CENTRO UNIVERSITÁRIO DA FEI
MARCELO UDO

INTERPRETAÇÃO E EQUIVALÊNCIA DE MODELOS UML.P E PDDL PARA PLANEJAMENTO AUTOMÁTICO

São Bernardo do Campo
2009

MARCELO UDO

INTERPRETAÇÃO E EQUIVALÊNCIA DE MODELOS UML.P E PDDL PARA PLANEJAMENTO AUTOMÁTICO

Dissertação de Mestrado apresentada ao Centro Universitário da FEI para obtenção do título de Mestre em Engenharia Elétrica, orientado pelo Professor Dr. Flavio Tonidandel.

São Bernardo do Campo
2009

Udo, Marcelo
Interpretação e equivalência de modelos UML.P e PDDL para Planejamento Automático / Marcelo Udo. São Bernardo do Campo, 2009.
83 f. : il.

Dissertação de Mestrado em Engenharia Elétrica – Centro Universitário da FEI.
Orientador: Prof. Dr. Flavio Tonidandel

1. Planejamento Automático. 2. Engenharia do Conhecimento. 3. Interpretação. 4. PDDL. 5. UML.P. 6. F-Logic. I. Tonidandel, Flavio, orient. II. Título.

CDU 681.3.02

MARCELO UDO

INTERPRETAÇÃO E EQUIVALÊNCIA DE MODELOS UML.P E PDDL PARA PLANEJAMENTO AUTOMÁTICO

Dissertação de Mestrado
Centro Universitário da FEI
Departamento de Engenharia Elétrica

Banca Examinadora:

_____
Prof. Dr. Flavio Tonidandel (Orientador) – FEI

_____
Prof. Dr. Flávio Soares Corrêa da Silva – USP

_____
Prof. Dr. Paulo Eduardo Santos – FEI

São Bernardo do Campo
2009

Ao Espírito Universal e a vocês, que bem-vindas d'Ele, possam me trazer o sonho da realidade como a própria realidade em sonho e eu possa me sentir completo como alguém que passou e fez diferença por aqui.

AGRADECIMENTOS

Agradeço as orientações, correções e sugestões do meu orientador Professor Dr. Flavio Tonidandel – e foram muitas até a conclusão dessa dissertação de Mestrado. Agradeço a oportunidade de estar participando de um projeto de ferramenta de Engenharia do Conhecimento para Planejamento Automático: a Ferramenta itSIMPLE. Agradeço aos professores do Centro Universitário FEI no que tange ao respeito e *background* na área de IA. Agradeço aos colegas da USP. Agradeço ao grande camarada Ferreira e por tanto tempo de amizade *sin cera*. Agradeço as lembranças boas dos colegas de classe: Cleber, Adriane, Cássio, Alexandre, Leandro, Lucas e Ferreira. Agradeço à minha esposa pelos momentos cujo diálogo era entre meus dedos e as teclas, meus olhos e a tela. Agradeço aos meus pais, Celso e Maria, por tUDO. Agradeço à Herika, minha irmã, por me ajudar com a revisão do texto. Agradeço ao amigo Alexandre Esteves, irmão de vontade. E para a senhora, minha avó, agradeço por me contar tantas histórias e quando penso na senhora, toda noite vira um palco onde escuto o mundo solitário respirar sozinho.

O exame da similaridade é proveitoso tanto para os argumentos indutivos quanto para os silogismos hipotéticos, bem como para a formulação das definições. Sua utilidade para o raciocínio indutivo se explica porque sustentamos que é pela indução dos particulares, com base nas similaridades, que inferimos o universal, visto não ser fácil empregar a inferência se desconhecemos os pontos de similaridade.
Aristóteles

RESUMO

Desde o início da área de Planejamento, muitas linguagens apareceram para descrever domínios e seus problemas. Atualmente, a PDDL é amplamente usada e aceita pelos pesquisadores devido a sua robustez e expressão. Contudo, alguns pesquisadores da área criticam a PDDL, pois ela é uma linguagem semelhante a um código fonte e difícil de ser utilizada como linguagem de modelagem de domínio. Isso foi uma das motivações para as pesquisas sobre Engenharia do Conhecimento para Planejamento Automático.

Atualmente, a Engenharia do Conhecimento para domínios de planejamento e seus problemas tem demonstrado relevância para as aplicações de Planejamento Automático. Desde o *ICKEPS* de 2005 (*International Competition on Knowledge Engineering for Planning & Scheduling*), a comunidade de Planejamento Automático tem pesquisado sobre novas ferramentas, métodos e linguagens de modelagem.

Nesse trabalho, a ferramenta de engenharia do conhecimento itSIMPLE possui a linguagem UML.P conhecida como a UML com abordagem para planejamento automático. A necessidade é interpretar os modelos UML.P em modelos PDDL. Como a UML.P e a PDDL são semiformais, optou-se por utilizar a F-Logic, que é uma lógica correta e completa para apoiar essa interpretação.

Palavras-chave: Planejamento Automático, Engenharia do Conhecimento, Interpretação, PDDL, UML.P e F-Logic.

ABSTRACT

Since the beginning of the Planning in Artificial Intelligence field, many languages have appeared to describe domains and its problems. Nowadays, PDDL is broadly used and accepted by the researchers due to its robustness and expression. However, some researchers in the automatic planning field criticize PDDL, because it is a language similar to a source code and difficult to be used as a domain modeling language. This was one of the motivations for the research in the Knowledge Engineering for Automatic Planning.

Nowadays, the Knowledge Engineering for planning domains and problems has shown relevance to the automatic planning applications. Since the ICKEPS´05 (*International Competition on Knowledge Engineering for Planning & Scheduling*), the automatic planning community has researched new tools, methods, and modeling languages.

In this work, the knowledge engineering tool itSIMPLE has the UML.P language known as UML for the Automatic Planning approach. The need is to interpret the UML.P models into PDDL models. As UML.P and PDDL are semi-formal languages, it was decided to use F-Logic, which is a sound and complete logic to give support to this interpretation.

Key words: Automated Planning, Knowledge Engineering, Interpretation, PDDL, UML.P, and F-Logic.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SÍMBOLOS

@ – Separador de método e parâmetros em F-Logic;

; – Separador de métodos em F-Logic;

: – Auxiliar de expressão “é-um” de objeto em F-Logic;

:: – Auxiliar de expressão “é-um” de classe em F-Logic;

-> – Auxiliar de expressão de dado escalar não herdável em F-Logic;

->> – Auxiliar de expressão de conjunto de dados não herdável em F-Logic;

*-> – Auxiliar de expressão de dado escalar herdável em F-Logic;

*->> – Auxiliar de expressão de conjunto de dados herdável em F-Logic;

=> – Define assinatura de dado escalar não herdável em F-Logic;

=>> – Define assinatura de conjunto de dados não herdável em F-Logic;

U – Domínio de uma F-Estrutura;

$\overset{I}{\longleftrightarrow}$ – Símbolo que denota a Interpretação Correspondente PDDL e F-Logic. Esse símbolo é diferente do símbolo lógico de equivalência, sendo ele apenas um símbolo apresentando uma possível interpretação correspondente entre duas linguagens.

SUMÁRIO

CAPÍTULO 1

1. INTRODUÇÃO

A Engenharia do Conhecimento (EC) para modelagem de domínios e problemas já é aceita atualmente na comunidade de Planejamento (*AI Planning & Scheduling Community*) como relevante para o sucesso das aplicações de Planejamento em Inteligência Artificial (IA). Sobre essa relevância pode-se citar o *ICKEPS* (*International Competition on Knowledge Engineering for Planning & Scheduling*), que existe desde 2005, e é uma motivação para as pesquisas na área. Apesar da EC ainda estar no início das atividades (MCCLUSKEY, 2002) e a área de Planejamento conviver com algumas restrições, tais como: ferramentas, métodos e, principalmente, linguagens de modelagem de domínios.

Desde o início das pesquisas em Planejamento, muitas linguagens têm sido desenvolvidas para descrever domínios e seus problemas, evidenciando a necessidade da EC para aprimorar a representação do conhecimento nessa área. Atualmente, a linguagem PDDL (MCDERMOTT, 1998), *Planning Domain Definition Language*, é amplamente usada e aceita pelos pesquisadores da comunidade de Planejamento devido a sua robustez e expressividade.

Contudo, a PDDL é similar a uma linguagem executável ou a um "código de máquina" (McCLUSKEY, 2002). Por essa razão, ela é difícil de ser aceita no contexto de uma linguagem de modelagem. De acordo com trabalhos importantes sobre a EC em (MCCLUSKEY, 2000) e em (MCCLUSKEY, 2002) para a comunidade de Planejamento, existe a necessidade de linguagem de modelagem, de métodos, de ferramentas e de um processo para a EC aplicada ao Planejamento Automático. Por isso, já no primeiro ano do *ICKEPS*, houve apresentações de ferramentas de EC que utilizavam outras linguagens: Gipo (SIMPSON et al. 2001) e UML.P (VAQUERO et al. 2005) de modelagem de domínios, buscando uma nova maneira de se projetar domínios de Planejamento Automático, que até então só podiam ser descritos em PDDL no que tange aos planejadores independentes de domínio. Essas ferramentas têm a proposta de auxiliar a modelagem de domínios de planejamento e, em seguida, gerar as descrições em PDDL automaticamente.

Uma dessas ferramentas de EC (VAQUERO et al, 2005) chama-se itSIMPLE que possui uma linguagem visual e intuitiva (VAQUERO et al, 2006), que até então tem demonstrado

sua utilização para modelagem de conhecimento de domínios de planejamento nos congressos nacionais e internacionais. A última demonstração sobre o itSIMPLE para a comunidade de Planejamento Automático foi no *ICAPS* (*International Conference on Automated Planning & Scheduling*) de 2008, onde foi apresentado em um *workshop* a aplicação do itSIMPLE resolvendo problemas de gestão do conhecimento, escalonamento e planejamento na área de engenharia de software (UDO et al. 2008). No mesmo ICAPS, foi demonstrada a extração de conhecimento do diagrama de classes (GOMES et al. 2008) e sugestões para a competição de engenharia do conhecimento (VAQUERO et al. 2008). O itSIMPLE tem demonstrado resultados advindos de mais de 3 anos de pesquisa na EC para Planejamento Automático.

O itSIMPLE possui a linguagem UML (BOOCH, JACOBSON, RUMBAUGH, 1999), que surgiu na área da Engenharia de Software na década de 90, com o objetivo de padronizar noções de requisitos, classes, objetos, estados e elementos arquiteturais. Atualmente é mantida pela *OMG* (*Object Modeling Group*) e pode ser usada em muitas áreas da ciência que necessitem de modelagem de domínios. O itSIMPLE utiliza a UML.P, que é a UML com abordagem para Planejamento Automático (VAQUERO et al, 2006).

Tanto a UML.P quanto a PDDL são entendidas como semiformais. A condição de ser semiformal neste contexto significa uma linguagem que não possui um mapeamento adequado entre sua sintaxe e sua semântica, apesar de existirem esquemas sobre como utilizar a sua sintaxe (LÍŠKA, 2005). No caso da UML.P, ela possui diagramas e elementos diagramáticos para modelar domínios. Os próprios diagramas e elementos não estão coerentemente mapeados em uma semântica, de modo que apresente problemas se existirem. No caso da PDDL, existe uma semântica de sua versão 2.1 (FOX, M.; LONG, 2003), mas ainda não há um mapeamento adequado de sua sintaxe com a sua semântica.

Atualmente, os modelos UML.P são interpretados em Modelos PDDL pelo itSIMPLE, o que leva à necessidade (figura 1) de uma formalização dessa interpretação.

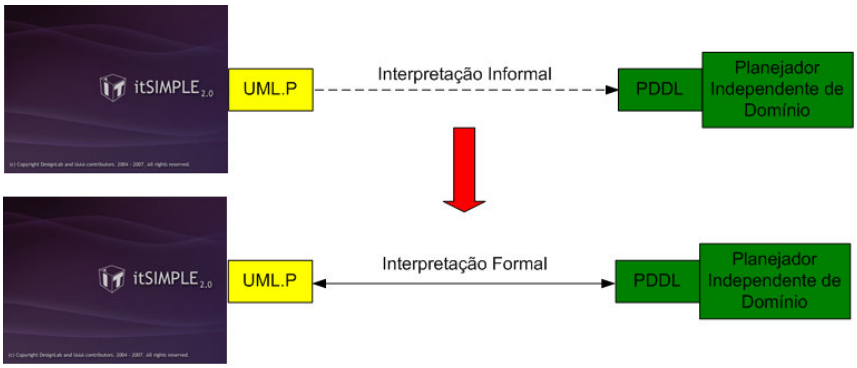


Figura 1 – A necessidade da Interpretação Formal.

Esse trabalho deve apresentar a Interpretação Formal, especificamente interpretação e equivalência de modelos, com a qual se garante que o modelo UML.P seja interpretado em modelo PDDL. Também, esse trabalho fornece condições iniciais onde as alterações em qualquer um dos modelos sejam refletidas em ambos modelos, tanto do modelo UML.P em modelo PDDL quanto modelo PDDL em modelo UML.P.

1.1. Motivações

As motivações desse presente trabalho, para melhor citá-las, foram classificadas em motivações acadêmicas e motivações industriais, assim como segue abaixo:

- Acadêmica
  - Atender às necessidades da área de Planejamento Automático, que por sua vez, desde seu início, os pesquisadores direcionaram a pesquisa da área para melhorar as técnicas e eficiência na geração de planos, deixando a Engenharia do Conhecimento em segundo plano (VAQUERO, 2007);
  - Formalizar a interpretação da UML.P em PDDL através de uma Lógica completa e correta como a F-Logic (KIFER; LAUSEN; WU, 1995) para futuramente manter Bases de Conhecimento de domínios de Planejamento Automático;
  - Atualmente, há idéias sobre o uso de ferramentas da Engenharia do Conhecimento por usuários comuns que não saibam quase nada sobre Planejamento Automático e necessitem de um plano com uma seqüência de ações e que leve em consideração os recursos utilizados (MCCLUSKEY, 2002).
- Industrial
  - A UML já é amplamente utilizada no quotidiano das empresas pelos engenheiros de software e o objetivo da área de Planejamento Automático em lidar com problemas reais, tal como: identificar a seqüência de ações a partir de um estado inicial do problema, fazendo o escalonamento de recursos para então alcançar o estado objetivo, que resolve o problema. Tudo isso, dado o conhecimento que se tem do domínio em questão para, então, gerar scripts de trabalho para vários profissionais atuarem com mais precisão e não-conformidades tendendo a zero no trabalho. Avanços importantes no

contexto industrial. A UML.P pode ser implantada nas empresas para uso dos engenheiros de software através da ferramenta itSIMPLE (VAQUERO et al. 2005), que possui planejadores independentes de domínio embarcados. No caso dessa dissertação, são planejadores que apenas precisam do domínio e do problema especificados em PDDL.

1.2. Objetivo

O objetivo deste trabalho é demonstrar que, para qualquer domínio de planejamento, a interpretação do diagrama de classes da UML.P, especificamente classes, atributos e associações, pode ser realizada em tipos e predicados da PDDL.

1.3. Estrutura da Dissertação

Para a formalização da interpretação do diagrama de classes da UML.P em tipos e predicados da PDDL, para que haja confiabilidade na interpretação automática da ferramenta de EC, itSIMPLE, o conteúdo deste trabalho foi organizado da seguinte maneira:

Capítulo 2: nesse capítulo, apresenta-se uma descrição dos conceitos envolvidos na área de Engenharia do Conhecimento e na área de Planejamento Automático. Há uma apresentação sobre o conceito de Planejamento de Ações e algumas de suas principais linguagens de modelagem de domínios e problemas. Há também algumas representações de conhecimento na forma de redes mostrando a necessidade da modelagem de conhecimento. Apresenta-se também a F-Logic para a representação de objetos. Serão apresentadas algumas ferramentas de Engenharia do Conhecimento inclusive para a área de Planejamento Automático e, por último, serão apresentados alguns itens importantes sobre a Engenharia do Conhecimento Aplicada para o Planejamento Automático. A partir desse capítulo, entende-se o uso da UML.P como uma representação de conhecimento mais promissora para o uso dos engenheiros do conhecimento do que as atuais linguagens da área de Planejamento Automático. Entende-se também o porquê da utilização da F-Logic para mapear a sintaxe visual da UML.P. Além disso, a F-Logic possui mapeamento com a Lógica de Predicados, o que favorece essa mesma lógica no mapeamento da sintaxe da PDDL. Assim, com esses entendimentos, cria-se as condições iniciais para a contribuição desse trabalho.

Capítulo 3: nesse capítulo apresenta-se a F-Logic com mais detalhes, especificamente definições sobre sua sintaxe e sua semântica, de modo a entender seu uso nos capítulos

posteriores nas interpretações de Modelos UML.P e PDDL. Pois, essas interpretações devem contar com o mapeamento de ambas linguagens para a F-Logic.

Capítulo 4: nesse capítulo apresenta-se a interpretação correspondente da F-Logic e PDDL. Essa apresentação mostra como os elementos da PDDL, tais como, tipos e predicados são interpretados em classes e métodos da F-Logic e *vice versa*.

Capítulo 5: nesse capítulo apresenta-se a *Interpretação Completa* do diagrama de classes da UML.P em F-Logic (GOMES, 2008) de modo que um subconjunto dessa interpretação, chamada de interpretação *IFR* de forma reduzida, é evidenciada para demonstrar a equivalência com a interpretação correspondente da PDDL e F-Logic.

Capítulo 6: nesse capítulo demonstra-se a equivalência de f-átomos da F-Logic provenientes da Interpretação Correspondente e da Interpretação *IFR* de Forma Reduzida, que respectivamente tratam da PDDL e da UML.P. Demonstra-se também que a Interpretação Correspondente é conseqüência lógica da Interpretação Completa (GOMES, 2008) e, assim, o Modelo UML.P pode ser interpretado em Modelo PDDL.

Capítulo 7: nesse capítulo apresenta-se a conclusão desse trabalho, assim como as conjecturas sobre trabalhos futuros.

CAPÍTULO 2

2. REVISÃO BIBLIOGRÁFICA

Neste capítulo foi realizada uma revisão bibliográfica com o intuito de avaliar conceitos necessários para este trabalho, que surgiram em pesquisas sobre Representação do Conhecimento, Engenharia do Conhecimento e Planejamento Automático; apresentar a UML e sua abordagem para planejamento automático – chamada UML.P (VAQUERO et al. 2006) e juntamente da PDDL apresentando algumas lacunas que necessitam ser preenchidas. Além disso, esse capítulo faz uma preparação dos conceitos que serão utilizados para mostrar que é factível para o engenheiro do conhecimento, ou projetista de domínio realizar a modelagem de qualquer domínio em UML.P e obter instantaneamente esse domínio já especificado em PDDL.

2.1. Descrição de Ações

Para a área de Planejamento Automático, a descrição das ações que afetam os estados do mundo com mudanças é importante como representação do conhecimento e componente fundamental para uma linguagem de modelagem de domínios (LIFSCHITZ; REN, 2007). De acordo com Dana Nau, adquirir esse conhecimento é uma das coisas mais importantes para a pesquisa de Planejamento Automático, pois os planejadores poderiam ser mais úteis ao lidar com problemas reais (NAU, 2007).

Surgiram muitas abordagens para representação de ações, tais como: o cálculo situacional (*situation calculus*) (MCCARTHY; HAYES, 1969), o cálculo de eventos (*event calculus*) (KOWALSKI; SERGOT, 1986) e o STRIPS (FIKES; NILSSON, 1971), todavia, na área de Planejamento Automático, a linguagem ADL (PEDNAUT, 1989) foi a que melhor atendeu a área na modelagem de domínios reais, até então, difíceis de serem modelados pelo STRIPS.

A maneira pela qual a ADL foi criada permite trabalhos de pesquisa sobre descrição de ações modernas como o C+ (LIFSCHITZ; REN, 2007), que toma como referência sua formalização e o raciocínio sobre os efeitos das ações (PEDNAULT, 1994).

Para este trabalho, logo abaixo, serão apresentadas algumas linguagens de modelagem de domínios de planejamento. Como a STRIPS, que define as ações como uma transformação de

um conjunto de fórmulas em lógica de primeira ordem, e a ADL, que possui uma descrição dos efeitos das ações para atender a efeitos dependentes de contexto e por último a PDDL, que incorpora a STRIPS, a ADL e outras linguagens na sua descrição.

Recordando que esse presente trabalho tem por objetivo tratar os tipos e predicados da PDDL, que são por sua vez importantes para a descrição das ações.

2.2. Linguagens de Modelagem de Domínios de Planejamento

2.2.1. STRIPS – *STanford Research Institute Problem Solver*

A partir do início da década de 70 houve a introdução da STRIPS (FIKES; NILSSON, 1971), que inovou a área de Planejamento com a descrição de estados do mundo (TONIDANDEL, 2003), além de uma estrutura para problemas de planejamento usada até os dias de hoje.

A STRIPS, além de ser um sistema de planejamento, tem uma representação conhecida com o mesmo nome STRIPS, que é uma representação clássica de Planejamento que usa em sua estrutura fórmulas em lógica de primeira ordem para descrever o mundo e um conjunto de operadores que representam as ações (COPPIN, 2004). O operador da STRIPS possui dois componentes: a Precondição e o Efeito, que são utilizados para demonstrar as regras para disparo de uma ação e mudanças no estado do mundo.

A **precondição** de um operador, segundo (RUSSELL; NORVIG, 2004), possui um conjunto de literais que devem ser válidos para que o operador seja disparado e, caso isso ocorra, o **efeito** do operador mudará o estado atual para um novo estado, porque possui um conjunto de literais positivos que deve ser adicionado a um determinado estado do mundo (lista de adição) e um conjunto de literais negativos que deve ser eliminado desse mesmo estado do mundo (lista de eliminação). Para exemplificar melhor a linguagem STRIPS, considere o exemplo abaixo retirado de (LUGER, 2004) que lida com (P) precondições, (A) a lista de adição e (E) a lista de eliminação:

Tabela 1 - STRIPS

Pega(X)	P: agarra( ) ∧ livre(X) ∧ sobre_a_mesa(X)
	A: agarra(X)
	E: sobre_a_mesa(X) ∧ agarra( )
Descarrega(X)	P: agarra(X)
	A: sobre_a_mesa(X) ∧ agarra( ) ∧ livre(X)
	E: agarra(X)

Fonte: Luger, 2004

Na tabela acima, há a demonstração das listas de precondições (P), de adição (A) e de eliminação (E). Nesse exemplo, o operador **pega(X)** significa que o operador é responsável pela ação pega o bloco **X**, entretanto, para que esse operador seja disparado, ele precisa satisfazer as precondições (P), isto é, a garra do robô deve estar livre (**agarra( )**), o bloco **X** não pode possuir nada sobre ele (**livre(X)**) e o bloco **X** deve estar sobre a mesa (**sobre_a_mesa(X)**). Se o operador **pega(X)** satisfizer essas precondições, ele deve aplicar a lista de eliminação (E), que exclui do estado atual o fato do bloco X estar sobre a mesa (**sobre_a_mesa(X)**) e o fato da garra estar livre (**agarra( )**), e deve aplicar a lista de adição (A), que inclui o fato da garra do robô segurar o bloco X (**agarra(X)**) no estado atual tranformando-o no estado sucessor.

Além disso, é importante citar que a precondição dos operadores deve possuir apenas literais positivos e isso é uma das restrições na STRIPS. Ou seja, não podem existir literais negativos na precondição dos operadores.

Outras restrições na STRIPS que devem ser citadas são: a **Hipótese de Mundo Fechado**, cujos literais que não são mencionados são tratados como falsos, o **Objetivo do Problema**, que deve conter apenas conjunções, assim como os **efeitos** de cada operador e Literais Básicos (sem quantificadores) (RUSSELL; NORVIG, 2004). Essas restrições acabaram evidenciando fragilidades na STRIPS ao lidar com problemas de Planejamento.

Entretanto, havia uma fragilidade da linguagem STRIPS na literatura que a deixava informal demais para uma linguagem com o objetivo de especificar domínios e problemas de Planejamento, ocasionando o mau uso da linguagem STRIPS. Daí, isso foi tratado por LIFSCHITZ (1987), que definiu uma semântica que a direcionou para um uso adequado.

Enfim, as outras fragilidades citadas acima também motivaram pesquisadores a tentar solucioná-las, ao passo que houve a criação de uma linguagem chamada ADL, que é uma extensão da STRIPS.

2.2.2. ADL – *Action Description Language*

A ADL (PEDNAUT, 1989) é também uma linguagem de especificação de domínios e problemas de Planejamento, que é mais expressiva que a STRIPS e, por isso, pode representar problemas de planejamento que seriam difíceis para a STRIPS representar (COPPIN, 2004).

A ADL proporciona, segundo (RUSSELL; NORVIG, 2004) (COPPIN, 2004), efeitos condicionais, tipos, disjunções de literais, predicado de igualdade e quantificadores, além da hipótese de mundo aberto. Os **efeitos condicionais** são efeitos que devem acontecer de acordo com sua condição para que a ação seja executada. Os **tipos** são associados às variáveis, por exemplo, **(b : Bloco, h : Mão)**. As **disjunções de literais** são representações tais como: **¬Sobre(blocoA, blocoB) ∧ (Em(blocoA, Mesa) ∨ Sobre(blocoA, blocoC))**. Na ADL há o **predicado de igualdade**, por exemplo, **(bloco1 = bloco2)**. A **hipótese de mundo aberto**, diferentemente da hipótese de mundo fechado na STRIPS, determina que os literais não citados são informações desconhecidas.

