser considerado completamente alcançado, visto que AgentSpeak(L) contém diversas simplificações em relação ao PRS. Estas simplificações também foram aplicadas ao dMARS, como pode ser observado na sua especificação formal [D'I 98a]. Além disto, quando o dMARS foi posteriormente especificado, nenhum esforço para relacioná-lo com o AgentSpeak(L) pôde ser percebido. Desta forma, estes dois trabalhos podem ser considerados mais como trabalhos paralelos do que formalismos equivalentes, como pode ser verificado em uma especificação refinada do AgentSpeak(L) [D'I 98b]. As simplificações mais significativas observadas no AgentSpeak(L) em relação ao PRS são:

- **Tipos de objetivos:** Em AgentSpeak(L) é possível declarar objetivos de alcance e teste de condições no mundo, além disto, é possível declarar e cancelar condições no mundo através das ações básicas do AgentSpeak(L). O que resulta na diminuição da expressividade da linguagem relacionada ao modelo formal de AgentSpeak(L), visto que não estão mais presentes objetivos de manutenção e espera por uma condição;

- **Componentes de Meta-nível:** Em AgentSpeak(L) não é possível especificar componentes de meta-nível como é feito no PRS, limitando a flexibilidade de definição do comportamento de um determinado agente.

2.4.5 X-BDI

O modelo X-BDI de agentes foi proposto com o objetivo de ser simultaneamente uma ferramenta de especificação formal e um *framework* executável de agentes [MÓR 99b, MÓR 99a] . Tal motivação se deve ao fato de que os modelos formais de agentes BDI são tradicionalmente especificados através de lógicas modais, as quais não possuem procedimentos de derivação corretos e completos. Além disto o tratamento computacional destas lógicas, apesar de decidível é um problema de alta complexidade [MÓR 99a].

Apesar de também ser uma ferramenta de desenvolvimento de agentes, o X-BDI é antes de tudo, um modelo formal de agentes. O fato de este modelo ser também uma ferramenta é um efeito colateral da sua implementação utilizando PROLOG, o resultado disto é que ambiente de execução (i.e. o PROLOG), também funciona como um ambiente de especificação de agentes. Desta forma o autor optou pela utilização da programação em lógica estendida com negação explícita (ELP - *Extended Logic Programming*) com a semântica bem-fundada estendida com a negação explícita (WSFX - *Well-Founded Semantics Extended for the explicit negation*) como ferramenta para a especificação e implementação de seu trabalho [MÓR 99a].

A construção do sistema em uma linguagem interpretada com uma implementação de referência significa que a especificação dos agentes também é executável, o que elimina o problema das discrepâncias na transposição dos modelos formais para uma implementação. Além disto a WSFX possui um procedimento de derivação correto e *top-down* chamado de derivação seletiva linear para programas estendidos (SLX - *Selected Linear resolution for extended programs*). Para uma explanação mais detalhada da ELP veja [ALF 96].

O X-BDI utiliza uma variedade de propriedades da ELP [ALF 96] para a construção do seu modelo BDI. Em especial a capacidade que este modelo lógico tem de lidar com contradições permite diversas formas de raciocínio não-monotônico, das quais são destacados o raciocínio revogável e o raciocínio abdutivo. Estes dois modelos de raciocínio são utilizados na construção das crenças, no caso do raciocínio revogável, e dos desejos e intenções, no caso do raciocínio

abdutivo. A representação do agente e suas propriedades é feita através de uma variante do Cálculo de Eventos (*Event Calculus*) adaptada para a ELP e modificada para permitir eventos simultâneos.

As crenças do agente são definidas como um programa consistente B. Os desejos de um agente são representados através de uma cláusula temporal suportada por uma conjunção de cláusulas que funciona como uma pré-condição para o desejo. As intenções do agente são geradas em tempo de execução e são divididas em dois tipos: intenções primárias e intenções relativas. Estas intenções estão relacionadas respectivamente com o comprometimento do agente e com a execução de um plano de ações, resultado do planejamento.

A principal contribuição do X-BDI é a especificação de um modelo executável de agente. Entretanto, O X-BDI não foi implementado com preocupações de performance e sua utilização em situações com tempo restrito é limitada.

2.5 Considerações

Este capítulo descreveu de maneira cronologicamente ordenada algumas teorias e implementações de agentes BDI. Pode-se dizer que o modelo BDI como teoria computacional foi proposto inicialmente com o trabalho descrito na Seção 2.3.1, que estabeleceu diversas propriedades e relações entre os componentes filosóficos da teoria de Bratman, sem no entanto desenvolver as especificidades da descrição do funcionamento contínuo de um agente BDI ao longo do tempo. Tal trabalho fez uso extensivo de lógica modal e suas propriedades no embasamento do formalismo e na ponderação sobre ele, uma característica que foi herdada pela lógica BDI_{CTL}, descrita na Seção 2.3.2. Esta lógica estendeu a anterior no raciocínio temporal utilizando um modelo infinito de tempo ramificado no futuro e adicionando a noção de utilidade utilizada em teoria da decisão. Na lógica BDI_{CTL}, o agente analisa a possibilidade de alcançar os seus desejos e a recompensa que eles trarão ao agente antes de escolher o conjunto de desejos que irá gerar as suas intenções. Desta forma a BDI_{CTL} diverge do modelo de Cohen e Levesque, na medida em que o agente irá ter o conhecimento do seu plano de ação antes de iniciar a execução das ações na direção de um objetivo. Esta abordagem, entretanto, assume que o agente irá ponderar sobre todos os caminhos de ação possíveis antes de tomar uma decisão através de um procedimento baseado em *tableaux*, o que é claramente indesejável considerando-se um agente com recursos limitados, tornando-a portanto inadequada para implementação, fato este corroborado pelos seus autores [RAO 97].

Um ponto importante explorado por ambas as teorias apresentadas na Seção 2.3 é a condição de adoção e abandono dos desejos. Desejos são adotados quando o agente acredita que o desejo é possível e são abandonados quando a agente acredita que eles são impossíveis. A noção de possibilidade de um desejo na teoria de Cohen e Levesque (Seção 2.3.1) está ligada ao con-

hecimento do agente da primeira ação que deve ser tomada na direção do cumprimento deste desejo. Esta noção deixa implícito que, ou o agente tem conhecimento das ações necessárias para o cumprimento de um desejo, ou o agente dispõe de uma maneira de "prever" que uma ação qualquer eventualmente irá concretizar um desejo. A noção de possibilidade na lógica BDI_{CTL} está associada a um caminhamento na árvore temporal do mundo onde o agente se encontra que leve o agente do estado de mundo atual até o estado onde o desejo do agente está satisfeito. Esta noção parte do pressuposto que o agente tem o conhecimento de todos os estados de mundo possíveis e do resultado de seqüências das suas ações na criação destes estados, o que é conhecido como onisciência do agente.

Considerando as arquiteturas descritas neste capítulo, observa-se que tanto IRMA (Seção 2.4.1) quanto PRS (Seção 2.4.2) e seus descendentes (*i.e.* dMARS e AgentSpeak(L)) não são derivados formalmente das teorias apresentadas na Seção 2.3. Entretanto, sua implementação serviu de base para a verificação empírica dos aspectos considerados na definição das teorias subseqüentes, *e.g.* a de Cohen e Levesque e a BDI_{CTL}. Na Seção 2.4.5, é descrito o X-BDI, que consiste de uma teoria e sua descrição no formalismo utilizado em sua implementação.

Uma característica presente nas arquiteturas IRMA, PRS, dMARS e AgentSpeak(L) é a presença de uma biblioteca de planos utilizada na formação das intenções. Na IRMA esta biblioteca "é considerada parte das crenças do agente, e especifica que ações o agente acredita serão úteis para atingir quais efeitos sob determinadas condições" [BRA 88]. Esta definição deixa implícito que o agente será capaz de selecionar ações a fim de concretizar seus objetivos, subentendendo um processo de planejamento e, ou o agente será capaz de seqüenciá-las para atingir objetivos de longo prazo, ou o agente já tem o conhecimento das seqüências de ações que ele irá utilizar na sua interação com o ambiente. No PRS a biblioteca de planos ou ACs "descreve como certas seqüências de ações e testes podem ser executadas para atingir determinados objetivos ou reagir a situações específicas" [GEO 89a]. A definição do PRS é mais restrita no sentido da capacidade de planejamento do agente, o que é resultado da crença dos seus autores de que a formação de planos em tempo de execução é inviável, e do seu objetivo de ter um limite de tempo definido para o processo de deliberação. O X-BDI parte de uma abordagem diferente para a formação das intenções utilizando um processo de abdução para formar uma teoria de ações em Cálculo de Eventos que irão constituir a base para as intenções adotadas pelo agente. Este processo é análogo a um processo de planejamento, uma vez que ele encadeia ações para estabelecer a ligação da situação atual do agente com os seus desejos.

A noção de possibilidade dos desejos em ambos os sistemas está ligada a existência de um plano em suas bibliotecas de planos que o satisfaça, considerando a situação atual do ambiente onde o agente se encontra. Um outro aspecto em comum é que a impossibilidade de um desejo só é considerada uma vez que o plano escolhido para satisfazê-lo falha, sendo que na definição

original do PRS [GEO 87] os autores afirmam que é extremamente difícil para o agente *provar* que um determinado objetivo é impossível com o conhecimento que o agente tem em tempo de execução. Em contrapartida, no X-BDI, a possibilidade de um desejo é validada pelo seu processo de geração de intenções, atrelando-a à existência de uma teoria de ações que corrobore com a validade do desejo em um futuro determinado. A impossibilidade de um desejo está ligada ao surgimento de alguma inconsistência na teoria de ações inferida pelo agente, o que pode ocorrer quando da modificação das crenças do agente em relação ao estado atual do mundo.

Analisando os trabalhos descritos neste capítulo e as conclusões estabelecidas nesta seção, pode-se observar que a formação de planos é um processo importante no contexto dos agentes BDI. Esta importância independe do momento em que estes planos são gerados, pois tanto no momento da criação do agente (*e.g.* IRMA e PRS), quanto no momento da execução do agente (*e.g.* X-BDI) a sua consistência é de importância fundamental para que o agente possa atingir os objetivos para o qual foi projetado. A prova de que um agente será capaz de atingir seus objetivos é um dos principais problemas em aberto na pesquisa em agentes deliberativos, tendo o problema sido descrito por Wooldridge como *Agent Design Problem* [WOO 00a]. Este problema é notório por sua alta complexidade para o caso geral, resultando em modelos de agente que, ou delegam esta verificação para o seu projetista [GEO 89a], ou realizam esta verificação antes da execução do agente [BOR 03]. Entretanto, a pesquisa na formação de planos utilizando o formalismo de planejamento proposicional [NEB 00] tem observado a criação de algoritmos com excelente desempenho em diversos domínios. Desta forma se for possível utilizar tal formalismo no processo de raciocínio meios-fim de agentes deliberativos, é possível aproximar a classe de problemas tratáveis por estes formalismos.

Capítulo 3

Planejamento Proposicional

Como visto no Capítulo 2, o raciocínio meios-fim, *i.e.* a habilidade de decidir que ações executar a fim de realizar um conjunto de objetivos, é um componente fundamental de qualquer agente racional [BRA 87, BRA 99a]. Este tipo de raciocínio envolve a formação de planos que serão utilizados pelo agente a fim de atingir seus objetivos. A formação dos planos pode ocorrer de maneira estática, antes da execução do agente, ou em tempo de execução. Em ambos os casos é necessário que estes planos estejam corretos no sentido de permitir ao agente atingir seus objetivos, o que é chamado de *Agent Design Problem* [WOO 00a]. Este não é um problema trivial, tendo complexidade PSPACE [PAP 94] para o caso geral [WOO 00a].

O processo de formação de planos é comumente realizado por sistemas de planejamento [RUS 94], que têm sua origem reconhecida no sistema STRIPS (*Stanford Research Institute Problem Solver*) [FIK 71], cuja contribuição mais perene é a linguagem de especificação de problemas. Este tipo de sistema representa uma abordagem bastante utilizada na implementação de agentes deliberativos [BRA 99a], além de ser útil na resolução de problemas em diversas outras áreas, por exemplo escalonamento [SMI 99]. Algoritmos de planejamento têm sido um dos principais objetivos da pesquisa em IA [RUS 94] pois o processo de formação e execução de planos é considerado fundamental no funcionamento de agentes inteligentes [BRA 88]. Genericamente, um problema de planejamento é definido por três componentes [WEL 99]:

- Uma descrição formal do *estado inicial*;
- Uma descrição formal dos *objetivos* que se deseja concretizar;
- Uma descrição formal das *ações que podem ser realizadas*.

Estes componentes serão fornecidos ao sistema de planejamento, que deverá gerar um conjunto de ações com alguma relação de ordem, que, quando aplicadas a um mundo onde a descrição do estado inicial é verdadeira, deverão tornar a descrição de objetivo verdadeira. O formalismo

utilizado para representar problemas de planejamento neste trabalho será melhor explorado na Seção 3.1.

É um fato conhecido que o planejamento é indecidível [CHA 87] e os problemas de planejamento, no caso geral, têm complexidade PSPACE [BYL 94]. Entretanto, restrições na linguagem de descrição dos problemas podem torná-los computáveis, ou até mesmo reduzi-los à complexidade polinomial, mesmo que limitando fortemente a expressividade da linguagem [BYL 94]. Apesar das características de complexidade provadas para o caso geral dos problemas de planejamento, avanços recentes na pesquisa em planejamento levaram à criação de algoritmos de planejamento cujo desempenho para diversas classes de problemas é significativamente melhor do que aproximações anteriores [WEL 99, NEB 00]. Os algoritmos referidos pertencem a duas classes:

- Algoritmos baseados no Graphplan [BLU 97];
- Algoritmos baseados na compilação do problema de planejamento em uma fórmula cuja satisfação é testada (SAT) [KAU 92].

Este trabalho irá se concentrar na extensão de um modelo de agentes BDI, a fim de dotá-lo da capacidade de utilizar sistemas de planejamento proposicionais para a realização do raciocínio meios-fim utilizado no processo de seleção de intenções. A utilização de planejadores proposicionais é motivada pelos avanços em termos de algoritmos de planejamento ocorridos recentemente. O algoritmo escolhido para a implementação do protótipo é o Graphplan. Sua escolha se deve ao fato de que até onde se sabe, existe uma variedade maior de trabalhos relacionados ao seu aprimoramento, *e.g.* planejamento em tempo real [SMI 99], inferência de mais informações no grafo de planejamento [HOF 99, LON 99], entre outros, que podem ser incorporadas em uma implementação do algoritmo.

Este capítulo aborda os algoritmos de planejamento utilizados na implementação realizada para este trabalho. A Seção 3.1 descreve o formalismo que foi utilizado para definir o domínios dos problemas tratados pelo Graphplan. A Seção 3.3 descreve o algoritmo Graphplan básico, suas principais fases de processamento e estruturas de dados, e é finalizada com algumas definições em relação a garantias que o Graphplan fornece para o término do algoritmo.

3.1 Formalismos de Planejamento Proposicional

Esta seção descreve o formalismo utilizado pelos planejadores considerados neste trabalho. Este formalismo é baseado na descrição presente em [NEB 00], sendo, de acordo com o referido autor, um formalismo $\mathcal{S}_{\mathcal{IL}}$, isto é, o formalismo STRIPS básico acrescido da possibilidade de especificações incompletas do estado inicial e da utilização de literais na descrição dos estados de mundo. A fim de manter este trabalho auto-contido, as definições relevantes serão replicadas a

seguir. É importante salientar que o formalismo definido por Nebel [NEB 00] é mais abrangente, porém, como o objetivo deste trabalho não é um estudo detalhado de formalismos de planejamento, utilizaremos uma versão mais simples de tal formalismo. Em especial, utiliza-se uma linguagem de lógica proposicional com variáveis apenas nas especificações de operadores, além de não permitirmos efeitos condicionais nos mesmos.

Considera-se inicialmente o alfabeto do formalismo. Define-se como $\mathbf{\Sigma}$ o conjunto contável infinito de *átomos proposicionais* ou *variáveis proposicionais* e Σ qualquer subconjunto finito de $\mathbf{\Sigma}$. A partir disto define-se como $\hat{\Sigma}$ o conjunto de *literais* sobre Σ, *i.e.* átomos e átomos negados. A *linguagem de lógica proposicional* sobre o conectivo lógico $\wedge$ e os átomos proposicionais Σ é denotada por $\mathcal{L}_\Sigma$, desta forma, consistindo apenas de conjunções de literais. Considerando um conjunto L de literais, $neg(L)$ e $pos(L)$ são respectivamente o subconjunto de literais negativos e positivos de L.

Um *estado* s é uma *atribuição-verdade* para os átomos em Σ, e uma *especificação de estado* S é um subconjunto de $\hat{\Sigma}$, ou seja, é uma teoria lógica composta apenas de literais. S é dita consistente se ela não contém literais complementares, e dita completa se $\forall p(p \in \Sigma \rightarrow (p \in S \vee \neg p \in S))$. Uma especificação de estado representa todos os estados s tais que $S \models s$, ou seja, estes estados são os modelos possíveis para S; quando uma especificação de estado é completa, ela possui apenas um modelo.

Operadores são pares $o = \langle pre, post \rangle$, onde $pre(o) \in \mathcal{L}_\Sigma$ e $post(o) \in \mathcal{L}_\Sigma$ denotam respectivamente as pré-condições e as pós-condições (ou efeitos) do operador o. No Exemplo 3.1 um domínio simples é descrito com este formalismo.

Exemplo 3.1 (Robô-Carteiro) *Consideramos um domínio onde existe um robô-carteiro, que, quando recebe pacotes, os envia ao local correto, e quando está sem bateria, se recarrega. O estado do mundo gira em torno da presença ou não de pacotes e da carga do robô. Como conjunto de átomos, temos:*

$$\Sigma = \{batt, package\}$$

Como operador de envio dos pacotes temos:

$$post = \langle \{package, batt\}, \{\neg package, \neg batt\} \rangle$$

Que essencialmente significa que ao enviar um pacote, o robô gasta sua bateria. Como operador de recarga do robô temos:

$$recharge = \langle \{\neg batt\}, \{batt\} \rangle$$

O resultado da aplicação de um operador em um estado de mundo é dado pela função de transição de estado descrita na Definição 3.1. Esta função define que, se as pré-condições de

um operador são válidas e seus efeitos não resultam em contradição, então os literais declarados como negativos nos efeitos do operador se tornam falsos e os literais declarados como positivos se tornam verdadeiros logo após a aplicação do operador ao estado.

Definição 3.1 (Função de Transição de Estado) *Uma Função de Transição de Estado θ_o, induzida pelo operador o é definida da seguinte forma:*

$$\begin{aligned} \theta_o : \quad & 2^{\Sigma} \rightarrow 2^{\Sigma} \\ \theta_o(s) \quad = \quad & \begin{cases} s - \neg neg(post(o)) \cup pos(post(o)) & \textit{se } s \models pre(o) \textit{ e } post(o) \not\models \bot \\ \textit{indefinido} & \textit{caso contrário} \end{cases} \end{aligned}$$

A aplicação de operadores sobre especificações de estado resulta em novas especificações de estado, como definido pela função R (Definição 3.2), que define o resultado da aplicação de um operador o de um conjunto de operadores $\mathbf{O}$ sobre uma especificação de estado.

Definição 3.2 (Função R) *O resultado da aplicação de um operador o sobre uma especificação de estado S é definido por:*

$$\begin{aligned} R : \quad & 2^{\hat{\Sigma}} \times \mathbf{O} \rightarrow 2^{\hat{\Sigma}} \\ R(S,o) \quad = \quad & \begin{cases} S - \neg neg(post(o)) \cup pos(post(o)) & \textit{se } S \models pre(o) \textit{ e } S \not\models \bot \textit{ e } post(o) \not\models \bot \\ \bot & \textit{caso contrário} \end{cases} \end{aligned}$$

A aplicação de um operador o sobre uma especificação de estado S é caracterizada por uma função R, cujo resultado é $S - \neg post(o) \cup post(o)$, se as pré-condições $pre(o)$ forem verdadeiras em S.

Ou seja, a aplicação de um operador o sobre uma especificação de estado resulta em um novo estado onde as pós-condições negativas do operador deixam de ser verdadeiras e as pós-condições positivas passam a ser verdadeiras. Utilizando-se a função R e considerando-se $\mathbf{O}^*$ como o conjunto de seqüências finitas de operadores, tem-se que os elementos Δ de $\mathbf{O}^*$ são chamados de planos. Desta forma, é possível definir recursivamente uma função da aplicação de Δ sobre uma descrição inicial de estado S da seguinte forma:

Definição 3.3 (Resultado de um plano) *O resultado da aplicação sucessiva de operadores sobre um estado inicial é definido por:*

$$\begin{aligned} Res : \quad & 2^{\hat{\Sigma}} \times \mathbf{O}^* \rightarrow 2^{\hat{\Sigma}} \\ Res(S, \langle\rangle) \quad = \quad & S \\ Res(S, \langle o_1, o_2, \ldots, o_n \rangle) \quad = \quad & Res(R(S, o_1), \langle o_2, \ldots, o_n \rangle) \end{aligned}$$

Utilizando-se a função Res, define-se:

Definição 3.4 (Instância de Planejamento) *Uma* instância de planejamento *é uma tupla* $\Pi = \langle \Xi, \mathbf{I}, \mathbf{G} \rangle$*, onde:*

- $\Xi = \langle \Sigma, \mathbf{O} \rangle$ *é a estrutura do domínio, que consiste de um conjunto finito de átomos proposicionais* Σ *e um conjunto finito de operadores* $\mathbf{O}$*;*
- $\mathbf{I} \subseteq \hat{\Sigma}$ *é a especificação do estado inicial;*
- $\mathbf{G} \subseteq \hat{\Sigma}$ *é a especificação do objetivo.*

Definição 3.5 (Plano) *Uma seqüência de operadores* Δ *é dita um plano para* Π*, ou a solução de* Π*, sse* $Res(\mathbf{I}, \Delta) \not\models \bot$ *e* $Res(\mathbf{I}, \Delta) \models \mathbf{G}$*.*

Uma seqüência de operadores Δ é dita um plano para Π, ou a solução de Π, sse $Res(\mathbf{I}, \Delta)$ não for inconsistente e suportar $\mathbf{G}$. Desta forma, tem-se que uma função de planejamento é definida como:

Definição 3.6 (Função de Planejamento) *Uma função de planejamento é definida por:*

$$Plan: \quad \Pi \rightarrow \mathbf{O}^*$$

$$Plan(\Pi) = \begin{cases} \Delta & \text{se } \exists \Delta \text{ e } \Delta \text{ é solução de } \Pi \\ \emptyset & \text{caso contrario} \end{cases}$$

Observando-se as definições desta seção, percebe-se que os planejadores considerados tratam apenas de átomos. Porém no contexto deste trabalho é desejável maior expressividade, em especial, a possibilidade de utilizar literais de, pelo menos primeira ordem. É possível evitar estas limitações. Para tanto utiliza-se transformações sintáticas, a fim de que tais planejadores operem sobre domínios descritos com literais de primeira ordem, desde que observando as seguintes restrições:

- As especificações de estado $\mathbf{I}$ e $\mathbf{G}$ só podem ser feitas através de literais *ground*[1];
- As descrições de operadores não podem criar novos literais através das suas pós-condições.

3.2 Compilação para SAT

As primeiras abordagens de planejamento se baseavam no conceito de planejamento como prova de teoremas, entretanto, esta abordagem não obteve resultados promissores de início. Desta forma, consideravam-se necessários algoritmos específicos para a realização de planejamento na

[1] Literais sem variáveis livres.

expectativa de desempenho aceitável. Experimentos realizados por Kautz e Selman [KAU 96] demonstraram que é possível a criação de algoritmos de planejamento tipo STRIPS através da compilação de problemas de planejamento com tamanho limitado para SAT (*SAtisfiability Testing*). Tais experimentos foram motivados por melhorias no desempenho de métodos de verificação de satisfatibilidade proposicional [COO 97]. A arquitetura típica de um resolvedor de problemas por SAT pode ser vista na Figura 3.1.

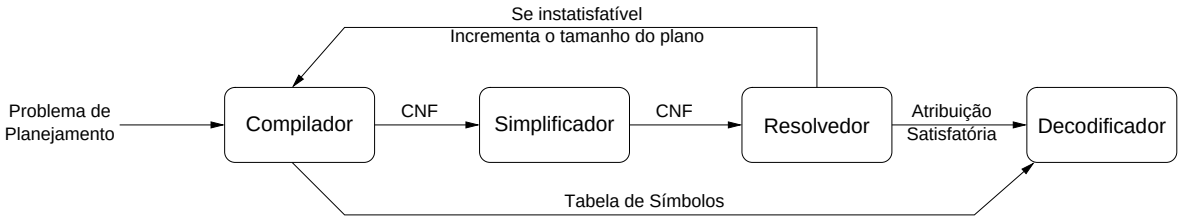

Figura 3.1: Estrutura de um planejador por SAT.

O funcionamento básico deste tipo de algoritmo inicia com um compilador recebendo um problema de planejamento como entrada. Este compilador supõe um tamanho de plano, e gera uma fórmula em lógica proposicional, que, se satisfeita, implica na existência de um plano que representa a solução do problema proposto. Uma *tabela de símbolos* guarda a correspondência entre as variáveis proposicionais criadas na compilação e a instância de planejamento recebida como entrada. Um processo de simplificação utiliza técnicas rápidas (de complexidade linear) para reduzir a fórmula em CNF (*Conjunctive Normal Form*) criada pelo compilador, tais como propagação de cláusulas unitárias (*Unit-Clause propagation*) e eliminação de literais puros (*Pure Literal Elimination*). O resolvedor utiliza métodos sistemáticos ou estocásticos para encontrar uma atribuição satisfatória para as variáveis da fórmula compilada. Se esta atribuição for encontrada, o decodificador traduz esta atribuição de variáveis, utilizando a tabela de símbolos, para um plano proposicional. Se o resolvedor determina que a fórmula é insatisfatível, então o compilador gera uma nova codificação refletindo um plano mais longo. Este processo de criação sistemática de codificações proposicionais mais longas é análogo ao processo de expansão do grafo de planejamento do Graphplan (Seção 3.3.1).

3.3 Graphplan

Graphplan [BLU 97] é um algoritmo de planejamento baseado na construção e busca em um grafo. O Graphplan é considerado um dos algoritmos de planejamento mais eficientes criados recentemente [WEL 99, NEB 00, SMI 99, HOF 01], tendo sido refinado em uma série de outros algoritmos, tais como o IPP (*Interference Progression Planner*) [KÖH 97] e o STAN (*STate*

Analysis) [LON 99]. A eficiência deste algoritmo foi comprovada empiricamente através dos diversos resultados positivos obtidos por instâncias deste algoritmo nas últimas competições de planejamento da AIPS (*International Conference on AI Planning and Scheduling*[2].) [KÖH 98, LON 00, GER 02, GHA 02].

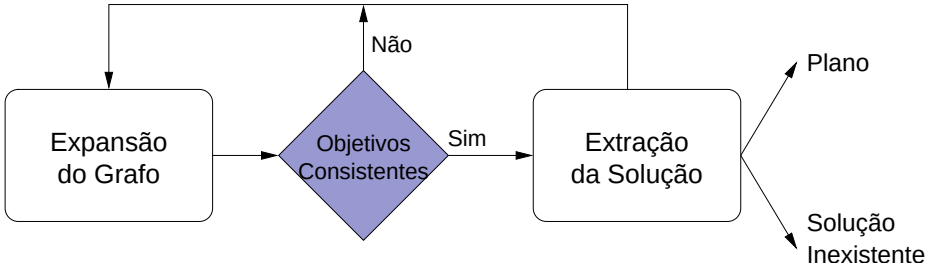

Figura 3.2: Visão Geral do Algoritmo Graphplan.

Neste algoritmo o planejamento é baseado no conceito de *Grafo de Planejamento*, que é uma estrutura de dados onde é armazenada informação a respeito do problema de planejamento de forma que a busca pela solução possa ser acelerada. O Grafo de Planejamento não é um grafo do espaço de estados, o qual pode ser extremamente grande [BLU 97]. Diferentemente do grafo do espaço de estados, onde um plano é um caminho através do grafo, um plano no Grafo de Planejamento é essencialmente um fluxo, no sentido de um fluxo em uma rede. Grafos de Planejamento podem ser construídos rapidamente, sendo a complexidade de tamanho do grafo e do tempo de construção polinomial em relação ao tamanho do problema [BLU 97]. Este grafo é então utilizado pelo planejador na busca por uma solução para o problema de planejamento utilizando os dados armazenados no grafo para acelerar o processo. O algoritmo básico do Graphplan (*i.e.* sem as otimizações adicionadas por outros autores) é dividido em duas fases distintas, expansão do grafo (Seção 3.3.1) e extração da solução (Seção 3.3.2), como visto na Figura 3.2. Nas próximas seções utilizaremos como base o problema descrito no Exemplo 3.2.

Exemplo 3.2 *Problema do Robô-Carteiro em STRIPS*

[2]Esta conferência foi substituída pelo ICAPS (*International Conference on Automated Planning & Scheduling*)

```
start(¬batt, package)
goal(batt, ¬package)
operator post
    preconds (batt, package)
    effects (¬batt, ¬package)
operator recharge
    preconds (¬batt)
    effects (batt)
```

No problema do Exemplo 3.2, criado utilizando o domínio do Exemplo 3.1, tem-se no estado inicial o robô-carteiro sem bateria e um pacote que deve ser encaminhado, e no estado final deseja-se que o robô encaminhe o pacote e esteja com a sua bateria carregada.

3.3.1 Expansão do Grafo

O Grafo de Planejamento é direcionado e nivelado, ou seja, os nodos do grafo podem ser divididos em conjuntos disjuntos de modo que as arestas conectem apenas nodos em níveis adjacentes. Considerando o fato de que um plano é composto por ações organizadas temporalmente, e, entre estas ações existem estados de mundo, os níveis no grafo são divididos em níveis de proposições e de ações, alternadamente. Níveis de proposições são compostos por nodos de proposição rotulados com proposições. Estes nodos são conectados aos nodos de ação no nível de ação subseqüente através de arcos de pré-condição. Nodos de ação são rotulados com operadores e são conectados aos nodos do nível de proposição subseqüente por arcos de efeito. Cada nível de proposição denota os literais possivelmente verdadeiros em um dado momento, de modo que o primeiro nível de proposição representa os literais possivelmente verdadeiros no tempo 1, o nível de proposição seguinte representa os literais possivelmente verdadeiros no tempo 2 e assim por diante.

Os níveis de ação denotam os operadores que podem ser executados em um dado momento no tempo de modo que o primeiro nível de ação representa os operadores que poderiam ser executados no tempo 1, e assim por diante. Os níveis de ação podem conter qualquer um dos operadores definidos no domínio de planejamento. Além destes operadores, um conjunto de operadores de manutenção de proposições é gerado automaticamente para as proposições do problema; estes operadores têm um único literal como pré-condição e o mesmo literal como efeito. Estes operadores são chamados de *no-op* ou *frame*.

O primeiro nível L_0 em um Grafo de Planejamento é um nível de proposição equivalente à especificação do estado inicial. Na primeira expansão do grafo, o nível L_1 será composto dos operadores que poderiam ser executados dadas as proposições de pré-condição presentes no nível anterior. No nível L_2 são incluídas todas as proposições presentes nos efeitos dos operadores

no nível L_1. Expansões subseqüentes do grafo ocorrem da mesma maneira, isto é, tomando um nível de proposição L_i, tal que i é par, como referência, é adicionado um nível de ação L_{i+1} contendo todos os operadores cujo conjunto de pré-condições está presente no nível L_i, os nodos de proposição no nível L_i são conectados aos nodos de ação apropriados através de arcos de pré-condição. A seguir, é adicionado um nível de proposição L_{i+2} contendo todas as proposições presentes nos efeitos de cada operador em L_{i+1}. Os nodos de ação em L_{i+1} são conectados às proposições apropriadas em L_{i+2} através de arcos de efeito. É importante salientar que, dado o conjunto de operadores de manutenção inseridos no conjunto de operadores, cada proposição presente em um dado nível de proposição L_i estará presente no nível de proposição L_{i+2} subseqüente através do operador *no-op* criado para aquela proposição. Um efeito desta propriedade do Grafo de Planejamento é que o número de proposições nos níveis pares aumenta monotonicamente, uma vez que as proposições ou serão adicionadas através dos operadores do domínio de planejamento ou serão trazidas do nível anterior por operadores de manutenção. Além disto, dada a monotonicidade no número de proposição, é trivial concluir que o número de operadores nos níveis de ação também aumenta monotonicamente.

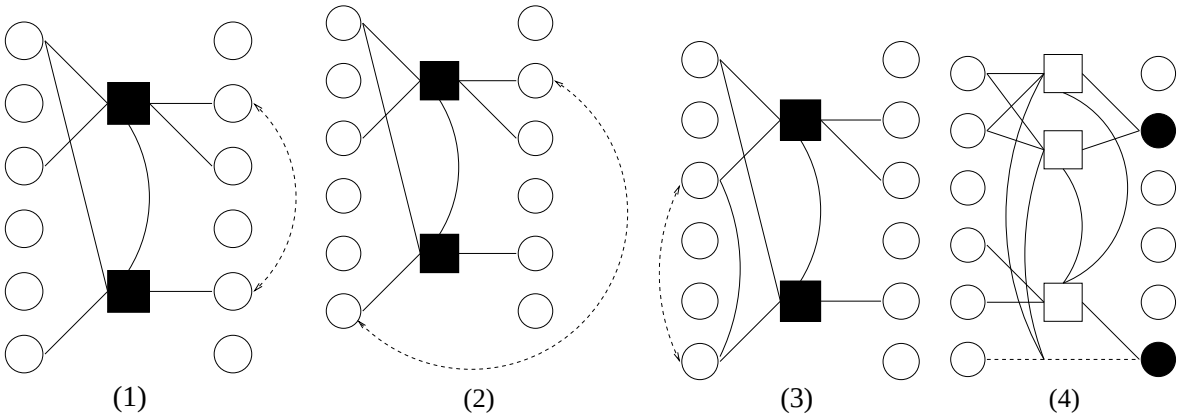

Figura 3.3: (1) Efeitos Inconsistentes - (2) Interferência - (3) Necessidades Concorrentes - (4) Suporte Inconsistente

Além de informação relativa a pré-condições e efeitos, o grafo de planejamento contém relações de exclusão mútua (*mutex*) entre os nodos em um mesmo nível do grafo. As relações de exclusão mútua têm um papel fundamental na eficiência do algoritmo. Uma relação de exclusão mútua entre dois nodos significa que estes (*i.e.* seus operadores ou proposições correspondentes) não podem estar presentes simultaneamente em um dado nível do grafo na mesma solução, isto é, duas proposições não podem ser verdadeiras no mesmo ponto no tempo, ou dois operadores não podem ser executados simultaneamente no mesmo passo do plano. Operadores podem ser mutuamente

exclusivos por dois motivos (Figura 3.3):

- **Interferência:** Se um operador possui uma pré-condição ou efeito que representa a negação de uma pré-condição ou efeito de outro operador, então eles são mutuamente exclusivos devido a interferência;
- **Necessidades Concorrentes:** Se existe um operador A com uma pré-condição P_a e um operador B com uma pré-condição P_b que é mutuamente exclusiva de P_a em um nível de proposição anterior, então A e B são ditas mutuamente exclusivas.

Nodos de proposição também podem ser mutuamente exclusivos por dois motivos básicos. Primeiro, uma proposição é mutuamente exclusiva com sua negação. Além disto, uma proposição p é mutuamente exclusiva de uma proposição q se todas as maneiras de se atingir p são exclusivas de todas as maneiras de se atingir q. Mais especificamente, se não existem dois operadores, ou um único operador que satisfaça ambas as proposições, que possam ser executados para satisfazer p e q, então estas duas proposições são mutuamente exclusivas devido a **Suporte Inconsistente** [WEL 99] (Figura 3.3).

Um exemplo de grafo de planejamento pode ser visto na Figura 3.4. Retângulos representam ações e elipses representam proposições, arcos representam relações de exclusão mútua e linhas representam os vértices do grafo.

A Figura 3.4 contém exemplos de relações de exclusão mútua. O operador `post` é mutuamente exclusivo em relação ao operador `recharge` por interferência, pois `post` resulta em ¬`batt` e `recharge` resulta em `batt`. Além disto, no nível 3 do grafo, os operadores `_nop-batt` e `_nopbatt` possuem necessidades concorrentes. Isto é, um deles necessita que ¬`batt` seja verdadeiro no nível de proposição anterior e o outro que `batt` seja verdadeiro e estas proposições são mutuamente exclusivas.

No nível 4 do grafo da Figura 3.4 tem-se um exemplo de proposições mutuamente excludentes por suporte inconsistente entre as proposições `batt` e ¬`package`. Isto ocorre pois não existem dois operadores que possam ser executados no nível 3 do grafo e que resultem em ambas as proposições.

A importância das relações de exclusão mútua não reside simplesmente na sua aplicação a cada nível do grafo no processo de extração de solução, mas também no fato de que propriedades importantes sobre o problema analisado são propagadas através do grafo durante a sua expansão. Esta propagação ocorre em decorrência das noções de **Necessidades Concorrentes** e de **Suporte Inconsistente** (Figura 3.3). A inferência de um grande número de relações de exclusão mútua implícitas no problema é um fator de aceleração importante no processo de extração de solução, visto que a construção do grafo tem complexidade bem menor do que a extração de solução [BLU 97].

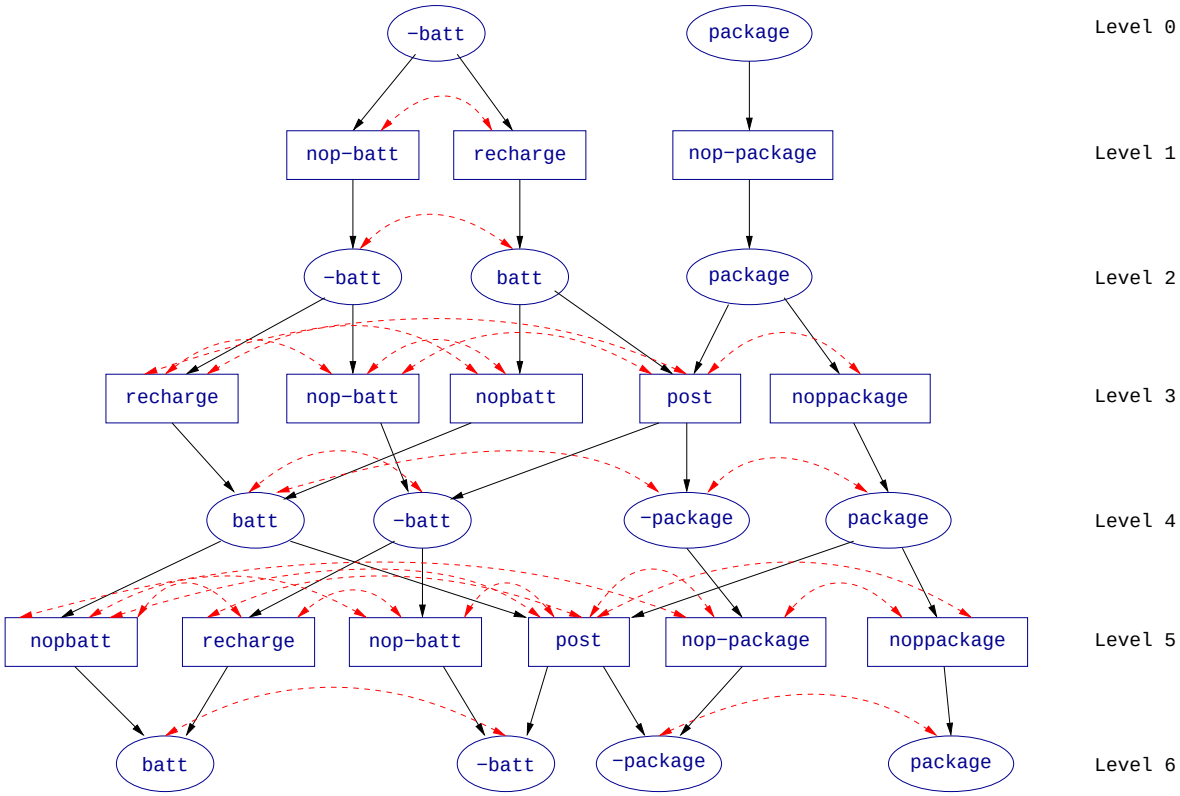

Figura 3.4: Grafo de planejamento para problema do Exemplo 3.2.

Um dos principais fatores aos quais é atribuída a eficiência do Graphplan é a complexidade polinomial da fase de expansão do grafo tanto em tempo de execução como em tamanho do grafo. Isto é provado através do Teorema 3.1 [BLU 97]. Este teorema foi modificado a fim de se adequar ao formalismo utilizado neste trabalho, uma vez que o formalismo utilizado originalmente no Graphplan é o STRIPS e neste trabalho utiliza-se o STRIPS com literais nas definições de estado. Apesar desta mudança no formalismo, as propriedades do algoritmo permanecem válidas.

Teorema 3.1 *Considerando um problema de planejamento com n objetos, p proposições no estado inicial e m operadores cada um com um número constante de parâmetros formais. Seja l o tamanho da maior lista de* efeitos positivos *de qualquer um dos operadores do problema. Então, o tamanho de um grafo com t níveis criado pelo Graphplan, e o tempo necessário para criá-lo são polinomiais em n, m, p, l e t.*

3.3.2 Extração da Solução

Após a expansão do grafo, ocorre a segunda fase do Graphplan, chamada de extração da solução. Ela utiliza uma estratégia de encadeamento reverso que atravessa o grafo nível por nível

tentando encontrar um fluxo a partir dos objetivos que leve às condições iniciais. Um fator de otimização importante aplicado a esta fase é nunca buscar uma solução a não ser que todas as proposições do objetivo estejam presentes e sejam consistentes, *i.e.* que não sejam mutuamente exclusivas, no último nível do grafo.

O algoritmo de extração da solução tenta, recursivamente, encontrar para cada nível, um conjunto de proposições ou ações consistentes que leve ao nível anterior do grafo até que a condição inicial no nível zero do grafo seja alcançada.

Algorithm extractSolution(iGraphLevel:integer,
oCurrentGoal:proposition,
pSelectedActions:plan):boolean

1: **if** $iGraphLevel = 0$ **then**
2: return true;
3: **end if**
4: **if** $lastGoalInLevel(oCurrentGoal)$ **then**
5: iGraphLevel = iGraphLevel - 2;
6: oCurrentGoal = getNextLevelGoals();
7: return extractSolution(iGraphLevel, oCurrentGoal, pSelectedActions);
8: **end if**
9: **for all** actions in $iGraphLevel - 1$ **do**
10: **if** $fulfills(aSupportAction, oCurrentGoal)$ and not $mutex(aSupportAction, pSelectedActions)$ **then**
11: **if** $extractSolution(iGraphLevel, nextGoal(oCurrentGoal), pSelectedActions)$ **then**
12: return true;
13: **end if**
14: **end if**
15: **end for**
16: return false;

Figura 3.5: Extração de solução.

Especificamente, o algoritmo inicia a busca no último nível L_i do grafo utilizando os objetivos do problema como objetivos daquele nível. Então, para cada objetivo, é escolhido um operador em L_{i-1} que o satisfaça, e que seja consistente com os demais operadores escolhidos, este operador é chamado de ponto de suporte. No caso de um objetivo sob análise já ter sido alcançado por um operador previamente escolhido, não é necessário adicionar um novo operador para satisfazê-lo. Quando pontos de suporte para todos os objetivos em um nível forem selecionados, as pré-

condições destes operadores são utilizadas como objetivos a serem alcançados no nível L_{i-2}, até que o algoritmo tenha atingido o nível zero do grafo, caso em que o algoritmo encontrou um plano válido. Uma vez que o algoritmo é chamado recursivamente sempre que um ponto de suporte para um objetivo for selecionado, se um plano incluindo um dado ponto de suporte não for encontrado, outros operadores são escolhidos até que todos os operadores possíveis em um dado nível de ação tiverem sido tentados. Quando todas os operadores possíveis para um dado objetivo tiverem sido tentados sem sucesso, o algoritmo retorna falso. Se esta falha ocorreu no último nível i do grafo, então o algoritmo provou que nenhum plano com $\frac{i}{2}$ ou menos níveis de ação existe para o dado problema.

A fim de acelerar o processo de busca por uma solução, os conjuntos de ações utilizadas como ponto de suporte só são utilizados nas recursões do algoritmo se eles forem conjuntos mínimos de ações (*minimal action sets*). Um conjunto de ações não exclusivas A no tempo $t-1$ é dito mínimo para o cumprimento de um conjunto de objetivos G em tempo t, se e somente se [BLU 97]:

1. Cada objetivo em G é um efeito de uma ação em A, e;

2. Nenhuma ação pode ser removida de A de modo que os efeitos das ações restantes ainda contenham G.

Além disto, uma das principais otimizações adicionadas ao algoritmo de extração de solução descrito na Figura 3.5 é a utilização da técnica de *memoização*, que é uma técnica algorítmica que consiste no armazenamento de uma resposta previamente computada para reuso posterior, ao invés de re-computá-la [Nat 03]. Esta técnica é utilizada no Graphplan da seguinte forma: quando um conjunto de objetivos em um dado nível do grafo é provado insolúvel, ele é *memoizado* em uma tabela antes do retorno da recursão. O algoritmo, então, antes de tentar chamar a função recursiva de extração de solução para um dado conjunto de objetivos, irá consultar esta tabela para verificar se o conjunto em questão já não foi provado insolúvel. Caso este conjunto seja encontrado na tabela, o algoritmo retorna falso ao invés de continuar a busca. O algoritmo modificado com todas estas otimizações é mostrado na Figura 3.6.

No caso do problema do Exemplo 3.2, o algoritmo de extração de solução irá concluir que o plano-solução é composto pela seqüência de ações `recharge`, `post` e `recharge`.

3.3.3 Condições de Finalização

Um aspecto importante do Graphplan é a garantia do algoritmo de que, caso um plano para o problema proposto exista, então o algoritmo irá encontrá-lo, e, caso contrário, o algoritmo irá determinar que o problema é insolúvel [BLU 97, IWE 02]. Isto significa que o Graphplan é correto e completo. A fim de provar que nenhum plano existe para um dado problema, alguns testes são

Algorithm extractSolution2(iGraphLevel:integer,
oCurrentGoal:proposition,
pSelectedActions:plan
htNoGoods:hashtable):boolean

```
1: if iGraphLevel = 0 then
2:    return true;
3: end if
4: if contains(htNoGoods,oCurrentGoal) then
5:    return false;
6: end if
7: if lastGoalInLevel(oCurrentGoal) then
8:    iGraphLevel = iGraphLevel - 2;
9:    oCurrentGoal = getNextLevelGoals();
10:   if isMinimalActionSet(iGraphLevel − 1, pSelectedActions) then
11:      return extractSolution(iGraphLevel, oCurrentGoal, pSelectedActions);
12:   end if
13: end if
14: for all actions in iGraphLevel − 1 do
15:   if fulfills(aSupportAction, oCurrentGoal) and not
      mutex(aSupportAction, pSelectedActions) then
16:      if extractSolution(iGraphLevel, nextGoal(oCurrentGoal), pSelectedActions) then
17:         return true;
18:      end if
19:   end if
20: end for
21: return false;
```

Figura 3.6: Extração de solução utilizando memoização.

realizados a cada ciclo de planejamento (*i.e.* a cada expansão do grafo com ou sem extração de solução), estes testes são descritos a seguir.