Essas melhorias proporcionadas pela ADL, ainda segundo (RUSSELL; NORVIG, 2004), facilitaram a leitura dos modelos de planejamento, de modo que a ADL tem sido usada em trabalhos atuais de Planejamento Automático para Sistemas de Transição de Grafos (EDELKAMP; JABBAR; LAFUENTE, 2005), o que demonstra ganhos nos tratamentos de problemas de Planejamento de domínios reais devido à flexibilidade da ADL. Algumas dessas melhorias foram também incorporadas na própria linguagem STRIPS.

No entanto, a área de Planejamento Automático, com as melhorias supracitadas com a criação da ADL, além de melhorias na própria STRIPS e outras linguagens que surgiam e haviam de surgir, demonstrava a falta de padronização na área e isso contribuía negativamente até para a avaliação do desempenho dos planejadores (MCDERMOTT, 1998). Esse status na área motivou a padronização de uma linguagem de definição de domínios de planejamento: PDDL.

2.2.3. PDDL – *Planning Domain Definition Language*

A PDDL em 1998 foi apresentada por Drew McDermott (1998) e se tornou um padrão como linguagem de especificação de domínios e problemas utilizada nas competições de planejamento. A PDDL incorpora as descrições feitas em STRIPS ou ADL. Apesar de alguns pesquisadores da comunidade de Planejamento Automático não apreciar algumas das funcionalidades da PDDL, tais como: a descrição de tempo, a semelhança com código fonte, o trabalho pouco intuitivo, etc.; mesmo assim, não há como negar o avanço que ela tem proporcionado à área (FOX, M.; LONG, 2003). A PDDL desde o seu começo teve um papel extremamente importante por ter pressionado a comunidade de Planejamento a produzir planejadores que controlassem problemas de planejamento mais realísticos (MCDERMOTT, 2003).

Ao buscar tratar problemas de Planejamento mais realísticos, ou seja, domínios reais e complexos, a modelagem em PDDL tornou-se bastante extensa e difícil de ser usada. Além disso, não é trivial apresentar ou compreender um modelo em PDDL. Isso era diferente quando a área de Planejamento lidava apenas com *Toy Problems* (MCCLUSKEY, 2002) porque a modelagem em PDDL não era extensa e reduzia a complexidade da linguagem devido ao tamanho do domínio.

Pode-se perceber que nos idos do Planejamento Clássico, quando a maioria dos problemas de Planejamento eram conhecidos como *Toy Problems,* não havia problemas com a STRIPS no que tange à modelagem extensa, porque o domínio era simples em relação ao que se passou a exigir da comunidade de Planejamento. Para McCluskey (2003), a PDDL não foi projetada para ser associada com modelagem de domínios, pois a mesma, segundo McCluskey, parece uma “linguagem de máquina”. Olhando singularmente para esse aspecto, a PDDL não é uma linguagem de modelagem prática do engenheiro do conhecimento ou projetista de domínio para ser aplicada a modelagem e isso se amplia quando se trata de domínios reais e complexos.

Dessa forma, há fragilidades na PDDL como uma linguagem de modelagem de domínios de planejamento, ao passo que, atualmente, ela possui alta expressividade ao lidar com domínios reais e complexos devido à sua especificação e semântica criada para a versão 2.1 (FOX, M.; LONG, 2003), que é uma extensão do trabalho de Lifschitz (1987). E isso aponta

para a necessidade da área de Planejamento Automático manter essa expressividade da PDDL e, se possível, prover melhoria nessa expressividade.

Na próxima subseção, há uma demonstração do uso da PDDL para modelar o domínio Mundo dos Blocos e um problema de planejamento de Três Blocos.

2.2.4. Modelagem de Domínio e de Problema em PDDL

A modelagem em PDDL categoriza em certa ordem as informações que serão necessárias para os planejadores independentes de domínio gerarem um plano. Para elucidar essas informações, abaixo há algumas partes da modelagem de domínio do **Mundo dos Blocos** para explicar as categorias Requerimentos, Tipagem, Precondições Negativas, Precondições Quantitativas, Tipos, Predicados, Funções, Restrições e Ação da PDDL:

```
(define (domain Mundo_dos_Blocos)
    (:requirements  :typing  :negative-preconditions  :quantified-
    preconditions :constraints)
    (:types
      Manipulador - object
      Bloco - object
      Mesa  - object
```

Figura 2 – Definindo os requisitos de um Domínio em PDDL.

Na figura acima, há a definição do nome do domínio como Mundo_dos_Blocos. A categoria **:requirements** é uma das categorias mais importantes na PDDL, pois a cada definição de um novo modelo de domínio, os *requirements* definem as características do domínio a ser tratado, ou seja:

- [*:disjunctive-preconditions*] onde é permitido que o operador lógico OU seja utilizado nos objetivos;
- [*:typing*] define que o modelo pode representar tipos de objetos;
- *:strips* - essa característica define os domínios no formato da linguagem STRIPS;
- :*adl* - essa característica define um super conjunto do formato STRIPS, que inclui as seguintes características:
    - Declaração de tipos (:*types*);
    - Precondições negativas (:*negative-preconditions*);
    - Precondições disjuntas (:*disjunctive-preconditions*);
    - Igualdades (:*equality*);

- Precondições quantificadas (:*quantified-preconditions*);
- Efeitos condicionais (:*conditional-effects*).

Ainda na figura acima, o domínio do Mundo dos Blocos foi definido como um domínio que pode definir tipos, definir quantificadores e negações como precondições de operadores e definir restrições para o domínio. Além disso, esse domínio tem definido em **:Types**, três tipos de classes: Manipulador, Mesa e Bloco.

Já na figura abaixo, há a representação da categoria **:predicates** em PDDL que define os predicados do domínio **Mundo dos Blocos**:

```
(:predicates
  (sobre ?blo - Bloco ?blo1 - Bloco)
  (segurando ?man - Manipulador ?blo - Bloco)
  (sobreMesa ?blo - Bloco ?mes - Mesa)
  (garraLivre ?man - Manipulador)
  (nadaSobre ?blo - Bloco)
)
```

Figura 3 – A categoria :Predicates da PDDL.

Na figura acima, o domínio Mundo dos Blocos tem definido seus predicados que relatam a seguinte informação: um bloco fica sobre outro bloco ou não, o manipulador pode segurar um bloco, podem existir blocos sobre a mesa e o manipulador pode ter sua garra livre e, por último, um bloco pode ter sua face topo livre. Nessa categoria, o projetista de domínio pode definir as relações rígidas (*state invariants*) do domínio, que são as relações que se mantêm imutáveis no domínio, e as relações flexíveis (*fluents*), que são relações mutáveis no domínio.

Ainda sobre relações flexíveis de domínio, o Mundo dos Blocos pode ter exemplificado outra categoria importante da PDDL que funciona como uma relação flexível, chamada de **:functions**, que define as funções que lidam com valores numéricos, assim como segue abaixo:

```
(:functions
      (pesoMesa ?mes - Mesa))
```

Figura 4 – Domínio em PDDL do Mundo dos Blocos.

Na **figura 4**, a função “pesoMesa” mais o símbolo “?” seguido por “mes” e “Mesa” significam que a função “pesoMesa” possui como atributo “mês” do tipo “Mesa”. Todas as vezes que aparecer o símbolo “?” na PDDL, o engenheiro do conhecimento está descrevendo um determinado atributo ou parâmetro já lhe atribuindo o Tipo correto.

Ainda na **figura 4**, o domínio Mundo dos Blocos possui a definição de uma função que deve medir o peso da Mesa dados os blocos sobre ela. Na PDDL há também uma categoria sobre algo que restrinja o domínio com **restrições**. Por exemplo, pode-se ter definido para o domínio Mundo dos Blocos um peso máximo que a mesa suporte. Na PDDL, isso pode ser tratado com outra categoria sobre restrições.

Na figura abaixo, tem-se a categoria **:constraints**, onde o projetista de domínio pode restringir o domínio com regras:

```
(:constraints
   (and
     (forall (?blo1 - Bloco ?blo2 - Bloco ?blo - Bloco)
       (always
         (imply
           (and
             (sobre ?blo ?blo1)
             (sobre ?blo ?blo2)
           )
           (= ?blo1 ?blo2)
         )
       )
     )
```

Figura 5 – Restrição para os blocos do Mundo dos Blocos.

Na figura acima, a restrição significa que, se um bloco está sobre outro bloco, então, só pode estar sobre esse bloco.

Já a representação das ações é parte essencial para a área de planejamento e a PDDL mantém inclusive a abrangência que a ADL (PEDNAULT, 1989) trouxe para a área de Planejamento Automático no que tange à descrição de ações, que tem atualmente apoiado o desenvolvimento de novos trabalhos na área sobre transformação de grafos (EDELKAMP; JABBAR; LAFUENTE, 2005). Para exemplificar essa descrição de ações na PDDL, segue a definição de uma ação do Mundo dos Blocos:

```
(:action pegar
     :parameters (?m - Manipulador ?x - Bloco ?t - Mesa)
     :precondition
       (and
         (sobreMesa ?x ?t)
         (nadaSobre ?x)
         (garraLivre ?m))
     :effect
       (and
         (segurando ?m ?x)
         (not (garraLivre ?m))
         (not (sobreMesa ?x ?t)))
 )
```

Figura 6 – Ação pegar em PDDL do Mundo dos Blocos.

Na figura acima, a ação pegar somente será executada se as precondições forem satisfeitas tais como: um bloco esteja sobre a mesa, não haja nada sobre esse bloco e a garra do manipulador esteja livre. Se forem satisfeitas as precondições, então, a ação pegar será executada transformando o estado atual em um estado sucessor com as seguintes mudanças: a garra do manipulador não está mais livre, o bloco não está mais sobre a mesa e o manipulador está segurando o bloco.

Descrevendo todas as ações do domínio Mundo dos Blocos como a ação **pegar** da **figura 6**, finaliza-se a descrição de domínio da PDDL, sendo necessário agora definir um ou mais problemas de Planejamento para o domínio modelado. Para tanto, serão apresentadas partes do arquivo de problema de planejamento em PDDL para o Mundo dos Blocos. A seguir estão os objetos envolvidos no problema:

```
(define (problem TresBlocos)
    (:domain Mundo_dos_Blocos)
    (:objects
      A - Bloco
      B - Bloco
      C - Bloco
      Manipulador1 - Manipulador
      Mesa1 - Mesa
    )
```

Figura 7 – Problema dos TresBlocos em Planejamento.

Na figura acima, o problema de planejamento TresBlocos segue a definição do domínio Mundo_dos_Blocos, onde há três blocos A, B e C, um manipulador chamado Manipulador1 e uma mesa chamada Mesa1. Esses são os objetos envolvidos no domínio.

A seguir, será apresentada a figura referente ao estado inicial do problema de planejamento:

```
(:init
   (sobreMesa C Mesa1)
   (sobre B C)
   (sobre A B)
   (nadaSobre A)
   (garraLivre Manipulador1)
 )
```

Figura 8 – Estado Inicial do Problema em PDDL do Mundo dos Blocos.

Na **figura 8**, referente ao estado inicial do problema, existem os seguintes fatos: o bloco C está sobre a Mesa1, o bloco B está sobre o bloco C, o bloco A está sobre o bloco B, não há nada sobre o bloco A e a garra do Manipulador1 está livre.

Abaixo, segue a definição do estado objetivo:

```
(:goal
  (and
    (sobre C B)
    (sobre B A)
    (sobreMesa A Mesa1)
    (nadaSobre C)
    (garraLivre Manipulador1)
  )
)
```

Figura 9 – Estado Objetivo do Problema em PDDL do Mundo dos Blocos.

Na figura acima, há a definição do estado objetivo onde o bloco C deve estar sobre o bloco B, o bloco B deve estar sobre o bloco A, o bloco A deve estar sobre a Mesa1, nada deve estar sobre o bloco C e o Manipulador1 deve ter sua garra livre.

Nessa subseção, foram expostas as categorias da PDDL e como o projetista de domínio pode definir domínios e problemas de planejamento utilizando essas categorias. Pode-se notar que essa linguagem não é apropriada para as atividades de aquisição, análise e modelagem de um processo de EC devido a sua semelhança com código fonte.

Na próxima seção, há uma revisão sobre representação de conhecimento na forma de redes com a finalidade de mostrar a propensão do ser humano desde épocas remotas a utilizar redes para melhor representar o conhecimento adquirido.

2.3. Representação de Conhecimento usando Quadros e Grafos

Primeiramente, é necessário definir o conceito **conhecimento** que esse presente trabalho aborda tomando definições da área de Engenharia do Conhecimento (SCHREIBER et al, 2000) que faz a seguinte distinção:

- **Dados** são sinais sem interpretação tais como caracteres, símbolos e concatenação desses últimos, que estão na vida do ser humano e máquinas em grande quantidade;
- **Informação** é o significado atribuído aos dados de tal forma que os dados sempre são os mesmos, entretanto, pode-se atribuir significado (informação) diferente aos dados e, logo, a informação nem sempre será a mesma;
- **Conhecimento** é o todo composto de dados e informação de tal maneira que tanto as pessoas como as máquinas podem pôr o uso desse conhecimento na prática. Dessa maneira, o Conhecimento possui dois aspectos distintos: um é obter o propósito para se alcançar o objetivo e o outro, capacidade de gerar nova informação.

Para exemplificar essas distinções, segue a tabela abaixo:

Tabela 2 - Distinções entre Dados, Informação e Conhecimento

Distinção entre Conceitos		
	Característica	Exemplo
Dado	Caracteres Brutos sem interpretação	...---...
Informação	Significado anexado ao dado	SOS
Conhecimento	Propósito e Competência para a Informação; Potencial para gerar ação.	Alerta de Emergência → Início de Operação de Resgate

Fonte: SCHREIBER et al, 2004

Dadas as distinções supracitadas entre dados, informação e conhecimento, parte desse trabalho cita modelos projetados em uma representação de conhecimento em forma de rede chamada de UML para Planejamento Automático, conhecida na área como UML.P. Essa linguagem possui diagramas que, de acordo com SOWA (2006), são redes. O fato de cada diagrama possuir um propósito, SOWA (2006) cita a UML como rede híbrida. Entretanto, nem todos os diagramas da UML são redes. A propósito, os quatro diagramas da UML.P são redes: casos de uso, máquina de estados, objetos e classes.

Dessa maneira, buscou-se com essa revisão avaliar a evolução dessa estrutura de representação de conhecimento até o que a UML.P é atualmente para Planejamento Automático.

2.3.1. Rede Semântica – Semantic Network

A primeira Rede Semântica apareceu na história no século 234-305 d.C. cujo filósofo Porfírio comentou o método de Aristóteles para categorização de supertipos e subtipos usando uma rede. Apenas, em 1909, Charles S. Peirce criou o que ele chamou de grafos existenciais com a primeira formalização da rede semântica usando a lógica moderna (RUSSELL; NORVIG, 2004) (SOWA, 2006).

Porém, houve um destaque na década de 60 (RUSSELL; NORVIG, 2004) quando Ross Quillian propôs um modelo computacional para a memória humana chamado de memória semântica justamente pelo seu interesse por memória humana e processamento de linguagens (BITTENCOURT, 2006). Já na década de 70, houve grande interesse pelas redes semânticas implementadas em sistemas para compreensão da linguagem natural devido à facilidade de representar o conhecimento com a sua notação.

A Rede Semântica é uma representação visual de nós e arcos que se relacionam. Os nós são qualificados como categorias ou classes, conceitos e objetos, ao passo que os arcos, geralmente, são qualificados como predicados em lógica de primeira ordem e possuem a restrição de apenas relacionar dois nós, ou seja, um relacionamento binário. Através desses componentes da rede semântica e uma orientação usada para modelá-la, obtém-se uma representação do domínio pretendido (SOWA, 2006).

A orientação para a modelagem de uma Rede Semântica ajuda na leitura do domínio representado e é composta de algumas regras, tais como (NILSSON, 1998):

- Há dois tipos de Nós:
  - Nós que podem ser categorias ou propriedades das categorias;
  - Nós que representam os próprios objetos do domínio.
- Há três tipos de Arco:
  - Arco denotando herança conhecido como arco “é-um”;
  - Arco para instância de categorias;
  - Arcos para funções.

Entretanto, Bittencourt (2006) cita também o arco “parte-de” e chama os “Arcos para funções” de específicos do domínio chamando-os de *Traços*. Essa definição de arcos e nós ajudam na interpretação do domínio. Todavia, não impede que o projetista cometa erros na modelagem da rede.

Apesar da Herança de Propriedades ser uma das características mais importantes do método de representação das redes semânticas, é necessário implementar políticas de herança para evitar problemas tais como a Herança Múltipla que leva a erros de dedução sobre a rede.

Para exemplificar como é uma rede semântica, há uma modelagem reduzida do capítulo 2 desse presente trabalho na **figura 10** e essa rede possui arcos "é-um", "é-parte" e "traços".

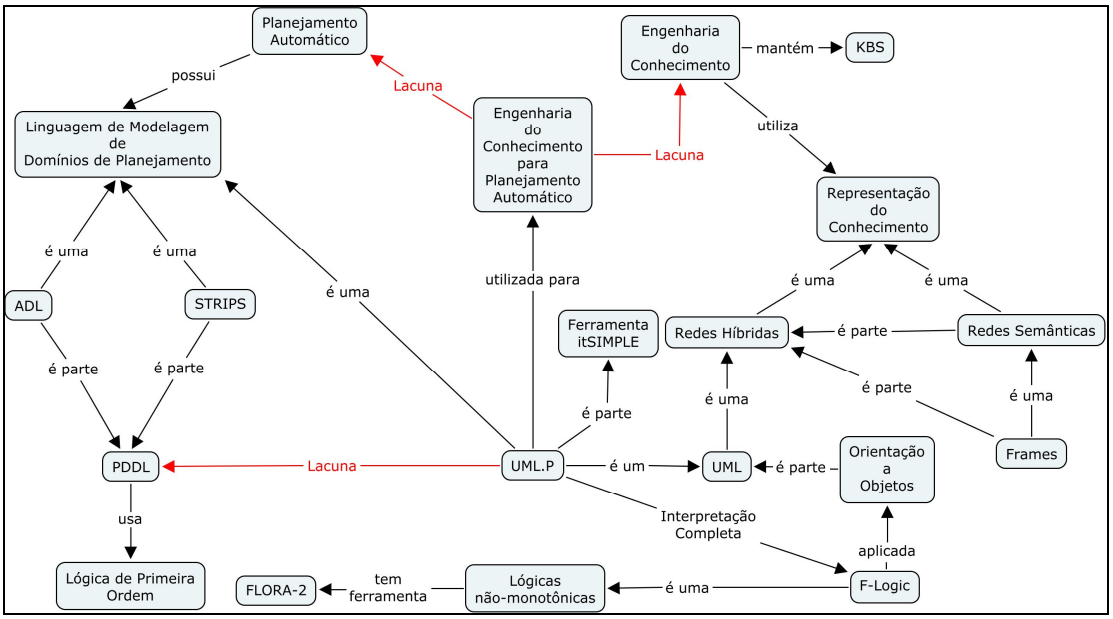


Figura 10 – Rede Semântica.

Com a rede semântica da **figura 10** tem-se uma leitura visual e intuitiva que facilita o entendimento do domínio. Por exemplo, sabe-se que a UML.P é uma especialização da UML e que ela é uma rede híbrida.

A coerência do domínio deve depender bastante da visão do projetista ou engenheiro do conhecimento, de tal forma que a Rede Semântica possa ser validada. Contudo, mesmo com essa restrição, é evidente o uso massivo das redes semânticas como representação do conhecimento, tanto que Marvin Minsky na década de 70, segundo (RUSSELL; NORVIG, 2004) e (BITTENCOURT, 2006), produziu um trabalho sobre *Frames,* usando as redes semânticas.

2.3.2. Quadros – *Frames*

Os Quadros, mais conhecidos como *Frames*, foram desenvolvidos com base nas redes semânticas cujo objetivo foi destacar as propriedades das categorias e evidenciar os

relacionamentos com outros quadros (BENDER, 1996). Um *frame* é algo que contém *Slots* (vaga em um esquema), onde cada atributo de uma categoria ou *frame* é um *slot.* Um *frame* pode se relacionar com outros *frames*. Por exemplo, dois *frames*: Veículo e Caminhão, onde Caminhão é-um Veículo. O *frame* Caminhão possui um *slot* que é o "é-um" do tipo Veículo, dessa forma, veja abaixo uma figura com esse exemplo:

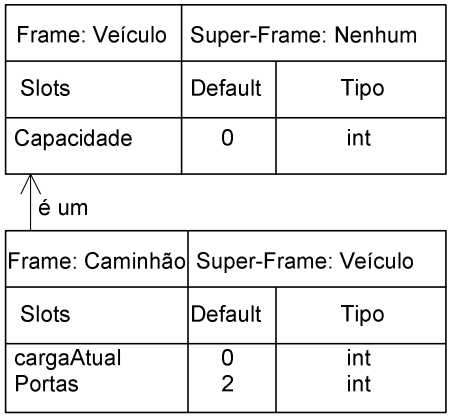


Figura 11 – Quadros.

Marvin Minsky foi criticado por criar algo já existente nas linguagens orientadas a objetos (RUSSELL; NORVIG, 2004). Todavia, seu trabalho acabou fornecendo uma melhor direção para as linguagens orientadas a objetos (BITTENCOURT, 2006) e ao longo da década de 70 houve esforços para formalizar a rede semântica e os *frames*.

Pode-se tratar os *frames* com lógica de primeira ordem como no exemplo a seguir (BENDER, 1996):

Tabela 3 - Convertendo Frames para Regras

Predicado	Significado
frame(X)	X é um frame
Is-a(X, Y)	Frame Y é um caso especial do frame X
slot(X, S)	S é um slot de X
default(X, S, V)	V é o valor default do slot S de X

Fonte: BENDER, 1996

Além da possibilidade de se aplicar a lógica em sistemas baseados em *frames* com seus atributos internos, esses sistemas podem tratar de um raciocínio guiado por expectativas, pois a partir dos atributos internos e os relacionamentos dos *frames* busca-se preencher um objetivo inter e intra *frames* (BITTENCOURT, 2006).

Entretanto, houve aplicações diferentes de redes que levou SOWA (2006) a citar a UML como rede híbrida com diagramas responsáveis por cenários do domínio (casos de uso),

atividades, estados, classes, objetos e outros diagramas. Próxima subseção deve detalhar melhor a UML.

2.3.3. Rede Híbrida – *Hybrid Network*

A rede híbrida (neste trabalho trata-se de uma linguagem com uma coleção de redes com diferentes propósitos) de maior uso atualmente é a UML, que foi criada por Booch, Jacobson, Rumbaugh (1999) na área da Engenharia de Software para a modelagem de sistemas (SOWA, 2006) e possui um subconjunto de diagramas que são redes. A UML é citada como uma rede híbrida devido ao fato de possuir diferentes tipos de redes (SOWA, 2006), entretanto, nem todos os diagramas da UML são redes, tais como: diagrama de seqüência, diagrama de tempo, etc. A UML.P (VAQUERO et al, 2006), que é uma abordagem da UML para a área de Planejamento Automático, possui 4 (quatro) diagramas (redes):

- Diagrama de Casos de Uso – esse diagrama trata das interações entre os Agentes, chamados de Atores, e os casos de uso, subobjetivos do domínio. A seguir, um exemplo do domínio Logística para o diagrama de casos de uso:

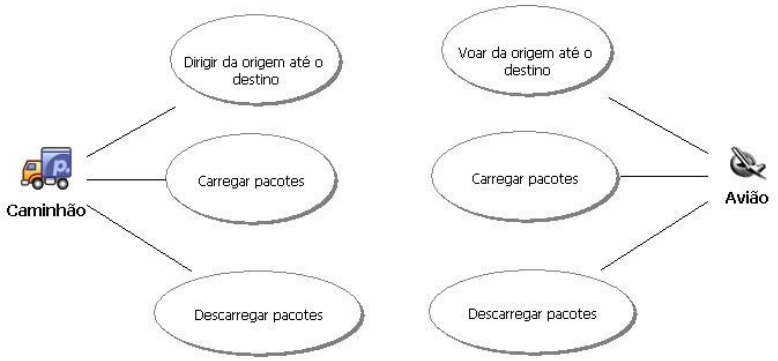


Figura 12 – Diagrama de Casos de Uso.

  Os casos de uso são como subobjetivos ou ações dos agentes na **figura 12** denotando “dirigir da origem até o destino”, “carregar pacotes”, “descarregar pacotes”, “voar da origem até o destino”, etc.

- Diagrama de Classes – é o principal diagrama da UML.P por ser o diagrama que demonstra a definição das relações entre as classes como tipos de objetos e as propriedades das classes. O diagrama de classe utilizado na UML.P como o diagrama que define o modelo estático dos domínios de planejamento automático possui propósitos semelhantes à rede semântica e aos *frames* quanto a modelar o

conhecimento dos domínios, além de possuir o conceito de multiplicidade entre as classes. A seguir, um exemplo de diagrama de classes do domínio Logística:

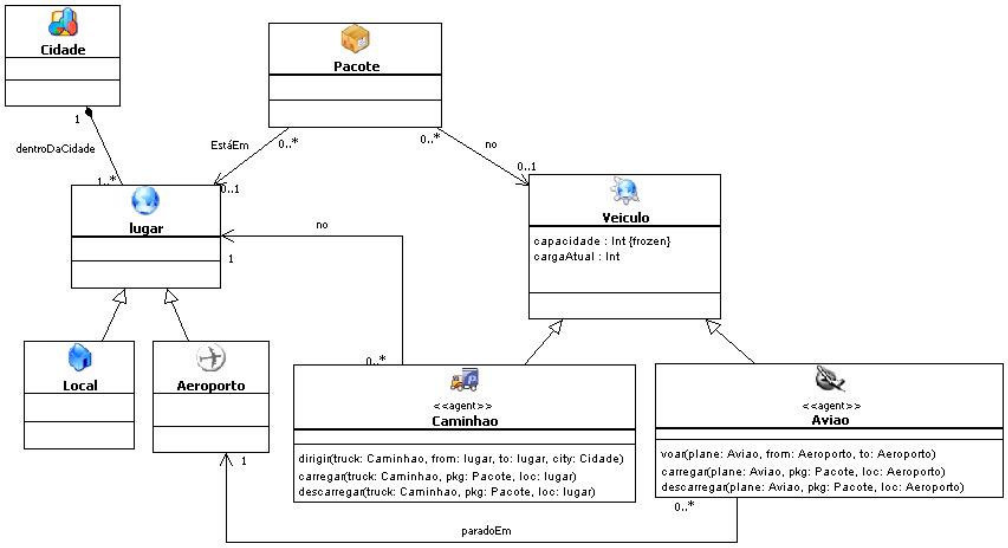


Figura 13 – Diagrama de Classes.

Na figura acima, o domínio Logística possui as classes **Avião** e **Caminhão** que são subclasses da classe **Veículo**. Um avião fica parado em um aeroporto, por isso, a associação **paradoEm** da classe **Avião** para a classe **Aeroporto** com multiplicidade **0..*** (zero ou muitos) para apenas **1**. A classe **Aeroporto** é subclasse da classe **Lugar**. O caminhão pode estar tanto no aeroporto quanto em algum local, por isso, a associação **no** da classe **Caminhão** para a classe **Lugar** com multiplicidade **0..*** (zero ou muitos) para apenas **1**. A classe **Lugar** é a superclasse das classes **Local** e **Aeroporto**. Qualquer lugar está dentro de apenas uma cidade, por isso, a classe **Cidade** possui associação do tipo "agregação" **dentroDaCidade** com a classe **Lugar**, onde a multiplicidade dessas duas classes significa que muitos lugares fazem parte exclusiva de uma cidade. Por último, qualquer pacote é carregado de um lugar para um veículo e transportado para outro lugar para, então, descarregá-lo lá mesmo. Por isso, a classe **Pacote** pode ter **0..*** instâncias de pacotes com associação **EstáEm** com a classe **Lugar** com multiplicidade **0..1**, ou seja, o pacote está no lugar ou não. O pacote pode estar no veículo, por isso a associação **no** da classe **Pacote** para a classe **Veículo** com multiplicidade **0..*** para **0..1**, com o mesmo significado da multiplicidade da classe **Pacote** para a classe **Lugar**.

- Diagrama de Máquinas de Estado – é um tipo especial de rede executável que não suporta paralelismo e não possui marcações para controlar o disparo de transições. Ao invés disso, utiliza-se da OCL, que é uma versão da Lógica de Primeira Ordem para a orientação a objeto, com a qual se especifica as precondições e os efeitos de um operador (SOWA, 2006). Entretanto, a OCL possui o mesmo problema que a PDDL: semelhança com código fonte. Abaixo, segue um exemplo desse diagrama no domínio Logística:

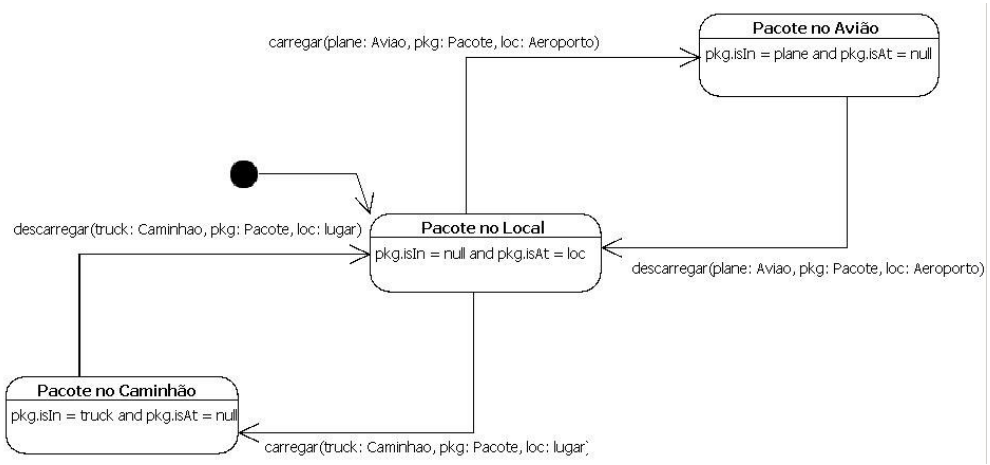


Figura 14 – Diagrama de Máquina de Estado.

- Diagrama de Objetos – é uma rede na qual há um grafo com nós que são objetos, instâncias de classes, com seus *links* que refletem os relacionamentos realizados no diagrama de classes. Esse diagrama é usado para definir o estado atual e o estado objetivo, que é uma necessidade para utilizar a tecnologia de Planejamento Automático. Segue um exemplo desse diagrama sobre o domínio Logística:

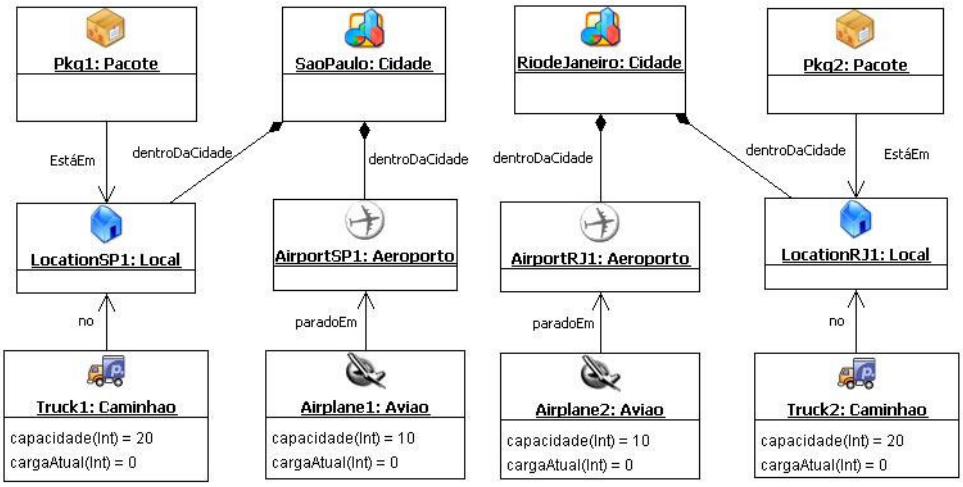


Figura 15 – Diagrama de Objetos.

Na figura acima, é apresentado o diagrama de objetos mostrando que há um pacote em um local na cidade de São Paulo. Nesse local há também um caminhão. Na cidade de São Paulo há um aeroporto e um avião parado lá. Da mesma forma há a mesma representação na cidade do Rio de Janeiro. Isso, por exemplo, pode ser a situação atual do domínio Logística e a situação objetivo pode ser o pacote de São Paulo no Rio de Janeiro e o pacote do Rio de Janeiro em São Paulo. Para fazer isso, há que se modelar outro diagrama de objetos para essa situação.

No diagrama de classes da UML.P existem particularidades semelhantes àquelas orientações que foram apresentadas nas Redes Semânticas quanto ao relacionamento de nós e essas particularidades servem para que a modelagem tenha uma sintaxe visual próxima aos requisitos do domínio em questão:

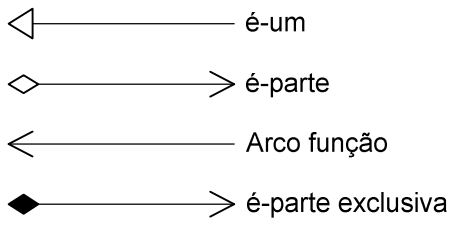


Figura 16 – Relacionamentos da UML.P.

Na UML há um relacionamento chamado de dependência, mas não é utilizada na UML.P para classes. Na **figura 16** demonstram-se 4 tipos de relacionamentos na UML.P, a primeira

seta é a que define herança assim como o "é-um" na rede semântica, a segunda seta significa que alguma classe chamada "parte" compõe outra classe chamada "todo", e a terceira seta é para as associações entre classes. A quarta seta não havia na rede semântica e é usada para intensificar a segunda seta "é-parte", pois ela significa que determinada parte só existe para compor determinado todo, por exemplo, o braço é parte de João e não existiria sem João.

Com os relacionamentos e as redes descritas acima é possível descrever um domínio real de Planejamento Automático e, também, descrever um conjunto de problemas pertinentes a esse domínio com o intuito de se obter Planos com uma seqüência de ações levando em consideração os recursos a serem utilizados. Exemplo disso, o Domínio de *Lean Software Development* na área de engenharia de software com um problema de cumprimento de prazos e escalonamento de recursos (UDO et al. 2008). Entretanto, falta uma definição formal para a UML.P que herdou essa deficiência da própria UML.

Essa deficiência foi tratada parcialmente em GOMES (2008), que formalizou a interpretação completa do diagrama de classes da UML.P, utilizando a lógica não-monotônica F-Logic que pode ser aplicada a objetos e será revisada na próxima seção.

2.4. Lógica desenvolvida para Classes e Objetos

Este trabalho utilizará a lógica não-monotônica chamada de *Frame Logic* ou *F-Logic*, porque se pretende fornecer base para a interpretação da UML.P em PDDL através dessa lógica. Lembrando que isso se deve ao fato de ambas serem semiformais. Vale ressaltar para esse fim que existem trabalhos em (RAMALHO; ROBIN, 2004) e em (RAMALHO; ROBIN; SCHIEL, 2004) que já apresentaram o diagrama de classes da UML sendo interpretado em F-Logic. Esses trabalhos motivaram a Interpretação Completa criada em (GOMES, 2008). Além disso, a F-Logic torna possível o uso futuro da UML.P com o raciocínio orientado para um sistema baseado em conhecimento (KBS).

A F-logic (KIFER; LAUSEN, WU, 1995) ou *Frame Logic* é um formalismo aplicado a linguagens orientadas a objeto ou baseadas em Quadros (*frames)*. O nome *Frame Logic* vem do trabalho de Marvin Minsky, em 1975, na área de IA, apresentado nesse trabalho na seção 2.3.2 sobre Quadros. Entretanto, ao invés de usar a terminologia usada nos Quadros (*frames*), os autores adotaram a terminologia da orientação a objeto.

A F-Logic é uma lógica de primeira ordem orientada a objeto (KIFER; LAUSEN, WU, 1995), acrescentando à lógica de predicados, as propriedades de objetos com estrutura interna complexa, hierarquias de classe e herança, tipos e encapsulamento.

A formalização da F-Logic visto em (KIFER; LAUSEN, WU, 1995) criou a possibilidade do cálculo de predicados ser analisado como um caso especial da F-Logic e com algum esforço pode-se adaptar a teoria da lógica clássica para a F-Logic, permitindo compatibilidade com muitos sistemas existentes.

Para mostrar o uso da F-Logic, será modelada abaixo uma classe em UML.P chamada Bloco, que possui dois atributos:

- nadaSobre – é um atributo que significa que não há nada em cima do bloco;
- sobre – é um relacionamento de associação que significa que um bloco pode estar sobre outro bloco.

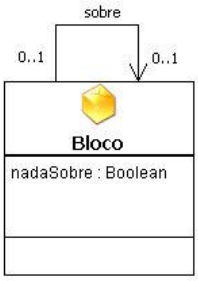


Figura 17 – Uma Classe em UML.P.

Na sintaxe da F-Logic a representação da classe acima deve ficar assim como segue:

Bloco[nadaSobre@=> Boolean; sobre@=> Bloco].

O símbolo "=>" significa que se trata da assinatura de método escalar para um atributo ou associação da UML.P. O símbolo "=>" será apresentado em detalhes no capítulo 3. Nesse caso, trata-se das regras sobre como um bloco é analisado no domínio mundo dos blocos.

E para criar, por exemplo, uma situação baseada nessa classe e em suas propriedades (atributos e associações), instancia-se objetos em UML.P assim como segue:

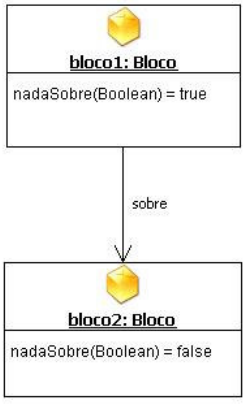


Figura 18 – Situação usando diagrama de objetos.

Na sintaxe da F-Logic, a representação dos objetos acima deve ficar assim como segue:

bloco1:Bloco[nadaSobre@->true, sobre@->bloco2].

bloco2:Bloco[nadaSobre@->false].

O símbolo "->" acima é o método escalar de dados para os objetos UML.P utilizando regras e representando fatos. O símbolo "->" será apresentado em detalhes no capítulo 3.

Ainda no capítulo 3, tanto a sintaxe da F-Logic quanto sua semântica correta e completa serão detalhadas, de modo a compreender a interpretação completa da UML.P em F-Logic realizada em (GOMES, 2008), além da criação da interpretação correspondente da PDDL em F-Logic e *vice versa*, tal como pode ser visto na seguinte figura:

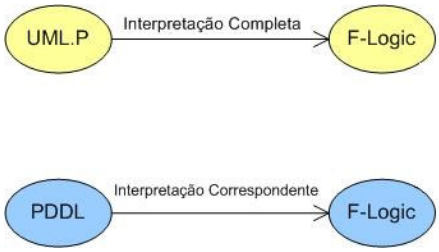


Figura 19 – Mapeamentos das sintaxes UML.P e PDDL em F-Logic.

Com a figura acima, pretende-se realizar a mesma estratégia de mapeamento utilizada na interpretação completa (GOMES, 2008) para a sintaxe da PDDL em F-Logic e *vice versa*, chamada Interpretação Correspondente simbolizada por $\overset{I}{\longleftrightarrow}$. A partir dessa estratégia das interpretações completa e correspondente, será demonstrada a equivalência dos modelos UML.P e PDDL.

Na próxima seção, haverá apresentações de ferramentas de engenharia do conhecimento. Uma dessas ferramentas é o itSIMPLE, que vem sendo apresentada como uma ferramenta que satisfaz os anseios da comunidade de Planejamento Automático considerando os processos de Engenharia do Conhecimento.

2.5. Ferramentas de Engenharia do Conhecimento

Desde o ICKEPS'05, competição de EC para Planejamento Automático, a comunidade de Planejamento Automático conheceu três ferramentas para modelagem de domínios e problemas de planejamento: o ModPlan, o GIPO e o itSIMPLE.

O **ModPlan** (EDELKAMP; MEHLER, 2005) é uma ferramenta de engenharia do conhecimento, onde é possível incluir os arquivos de domínio e de problema em formato

PDDL. Com o ModPlan também é possível escolher uma técnica de Planejamento Automático cujo Planejador será escolhido para tratar o problema; nele há algoritmos de aprendizagem, visualização de planos, planejadores embarcados e validação de domínios e problemas.

O **GIPO** (SIMPSON et al, 2001) é uma ferramenta que possui como representação do conhecimento a linguagem OCL_h, que significa *Object-Centred Language*, com o objetivo de aprimorar a aquisição de conhecimento e o processo de validação, além de melhorar o processo de geração dos planos. O GIPO significa *Graphical Interface for Planning with Objects* e foi construído para que não fosse necessário um especialista em Planejamento Automático, ou seja, para que pudesse ser manuseada por um usuário comum; todavia, ainda possui uma interface complexa para esse objetivo. O GIPO também é uma ferramenta para ser usada em um processo de Engenharia do Conhecimento para Planejamento Automático.

O **itSIMPLE** (VAQUERO et al. 2005), *Integrated Tool Software Interface for Modeling Planning Environments,* é uma ferramenta para o processo de Engenharia do Conhecimento cuja modelagem é realizada usando a UML.P. Atualmente, o itSIMPLE possui planejadores em linux e windows embarcados para geração de plano e, também, há a possibilidade de avaliar as mudanças de estado no domínio com a navegação dentro das ações do plano, além de avaliar os gráficos de uso dos recursos do domínio em um dado problema. Para os diagramas de máquina de estado, pode-se fazer a análise dos resultados usando Redes de Petri.

As técnicas, métodos, práticas da EC estão pouco-a-pouco fazendo parte da área de Planejamento, apesar de ainda estar no início; o que demonstra muita pesquisa a ser realizada pela comunidade (MCCLUSKEY, 2003).

Nesta seção foram apresentadas três ferramentas da EC para Planejamento Automático, todavia, faz-se necessário apresentar outra ferramenta que não foi construída para a área de Planejamento Automático, mas como uma ferramenta de programação do conhecimento de propósito geral. Essa ferramenta é o **FLORA-2,** que é baseada no compilador XSB e possui como lógicas a *Frame Logic* e a *Transaction Logic* (LUDÄSCHER; YANG; KIFER, 1999)**.** O interessante sobre a *Transaction Logic*, de acordo com (KIFER; LAUSEN; WU, 1995), é que ela pode ser estendida de modo que seu Oráculo de dados utilize fórmulas bem formadas em F-Logic, passando a ser chamada de *Transaction F-Logic* por (KIFER; LAUSEN; WU, 1995). Isso demonstra a possibilidade de usar o raciocínio dinâmico dos objetos e, para a UML.P, interpretar seu diagrama de máquina de estados.

Para esclarecer melhor como o **FLORA-2** funciona, abaixo há uma apresentação de sua interface, onde:

- Há uma definição de classes, regras e objetos;
- No caso abaixo, a regra é:
  - ?X:Bloco[nadaSobre->false] :- ?Y:Bloco[nadaSobre->true, sobre->?X].

  Se o bloco Y não tem nada sobre ele e ele está sobre outro bloco X, **então**, o bloco X tem outro bloco sobre ele;
- O bloco 1 não tem nada sobre ele e ele está sobre o bloco 2;
- Foi perguntado à Base de Conhecimento (BC) do Flora: o bloco 2 não tem nada sobre ele, tem? O Flora responde: tem. Isso pode ser visto na tela como:
  - bloco2:Bloco[nadaSobre->true].

    No.

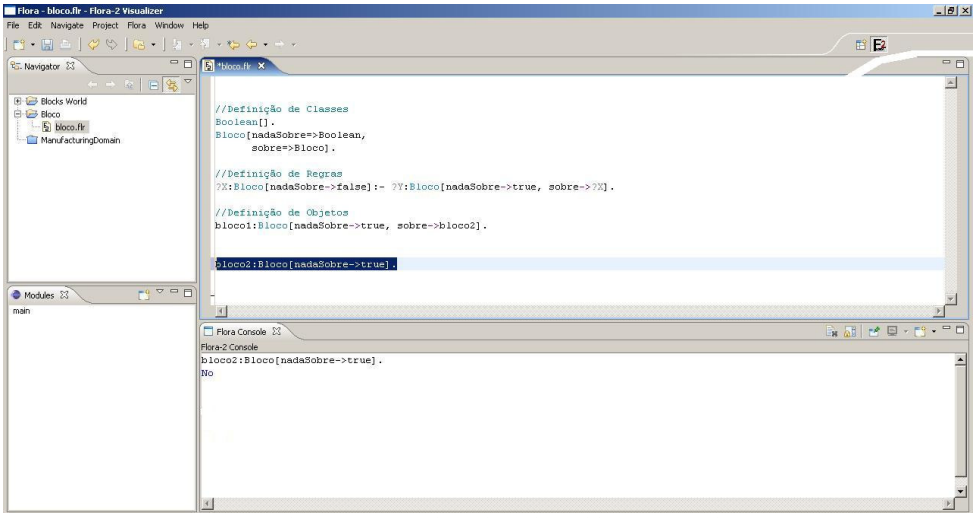


Figura 20 – Raciocínio sobre objetos com Flora-2.

Na **figura 20** segue a descrição em F-Logic:

```
//Definição de Classes
Boolean[].
Bloco[nadaSobre=>Boolean,
     sobre=>Bloco].
//Definição de Regras
?X:Bloco[nadaSobre->false]:-?Y:Bloco[nadaSobre->true,sobre->?X].
//Definição de Objetos
bloco1:Bloco[nadaSobre->true, sobre->bloco2].
//Consulta à BC
bloco2:Bloco[nadaSobre->false].
```

O Flora-2 também possui um visualizador das classes e objetos criados assim como seus atributos e valores:

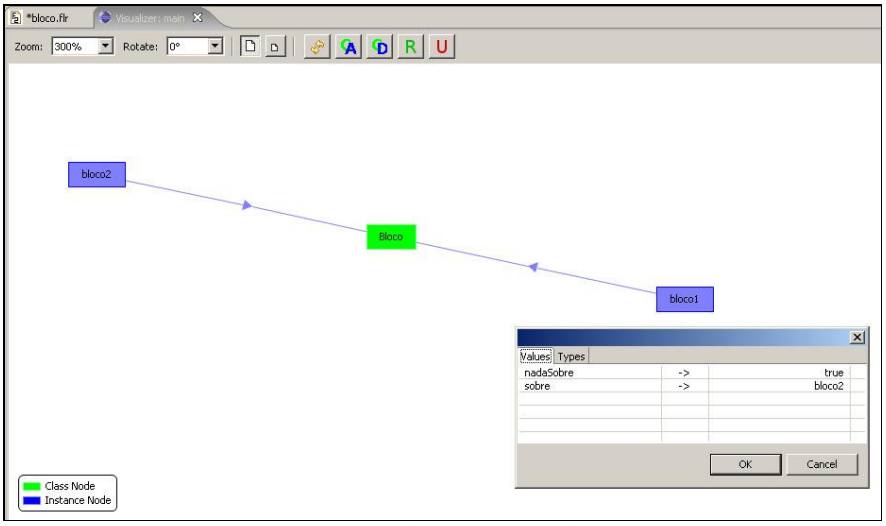


Figura 21 – Visualizador do Flora.

Visualizadores como o da **figura 21** facilita manter a modelagem de domínio e de problemas para o projetista de domínio ou engenheiro do conhecimento. Além disso, o uso de BCs e interfaces intuitivas podem ser o caminho para levar o planejamento automático ao uso comercial.

De acordo com McCluskey (2000), existe a necessidade de um *KBS* (*Knowledge Base System*) para conhecimento sobre ações que suporte a reusabilidade de modelagem para domínios de Planejamento e o **FLORA-2** pode auxiliar nessa necessidade. Para este trabalho, o **FLORA-2** foi apresentado como uma ferramenta onde a F-Logic pode ser usada como um possível *KBS* para os domínios e problemas de Planejamento Automático modelados pelo **itSIMPLE**.

2.6. Engenharia do Conhecimento e o Planejamento Automático

A Engenharia do Conhecimento (SIMPSON et al, 2001) para Planejamento Automático é o processo que lida com a aquisição, validação e manutenção dos modelos de domínio de planejamento, além da seleção e otimização apropriada dos recursos para planejamento. Assim, são os processos da engenharia do conhecimento que suportam os processos de planejamento. Para esses processos, há uma proposta de McCluskey de um modelo que é um ambiente para a engenharia do conhecimento (MCCLUSKEY, 2000) destinado ao Planejamento Automático:

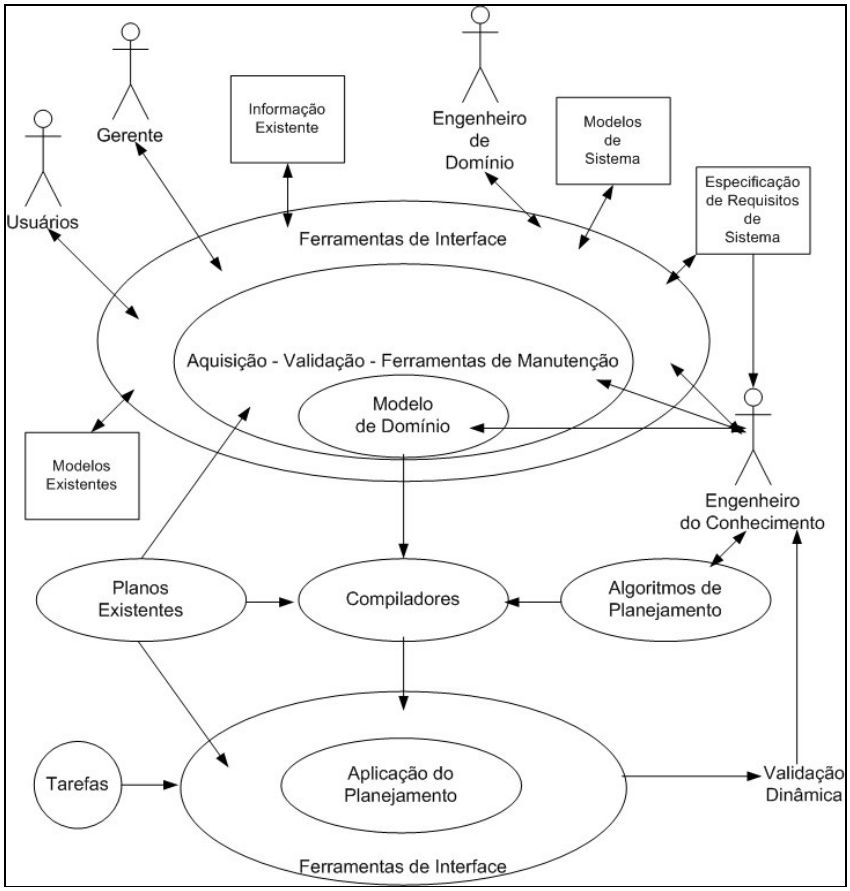


Figura 22 – Ambiente de Engenharia do Conhecimento (MCCLUSKEY, 2000).