O primeiro teste se baseia no aumento monotônico do número de nodos de proposição a cada nível do grafo e da redução monotônica do número de relações de exclusão mútua entre eles. Devido às operações de manutenção, sempre que uma proposição ocorrer em um nível do grafo, ela necessariamente irá ocorrer em todos os níveis futuros. Também devido às operações de manutenção, sempre que um par de proposições não for marcado como mutuamente exclusivo em

um nível, ele nunca será marcados como tal em níveis futuros. Isto ocorre pois a única maneira de proposições serem marcadas como exclusivas, além de serem logicamente opostas, seria a inexistência de ações que pudessem alcançá-las simultaneamente. Entretanto, sempre que duas proposições forem marcadas como não-exclusivas, as operações de manutenção utilizadas para transpô-las para o nível seguinte não poderão ser exclusivas, sendo que esta propriedade irá se manter no grafo *ad infinitum*. Desta forma, estas propriedades de monotonicidade, aliadas ao tamanho finito da base de Herbrand dos problemas de planejamento considerados, garantem que existirá um ponto no processo de expansão do grafo onde os níveis de proposição serão sempre iguais a cada nova expansão. Neste ponto, diz-se que o grafo se estabilizou, o que pode ser visto como um ponto fixo da operação de expansão do grafo.

No momento em que o grafo se estabilizou, se uma proposição contida nos objetivos do problema não estiver presente no último nível do grafo, ou se duas proposições dos objetivos estiverem marcadas como mutuamente exclusivas, é possível afirmar que não existe nenhum plano que cumpra os objetivos propostos para o problema proposto. Esta propriedade é facilmente observada na medida em que, se os nodos de proposição nunca irão conter mais proposições ou menos relações de exclusão mútua, então não poderá ocorrer uma expansão do grafo que gere os objetivos necessários ou elimine as relações de exclusão. Considerando as condições de início da extração de solução, nenhuma tentativa de executá-la irá ocorrer neste caso, pois o grafo nunca terá atingido um ponto onde ela pudesse ser executada [BLU 97].

O primeiro teste permite que se determine de maneira extremamente rápida a inexistência de solução para um grande número de problemas, mas não garante que o algoritmo irá determinar a insolubilidade para todos os problemas possíveis. A fim de resolver esta limitação, os autores utilizam a tabela de memoização para postular o Teorema 3.2

Teorema 3.2 *O Graphplan retorna "Nenhum plano existe" se e somente se o problema é insolúvel.*

Uma maneira fácil de demonstrar a validade deste teorema é a seguinte. Seja S_i^t a coleção de conjuntos de objetivos armazenados na tabela para o nível i do grafo após a execução do ciclo t onde nenhuma solução foi encontrada no grafo. Após o ciclo t, o algoritmo terá provado que nenhum plano com t ou menos passos pode tornar verdadeiro em tempo t qualquer um dos conjuntos de objetivos em S_i^t, e que nenhum dos conjuntos de objetivos armazenados em S_i^t é alcançável em t passos. Baseados nestas conclusões, os autores afirmam que, se o grafo se estabilizou em algum nível n e um ciclo t ocorre no qual $|S_n^{t-1}| = |S_n^t|$, então não existe nenhum plano para o problema [BLU 97].

3.4 Considerações

A escolha do Graphplan como algoritmo de planejamento neste trabalho se deu pela existência, até onde se sabe, de uma quantidade maior de trabalhos relacionados ao Graphplan e sua extensão [KÖH 98, SMI 98, SMI 99, WEL 99, LON 00]. Tais trabalhos subseqüentes aperfeiçoaram o algoritmo original do Graphplan, incorporando a ele diversos avanços no campo de planejamento proposicional [KÖH 98, LON 00], em especial através do pré-processamento dos problemas de planejamento para eliminar informação irrelevante e da inferência de mais informações dentro do grafo de planejamento. A versão do Graphplan implementada neste trabalho não incorpora todos estes aperfeiçoamentos, entretanto é importante salientar que o desempenho obtido empiricamente na implementação proposta pode ser melhorado através de versões mais avançadas do Graphplan. Além disto, o Graphplan possui uma série de características que o destacam dos outros algoritmos de planejamento considerados em relação à sua aplicabilidade em sistemas multi-agentes.

Uma das principais características dos planos gerados pelo Graphplan é a representação da possibilidade de execução de ações em paralelo. Esta possibilidade é resultado da análise das relações de exclusão mútua nível-a-nível na fase de extração de solução. Desta forma, um agente que tenha condições de executar múltiplas ações simultaneamente pode se beneficiar deste tipo de plano. Além disto, o algoritmo de extração de solução não apenas é correto e completo, mas também fornece a garantia de que o plano gerado é o *mais curto* possível, considerando a ocorrência de ações em paralelo.

Além da exploração do paralelismo em um agente, o grafo de planejamento e a tabela de *memoização* podem ser armazenados pelo agente entre as execuções do planejamento. A informação contida nestas estruturas permite que o agente possa reaproveitá-las caso ele deseje verificar a existência de planos para múltiplos objetivos dada uma mesma situação inicial, evitando a reconstrução do grafo até o último nível verificado na execução anterior do algoritmo. Uma possibilidade levantada pelo autor é que tal informação também poderia ser utilizada pelo agente em algum tipo de processo de aprendizado sobre o domínio onde ele opera.

Finalmente, acredita-se que o Graphplan tem o potencial de prover um procedimento eficiente no processo de construção da árvore temporal utilizada na lógica BDI_{CTL}. Esta hipótese leva em consideração a noção de *Conformant Plans* definida em uma extensão do Graphplan que considera a incerteza do agente em relação à sua situação em múltiplos estados de mundo [SMI 98].

Capítulo 4

X-BDI

X-BDI é um modelo de agentes BDI definido com o intuito de criar um modelo formal de agentes que pudesse ser executado diretamente [MÓR 99a], evitando assim discrepâncias entre definição e implementação. A fim de atingir este objetivo, utilizou-se um formalismo chamado de *Extended Logic Programming* (ELP), o qual possui uma implementação de referência que provê um ambiente operacional para a execução de programas definidos neste formalismo [ALF 96]. Este capítulo irá descrever o modelo X-BDI de agentes definido por Móra *et al.* [MÓR 99a]. A Seção 4.1 contém uma descrição resumida do Framework Lógico utilizado no embasamento do modelo X-BDI e a Seção 4.2 descreve o modelo de agentes propriamente dito, sendo o processo de deliberação do agente descrito na Seção 4.3. No final do capítulo uma breve discussão sobre as limitações do modelo de agente original e possíveis melhorias é apresentada.

4.1 Framework Lógico

A programação em lógica tem sido utilizada largamente em implementações na área de IA devido a sua natureza declarativa, o que a torna bastante adequada no processo de transposição de modelos teóricos para implementações. A implementação do modelo de agentes X-BDI foi realizada utilizando uma extensão da programação em lógica chamada de Programação em Lógica Estendida com Negação Explícita, ou ELP, que estende a lógica normal ao permitir a representação de fatos explicitamente falsos. Este formalismo possui um modelo semântico próprio, descrito na Seção 4.1.1, e um procedimento de derivação e prova de predicados diferenciado, descrito Seção 4.1.2. Na Seção 4.1.3 é descrito o procedimento de revisão de cláusulas utilizado pela SLX a fim de lidar com a contradição de um programa.

4.1.1 Semântica WFSX

A semântica $WSFX$(*Well-Founded Semantics EXtended*) [PER 92] objetiva dar significado a programas em ELP. Este modelo semântico estende a semântica WFS(*Well-Founded Semantics*) [GEL 88, GEL 91] incluindo a negação explícita. A característica principal da $WFSX$ é a definição do significado de qualquer programa não contraditório em ELP através da noção de *stable models*.

4.1.2 Procedimento SLX

O procedimento de derivação SLX (*Selected Linear Derivation Extended with Explicit Negation*) [ALF 95, ALF 96] foi criado a fim de operacionalizar a semântica $WSFX$. A derivação SLX é inspirada no procedimento SLDNF (*Selected Linear Derivation with Negation as Failure*) para lógica normal, desta forma, as noções de derivação, refutação e falha são bastante semelhantes a este procedimento. Entretanto, a SLX trata do problema das derivações infinitas presente na SLDNF.

4.1.3 Revisão de Cláusulas na SLX

Como visto na Seção 4.1.1, a $WSFX$ é capaz de executar programas não-contraditórios. Entretanto, a utilização de uma lógica com negação explícita pressupõe a possibilidade de contradições em um programa, visto que um programa pode, em um dado momento, conter em seu modelo o mesmo literal na sua forma positiva e negativa. Desta forma, não é suficiente poder executar programas não-contraditórios, também é necessário encontrar e tratar as contradições possíveis em um programa. Conseqüentemente, os autores da SLX propõem um procedimento de detecção de contradição que utiliza uma versão *paraconsistente* da semântica $WSFX$, chamada de $WFSX_P$ [ALF 96]. A $WFSX_P$ tem o objetivo de calcular de forma coerente todas as conseqüências de um programa, inclusive aquelas que levam a contradição, ou resultam dela. Através deste cálculo é gerado um conjunto de literais ditos *básicos*, os quais são considerados a origem da contradição em um programa. A este conjunto de literais é aplicado um procedimento que altera os valores-verdade de alguns dos literais pertencentes ao conjunto de modo que a contradição é removida.

Além de modificar o programa para remover contradições relacionadas com o uso da negação explícita, a SLX define um mecanismo para especificar contradições relacionadas com determinados estados de um programa em lógica. Este mecanismo é baseado em restrições de integridade que, quando violadas, causam a revisão do programa de modo que o programa não contenha mais o estado de programa declarado como contraditório.

4.2 Modelo do Agente

Considerando que o X-BDI [MÓR 99a] é um modelo formal de agentes, a implementação de seus processos de funcionamento pode ser concretizada de múltiplas maneiras. Entretanto, a maneira escolhida pelo autor foi a utilização do formalismo ELP [ALF 96] descrito na Seção 4.1. A união da especificação formal com a implementação do framework lógico empregado resulta em um conjunto de processos que implementam a lógica BDI formulada para o modelo X-BDI. Esta seção tem por objetivo apresentar as principais definições formais que descrevem o comportamento de um agente X-BDI.

O X-BDI é um modelo de agentes baseado na teoria de raciocínio prático de Bratman [BRA 90], utilizando as intenções como componente necessário para o raciocínio em agentes com recursos limitados. Desta forma, o processo de raciocínio de um agente neste formalismo é centrado na construção de um conjunto de intenções que irá, em última instância, definir que cursos de ação ele utilizará para cumprir seus objetivos. Um esquema de funcionamento do processo de formação de intenções e seus estados intermediários pode ser visto na Figura 4.1.

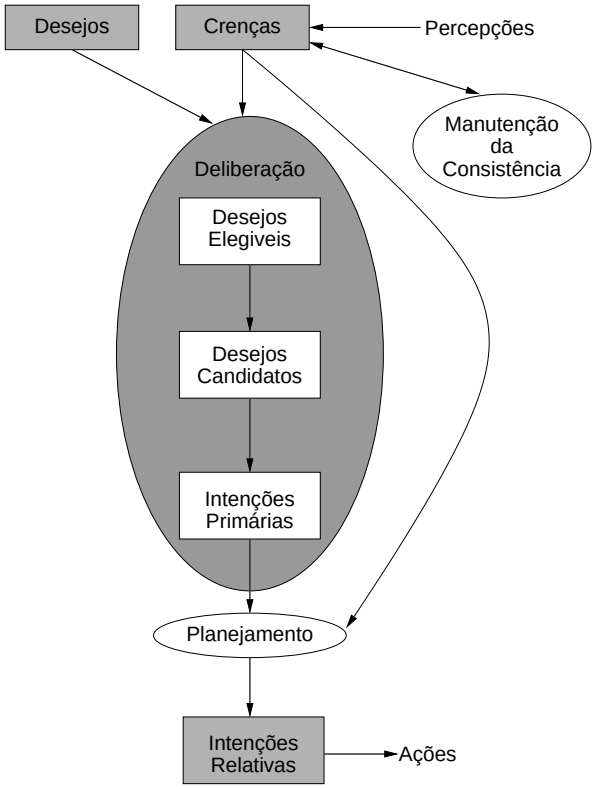

Figura 4.1: Esquema de funcionamento do X-BDI

Os estados intermediários presentes na Figura 4.1 são formados a partir de uma descrição do

agente que é utilizada para compor a estrutura cognitiva de um agente X-BDI, que é descrita pela Definição 4.1 [MÓR 99b]:

Definição 4.1 (Estrutura Cognitiva do Agente) *A estrutura cognitiva de um agente X-BDI é uma tupla* $\mathcal{A}g = \langle \mathcal{B}, \mathcal{D}, \mathcal{I}, \mathcal{T}Ax \rangle$*, onde:*

- $\mathcal{B}$ *é o conjunto de crenças do agente;*
- $\mathcal{D}$ *é o conjunto de desejos do agente;*
- $\mathcal{I}$ *é o conjunto de intenções do agente;*
- $\mathcal{T}Ax$ *é o conjunto de axiomas de tempo;*

Um agente X-BDI possui os componentes tradicionais de um agente BDI, *i.e.* um conjunto de Crenças, Desejos e Intenções. Além disto, dada a sua definição em lógica estendida, ele possui também um conjunto de axiomas de tempo definidos a partir de uma variação do *Cálculo de Eventos* [MÓR 99b, KOW 86]. A versão do Cálculo de Eventos utilizada no X-BDI é descrita na Definição 4.2.

Definição 4.2 (Axiomas de tempo $\mathcal{T}Ax$) -

Predicados básicos:

$$
\begin{aligned}
holds\_at(Prop,T) &\leftarrow initially(Prop),\\
&\quad persists(0,Prop,T).\\
holds\_at(Prop,T) &\leftarrow initiates(TEvt,Action,Prop),\\
&\quad TEvt < T,\\
&\quad persists(TEvt,Prop,T).\\
holds\_at(bel(Ag,P),T) &\leftarrow holds\_at(identity(Ag),T),\\
&\quad P \neq bel(A,B),\\
&\quad holds\_at(P,T).\\
holds\_at(Prop,T) &\leftarrow sense(Prop,TP),\\
&\quad TP < T,\\
&\quad persists(TP,Prop,T).\\
persists(TEvt,Prop,T) &\leftarrow not\ (clipped(TEvt,Prop,T)).\\
clipped(TEvt,Prop,T) &\leftarrow act(TIntEvt,Action),\\
&\quad terminates(TIntEvt,Action,Prop),\\
&\quad not\ (out(TIntEvt,TEvt,T)).\\
clipped(TP,bel(Ag,Prop),T) &\leftarrow holds\_at(identity(Ag),T),\\
&\quad Prop \neq bel(A,B),\\
&\quad clipped(TP,Prop,T).\\
clipped(TP,Prop,T) &\leftarrow sense(\neg(Prop),TIntP),\\
&\quad not\ (out(TIntP,TP,T)).\\
out(TIntEvt,TEvt,T) &\leftarrow T \leq TIntEvt.\\
out(TIntEvt,TEvt,T) &\leftarrow TEvt \geq TIntEvt.
\end{aligned}
$$

Extensões do cálculo de eventos relativas a ELP:

$$
\begin{aligned}
holds\_at(\neg P,T) &\leftarrow \neg holds\_at(P,T)..\\
\neg holds\_at(P,T) &\leftarrow not\ holds\_at(P,T)..\\
&\leftarrow holds\_at(P,T),\\
&\leftarrow holds\_at(-P,T).\\
&\leftarrow act(T,A),\\
&\quad act(T,B),\\
&\quad A \neq B.
\end{aligned}
$$

Neste contexto o predicado $holds\_at(P,T)$ indica que uma propriedade P é verdadeira no tempo T, o predicado $happens(E,T_i,T_f)$ indica que um evento E ocorre entre o tempo T_i e o tempo T_f. O predicado $initiates(E,T_P,P)$ denota que uma propriedade P passa a ser verdadeira a partir do tempo T_P através de um evento E. O predicado $persists(T_P,P,T)$ denota que uma

propriedade P é verdadeira a partir do tempo T_P até pelo menos o tempo T. O predicado $clipped(T_P, P, T)$ significa que ocorre um evento qualquer entre o tempo T_P e o tempo T que torna uma propriedade P falsa. O predicado $sense(E, T_i)$ indica que um evento E ocorre no tempo T_i, representando o sensoriamento pelo agente de alguma mudança no ambiente.

O X-BDI utiliza um processo de raciocínio abdutivo para verificar a possibilidade de um determinado conjunto de desejos. Este processo é muito semelhante a um processo de planejamento *backward chaining*, visto que o processo de abdução tenta encontrar uma teoria de ações em cálculo de eventos que torne as propriedades especificadas nos desejos verdadeiras. A fim de encadear os predicados do cálculo de eventos, um conjunto de regras de integridade é fornecido na Definição 4.3 [MÓR 99b].

Definição 4.3 (Regras de integridade para abdução de predicados) -

$$\begin{aligned}
\bot \quad &\Leftarrow \quad happens(E, T_{1i}, T_{1f}),\ happens(E, T_{2i}, T_{2f}),\\
&\qquad not(T_{1i} = T_{2i}, T_{1f} = T_{2f}).\\
\bot \quad &\Leftarrow \quad happens(E, T_i, T_f),\ not(T_f < T_i).\\
\bot \quad &\Leftarrow \quad happens(E, T_i, T_f),\ not(act(E, A)).
\end{aligned}$$

O conjunto de crenças que compõe a estrutura cognitiva do agente é definido da seguinte forma [MÓR 99b]:

Definição 4.4 (Conjunto de Crenças) *As crenças de um agente são o programa consistente em lógica estendida $\mathcal{B}$, i.e. $\mathcal{B} \not\models_P \bot$.Um agente A acredita que uma propriedade P é válida no tempo T sse $\{\mathcal{B} \cup \mathcal{T}Ax\} \models_P holds\_at(bel(A, P), T)$.*

Desta forma, o conjunto de crenças é simplesmente uma formalização de fatos em programação lógica individualizados para um agente em específico. Além da definição do conjunto de crenças, é relevante a definição do processo de manutenção da consistência na base de crenças.

Revisão de Crenças A revisão de crenças no X-BDI está ligada ao processo de revisão de literais na SLX, uma vez que é assumido que a base de crenças esteja sempre consistente como decorrência do funcionamento da SLX. Desta forma, assim que um evento modifique a base de crenças de forma que esta se torne inconsistente, o processo de revisão de literais da SLX irá garantir que a base de crenças seja modificada de forma que se torne consistente e realizando o menor número de modificações possível. Este processo está diretamente associado aos predicados $happens/3$ e $sense/2$, uma vez que estes são os únicos predicados associados a modificação da base de crenças do agente através dos atos do agente e de eventos percebidos do ambiente, respectivamente. Desta forma, a revisão de literais da SLX será ativada sempre que um destes dois

predicados estiver envolvido em uma derivação.

O componente seguinte na estrutura cognitiva do agente é o conjunto de desejos, que é definido da seguinte forma:

Definição 4.5 (Conjunto de Desejos) *O conjunto de desejos de um agente é o conjunto de sentenças na forma:*

$$\mathcal{D} = \{holds\_at(des(D, Ag, P, T_D, Atr), T) \leftarrow Body\}$$

onde D é o identificador do desejo, P é a propriedade cuja veracidade é desejada no tempo T_D, T é um ponto no tempo, Ag é o identificador do agente, e Atr é uma lista de atributos. $Body$ é qualquer conjunção de literais. Um agente deseja no tempo T que uma propriedade P seja verdadeira no tempo T_D sse:

$$(holds\_at(des(D, Ag, P, T_D, Atr), T) \leftarrow Body) \in \mathcal{D},$$

para algum D, Ag, Atr.

Observando a forma das sentenças que definem os desejos, pode-se notar que os desejos, representados em regras de programação em lógica, são condicionados pelo corpo da regra ($Body$). Desta forma, os literais componentes de $Body$ podem ser vistos como pré-condições para que um agente deseje uma determinada propriedade, uma vez que tais literais devem ser verdadeiros para satisfazer a implicação $Body \rightarrow Desejo$[1]. No caso de $Body$ ser vazio, temos que o agente deseja que a propriedade P seja verdadeira incondicionalmente, já que, neste caso, o desejo será um fato (*i.e.* $Desejo \leftarrow$) do ponto de vista de programação em lógica. Além disto, o agente pode desejar alguma propriedade em um ponto específico de tempo no caso de a variável T_D estar associada a um ponto no tempo. A variável Atr corresponde a um lista de atributos que especificam a prioridade do desejo, que é utilizada no processo de escolha dos desejos.

A viabilidade de um desejo é verificada através de raciocínio abdutivo. Utilizando uma teoria T, composta pelas crenças do agente, e um conjunto de observações O representando as propriedades que se deseja concretizar, o processo de abdução tenta encontrar uma teoria Δ tal que $T \cup \Delta \models O$, e tal que $T \cup \Delta$ seja consistente. Esta teoria Δ é formada utilizando um conjunto de literais marcados como abdutíveis pela ELP, que no caso do X-BDI são as ações do agente. Desta forma, a possibilidade de um conjunto de desejos é comprovada pela existência de um conjunto de ações que possa justificar a validade das propriedades desejadas pelo agente.

Definição 4.6 (Conjunto de Intenções) *As intenções de um agente no tempo T são o conjunto:*

[1] Onde $Desejo$ equivale a $holds\_at(des(D, Ag, P, T_D, Atr), T)$

$$\mathcal{I} = \{X / X = int\_that(I, Ag, P, T_I, Atr) \vee X = int\_to(I, Ag, Act, T_I, Atr)\}$$

onde I é o identificador da intenção, P é a propriedade que se deseja tornar verdadeira, Ag é um agente, Atr é uma lista de atributos, Act é uma ação, e T_I é o tempo em que a intenção deve ser concretizada, e tal que:

1. $holds\_at(int\_that(I, Ag, P, T_I, Atr), T) \ \vee \ holds\_at(int\_to(I, Ag, Act, T_I, Atr), T)$;
2. $\forall I \ int\_that(I, Ag, P, T_I, Atr) \in \mathcal{I}.(Now \leq T_I)$;
3. $\forall I \ int\_to(I, Ag, Act, T_I, Atr) \in \mathcal{I}.(Now \leq T_I)$;
4. $\forall I \ int\_to(I, Ag, Act, T_I, Atr) \in \mathcal{I}.(\mathcal{B} \cup \mathcal{T}Ax \not\models_P (happens(E, T_I, T_F), act(E, Act)))$;
5. $\exists \Delta.(P' \cup \Delta \not\models_P \bot)$, *onde:*
   (a) P' *é o framework abdutivo* $\langle \{\mathcal{B} \cup \mathcal{T}Ax\}, \{happens/3, act/2\}, IC(\mathcal{I})\rangle$;
   (b) $IC(\mathcal{I})$ *é o conjunto de restrições gerado pelas intenções, definido como:*

$$holds\_at(bel(Ag, P), T) \Leftarrow$$

para cada $int\_that(I, Ag, P, T_I, Atr)$ *em* $\mathcal{I}$ *e*

$$happens(E, T, T_F) \Leftarrow$$
$$act(E, Act) \Leftarrow$$

para cada $int\_to(I, Ag, Act, T_I, Atr)$ *em* $\mathcal{I}$

um agente tem, no tempo T a intenção de que uma propriedade P seja verdadeira no tempo T_I sse $\mathcal{B} \cup \mathcal{T}Ax \cup \mathcal{R} \models_P holds\_at(int\_that(I, Ag, P, T_I, Atr), T)$, e ele tem, no tempo T, a intenção de executar a ação Act no tempo T_I sse $\mathcal{B} \cup \mathcal{T}Ax \cup \mathcal{R} \models_P holds\_at(int\_to(I, Ag, Act, T_I, Atr), T)$.