Esse Modelo de Ambiente da Engenharia do Conhecimento para o Planejamento Automático mostra que usuários e gerentes de domínio possam ter ferramentas de interface amigáveis com informações de Planejamento sem qualquer necessidade de entender a inerência do Planejamento Automático. Já o engenheiro do conhecimento tem a responsabilidade sobre os níveis desse modelo: manter as ferramentas de interface, manter as ferramentas de manutenção (aquisição e validação de conhecimento), criar e manter os modelos de domínio. Para o nível da aplicação de Planejamento Automático, o engenheiro do conhecimento deve manter os planos existentes, prover os algoritmos de planejamento para a criação de novos planos de acordo com as tarefas do domínio em questão (MCCLUSKEY, 2000).

McCluskey propôs esse ambiente de suporte para a Engenharia do Conhecimento e lista algumas ações para a comunidade de Planejamento. Uma das ações seria a realização de pesquisas sobre ferramentas visuais que possam ajudar na modelagem de todo o modelo de domínio tais como: objetos, estados, hierarquias de objetos, assim como ações. McCluskey

também compôs ações para avaliar os métodos da engenharia usada para o Planejamento e também sugere que as linguagens de modelagem de domínio sejam estruturadas, além de uma BC de Planejamento ampla. McCluskey propôs ações, também, no que tange à semântica das linguagens utilizadas em planejamento, assim como a PDDL, para que haja um processo forte de padronização para dar suporte a qualquer informação necessária e relacionada ao Planejamento Automático.

Sendo assim, essa visão sobre a Engenharia do Conhecimento para Planejamento Automático indica o porquê dessas ferramentas de EC como Gipo e itSIMPLE e suas linguagens visuais serem aceitas na comunidade. Além disso, a motivação de um dia possuir *KBS* para a área de Planejamento Automático (MCCLUSKEY, 2000) implica formalização dos domínios e problemas. Portanto, o objetivo desse trabalho que visa demonstrar a formalização da interpretação do diagrama de classes da UML.P em tipos e predicados da PDDL é parte do caminho para um dia existir um *KBS* para a área de Planejamento Automático.

CAPÍTULO 3

3. A F-LOGIC

Essa dissertação tem como objetivo demonstrar que modelos UML.P provenientes do diagrama de classes podem ser interpretados em modelos PDDL, especificamente, tipos e predicados. Contudo, devido às duas linguagens, UML.P e PDDL, serem semiformais, utilizou-se a F-Logic, que é uma lógica correta e completa, para demonstrar como interpretar os elementos dos modelos UML.P em elementos dos modelos PDDL. A Interpretação Completa já foi criada em (GOMES, 2008), e ela faz o mapeamento da sintaxe visual do diagrama de classes da UML.P na F-Linguagem da F-Logic. Assim, esse trabalho também deve utilizar-se da mesma estratégia: mapear a sintaxe da PDDL na F-Linguagem da F-Logic. Portanto, esse capítulo deve detalhar as definições necessárias para compreender as Interpretações Completa e Correspondente e, também, sobre a equivalência dos modelos UML.P e PDDL a partir da teoria da F-Logic.

Dessa maneira, a partir da próxima seção, devem ser apresentadas as propriedades da F-Logic (KIFER, LAUSEN, WU, 1995).

3.1. Sintaxe e Semântica da F-Logic

A F-Logic é uma lógica de primeira ordem para objetos de acordo com (KIFER, LAUSEN, WU, 1995), pois possui propriedades claramente atreladas ao objeto, tais como, classes e métodos. Dadas essas considerações, para que a sua sintaxe tenha como demonstrar as propriedades atreladas ao objeto, em (KIFER, LAUSEN, WU, 1995) há uma definição para F-Moléculas que trata dessa questão e ela será apresentada tal como segue:

Definição 3.1: Uma molécula em F-Logic (F-Molécula) é dada pelas seguintes afirmações (KIFER, LAUSEN, WU, 1995):

1. Uma definição *IS-A* é dada na forma `C::D` ou na forma `O:C`, onde `C`, `D` e `O` são *id-termos*;

2. Uma molécula de objeto é dada na forma de `O[`uma lista separada por '**;**' de expressões de métodos`]`. Uma expressão de método pode ser:

a. Expressão de dados não herdável:

i. Expressão escalar ($k \geq 0$):

```
ScalM@Q1,...,Qk -> T
```

ii. Expressão de conjunto de valores ($l, m \geq 0$):

```
SetM@R1,...,Rl ->> {S1,...,Sm}
```

b. Expressão de dados escalares e de conjunto de valores herdáveis. Essas expressões são como as expressões de dados não herdáveis exceto que os símbolos `->` e `->>` são substituídos por `*->` e `*->>` respectivamente.

c. Expressão de Assinatura:

i. Assinatura escalar ($n, r, \geq 0$):

```
ScalM@V1,...,Vn => {A1,...,Ar}
```

ii. Assinatura de conjunto de valores ($s, t, \geq 0$).

```
SetM@W1,...,Ws =>> {B1,...,Bt}
```

A definição acima mostra como é a sintaxe da f-molécula e, a partir dela, há como representar frames e objetos. Para esclarecer essa definição, no **item 1**, os ***id-termos*** são identificadores de objetos e a f-molécula *IS-A* `C::D` significa que "C é um tipo de D" e `O:C` significa que "O é um elemento de C".

Os **itens 2a e 2b** não serão utilizados nesse trabalho porque o **item 2a** lida com a descrição de fatos em F-Logic e o **item 2b** lida com a descrição de fatos que foram herdados. Como esse trabalho lida apenas com a descrição de domínios de Planejamento Automático, então, apenas deve-se utilizar a descrição em F-Logic que lida com as f-moléculas *IS-A* e de assinaturas de métodos, que são os **itens 1 e 2c** respectivamente.

Entretanto, para um entendimento completo da definição, o **item 2a**, **ScalM** é um método que possui como retorno uma tupla de objetos e **SetM** é um método que possui uma ou mais tuplas de objetos. Por exemplo, a f-molécula `O[ScalM@Q1,...,Qk -> T]` significa que **O** é o id-termo objeto, **Q1, ..., Qk** são objetos como parâmetros e **T** é também um objeto. O símbolo "`@`" da F-Logic separa a descrição de método **ScalM** ou **SetM** de seus parâmetros e o símbolo "`->`" da F-Logic representa que deve existir apenas uma tupla de objeto de determinado tipo: `T`. Dessa maneira, essa f-molécula pode ser representada tal como segue: **Robo1[segurando@BlocoA, Mesa1->True]**. Já o método `O[SetM@R1,...,Rl`

`–>>{S1,...,Sm}]` possui definição de f-molécula semelhante a do **ScalM** descrita acima, exceto o símbolo "–>>" da F-Logic, que representa a existência de uma ou mais tupla de objetos para cada tipo representado: `{S1,...,Sm}`. Dessa maneira, esse método pode ser representado como **Caneta[tem@->>{Tampa, Refil, Corpo}].**

Já o **item 2b** lida com a descrição de fatos que são herdados das assinaturas do **item 1** e **2c**, por exemplo:

- a f-molécula do **item 1** pode ser Homem :: Pessoa, "Homem é do tipo Pessoa";
- a f-molécula do **item 2c** pode ter os métodos Pessoa[temRG@=>String; temCPF@=>String];
- Para utilizar a f-molécula do **item 2b**, sobre herança de métodos, a classe Homem herda as assinaturas dos métodos **temRG** e **temCPF** da classe Pessoa da seguinte maneira: Marcelo : Homem[temRG@*–>24144590-14; temCPF@*–>370021198-01]. Lembrando que "Marcelo : Homem" significa que Marcelo é um elemento da classe Homem;

As f-moléculas do **item 2c** lidam com a descrição de assinaturas de métodos da F-Logic como apresentado acima. Essas f-moléculas definem a tipagem dos programas em F-Logic, que é a condição para que as f-moléculas dos **itens 2a e 2b** possam ser descritas.

Dessa maneira, usando o mesmo exemplo, a f-molécula de assinatura de métodos do **item 2c**, **Robo[segurando@Bloco; Mesa => Booleano]** possibilita a existência da f-molécula de descrição de fatos **Robo1[segurando@BlocoA, Mesa1->True]** para que seja satisfazível e, da mesma maneira, a f-molécula de assinatura de métodos **Produto[tem@=>>{Atributo}]** possibilita a existência da f-molécula de descrição de fatos **Caneta[tem@->>{Tampa, Refil, Corpo}]** para que também seja satisfazível.

Nessa dissertação apenas se levará em consideração as expressões de métodos de assinatura (**definição 3.1 item 2c**) para um valor, que é a assinatura escalar **ScalM**, e para vários valores, que é a assinatura de conjunto de valores **SetM**. Essas expressões de métodos de assinatura escalar e de conjunto de valores são as utilizadas em F-Logic para lidar com classes, que é parte do foco dessa dissertação.

Com a F-Mólecula descreve-se os programas na F-Linguagem, ou seja, a sintaxe da F-Logic. Nas subseções a seguir, devem ser apresentadas propriedades da F-Logic como a F-estrutura, que é a semântica da F-Logic.

3.1.1. Propriedades da F-Logic

Em (KIFER, LAUSEN, WU, 1995) apresenta-se fórmulas complexas e sobre como descrevê-las em F-Logic, chamadas de F-Fórmulas. Algumas diretrizes sobre essas fórmulas estão como segue:

i) F-Moléculas são F-Fórmulas;

ii) $\varphi \wedge \psi$, $\varphi \vee \psi$, $\neg\varphi$ são F-Fórmulas se φ e ψ também forem;

iii) um literal é uma fórmula molecular ou sua negação;

iv) o conectivo de implicação "←" com o mesmo significado da Lógica Clássica, mas diferente em termos de sentido da seta para não ser confundido com a definição 3.1 item 2a;

v) usa-se os demais símbolos iguais ao da Lógica Clássica exceto o símbolo implicação descrito em iv);

Dentro de uma f-molécula podem existir vários métodos que juntos de sua classe são considerados f-átomos. Em (KIFER, LAUSEN, WU, 1995), existe também a explicação dos f-átomos, tal como segue:

A f-molécula **X[ScalM@=>A; SetM@=>>B]** é equivalente à conjunção de seus f-átomos **X[ScalM@=>A]** $\wedge$ **X[SetM@=>>B]**.

Os f-átomos apresentados acima são importantes para se entender que uma f-molécula muitas vezes não é atômica como nas f-moléculas *IS-A*, apresentadas na definição 3.1 item 1. Por isso, f-moléculas podem ser decompostas em um conjunto de f-átomos.

Portanto, nesse trabalho será utilizado o termo **f-átomo**, que é a parte atômica da F-Linguagem. Os f-átomos serão bastante utilizados no capítulo 4 sobre a Interpretação Correspondente, no capítulo 5 sobre a Interpretação *IFR* de forma reduzida e no capítulo 6 sobre equivalência e interpretação dos modelos UML.P e PDDL.

Outra propriedade essencial da F-Logic é a sua semântica chamada de F-estrutura, que é um mapeamento contendo estruturas semânticas de acordo com sua tupla $\boldsymbol{\mathcal{I}} = \langle U, \prec_U, \in_U, \mathcal{I}_{\mathcal{F}}, \mathcal{I}_{\rightarrow}, \mathcal{I}_{->>}, \mathcal{I}_{*\rightarrow}, \mathcal{I}_{*->>}, \mathcal{I}_{=>}, \mathcal{I}_{=>>} \rangle$, onde cada elemento dessa tupla possui um mapeamento com sua sintaxe, também conhecida como *F-Linguagem* (KIFER; LAUSEN; WU, 1995). A saber, as estruturas semânticas são:

– U é o domínio de $\boldsymbol{\mathcal{I}}$;

– $\prec_U$ é o mapeamento que denota a herança de classes em F-Logic;

– $\in_U$ é o mapeamento que denota quando um objeto é membro de uma classe;
– $\mathcal{I}_{\mathcal{F}}$ é um mapeamento dos programas em F-Logic;
– $\mathcal{I}_{\rightarrow}$ e $\mathcal{I}_{->>}$ são os mapeamentos de métodos escalar e de conjunto de valores não herdáveis (objetos);
– $\mathcal{I}_{*\rightarrow}$ e $\mathcal{I}_{*->>}$ são os mapeamentos de métodos escalar e de conjunto de valores ambos herdáveis (objetos);
– $\mathcal{I}_{=>}$ e $\mathcal{I}_{=>>}$ são os mapeamentos de métodos escalar e de conjunto de valores de assinatura (Classes).

Apenas os itens da tupla $\mathcal{I}=\langle U, \prec_U, \mathcal{I}_{=>}, \mathcal{I}_{=>>}\rangle$ fazem parte dessa dissertação no que tange a interpretar o diagrama de classes na UML.P e os tipos e predicados da PDDL. Para relembrar, o símbolo $\prec_U$ é o mapeamento da sintaxe da f-molécula descrita na definição 3.1 item 1 e os símbolos $\mathcal{I}_{=>}$ e $\mathcal{I}_{=>>}$ são os mapeamentos da sintaxe descrita na definição 3.1 item 2c.

Com a explicação da F-estrutura da F-Logic, apresenta-se como a definição a seguir utiliza a semântica para demonstrar a satisfação das F-Moléculas (KIFER; LAUSEN; WU, 1995):

Definição 3.2 (Satisfação das F-Moléculas) Seja $\mathcal{I}$ uma F-estrutura e G uma F-Molécula. Escreve-se $\mathcal{I} \models_{\nu} G$ se e somente se as regras abaixo se mantêm:

1. Quando uma F-Molécula G é uma asserção *IS-A* então:
   i) $\nu(Q) \preceq_U \nu(P)$, se G = Q :: P; ou
      $\nu(Q) \in_U \nu(P)$, se G = Q : P
   ii) Quando uma F-Molécula G é uma molécula de objeto da forma O[a ';' – lista separada de expressões de método], então para toda expressão de método E em G, as seguintes condições devem-se manter:
      a) Se E é um método escalar não herdável na forma $ScalM@Q_1, \ldots, Q_k \rightarrow T$, o elemento $\mathcal{I}^{(k)}_{\rightarrow}(\nu(ScalM))(\nu(O), \nu(Q_1), \ldots, \nu(Q_k))$ deve ser definido e igual a $\nu(T)$. As mesmas condições devem ser mantidas para o caso de E ser uma expressão escalar herdável, apenas é necessário mudar o $\mathcal{I}^{(k)}_{\rightarrow}$ por $\mathcal{I}^{(k)}_{*\rightarrow}$.
      b) Se E é um método de conjunto de valores não herdável na forma $SetM@R_1, \ldots, R_l \text{ ->> } \{S_1, \ldots, S_m\}$, o conjunto $\mathcal{I}^{(l)}_{->>}(\nu(SetM))(\nu(O), \nu(R_1), \ldots, \nu(R_l))$ deve ser definido e conter o conjunto $\{\nu(S_1), \ldots, \nu(S_m)\}$. As mesmas condições

devem ser mantidas para o caso de E ser uma expressão de vários valores herdável, apenas é necessário mudar o $\mathcal{I}^{(k)}_{->>}$ por $\mathcal{I}^{(k)}_{*->>}$.

c) Se E é um método escalar de assinatura na forma ScalM@Q_1, ... , Q_n => {R_1, ... , R_u}, então o conjunto $\mathcal{I}^{(n)}_{=>}(\nu(\text{ScalM}))(\nu(O), \nu(Q_1), \ldots, \nu(Q_n))$ deve ser definido e conter o conjunto $\{\nu(R_1), \ldots, \nu(R_u)\}$.

d) Se E é um método de conjunto de valores de assinatura na forma SetM@V_1, ... , V_s =>> {W_1, ... , W_v}, então o conjunto $\mathcal{I}^{(s)}_{=>>}(\nu(\text{SetM}))(\nu(O), \nu(V_1), \ldots, \nu(V_s))$ deve ser definido e conter o conjunto $\{\nu(W_1), \ldots, \nu(W_v)\}$**".**

De acordo com (KIFER; LAUSEN; WU, 1995) as condições da definição 3.2 denotam o seguinte:

- A condição (i) apresenta ν(Q) como uma subclasse de ν(P) na forma G = Q :: P, ou ν(Q) como um membro da classe ν(P) na forma G = Q : P. Como exemplo, pode-se citar o domínio Logística onde a classe Caminhão é uma subclasse da classe Veículo, ou um objeto caminhão Mercedes Benz que é membro da classe Caminhão;
- As condições (a) e (b) denotam que a função de interpretação, para instâncias, deve ser definida com os tipos apropriados e permitir resultados compatíveis com esses tipos. Esses tipos compatíveis fazem parte das assinaturas de métodos nas condições (c) e (d);
- As condições (c) e (d) denotam que o tipo do método ScalM ou SetM devem ser especificados pela expressão com os tipos associados a este método pela função $\mathcal{I}$.

Nesta dissertação, a satisfação das f-moléculas se dará pelas condições i), c) e d). Isso porque se levará em consideração a asserção *IS-A* e as expressões de método para assinatura, que são as condições que descrevem o domínio de Planejamento Automático. Já as outras condições, seguem o especificado nas condições i), c) e d) para os problemas de Planejamento Automático.

Na F-Logic há importantes propriedades: a asserção *IS-A,* a Igualdade e as Expressões de Assinaturas. Assim será apresentada cada uma delas de acordo com (KIFER; LAUSEN; WU, 1995).

Propriedades do Relacionamento *IS-A*

a) Reflexibilidade *IS-A*

$\mathbf{I} \models p :: p$ significa que *p é subclasse dele mesmo e isso decorre logicamente da F-estrutura* **I**;

b) Transitividade *IS-A*

Se $\mathbf{I} \models p :: q$ e $\mathbf{I} \models q :: r$, então, $\mathbf{I} \models p :: r$

Se *p* é subclasse de *q* e *q* é subclasse de *r*, ambos decorrendo logicamente da F-estrutura **I**, então, *p* é também subclasse de *r*, decorrendo logicamente da mesma F-estrutura **I**.

c) Relacionamento Acíclico *IS-A*

Se $\mathbf{I} \models p :: q$ e $\mathbf{I} \models q :: p$, então, $\mathbf{I} \models p \doteq q$

Se *p* é subclasse de *q* e *q* é também subclasse de *p*, ambos decorrendo logicamente da F-estrutura **I**, então, *p* é igual a *q* como conseqüência lógica da F-estrutura **I**.

d) Inclusão de Subclasse

Se $\mathbf{I} \models p : q$ e $\mathbf{I} \models q :: r$, então, $\mathbf{I} \models p : r$

Se *p* é membro da classe *q* e *r* é superclasse de *q*, ambos decorrendo logicamente da F-estrutura **I**, então, *p* é também membro da classe *r* como conseqüência lógica da F-estrutura **I**.

As propriedades do Relacionamento *IS-A* utilizadas nessa dissertação são aquelas citadas nos itens a), b) e c), que serão utilizadas nas provas dos teoremas no capítulo 6. Já o item d) lida com a descrição de fatos em F-Logic, como já explicado, trata-se de problemas de Planejamento Automático e está fora do escopo deste trabalho que lida apenas com os domínios de Planejamento Automático.

Outras propriedades importantes na F-Logic são aquelas que determinam como representar a igualdade:

Propriedades da Igualdade (*Equality)*

Os lemas abaixo servem para expressar as propriedades usuais da igualdade.

a) Propriedade Reflexiva

Para todo $p \in U(\mathcal{F})$, $\mathbf{I} \models p \doteq p$;

b) Propriedade Simétrica

Se $\mathbf{I} \models p \doteq q$, *então,* $\mathbf{I} \models q \doteq p$;

c) Propriedade Transitiva

Se $\mathbf{I} \models p \doteq q$ *e* $\mathbf{I} \models q \doteq r$, *então,* $\mathbf{I} \models p \doteq r$.

As propriedades da Igualdade na F-Logic permitem que se utilize inferências para tratar os programas em F-Logic que lidam com igualdade. Para finalizar, existem outras propriedades, também importantes, chamadas de expressões de assinaturas:

Propriedades das Expressões de Assinatura

O símbolo utilizado abaixo $\approx>$ denota tanto => quanto =>>. O símbolo *s* significa uma classe em F-Logic, o *r* e *p* denotam também classes em F-Logic, $q_1, ..., q_k$ são parâmetros de classes em F-Logic e *mthd* denota tanto um método escalar quanto um método de conjunto de valores. Dessa maneira, em F-Logic pode-se ter as seguintes regras de inferência:

a) Herança de Tipos

Se $\mathbf{I} \vDash p[mthd@q_1, ..., q_k \approx> s]$ e $\mathbf{I} \vDash r :: p$, então, $\mathbf{I} \vDash r[mthd@q_1, ..., q_k \approx> s]$;

Se $p[mthd@q_1, ..., q_k \approx> s]$ é conseqüência lógica de **I** e $r :: p$ é também conseqüência lógica de **I**, então, deduz-se que $r[mthd@q_1, ..., q_k \approx> s]$ é conseqüência lógica de **I**.

b) Restrição de *Input*

Se $\mathbf{I} \vDash p[mthd@q_1, ..., q_i, ..., q_k \approx> s]$ e $\mathbf{I} \vDash q_i' :: q_i$, então, $\mathbf{I} \vDash r[mthd@q_1, ... , q_i', ..., q_k \approx> s]$;

Se $p[mthd@q_1, ..., q_i, ..., q_k \approx> s]$ é conseqüência lógica de **I** e $q_i' :: q_i$ é também conseqüência lógica de **I**, então, deduz-se que $r[mthd@q_1, ... , q_i', ..., q_k \approx> s]$ é conseqüência lógica de **I**.

c) *Output* Relaxado

Se $\mathbf{I} \vDash p[mthd@q_1, ..., q_k \approx> r]$ e $\mathbf{I} \vDash r :: s$, então, $\mathbf{I} \vDash r[mthd@q_1, ..., q_k \approx> s]$;

Se $p[mthd@q_1, ..., q_k \approx> r]$ é conseqüência lógica de **I** e $r :: s$ é também conseqüência lógica de **I**, então, deduz-se que $r[mthd@q_1, ..., q_k \approx> s]$ é conseqüência lógica de **I**.

Com as propriedades supracitadas, é possível elaborar programas bem tipados em F-Logic. Para ilustrar essa elaboração de F-Programas, considere o exemplo dado em (KIFER; LAUSEN; WU, 1995):

empregado :: pessoa	*pessoa[nome@=>texto]*
assistente :: estudante	*estudante[bebe@=>>cerveja; mantém@=>acordo]*
assistente :: empregado	*empregado[salário@=>inteiro; mantém@=>carro]*
	assistente[mantém@=>antigüidade]

O exemplo acima mostra que a classe assistente acumula os seguintes f-átomos em sua f-molécula:

assistente[nome@=>texto; bebe@=>cerveja; mantém@=>(acordo, carro, antigüidade)]

O método da classe **assistente** é *mantém@=>antigüidade*, mas possui também como herança duas assinaturas da classe **estudante** e **empregado**.

Como pode ser visto nesse exemplo, a F-Logic possui tanto sintaxe como semântica robusta para tratar os modelos de engenharia do conhecimento baseada em Objetos.

As interpretações propostas nessa dissertação para a UML.P e para a PDDL levam em consideração essas propriedades da F-Logic, apesar de não citá-las diretamente nas definições. Ou seja, respeitam-se essas propriedades para que se tenha F-Fórmulas bem tipadas.

O próximo capítulo, utilizará a F-Logic para interpretar tipos e predicados da PDDL e *vice versa.*

CAPÍTULO 4

4. INTERPRETAÇÃO DA PDDL EM F-LOGIC

A PDDL é utilizada para descrever o conhecimento sobre domínios de planejamento. Entretanto, a PDDL é uma linguagem semiformal. Como fora explicado no capítulo 3, a F-Logic será utilizada para provar que a UML.P pode ser descrita em PDDL devido ao fato de ela ser uma lógica formal correta e completa. Isso garante uma forma de se verificar a descrição dos domínios caso haja contradições.

Nesse capítulo haverá definições para a Interpretação Correspondente $\overset{I}{\longleftrightarrow}$ de tipos e predicados da descrição PDDL em classes e métodos da descrição F-Logic da figura abaixo:

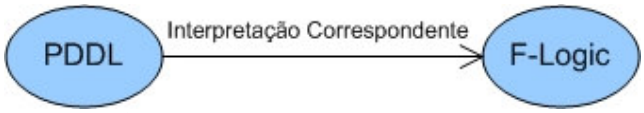


Figura 23 – Interpretação Correspondente.

Essa estratégia de interpretação já foi provada em (KIFER; LAUSEN; WU, 1995) a partir do **Teorema 18.1**, que demonstra os mapeamentos entre a F-Logic e a Lógica de Predicados, onde esse teorema demonstra que é factível a interpretação de um para outro e *vice versa.* Em (DECKER, 1998), há também um trabalho sobre como representar a F-Logic em Lógica de Predicados (o símbolo "⇒" na **tabela 4** significa "interpretado em") a partir de uma tabela chamada de esquema de tradução para a F-Logic:

Tabela 4 – Tradução de Modelos

Orientação a Objeto		Lógica de Predicados
C::D	⇒	Sub(C, D)
O:C	⇒	instance(O, C)
O[M->>V]	⇒	Method(O,M,V)
C[M=>>D]	⇒	methodtype(C,M,D)
O:C[M->>V:D]	⇒	instance(O, C) ∧ method(O,M,V) ∧ instance(V, D)

Fonte: DECKER, 1998

A tabela acima possui as seguintes traduções de F-Logic em Lógica de Predicados:

- C::D é um f-átomo e significa que C é subclasse de D que pode ser traduzido na Lógica de Predicados através do predicado Sub(C, D), que significa que C é subclasse de D;
- O::C é um f-átomo e significa que O é uma instância (membro) de C que pode ser traduzido na Lógica de Predicados através do predicado instance(O, C), que significa que C é instância de D;
- O[M->>V] é um f-átomo e significa que O possui o método de dados M com retorno V que pode ser traduzido na Lógica de Predicados através do predicado Method(O,M,V), que significa que o predicado Method está definido para O com o nome do método M e o valor V;
- C[M=>>D] é um f-átomo e significa que C possui o método de assinatura M com retorno D que pode ser traduzido na Lógica de Predicados através do predicado methodtype(C,M,D), que significa que o predicado methodtype está definido para C com o nome do tipo do método M e o tipo D;
- Em F-Logic, os f-átomos O:C[M->>V:D] necessitam da seguinte tradução em Lógica de Predicados: instance(O, C) & method(O,M,V) & instance(V, D).

Ainda na tabela acima, juntamente com o trabalho de (DECKER, 1998), discute-se a representação de conhecimento para domínios específicos onde se demonstra um esquema de tradução para a F-Logic desde a Lógica de Predicados. Direcionamento semelhante já foi fornecido em (KIFER; LAUSEN; WU, 1995) onde se apresenta a F-Logic como uma adaptação da Lógica de Predicados para objetos, chamando-a de Lógica de Primeira Ordem para Objetos. Para essa dissertação, a utilidade desses **Mapeamentos** do teorema 18.1 em (KIFER; LAUSEN; WU, 1995) torna possível explorar a interpretação de modelos PDDL em F-Logic e *vice versa*.

Como descrito em (KIFER; LAUSEN; WU, 1995), há uma técnica de tradução da F-Logic para a Lógica de Predicados chamada de ***flattening*** com predicados chamados de ***wrapper***, semelhante ao esquema de tradução descrito em (DECKER, 1998) e exposto na tabela acima, para codificar diferentes tipos de especificação. Entretanto, nesta dissertação optou-se por utilizar uma pequena variação dessa tradução descrita em (KIFER; LAUSEN; WU, 1995) e em (LUDÄSCHER; YANG; KIFER, 1999). Essa pequena variação da tradução não utiliza os ***wrapper predicates***. Como exemplo dessa pequena variação, seja um f-átomo do Mundo dos Blocos: **Manipulador[Segurando@=>Bloco]**, esse f-átomo pode ser

interpretado em Lógica de Predicados como **Segurando(Manipulador, Bloco)**. E é desse modo que as definições a seguir devem denotar a Interpretação Correspondente $\overset{I}{\longleftrightarrow}$.

Com essa finalidade, na seção 4.1 apresenta-se a Interpretação Correspondente da descrição de tipos, predicados 1-ários, predicados 2-ários e predicados n-ários da PDDL e as classes e métodos da F-Logic.

4.1. Correspondência de Interpretação PDDL e F-Logic

Nessa seção serão apresentadas algumas definições necessárias para mostrar como é a Interpretação Correspondente entre os elementos da PDDL, tais como, Tipos e Predicados e as classes e métodos da F-Logic. Vale informar que o símbolo utilizado para denotar a Interpretação Correspondente nas definições é $\overset{I}{\longleftrightarrow}$.

A Interpretação Correspondente $\overset{I}{\longleftrightarrow}$ mostra uma interpretação possível da PDDL em F-Logic e *vice versa*. Nesse contexto, interpretação significa ler uma descrição PDDL e descrevê-la em F-Logic e *vice versa.*

No caso dos **tipos** e **subtipos** da PDDL existe uma interpretação em F-Logic baseada na propriedade *IS-A*. Para exemplificar essa interpretação, pode-se citar o domínio Logística que é já amplamente exposto nos trabalhos sobre PDDL, que possui os tipos Veículo, Pacote, Lugar, Cidade, Caminhão, Avião, Aeroporto e Localização. A descrição em F-Logic para os tipos do domínio Logística é:

Cidade :: $\top$, Veículo :: $\top$, Pacote :: $\top$ e Lugar :: $\top$. Essa descrição significa que cada Tipo está contido no conjunto universal de Tipos $\top$;

Caminhão :: Veículo, Avião :: Veículo, Aeroporto :: Local, Localização :: Local. Essa descrição significa que cada subtipo é um subconjunto de um dado Tipo. Abaixo, apresenta-se a definição para a Interpretação Correspondente de Tipos e Subtipos da PDDL e das classes e subclasses da F-Logic.

Definição 4.1 (Interpretação Correspondente de Tipos PDDL e Classes F-Logic) Os tipos da PDDL descritos no elemento **(:types** e as Classes da F-Logic têm uma interpretação correspondente da seguinte forma $\forall\ t_i \subseteq T \overset{I}{\longleftrightarrow} \forall\ Cl_i :: Cl$.

Com a definição 4.1, o domínio Mundo dos Blocos, por exemplo, pode ser interpretado de PDDL $\overset{I}{\longleftrightarrow}$ em F-Logic e *vice-versa.*

Tabela 5 - Classes F-Logic em Tipos PDDL

F-Logic – Classes	**Interpretação**	**PDDL – Tipos**

`Manipulador :: Cl.`
`Bloco :: Cl.`
`Mesa :: Cl.`
	$\overset{I}{\longleftrightarrow}$	`(:types`
		`  Manipulador - T`
		`  Bloco - T`
		`  Mesa - T)`

Em F-Logic a interpretação precisa satisfazer sua semântica a partir da F-estrutura **I** vista no capítulo 3. Por exemplo, considere o domínio Logística e suponha as F-fórmulas em:

- **I** = {Veículo, Local, Pacote, Cidade, Caminhão :: Veículo, Avião :: Veículo, Localização :: Local, Aeroporto :: Local};
- Se G = {Caminhão :: Veículo}, então, $\mathbf{I} \models_{\forall} G$ será satisfeita.
- Do contrário, se G = {Caminhão :: Local}, então, $\mathbf{I} \not\models_{\forall} G$, ou seja, $\mathbf{I} \models_{\forall} G$ não foi satisfeita.

A definição 4.1 mostra como interpretar os tipos e subtipos da PDDL e as classes e subclasses da F-Logic.

Tendo a definição de como interpretar os tipos PDDL e as Classes F-Logic, pode-se agora interpretar os **Predicados 1-ários** da PDDL e seus **correspondentes Métodos** F-Logic. Esses predicados são aqueles que determinam a propriedade de um dado tipo, por exemplo, **(disponível ?maq – Máquina)** que semanticamente descreve se uma dada Máquina está disponível ou não. Levando em consideração, esse exemplo, sua interpretação em F-Logic de (disponível ?maq – Máquina) é Máquina[disponível@=> Booleano] onde Booleano é o tipo primitivo descrito em F-Logic para Verdadeiro ou Falso. A definição abaixo demonstra como é essa interpretação correspondente entre os predicados 1-ários da PDDL e métodos da F-Logic.

Definição 4.2 (Interpretação Correspondente de Predicados 1-ários PDDL e Métodos F-Logic) Os Predicados 1-ários da PDDL descritos no elemento **(:predicates** na forma $(P_i^1 ?\text{tipo} - t_j)$ e Métodos F-Logic têm uma interpretação correspondente da seguinte forma

$$\forall_{i,j}\ P_i^1(t_j) \overset{I}{\longleftrightarrow} \forall_{i,j}\ Cl_j\,[ScalM_i@=>\ Booleano],\ \text{onde:}$$

a) PDDL: $\forall_{i,j}\ P_i^I(t_j)\ \overset{I}{\longleftrightarrow}$ em F-Logic: $Cl_j \subseteq G$ e $ScalM_i@=>\{Booleano\} \subseteq E$ sendo que $E \subseteq G$ onde E é o conjunto de métodos e Booleano é um tipo primitivo para que a F-estrutura $\mathbf{I} \models_{\vee} G$ seja satisfeita;

b) A interpretação Correspondente é feita para cada termo envolvido das duas linguagens: $P_i^I \overset{I}{\longleftrightarrow} ScalM_i$, $t_j \overset{I}{\longleftrightarrow} Cl_j$ e o tipo primitivo Booleano representa o verdadeiro ou falso da Lógica de Predicados.

Seguindo a definição 4.2, pode-se interpretar os métodos sem parâmetros da F-Logic e os predicados 1-ários em PDDL.

A tabela abaixo demonstra como interpretar o domínio Mundo dos Blocos de acordo com a definição 4.2:

Tabela 6 - Métodos escalar F-Logic em Predicados 1-ários PDDL

F-Logic – Métodos sem parâmetros	**Interpretação**	**PDDL – Predicados 1-ários**

Bloco[nadaSobre@=>Booleano].
Manipulador[garraVazia@=>Booleano].
	$\overset{I}{\longleftrightarrow}$	`(:predicates`
		`  (nadaSobre ?tip - Bloco)`
		`  (garraVazia ?tip - Manipulador)`
		`)`

Assim, para todo predicado em PDDL na forma (P_i^I? tipo – t_j), que é idêntico a um método **E** na F-Logic, bem tipado de acordo com a F-estrutura $\mathbf{I}^{(n)}_{=>}$, de modo que $\mathbf{E \subseteq G}$ e $\mathbf{I} \models_{\vee} \mathbf{G}$ seja satisfeita. Logo, todas as interpretações de PDDL em F-Logic devem satisfazer sua F-estrutura.

Para esclarecer a definição 4.2 no que tange à F-Logic, tenha o domínio Planta de Manufatura onde o tipo Máquina foi interpretado de PDDL em F-Logic:

- G = {Máquina :: ⊤};
- E = {Máquina[disponível@=>Booleano]};
- Portanto, $E \subseteq G$ para que $\mathbf{I} \models_{\vee} G$ seja satisfeita.

No caso dos Predicados 2-ários e n-ários, há a necessidade de definir um tipo do predicado como o mais relevante entre os tipos para denominá-lo como **Tipo mais Relevante** (TMR) do predicado. Houve uma abordagem semelhante a essa utilizada para interpretar a PDDL em Objetos em (TONIDANDEL et al, 2006). A definição abaixo para obter o TMR é uma abordagem com a qual se pode ter o uso da Lógica de Primeira Ordem para Objetos como vista em (KIFER; LAUSEN; WU, 1995), já que na Lógica de Predicados isso não é necessário. Para a PDDL, o TMR não provoca nenhuma perda semântica.

Definição 4.3 (TMR – Tipo mais Relevante e Parâmetros) Os Predicados da PDDL 2-ários na forma (P_i^2 ?tipo– t_j?tipo1–t_l) e n-ários na forma (P_i^2 ?tipo– t_j?tipo1–t_l? ... ?tipon–t_n) com n > 2 possuem apenas um tipo mais relevante da lista de tipos.

1. Para o Predicado 2-ário, há uma lista com 2 tipos onde o primeiro tipo é o TMR:

   Seja o predicado P_i^2 (t_j, t_l), onde o tipo t_j é o Tipo mais Relevante e t_l é o segundo tipo;

2. O tipo TMR de um predicado PDDL é o id-termo Classe F-Logic;
3. Para o Predicado n-ário para n > 2, há uma lista com n tipos onde o primeiro tipo é o TMR e os demais, Parâmetros:

   Seja P_i^n (t_j, t_l,..., t_n), o tipo t_j é o Tipo mais Relevante e t_l,..., t_n são Parâmetros.

O Tipo mais relevante (TMR) foi definido como o tipo com o qual se terá o ***id-termo*** Classe de cada assinatura de método em F-Logic e os Parâmetros são classes que estão nos métodos como parâmetros. Para a lógica de objetos F-Logic, as ações e transformações de estados acontecem a partir dos objetos e, por isso, é necessário ter um tipo mais relevante que seja responsável por essas ações. Isso não tem relevância para a PDDL.

A F-Logic é uma Lógica de Primeira Ordem para Objetos e, assim, quando há mais de um tipo na lista de tipos de um predicado, para interpretar esse predicado em F-Logic é necessário assumir que o primeiro Tipo da lista de tipos é o TMR e, assim, ele será o ***id-termo*** Classe em F-Logic.

Com a definição 4.3 tem-se, então, o ***id-termo*** Classe escolhido em predicados com mais de um tipo. Para completar a definição da Interpretação Correspondente de predicados n-ários PDDL, com n > 2, e métodos escalares F-Logic segue a definição abaixo:

Definição 4.4 (Interpretação Correspondente de Predicados n-ários PDDL e de métodos escalares com parâmetros F-Logic) Os Predicados n-ários, com n > 2, da PDDL descritos no elemento (**:predicates** na forma (P_i^n ?$tipo_j$– t_j?$tipo_l$–t_l,..., ?$tipo_n$–t_n) e métodos escalares com parâmetros da F-Logic na forma Cl_j[$ScalM_i$@Q_l,..., Q_n=>Booleano] têm uma interpretação correspondente da seguinte forma

$\forall_{i,j,l,...,n}\ P_i^n(t_j, t_l,..., t_n) \overset{I}{\longleftrightarrow} \forall_{i,j,l,...,n}\ Cl_j[ScalM_i@Q_l,..., Q_n => Booleano]$, onde:

a) PDDL: $\forall_{i,j,l,...,n}\ P_i^n(t_j, t_l,..., t_n) \xleftrightarrow{I}$ F-Logic:$Cl_j \subseteq G$ e $ScalM_i@Q_l,...,Q_n=>\{Booleano\} \subseteq E$ onde:

   i) Em PDDL, o tipo t_j é o TMR;

   ii) Em F-Logic, o ***id-termo*** classe Cl_j é o TMR;

   iii) Em PDDL, os tipos t_l, ..., t_n da Lista de Tipos dos predicados são Parâmetros dos métodos escalares na F-Logic;

   iv) Em F-Logic, o TMR e os Parâmetros dos métodos escalares são os tipos da Lista de Tipos de predicados em PDDL:

   Lista de Tipos LT = {TMR, {Parâmetros}};

   v) Do predicado n-ário da PDDL em método escalar da F-Logic tem-se:

   TMR[P_i^n @Parâmetros=>Booleano];

   vi) Do método escalar em F-Logic em predicado n-ário da PDDL tem-se:

   $ScalM_i$(TMR, Parâmetros);

b) A interpretação Correspondente é feita para cada termo envolvido das duas linguagens: $P_i^n \xleftrightarrow{I} ScalM_i$, $t_j \xleftrightarrow{I} Cl_j$, $t_l,...,t_n \xleftrightarrow{I} Q_l,...,Q_n$ e o tipo primitivo Booleano representa o verdadeiro ou falso da Lógica de Predicados.

Para esclarecer a definição acima, seja o predicado **(Acessível ?tipo$_1$ – Veículo ?tipo$_2$?tipo$_3$ – Localização)** do domínio **MetricVehicle**. Pela definição 4.3, o tipo **TMR** é **Veículo** e ele é o ***id-termo*** classe. Pela definição 4.4, o predicado **Acessível** deve ser interpretado em F-Logic como **Veículo[Acessível@Localização, Localização=>Booleano]**. Em seguida, pode-se demonstrar também a interpretação desse método com parâmetros da F-Logic Veículo[Acessível@Localização, Localização=>Booleano] em predicado n-ário da PDDL (Acessível ?tipo – Veículo ?tipo$_1$?tipo$_2$ – Localização). Por exemplo, pela definição 4.3 sabe-se que Veículo é o **TMR** e os **Parâmetros** são a classe Localização, de modo que se tem a lista de tipos **LT={TMR,{Parâmetros}}**. Pela definição 4.4, o método escalar com parâmetros $ScalM_i$ Acessível é interpretado como Predicado P_i^n. E, logo, a interpretação é P_i^n(LT), que nesse exemplo trata-se do predicado (Acessível ?tipo – Veículo ?tipo$_1$?tipo$_2$ – Localização). Dessa maneira, todos os predicados n-ários, com n > 2, e os métodos com parâmetros da F-Logic têm uma Interpretação Correspondente.

No caso dos Predicados 2-ários da PDDL na forma **(P_i^2 ?tipo$_j$–t$_j$?tipo$_l$–t$_l$)** pode-se interpretar esse predicado de duas formas em F-Logic se a descrição de domínio em PDDL for a única fonte para a interpretação. Isso acontece porque em F-Logic, pela sua assinatura de método, **não** há como saber se haverá tuplas de objeto ou apenas uma tupla.

Para um melhor esclarecimento sobre "=>", no domínio Mundo dos Blocos tem-se que:

- o predicado 2-ário em PDDL (sobre ?tip–Bloco?tipo–Bloco) deve ser interpretado adequadamente como Bloco[sobre@=>Bloco];
- Assim só pode existir estados onde apenas um bloco fica sobre outro bloco, isto é, (sobre A B) em PDDL ou A[sobre@->B] em F-Logic. Não poderia nesse caso existir (sobre C B) em PDDL ou C[sobre@->B] em F-Logic;
- O símbolo "=>" na F-Logic denota casos onde só pode haver uma tupla de objetos. Na PDDL, ou isso é tratado em uma *constraint* ou pelas descrições das ações.

Para um melhor esclarecimento sobre "=>>", no domínio Planta de Manufatura tem-se que:

- O predicado 2-ário (tem?tip–Produto?tipo–Atributo) deve ser interpretado adequadamente como Produto[tem @=>>Atributo];
- Assim, um produto pode ter mais de um atributo;
- Em PDDL tem-se os seguintes literais (tem Caneta Corpo) (tem Caneta Tampa) (tem Caneta Refil) mostrando a idéia de que um produto pode ter *n* atributos;
- Em F-Logic, a mesma descrição feita para a PDDL acima, é Caneta[tem @->> {Corpo, Tampa, Refil}] também mostrando a idéia de que um produto pode ter *n* atributos;
- O símbolo "=>>" na F-Logic denota casos onde há mais de uma tupla de objetos. Na PDDL isso não é levado em consideração.

Dessa maneira, fazendo a interpretação de Predicados 2-ários da PDDL apenas a partir das informações dos predicados, não há como evidenciar a interpretação adequada em F-Logic. Poderia existir uma política de modelagem onde o engenheiro do conhecimento devesse criar *constraints* em todos os casos que houvesse predicados como o predicado (sobre ?tip–Bloco?tipo–Bloco). Assim, quando não houver *constraints*, então, a escolha do símbolo é "=>>".

Portanto, pela definição abaixo, optou-se pela Interpretação Correspondente dos Predicados 2-ários da descrição de domínio da PDDL e métodos de conjunto de valores da F-Logic que fornecesse maior abrangência ao seu uso. Assim, escolheu-se o símbolo "=>>", que significa uma ou mais tuplas de objetos.

Definição 4.5 (Interpretação Correspondente e Abrangente de Predicados 2-ários PDDL e métodos F-Logic) Os Predicados 2-ários da PDDL da descrição de domínio na forma (P_i^2 ?$tipo_j$–t_j ?$tipo_l$–t_l) e o método F-Logic na forma Cl_j [$SetM_i$@=>> Cl_l] têm uma interpretação correspondente da seguinte forma

$$\forall_{i,j,l}\ P_i^2(t_j, t_l) \overset{I}{\longleftrightarrow} \forall_{i,j,l}\ Cl_j[SetM_i@=>>Cl_l], \text{ onde:}$$

a) Como t_j é o TMR e só existe o segundo tipo t_l na lista de tipos, então, trata-se de um predicado 2-ário;
b) Do predicado 2-ário PDDL em método de conjunto de valores F-Logic,
   TMR[Predicado@=>>Segundo Tipo];
c) Do método de conjunto de valores F-Logic em predicado 2-ário PDDL
   $SetM_i$(id-termo classe, Cl_l);
d) A interpretação Correspondente é feita para cada termo envolvido das duas linguagens: $P_i^2 \overset{I}{\longleftrightarrow} SetM_i$, $t_j \overset{I}{\longleftrightarrow} Cl_j$ e $t_l \overset{I}{\longleftrightarrow} Q_l$.

Com a definição 4.5 acima, mostra-se que a partir da descrição de domínio da PDDL optou-se por interpretar os predicados 2-ários de forma abrangente, ou seja, fornecendo a todo predicado 2-ário a possibilidade de lidar com mais de uma tupla de objetos na F-Logic. Justamente porque na PDDL os seus predicados não possuem essa informação.

Do contrário, como já explanado, poderia utilizar-se o elemento **(:contraints** para suprir o modelo PDDL com essa informação. Por exemplo, sabe-se que no domínio Mundo dos Blocos, um bloco somente pode ficar sobre outro Bloco, ou em cima da mesa, ou ainda na garra do robô. Para tratar isso como uma ***constraint*** abaixo segue a descrição em PDDL:

```
(:constraints
  (and
    (forall (?blo1 - Block ?blo2 - Block ?blo - Block)
      (always
        (imply(and(on ?blo ?blo1)(on ?blo ?blo2))(= ?blo1 ?blo2)
        ))))
```

Essa ***constraint*** auxiliaria como interpretar predicados 2-ários em PDDL para o símbolo => em F-Logic.

Portanto, os predicados 2-ários da PDDL serão interpretados conforme a definição 4.5 da Interpretação Correspondente com o símbolo "=>>" que lhe fornece maior abrangência na F-Logic. Lembrando que os predicados na PDDL não possuem esse tipo de informação.

Dessa forma, cumpre-se as interpretações de classes, métodos escalares e métodos de conjunto de valores da F-Logic e os tipos, predicados 1-ários, 2-ários e n-ários da PDDL. Para exemplificar as últimas definições a tabela abaixo mostra algumas dessas interpretações:

Tabela 7 - Métodos F-Logic em Predicados 2-ários e n-ário PDDL

F-Logic – Métodos	**Interpretação**	**PDDL – Predicados 2-ários e n-ários**
Bloco[sobre@=>> Bloco]. Bloco[encima@=>> Mesa]. Robô[segurando@Bloco, Mesa => Booleano].	$\overset{I}{\longleftrightarrow}$	(:predicates (sobre ?tip ?tip$_1$ – Bloco) (encima ?tip – Bloco ?tip$_1$ – Mesa) (segurando ?tip – Robo ?tip – Bloco ?tip$_1$ – Mesa)))

No domínio Mundo dos Blocos da tabela acima, para explicar uma das interpretações da tabela, colocou-se um robô para organizar blocos em mais de uma mesa. Por isso, o método Robo[segurando@Bloco, Mesa => Booleano]. Daí a sua interpretação em um predicado n-ário segurando(Robo, Bloco, Mesa) ou em PDDL (segurando ?tip – Robo ?tip – Bloco ?tip1 – Mesa)). As interpretações ora apresentadas na tabela estão de acordo com as últimas definições que completam o como interpretar classes, métodos escalar e de conjunto de valores da F-Logic, além dos tipos, predicados 1-ários, 2-ários e n-ários da PDDL.

Próximo capítulo, será discutida a interpretação de modelos UML.P em F-Logic apresentada em (GOMES, 2008) para extração de conhecimento semelhante à discussão apresentada aqui sobre PDDL e F-Logic

CAPÍTULO 5

5. INTERPRETAÇÃO DA UML.P EM F-LOGIC

Nesse capítulo, será apresentada a interpretação do diagrama de classes da UML.P em classes e métodos da descrição F-Logic de acordo com (GOMES, 2008). Dessa Interpretação Completa em F-Logic do diagrama de classes da UML.P, criar-se-á uma interpretação chamada *IFR* de **forma reduzida** que está contida na Interpretação Completa. Essa interpretação *IFR* de forma reduzida será utilizada para demonstrar a equivalência com a Interpretação Completa entre a F-Logic e PDDL.

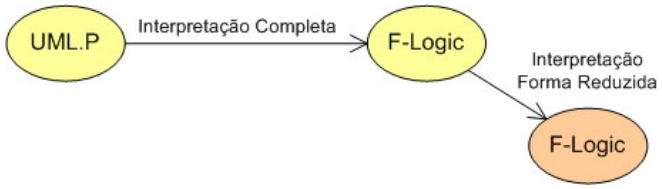


Figura 24 – Interpretação Completa da UML.P em F-Logic e sua Interpretação Forma Reduzida – *IFR*.

Vale relembrar que os trabalhos em (RAMALHO; ROBIN, 2004) e em (RAMALHO; ROBIN; SCHIEL, 2004) possuem o diagrama de classes da UML interpretado em F-Logic. Esses trabalhos motivaram a Interpretação Completa criada em (GOMES, 2008) e a figura acima apresenta o subconjunto dessa Interpretação Completa, que é a Interpretação *IFR* de forma reduzida.

Através das próximas seções serão demonstradas as interpretações Completa e Forma Reduzida.

5.1. Interpretação Completa UML.P em F-Logic

As assinaturas auxiliares foram interpretadas em (GOMES, 2008) para fornecer suporte aos conceitos da UML.P, tais como, tipos primitivos (booleano, inteiro e nulo), limites de conjunto, mutabilidade, tipo da associação, tipo do estereótipo, a característica da associação e a classe fundamental, que é a classe universal que contém todas as classes.

Tabela 8 – Assinatura Auxiliares para Modelos UML.P

Assinaturas Auxiliares	Descrição
`tipoPrimDado[existe => booleano; existe *-> verdadeiro]`	Determina os tipos de dados primários que serão utilizados na interpretação da UML.P. Os tipos de dados serão: Inteiro, booleano, entre outros.
`limConjunto [existe => booleano; existe *-> verdadeiro]`	Determina os limites de qualquer conjunto dado na UML.P.
`mutabilidade[existe => booleano; existe *-> verdadeiro]`	Determina a mutabilidade das associações.
`assocTipoEx[existe => booleano; existe *-> verdadeiro]`	Determina o tipo da associação.
`tipoEstereotipo[existe => booleano; existe *-> verdadeiro]`	Determina o tipo do estereótipo das classes.
`caracteristica[existe => booleano; existe *-> verdadeiro]`	Determina as características das associações.
`classeFund[existe => booleano ; estereotipo => tipoEstereotipo; existe *-> verdadeiro]`	Determina a classe fundamental que estruturará todas as classes dos domínios.