O item 1 da Definição 4.6 apresenta os dois tipos possíveis de intenção no X-BDI, que são: intenções primárias (int_that) e intenções relativas (int_to). As diferenças entre elas serão melhor exploradas na Seção 4.3.2. Os itens 2 e 3 estipulam que um agente não pode querer algo no passado. O item 4 estipula que um agente não pode ter a intenção de realizar algo que já seja verdadeiro. A última restrição sobre o conjunto de intenções é que um agente não tem intenções as quais ele acredita serem impossíveis. Uma propriedade importante desta definição é que ela evita o problema do pacote (Problema 2.4) ao não permitir que efeitos colaterais de uma intenção afetem a adoção futura de novas intenções ou causem a reconsideração do conjunto de intenções.

4.3 Processo de Deliberação no X-BDI

Esta seção contém as definições relativas ao processo deliberativo de um agente X-BDI. Este processo inicia com a seleção dos desejos que o agente irá tentar satisfazer (Seção 4.3.1), os quais irão determinar as intenções com as quais o agente irá se comprometer (Seção 4.3.2). Estas intenções irão guiar as ações do agente até que o agente as tenha satisfeito, ou até que ele tenha que reconsiderá-las (Seção 4.3.3).

4.3.1 Seleção dos Desejos

O processo de raciocínio dos agentes X-BDI inicia na escolha dos desejos elegíveis. Desejos elegíveis são os desejos que o agente acredita não estarem satisfeitos e que são possíveis. O conjunto de desejos de um agente (vide Definição 4.6) não está sujeito a nenhuma restrição, e, portanto, pode conter desejos contraditórios; o conjunto dos desejos elegíveis também não está sujeito a este tipo de restrição.

Definição 4.7 (Desejos Elegíveis) *Seja $\mathcal{D}$ o conjunto de desejos de um agente. O seguinte conjunto é chamado de* desejos elegíveis *no tempo T:*

$$\begin{aligned}\mathcal{D}' = \{des(D, Ag, P, T_D, Atr) / \quad & [holds\_at(des(D, Ag, P, T_D, Atr), T) \leftarrow Body \in \mathcal{D}] \wedge \\ & Now \leq T \wedge (\mathcal{B} \models_P Body) \wedge \\ & \{\mathcal{B} \cup \mathcal{T}Ax\} \models_P \neg holds\_at(bel(Ag, P), T_D)\}\end{aligned}$$

A definição dos desejos elegíveis estipula restrições de racionalidade semelhantes às estabelecidas para as intenções no sentido de que um agente não irá desejar nada no passado, e nem algo que o agente acredite que irá se concretizar sem a sua interferência. Além disto, as crenças do agente devem suportar a pré-condição do desejo estabelecida no corpo ($Body$) da definição do mesmo. No decorrer do processo de raciocínio do agente, este irá utilizar o conjunto dos desejos elegíveis (que podem ser mutuamente contraditórios) para gerar subconjuntos de desejos mutuamente consistentes. Para possibilitar ao agente a escolha de um destes conjuntos, uma relação entre estes conjuntos deve ser estabelecida [MÓR 99b].

Definição 4.8 (Relação de Preferência dos Desejos $<_{Pref}$) *Seja $\mathcal{D}$ o conjunto de desejos de um agente, $\mathcal{D}'$ o conjunto de desejos elegíveis de $\mathcal{D}$, $\mathcal{P}(\mathcal{D}')$ o conjunto potência*[2] *de $\mathcal{D}'$ e $R, S \in \mathcal{P}(\mathcal{D}')$. Diz-se que $R <_{Pref} S$ (R é menos preferido do que S) se o maior valor de importância que ocorre em S e não ocorre em R é maior do que o maior valor de importância que ocorre em R e não ocorre em S; se não existir tal valor maior em S, então R é menos preferido do que S se S possuir mais elementos do que R.*

[2] *i.e.* o conjunto de todos os subconjuntos.

Exemplo 4.1 (Escolha de Desejos) *Considere o seguinte conjunto de desejos elegíveis:*

$$D' = \{des(d1, Ag, p1, t1, [0.7]), des(d2, Ag, p2, t2, [0.4]), des(d3, Ag, p3, t3, [0.4])\}$$

Então a ordem de preferência dos elementos do conjunto potência de D' é:

$$\{d2\}, \{d3\} <_{Pref} \{d2, d3\} <_{Pref} \{d1\} <_{Pref} \{d1, d2\}, \{d1, d3\} <_{Pref} \{d1, d2, d3\}$$

Ou seja, a relação de preferência de desejos coloca o conjunto com o desejo mais importante antes dos demais conjuntos. Como critério secundário de organização dos elementos na relação é utilizado o tamanho dos conjuntos. Entretanto, esta relação não organiza os sub-conjuntos de desejos elegíveis de modo que nenhum conjunto tenha a mesma importância que todos os outros. Logo, tem-se que a relação de preferência de desejos não é uma relação de ordem total, isto é, não é possível estabelecer uma relação entre qualquer par de elementos do conjunto, por exemplo, os conjuntos $\{d2\}$ e $\{d3\}$ não estão relacionados, isto pode ser visto do diagrama de Hasse [DAV 02] mostrado na Figura 4.2.

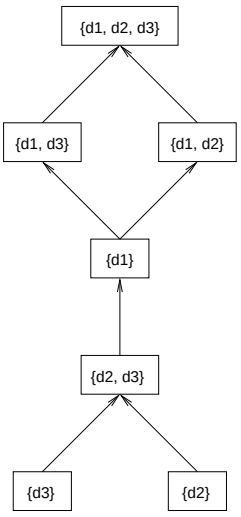

Figura 4.2: Relações entre os elementos de $\mathbb{P}(D')$

Uma vez que há a possibilidade de nem todos os desejos serem mutuamente consistentes, é necessário uma forma de marcar no programa quais serão os desejos válidos em determinado momento, após estabelecido o sub-conjunto dos desejos que serão utilizados pelo agente. Para realizar isto, cada desejo tem associado a si um literal *default* chamado de $unsel(D)$, onde D é o identificador do desejo associado a este literal. Estes literais serão revisados no momento da escolha dos desejos candidatos.

Definição 4.9 (Grafo de Preferência dos Desejos) *Sejam $\mathcal{D}$ o conjunto de desejos de um agente, $\mathcal{D}'$ o conjunto de desejos elegíveis de $\mathcal{D}$, $\ll$ a função de sucessor. Seja $Revisable$ o conjunto:*

$$Revisable = \{unsel(D)/\exists D.des(D, Ag, P, T_D, Atr) \in \mathcal{D}'\}$$

e $index : \mathcal{P}(Revisable) \longrightarrow \mathbb{N}^+$ uma função do conjunto potência de $Revisable$ nos números naturais (sem o zero) que atribui um número de nível aos elementos de $\mathcal{P}(Revisable)$. O grafo de preferência dos desejos *é o grafo definido por:*

1. $Rev(\mathbf{bottom}) = \{happens(E, T_i, T_f), act(E, A)\}$;
2. $Rev(i) = R \cup \{happens(E, T_i, T_f), act(E, A)\}$, *onde* $R \in \mathcal{P}(Revisable)$ *e* $i = index(R)$;
3. $i \ll \mathbf{bottom}$, *onde* $i = index(R)$, $R \in \mathcal{P}(Revisable)$ *e* $S \in \mathcal{P}(Revisable).(S <_{Pref} R)$;
4. $j \ll k_1, \ldots, k_n$, *onde* $j = index(R)$, $k_i = index(S_i)(1 \leq i \leq n)$, $R, S_i \in \mathcal{P}(Revisable)(1 \leq i \leq n)$ *e* R *é um antecedente de* S.

O grafo de preferência dos desejos é utilizado na geração de um grafo de revisões preferidas. A raiz deste grafo propõe revisões apenas nos literais $act/3$ e $happens/3$, ou seja, ele dá preferência para as revisões advindas da abdução dos planos para concretizar todos os desejos simultaneamente. Entretanto, como visto anteriormente, o conjunto de todos os desejos pode não ser mutuamente consistente, desta forma a relação de preferência dos desejos é utilizada para gerar os níveis subseqüentes deste grafo, estabelecendo que os desejos menos preferidos devem ser eliminados antes através do posicionamento dos literais $unsel(D)$ associados aos desejos menos preferidos, isto é, aqueles com número de importância menor, ou pertencentes a um conjunto com menos desejos, mais próximos da raiz do grafo de preferência de revisões. Além disto, o fato de a relação de preferência dos desejos não ser uma relação de ordem total requer a definição de método determinístico para determinar quais serão os desejos escolhidos no processo de raciocínio.

O processo de seleção dos desejos candidatos no X-BDI objetiva escolher dentre os desejos elegíveis, um conjunto cujos desejos são mutuamente consistentes e viáveis. Um desejo viável é um desejo que apresenta um conjunto de ações cujo resultado seja a validade de uma propriedade P associada a um desejo [MÓR 99b].

Definição 4.10 (Conjunto dos Desejos Candidatos) *Sejam $\mathcal{D}$ o conjunto de desejos de um agente e $\mathcal{D}'$ o conjunto de desejos elegíveis de $\mathcal{D}$ com um grafo de preferência associado a ele. Chama-se* conjunto de desejos candidatos *qualquer conjunto:*

$$\begin{aligned} \mathcal{D}'_C = \quad & \{des(D, Ag, P, T_D, Atr)/(des(D, Ag, P, T_D, Atr) \in \mathcal{D}') \wedge \\ & \exists \Delta.(P' \cup \Delta \not\models_P \bot) \wedge \\ & [(\mathcal{B} \cup \mathcal{T}Ax \cup \mathcal{R} \cup \Delta \models_P (holds\_at(bel(Ag, P), T_D), not\, unsel(D)))]\} \end{aligned}$$

onde

1. P' *é o framework abdutivo*

$$\langle \mathcal{B} \cup \mathcal{T}Ax \cup \mathcal{R}, \{happens(E, T_i, T_f), act(E, Act), unsel(D)\}, IC \rangle;$$

2. IC *é o conjunto de restrições na forma*

   - $\{holds\_at(bel(Ag, P), T) \Leftarrow notunsel(D)\}$ *para cada* $des(D, Ag, P, T_D, Atr)$ *em* $\mathcal{D}'$*;*
   - *as restrições* $IC(\mathcal{I})$ *geradas pelas intenções (vide definição 4.6)*

A seleção dos desejos candidatos é o processo mais complexo do modelo X-BDI. A fim de escolher um subconjunto de desejos elegíveis que seja mutuamente consistente é utilizado o processo de revisão de literais da SLX (Seção 4.1.3). Utilizando o grafo de revisões preferidas (Definição 4.9) para ordenar os conjuntos de desejos consistentes, o processo de raciocínio abdutivo tenta provar sistematicamente a possibilidade de satisfação dos conjuntos com maior prioridade até que se obtenha a prova de que um destes conjuntos é possível. Neste caso, os desejos pertencentes a este conjunto irão constituir o conjunto de desejos candidatos. Se todos os conjuntos forem testados, e a possibilidade de nenhum deles puder ser provada, então o conjunto de desejos candidatos será vazio.

Os literais da teoria Δ gerados pelo processo de abdução constituem um plano de alto nível que indicam ao agente como satisfazer as propriedades P associadas aos desejos candidatos. Estes literais não levam em consideração o refinamento do plano de alto nível até o nível de ações concretas, nem as contradições que podem ser geradas pela execução deste plano.

O X-BDI evita o problema do efeito colateral descrito por Bratman em [BRA 99b]. O problema é evitado na medida em que as ações produzidas pelo processo de abdução não são consideradas contraditórias caso seus efeitos se cancelem. Na realidade, uma revisão do programa só irá ocorrer se uma ação a necessária para satisfazer uma intenção i cancelar uma propriedade P que também é uma intenção (*i.e.* é a propriedade associada a uma intenção), uma vez que ao cancelar esta propriedade, a restrição de integridade utilizada para abduzir o curso de ação para se chegar a P, em específico a restrição $P \Leftarrow$ será violada.

4.3.2 Formação das Intenções

Uma vez definido o conjunto de desejos candidatos, o agente irá se comprometer em tentar satisfazê-los. Este comprometimento é representado através de intenções, que no X-BDI são divididas em dois tipos: intenções primárias e intenções relativas. As intenções primárias são essencialmente a transposição do conjunto de desejos candidatos escolhidos D'_C para novos predicados que representam as intenções do agente. As intenções primárias serão utilizadas no processo

de refinamento das intenções até que o agente possua um plano de ações concreto, que quando executado, resulta no cumprimento das propriedades desejadas. Intenções primárias são definidas da seguinte forma [MÓR 99b]:

Definição 4.11 (Intenções Primárias) *Sejam $\mathcal{D}$ o conjunto dos desejos do agente, $\mathcal{D}'_C$ o conjunto de desejos candidatos de $\mathcal{D}$. As* intenções primárias *de um agente é o conjunto:*

$$\{int\_that(D, Ag, P, T_I, Atr)/des(D, Ag, P, T_I, Atr) \in \mathcal{D}'_C\}$$

As intenções primárias podem ser vistas como planos de alto nível representando o comprometimento do agente com um determinado curso de ação. Conseqüentemente, o agente irá limitar o seu esforço de raciocínio ao cumprimento destes objetivos ao invés de reavaliar os seus desejos constantemente. A maneira de concretizar este comprometimento sofrerá refinamentos até o ponto onde o agente possui um conjunto de ações ordenadas no tempo, representando um plano concreto para a realização de um objetivo. O resultado final do processo de refinamento das intenções primárias é representado pelas intenções relativas.

As intenções relativas correspondem aos passos de planos concretos e ordenados no tempo visando a satisfação das intenções primárias estabelecidas anteriormente.

Definição 4.12 (Intenções Relativas) *Seja $\mathcal{I}_P$ o conjunto de intenções primárias. O* processo de planejamento *é um procedimento que, para cada $i \in \mathcal{I}_P$, irá gerar um conjunto $\mathcal{I}_R$ de ações ordenadas temporalmente que alcança i, tal que $\mathcal{B} \cup \mathcal{T}Ax \cup \mathcal{I}_P \cup \mathcal{I}_R$ é não-contraditório. O conjunto $\mathcal{I}_R$ são as intenções relativas do agente.*

Na definição observa-se que as intenções relativas compõem os sub-planos relativos ao cumprimento das intenções primárias [MÓR 99b]. A noção do comprometimento do agente com as intenções primárias escolhidas é atingida através da necessidade que as intenções relativas têm de serem não contraditórias em relação às intenções primárias.

4.3.3 Revisão de Intenções

Os processos de raciocínio utilizados pela implementação em ELP do X-BDI são muito custosos na medida em que se baseiam fortemente na revisão de literais, cuja ênfase na implementação foi facilitar a prova de propriedades ao invés de obter eficiência computacional. Desta forma, é interessante que a reconsideração do curso de ação de um agente X-BDI não ocorra constantemente [MÓR 99b]. A alternativa proposta pelo autor foi a utilização de restrições de integridade que, geradas no momento da definição das intenções, funcionam como gatilhos para a sua revisão. Conseqüentemente, o agente não precisará re-avaliar seu curso de ação de maneira constante, sendo que esta re-avaliação irá ocorrer automaticamente no momento que as restrições de integridade forem violadas.

Definição 4.13 (Restrições-Gatilho das Intenções) *Sejam $\mathcal{B}$ o conjunto de crenças do agente e $\mathcal{I}$ suas intenções. São adicionadas a $\mathcal{B}$ as seguintes* restrições-gatilho*:*

- $(\bot \Leftarrow Now > T, not\ rev\_int)$, *para cada* $(int\_that(I, Ag, P, T_I, Atr), int\_to(I, Ag, Act, T_I, Atr) \in \mathcal{I})$;
- $(\bot \Leftarrow happens(E, T_i, T_f), act(E, Act), not\ rev\_int)$, *para cada* $(int\_to(I, Ag, Act, T_I, Atr) \in \mathcal{I})$.

Definição 4.14 (Restrições-Gatilho dos Desejos) *Sejam $\mathcal{D}$ o conjunto dos desejos do agente e $\mathcal{D}'$ o conjunto dos desejos elegíveis de $\mathcal{D}$. Define-se `restrições-gatilho dos desejos` como:*

- *Para cada* $(des(D, Ag, P, T_D, Atr) \leftarrow Body) \in \mathcal{D}$ *e não pertencente a $\mathcal{D}'$ com importância A maior do que a maior importância nas intenções, define-se a restrição-gatilho* $\bot \Leftarrow Body, not\ rev\_int$;
- *dado o conjunto de ações Δ abduzido pelo agente (vide definição 4.10), para cada* $des(D, Ag, P, T_D, Atr) \in (\mathcal{D}' \in \mathcal{D}'_C)$ *com importância A maior do que a maior importância nas intenções e pré-condições $C_1, \ldots, C_n$, define-se a restrição-gatilho* $\bot \Leftarrow C_1, \ldots, C_n, not\ rev\_int$, *onde $C_i (1 \leq i \leq n)$ são as condições que o agente não conseguiu realizar quando da seleção do conjunto de desejos candidatos.*

A partir das definições 4.13 e 4.14 tem-se que a reconsideração das intenções é baseada em gatilhos, que são representados na forma de restrições de integridade (Seção 4.1.3) na base de crenças. Uma vez que a base de crenças sofre um processo de revisão sempre que for atualizada através dos predicados $happens/3$ e $sense/2$ (Seção 4.2), tem-se que a cada revisão de crenças é possível que uma destas restrições seja violada. No momento em que uma restrição de integridade relacionada aos gatilhos das definições 4.13 e 4.14 for violada, o processo de deliberação será reiniciado.

4.4 Considerações

O objetivo inicial do X-BDI de constituir um modelo de agentes que pudesse ser diretamente executado foi atingido na medida em que a sua especificação em ELP pode ser interpretada pela implementação da ELP em Prolog. Desta forma, teoricamente elimina-se as possíveis discrepâncias entre especificação e implementação. Entretanto, a possibilidade de falhas na implementação da ELP significa que nem todas as propriedades definidas no modelo teórico podem ser verificadas na implementação do X-BDI.

Uma das características mais importantes do X-BDI é a sua capacidade de formar planos em tempo de execução a fim de satisfazer os desejos do agente. Esta característica está presente em um número muito pequeno de trabalhos conhecidos [NID 02]. Entretanto, a utilização do processo de revisão de literais inúmeras vezes durante a formação de planos resulta na limitação da classe de problemas para os quais o X-BDI é capaz de gerar planos em um tempo aceitável. Esta limitação é resultado, em grande parte, do fato de a implementação da SLX não ter sido focada em eficiência, mas sim em facilitar a prova de teoremas sobre as suas propriedades lógicas. Conseqüentemente, a substituição do processo de planejamento presente no X-BDI original por um processo mais eficiente é desejável na medida que permitirá ao agente tratar um número maior de problemas.

Parte II

Descrição do Trabalho

Capítulo 5

X-BDI Estendido

Considerando o objetivo deste trabalho de definir um mapeamento entre o raciocínio meios-fim de agentes BDI e planejamento proposicional, escolheu-se o X-BDI como modelo de agentes para a implementação de um protótipo da utilização deste mapeamento. Desta forma, este capítulo descreve as extensões criadas no modelo X-BDI no sentido de que ele possa utilizar um algoritmo de planejamento proposicional que realize o processo de raciocínio meios-fim. A Seção 5.1 descreve as modificações realizadas na estrutura cognitiva de um agente X-BDI; a Seção 5.2 descreve as modificações realizadas no processo de raciocínio do agente para permitir a utilização de um módulo de planejamento externo; e a Seção 5.3 descreve as alterações necessárias para adaptar os gatilhos de reconsideração do agente às modificações descritas nas seções anteriores. Finalmente, a Seção 5.4 contém considerações sobre o que foi desenvolvido neste capítulo.

5.1 Estrutura Cognitiva do Agente

A estrutura cognitiva de um agente X-BDI possui os componentes tradicionais de um agente BDI, *i.e.* um conjunto de Crenças, Desejos e Intenções. Além disto, dada a sua definição em lógica estendida, ele possui também um conjunto de axiomas de tempo definidos a partir de uma variação do *Cálculo de Eventos* [MÓR 99b, KOW 86]. Na extensão proposta neste trabalho, as definições de crenças e desejos na estrutura do agente permanecem as mesmas do X-BDI original. Entretanto, a estrutura cognitiva do agente é alterada para que a mesma acomode a existência de um módulo de planejamento externo (Definição 5.1).

Definição 5.1 (Estrutura Cognitiva do Agente) *A estrutura cognitiva modificada de um agente X-BDI é uma tupla* $\mathcal{A}g = \langle \mathcal{B}, \mathcal{D}, \mathcal{I}, \mathcal{T}Ax, Plan \rangle$*, onde:*

- $\mathcal{B}$ *é o conjunto de crenças do agente;*
- $\mathcal{D}$ *é o conjunto de desejos do agente;*

- $\mathcal{I}$ *é o conjunto de intenções do agente;*
- $\mathcal{T}Ax$ *é o conjunto de axiomas de tempo;*
- $Plan$ *é uma função de planejamento proposicional (Definição 3.6).*

Considerando-se que as intenções são o resultado do processo de raciocínio meios-fim de um agente, sua definição deve acomodar a sua origem em um plano proposicional (Definição 5.2).

Definição 5.2 (Conjunto de Intenções) *As intenções de um agente no tempo T formam o conjunto:*

$$\mathcal{I} = \{X / X = int\_that(I, Ag, P, T_I, Atr) \vee X = int\_to(I, Ag, Act, T_I, Atr)\}$$

onde I é o identificador da intenção, P é a propriedade que se deseja tornar verdadeira, Ag é um agente, Atr é uma lista de atributos, e Act é uma ação, e tal que:

1. $holds\_at(int\_that(I, Ag, P, T_I, Atr), T)$ *ou* $holds\_at(int\_to(I, Ag, Act, T_I, Atr), T)$*;*
2. $\forall I.int\_that(I, Ag, P, T_I, Atr) \in \mathcal{I}.(Now \leq T_I)$*;*
3. $\forall I.int\_to(I, Ag, Act, T_I, Atr) \in \mathcal{I}.(Now \leq T_I)$*;*
4. $\forall I.int\_to(I, Ag, Act, T_I, Atr) \in \mathcal{I}.(\mathcal{B} \cup \mathcal{T}Ax \not\models_P (happens(E, T_I, T_F), act(E, Act)))$*;*
5. $\exists \Delta.(Plan(\Pi) = \Delta)$*, onde:*
   (a) $\Pi = \langle \Xi, \mathbf{I}, \mathbf{G} \rangle$*;*
   (b) Ξ *são as ações em* $\mathcal{B}$ *que o agente pode realizar;*
   (c) $\mathbf{I} = \{b \in \mathbf{I}\ sse\ \{\mathcal{B} \cup \mathcal{T}Ax\} \models_P holds\_at(bel(A, b), T).(Now \geq T)\}$*;*
   (d) $\mathbf{G} = \{P \in \mathbf{G}\ sse\ \exists I.int\_that(I, Ag, P, T, Atr) \in \mathcal{I}\}$

   um agente tem a intenção de que uma propriedade P seja verdadeira no tempo T sse $\mathcal{B} \cup \mathcal{T}Ax \cup \mathcal{R} \models_P holds\_at(int\_that(I, Ag, P, Atr), T)$, e ele tem a intenção de executar a ação Act no tempo T sse $\mathcal{B} \cup \mathcal{T}Ax \cup \mathcal{R} \models_P holds\_at(int\_to(I, Act, P, Atr), T)$.

Considerando a existência de dois tipos de intenções no X-BDI, Intenções Primárias (int_that), que se referem a propriedades almejadas, e Intenções Relativas (int_to), que se referem a ações a fim de concretizar estas propriedades, o seu relacionamento com o problema de planejamento é diferenciado. Um agente não pode ter como intenção algo no passado ou que já é verdadeiro. Além disto, intenções não podem ser impossíveis, isto é, deve existir pelo menos um plano executável pelo agente cujo resultado é um estado de mundo onde a intenção é verdadeira. Esta definição

representa a primeira mudança feita no X-BDI realizada neste trabalho. No X-BDI original a possibilidade de uma propriedade era verificada através da abdução de uma teoria de ações em Cálculo de Eventos que a tornasse verdadeira (Definição 4.6). Este processo essencialmente realizava o planejamento de um curso de ações para realizar a propriedade. Na extensão definida neste trabalho, modifica-se o processo de planejamento a fim de abstraí-lo da definição operacional do X-BDI, permitindo que qualquer processo de planejamento que satisfaça as condições da Definição 3.6 (vide Seção 3.1) seja utilizado no X-BDI.

5.2 Processo de Raciocínio do Agente

O processo de raciocínio realizado pelo X-BDI é iniciado com a seleção dos Desejos Elegíveis, que representam os desejos cuja pré-condição foi satisfeita e que ainda não estão satisfeitos. A seguir, são gerados os Desejos Candidatos, que representam um conjunto consistente e possível de Desejos Elegíveis, que serão adotados como Intenções Primárias. A fim de satisfazer as Intenções Primárias, o processo de planejamento cria uma seqüência de ações ordenadas no tempo que constituem as Intenções Relativas.

Desejos Elegíveis possuem restrições de racionalidade similares àquelas impostas sobre as intenções no sentido de que um agente não desejará algo no passado ou algo que o agente acredite que ocorrerá sem sua interferência. As crenças do agente também devem suportar as pré-condições do desejo definidas no $Body$. No processo de raciocínio do agente, estes desejos darão origem a subconjuntos mutuamente consistentes organizados segundo uma relação de ordem parcial.

O processo de seleção de Desejos Candidatos busca escolher dentre os Desejos Elegíveis, um subconjunto contendo apenas desejos possíveis e internamente consistentes. Um desejo possível é aquele que possui uma propriedade P que pode ser satisfeita através de uma seqüência de ações. A fim de escolher entre múltiplos conjuntos de Desejos Candidatos, o X-BDI faz uso de construções da ELP que permitem definir revisões preferidas. Desta forma, o X-BDI define uma relação de preferência dos desejos a partir de um conjunto de revisões preferidas gerado com base nas prioridades dos desejos. Através desta relação de preferência, um grafo de preferência de desejos é gerado relacionando todos os sub-conjuntos de Desejos Elegíveis.

Definição 5.3 (Conjunto dos Desejos Candidatos) *Sejam $\mathcal{D}$ o conjunto de desejos de um agente e $\mathcal{D}'$ o conjunto de desejos elegíveis de $\mathcal{D}$ com um grafo de preferência associado a ele. Chama-se* conjunto de desejos candidatos *qualquer conjunto:*

$$\mathcal{D}'_C = \{des(D, Ag, P, T_D, Atr)/(des(D, Ag, P, T_D, Atr) \in \mathcal{D}') \wedge \exists\Delta.(Plan(\Pi) = \Delta\}$$

onde $\Pi = \langle \Xi, P_B, P_D \rangle$ é um problema de planejamento criado a partir das crenças e desejos do agente da seguinte forma:

1. *Ξ são as ações em $\mathcal{B}$ que o agente pode realizar;*
2. *P_B é o conjunto das propriedades P que o agente acredita no momento do planejamento T (Definição 4.4);*
3. *P_D é o conjunto das propriedades P de todos os desejos presentes no subconjunto de Desejos Elegíveis com maior valor de preferência no grafo de preferência de desejos para qual um plano (Definições 3.5 e 3.6) puder ser gerado.*

A formação dos Desejos Candidatos é a modificação mais significativa realizada neste trabalho em relação ao X-BDI original [MÓR 99a], onde a possibilidade de um determinado desejo era verificada através da abdução de uma teoria em Cálculo de Eventos onde a crença na validade da propriedade P desejada possa ser verdadeira. Este processo de abdução é, na realidade, uma forma de planejamento. Como o objetivo principal deste trabalho é a separação do processo de planejamento antes embutido no X-BDI, a noção de possibilidade de um desejo teve que ser re-definida. Desta forma, tem-se que um conjunto de Desejos Candidatos é o subconjunto de Desejos Elegíveis com valor de preferência mais alto e cujas propriedades podem ser satisfeitas. A satisfatibilidade é verificada através da execução de um planejador proposicional tendo como estado inicial do planejamento as propriedades acreditadas no momento do planejamento. Conseqüentemente, o processo de planejamento é realizado entre a geração dos Desejos Elegíveis e dos Desejos Candidatos, o que pode ser visto na Figura 5.1.

As propriedades P presentes nos Desejos Candidatos são utilizadas na geração do conjunto de Intenções Primárias, que representam o comprometimento do agente com a satisfação das propriedades escolhidas na formação dos desejos (Definição 5.4).

Definição 5.4 (Intenções Primárias) *Sejam $\mathcal{D}$ o conjunto dos desejos do agente, $\mathcal{D}'_C$ o conjunto de desejos candidatos de $\mathcal{D}$. As* intenções primárias *de um agente é o conjunto:*

$$\{int\_that(D, Ag, P, T, Atr)/des(D, Ag, P, T, Atr) \in \mathcal{D}'_C\}$$

Uma vez tendo o agente se comprometido com o cumprimento de determinados objetivos, ele deverá se comprometer com o curso de ações que irão cumprir tais objetivos. Este comprometimento visa evitar o problema de Buridan (Problema 2.1), e garantir a execução de um curso consistente de ações por parte do agente. Considerando a realização prévia do planejamento proposicional, os passos do plano utilizado pelo agente para levar a cabo seus objetivos devem simplesmente ser mapeados do plano proposicional para os estados mentais do agente, como visto na Definição 5.5.

Definição 5.5 (Intenções Relativas) *Seja $\mathcal{I}_P$ o conjunto de intenções primárias cuja satisfatibilidade foi comprovada pela existência de um plano proposicional Δ de operadores ordenados no*

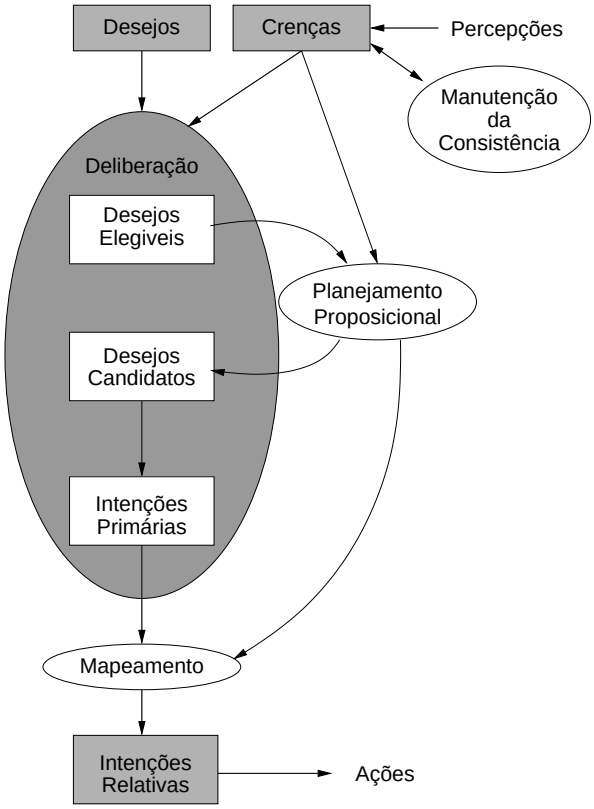

Figura 5.1: Esquema modificado do X-BDI.

tempo. O mapeamento *do plano Δ para um conjunto de intenções relativas $\mathcal{I}_R$ é um procedimento que, para cada $O_i \in \Delta$, irá gerar uma intenção relativa $int\_to(I, Ag, O_i, Now + i, Atr) \in \mathcal{I}_R$, e tal que $\mathcal{B} \cup \mathcal{T}Ax \cup \mathcal{I}_P \cup \mathcal{I}_R$ é não-contraditório. O conjunto $\mathcal{I}_R$ são as intenções relativas do agente.*

5.3 Revisão de Intenções

O esforço computacional e o tempo necessário para re-considerar todo o conjunto de intenções de um agente com recursos limitados normalmente é significativo em relação à taxa de mudança do ambiente. Desta forma, a reconsideração das intenções não deve ocorrer constantemente, mas apenas quando o mundo muda de forma que seus planos estejam ameaçados, ou quando uma oportunidade de satisfazer objetivos mais importantes se apresenta. Conseqüentemente, o X-BDI utiliza um conjunto de "gatilhos" de reconsideração gerados no momento da seleção das intenções, e que causam a reconsideração do agente quando ativados. Estes gatilhos também tiveram que ser atualizados para comportar as novas noções de possibilidade de desejos e intenções, como

pode ser visto nas Definições 5.6 e 5.7.

Definição 5.6 (Restrições-Gatilho das Intenções) *Sejam $\mathcal{B}$ o conjunto de crenças do agente e $\mathcal{I}$ suas intenções. E seja P_I um conjunto de propriedades $P_I = \{P \in P_I$ sse $int\_that(I, Ag, P, T_I, Atr)\}$ São adicionadas a $\mathcal{B}$ as seguintes* restrições-gatilho*:*

- $\bot \Leftarrow Now \leq T, \bigwedge_{p \in P_I}(holds\_at(bel(Ag, P, Now), T))$;
- $\bot \Leftarrow Now > T, \bigvee_{p \in P_I}(not\ holds\_at(bel(Ag, P, Now), T))$;

As alterações realizadas nos gatilhos de reconsideração do X-BDI visaram permitir a manutenção das condições de reconsideração estabelecidas por Bratman [BRA 88]. Em especial, se todas as Intenções Primárias do agente forem satisfeitas antes do tempo planejado para elas, então o agente irá reiniciar o processo deliberativo, já que atingiu seus objetivos. Em contrapartida, se uma das Intenções Primárias não tiver sido atingida no tempo planejado, o agente deverá re-considerar suas intenções pois seus planos falharam.

Definição 5.7 (Restrições-Gatilho dos Desejos) *Sejam $\mathcal{D}$ o conjunto dos desejos do agente e $\mathcal{D}'$ o conjunto dos desejos elegíveis de $\mathcal{D}$. Define-se* restrições-gatilho dos desejos *como:*

- *Para cada $(des(D, Ag, P, T_D, Atr) \leftarrow Body) \in \mathcal{D}$ e não pertencente a $\mathcal{D}'$ com importância A maior do que a maior importância nas intenções, define-se a restrição-gatilho $\bot \Leftarrow Body, not\ rev\_int$;*
- *Dado o conjunto de ações Δ gerado pelo agente (vide Definição 5.3), para cada $des(D, Ag, P, T_D, Atr) \in (\mathcal{D}' \in \mathcal{D}'_C)$ com importância A maior do que a maior importância nas intenções, define-se a restrição-gatilho $\bot \Leftarrow C_1, \ldots, C_n, not\ rev\_int$, onde $C_i (1 \leq i \leq n)$ são as condições que o agente não conseguiu realizar quando da seleção do conjunto de desejos candidatos.*

Se um desejo de prioridade maior do que os desejos escolhidos se tornar possível através de sua pré-condição, então o agente irá re-considerar seus desejos para aproveitar a oportunidade surgida. A reconsideração é completamente baseada em restrições de integridade sobre as crenças. Desta forma, na medida em que as crenças são revisadas a cada ciclo de sensoriamento, é possível que uma reconsideração ocorra devido à ativação dos "gatilhos" de reconsideração.

5.4 Considerações

O mapeamento entre os estados mentais descritos no formalismo do X-BDI e formalismos de planejamento proposicional tem o potencial de ser aplicado a outros formalismos de agentes BDI.

Uma hipótese levantada ao longo deste trabalho é a utilização deste mapeamento no sentido de implementar a construção da árvore de mundos possíveis utilizadas em sistemas derivados do PRS de maneira computacionalmente aceitável.

O formalismo de planejamento proposicional pode ser utilizado também como uma interface para acesso a uma biblioteca de planos. Isto significa que, ao invés de o algoritmo de planejamento tentar encadear as ações do agente em tempo de execução, ele pode simplesmente possuir um repositório de planos acessados através de um algoritmo de busca que utilize o estado inicial e o estado final como chaves de busca. Esta possibilidade representa um possível ponto de intersecção entre os formalismos do PRS e do X-BDI. Além disto, a utilização deste tipo de "algoritmo" de planejamento também pode ser utilizada como um método de aprendizado do agente, na medida em que os planos gerados pelo agente puderem ser armazenados neste repositório, evitando que o agente tenha de realizar o planejamento em situações que o agente já enfrentou pelo menos uma vez.

Um aspecto da interação do Graphplan com o X-BDI levantado no decorrer deste trabalho é a utilização pelo agente das informações sobre o domínio do problema inferidas ao longo do planejamento. Mais especificamente, o agente poderia aproveitar de alguma forma as informações deduzidas pelo Graphplan e representadas no grafo de planejamento para ampliar o seu conhecimento sobre o domínio de problema onde está agindo. Em um nível mais básico, o agente poderia guardar este grafo entre as múltiplas execuções do planejamento no momento da escolha dos desejos candidatos, e assim saber que conjuntos de desejos são impossíveis ou mutuamente exclusivos. Estas informações estão armazenadas na forma de relações de exclusão mútua no grafo de planejamento e dos conjuntos de *no-goods* na tabela de memoização.

As modificações realizadas no X-BDI alteraram o seu funcionamento de modo que este utilize planejadores proposicionais como base do processo de raciocínio meios-fim e como verificadores de possibilidade no processo de raciocínio prático, como pode ser visto na Figura 5.1. No decorrer destas modificações, novas definições de desejos e intenções foram criadas a fim de que o X-BDI mantivesse as propriedades teóricas presentes na sua versão original, em especial no tocante à definição de possibilidade de desejos e intenções. Além disto, foi necessário definir um mapeamento entre os componentes estruturais de um agente BDI e problemas de planejamento proposicionais. O resultado destas modificações pode ser observado na prática através da implementação descrita no Capítulo 6, e dos estudos de caso descritos no Capítulo 7.

Capítulo 6

Implementação do Protótipo

Considerando o objetivo deste trabalho de integrar uma ferramenta BDI com um algoritmo de planejamento proposicional, e a escolha do X-BDI e do Graphplan, respectivamente, como representantes destes componentes para a implementação de um protótipo, foi necessária a criação de uma arquitetura que comportasse as suas implementações individuais. Neste sentido, tomou-se como base uma implementação em Prolog do X-BDI, que foi modificada com a inclusão dos mecanismos de mapeamento entre os estados mentais do agente para problemas de planejamento proposicional e de planos proposicionais de volta para os estados mentais do agente (Capítulo 5). Além disto, uma versão do algoritmo Graphplan foi implementada segundo a descrição original do algoritmo [BLU 97], e utilizando o formalismo proposicional descrito na Seção 3.1.

Este capítulo irá descrever a arquitetura geral do protótipo implementado (Seção 6.1) e as particularidades presentes na implementação do Graphplan realizada para este trabalho (Seção 6.2). No final do capítulo (Seção 6.3) é descrita uma ferramenta gráfica de operação e visualização do funcionamento de agentes criados utilizando a implementação descrita neste capítulo.

6.1 Arquitetura da Implementação

O protótipo implementado neste trabalho é composto essencialmente de três partes, o *kernel* do X-BDI, implementado em Prolog, uma biblioteca de planejamento contendo uma implementação em C++ do Graphplan, e uma interface gráfica em Java utilizada para facilitar a operação do X-BDI e a visualização da interação do mesmo com o ambiente; esta arquitetura é esquematizada na Figura 6.1.

A interface gráfica *Agent Viewer* se comunica com o X-BDI através de *sockets*, enviando para o agente as entradas vindas do ambiente onde o agente está inserido e recebendo o resultado da deliberação do agente a cada intervalo de tempo. Uma vez iniciado o funcionamento de um agente no X-BDI, este se comunica com a biblioteca de planejamento através de arquivos no

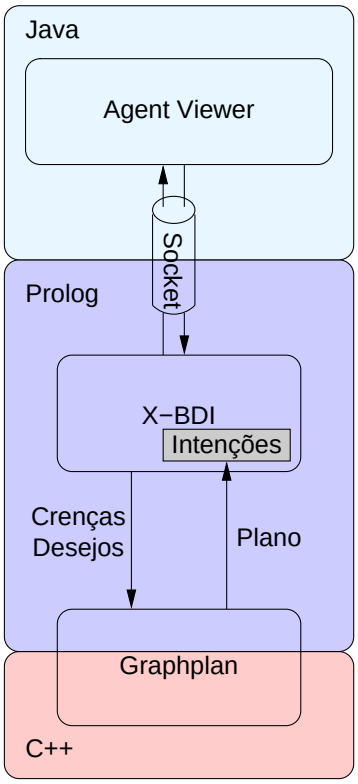

Figura 6.1: Arquitetura da Solução

sistema operacional e da interface PROLOG/C++ presente no SICStus Prolog [SIC 01].

Durante o processo deliberativo de um agente X-BDI, é necessário que o agente gere um conjunto de desejos candidatos cuja viabilidade é verificada por um algoritmo de planejamento (Definição 5.3). Neste momento o X-BDI irá realizar o mapeamento das suas crenças e dos desejos cuja viabilidade deve ser verificada gerando um problema de planejamento que é escrito em um arquivo do sistema operacional. Quando este mapeamento for completado, o X-BDI irá invocar a biblioteca de planejamento, ativando o Graphplan, que tentará encontrar um plano para o problema especificado no arquivo. Esta invocação é realizada através de um predicado em Prolog que, ao ser consultado, inicia a execução do planejador. A consulta ao predicado de planejamento tem sucesso no caso do algoritmo de planejamento encontrar um plano para o problema proposto e falha caso o algoritmo determine que não há solução para tal problema. Além disto, quando o algoritmo encontra uma solução para o problema proposto pelo X-BDI, o plano resultante é escrito em outro arquivo do sistema operacional. O X-BDI irá gerar sub-conjuntos de desejos elegíveis consistentes e verificar a existência de um plano que os satisfaça sistematicamente, até que a consulta ao predicado de planejamento tenha sucesso, ou que todos os sub-conjuntos de desejos elegíveis tenham sido provados impossíveis.

Quando um conjunto de desejos elegíveis for gerado, o processo de deliberação do agente procede até a geração das intenções relativas, momento em que o plano gerado pela biblioteca de planejamento é utilizado em um processo de mapeamento na geração das intenções relativas (Definição 5.5).

6.2 Implementação do Graphplan

A versão do Graphplan utilizada neste trabalho usa um formalismo diferente do STRIPS básico presente na sua versão original [BLU 97], especificamente, o formalismo da implementação realizada neste trabalho está descrito na Seção 3.1. A principal diferença deste formalismo em relação ao STRIPS básico é a presença de literais de primeira ordem ao invés de átomos. A maior expressividade do formalismo não incorre em perda de eficiência do algoritmo, visto que é possível converter os literais de primeira ordem utilizados no formalismo em átomos através de transformações sintáticas, desde que as seguintes restrições sejam observadas:

- As especificações de estado $\mathbf{I}$ e $\mathbf{G}$ (vide definições na Seção 3.1) só podem ser feitas através de literais *ground*[1];
- As descrições de operadores não podem criar novos literais através das suas pós-condições.

Uma outra modificação da linguagem de descrição do problema utilizada neste trabalho é a atribuição de tipos aos objetos utilizados no problema. No Graphplan original a descrição de problemas permite que os termos em um predicado sejam identificados com tipos, permitindo que proposições irrelevantes sejam desconsideradas durante a criação do grafo de planejamento, resultando em aumento de velocidade no algoritmo de extração de solução. Na versão do Graphplan implementada para este trabalho os termos de um problema não podem ser tipados, visto que a linguagem utilizada pelo X-BDI, *i.e.* a lógica de primeira ordem do Prolog, não os utiliza. Entretanto, é possível definir tipos para termos de maneira implícita utilizando o mesmo formalismo do X-BDI, o que resulta em um ganho de desempenho do planejador muito semelhante àquele que pode ser obtido com tipos explícitos na linguagem. Esta técnica de definição de tipos consiste em especificar proposições com o nome do tipo desejado e aplicá-las aos objetos cujo tipo se deseja definir, e então especificar estas propriedades no estado inicial e nas pré-condições dos operadores. O Exemplo 6.1 contém uma descrição de problema onde os tipos são representados desta forma. Neste exemplo o objeto `a` é representado como sendo do tipo `typeA` pelo predicado `typeA(a)`, e o parâmetro `A` do operador `typedOper` é declarado como sendo do tipo `typeA` através da pré-condição `typeA(A)`.

[1] Literais sem variáveis livres.

Exemplo 6.1 *Utilização de tipos no formalismo de planejamento*

```
start(typeA(a), typeB(b), someprop(a), anotherprop(b))
...
operator typedOper(A, B)
preconds(typeA(A), typeB(B), someprop(A), anotherprop(B))
effects(not someprop(A))
```

O efeito desta solução é delegar a análise de tipos previamente realizada por um pré-processador para o algoritmo de expansão do grafo. Considerando o fato de que o algoritmo de expansão do grafo tem complexidade linear, o decréscimo em desempenho resultante desta mudança em relação ao Graphplan original não é grande. Além disto, o algoritmo de extração de solução não será afetado pela presença de proposições irrelevantes no modelo do problema pois a informação de tipos denotada nos predicados irá evitar que o algoritmo de expansão introduza proposições irrelevantes no grafo de planejamento, ou, pelo menos, irá evitar a introdução de proposições irrelevantes que poderiam ter sido evitadas utilizando a análise de tipos explicitamente definidos.

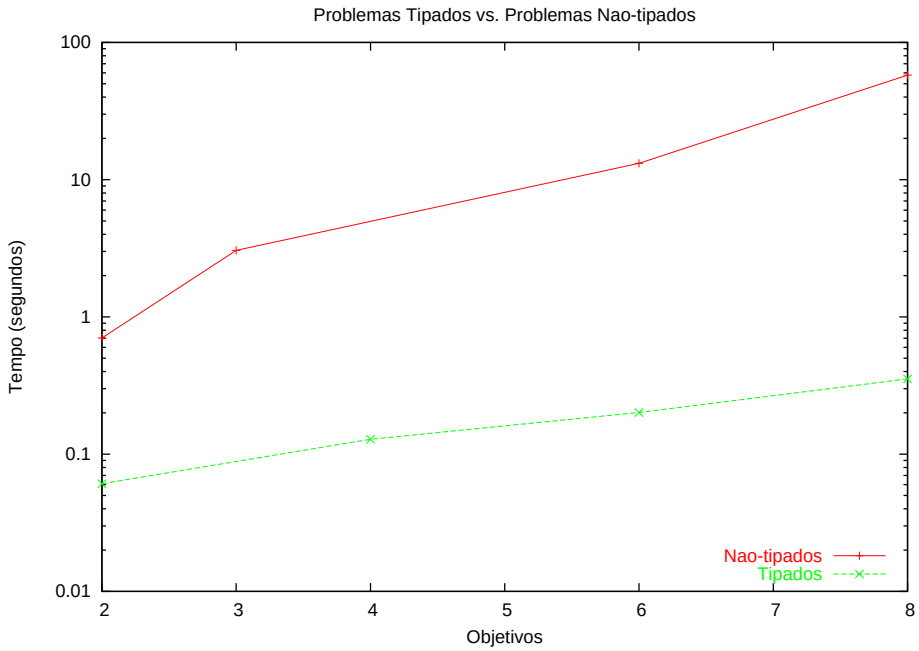

Figura 6.2: Tempos de execução para o domínio *Rockets*.

Avaliações empíricas mostram que a atribuição de tipos a objetos em domínios de planejamento abrevia fortemente o tempo de busca por solução, o que pode ser observado em uma comparação realizada com problemas do domínio *Rockets* [BLU 97] sendo processados pela implementação do

Graphplan usada neste trabalho. A Figura 6.2 mostra os tempos de execução de problemas tipados e não-tipados, com um número cada vez maior de objetivos a serem alcançados no domínio citado anteriormente. Em um problema com três objetivos, o tempo de processamento do problema tipado é inferior a um segundo enquanto que o tempo para o mesmo problema modelado sem a representação de tipos se aproxima de dez segundos. Na medida em que o número de objetivos aumenta para oito, o tempo de execução para o problema sem representação de tipos ultrapassa um minuto enquanto que a sua versão tipada continua abaixo de um segundo.

6.3 AgentViewer

A ferramenta AgentViewer foi criada para este trabalho no intuito de facilitar a utilização do X-BDI através de uma ferramenta gráfica que permita tanto a ativação e configuração do X-BDI como a interação remota com um agente rodando neste ambiente. Esta ferramenta foi implementada utilizando a biblioteca gráfica Swing presente na linguagem Java, sendo que a Figura 6.3 mostra uma tela deste sistema.

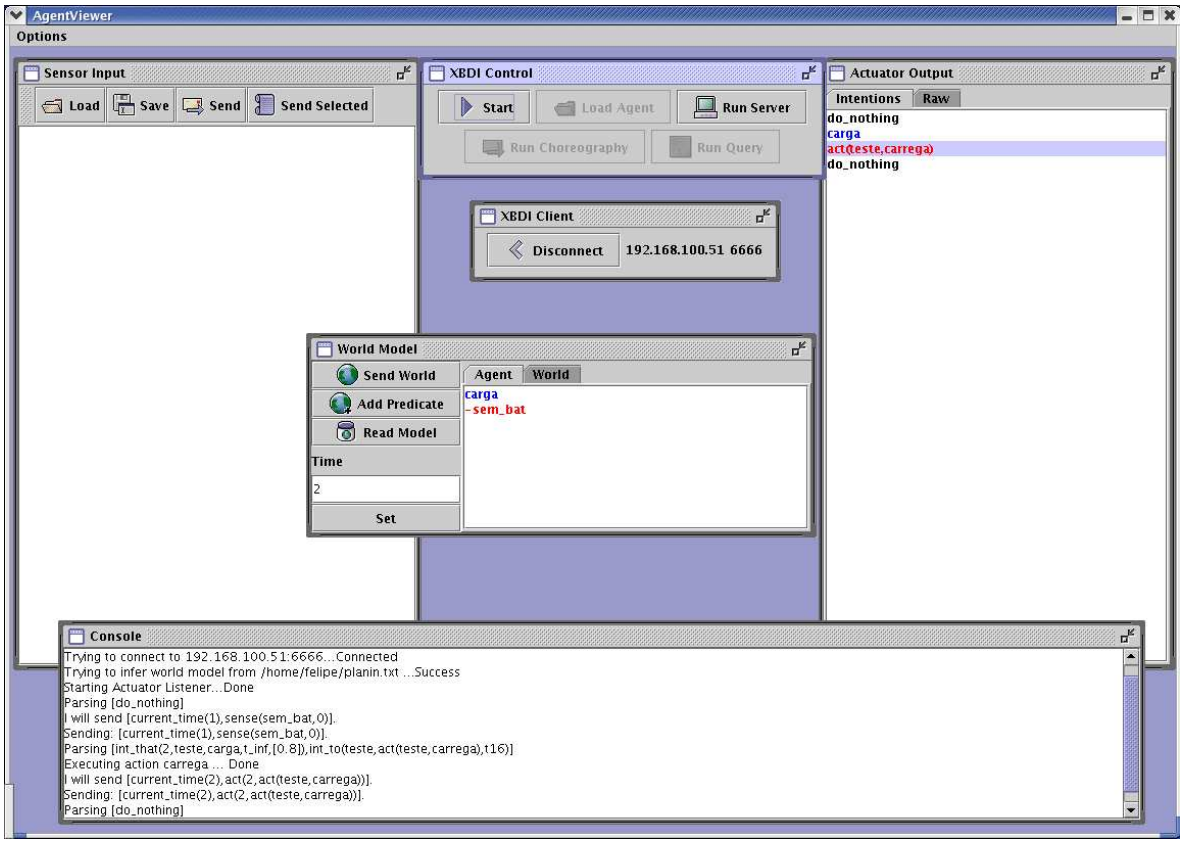

Figura 6.3: A ferramenta AgentViewer

As principais funcionalidades do AgentViewer são o controle da execução do X-BDI, descrito na Seção 6.3.1, a comunicação com um agente X-BDI via *sockets*, descrita na Seção 6.3.2 e a representação de um modelo de mundo com o qual um agente X-BDI interage, descrita na Seção 6.3.3. Informações a respeito do estado de execução do AgentViewer e da instância do X-BDI executando sob o seu controle são apresentadas pela janela `Console`.

6.3.1 Controle do X-BDI

A ferramenta X-BDI é capaz de controlar uma instância do Prolog executando em um processo subordinado à máquina virtual Java. Nesta instância do Prolog o X-BDI é carregado, juntamente com um agente especificado através do AgentViewer. O controle sobre o X-BDI é feito através de consultas Prolog realizadas pela ferramenta, sendo que o usuário do AgentViewer pode, a qualquer momento fazer consultas quaisquer na instância Prolog subjacente.

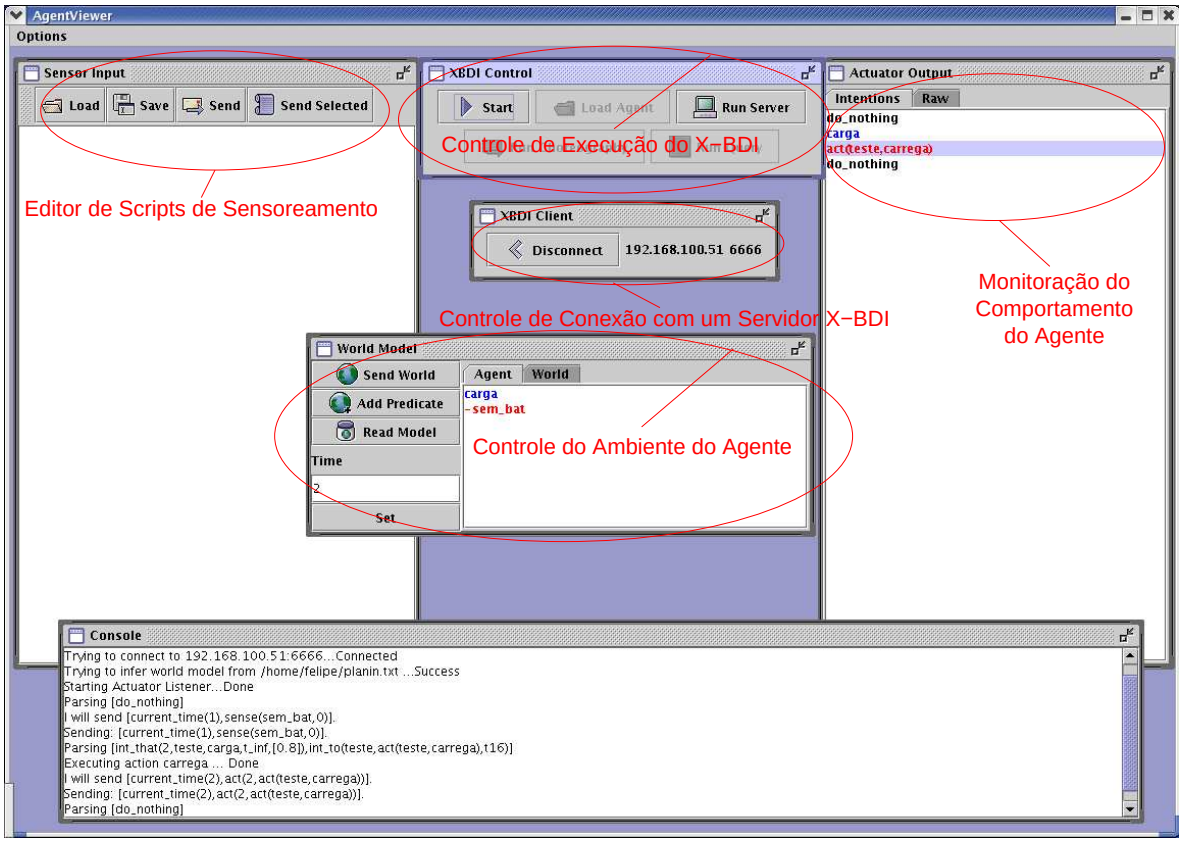

Figura 6.4: Funcionalidades do AgentViewer

Estas funcionalidades podem ser acessadas através dos seguintes botões na janela `X-BDI Control` (Figura 6.4):

- `Start/Restart`: Inicia ou reinicia um processo do sistema operacional que executa uma instância do SICStus Prolog, onde o X-BDI é carregado e executado;
- `Load Agent`: Instrui o X-BDI subjacente a carregar um agente de um arquivo no sistema operacional;
- `Run Server`: Instrui o X-BDI a executar no modo *kernel*, onde o agente interage com o ambiente através de uma conexão de rede usando *sockets* do sistema operacional, recebendo informações sobre o ambiente e enviando suas decisões através desta conexão.
- `Run Choreography`: Instrui o X-BDI a carregar uma seqüência pré-definida de entradas do ambiente, que é chamada de coreografia, sendo o resultado da deliberação do agente apresentado pela janela `Console`;
- `Run Query`: Permite ao usuário executar uma consulta no Prolog sob execução subordinado ao AgentViewer.

6.3.2 Comunicação com o X-BDI

Quando uma instância do X-BDI executando no modo *kernel* está disponível na rede (ou foi criada na máquina local através do AgentViewer), é possível ao usuário conectar-se a ela e realizar a interação com o agente X-BDI através de uma conexão utilizando a interface de *sockets* do sistema operacional. Esta funcionalidade é acessível através do botão `Connect` na janela `X-BDI Client`. Quando o AgentViewer for conectado a uma instância do X-BDI, as janelas `Sensor Input` e `Actuator Output` de interação com o agente se tornarão disponíveis para o usuário.

As janelas de interação com o agente disponibilizam, respectivamente, informações a respeito do envio de entradas para os sensores do agente e da recepção de ações de seus atuadores. Na janela `Sensor Input`, o usuário pode carregar um arquivo de coreografia e gerenciar o envio das suas informações para o agente. Na janela `Actuator Output` o usuário pode visualizar o resultado da deliberação do agente, sendo que esta informação é apresentada para o usuário tanto na forma exata como o agente a envia para o ambiente, como de forma formatada, destacando as propriedades presentes em intenções primárias (em azul) das ações presentes nas intenções relativas (em vermelho). A janela `Actuator Output` também é utilizada pela funcionalidade de controle do modelo de mundo descrita na Seção 6.3.3.

6.3.3 Modelo de Mundo

Além de facilitar a interação com o X-BDI, o AgentViewer é capaz de manter e manipular uma descrição de modelo de mundo que o usuário pode modificar livremente, enviar para o X-BDI, e interferir nos resultados da atuação do agente a fim de verificar seu comportamento. Esta

funcionalidade é controlada através da janela `World Model`, que contém uma representação das propriedades verdadeiras (em azul) e falsas (em vermelho) no estado atual do mundo.

O modelo de mundo mantido pelo AgentViewer é inferido no momento em que este se conecta a um agente X-BDI. Neste momento o AgentViewer carrega o último arquivo contendo um problema de planejamento criado pelo X-BDI na sua comunicação com o seu algoritmo de planejamento. Nesta descrição de problema de planejamento, o estado inicial representa todas as propriedades acreditadas pelo agente no momento da deliberação, que são utilizadas para construir o modelo de mundo inicial armazenado pelo AgentViewer. Caso a carga deste arquivo não seja possível, o modelo de mundo deverá ser definido pelo usuário.

Uma vez que o AgentViewer tenha uma representação de modelo de mundo, o usuário pode modificá-lo, e enviá-lo para o agente X-BDI. Estas funcionalidades podem ser acessadas através dos seguintes botões (Figura 6.4):

- `Send World`: Envia o modelo de mundo atual para o agente;
- `Add Predicate`: Adiciona um predicado especificado pelo usuário ao modelo de mundo, este predicado pode ser tanto positivo quanto negativo (indicado pelo símbolo `-`, ou a palavra reservada `not`), sendo que todos os demais predicados são considerados falsos por *default*;
- `Read Model`: Substitui o modelo de mundo atual por um outro presente em um arquivo de descrição de problema STRIPS especificado pelo usuário;
- `Time/Set`: Especifica um valor inteiro denotando o momento atual no modelo de mundo representado pelo AgentViewer. A cada envio do modelo para o agente este valor é incrementado a fim de denotar o aspecto temporal da evolução do modelo de mundo;
- Além destes botões o usuário pode apagar predicados no modelo de mundo ou inverter o seu valor verdade, o que é realizado através de *Context Menus* acessíveis com o botão direito do mouse;
- Ao receber o resultado da deliberação do agente pela janela `Actuator Output`, o usuário pode, através de *Context Menus*, executar as ações especificadas nas intenções relativas do agente sobre o estado de mundo mantido pelo AgentViewer, notificando ou não o sucesso de sua execução para o agente.

Através desta funcionalidade, a ferramenta AgentViewer não apenas provê maior flexibilidade no teste de um agente X-BDI, como também provê um mecanismo de injeção de falhas bastante útil no teste de um agente.

No próximo capítulo esta ferramenta será utilizada na descrição de dois estudos de caso. Um deles descreve uma célula de produção responsável por escalonar o processamento de peças em seus componentes. O outro estudo de caso descreve uma célula de produção onde é prevista a ocorrência de falhas, que são inseridas utilizando o *AgentViewer*.

Capítulo 7

Estudos de Caso

Este capítulo descreve os estudos de caso utilizados no teste da implementação realizada para este trabalho. Além de demonstrar a aplicabilidade da abordagem proposta na resolução de problemas práticos, os estudos de caso visam demonstrar características importantes da implementação desenvolvida. Em particular, a célula de produção descrita na Seção 7.1 foi utilizada para mensurar o tempo de execução e a escalabilidade do algoritmo de planejamento utilizado neste trabalho. Em contrapartida o estudo de caso da Seção 7.2 visa demonstrar a capacidade do agente em tratar falhas ocorridas no sistema ou na comunicação deste com o agente.

7.1 Célula de Produção

A utilização racional de equipamentos em instalações industriais é um problema complexo, em especial o escalonamento da utilização destes equipamentos. Este problema é potencializado quando as instalações são utilizadas a fim de se produzir múltiplos tipos de componentes, onde cada um utiliza um subconjunto dos equipamentos disponíveis. Neste trabalho utilizaremos um agente baseado no modelo BDI a fim modelar uma célula de produção como estudo de caso. A célula de produção BDI apresentada neste trabalho foi inspirada em uma desenvolvida pelo *Forschungszentrum Informatik*[1] (FZI) de Karlsruhe na Alemanha [LEW 95]. A célula de produção proposta é composta de sete dispositivos, uma Esteira Alimentadora, uma Esteira Depósito, quatro Unidades de Processamento e um Guindaste $L1$ de movimentação de componentes com liberdade de movimento sobre todos os demais dispositivos. Esta célula de produção é ilustrada na Figura 7.1.

Componentes a serem processados são introduzidos na célula de produção através da Esteira Alimentadora, e, uma vez processados por todas as Unidades de Processamento apropriadas, são retirados da célula através da Esteira Depósito. Cada Unidade de Processamento é responsável por

[1] Centro de Pesquisa em Informática

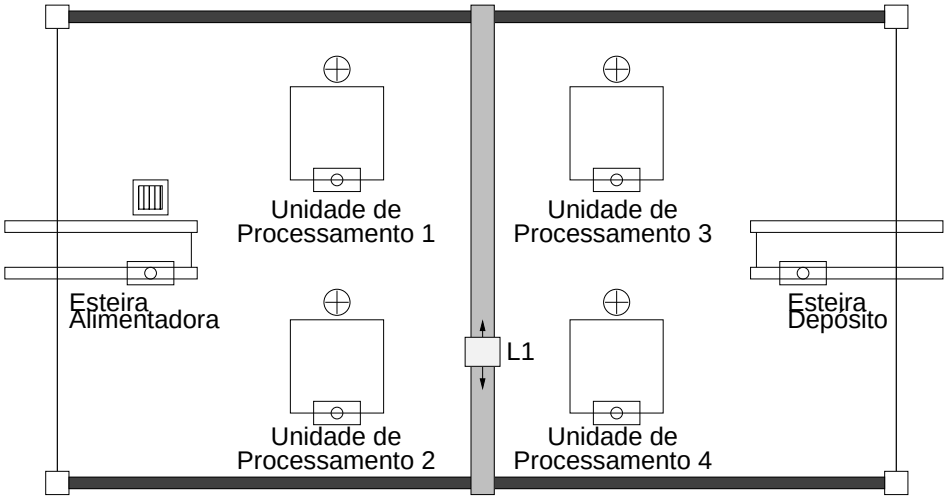

Figura 7.1: Uma Célula de Produção BDI.

realizar um tipo diferente de operação no componente sendo processado, e pode conter apenas um componente em um dado momento. Cada componente fornecido à célula deverá ser processado por uma ou mais Unidades de Processamento, o que será determinado pelo tipo do componente. Diferentes tipos de componente têm prioridades de processamento diferentes. O controle desta célula de produção será delegado a um agente BDI implementado utilizando o X-BDI, que deverá escalonar o trabalho da célula de produção através de suas crenças e desejos, re-escalonando o trabalho da célula quando ocorrer alguma mudança do estado do sistema.

O primeiro passo na modelagem de qualquer problema em um formalismo baseado em lógica de predicados é a escolha dos predicados utilizados para representar os tipos de objetos do problema e seus estados. Desta forma temos os seguintes predicados representando objetos:

- `bloc(B)` denota que `B` é um Componente a ser processado;
- `procUnit(P)` denota que `P` é uma Unidade de Processamento, Unidades de Processamento também são Dispositivos;
- `device(D)` denota que `D` é um Dispositivo;
- `feedBelt` representa a Esteira Alimentadora;
- `depositBelt` representa a Esteira do Depósito.

Da mesma forma temos os seguintes predicados representando estados do sistema:

- `over(B,D)` denota que o Componente `B` está sobre o Dispositivo `D`;

- `empty(P)` denota que a Unidade de Processamento `P` está vazia, isto é, não possui nenhum Componente sobre ela;

- `processed(B,P)` denota que o Componente `B` já foi processado pela Unidade de Processamento `P`;

- `finished(B)` denota que o Componente `B` já foi processado por todas as Unidades de Processamento apropriadas e foi removido da célula de produção.

A seguir são definidas as ações que o agente é capaz de realizar no contexto do problema analisado:

- A ação `process(B,P)` tendo como pré-condições `procUnit(P)`, `bloc(B)` e `over(B,P)`, e como efeito `processed(B,P)`. Esta ação representa o processamento que uma Unidade de Processamento `P` realiza em um Componente `B` depositado sobre ela;

- A ação `consume(B)` tendo como pré-condições `bloc(B)` e `over(B,depositBelt)` e como efeitos ¬`over(B,depositBelt)`, `empty(depositBelt)` e `finished(B)`. Esta ação representa a remoção do componente `B` da célula de produção através da Esteira Depósito;

- A ação `move(B,D1,D2)` tendo como pré-condições `over(B,D1)`, `empty(D2)`, `bloc(B)`, `device(D1)`, `device(D2)` e como efeitos `over(B, D2)`, ¬`over(B,D1)`, ¬`empty(D2)` e `empty(D1)`. Esta ação representa a movimentação do Componente `B` do Dispositivo `D1` para o Dispositivo `D2`.

Os requisitos de processamento dos componentes e suas prioridades são modelados através dos desejos do agente. Desta forma, podemos modelar a necessidade do agente `pCell` de processar o Componente `bloc1` pelas Unidades de Processamento `procUnit1`, `procUnit2` e `procUnit3` assim que este bloco é inserido na célula de produção através dos seguintes desejos, onde `Tf` é o tempo de validade do desejo, e o valor especificado a seguir é a prioridade do desejo:

```
des(pCell,finished(bloc1),Tf,[0.7])
  if bel(pCell, bloc(bloc1)),
     bel(pCell, processed(bloc1,procUnit1)),
     bel(pCell, processed(bloc1,procUnit2)),
     bel(pCell, processed(bloc1,procUnit3)).

des(pCell,processed(bloc1,procUnit1),Tf,[0.6])
   if bel(pCell, bloc(bloc1)).

des(pCell,processed(bloc1,procUnit2),Tf,[0.6])
   if bel(pCell, bloc(bloc1)).

des(pCell,processed(bloc1,procUnit3),Tf,[0.6])
   if bel(pCell, bloc(bloc1)).
```

Da mesma forma, podemos modelar a necessidade do agente de processar o Componente `bloc2` pelas Unidades de Processamento `procUnit3` e `procUnit4` através dos seguintes desejos:

```
des(pCell,finished(bloc2),Tf,[0.6])
   if bel(pCell, bloc(bloc2)),
     bel(pCell, processed(bloc2,procUnit3)),
     bel(pCell, processed(bloc2,procUnit4)).

des(pCell,processed(bloc2,procUnit3),Tf,[0.5])
   if bel(pCell, bloc(bloc2)).

des(pCell,processed(bloc2,procUnit4),Tf,[0.5])
   if bel(pCell, bloc(bloc2)).
```

Finalmente modela-se o conhecimento constante do agente em relação ao domínio do problema, em especial a classe dos objetos e o estado inicial do mundo com as seguintes crenças:

```
bel(pCell, procUnit(procUnit1)).
bel(pCell, procUnit(procUnit2)).
bel(pCell, procUnit(procUnit3)).
bel(pCell, procUnit(procUnit4)).
bel(pCell, device(procUnit1)).
bel(pCell, device(procUnit2)).
bel(pCell, device(procUnit3)).
bel(pCell, device(procUnit4)).
bel(pCell, device(depositBelt)).
bel(pCell, device(feedBelt)).
bel(pCell, empty(procUnit1)).
bel(pCell, empty(procUnit2)).
bel(pCell, empty(procUnit3)).
bel(pCell, empty(procUnit4)).
bel(pCell, empty(depositBelt)).
```

A chegada de um novo componente na célula de produção é sinalizada pelos sensores através da inclusão na base de dados do agente das crenças `bloc(bloc1)` e `over(bloc1,feedBelt)`, ativando o processo de re-consideração do agente. Dadas as pré-condições dos desejos descritos anteriormente, apenas os desejos relacionados às seguintes propriedades se tornam Elegíveis:

- `processed(bloc1,procUnit1)`.
- `processed(bloc1,procUnit2)`.
- `processed(bloc1,procUnit3)`.

Cabe ressaltar que o desejo relacionado à propriedade `finished(bloc1)` não se torna elegível pois suas pré-condições ainda não são verdadeiras no momento da deliberação. Os desejos elegíveis são então analisados pelo processo de seleção dos Desejos Candidatos. Neste processo, os Desejos Elegíveis e as crenças do agente são utilizados para a criação de problemas de planejamento e enviados para resolução com o Graphplan. O resultado do processamento do Graphplan é um plano que satisfaz todos os Desejos Elegíveis, com os seguintes passos:

1. `move(bloc1,feedBelt,procUnit2)`.
2. `process(bloc1,procUnit2)`.
3. `move(bloc1,procUnit2,procUnit1)`.
4. `process(bloc1,procUnit1)`.

5. `move(bloc1,procUnit1,procUnit3).`

6. `process(bloc1,procUnit3).`

A existência deste plano indica para o X-BDI que o conjunto especificado de Desejos Elegíveis é possível, logo tornando estes desejos Candidatos, que se tornarão Intenções Primárias, representando o comprometimento do agente. A seguir, as Intenções Relativas são geradas com base no plano gerado, uma para cada item, que conduzirão o agente a realizar as ações respectivas. Uma vez executadas estas ações, os Desejos Candidatos resultantes da deliberação anterior, *i.e.* `processed(bloc1,procUnit1)`, `processed(bloc1,procUnit2)` e `processed(bloc1,procUnit3)`, são satisfeitos. Além disto, a pré-condição do desejo de realizar `finished(bloc1)` se torna verdadeira, ativando novamente o processo deliberativo do agente e gerando o seguinte plano:

1. `move(bloc1,procUnit3,depositBelt).`

2. `consume(bloc1).`

Novamente, este plano irá dar origem às intenções do agente e, eventualmente, levá-lo a agir. Uma situação possível durante o funcionamento do agente seria a entrada de um novo componente na Célula de Produção, isto poderia ocorrer logo após a deliberação do primeiro plano, sinalizada pelos sensores do agente através da inclusão das crenças `bloc(bloc2)` e `over(bloc2,feedBelt)` na base de crenças, o que modificaria o conjunto de Desejos Elegíveis escolhidos no segundo ciclo de deliberação para:

- `finished(bloc1).`
- `processed(bloc2,procUnit3).`
- `processed(bloc2,procUnit4).`

Estes desejos se tornam Desejos Candidatos pois o Graphplan é capaz de gerar um plano que satisfaz todos os desejos mutuamente, que seria:

1. `move(bloc1,procUnit3,depositBelt).`

2. `move(bloc2,feedBelt,procUnit4).`

3. `consume(bloc1).`

4. `process(bloc2,procUnit4).`

5. `move(bloc2,procUnit4,procUnit3).`

6. `process(bloc2,procUnit3).`

7. `move(bloc2,procUnit3,depositBelt).`

8. `consume(bloc2).`

Assim, os passos deste plano geram Intenções Relativas, eventualmente levando o agente à execução das ações.

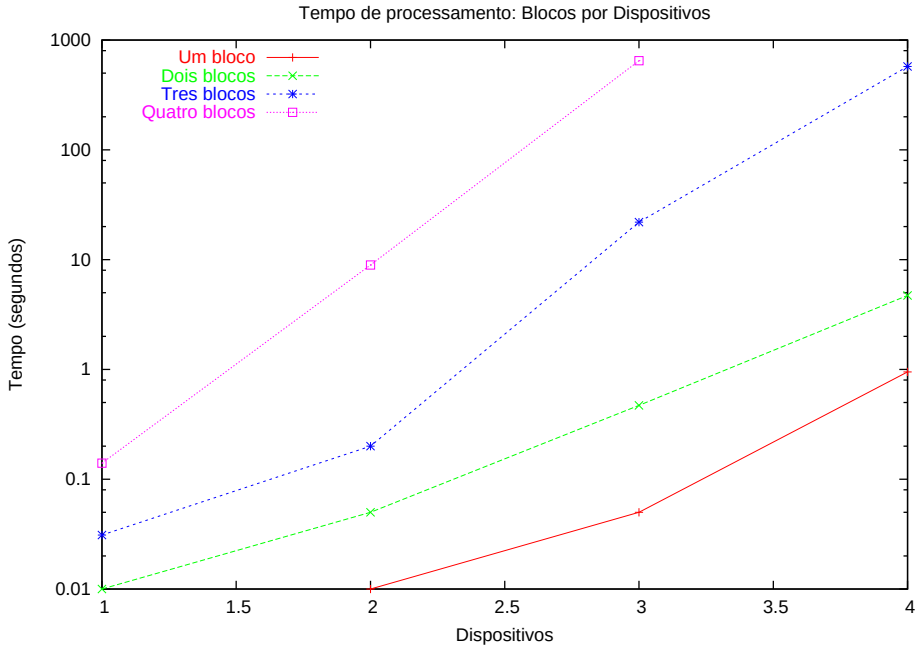

Figura 7.2: Tempos de execução para variações do problema.

Variações do estudo de caso proposto foram utilizadas na avaliação da implementação do algoritmo de planejamento utilizado neste trabalho. Os tempos de execução para diversas combinações de unidades de processamento e blocos a serem processados podem ser observados na Figura 7.2.

7.2 Injeção de Falhas utilizando o *AgentViewer*

Injeção de Falhas é um processo empírico de verificação de sistemas baseado na introdução deliberada de falhas em um sistema [ARL 89]. Este processo de verificação da confiabilidade de sistemas na presença de falhas é considerado uma ferramenta importante no desenvolvimento de

sistemas tolerantes a falhas [AID 01, ARL 02]. Um processo de verificação por injeção de falhas consiste na introdução de estados errôneos no ambiente onde um sistema trabalha e/ou nos dados disponíveis para o sistema [ARL 89].

Considerando a representação de mundo na ferramenta *AgentViewer*, é possível modificar arbitrariamente o estado atual do mundo, ou mesmo enviar uma descrição errada do estado atual do mundo para o agente. Desta forma, é possível inserir falhas tanto no estado atual do mundo quanto na percepção do mundo pelo agente, permitindo que se verifique a capacidade do agente de agir corretamente na presença de falhas.

A fim de verificar a capacidade de funcionamento na presença de falhas de um agente X-BDI, modelou-se uma célula de produção onde é prevista a ocorrência de falhas nos seus componentes. A célula de produção proposta é composta por uma Esteira Depósito, duas Prensas para o processamento de Placas de metal, um Robô com dois Braços perpendiculares utilizados para a movimentação dos componentes na célula e uma Mesa onde os componentes que entram na célula são depositados (Figura 7.3). Qualquer um dos componentes da célula é passível de falha. Estes componentes são modelados com os seguintes predicados:

- `failed(X)` denota que o componente X falhou;
- `empty(X)` denota que o componente X está vazio;
- `plate(P)` denota que P é uma placa de metal a ser processada pela célula;
- `robot(R)` denota que R é um Robô na célula;
- `arm(A,R)` denota que A é um braço do robô R;
- `table(T)` denota que T é uma mesa de alimentação;
- `press(P)` denota que P é uma prensa de placas de metal;
- `depositBelt(D)` denota que D é uma esteira depósito;
- `loaded(X,P)` denota que o componente X está carregado com a placa P;
- `done(P)` denota que a placa P está pronta.

As ações permitidas neste domínio são as seguintes:

- A ação `unloadTable(R, A, T, P)`, com pré-condições `plate(P)`, `robot(R)`, `arm(A,R)`, `empty(A)`, `table(T)`, `loaded(T, P)`, ¬`failed(A)`, ¬`failed(R)` e ¬`failed(T)` e efeitos `empty(T)`, ¬`loaded(T, P)`, ¬`empty(A)` e `loaded(A, P)`, representa a descarga da placa P da mesa T pelo braço A do robô R;

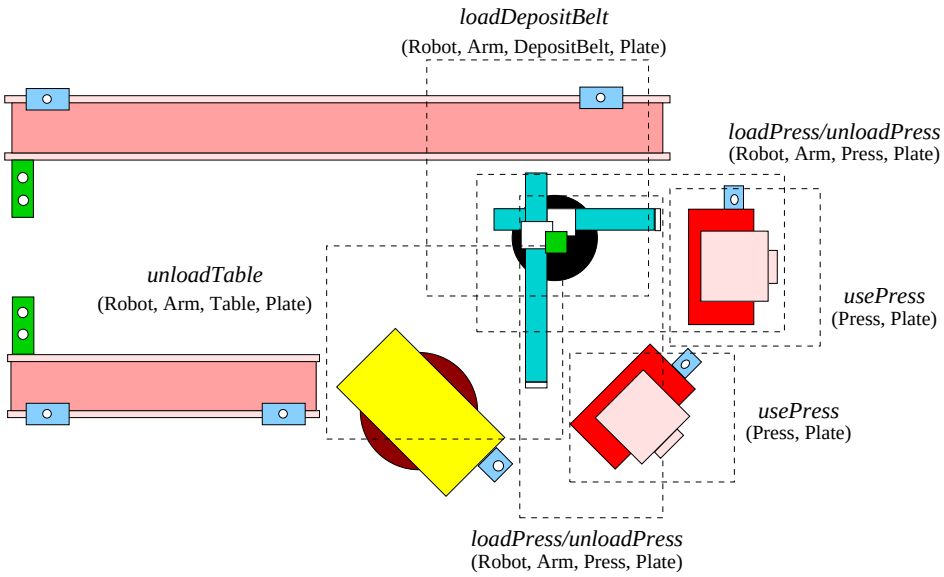

Figura 7.3: Célula de produção.

- A ação `loadPress(R, A, Pr, P)`, com pré-condições `plate(P)`, `robot(R)`, `arm(A, R)`, `loaded(A, P)`, `press(Pr)`, `empty(Pr)`, ¬`failed(A)`, ¬`failed(R)` e ¬`failed(Pr)`, e efeitos ¬`loaded(A, P)`, `empty(A)`, `loaded(Pr, P)`, ¬`empty(Pr)`, representa a carga da placa P na prensa Pr pelo braço A do robô R;

- A ação `unloadPress(R, A, Pr, P)`, com pré-condições `plate(P)`, `robot(R)`, `arm(A, R)`, `empty(A)`, `press(Pr)`, `loaded(Pr, P)`, ¬`failed(A)`, ¬`failed(R)` e ¬`failed(Pr)`, e efeitos `loaded(A, P)`, ¬`empty(A)`, ¬`loaded(Pr, P)` e `empty(P)`, representa a descarga da placa P da prensa Pr pelo braço A do robô R;

- A ação `usePress(Pr, P)`, com pré-condições `plate(P)`, `press(Pr)`, `loaded(Pr, P)` e ¬`failed(Pr)`, e efeito `done(P)`, representa a utilização da prensa Pr sobre a placa P;

- A ação `loadDepositBelt(R, A, D, P)`, com pré-condições `plate(P)`, `robot(R)`, `arm(A, R)`, `loaded(A, P)`, `depositBelt(D)`, `empty(D)`, ¬`failed(A)`, ¬`failed(R)` e ¬`failed(D)`, e efeitos ¬`loaded(A, P)`, `empty(A)`, `loaded(D, P)` e ¬`empty(D)`, representa a carga da placa P na esteira depósito D pelo braço A do robô R.

Os objetivos do agente neste contexto são bastante simples, a finalização de uma placa P qualquer assim que ela entrar na célula e a sua carga na esteira depósito uma vez pronta. Isto é modelado com os seguintes desejos:

```
des(fzi, done(P),Tf,[0.8])
   if bel(fzi, plate(P)).

des(fzi, loaded(depositBelt, P),Tf,[0.9])
   if bel(fzi, done(P)).
```

O conhecimento do agente sobre este domínio é modelado com as seguintes crenças iniciais:

```
bel(fzi, table(table)).
bel(fzi, robot(robot)).
bel(fzi, arm(arm1,robot)).
bel(fzi, empty(arm1)).
bel(fzi, arm(arm2,robot)).
bel(fzi, empty(arm2)).
bel(fzi, press(press1)).
bel(fzi, empty(press1)).
bel(fzi, press(press2)).
bel(fzi, empty(press2)).
bel(fzi, depositBelt(depositBelt)).
bel(fzi, empty(depositBelt)).
bel(fzi, ¬failed(table)).
bel(fzi, ¬failed(robot)).
bel(fzi, ¬failed(arm1)).
bel(fzi, ¬failed(arm2)).
bel(fzi, ¬failed(press1)).
bel(fzi, ¬failed(press2)).
bel(fzi, ¬failed(depositBelt)).
```

A deposição de uma nova placa de metal na Mesa de chegada na célula é representada pela inclusão das propriedades `plate(plate1)` e `loaded(table, plate1)` na base de crenças do agente. Esta modificação irá desencadear o início do processo deliberativo do agente. Neste processo, é verificado que a pré-condição do desejo de concretizar `done(P)` foi satisfeita, tornando este desejo elegível. Para satisfazer este desejo, o algoritmo de planejamento do agente gera o seguinte plano:

1. `unloadTable(robot,arm1,table,plate1).`
2. `loadPress(robot,arm1,press1,plate1).`
3. `usePress(press1,plate1).`

Este plano comprova, portanto, a possibilidade de satisfazer o desejo elegível escolhido anteriormente, logo, este desejo se torna candidato. Este desejo candidato irá então dar origem as intenções primárias e relativas, levando o agente a executar as ações especificadas. Durante o processo de execução das ações, é possível que ocorra uma falha em um dos componentes, por exemplo, a prensa número um (`press1`), denotado através da crença na propriedade `failed(press1)`. Neste caso, a ação `loadPress(robot,arm1,press1,plate1)` irá se tornar impossível pois uma de suas pré-condições agora é falsa. Ao perceber esta falha, o agente irá ter que re-planejar o seu curso de ação. Supondo que o agente já tenha executado a ação `unloadTable(robot,arm1,table,plate1)`, o novo plano gerado pelo agente é:

1. `loadPress(robot,arm1,press2,plate1).`
2. `usePress(press2,plate1).`

Como o agente foi capaz de gerar um novo plano para satisfazer o desejo inicial, este desejo permanece como candidato, modificando apenas as intenções relativas do agente para refletir o comprometimento com um curso de ação diferente. Neste novo curso de ação o agente irá utilizar a prensa de número dois (`press2`), ao invés da de número um, para processar a placa de metal. Se nesta mesma situação o componente falho for o braço de número um (`arm1`), denotado pela crença na propriedade `failed(arm1)` o cumprimento do desejo de finalizar a placa de metal se torna impossível, visto que a placa estará presa a um braço defeituoso.

Diversas outras combinações de falhas em diversos dos componentes da célula foram testadas, onde pode ser verificada a capacidade do agente de verificar se ações de correção eram possíveis ou se a falha impedia o agente de atingir seus objetivos.

Capítulo 8

Considerações Finais

Neste trabalho foi descrita a relação entre algoritmos de planejamento proposicional e o raciocínio meios-fim de agentes BDI. Para testar a viabilidade desta abordagem foram realizadas modificações no modelo X-BDI de agentes a fim de que este pudesse utilizar um módulo de planejamento externo dotado de uma implementação do Graphplan. Durante este processo de modificação, foram criadas novas definições de desejos e intenções de modo a manter as propriedades teóricas presentes na sua versão original, em particular relativas à definição de impossibilidade de desejos e intenções. Além disto, foi necessário definir um mapeamento entre os componentes estruturais de um agente BDI e problemas de planejamento proposicional, e de planos proposicionais de volta para os estados mentais do agente. O resultado destas modificações foi implementado em um protótipo descrito no Capítulo 6, que reúne tanto uma versão modificada do X-BDI quanto uma implementação do Graphplan, além de prover uma ferramenta de operação e teste dos agentes criados com este sistema.

Através da junção de agentes BDI e algoritmos de planejamento rápidos, espera-se que a classe de problemas cuja resolução em tempo de execução é viável possa ser expandida na direção dos algoritmos de planejamento proposicionais utilizados em conjunto com o agente, sejam estes quais forem. Os resultados da aplicação prática da implementação desenvolvida foram observados em uma série de estudos de caso descritos no Capítulo 7, sendo que o problema da Seção 7.1 foi executado pelo X-BDI original ininterruptamente por 48 horas sem produzir resultados, enquanto que a versão atual do mesmo levou menos de um segundo.

Considerando o fato de que a maioria das implementações de agentes BDI de que se tem conhecimento utilizam uma biblioteca de planos no processo de raciocínio meios-fim a fim de evitar a complexidade inerente à realização de planejamento em tempo de execução, o X-BDI oferece uma maneira inovadora de implementar agentes mais flexíveis. Considerando ainda o fato de que o X-BDI original era limitado pelo seu processo de planejamento ineficiente, uma importante contribuição deste trabalho reside na definição de um mecanismo de mapeamento entre o raciocínio

meios-fim de agentes BDI e algoritmos rápidos de planejamento, como o Graphplan [MEN 04].

Além dos resultados na área de IA obtidos por este trabalho, a ferramenta construída se mostrou útil para o processo de desenvolvimento de sistemas tolerantes a falhas baseados em agentes [MEN 03]. Esta contribuição é representada pela aplicabilidade do *AgentViewer* como ferramenta de injeção de falhas para sistemas modelados em termos de agentes BDI [ZOR 04].

8.1 Trabalhos Futuros

Algumas ramificações deste trabalho podem ser visualizadas como trabalhos futuros, em especial, a incorporação dos diversos aprimoramentos no Graphplan, além da realização de testes com outros algoritmos de planejamento proposicionais, como por exemplo os algoritmos baseados em SAT. Além disto, a determinação da classe de problemas os quais as combinações de Agente BDI e demais algoritmos de planejamento são capazes de tratar representa uma extensão teórica muito interessante a este trabalho.

Na área de tolerância a falhas, os estudos de caso desenvolvidos permitem intuir que o *Agent Design Problem* está relacionado à questão de *Dependability Testing*, que consiste em verificar o grau de confiabilidade de um sistema frente a falhas. Este relacionamento é justificado na medida em que a prova da capacidade ou não de um agente em realizar uma tarefa em um ambiente onde falhas são possíveis é um indicador bastante preciso da confiabilidade do agente.

Referências Bibliográficas

[AID 01] AIDEMARK, J. et al. GOOFI: Generic object-oriented fault injection tool. In: PROCEEDINGS OF THE 2001 INTERNATIONAL CONFERENCE ON DEPENDABLE SYSTEMS AND NETWORKS (DSN 2001), 2001. **Proceedings...** Göteborg, Sweden: IEEE Computer Society, 2001. p.83–88.

[ALF 95] ALFERES, J. J.; DAMASIO, C. V.; PEREIRA, L. M. A logic programming system for nonmonotonic reasoning. **Journal of Automated Reasoning**, The Netherlands: Kluwer Academic Publishers, v.14, n.1, p.93–147, 1995.

[ALF 96] ALFERES, J. J.; PEREIRA, L. M. **Reasoning with Logic Programming**. Germany: Springer-Verlag, 1996.

[ARL 89] ARLAT, J.; CROUZET, Y.; LAPRIE, J.-C. Fault injection for dependability validation of fault-tolerant computing systems. In: DIGEST OF PAPERS 19TH INTERNATIONAL SYMPOSIUM ON FAULT-TOLERANT COMPUTING (FTCS-19), 1989. **Proceedings...** Chicago, IL: IEEE Computer Society, 1989. p.348–355.

[ARL 02] ARLAT, J. From experimental assessment of fault-tolerant systems to dependability benchmarking. In: PROCEEDINGS OF THE INTERNATIONAL PARALLEL AND DISTRIBUTED PROCESSING SYMPOSIUM 2002 (IPDPS 2002), 2002. **Proceedings...** Fort Lauderdale, USA: IEEE Computer Society, 2002. p.135–136.

[ASH 02] ASHRI, R.; LUCK, M.; D'INVERNO, M. Infrastructure support for agent-based development. In: M. D'Inverno, M. Luck, M. F.; (Eds.), C. P., editors, FOUNDATIONS AND APPLICATIONS OF MULTI-AGENT SYSTEMS, v.2403 of **Lecture Notes in Artificial Intelligence**, p.73–88. Springer-Verlag, Germany, 2002.

[BLU 97] BLUM, A. L.; FURST, M. L. Fast planning through planning graph analysis. **Artificial Intelligence**, The Netherlands: Elsevier Science Publishers, v.90, n.1-2, p.281–300, 1997.

[BOR 02] BORDINI, R. H. et al. AgentSpeak(XL): efficient intention selection in BDI agents via decision-theoretic task scheduling. In: PROCEEDINGS OF THE FIRST INTERNATIONAL JOINT CONFERENCE ON AUTONOMOUS AGENTS AND MULTIAGENT SYSTEMS, 2002. **Proceedings...** Bologna, Italy: ACM Press, 2002. p.1294–1302.

[BOR 03] BORDINI, R. H. et al. Model checking AgentSpeak. In: PROCEEDINGS OF THE 2ND INTERNATIONAL CONFERENCE ON AUTONOMOUS AGENTS AND MULTIAGENT SYSTEMS (AAMAS-03), 2003. **Proceedings...** Melbourne, Australia: ACM Press, 2003. p.409–416.

[BRA 84] BRATMAN, M. E. Two faces of intention. **Philosophical Review**, New York, USA: Cornell University, v.93, p.375–405, 1984.

[BRA 87] BRATMAN, M. E. **Intention, Plans and Practical Reason**. Cambridge, USA: Harvard University Press, 1987.

[BRA 88] BRATMAN, M. E.; ISRAEL, D. J.; POLLACK, M. E. Plans and resource-bounded practical reasoning. **Computational Intelligence**, UK: Blackwell Publishers, v.4, n.4, p.349–355, 1988.

[BRA 90] BRATMAN, M. E. What is intention? **Intentions in Communication**, Cambridge, USA: MIT Press, 1990.

[BRA 99a] BRATMAN, M. E. **Faces of Intention: Selected Essays on Intention and Agency**. Cambridge, USA: Cambridge University Press, 1999.

[BRA 99b] BRATMAN, M. E. **Intentions, Plans and Practical Reason**. Stanford, CA: CLSI Publications, 1999.

[BUS 00] BUSSMANN, S.; JENNINGS, N. R.; WOOLDRIDGE, M. On the identification of agents in the design of production control systems. In: Ciancarini, P.; Wooldridge, M., editors, PROCEEDINGS OF THE FIRST INTERNATIONAL WORKSHOP ON AGENT-ORIENTED SOFTWARE ENGINEERING (AOSE), v.1957 of **Lecture Notes in Computer Science**, p.141–162. Springer-Verlag, Limerick, Ireland, 2000.

[BYL 94] BYLANDER, T. The computational complexity of propositional STRIPS planning. **Artificial Intelligence**, The Netherlands: Elsevier Science Publishers, v.69, n.1-2, p.165–204, 1994.

[CHA 87] CHAPMAN, D. Planning for conjunctive goals. **Artificial Intelligence**, The Netherlands: Elsevier Science Publishers, v.32, n.3, p.333–377, 1987.

[CM 98] CAMARINHA-MATOS, L. M.; AFSARMANESH, H.; MARÍK, V. **Intelligent Systems for Manufacturing: Multi-Agent Systems and Virtual Organizations**, v.130 of **IFIP International Federation for Information Processing**. The Netherlands: Kluwer Academic Publishers, 1998.

[COH 89] COHEN, P. R. et al. Trial by fire: Understanding the design requirements for agents in complex environments. **AI Magazine**, USA: AAAI Press, v.10, n.3, p.32–48, 1989.

[COH 90] COHEN, P. R.; LEVESQUE, H. J. Intention is choice with commitment. **Artificial Intelligence**, The Netherlands: Elsevier Science Publishers, v.42, n.2-3, p.213–261, 1990.

[COO 97] COOK, S. A.; MITCHEL, D. G. Finding hard instances of the satisfiability problem: A survey. In: Du, D.; Gu, J.; Pardalos, P. M., editors, SATISFIABILITY PROBLEM: THEORY AND APPLICATIONS, v.35 of **DIMACS Series in Discrete Mathematics and Theoretical Computer Science**, p.11–13. American Mathematical Society, Providence, RI, 1997.

[DAV 63] DAVIDSON, D. **Actions, Reasons, and Causes**, chapter1, p.3–20. Oxford University Press, UK, 1963.

[DAV 88] DAVIS, M. Influences of mathematical logic on computer science. In: A HALF-CENTURY SURVEY ON THE UNIVERSAL TURING MACHINE, 1988. **Proceedings...** Berlin, Germany: Oxford University Press, 1988. p.315–326.

[DAV 02] DAVEY, B. A.; PRIESTLEY, H. A. **Introduction to Lattices and Order**. 2nd Edition. ed. Cambridge, USA: Cambridge University Press, 2002.

[DEN 78] DENNETT, D. C. **Brainstorms: Philosophical Essays on Mind and Psychology**. Montgomery, USA: Bradford Books and Hassocks, 1978.

[DEN 87] DENNETT, D. C. **The Intentional Stance**. Cambridge, USA: MIT Press / Bradford Books, 1987.

[D'I 98a] D'INVERNO, M. et al. A formal specification of dMARS. In: Singh, M. P.; Rao, A. S.; Wooldridge, M., editors, AGENT THEORIES, ARCHITECTURES, AND LANGUAGES, v.1365 of **Lecture Notes in Computer Science**, p.155–176. Springer-Verlag, Germany, 1998.

[D'I 98b] D'INVERNO, M.; LUCK, M. Engineering AgentSpeak(L): A formal computational model. **Journal of Logic and Computation**, UK: Oxford University Press, v.8, n.3, p.233–260, 1998.

[EME 90] EMERSON, E. A. **Temporal and Modal Logic**, v.B, chapter16, p.996–1072. Elsevier Science Publishers, The Netherlands, 1990. Handbook of Theoretical Computer Science.

[ET 01] EXCELENTE-TOLEDO, C. B.; BOURNE, R. A.; JENNINGS, N. R. Reasoning about commitments and penalties for coordination between autonomous agents. In: PROCEEDINGS OF THE 5TH INTERNATIONAL CONFERENCE ON AUTONOMOUS AGENTS, 2001. **Proceedings...** Montreal, Canada: ACM Press, 2001. p.131–138.

[FAT 01] FATIMA, S. S.; WOOLDRIDGE, M. Adaptive task resources allocation in multi-agent systems. In: PROCEEDINGS OF THE 5TH INTERNATIONAL CONFERENCE ON AUTONOMOUS AGENTS, 2001. **Proceedings...** Montreal, Canada: ACM Press, 2001. p.537–544.

[FIK 71] FIKES, R.; NILSSON, N. STRIPS: A new approach to the application of theorem proving to problem solving. **Artificial Intelligence**, The Netherlands: Elsevier Science Publishers, v.2, n.3-4, p.189–208, 1971.

[GEL 88] GELFOND, M.; LIFSCHITZ, V. The stable model semantics for logic programming. In: Kowalski, R. A.; Bowen, K. A., editors, PROCEEDINGS OF THE 5TH INTERNATIONAL CONFERENCE ON LOGIC PROGRAMMING, 1988. **Proceedings...** Cambridge, USA: MIT Press, 1988. p.1070–1080.

[GEL 91] GELDER, A. V.; ROSS, K.; SCHLIPF, J. S. The well-founded semantics for general logic programs. **Journal of the ACM**, New York, USA: ACM Press, v.38, n.3, p.620–650, 1991.

[GEO 86] GEORGEFF, M. P.; LANSKY, A. L. Procedural knowledge. **Proceedings of the IEEE, Special Issue on Knowledge Representation**, USA: IEEE Computer Society, v.74, n.10, p.1383–1898, 1986.

[GEO 87] GEORGEFF, M. P.; LANSKY, A. L. Reactive reasoning and planning. In: PROCEEDINGS OF THE AMERICAN ASSOCIATION FOR ARTIFICIAL INTELLIGENCE (AAAI), 1987. **Proceedings...** Seattle, WA: Morgan Kaufmann Publishers, 1987. p.677–682.

[GEO 89a] GEORGEFF, M. P.; INGRAND, F. F. Decision-making in an embedded reasoning system. In: PROCEEDINGS OF THE 11TH INTERNATIONAL JOINT CONFERENCE ON ARTIFICIAL INTELLIGENCE (IJCAI'89), 1989. **Proceedings...** Detroit, MI: Morgan Kaufmann Publishers, 1989. p.972–978.

[GEO 89b] GEORGEFF, M. P.; INGRAND, F. F. Monitoring and control of spacecraft systems using procedural reasoning. In: PROCEEDINGS OF THE SPACE OPERATIONS AND ROBOTICS WORKSHOP, 1989. **Proceedings...** Houston, TX: [s.n.], 1989. p.n/a.

[GEO 99] GEORGEFF, M. et al. The belief-desire-intention model of agency. In: Müller, J.; Singh, M. P.; Rao, A. S., editors, PROCEEDINGS OF THE 5TH INTERNATIONAL WORKSHOP ON INTELLIGENT AGENTS V : AGENT THEORIES, ARCHITECTURES, AND LANGUAGES (ATAL-98), 1999. **Proceedings...** Germany: Springer-Verlag, 1999. v.1555, p.1–10.

[GER 02] GEREVINI, A.; SERINA, I. LPG: A planner based on local search for planning graphs with action costs. In: Ghallab, M.; Hertzberg, J.; Traverso, P., editors, PROCEEDINGS OF THE 6TH INTERNATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE PLANNING SYSTEMS, 2002. **Proceedings...** Toulouse, France: AAAI Press, 2002. p.13–22.

[GHA 02] GHALLAB, M.; HERTZBERG, J.; TRAVERSO, P. **Proceedings of the Sixth International Conference on Artificial Intelligence Planning Systems**. Toulouse, France: AAAI Press, April 23-27, 2002.

[HAL 92] HALPERN, J. Y.; MOSES, Y. A guide to completeness and complexity for modal logics of knowledge and belief. **Artificial Intelligence**, The Netherlands: Elsevier Science Publishers, v.54, n.2, p.319–379, 1992.

[HOE 03] HOEK, W. V. D.; WOOLDRIDGE, M. Towards a logic of rational agency. **Logic Journal of the IGPL**, UK: Oxford University Press, v.11, n.2, p.133–157, 2003.

[HOF 99] HOFFMANN, J.; KÖHLER, J. A new method to index and query sets. In: PROCEEDINGS OF THE 16TH INTERNATIONAL JOINT CONFERENCE ON ARTIFICIAL INTELLIGENCE, 1999. **Proceedings...** San Francisco, CA: Morgan Kaufmann Publishers, 1999. p.462–467.

[HOF 01] HOFFMANN, J.; NEBEL, B. The FF planning system: Fast plan generation through heuristic search. **Journal of Artificial Intelligence Research (JAIR)**, USA: AAAI Press, v.14, p.253–302, 2001.

[ING 92] INGRAND, F. F.; GEORGEFF, M. P.; RAO, A. S. An architecture for real-time reasoning and system control. **IEEE Expert, Knowledge-Based Diagnosis in Process Engineering**, USA: IEEE Computer Society, v.7, n.6, p.33–44, 1992.

[ING 96] INGRAND, F. F. et al. PRS: A high level supervision and control language for autonomous mobile robots. In: PROCEEDINGS OF THE IEEE INTERNATIONAL CONFERENCE ON ROBOTICS AND AUTOMATION, 1996. **Proceedings...** Minneapolis, USA: IEEE Computer Society, 1996. p.43–49.

[ING 01] INGRAND, F. F.; COUTANCE, V. Real-time reasoning using procedural reasoning. LAAS/CNRS, France: LAAS/CNRS, January, 2001. Technical Report93104.

[IWE 02] IWEN, M.; MALI, A. D. Distributed graphplan. In: PROCEEDINGDS OF THE 14TH IEEE INTERNATIONAL CONFERENCE ON TOOLS WITH ARTIFICIAL INTELLIGENCE (ICTAI'02), 2002. **Proceedings...** Washington, DC: IEEE Computer Society, 2002. p.138–145.

[JEN 99] JENNINGS, N. R. Agent-Oriented Software Engineering. In: Garijo, F. J.; Boman, M., editors, PROCEEDINGS OF THE 9TH EUROPEAN WORKSHOP ON MODELLING AUTONOMOUS AGENTS IN A MULTI-AGENT WORLD: MULTI-AGENT SYSTEM ENGINEERING (MAAMAW-99), 1999. **Proceedings...** Germany: Springer-Verlag, 1999. v.1647, p.1–7.

[JEN 00] JENNINGS, N. R. On agent-based software engineering. **Artificial Intelligence**, The Netherlands: Elsevier Science Publishers, v.117, n.2, p.277–296, 2000.

[JON 77] JONES, J. M. **Introduction to Decision Theory**. Homewood, IL: Richard D Irwin, 1977.

[KAU 92] KAUTZ, H.; SELMAN, B. Planning as satisfiability. In: PROCEEDINGS OF THE TENTH EUROPEAN CONFERENCE ON ARTIFICIAL INTELLIGENCE, 1992. **Proceedings...** Chichester, UK: Wiley Publishers, 1992. p.359–363.

[KAU 96] KAUTZ, H.; SELMAN, B. Pushing the envelope: Planning, propositional logic and stochastic search. In: PROCEEDINGS OF THE THIRTEENTH NATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE AND THE EIGHTH INNOVATIVE APPLICATIONS OF ARTIFICIAL INTELLIGENCE CONFERENCE, 1996. **Proceedings...** Menlo Park, USA: AAAI Press / MIT Press, 1996. p.1194–1201.

[KIN 90] KINNY, D. N. Measuring the effectiveness of situated agents. Australian Artificial Intelligence Institute, Australia: Australian Artificial Intelligence Institute, November, 1990. Technical Report11.

[KIN 91] KINNY, D. N.; GEORGEFF, M. P. Commitment and effectiveness of situated agents. In: PROCEEDINGS OF THE 12TH INTERNATIONAL JOINT CONFERENCE ON ARTIFICIAL INTELLIGENCE (IJCAI), 1991. **Proceedings...** Sydney, Australia: Morgan Kaufmann Publishers, 1991. p.82–88.

[KIS 92] KISS, G.; REICHGELT, H. Towards a semantics of desires. In: Werner, E.; Demazeau, Y., editors, DECENTRALIZED AI 3 — PROCEEDINGS OF THE 3RD EUROPEAN WORKSHOP ON MODELLING AUTONOMOUS AGENTS IN A MULTI-AGENT WORLD (MAAMAW-91), 1992. **Proceedings...** The Netherlands: Elsevier Science Publishers, 1992. p.115–128.

[KNU 02] KNUBLAUCH, H. Extreme programming of multi-agent systems. In: PROCEEDINGS OF THE FIRST INTERNATIONAL JOINT CONFERENCE ON AUTONOMOUS AGENTS AND MULTIAGENT SYSTEMS, 2002. **Proceedings...** Bologna, Italy: ACM Press, 2002. p.704–711.

[KÖH 97] KÖHLER, J. et al. Extending planning graphs to an ADL subset. In: Steel, S., editor, PROCEEDINGS OF THE 4TH EUROPEAN CONFERENCE ON PLANNING, v.1348 of **Lecture Notes in Computer Science**, p.273–285. Springer-Verlag, Germany, 1997.

[KÖH 98] KÖHLER, J. Solving complex planning tasks through extraction of subproblems. In: Simmons, R.; Veloso, M.; Smith, S., editors, PROCEEDINGS OF THE FOURTH INTERNATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE PLANNING SYSTEMS, 1998. **Proceedings...** Pittsburg, USA: AAAI Press, 1998. p.62–69.

[KOW 86] KOWALSKI, R. A.; SERGOT, M. J. A logic-based calculus of events. **New Generation Computing**, Japan: Omsha Ltd., v.4, n.1, p.67–95, 1986.

[LEW 95] LEWERENTZ, C.; LINDNER, T. Formal development of reactive system: Case study production cell. In: FORMAL DEVELOPMENT OF REACTIVE SYSTEM: CASE STUDY PRODUCTION CELL, v.891 of **Lecture Notes in Computer Science**. Springer-Verlag, Germany, 1995.

[LON 99] LONG, D.; FOX, M. Efficient implementation of the plan graph in STAN. **Journal of Artificial Intelligence Research (JAIR)**, USA: AAAI Press, v.10, p.87–115, 1999.

[LON 00] LONG, D.; FOX, M. Automatic synthesis and use of generic types in planning. In: Chien, S.; Kambhampati, S.; Knoblock, C. A., editors, PROCEEDINGS OF THE FIFTH INTERNATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE PLANNING SYSTEMS, 2000. **Proceedings...** Breckenridge, Colorado: AAAI Press, 2000. p.196–205.

[MCC 79] MCCARTHY, J. Ascribing mental qualities to machines. Stanford University AI Lab: Stanford University AI Lab, 1979. Technical Report326.

[MEN 03] MENEGUZZI, F. R. Modelling fault-tolerant systems using BDI agents. In: Brasileiro, F.; Zorzo, A. F., editors, PROCEEDINGS OF THE 2ND WORKSHOP ON THESES AND DISSERTATIONS ON DEPENDABLE COMPUTING, 2003. **Proceedings...** São Paulo, Brazil: EPUSP, 2003. p.19–24.

[MEN 04] MENEGUZZI, F. R.; ZORZO, A. F.; MÓRA, M. D. C. Propositional planning in BDI agents. In: PROCEEDINGS OF THE 2004 ACM SYMPOSIUM ON APPLIED COMPUTING, 2004. **Proceedings...** Nicosia, Cyprus: ACM Press, 2004. p.58–63.

[MÓR 99a] MÓRA, M. C. **Um Modelo Formal e Executável de Agentes BDI**. Porto Alegre: CPGCC/UFRGS, 1999. Tese de Doutorado.

[MÓR 99b] MÓRA, M. C. et al. BDI models and systems: Reducing the gap. In: Müller, J. P.; Singh, M. P.; Rao, A. S., editors, PROCEEDINGS OF THE 5TH INTERNATIONAL WORKSHOP ON INTELLIGENT AGENTS V : AGENT THEORIES, ARCHITECTURES, AND LANGUAGES (ATAL-98), v.1555 of **Lecture Notes in Computer Science**. Springer-Verlag, Germany, 1999.

[MÜL 96] MÜLLER, J. P. The design of intelligent agents: A layered approach. In: THE DESIGN OF INTELLIGENT AGENTS: A LAYERED APPROACH, v.1177 of **Lecture Notes in Computer Science**. Springer-Verlag, Germany, 1996.

[NAI 03] NAIR, R.; TAMBE, M.; MARSELLA, S. Integrating belief-desire-intention approaches with POMDPs: The case of team-oriented programs. In: Doherty, P.; McCarthy, J.; Williams, M.-A., editors, LOGICAL FORMALIZATION OF COMMONSENSE REASONING 2003 AAAI SPRING SYMPOSIUM, 2003. **Proceedings...** USA: AAAI Press, 2003. p.107–115.

[Nat 03] National Institute for Standards and Technologies. **Memoization Algorithmic Technique**. Extracted from http://www.nist.gov/dads/HTML/memoize.html in 4/2003.

[NEB 00] NEBEL, B. On the compilability and expressive power of propositional planning formalisms. **Journal of Artificial Intelligence Research (JAIR)**, USA: AAAI Press, v.12, p.271–315, 2000.

[NID 02] NIDE, N.; TAKATA, S. Deduction systems for BDI logics using sequent calculus. In: PROCEEDINGS OF THE FIRST INTERNATIONAL JOINT CONFERENCE ON AUTONOMOUS AGENTS AND MULTIAGENT SYSTEMS, 2002. **Proceedings...** Bologna, Italy: ACM Press, 2002. p.928–935.

[PAP 94] PAPADIMITRIOU, C. H. **Computational Complexity**. Reading, MA: Addison-Wesley, 1994.

[PER 92] PEREIRA, L. M.; ALFERES, J. J. Well founded semantics for logic programs with explicit negation. In: EUROPEAN CONFERENCE ON ARTIFICIAL INTELLIGENCE, 1992. **Proceedings...** Austria: Wiley Publishers, 1992. p.102–106.

[POL 90] POLLACK, M. E.; RINGUETTE, M. Introducing the tileworld: experimentally evaluating agent architectures. In: Dietterich, T.; Swartout, W., editors, PROCEEDINGS OF THE 8TH NATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE, 1990. **Proceedings...** Menlo Park, CA: AAAI Press, 1990. p.183–189.

[POL 94] POLLACK, M. E. et al. Experimental investigation of an agent commitment strategy. Pittsburgh, PA 15260: University of Pittsburgh, 1994. Relatório Técnico94–31.

[POT 96] POTTER, B.; SINCLAIR, J.; TILL, D. **An Introduction to Formal Specification and Z**. 2nd. ed. UK: Prentice Hall, 1996.

[RAO 91a] RAO, A. S.; GEORGEFF, M. P. Deliberation and its role in the formation of intentions. In: PROCEEDINGS OF THE SEVENTH ANNUAL CONFERENCE ON UNCERTAINTY IN ARTIFICIAL INTELLIGENCE (UAI–91), 1991. **Proceedings...** San Mateo, California: Morgan Kaufmann Publishers, 1991. p.300–307.

[RAO 91b] RAO, A. S.; GEORGEFF, M. P. Modeling rational agents within a BDI-architecture. In: Allen, J.; Fikes, R.; Sandewall, E., editors, PROCEEDINGS OF THE 2ND INTERNATIONAL CONFERENCE ON PRINCIPLES OF KNOWLEDGE REPRESENTATION AND REASONING (KR'91), 1991. **Proceedings...** San Mateo, CA, USA: Morgan Kaufmann Publishers, 1991. p.473–484.

[RAO 95a] RAO, A. S.; GEORGEFF, M. P. BDI-agents: from theory to practice. In: PROCEEDINGS OF THE FIRST INTERNATIONAL CONFERENCE ON MULTIAGENT

SYSTEMS ICMAS-95, 1995. **Proceedings...** San Francisco, USA: AAAI Press, 1995. p.312–319.

[RAO 95b] RAO, A. S.; GEORGEFF, M. P. Formal models and decision procedures for multi-agent systems. 171 La Trobe Street, Melbourne, Australia: Australian Artificial Intelligence Institute, 1995. Relatório Técnico61.

[RAO 96] RAO, A. S. AgentSpeak(L): BDI agents speak out in a logical computable language. In: van Hoe, R., editor, 7TH EUROPEAN WORKSHOP ON MODELLING AUTONOMOUS AGENTS IN A MULTI-AGENT WORLD, v.1038 of **Lecture Notes on Computer Science**, p.42–55. Springer-Verlag, Eindhoven, Netherlands, 1996.

[RAO 97] RAO, A. A unified view of plans as recipes. In: Holmstrom-Hintikka, G.; Tuomela, R., editors, CONTEMPORARY ACTION THEORY. Kluwer Academic Publishers, The Netherlands, 1997.

[RUS 94] RUSSEL, S. J.; NORVIG, P. **Artificial Intelligence: A Modern Approach**. New Jersey: Prentice Hall, 1994.

[SCH 01] SCHUT, M.; WOOLDRIDGE, M. The control of reasoning in resource-bounded agents. **The Knowledge Engineering Review**, Cambridge, USA: Cambridge University Press, v.16, n.3, 2001.

[SEA 82] SEARLE, J. R. **What is an Intentional State?**, v.42, p.213–261. MIT Press, Cambridge, USA, 1982.

[SHO 93] SHOHAM, Y. Agent-oriented programming. **Artificial Intelligence**, The Netherlands: Elsevier Science Publishers, v.60, n.1, p.51–92, 1993.

[SHO 94] SHOHAM, Y.; COUSINS, S. B. Logics of mental attitudes in AI. In: Lakemeyer, G.; Nebel, B., editors, FOUNDATIONS OF KNOWLEDGE REPRESENTATION AND REASONING, v.810 of **Lecture Notes in Computer Science**, p.296–309. Springer-Verlag, Germany, 1994.

[SIC 01] SICS, I. S. L. **SICStus Prolog User's Manual**. Intelligent Systems Laboratory (SICS), Kista, Sweden, 2001.

[SMI 98] SMITH, D. E.; WELD, D. S. Conformant graphplan. In: PROCEEDINGS OF THE 10TH ANNUAL CONFERENCE ON INNOVATIVE APPLICATIONS OF ARTIFICIAL INTELLIGENCE, 1998. **Proceedings...** New York, USA: ACM Press, 1998. p.889–896.

[SMI 99] SMITH, D. E.; WELD, D. S. Temporal planning with mutual exclusion reasoning. In: PROCEEDINGS OF THE 17TH INTERNATIONAL JOINT CONFERENCE ON ARTIFICIAL INTELLIGENCE (IJCAI), 1999. **Proceedings...** Stockholm, Sweden: Morgan Kaufmann Publishers, 1999. p.326–337.

[TUR 48] TURING, A. M. Intelligent machinery. **Machine Intelligence**, Edinburgh, UK: Edinburgh University Press, v.5, p.3–23, 1948.

[WEL 99] WELD, D. S. Recent advances in AI planning. **AI Magazine**, USA: AAAI Press, v.20, n.2, p.93–123, 1999.

[WOO 97] WOOLDRIDGE, M. Agent-based software engineering. **The Institution of Electrical Engineers (IEE) Proceedings on Software Engineering**, UK: IEE Press, v.144, n.1, p.26–37, 1997.

[WOO 99] WOOLDRIDGE, M. **Intelligent Agents**, chapter2. MIT Press, Cambridge, USA, 1999.

[WOO 00a] WOOLDRIDGE, M. The computational complexity of agent design problems. In: Durfee, E., editor, PROCEEDINGS OF THE FOURTH INTERNATIONAL CONFERENCE ON MULTI-AGENT SYSTEMS (ICMAS 2000), 2000. **Proceedings...** Boston, USA: IEEE Computer Society, 2000. p.341–348.

[WOO 00b] WOOLDRIDGE, M. **Reasoning about Rational Agents**. Cambridge, USA: MIT Press, 2000.

[WOO 00c] WOOLDRIDGE, M.; CIANCARINI, P. Agent-oriented software engineering: The state of the art. In: Ciancarini, P.; Wooldridge, M., editors, FIRST INTERNATIONAL WORKSHOP ON AGENT-ORIENTED SOFTWARE ENGINEERING (AOSE), v.1957 of **Lecture Notes in Computer Science**, p.1–28. Springer-Verlag, Limerick, Ireland, 2000.

[WOO 01] WOOLDRIDGE, M.; DUNNE, P. E. Optimistic and disjunctive agent design problems. In: Castelfranchi, C.; Lespérance, Y., editors, INTELLIGENT AGENTS VII. AGENT THEORIES ARCHITECTURES AND LANGUAGES, v.1986 of **Lecture Notes in Computer Science**, p.1–14. Springer-Verlag, Boston, 2001.

[ZAM 00] ZAMBONELLI, F.; JENNINGS, N. R.; WOOLDRIDGE, M. Organisational abstractions for the analysis and design of multi-agent systems. In: Ciancarini, P.; Wooldridge, M., editors, AGENT-ORIENTED SOFTWARE ENGINEERING, FIRST

INTERNATIONAL WORKSHOP, AOSE, v.1957 of **Lecture Notes in Computer Science**. Springer-Verlag, Limerick, Ireland, 2000.

[ZOR 04] ZORZO, A. F.; MENEGUZZI, F. R. An agent model of fault-tolerant systems. In: PROCEEDINGS OF THE 2004 INTERNATIONAL CONFERENCE ON DEPENDABLE SYSTEMS AND NETWORKS (DSN 2004), 2004. **Proceedings...** Florence, Italy: IEEE Computer Society, 2004. Submitted.