Fonte: Gomes, 2008

Qualquer diagrama de classes projetado para um domínio em questão pode utilizar as assinaturas auxiliares para enriquecer o Modelo. Essa propriedade da Interpretação Completa realizada em (GOMES, 2008) adaptou-se ao considerar todos os elementos visuais do diagrama de classes.

Na tabela a seguir há a interpretação F-Logic para o elemento Classe da UML.P.

Tabela 9 – Interpretação de classes em F-Logic

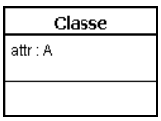
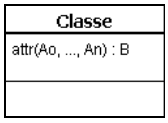
Simbologia UML.P	Estereótipo	Interpretação F-Logic
<<estereótipo>> Classe	< >	`classe::classeFund classe[ estereotipo *-> ester_nulo]`
	<Agente>	`classe::classeFund ester_Agente:tipoEstereotipo classe[ estereotipo *-> Ester_Agente]`
	<Utilitário>	`Classe::classeFund ester_Utilitario:tipoEstereotipo classe[ estereotipo *-> Ester_Utilitario]`

Fonte: GOMES, 2008

Essa interpretação torna possível descrever em F-Logic todas as classes projetadas para um determinado domínio.

Com a interpretação F-Logic para classes, agora apresenta-se como interpretar os atributos de uma classe UML.P assim como segue na tabela a seguir:

Tabela 10 – Interpretação de atributos

Simbologia UML.P	Interpretação F-Logic
	`Classe[attr => A; attr@limConjunto => inteiro; attr@lim_Inferior *-> 1; attr@lim_Superior *-> 1; attr@caracteristica => mutabilidade; attr@atrib_carac *-> mut_Alteravel]`
	`Classe[attr@A0, …, An => B; attr@limConjunto => inteiro; attr@lim_Inferior *-> 1; attr@lim_Superior *-> 1; attr@caracteristica => mutabilidade; attr@atrib_carac *-> mut_Alteravel]`

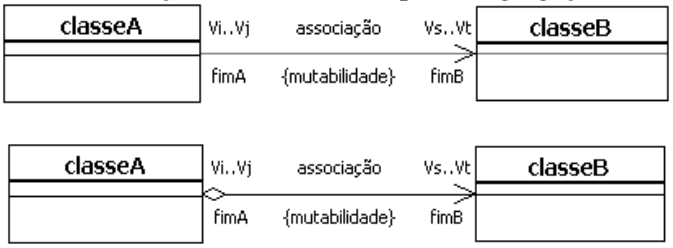
Fonte: GOMES, 2008

Como apresentado na tabela acima, há dois tipos de atributos interpretados em F-Logic: atributo com parâmetros e sem parâmetros. A tabela acima mostra o f-átomo Classe[attr @=>A] onde A é um tipo primitivo.

Após apresentar as interpretações F-Logic para classe e atributos, serão mostradas as interpretações para associações de classes tal como segue na próxima tabela.

Tabela 11 – Interpretação de Associação de Classes

Associação de Classes: Simples e Agregação



Interpretação F-Logic

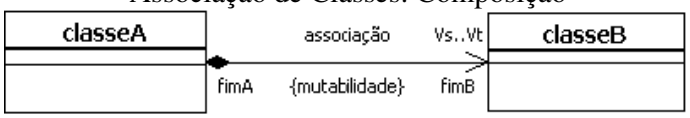
P/ $V_{j,t} = 1$

```
classeA[associação_classeB => mutabilidade; associação_classeB *-> <mutabilidade>;
associação_classeB_fimB => classeB;
associação_classeB_fimB@limConjunto => inteiro;
associação_classeB_fimB@lim_Inferior *-> <Vs>;
associação_classeB_fimB@lim_Superior *-> <Vt>;
associação_classeB_fimB@caracteristica => assocTipoEx;
associação_classeB_fimB@assoc_TipoEx *-> <AssociaçãoTipo>]
```

P/ $V_{j,t} > 1$

As assinaturas serão idênticas as mostradas acima, porém o método `associação_classeX_fimX=>classeX` terá o símbolo => substituído por =>>.

Associação de Classes: Composição



Interpretação F-Logic

A interpretação para a ClasseA será idêntica ao primeiro e segundo exemplo desta tabela, observando a navegabilidade da associação e ao tipo de associação, e a ClasseB terá sua interpretação da seguinte maneira:

```
classeB[associação_classeA => mutabilidade;
associação_classeA *-> <mutabilidade>;
associação_classeA_fimA => classeA;
associação_classeA_fimA@limConjunto => inteiro;
associação_classeA_fimA@lim_Inferior *-> 1;
associação_classeA_fimA@lim_Superior *-> 1;
associação_classeA_fimA@caracteristica => assocTipoEx;
associação_classeA_fimA@assoc_TipoEx *-> assoc_Composicao]
```

Fonte: GOMES, 2008

Dadas as interpretações expostas nas tabelas acima, a seguir as F-Moléculas com seus F-átomos:

ASSINATURAS AUXILIARES (FIXO)

```
tipoPrimDado[existe => booleano; existe *-> verdadeiro]
limConjunto[existe => booleano; existe *-> verdadeiro]
mutabilidade[existe => booleano; existe *-> verdadeiro]
assocTipoEx[existe => booleano; existe *-> verdadeiro]
tipoEstereotipo[existe => booleano; existe *-> verdadeiro]
caracteristica[existe => booleano; existe *-> verdadeiro]
classeFund[existe => booleano ; estereotipo => tipoEstereotipo; existe *->
verdadeiro].
```

As assinaturas auxiliares possibilitam que os diagramas de classes da UML.P sejam interpretados em F-Logic tendo um F-átomo para cada elemento do diagrama. Por exemplo, há um F-átomo para os tipos primitivos de dados, tais como, booleano e inteiro. Há um F-átomo para definir as condições limítrofes da multiplicidade das associações de classes. Um F-átomo para a mutabilidade. F-átomos para o tipo de associação, o tipo de estereótipo da classe, a característica e a classe universal.

Após mostrar a padronização das assinaturas auxiliares em qualquer diagrama de classes, leva-se em consideração o domínio Mundo dos Blocos para demonstrar as interpretações F-Logic:

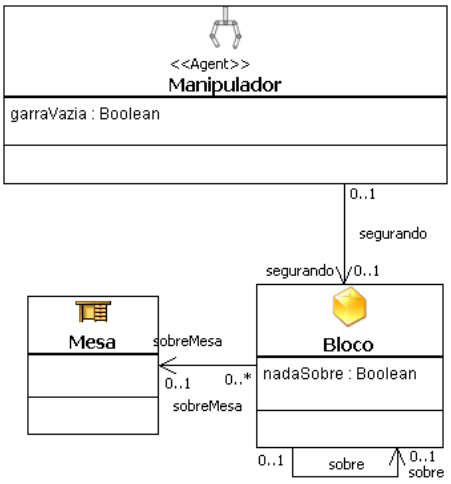


Figura 25 – Interpretação do domínio Mundo dos Blocos.

Generalizações de Classes

```
Bloco::classeFund
Mesa::classeFund
Manipulador::classeFund
```

As generalizações de classes expostas acima como F-átomos, todas elas são subclasses da classeFund, que é a classe universal. Após demonstrar como descrever a criação das classes, abaixo seguem as descrições de assinaturas de classes:

Assinaturas de Classes

```
ester_agente : tipoEstereotipo
```

No F-átomo acima, descreve-se que o estereótipo Agente pertence à classe Tipo de Estereótipo, onde objetos Agentes são aqueles que operam mudanças no domínio. Abaixo, o F-átomo estereótipo da classe Manipulador mostra-se como Agente no domínio Mundo dos Blocos.

```
Manipulador[estereotipo@*->ester_Agente;
garraVazia@=> Booleano;
```

No F-átomo acima, garraVazia é um método escalar com o tipo primitivo Booleano. Para a classe Manipulador há duas situações: sua garra está vazia ou não. Nos três F-átomos logo

abaixo, há a descrição sobre as condições limítrofes do método garra vazia e os dois últimos F-átomos lidam com a mutabilidade e seu tipo para a classe Manipulador:

```
garraVazia@limConjunto => inteiro;
garraVazia@lim_Inferior *-> 1;
garraVazia@lim_Superior *-> 1;
garraVazia@caracteristica=>mutabilidade;
garraVazia@atrib_carac*->mut_Estatico]
```

A classe Bloco abaixo possui na sua descrição o formato semelhante ao da classe Manipulador, exceto o F-átomo que descreve Agente. Pois Bloco não opera mudanças no domínio.

```
Bloco[nadaSobre@=> Booleano;
nadaSobre@limConjunto => inteiro;
nadaSobre@lim_Inferior *-> 1;
nadaSobre@lim_Superior *-> 1;
nadaSobre@caracteristica=>mutabilidade;
nadaSobre@atrib_carac*->mut_Estatico]
```

Dessa forma, foram apresentados os F-átomos pertinentes a descrição dos atributos de classes, ou métodos em F-Logic. Abaixo estão os F-átomos pertinentes às associações de classes:

Assinatura das Associações

```
Manipulador[segurando_Bloco@=>mutabilidade;
segurando_Bloco@*->mut_Estatico;
segurando_Bloco_segurando@=>Bloco;
```

Os F-átomos acima são F-átomos compactos, pois, possuem na sua descrição as seguintes estruturas: *Manipulador [nA_nCEO @ => mutabilidade]* no primeiro f-átomo e *Manipulador [nA_nCEO_nEOA @ => nCEO]* no segundo f-átomo, onde nA é nome da associação, nCEO é o nome da classe de extremidade oposta e nEOA é o nome da extremidade oposta da associação de acordo com (GOMES, 2008). Abaixo, os primeiros três F-átomos lidam com as condições limítrofes da multiplicidade da associação entre Manipulador e Bloco. Os dois últimos F-átomos dizem respeito à associação.

```
segurando_Bloco_segurando@limConjunto=>inteiro;
segurando_Bloco_segurando@lim_Inferior*->1;
segurando_Bloco_segurando@lim_Superior*->*;
segurando_Bloco_segurando@caracteristica=>assocTipoEx;
segurando_Bloco_segurando@assoc_TipoEx*->assoc_Simples]
```

Abaixo estão os F-átomos descritos para a classe Bloco. Como pode ser visto, essa descrição é bastante semelhante à descrição realizada acima para classe Manipulador. É óbvio que se trata de uma descrição maior porque ela lida com duas associações para a classe em questão:

```
Bloco[sobre_Bloco@=>mutabilidade;
sobre_Bloco_sobre@=>Bloco;
sobre_Bloco_sobre@limConjunto=>inteiro;
sobre_Bloco_sobre@caracteristica=>assocTipoEx;
sobre_Bloco*->mut_Estatico;
sobre_Bloco_sobre@lim_Inferior*->1;
sobre_Bloco_sobre@lim_Superior*->1;
sobre_Bloco_sobre@assoc_TipoEx*->assoc_Simples;
sobreMesa_Mesa@=>mutabilidade;
sobreMesa_Bloco_sobreMesa@=>Bloco;
sobreMesa_Bloco_sobreMesa@limConjunto=>inteiro;
sobreMesa_Bloco_sobreMesa@caracteristica=>assocTipoEx;
sobreMesa_Bloco@*->mut_Estatico;
sobreMesa_Bloco_sobreMesa@lim_Inferior*->1;
sobreMesa_Bloco_sobreMesa@lim_Superior*->1;
sobreMesa_Bloco_sobreMesa@assoc_TipoEx*->assoc_Simples]
```

Com os exemplos de descrição apresentados anteriormente para o domínio Mundo dos Blocos, buscou-se esclarecer o uso das interpretações postas em tabelas de acordo com (GOMES, 2008).

Como fora relatado no início desse capítulo, a descrição em F-Logic da Interpretação Completa do diagrama de classes UML.P possui muitos F-átomos que apropriadamente estão adaptados para descrever cada elemento do diagrama de classes. Entretanto, também fora relatado no início desse capítulo que será criada uma Interpretação *IFR* de Forma Reduzida que terá alguns F-átomos da Interpretação Completa.

Definição 5.1 (*IFR* – Interpretação F-Logic Forma Reduzida) A Forma Reduzida é um conjunto de F-átomos extraídos da Interpretação Completa em F-Logic de acordo com as seguintes asserções:

1. A interpretação Completa em F-Logic é uma fórmula λ composta de F-átomos;
2. A interpretação Forma Reduzida é uma fórmula μ composta de F-átomos;
3. De tal maneira que $\lambda \models_F \mu$ na teoria da F-Logic;
4. Assim, a Forma Reduzida possui os seguintes F-átomos:
   a. Classe – *Classe::classeFund*;
   b. Atributo Simples – *Classe[attr@=> A]* onde A é o tipo primitivo Booleano;

c. Atributo com parâmetros – *Classe[attr@A0, …, An@=> B]* onde B é o tipo primitivo Booleano e A0, …, An são parâmetros;
d. Associação de Classes – *Classe[associação_classeB_fimB@=> classeB]:*
   i. É um F-átomo compacto;
   ii. Sua forma estendida é: *Classe[associação@=> classeB]* $\wedge$ *Classe[fimB@=> classeB]* $\leftarrow$ *Classe[associação_classeB_fimB@=>classeB];*
e. O F-átomo escolhido da forma estendida é *Classe[associação@=>classeB].*

Com a definição acima da Interpretação *IFR* de Forma Reduzida, pode-se demonstrar que apropriadamente ela é um subconjunto da Interpretação Completa.

Teorema 5.1 (Da Interpretação Completa deriva-se a Forma Reduzida) A Interpretação F-Logic Completa do diagrama de classes possui *n* F-átomos.

Seja *IFR* a Interpretação Forma Reduzida levando em consideração os seus F-átomos da definição 5.1, *IFR* $\subseteq$ *Interpretação Completa.*

Prova

1. *Seja a definição 5.1 itens a), b), c) e e), há 4 F-átomos para a descrição da IFR; Onde IFR={ f-atom*$_1$ $\wedge$ *f-atom*$_2$ $\wedge$ *f-atom*$_3$ $\wedge$ *f-atom*$_4$*}.*
2. *Os 4 tipos de f-átomos são:*
   a. *f-atom*$_1$ *= todas as classes na forma Classe :: classeFund;*
   b. *f-atom*$_2$ *= todos os atributos simples das classes na forma Classe[attr@=>A] onde A é o tipo primitivo Booleano;*
   c. *f-atom*$_3$ *= todos os atributos com parâmetros das classes na forma Classe[attr@A0, …, An@=>B] onde B é o tipo primitivo Booleano e A0, …, An são parâmetros de classes;*
   d. *f-atom*$_4$ *= todas as associações de classes na forma Classe[associação@=>classeB];*
3. *A Interpretação F-Logic Completa possui uma conjunção de n tipos de F-átomos;*
4. *A Interpretação IFR de Forma Reduzida possui uma conjunção de 4 tipos de F-átomos contidos na conjunção de n tipos de F-átomos da Interpretação Completa;*
5. *Assim, obtém-se os 4 tipos de F-átomos da Interpretação Completa:*

$$\frac{f-atom_1 \wedge f-atom_2 \wedge \ldots \wedge f-atom_{n-2} \wedge f-atom_{n-1} \wedge f-atom_n}{f-atom_1 \wedge f-atom_2 \wedge f-atom_3 \wedge f-atom_4}$$

6. *∴ IFR ⊆ Interpretação Completa □*

Com o Teorema 5.1 demonstra-se como obter a *IFR* da descrição de F-átomos realizada em (GOMES, 2008). Também é importante citar que a interpretação *IFR* de forma reduzida são 4 tipos de f-átomos que possuem equivalência com os f-átomos da Interpretação Correspondente da PDDL e F-Logic, que foi apresentada no capítulo 4. Esses 4 tipos de f-átomos da *IFR* são o f-átomo de classe, o f-átomo de Associação, o f-átomo de Atributo simples e o f-átomo de Atributo com parâmetros. Esses f-átomos são equivalentes respectivamente ao f-átomo de Tipo, o f-átomo de Predicado 2-ário, o f-átomo de Predicado 1-ário e o f-átomo de Predicado n-ário da Interpretação Correspondente.

No próximo capítulo serão demonstradas as equivalências entre F-átomos da UML.P e F-átomos da PDDL, além de demonstrar a equivalência do modelo UML.P com o modelo PDDL.

CAPÍTULO 6

6. INTERPRETAÇÃO DA UML.P EM PDDL UTILIZANDO A F-LOGIC

No capítulo 4 apresentou-se como interpretar tipos, predicados 1-ários, predicados 2-ários e predicados n-ários da PDDL em classes e métodos da F-Logic e *vice versa*. Já no capítulo 5 utilizou-se da Interpretação Completa do Diagrama de Classes da UML.P em classes e métodos da F-Logic realizada em (GOMES, 2008). E como se trata de um trabalho sobre extração de conhecimento há ***n*** tipos de F-átomos descritos nessa Interpretação de modo a apropriadamente representar o diagrama de classes. Entretanto, para essa dissertação o objetivo é demonstrar que classes, atributos de classes e associações na UML.P podem ser interpretados em tipos e predicados da PDDL. Como já fora citado, UML.P e PDDL são linguagens semiformais e, por isso, escolheu-se a F-Logic. Daí, no mesmo capítulo 5, obteve-se 4 tipos de F-átomos da Interpretação Completa da UML.P correspondentes aos F-átomos da Interpretação Correspondente do capítulo 4, chamando-a de interpretação *IFR* de forma reduzida.

Dessa forma, pretende-se neste capítulo provar a interpretação de elementos do modelo UML.P em elementos do modelo PDDL de acordo com a figura a seguir:

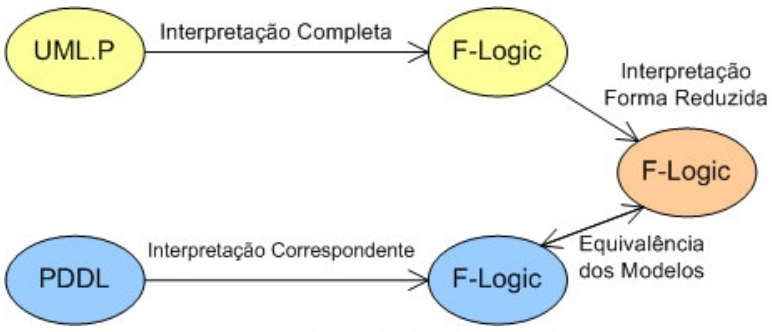


Figura 26 – Equivalência de Modelos.

A figura acima representa a Interpretação Correspondente do capítulo 4, a Interpretação Completa de acordo com (GOMES, 2008), que por inferência criou-se a *IFR*, Interpretação de Forma Reduzida, ambas do capítulo 5. A *IFR* tem elementos que possuem equivalência com

os elementos da Interpretação Correspondente, que será demonstrada nesse capítulo. E para finalizar, pretende-se demonstrar que os elementos dos Modelos PDDL são conseqüência lógica dos elementos dos Modelos UML.P de acordo com a figura abaixo:

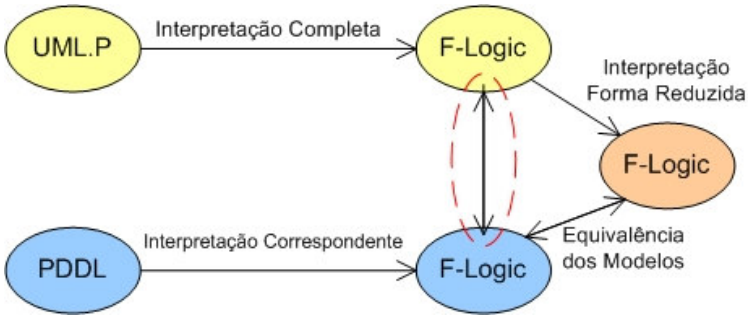


Figura 27 – Modelos UML.P e Modelos PDDL.

Portanto, através das próximas seções, haverá demonstrações sobre as provas de equivalência de modelos.

6.1. Estudo da Interpretação da UML.P para PDDL através da F-Logic

Esta seção põe em discussão o objetivo desse trabalho que é apresentar demonstrações da interpretação da UML.P em PDDL. Entretanto, foi necessário definir um conjunto de elementos para tratar domínios de Planejamento Automático. Esse conjunto de elementos possui Classes na linguagem UML.P e Tipos na linguagem PDDL, Associações e Atributos na linguagem UML.P e Predicados na linguagem PDDL.

Para tanto, houve a necessidade de produzir a seguinte definição com o objetivo de apresentar exatamente os elementos que devem ser tratados nessa seção.

Definição 6.1 (Suposição Restritiva de Declaração de Domínios) A Suposição Restritiva de Declaração de Domínios (SRDD) define os elementos que os modelos em UML.P e PDDL possuem em comum para Domínios de Planejamento Automático.

i. Seja o mesmo domínio U modelado tanto em UML.P quanto em PDDL;
ii. Sendo que em UML.P existe um conjunto de classes chamado Cl $=\{Cl_1, Cl_2, ..., Cl_n\}$ onde cada classe de Cl possui nomenclatura idêntica a um tipo de T em PDDL do conjunto de tipos T $=\{T_1, T_2, ..., T_n\}$ formando uma bijeção;

iii. Existe em UML.P um conjunto de atributos At $=\{At_1, At_2, ..., At_i\}$, onde cada atributo de At possui nomenclatura idêntica a um predicado 1-ário de P^1 em PDDL do conjunto $P^1=\{ P^1_1, P^1_2, ..., P^1_i \}$ formando uma bijeção;

iv. Existe em UML.P um conjunto de associações As $=\{As_1, As_2, ..., As_j\}$, onde cada associação de As possui nomenclatura idêntica a um predicado 2-ário de P^2 em PDDL do conjunto $P^2=\{ P^2_1, P^2_2, ..., P^2_j \}$ formando uma bijeção;

v. Existe em UML.P um conjunto de atributos com parâmetros Atp $=\{Atp_1, Atp_2, ..., Atp_i\}$, onde cada atributo com parâmetros de Atp possui nomenclatura idêntica a um predicado n-ário de P^n em PDDL do conjunto $P^n=\{ P^n_1, P^n_2, ..., P^n_i \}$ formando uma bijeção;

Para esclarecer o objetivo da SRDD definida acima, a figura abaixo apresenta que considerando a SRDD, há para o mesmo Domínio U, dois modelos – um em UML.P e outro em PDDL. Sendo que esses dois modelos possuem quatro conjuntos e o essencial é que um modelo em UML.P com esses quatro conjuntos possui a mesma nomenclatura com os outros quatro conjuntos do modelo em PDDL, formando ambos bijeções.

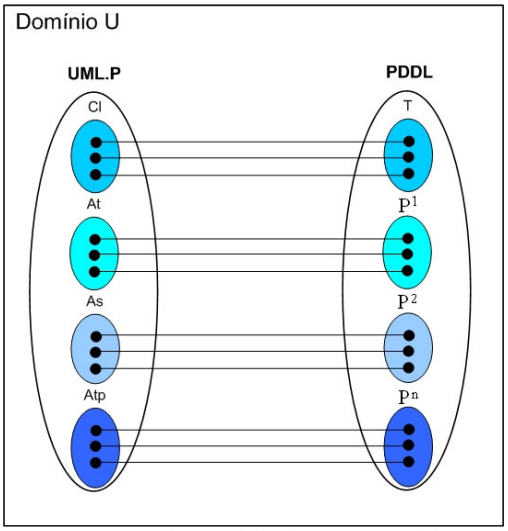


Figura 28 – Nomenclaturas idênticas nos modelos UML.P e PDDL.

Tomando o domínio Mundo dos Blocos, a SRDD restringe que em UML.P haja o elemento visual classe com nomenclatura “Bloco” e restringe que em PDDL haja a descrição de tipo com nomenclatura “Bloco”. Qualquer variação sobre isso já não permite fazer a correspondência de modelos.

Outro fato importante na correspondência de modelos UML.P e PDDL, foi o levantamento de algumas propriedades visíveis a partir da interpretação em F-Logic. Pois, para um predicado 2-ário em PDDL não importa a multiplicidade. Já a UML.P com sua associação importa muito a multiplicidade. Quando os modelos são interpretados em F-Logic, na UML.P há distinção clara entre uma tupla de objetos na descrição F-Logic com o símbolo "=>" e mais de uma tupla de objetos na descrição F-Logic com o símbolo "=>>". Entretanto, como foi visto no capítulo 4, para a PDDL não faz diferença e por isso na Interpretação Correspondente, um predicado 2-ário da PDDL é interpretado com o símbolo "=>>" em F-Logic, porque garante uma abrangência maior à PDDL. Porém, para a correspondência entre a UML.P e a PDDL, foi necessário aplicar um relaxamento na associação em UML.P interpretada em F-Logic para garantir a equivalência com o predicado 2-ário em PDDL.

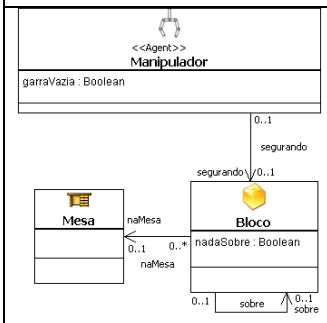
Definição 6.2 (F-átomo de associação UML.P relaxado) Todo o f-átomo obtido de modelos UML.P a partir da Interpretação Completa e Interpretação *IFR* de forma reduzida deve ser relaxado substituindo o símbolo "=>" para o símbolo "=>>".

A definição acima permite que os f-átomos de associação de classes em UML.P sejam equivalentes aos f-átomos de predicados 2-ários em PDDL. Entretanto, isso já demonstra uma perda semântica na tradução de modelos de UML.P para PDDL.

Para o desenvolvimento desse trabalho, a definição 6.2 e a SRDD juntamente das Interpretações Completa, *IFR* e Correspondente permitem mostrar para o mesmo domínio, que um modelo PDDL pode ser obtido automaticamente a partir de um modelo em UML.P, que é a proposta da ferramenta de EC, chamada itSIMPLE.

Por exemplo, na tabela a seguir, para o domínio Mundo dos Blocos, tem-se na primeira coluna o diagrama de classes da UML.P, na última coluna apresenta-se a descrição em PDDL e, em comum a essas colunas a declaração em F-Logic.

Tabela 12 – Intersecção de F-Moléculas entre UML.P e PDDL.

UML.P	F-Logic em comum às duas Linguagens	PDDL
	Manipulador :: ⊤ Bloco :: ⊤ Mesa :: ⊤ Bloco[sobre @=>> Bloco, naMesa @=>> Mesa, nadaSobre @=> booleano] Manipulador[segurando @=>> Bloco, garraVazia @=> Booleano]	(define(domain undoDosBlocos) (:requirements : typing) (:types Manipulador – object Bloco – object Mesa – object) (:predicates (sobre ?blo - Bloco ?blo1 - Bloco) (segurando ?man - Manipulador ?blo - Bloco) (naMesa ?blo - Bloco ?mes - Mesa) (garraVazia ?man - Manipulador) (nadaSobre ?blo – Bloco))

A tabela acima mostra como classes e tipos, associações, atributos e predicados são ambos interpretados facilmente em uma lógica correta e completa como a F-Logic. Dessa maneira, comprovando interpretações como essa, alcança-se o objetivo desse trabalho que é validar as modelagens em UML.P para PDDL.

Dessa maneira, demonstra-se através dos teoremas abaixo que os f-átomos da UML.P, na Interpretação *IFR* de Forma Reduzida, são equivalentes aos f-átomos da Interpretação Correspondente da PDDL em F-Logic.

Teorema 6.1 (Equivalência de f-átomo de classe UML.P e f-átomo de tipo PDDL) Considerando a SRDD e sendo γ um f-átomo *IS-A* de subclasse e classe da UML.P e λ um f-átomo *IS-A* de subtipo e tipo da PDDL, tem-se que $\vdash \gamma \leftrightarrow \lambda$, se a nomenclatura de Cl_i e Cl são idênticas às de T_i e T.

Prova

De acordo com a SRDD, a nomenclatura de classe é idêntica a de Tipo. Pela Interpretação Completa e a Interpretação IFR de forma reduzida tem-se o f-átomo γ de classe e subclasse na forma $Cl_i :: Cl$ da F-Logic. Pela Interpretação Correspondente tem-se o f-átomo λ de tipo e subtipo na forma $T_i :: T$ da F-Logic. Para que os f-átomos sejam iguais na F-Logic, utiliza-se a propriedade acíclica da IS-A de acordo com a semântica da F-estrutura I:

Se $I \vDash Cl :: T$ e $I \vDash T :: Cl$, então, $I \vDash Cl \doteq T$

Se $I \vDash Cl_i :: T_i$ e $I \vDash T_i :: Cl_i$, então, $I \vDash Cl_i \doteq T_i$

Pela propriedade da igualdade (equality) da F-logic, tem-se a simetria:

Se $I \models Cl \doteq T$, *então,* $I \models T \doteq Cl$

Se $I \models Cl_i \doteq T_i$, *então,* $I \models T_i \doteq Cl_i$

Como o f-átomo $\gamma = Cl_i :: Cl$ *e o f-átomo* $\lambda = T_i :: T$, *portanto,* $\gamma \leftrightarrow \lambda$ □

Com o teorema 6.1 tem-se que qualquer f-átomo de classe e sua subclasse em UML.P tem equivalência com um f-átomo de tipo e seu subtipo em PDDL de acordo com a SRDD.

Já no próximo teorema busca-se provar que todos os f-átomos *IS-A* para classes do modelo UML.P são equivalentes aos f-átomos *IS-A* para tipos do modelo PDDL.

Teorema 6.2 (Equivalência de Modelos para f-átomos *IS-A*) Considerando a SRDD e sendo a fórmula $\gamma_{UML.P}$, que possui todos os f-átomos *IS-A* para o modelo UML.P e a fórmula λ_{PDDL}, que possui todos os f-átomos *IS-A* para o modelo PDDL, tem-se que $\vdash \gamma_{UML.P} \leftrightarrow \lambda_{PDDL}$, se todas as classes de Cl têm nomenclatura idêntica aos tipos de T.

Prova

De acordo com a SRDD, a nomenclatura das classes do conjunto Cl é idêntica a dos tipos do conjunto T, formando uma bijetora. O fato de ser uma bijetora supõe-se que haja uma função f e sua inversa f^{-1}, *de modo que* $f : Cl \rightarrow T$ *se e somente se* $f^{-1} : T \rightarrow Cl$. *Daí, pelo teorema 6.1 para toda nomenclatura idêntica entre Cl e T demonstra-se a equivalência entre f-átomos. Como Cl e T formam uma bijeção, eles têm a mesma cardinalidade, então, os modelos são equivalentes no que tange às classes e aos tipos* $\gamma_{UML.P} \leftrightarrow \lambda_{PDDL}$ □

Com o teorema 6.2, todas as classes da UML.P e tipos da PDDL foram interpretados em F-Logic, de modo que todos eles tenham a forma de f-átomos *IS-A*.

A seguir, o teorema demonstrará a equivalência entre um f-átomo de associação de classe e um f-átomo de predicado 2-ário.

Teorema 6.3 (Equivalência de f-átomo de associação UML.P e f-átomo de predicado 2-ário PDDL) Considerando a SRDD, e sendo γ um f-átomo de associação de classe de As da UML.P e λ um f-átomo de predicado 2-ário de P^2 da PDDL, tem-se que $\vdash \gamma \leftrightarrow \lambda$, se a nomenclatura de associação de classe As_j é idêntica à de predicado 2-ário P^2_j.

Prova

De acordo com a SRDD, a nomenclatura de associação de classe é idêntica a de predicado 2-ário. Pela Interpretação Completa e a Interpretação IFR de forma reduzida tem-se o f-átomo γ *de associação de classe na forma* $Cl_i[As_j@=>Cl_l]$ *da F-Logic. Pela definição 6.2, o f-átomo deve ser relaxado em* $Cl_i[As_j@=>>Cl_l]$. *Pela Interpretação Correspondente tem-se o f-átomo* λ *de predicado 2-ário na forma* $T_i[\mathrm{P}^2_j @=>>T_l]$ *da F-Logic. Pelo teorema 6.2,* $Cl_i \doteq T_i$ *e* $Cl_l \doteq T_l$. *Como* As_j *tem a mesma nomenclatura que* P^2_j, *então,* $Cl_i[As_j@=>>Cl_l] \doteq T_i[\mathrm{P}^2_j @=>>T_l]$.

Portanto, $\gamma \leftrightarrow \lambda$ □

Com o teorema 6.3 tem-se que qualquer f-átomo de associação de classes em UML.P tem equivalência com um f-átomo de predicado 2-ário em PDDL de acordo com a SRDD.

A seguir, o teorema 6.4 demonstrará que todos os f-átomos de associação de classes do modelo UML.P e de predicados 2-ários do modelo PDDL são equivalentes.

Teorema 6.4 (Equivalência de Modelos para f-átomos de associação de classes e de predicados 2-ários) Considerando a SRDD e sendo a fórmula $\gamma_{\text{UML.P}}$, que possui todos os f-átomos de associação de classes do modelo UML.P e a fórmula λ_{PDDL}, que possui todos os f-átomos de predicados 2-ários do modelo PDDL, tem-se que $\vdash \gamma_{\text{UML.P}} \leftrightarrow \lambda_{\text{PDDL}}$, se todas as associações de classes de As têm nomenclatura idêntica aos predicados 2-ários de P^2.

Prova

De acordo com a SRDD, a nomenclatura das associações de classes do conjunto As é idêntica a dos predicados 2-ários do conjunto P^2, *formando uma bijetora. O fato de ser uma bijetora supõe-se que haja uma função f e sua inversa* f^{-1}, *de modo que* f: $\mathrm{As} \rightarrow \mathrm{P}^2$ *se e somente se* f^{-1}: $\mathrm{P}^2 \rightarrow \mathrm{As}$. *Daí, pelo teorema 6.3 para toda nomenclatura idêntica entre As e* P^2 *demonstra-se a equivalência entre f-átomos desde que pelo teorema 6.2, as classes e tipos tenham igualdade. Como As e* P^2 *formam bijeção, eles têm a mesma cardinalidade e os modelos UML.P e PDDL são equivalentes no que tange às associações de classes e aos predicados 2-ários* $\gamma_{UML.P} \leftrightarrow \lambda_{PDDL}$ □

Com o teorema 6.2 e 6.4, provou-se que dado um mesmo domínio, onde o engenheiro do conhecimento ou projetista de domínio use a mesma nomenclatura para definir seus tipos e

predicados 2-ários no modelo PDDL e as classes e associações de classes no modelo UML.P, esses modelos seriam equivalentes sempre.

A seguir, demonstrar-se-á que um atributo de classe da UML.P é equivalente a um predicado 1-ário da PDDL.

Teorema 6.5 (Equivalência de f-átomo de atributo de classe UML.P e f-átomo de predicado 1-ário PDDL) Considerando a SRDD e sendo γ um f-átomo de atributo de classe de At da UML.P e λ um f-átomo de predicado 1-ário de P^1 da PDDL, tem-se que $\vdash \gamma \leftrightarrow \lambda$, se a nomenclatura de At_i é idêntica à de P^1_i.

Prova

De acordo com a SRDD, a nomenclatura de atributo de classe é idêntica a de predicado 1-ário. Pela Interpretação Completa e a Interpretação IFR de forma reduzida tem-se o f-átomo γ de atributo de classe da UML.P na forma $Cl_i[At_i$@=>Booleano$]$ da F-Logic. Pela Interpretação Correspondente tem-se o f-átomo λ de predicado 1-ário da PDDL na forma T_i [P^1_i @=>Booleano] da F-Logic. Pelo teorema 6.2, tem-se que $Cl_i \doteq T_i$. Como At_i tem a mesma nomenclatura que P^1_i, então, $Cl_i[At_i$@=>Booleano$] \doteq T_i$ [P^1_i @=>Booleano] $\therefore \gamma \leftrightarrow \lambda$ □

Com o teorema 6.5 tem-se que qualquer f-átomo de atributo simples em UML.P tem equivalência com um f-átomo de predicado 1-ário em PDDL de acordo com a SRDD. O importante é demonstrar no teorema a seguir que todos os atributos simples de um modelo em UML.P são equivalentes aos predicados 1-ários de um modelo em PDDL.

Teorema 6.6 (Equivalência de Modelos para f-átomos de atributo de classes e de predicados 1-ários) Considerando a SRDD e sendo a fórmula $\gamma_{UML.P}$, que possui todos os f-átomos de atributos de classes do modelo UML.P e a fórmula λ_{PDDL}, que possui todos os f-átomos de predicados 1-ários do modelo PDDL, tem-se que $\vdash \gamma_{UML.P} \leftrightarrow \lambda_{PDDL}$, se todos os atributos de classes de At têm nomenclatura idêntica aos predicados 1-ários de P^1.

Prova

De acordo com a SRDD, a nomenclatura dos atributos de classes do conjunto At é idêntica a dos predicados 1-ários do conjunto P^1, formando uma bijetora. O fato de ser

uma bijetora supõe-se que haja uma função f e sua inversa f^{-1}, de modo que f: $At \rightarrow P^1$ *se e somente se* f^{-1}: $P^1 \rightarrow At$. *Daí, pelo teorema 6.5 para toda nomenclatura idêntica entre At e* P^1 *demonstra-se a equivalência entre f-átomos desde que pelo teorema 6.2, as classes e tipos tenham igualdade. Como At e* P^1 *formam bijeção, eles têm a mesma cardinalidade e os modelos UML.P e PDDL são equivalentes no que tange aos atributos de classes e aos predicados 1-ários* $\gamma_{UML.P} \leftrightarrow \lambda_{PDDL}$ □

Com o teorema 6.6, todo atributo de classe pode ser interpretado em predicado 1-ário da PDDL.

A seguir, o próximo teorema lida com a equivalência entre f-átomos de atributo com parâmetros de classe e de predicado n-ário.

Teorema 6.7 (Equivalência de f-átomo de atributo de classe UML.P com parâmetros e f-átomo de predicado n-ário PDDL) Considerando a SRDD e sendo γ um f-átomo de atributo de classe com parâmetros de Atp da UML.P e λ um f-átomo de predicado n-ário de P^n da PDDL, tem-se que $\vdash \gamma \leftrightarrow \lambda$, se a nomenclatura de Atp_i é idêntica à de P^n_i.

Prova

De acordo com a SRDD, a nomenclatura de atributo com parâmetros de classe é idêntica a de predicado n-ário. Pela Interpretação Completa e a Interpretação IFR de forma reduzida tem-se o f-átomo γ *de atributo com parâmetros de classe da UML.P na forma* $Cl_i\ [Atp_i@p_1, \ldots, p_n => Booleano]$ *da F-Logic. Pela Interpretação Correspondente tem-se o f-átomo* λ *de predicado n-ário da PDDL na forma* $T_i[P^n_i @p_1,\ldots,p_n => Booleano]$ *da F-Logic. Pelo teorema 6.2, tem-se que* $Cl_i \doteq T_i$. *Os parâmetros* $p_1,\ldots,p_n$ *são classes ou tipos, então, também pelo teorema 6.2,* $p_1 \doteq p'_1, \ldots, p_n \doteq p'_n$. *A ordem dos parâmetros importa devido à Interpretação Correspondente utilizar o TMR para interpretar predicados em F-Logic. Como o atributo de classe com parâmetros* Atp_i *tem a mesma nomenclatura que o predicado n-ário* P^n_i, *então,* $Cl_i\ [Atp_i@p_1, \ldots, p_n => Booleano] \doteq T_i[P^n_i @p_1,\ldots,p_n => Booleano]$ $\therefore \gamma \leftrightarrow \lambda$ □

Com o teorema 6.7, sabe-se que um f-átomo de atributo com parâmetros de classe é equivalente a um f-átomo de predicado n-ário. O próximo teorema demonstra que todos esses atributos são equivalentes aos predicados n-ários de acordo com sua demonstração.

Teorema 6.8 (Equivalência de Modelos para f-átomos de atributo com parâmetros de classes e de predicados n-ários) Considerando a SRDD e sendo a f-fórmula $\gamma_{UML.P}$, que possui todos os f-átomos de atributos com parâmetros de classes do modelo UML.P e a f-fórmula λ_{PDDL}, que possui todos os f-átomos de predicados n-ários do modelo PDDL, tem-se que $\gamma_{UML.P} \leftrightarrow \lambda_{PDDL}$, se todos os atributos com parâmetros de classes de Atp têm nomenclatura idêntica aos predicados n-ários de P^n.

Prova

De acordo com a SRDD, a nomenclatura dos atributos com parâmetros de classes do conjunto Atp é idêntica a dos predicados n-ários do conjunto P^n*, formando uma bijetora. O fato de ser uma bijetora supõe-se que haja uma função f e sua inversa* f^{-1}*, de modo que f*: Atp$\rightarrow P^n$ *se e somente se* f^{-1}: $P^n \rightarrow$ Atp. *Daí, pelo teorema 6.7 para toda nomenclatura idêntica entre Atp e* P^n *demonstra-se a equivalência entre f-átomos desde que pelo teorema 6.2, as classes e tipos tenham igualdade, assim como os parâmetros que também são classes e têm de possuir mesma ordem. Como Atp e* P^n *formam bijeção, eles têm a mesma cardinalidade e os modelos UML.P e PDDL são equivalentes no que tange aos atributos com parâmetros de classes e aos predicados n-ários* $\gamma_{UML.P} \leftrightarrow \lambda_{PDDL}$ □

Com o teorema 6.8 tem-se que qualquer f-átomo de atributo com parâmetros em UML.P tem equivalência com um f-átomo de predicado n-ário em PDDL de acordo com a SRDD.

Os teoremas 6.1-8 demonstram que os f-átomos dos modelos em UML.P e PDDL de acordo com a SRDD são equivalentes.

(Exemplo dos Teoremas 6.1-8) No domínio Mundo dos Blocos, pode-se exemplificar o teorema da seguinte maneira:

1. Para $\mathbf{Cl_1 :: Cl \leftrightarrow T_1 :: T}$:
   a. $Cl_1 \doteq T_1 \doteq$ Manipulador e Cl $\doteq$ T, que são idênticos e tratam-se do conjunto universal dos tipos ou classes, de modo que Manipulador $\leftrightarrow$ Manipulador;
2. Para $\mathbf{Cl_1[As_1@{=>}Cl_2] \leftrightarrow T_1[}\, P_1^2\, \mathbf{@{=>}T_2]}$:

a. $Cl_1 \doteq T_1 \doteq$ Manipulador, $Cl_2 \doteq T_2 \doteq$ Bloco, de modo que As_1 e P_1^2 sejam o método "Segurando". Para que a descrição do método de conjunto de valores seja: Manipulador[Segurando@=>>Bloco] ↔ Manipulador[Segurando@=>>Bloco];

3. Para **Cl_1[At_1@=> booleano] ↔ T_1[** P_1^1 **@=> Booleano]**:

   a. $Cl_1 \doteq T_1 \doteq$ Manipulador, de modo que At_1 e P_1^1 sejam o método "garraVazia", e Booleano é o tipo primitivo para verdadeiro ou falso. Assim, a descrição de método escalar seja:

      Manipulador[garraVazia@=>Booleano] ↔ Manipulador[garraVazia@=>Booleano].

4. Para **Cl_1[Atp_1@ Cl_2, Cl_3=> booleano] ↔ T_1[** P_1^n **@T_2, T_3=> booleano]**:

   a. $Cl_1 \doteq T_1 \doteq$ Manipulador, $Cl_2 \doteq T_2 \doteq$ Bloco e $Cl_3 \doteq T_3 \doteq$ Mesa, de modo que Atp_1 e P_1^n sejam o método "segurando", e booleano é o tipo primitivo para verdadeiro ou falso. Assim, a descrição de método escalar é Manipulador [segurando@Bloco, Mesa => booleano] ↔ Manipulador [segurando@Bloco, Mesa =>booleano].

Como houve a demonstração de que os Modelos UML.P e PDDL possuem f-átomos de acordo com a SRDD, *IFR* e Interpretação Correspondente, de modo que estes f-átomos sejam equivalentes. Isso leva a provar que uma fórmula que representa a conjunção de f-átomos tanto do modelo UML.P quanto do modelo PDDL sejam equivalentes também. Dessa forma, através do teorema abaixo busca-se provar que um modelo UML.P na *IFR* e SRDD é equivalente a outro modelo PDDL na Interpretação Correspondente e SRDD.

Teorema 6.9 (Equivalência de modelos UML.P e PDDL) Considerando a SRDD, duas f-fórmulas possuem todos os f-átomos dos modelos UML.P e PDDL, onde ϕ é a f-fórmula decorrente da Interpretação *IFR* de forma reduzida da UML.P e δ é a f-fórmula decorrente da Interpretação Correspondente da PDDL, que são consideradas equivalentes $\phi \leftrightarrow \delta$, se as classes Cl são equivalentes aos tipos T, se os atributos de classes At são equivalentes aos predicados 1-ários P^1, se os atributos com parâmetros de classes Atp são equivalentes aos predicados n-ários P^n e se as associações de classes As são equivalentes aos predicados 2-ários P^2.

Prova

A f-fórmla ϕ possui conjunções de todos os f-átomos do modelo UML.P e a f-fórmula δ possui conjunções de todos os f-átomos do modelo PDDL. As f-fórmulas ϕ e δ seguem a

restrição da SRDD e como possuem conjunções de f-átomos não importa a ordem deles dentro da f-fórmula devido à lei da comutação. Assim, Pelo teorema 6.2 tem-se que todas as classes Cl são equivalentes aos tipos T. Pelo teorema 6.6 tem-se que todos os atributos de classes At são equivalentes aos predicados 1-ários P^1. *Pelo teorema 6.8 tem-se que todos os atributos com parâmetros de classes Atp são equivalentes aos predicados n-ários* P^n. *Pelo teorema 6.4 tem-se que todas as associações de classes As são equivalentes aos predicados 2-ários* P^2. *Portanto, há equivalência de modelos entre UML.P e PDDL de acordo com a SRDD, de modo que* $\phi \leftrightarrow \delta$ □

(Exemplo do Teorema 6.9) Aproveitando o exemplo dos teoremas 6.1-8 para exemplificar o teorema 6.9, tem-se as seguintes f-fórmulas do Mundo dos Blocos:

Os teoremas 6.1-8 expõem que as f-fórmulas ϕ e δ possuem f-átomos $(\phi_1, \phi_2, ..., \phi_n) \leftrightarrow (\delta_1, \delta_2, ..., \delta_n)$ e que podem ser demonstrados para cada f-átomo $(\phi_n \leftrightarrow \delta_1, \phi_{n-1} \leftrightarrow \delta_2, ..., \phi_2 \leftrightarrow \delta_{n-1}, \phi_1 \leftrightarrow \delta_n)$, entretanto, a ordem dos f-átomos nas f-fórmulas não se trata de algo que possa ser intuitivo, isto é, $\phi_{20} \leftrightarrow \delta_5$. Mas pela lei da comutação garante que as f-fórmulas são idênticas. O exemplo abaixo para $\phi \leftrightarrow \delta$ demonstra os f-átomos com ordem diferente dentro de cada f-fórmula. Entretanto, ambos conjuntos de f-fórmulas ϕ e δ possuem os mesmos f-átomos:

ϕ = (Bloco[sobre@=>> Bloco, nadaSobre@=> Booleano, naMesa@=>> Mesa], Manipulador[garraVazia@=> Booleano, segurando@=>> Bloco]) $\leftrightarrow$ δ = (Bloco[nadaSobre@=> Booleano, naMesa@=>> Mesa, sobre@=>> Bloco], Manipulador[segurando@=>> Bloco, garraVazia@=> Booleano]). De tal modo que as f-fórmulas sejam $\phi \leftrightarrow \delta$.

Com o teorema 6.9 demonstrou-se que o modelo UML.P de acordo com a *IFR* e a SRDD é equivalente ao modelo PDDL de acordo com a Interpretação Correspondente e a SRDD. Entretanto, na Interpretação Completa do modelo UML.P há muitos outros f-átomos (GOMES, 2008). Por isso, no teorema 6.10 abaixo demonstra-se que um modelo PDDL é conseqüência lógica de um modelo UML.P, mas o contrário não é verdadeiro. Essa demonstração de que o modelo PDDL é conseqüência lógica de um modelo UML.P só é possível porque o teorema 11.3 em (KIFER; LAUSEN; WU, 1995) demonstra que a dedução F-Logic é correta utilizando suas propriedades e regras de inferência.

Teorema 6.10 (Conseqüência Lógica da Interpretação) O modelo Λ da PDDL de acordo com a SRDD e a Interpretação Correspondente sobre tipos e predicados é conseqüência lógica na teoria da F-Logic de um modelo Γ em UML.P sobre o diagrama de classes a partir da Interpretação Completa, tal que $\Gamma \models_F \Lambda$ para o mesmo Domínio α.

Prova

Sendo a SRDD e a Interpretação Correspondente utilizadas para o modelo Λ apenas e a Interpretação Completa para o modelo Γ, então, tem-se os modelos interpretados em f-átomos. Como o modelo Γ em UML.P possui mais f-átomos que o modelo Λ em PDDL e o teorema 6.9 prova que o modelo UML.P e PDDL são equivalentes de acordo com a SRDD e a Interpretação IFR de forma reduzida e, ainda, pelo teorema 5.1 prova-se que a Interpretação IFR de forma reduzida $\subseteq$ Interpretação Completa, tem-se que $\Lambda \subseteq \Gamma$ e, logo, $\Gamma \models_F \Lambda$ □

Com os teoremas apresentados nesse capítulo, essa dissertação demonstrou que é factível a interpretação de modelos da UML.P em modelos da PDDL de acordo com a suposição SRDD.

6.2. Exemplos da Equivalência entre Modelos UML.P e PDDL utilizando a F-Logic

Esta seção demonstra alguns exemplos de modelos em UML.P e PDDL considerando as definições e teoremas dos capítulos 3, 4, 5 e 6.

A seguir se demonstrará quatro domínios de Planejamento Automático: Satélite, Jantar dos Filósofos, Logística e *Zeno Travel* de acordo com o propósito dessa dissertação.

Para o domínio Satélite, segue seu diagrama de classes com as associações, atributos de classes e classes:

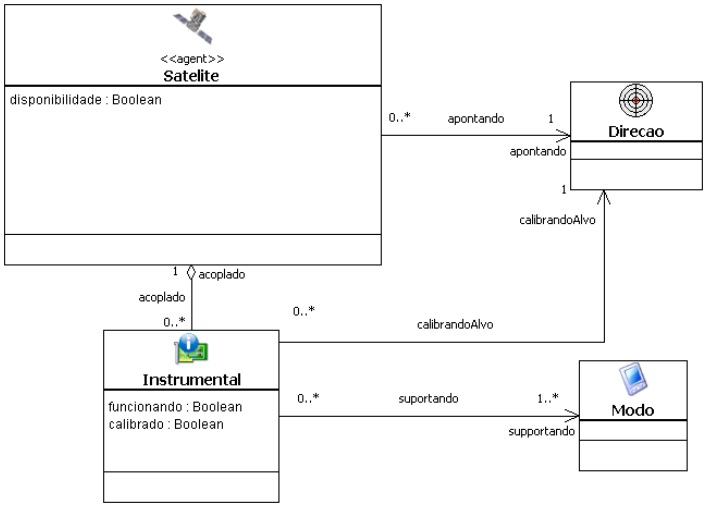


Figura 29 – Diagrama de Classes para o domínio Satélite.

O diagrama de classes UML.P para o domínio Satélite possui 4 (quatro) classes, 4 (quatro) associações de classes e 3 (três) atributos de classes. As classes, seus atributos e suas associações fazem parte da suposição SRDD. Dessa forma, esses são os elementos que possuem equivalência com os elementos PDDL de acordo com a Interpretação Correspondente em F-Logic.

Para o diagrama de classes do domínio Satélite, abaixo segue a tabela com a descrição F-Logic em comum à descrição PDDL e ao diagrama de classes.

Tabela 13 – Equivalência entre UML.P e PDDL para o domínio Satélite

F-Logic – Classes e Métodos	PDDL – Tipos e Predicados

`Satelite :: ClasseFund.`
`Instrumental :: ClasseFund.`
`Modo :: ClasseFund.`
`Direcao :: ClasseFund.`
 	
`Instrumental[acoplado@=>>Satelite;`
`calibrandoAlvo@=>>Direcao;`
`suportando=>>Modo;`
`funcionando@=>Booleano;`
`calibrado@=>Booleano].`
 	
`Satelite[disponibilidade@=>Booleano;`
`apontando=>>Direcao].`
	`(define (domain Satellite_Domain)`
	`(:requirements :typing)`
	`(:types`
	`Satelite - T`
	`Instrumental - T`
	`Modo - T`
	`Direcao - T)`
	`(:predicates`
	`(acoplado ?ins - Instrumental ?sat - Satelite)`
	`(apontando ?sat - Satelite ?dir - Direcao)`
	`(calibrandoAlvo ?ins - Instrumental ?dir - Direcao)`
	`(suportando ?ins - Instrumental ?mod - Modo)`
	`(disponibilidade ?sat - Satelite)`
	`(funcionando ?ins - Instrumental)`
	`(calibrado ?ins - Instrumental)))`

Na tabela acima, os elementos PDDL, que são os tipos, predicados 1-ários e predicados 2-ários equivalentes ao elementos da UML.P de acordo com a SRDD, advém da Interpretação Correspondente de ambas com a F-Logic. Portanto, as classes e métodos da F-Logic são iguais para ambas.

O Próximo exemplo é o domínio Jantar de Filósofos, que se encontra no diagrama de classes com as associações, atributos de classes e classes:

Figura 30 – Diagrama de Classes para o domínio Jantar dos Filósofos.

Da mesma maneira demonstrada para o domínio Satélite, o diagrama de classes para o Jantar de Filósofos foi projetado. Percebe-se que ele possui os mesmos elementos: classes, atributos e associações de classes.

Para o diagrama de classes acima, abaixo segue a tabela com a descrição F-Logic em comum à descrição PDDL e ao diagrama de classes.

Tabela 14 – Equivalência entre UML.P e PDDL para o domínio Jantar dos Filósofos

F-Logic – Classes e Métodos	PDDL – Tipos e Predicados

`Filosofo :: ClasseFund.`
`Garfo :: ClasseFund.`
 	
`Filosofo[direito@=>>Garfo;`
`esquerdo@=>>Garfo; segurando=>>Garfo;`
`faminto@=>Booleano;`
`temEsquerdo@=>Booleano;`
`temDireito@=>Booleano;`
`comendo@=>Booleano;`
`pensando@=>Booleano].`
	`(define (domain Dining_Philosophers)`
	`(:requirements :typing)`
	`(:types`
	`Filosofo - T`
	`Garfo - T)`
	`(:predicates`
	`(direito ?fil - Filosofo ?gar - Garfo)`
	`(esquerdo ?fil - Filosofo ?gar - Garfo)`
	`(segurando ?fil - Filosofo ?gar - Garfo)`
	`(faminto ?fil - Filosofo)`
	`(temEsquerdo ?fil - Filosofo)`
	`(comendo ?fil - Filosofo)`
	`(pensando ?fil - Filosofo)`
	`(temDireito ?fil - Filosofo)`
	`)`

Percebe-se também a partir da tabela, as mesmas interpretações em F-Logic no que tange aos seus Métodos e Classes de acordo com as definições e a suposição SRDD. Pode-se também verificar a partir dos teoremas do capítulo 6, a equivalência dos f-átomos que foram interpretados da UML.P e da PDDL em uma descrição comum em F-Logic.

No próximo domínio Logística, se encontra o diagrama de classes com as associações, atributos de classes, classes e subclasses. Também existe a mesma orientação para a interpretação.

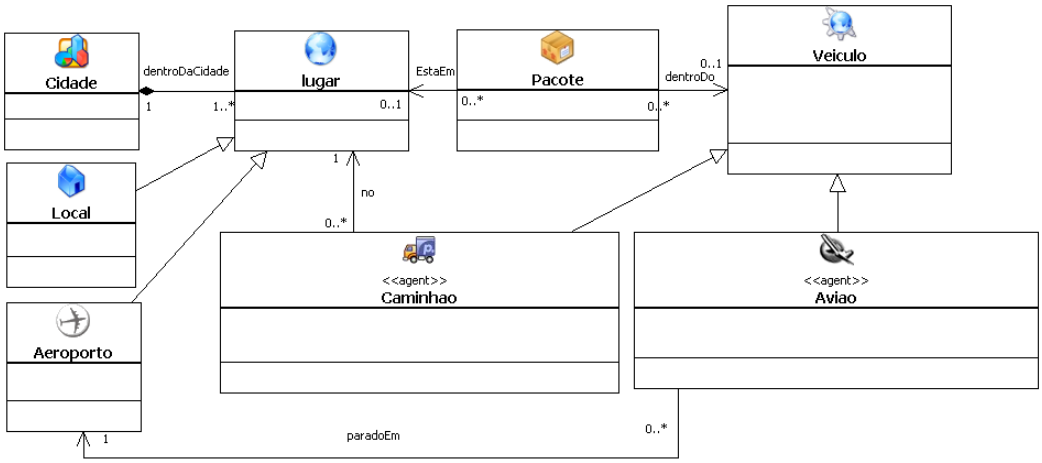


Figura 31 – Diagrama de Classes para o domínio Logística.

O diagrama de classes projetado para o domínio Logística possui como diferença aqui os relacionamentos de Herança. Por exemplo,as classes Caminhão e Avião são do tipo Veículo e as classes Local e Aeroporto são do tipo Lugar.

Para o diagrama de classes acima, abaixo segue a tabela com a descrição F-Logic em comum à descrição PDDL e ao diagrama de classes.

Tabela 15 – Equivalência entre UML.P e PDDL para o domínio Logística

F-Logic – Classes e Métodos	PDDL – Tipos e Predicados
Veiculo :: ClasseFund. Caminhao :: Veiculo. Aviao :: Veiculo. Lugar :: ClasseFund. Local :: Lugar. Aeroporto :: Lugar. Pacote :: ClasseFund. Cidade :: ClasseFund. Pacote[dentroDo@=>>Veiculo; estaEM@=>>Lugar]. Lugar[dentroDaCidade=>>Cidade]. Caminhão[no@=>>Lugar]. Avião[paradoEm@=>>Aeroporto].	(define (domain Logistic_Domain) (:requirements :typing) (:types Veiculo - T Caminhao - Veiculo Aviao - Veiculo lugar - T Local - lugar Aeroporto - lugar Pacote - T Cidade - T) (:predicates (dentroDo ?pac - Pacote ?vei - Veiculo) (EstaEm ?pac - Pacote ?lug - lugar) (dentroDaCidade ?lug - lugar ?cid - Cidade)

	(no ?cam - Caminhao ?lug - lugar) (paradoEm ?avi - Aviao ?aer - Aeroporto))

Percebe-se que a partir das definições do capítulo 4 é possível fazer a Interpretação Correspondente em F-Logic. E a partir da interpretação *IFR* de Forma Reduzida do capítulo 5 tem-se também os f-átomos da F-Logic. Para o mesmo domínio de acordo com a SRDD, os modelos UML.P e a PDDL possui equivalência.

Para o último domínio *Zeno Travel* abaixo, se encontra o diagrama de classes com as associações, atributos de classes e classes:

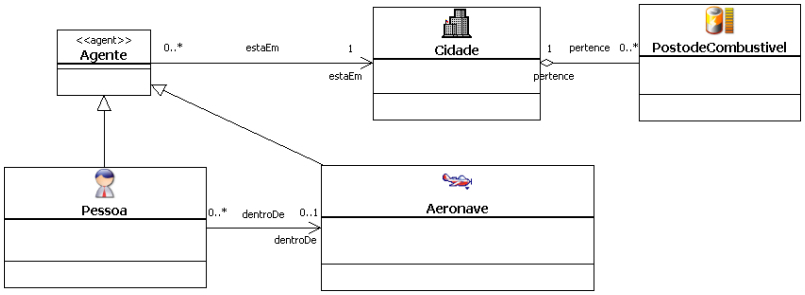


Figura 32 – Diagrama de Classes para o domínio *Zeno Travel*.

Para o diagrama de classes acima, abaixo segue a tabela com a descrição F-Logic em comum à descrição PDDL e ao diagrama de classes.

Tabela 16 – Equivalência entre UML.P e PDDL para o domínio Zeno Travel

F-Logic – Classes e Métodos	PDDL – Tipos e Predicados
Cidade :: ClasseFund. Agente :: ClasseFund. Pessoa :: Agente. Aeronave :: Agente. PostodeCombustivel :: ClasseFund. Agente[estaEm@=>>Cidade]. Pessoa[dentroDe@=>>Aeronave]. PostodeCombustivel[pertence=>>Cidade].	(define (domain Zeno_Travel_Domain) (:requirements :typing) (:types Cidade - T Agente - T Pessoa - Agente Aeronave - Agente PostodeCombustivel - T) (:predicates (estaEm ?age - Agente ?cid - Cidade) (dentroDe ?pes - Pessoa ?aer - Aeronave) (pertence ?pos - PostodeCombustivel ?cid - Cidade))

Esses exemplos demonstram a equivalência de modelos UML.P e PDDL no que tange à suposição SRDD, a Interpretação Correspondente e a interpretação *IFR* da Forma Reduzida de modo que os F-átomos da F-Logic são equivalentes.

Próxima seção, serão apresentadas as perdas ao interpretar o modelo UML.P em modelo PDDL.

6.3. Perdas Semânticas na Interpretação do modelo UML.P em modelo PDDL

As perdas semânticas na interpretação do modelo UML.P em modelo PDDL são inerentes dos f-átomos da Interpretação Completa (GOMES, 2008) que não fazem parte da Interpretação *IFR* de forma reduzida considerando as restrições da SRDD. Ou seja, os f-átomos sobre classes, associação de classes e atributos de classes. Na tabela abaixo, a primeira coluna é referente ao f-átomo não interpretado em PDDL e a segunda coluna trata do seu significado e, conseqüentemente, sua perda semântica ao não ser interpretado em PDDL.

Tabela 17 – F-átomos perdidos na Interpretação em modelo PDDL

F-átomo da Interpretação Completa não utizado na Interpretação *IFR* de forma reduzida	Significado da Interpretação de classe, associação de classes e atributos de classe em F-Logic
`ester_Agente:tipoEstereotipo` `classe[ estereotipo *-> Ester_Agente]`	Esses dois f-átomos representam que dada "classe" é uma agente no modelo, isto é, a classe que provoca mudanças nos estados do mundo com seus operadores. Na PDDL, não estão claramente representadas que certas ações são de alguns Agentes. Desse modo, não tem sido interessante saber no modelo quem são os objetos agentes.
`ester_Utilitario:tipoEstereotipo` `classe[ estereotipo *-> Ester_Utilitario]`	Esses dois f-átomos representam que dada "classe" é uma classe utilitária. Não provoca mudanças nos estados do mundo. Como explicado acima, se não há Agentes, também não se mostra interessante na PDDL saber os objetos que são passivos no modelo.

`Classe[attr@limConjunto => inteiro;`
`attr@lim_Inferior *-> 1;`
`attr@lim_Superior *-> 1]`
	Esses três f-átomos representam a multiplicidade de um atributo de classe. Isso pode ser tratado na PDDL a partir do elemento (:Constraints com algum esforço.

`Classe[attr@caracteristica => mutabilidade;` `attr@atrib_carac *-> mut_Alteravel]`	Esses dois f-átomos representam a mutabilidade de um atributo. Na PDDL não há esse tipo de informação no modelo.
`classeA[associação_classeB => mutabilidade;` `associação_classeB *-> <mutabilidade>]`	Esses dois f-átomos representam como é a associação de classe: estática ou alterável. Na PDDL não há esse tipo de informação no modelo.

`ClasseA[associação_classeB_fimB@limConjunto => inteiro;`
`associação_classeB_fimB@lim_Inferior *-> <Vs>;`
`associação_classeB_fimB@lim_Superior *-> <Vt>]`
	Esses três f-átomos representam como é exatamente a multiplicidade da associação de classes A e B pelos valores <Vs> e <Vt>. Isso pode ser tratado na PDDL a partir do elemento (:Constraints com algum esforço.

`ClasseA[`
`associação_classeB_fimB@caracteristica => assocTipoEx;`
`associação_classeB_fimB@assoc_TipoEx *-> <AssociaçãoTipo>]`
	Esses dois f-átomos representam como é exatamente o tipo de associação em cada extremidade da associação. Ou seja, a ClasseA tem uma extremidade chamada

	fimB, que é a extremidade de contato com a classe B, cuja <AssociaçãoTipo> pode ser simples, ou agregação, ou composição. Essa perda semântica não é relevante para o modelo em PDDL.
`ClasseA[associação@=>ClasseB] relaxado em ClasseA[associação@=>>ClasseB]`	O relaxamento da associação de "=>" em "=>>" ocasiona uma perda semântica porque na PDDL isso não faz diferença. Esse relaxamento permite tratar mais de uma tupla de objetos sempre.

Os f-átomos que foram perdidos na interpretação em PDDL podem servir de motivação para enriquecer a descrição em PDDL e, conseqüentemente, otimizar os algoritmos dos planejadores. Já a UML.P pode fornecer com sua descrição em F-Logic uma base de conhecimento rica em domínios e problemas de planejamento automático, de modo que o engenheiro do conhecimento ou projetista de domínio pudesse obter informação quando estivesse projetando domínios ou apenas precisando de informações sobre algum domínio.

Próximo capítulo serão apresentadas as conclusões desse trabalho.

CAPÍTULO 7

7. CONCLUSÃO

Nesse trabalho foi possível demonstrar a equivalência de modelos UML.P e PDDL de acordo com a suposição SRDD – Suposição Restritiva de Declaração de Domínios, que através da utilização da F-Logic houve como interpretar ambos modelos em F-átomos e pelos teoremas aqui apresentados demonstrou-se a equivalência entre esses f-átomos. Além disso, sabe-se que tanto a PDDL como a UML.P são linguagens semiformais e caso fosse feito apenas com essas linguagens essa formalização, teríamos resultados ainda com essa deficiência.

Portanto, a partir da modelagem do diagrama de classes da UML.P pode-se interpretar tipos e predicados da PDDL através da formalização da F-Logic. Isso garante que qualquer domínio de Planejamento Automático possa ser modelado pelo itSIMPLE através da UML.P e, depois, ter a descrição em PDDL.

7.1. Trabalhos Futuros

Nesse trabalho conjecturou-se a mesma abordagem para as funções da PDDL. Para as funções pode-se utilizar dois tipos primitivos da UML.P: o tipo “int” que significa inteiro e o tipo “float”, ponto flutuante. O inteiro para cálculos discretos e o ponto flutuante para cálculos contínuos. Dessa maneira, utilizando a F-Logic pode-se ter F-átomos para as funções em PDDL tais como: Classe[ScalM@=>int] ou Classe[ScalM@=>float].

Já para tratar os operadores da UML.P e as ações da PDDL pode-se utilizar os operadores da UML.P já interpretados em F-Logic na Interpretação Completa (GOMES, 2008) e uma extensão da F-Logic para a Lógica de Transações (BONNER; KIFER, 1993) (TONIDANDEL, 2003). Essa extensão é apresentada em (KIFER; LAUSEN; WU, 1995) como Transaction F-Logic, que possui em seu oráculo de dados a descrição em F-Logic. Dessa maneira, as ações de Planejamento Automático podem ser definidas em cláusulas de Horn utilizando a Lógica de Primeira Ordem para Objetos. Além disso, manter essas descrições em Transaction F-Logic como Base de Conhecimento de Planejamento Automático.

REFERÊNCIAS BIBLIOGRÁFICAS

ARISTÓTELES. *Órganon*. Trad. Edson Bini. Bauru: Edipro, 2005.

BENDER, E.A. *Mathematical Methods in Artificial Intelligence*. IEEE Computer Society Press, 1996.

BITTENCOURT, G. *Inteligência Artificial – Ferramentas e Teorias*. UFSC. 3ª Edição. 2006.

BONNER, A. J.; KIFER, M. *Transaction Logic programming*. Technical Report CSRI-323, Computer Systems Research Institute, university of Toronto, 1993.

BOOCH, Grady; JACOBSON, Ivar; RUMBAUGH, James. *The Unified Modeling Language User Guide*. Addison-Wesley, 1999.

COPPIN, B. *Artificial Intelligence Illuminated*. Sudbury, MA, Jones and Bartlett, 2004.

DECKER, S. *On domain-specific declarative knowledge representation and database languages*. In Proc. of the 5th International KRDB Workshop, pages 9.1–9.7, 1998.

EDELKAMP, S.; MEHLER, T. *Knowledge Acquisition and Knowledge Engineering in the ModPlan Workbench*. Proceedings of the International Planning Competition. International Conference on Automated Planning and Scheduling. Monterey, pages 26.33, 2005.

EDELKAMP, S.; JABBAR, S.; LAFUENTE, A. L. *Action Planning for Graph Transition Systems*. Verification and Validation of Model-Based Planning and Scheduling Systems (VVPS). Monterey, AAAI Press, pages 58-66, 2005.

FIKES, R.E.; NILSSON, N.J. *STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving*. Technical Note 43r. AI Center, SRI International, 333 Ravenswood Ave, Menlo Park, CA 94025, May 1971.

FOX, M.; LONG, D. 2003. *PDDL 2.1: An Extension to PDDL for Expressing Temporal Planning Domains*. Journal of Artificial Intelligence Research 20:61-124.

GOMES, M. L. *EXTRAÇÃO DE CONHECIMENTO DE DOMÍNIOS DE PLANEJAMENTO DESCRITOS EM UML.P*. 2008. Monografia (Mestrado em Engenharia Elétrica).

GOMES, M. L.; VAQUERO, T. S.; SILVA, J. R.; TONIDANDEL, F. *Extracting State Constraints from UML Planning Models*. In: ICAPS-08 Workshop on Knowledge Engineering for Planning and Scheduling (KEPS). Sydney, Australia. 2008.

KIFER, M.; LAUSEN, G.; WU, J. *Logical Foundations of object-oriented and frame-based languages*. Journal of ACM, v. 42, p. 741-843, May 1995.

KOWALSKI, R.; SERGOT, M. *A logic-based calculus of events.* New Generation Computing, 4(1), p. 67-95, 1986.

LIFSCHITZ, V. *On the Semantics of STRIPS*, in *Workshop Reasoning about Actions and Plans*, s.1., 1987. ***Proceedings.*** s.n. 1987, p.1-9.

LIFSCHITZ, V; REN, W. *The Semantics of Variables in Action Descriptions*, in *Proceedings of the Twenty-Second National Conference on Artificial Intelligence (AAAI)*, 2007. 1025-1030.

LÍŠKA, M. *Extensional and intensional semantics of pUML objects*. In: IIT.SRC 2005 – Informatics and Information Technology Student Research Conference. Bratislava, April 2005, J. Minárová (supervisor), – pp. 167–174.

LUDÄSCHER, B.; YANG, G.; KIFER, M. *Flora: The Secret of Object-Oriented Programming*. Technical Report, SUNY at Stony Brook, 1999.

LUGER, George F. *Artificial Intelligence: Structures and Strategies for Complex Problem Solving* (5th Edition). Addison-Wesley, 2004.

MCCARTHY, J.; HAYES, P. J. *Some Philosophical Problems from the Standpoint of Artificial Intelligence*. In *B. Meltzer and D. Michie, editors, Machine Intelligence 4, p. 463-502*. Edinburgh University Press, 1969.

MCCLUSKEY T. L. *The Knowledge Engineering for Planning Roadmap in the PLANET* final report to the EC, November 2000.

MCCLUSKEY T. L. *Knowledge Engineering: Issues for the AI Planning Community. Proceedings of the AIPS-2002 Workshop on Knowledge Engineering Tools and Techniques for AI Planning, Toulouse, France, April 2002.*

MCCLUSKEY T. L. *PDDL: A Language with a Purpose?*, Proceedings of the ICAPS-03 workshop on PDDL, International Conference on Automated Planning and Scheduling, June, 2003 (ICAPS'03).

MCDERMOTT, D. & the AIPS-98 Planning Competition Committee 1998. *PDDL – The Planning Domain Definition Language*. Tech. rep., Available at: www.cs.yale.edu/homes/dvm.

MCDERMOTT, D. *PDDL2.1 - The Art of the Possible?* Commentary on Fox and Long. Journal of Artificial Intelligence Research 20:145 – 148, 2003.

NILSSON, N. J. *Artificial Intelligence : A New Synthesis*. Morgan Kaufmann; 1st edition,1998.

OMG – Object Management Group, 2001. Unified Modeling Language Specification: version 2.1. URL: www.omg.org/uml.

PEDNAULT, E. *ADL: Exploring the middle ground between STRIPS and the situation calculus*. In *International Conference on Principles of Knowledge Representation and Reasoning,* 1., Toronto, 1989. *Proceedings***.** s.n. 1989, p.324-332.

PEDNAULT, E. *ADL and the state-transition model of action*. Journal of Logic and Computation, 4:467–512, 1994.

RAMALHO, F., ROBIN, J. and SCHIEL, U. *Concurrent Transaction Frame Logic Formal Semantics for UML Activity and Class Diagrams*. Electronic Notes in Theoretical Computer Science, Volume 95, 17 May 2004 Edited by A. Cavalcanti and P. Machado, Elsevier, 2004.

RAMALHO, F., ROBIN, J. *Mapping UML Class Diagrams to Object-Oriented Logic Programs for Formal Model-Driven Development*. 3rd Workshop in Software Model Engineering, 11 Oct 2004.

RUSSELL, S.; NORVIG, P. *Inteligência Artificial*, ed Campus, 2004.

SCHREIBER G., AKKERMANS H., ANJEWIERDEN A., DE HOOG R., SHADBOLT N., VAN de Velde W., WIELINGA B. *Knowledge engineering and management. The CommonKADS methodology.* MIT Press, 2000.

SIMPSON R. M.; MCCLUSKEY T. L.; ZHAO W.; AYLETT R.S.; DONIAT C. *An Integrated Graphical Tool to support Knowledge Engineering in AI Planning.* Proceedings of the European Conference on Planning, Toledo, Spain, September 2001.

SOWA, J. F. (2006). *Semantic networks*. (Revised and extended version of an article originally written for the Encyclopedia of Artificial Intelligence, edited by Stuart C. Shapiro, Wiley, 1987, second edition, 1992). http://www.jfsowa.com/pubs/semnet.htm

TONIDANDEL, F. *Desenvolvimento e implementação de um sistema de planejamento baseado em casos*. 2003. 188 p. Tese (Doutorado) - ESC POLITECNICA, Universidade de São Paulo, São Paulo, 2003.

TONIDANDEL F., VAQUERO T.S., and SILVA J.R. *Reading PDDL, Writing an Object-Oriented Model*. In Leture Notes in Computer Science. International Joint Conference on AI IBERAMIA/SBIA. Springer-Verlag. 2006.

UDO, M. ; VAQUERO, T. S. ; SILVA, J. R. ; TONIDANDEL, F. *Lean Software Development Domain*. In: 18th International Conference on Automated Planning and Scheduling (ICAPS), 2008, Sydney. Australia. ICAPS-08 Workshop on Scheduling and Planning Applications (SPARK), 2008.

VAQUERO, T. S.; TONIDANDEL, F.; SILVA, J. R. 2005. *The itSimple tool for Modeling Planning Domains*. ICAPS 2005 Competition ICKEP, Monterey, California, USA.

VAQUERO, T. S.; TONIDANDEL, F.; BARROS, L. N. de; SILVA, J. R. 2006. *On the Use of UML.P for Modeling a Real Application to the Planning Problem.*

VAQUERO, T. S. *itSIMPLE: Ambiente Integrado de Modelagem e Análise de Domínios de Planejamento Automático.* 2007. Monografia (Mestrado em Engenharia Mecatrônica).

VAQUERO, T. S.; TONIDANDEL, F.; BARROS, L.N. ; SILVA, J. R. *Modeling a Real Application as a Planning Problem by using UML.P*. In: VIII SBAI - Simpósio Brasileiro de Automação Inteligente, 2007, Florianópolis, Brazil, 2007. Full Paper.

VAQUERO, T. S.; TONIDANDEL, F.; SILVA, J. R. *Suggestions for the Knowledge Engineering Competition.* In: ICAPS-08 Workshop on Knowledge Engineering for Planning and Scheduling (KEPS). Sydney, Australia. 2008.

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo