ROSIMARCI P. TONACO-BASBAUM

MODELAGEM HIERÁRQUICA E ANÁLISE DE REQUISITOS DE PROBLEMAS REAIS EM PLANEJAMENTO AUTOMÁTICO

São Paulo

2015

ROSIMARCI P. TONACO-BASBAUM

MODELAGEM HIERÁRQUICA E ANÁLISE DE REQUISITOS DE PROBLEMAS REAIS EM PLANEJAMENTO AUTOMÁTICO

Tese apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do título de Doutor em Engenharia Mecânica

Área de concentração:
Engenharia de Controle e Automação Mecânica

Orientador:
Prof. Dr. José Reinaldo Silva

São Paulo
2015

Este exemplar foi revisado e corrigido em relação à versão original, sob responsabilidade única do autor e com a anuência de seu orientador.

São Paulo, _____ de _____ de _____

Assinatura do autor: _____

Assinatura do orientador: _____

Catalogação-na-publicação

TONACO-BASBAUM, ROSIMARCI
MODELAGEM HIERÁRQUICA E ANÁLISE DE REQUISITOS DE PROBLEMAS REAIS EM PLANEJAMENTO AUTOMÁTICO / R. TONACO BASBAUM -- versão corr. -- São Paulo, 2015.
119 p.

Tese (Doutorado) - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia Mecatrônica e de Sistemas Mecânicos.

1.INTELIGÊNCIA ARTIFICIAL (PLANEJAMENTO;AUTOMAÇÃO) 2.ANÁLISE DE REQUISITOS 3.REDES DE PETRI 4.ADMINISTRAÇÃO DE PROJETOS (PROCESSOS) 5.UML I.Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia Mecatrônica e de Sistemas Mecânicos II.t.

Aos meus adoráveis garotos, Arthur, Miguel e Antônio, fontes da minha alegria e inspiração.

Agradecimentos

Ao d-lab e ao Departamento de Engenharia Mecânia da Escola Politécnica da Universidade de São Paulo pelos recursos físicos fornecidos.

Ao meu orientador prof. Dr. José Reinaldo Silva, pela orientação e paciência.

Aos professores que formam o corpo docente do programa de pós-graduação em Engenharia Mecânica.

Ao prof. Dr. Flávio Soares Correa da Silva pela convivência e ensinamentos.

A todos os amigos e colegas do d-lab. Em particular San Pedro e Javier pela gentileza e interminável boa vontade em compartilhar conhecimento.

Aos amigos e familiares que de uma forma ou de outra se fizeram presentes nessa jornada acadêmica.

Aos meus pais, Olavo e Noeme, por anos de apoio e confiança no meu sucesso.

E por fim, ao meu marido Arthur por ser o melhor companheiro me oferecendo suporte e motivação no decorrer desse processo. Obrigada pela paciência, compreensão e amor incondicional.

Resumo

O design de sistemas automatizados inteligentes é fortemente dependente da etapa inicial de análise de requisitos, que além de suprimir possíveis incongruências - próprias desta fase inicial - provê um modelo inicial e funcional do sistema a ser implementado, capaz de orientar a definição dos parâmetros *(design parameters)* e em seguida a própria implementação. Um grande esforço tem sido empregado na área de Inteligência Artificial para definir planejadores automáticos confiáveis que possam ser usados na solução de problemas reais, que geralmente possuem um número elevado de parâmetros. Isto leva a uma situação onde métodos formais, geralmente aplicados em abordagens independentes de domínio, precisam ser aliados a métodos mais pragmáticos para produzir bons resultados. Seguindo esse princípio, o presente trabalho propõe um método de projeto no qual o usuário podem fazer a aquisição de conhecimento, modelar o domínio (tanto o domínio de aplicação quanto o domínio do problema de planejamento), fazer uma análise dinâmica do modelo e eventualmente verificá-lo usando linguagens conhecidas como UML, as Redes de Petri, e HTN, mantendo as características hierárquicas do problema. Esta abordagem demanda novos planejadores automáticos que consideram a abstração do sistema hierárquico, que é derivado de um modelo hierárquico de requisitos e de uma análise unificada feita também em redes de Petri hierárquicas.

Palavras-chave: Modelos Hierárquicos. Análise de Requisitos. Planejamento Automático. Redes de Petri. UML. Processo de projeto.

Abstract

The design of intelligent systems is strongly dependent of the requirements analysis initial stage, wich can remove possible inconsistencies - own this early stage - providing a primary functional model of the system being implemented. Besides that, it can be able to guide the definition of the parameters (design parameters) and then the implementation itself. A lot of effort has been employed in Artificial Intelligence field to define reliable automated planning systems that can be used to solve real problems, which generally have a large number of parameters. This leads to a situation where formal methods, generally applied to solve domain-independent problems, need to be combined with more pragmatic methods to produce good results. Following this principle, this work proposes a design method in which the user can make the acquisition of knowledge, model the domain (both the work domain and the planning problem domain), make a dynamic analysis of the model and eventually simulate it using well-kown language as UML, Petri Nets, and HTN, maintaining the hierarchical characteristics of the problem. This approach demands new automated planners who consider the abstraction of hierarchical system, which is derived from a hierarchical model requirements and an unified analysis made in hierarchical Petri nets.

Keywords: Hierarchical models. Requirements analysis. Automatic planning. Petri nets. UML. Design process.

Lista de Figuras

2.1 Modelo conceitual de um Sistema de Planejamento Automático. Adaptado de [25] 12

2.2 Hierarquia dos diagramas UML na versão 2.4. Figura retirada de uml.diagramas.org. 28

2.3 Exemplo de Diagrama de Classe. Figura retirada de [Vaquero 2007]. . 31

2.4 Exemplo de rede de Petri. 34

2.5 Disparo de uma transição em redes de Petri. 34

2.6 Exemplo da hierarquia na rede GHENeSys. Figura retirada de [Miralles 2012]. 39

2.7 Diagrama de classes da rede GHENeSys.Figura retirada de [Miralles 2012]. 40

3.1 Esquema geral para aplicações de planejamento. 50

3.2 Processo de projeto clássico do itSIMPLE. 50

3.3 Etapas de projeto do método DynA. 52

3.4 Fluxo das etapas definidas no processo de projeto do modelo UML. . 57

3.5 Ciclo de Vida do Planejamento Automático. Figura adaptada de [Vaquero 2007]. 58

3.6 Tradução do Diagrama de Estados em Rede de Petri. Figura adaptada de [Vaquero 2007]. 59

3.7 Fluxo de atividades para o processo de tradução do modelo UML em redes de Petri 61

4.1 Diagrama de Classe para o desafio ROADEF 2005. 66

4.2 Diagrama de Estados Comportamental para o desafio ROADEF 2005. 67

4.3 Estado inicial representado no Diagrama de Objetos para o desafio ROADEF 2005. 67

4.4 Estado meta representado no Diagrama de Objetos para o desafio ROADEF 2005. 68

4.5 Rede de Petri gerada pelo itSIMPLE para o ROADEF 2005. 68

4.6 Rede de Petri, para o ROADEF 2005, gerada usando o algoritmo de tradução proposto no capítulo anterior. 69

4.7 Rede de Petri para o ROADEF 2005: versão itSIMPLE 4.0 versus versão itSIMPLE + Dyna. 70

4.8 Diagrama de Classe para o problema da Petrobrás. 73

4.9 Diagrama de Estados Comportamental para o problema da Petrobrás. 73

4.10 Diagrama de Estados Comportamental para operações realizadas pelos navios. 74

4.11 Diagrama de Objeto (estado inicial) para os navios. 74

4.12 Diagrama de Objeto (estado final) para os navios. 75

4.13 Pseudo código para o algoritmo de tradução de UML para rede de Petri. 76

4.14 Rede hierárquica para o problema da Petrobrás. 77

4.15 Nível mais alto de abstração da rede de Petri hierárquica. 77

4.16 Rede secundária para a operação dos navios no problema da Petrobrás. 78

4.17 Matriz de incidência. 78

4.18 Invariantes de lugar. 79

4.19 Invariantes de transição. 79

4.20 Diagrama de estado que representa o plano de ação para o navio 1. . 80

4.21 Diagrama de estado que representa o plano de ação para o navio 2. . 80

4.22 Diagrama de estado que representa o plano de ação para o navio 3. . 80

Sumário

1 Introdução **1**

1.1 Contextualização do Projeto 3

1.2 Motivação 6

1.3 Objetivos 8

1.4 Contribuição da tese 9

1.5 Organização do trabalho 9

2 Revisão da Literatura **11**

2.1 Processo de Projeto em Planejamento 11

2.2 Engenharia do Conhecimento em Planejamento 14

2.2.1 Critérios de Avaliação para o Desenvolvimento de Aplicações de Planejamento. 20

2.2.2 Planejamento Hierárquico 23

2.3 UML 26

2.3.1 Diagramas 28

2.3.2 Uso da UML em Planejamento Automático 29

2.3.3 UML na Modelagem de Problemas Estado-Transição 29

2.3.4 OCL - Object Constraint Language 31

2.4 Redes de Petri 32

2.4.1 Conceitos Básicos 32

2.4.2 Definição Formal 34

2.4.3 Principais Propriedades 35
2.4.4 Rede GHENeSys 37
2.4.5 Definição Original da Rede GHENeSys 38
2.4.6 Redes de Petri em Planejamento Automático 40
2.4.7 UML e Redes de Petri na Modelagem de Sistemas 42
2.5 Trabalhos Relacionados 44

3 O Processo de Projeto DynA (Dynamic Analyser) 47

3.1 Apresentação 47
3.2 O Método DynA 49
3.2.1 Detalhando o Método DynA 51
3.3 Modelagem do Problema Usando UML 54
3.4 Análise de Requisitos Usando Redes de Petri 57
3.4.1 Discussões 62

4 Estudos de Caso 64

4.1 ROADEF 2005 - Sequenciamento de Carros 64
4.1.1 Modelo UML 65
4.1.2 Transformando os Diagramas UML em Redes de Petri 66
4.2 Operações de Navios em Plataformas de Petróleo e Portos 70
4.2.1 Modelo UML 72
4.2.2 Transformando os Diagramas UML em Redes de Petri 72
4.3 Resultados e Discussões 81

5 Conclusões e Trabalhos Futuros 84

5.1 Contribuições 84
5.2 Trabalhos Futuros 87
5.2.1 Modelos Baseados em Tempo 87

5.2.2 Interpretação dos Planos Oferecidos pela Família SHOP2 . . . 88

5.2.3 Mapear Semelhança entre as Estruturas Topológicas 88

5.2.4 Implementação do Método Proposto 89

5.3 Considerações Finais 89

Referencias Bibliograficas **91**

A Apêndice A - Trecho de código da classe Java para o algoritmo de Tradução de UML para Redes de Petri. **101**

Capítulo 1

Introdução

Muitos pesquisadores têm empregado um grande esforço para modelar sistemas inteligentes de Planejamento Automático que sejam confiáveis e direcionados para aplicações do mundo real, como é o caso de [Vaquero 2012], [Vaquero 2009], [Reddy 2008], [31] e [Jain 2003]. De fato, estas aplicações de planejamento dependem diretamente do conhecimento do domínio e de sua representação, pois os algoritmos de planejamento automático raciocinam sobre uma base de conhecimento para gerar os planos de modo a atingir um determinado objetivo. Um fator agravante em relação ao conhecimento do domínio é que em problemas de planejamento existentes no mundo real, o conhecimento do domínio é incompleto e difícil de ser obtido[Vaquero 2006].

Devido à dependência do conhecimento do domínio, geralmente elicitado daqueles que o detém (por exemplo, usuários, especialistas, analistas, stakeholders, dentre outros) e consequentemente do seu processo de modelagem, o desenvolvimento de aplicações reais de planejamento não se limita apenas à construção de algoritmos de planejamento automático, mas exige um ciclo de vida completo e estruturado segundo às necessidades destas aplicações.

A comunidade de Planejamento Automático vem mostrando o interesse em aplicar as técnicas de projeto disponíveis em domínios reais, como por exemplo a

Engenharia de Requisitos e a Engenharia do Conhecimento, na utilização de técnicas de planejamento para resolver problemas reais [15], [Shah 2013], [91]. A representação de domínios reais requer o uso de estruturas e modelos que sejam capazes de expressar todos os níveis de abstração que o problema possui. Entretanto, os métodos clássicos de planejamento automático não têm oferecido resultados satisfatórios, especialmente por causa da limitação em tratar estes níveis de abstração que podem ser representados através de estruturas hierárquicas [Sohrabi e McIlraith 2008].

O Planejamento Automático desperta um grande interesse dos pesquisadores da Engenharia, especificamente na área da Engenharia do Conhecimento, que vem ganhando espaço dentro do campo de pesquisa de Planejamento Automático, pois a necessidade de captura e estruturação do conhecimento do domínio é cada vez mais evidente [Shah 2013]. Por isso, nos últimos anos, ferramentas de Engenharia do Conhecimento aplicadas ao Planejamento Automático têm sido desenvolvidas com o objetivo de auxiliar projetistas durante o ciclo de vida de projetos reais. Alguns destes processos são: levantamento e análise de requisitos, aquisição de conhecimento, especificação, modelagem do conhecimento, verificação de modelos e validação, dentre outros.

Ferramentas como o itSIMPLE (Integrated Tools Software Interface for Modeling PLanning Environments) [Vaquero 2012], propõem um ambiente integrado que auxilia na fase de modelagem (incluindo a fase de análise de requisitos). Tais ferramentas procuram tratar principalmente os processos que antecedem a utilização das técnicas de planejamento e a aplicação dos planos gerados, porém o conhecimento do domínio muitas vezes é incompleto mesmo após as fases de modelagem e muitos aspectos importantes do domínio são evidenciados e descobertos depois que estas técnicas foram utilizadas. Ou seja, ao avaliar uma solução fornecida por um algoritmo de planejamento sobre o conhecimento do domínio, é possível que novos aspectos se tornem aparentes. Diante disto, surge a necessidade de investir no desenvolvimento de métodos e processos de projeto, que antecipem tais aspectos garantindo a detecção de possíveis falhas no modelo.

Devido ao direcionamento para aplicações reais, a modelagem no ciclo de vida do projeto fica mais difícil, e os processos de validação, verificação e avaliação do modelo ficam mais complexos. Diante disso, torna-se necessário ter uma disciplina de projeto que auxilie principalmente na etapa de modelagem do problema. A literatura [Hoffmann 2003] [McCluskey 2003] já aponta para a necessidade de se ter um processo de projeto que funcione para sistemas reais, dado que o interesse da comunidade de Planejamento Automático tem se voltado recentemente para problemas do mundo real.

Com a alta complexidade dos modelos que representam problemas reais, as atividades de projeto não podem ser realizadas somente pelo especialista do domínio, ou exclusivamente por especialistas em planejamento [Perez 2006], mas sim por toda a equipe de projeto engajada com os diferentes pontos de vista (especialistas, usuários, analistas, stakeholders). Além de uma equipe estruturada, fica evidente a necessidade de ambientes ou ferramentas de apoio a modelagem e verificação, principalmente para o modelo desenhado e o domínio que este pretende representar.

O presente trabalho apresenta um processo de projeto desenvolvido para guiar a equipe de projeto na modelagem e verificação (especialmente) de problemas do mundo real em planejamento automático. Tal método utiliza modelos hierárquicos capazes de representar todos os níveis de abstração do modelo, seus requisitos e restrições de forma clara e concisa desde a etapa de modelagem até a geração dos planos.

1.1 Contextualização do Projeto

Dado o recente avanço das técnicas de Planejamento Automático, suas aplicações em problemas reais se tornaram um processo necessário. O grande foco da comunidade de pesquisa na área de planejamento em Inteligência Artificial ainda é, de um modo geral, a busca pela eficiência dos planejadores (no tempo que se gasta para produzir um plano aceitável). Os aspectos de análise vêm sendo negligenciados

e poucos trabalhos são encontrados nesta área [Zimmerman e Kambhampati 2003] [Upal 2005].

Para a realização do planejamento de uma atividade, é necessário determinar todas as características do sistema no qual ela está inserida. Vários fatores devem ser levados em consideração como, por exemplo, sub-sistemas envolvidos, variáveis internas, correlações entre esses sub-sistemas e o sistema, constantes e restrições. Tal determinação é chamada de modelagem do sistema, e dela depende o sucesso dos resultados obtidos no processo de planejamento. Nesse aspecto passam a ser importantes diversos pontos, como a complexidade do modelo proposto, as simplificações realizadas no processo de modelagem, e sua fidelidade ao sistema original.

Atualmente, para a realização de planejamento em sistemas não-triviais, ferramentas computacionais são utilizadas. A modelagem do sistema pode ser feita usando linguagens específicas como, a tradicional PDDL (Planning Domain Definition Language) [McDermott 2000], ou a UML (Unified Modeling Language) [OMG 2001], ou ainda a HTN (Hierarchical Task Network) apresentada em [25]. Para auxiliar no processo de modelagem e análise de requisitos existem sistemas como o itSIMPLE [Vaquero 2007], [Vaquero 2012], que possuem foco nas fases iniciais do processo de projeto como especificação de requisitos, e modelagem. Após modelados, os sistemas são processados por ferramentas denominadas planejadores, responsáveis pela iteração e resolução dos mesmos [McCluskey 2003].

Diante disso, dois aspectos são apresentados: (1) uma disciplina de projeto para a modelagem de sistemas reais, caracterizados por sua alta complexidade, o que traz como consequência a dificuldade de representação e, principalmente análise. Um fator a ser considerado é a diferença existente entre o desenvolvimento de um sistema puramente voltado à pesquisa (sistemas teóricos ou experimentais) e um sistema real como, por exemplo, um sistema de manufatura; (2) a análise dos modelos gerados na etapa anterior para obter informações que possam antecipar problemas no modelo, auxiliando assim, ainda na fase de design, a geração de planos de ação que sejam adequados ao problema que se está modelando.

Com a finalidade de implementar os aspectos proposto no parágrafo anterior, foi desenvolvido um processo de projeto usando modelos hierárquicos para sistemas de planejamento automático, que é composto de duas camadas: uma onde se aplicam métodos independentes do domínio e outra onde se usa o conhecimento específico para realizar análise de requisitos através de modelos orientados a objetos. Isto aumenta a qualidade das soluções de problemas de planejamento em Inteligência Artificial.

No primeiro caso identificou-se uma estrutura comum (composta por diagramas da UML) para os problemas, propondo a formalização desta estrutura usando rede de Petri Hierárquica. Deste modo, suas propriedades serviram para analisar semelhanças, ciclos repetitivos, invariantes e outras propriedades interessantes entre os modelos (além de servir de base para a reutilização no pós-design [Vaquero 2011], que está fora do escopo do presente trabalho).

No segundo nível o conhecimento específico pode ser incluído na forma de relações dinâmicas e propriedades das ações e restrições que serão inseridas em UML e transferidas para a rede de Petri (portanto tem um formato compatível com a fase anterior). Neste caso, a estrutura recebe o conhecimento dependente do domínio para que se possa aplicar as técnicas de análise dos métodos orientados a objetos baseados em hierarquia. Estas técnicas são particularmente sensíveis quando aplicadas a problemas reais necessitando de uma abordagem diferente das aplicações acadêmicas. Sistemas reais devem seguir um processo de projeto bastante disciplinado, fundamentado na Engenharia do Conhecimento, cuja etapa inicial é composta da elicitação e análise de requisitos.

O processo de projeto neste caso tem duas partes importantes: de um lado a fase de elicitação e documentação dos requisitos, tendo a UML 2.4 [OMG 2011] como base de representação, especialmente os diagramas de Pacotes, Classes, Estado Comportamental e Objetos; de outro lado a transferência e síntese destes diagramas em uma rede de Petri hierárquica (sistema GHENeSys - General Hierarchical Enhanced Net System). A segunda fase é a análise propriamente dita dos requisitos

depois que os diagramas foram sintetizados em uma rede hierárquica. A primeira etapa detecta basicamente contradições e conflitos entre os diversos requisitos ou os diferentes pontos de vista refletidos nos diagramas. A segunda etapa detecta inconsistências mais profundas, que se escondem na perspectiva dinâmica do plano que está sendo especificado, bem como informações adicionais ao modelo. Estas podem ser interpretadas pelos planejadores de modo a melhorar a performance do processo de planejamento automático. Entre estas informações adicionais inclui-se novas restrições, invariantes, estratégias parciais de solução, características e soluções específicas que podem contribuir para uma melhor resposta do planejador, seja no quesito tempo de processamento ou na qualidade dos planos gerados.

1.2 Motivação

A área de planejamento automático apresenta uma característica inerente ao seu desenvolvimento e evolução:

- Todo o estudo feito até hoje está historicamente ligado à tentativa de achar métodos para a solução de problemas de planejamento, de forma automática e independente de domínio objetivando a otimização dos planejadores. Assim, as soluções podem ser inseridas em mecanismos automatizados inteligentes, como robôs e outros sistemas autônomos.

- As técnicas formais desenvolvidas, independentemente de domínio, deram origem a técnicas que hoje são de vital importância em várias áreas como logística, sistemas de diagnóstico, navegação, robôs de uso espacial, posicionamento de satélite dentre outros. Esta demanda independe da adequação das técnicas formais a sistemas baseados em conhecimento (específico) para prover soluções reais e, ao mesmo tempo, potencializa a descoberta de novas soluções teóricas independentes de domínio.

Portanto, a pesquisa nessa área é de vital importância, dado que a solução prática para este dilema é a inserção de uma agenda de pesquisa que possa convergir de forma harmônica as duas tendências citadas, ao mesmo tempo que promove a aproximação das áreas de Inteligência Artificial (planning e scheduling em particular), a área de Engenharia do Conhecimento e da Engenharia de Controle e Automação.

A proposta de um processo de projeto para sistemas de planejamento automático é a principal motivação, dado que contrubui para uma área em processo de evolução e muito importante de Engenharia de Design, que é justamente a fase inicial, onde se faz o levantamento e a análise de requisitos (prometida mas efetivamente pouco usada, tanto na teoria como na prática).

A análise de requisitos de sistemas reais em planejamento, é um tema que tem sido discutido em importantes conferências e workshops da area de Planejamento Automático [McCluskey 2003], [Hoffmann 2003]. Entretanto, não existem ainda muitos trabalhos nesta linha de pesquisa, especialmente usando rede de Petri, na análise e validação dos modelos.

Trabalhos recentes evidenciam a importância e a necessidade da análise de requisitos de sistemas reais em aplicações de planejamento, onde problemas complexos são modelados usando UML [Simpson 2006], [77], [Vaquero 2012]. Entretanto, a literatura ainda é muito escassa quando o assunto é rede de Petri aplicada no processo de projeto, especialmente no que se refere a problemas reais de planejamento.

Além disso, o uso de Redes de Petri se mostrou bastante promissor no que se refere ao ciclo de melhoria dos planos, bem como no desempenho dos planejadores automáticos, uma vez que antecipa possíveis falhas no modelo a partir do qual o plano será derivado. Com o uso de Redes de Petri, é possível resolver fenômenos que ocorrem durante a geração do plano, tais como travamentos (deadlocks) e obtenção de uma ordenação parcial de metas muito adotada no planejamento hierárquico [25]. Estes problemas, até então só eram descobertos na geração dos planos. Com o processo de projeto proposto neste trabalho poderão ser antecipados na fase de

análise de requisitos através da análise via rede de Petri.

Outro fator importante a ser considerado é a complexidade dos problemas reais. Discussões já existem sobre a adequação dos métodos de modelagem usados em aplicações acadêmicas, a problemas reais [Hoffmann 2003], [McCluskey 2003]. Sabe-se que problemas reais de planejamento requerem um processo de projeto mais detalhado, abrangendo todo o ciclo de vida do problema e seus pontos de vista. Diante disto, surge a necessidade de definir uma disciplina de projeto para tais problemas.

Seguindo a tendência de se aplicar técnicas de planejamento em aplicações do mundo real, a análise de requisitos é um tópico de pesquisa bastante promissor, principalmente pelo fato deste processo ser um passo inevitável dentro do ciclo de vida de projeto de tais aplicações. O avanço da pesquisa nesta área traz contribuições para a crescente utilização e evolução das técnicas de Planejamento Automático em problemas reais.

1.3 Objetivos

Dito isto, vale delimitar os objetivso deste trabalho:

- Criar uma disciplina de projeto para auxiliar na etapa de modelagem do problema. Serão usados de métodos independentes do domínio para a modelagem do sistema. O conhecimento específico será inserido no modelo e análise através de estruturas hierárquicas, que também auxiliarão na análise de requisitos do mesmo.

- Modelar o domínio separadamente do problema. (Hipótese de independência domínio/problema).

- Desenvolver um método de modelagem e análise de requisitos, chamado de DynA (Dynamic Analyzer), composto por duas etapas: i) a primeira usa UML

2.4, e apresenta um conjunto mínimo de diagramas; ii) na segunda etapa será utilizada rede de Petri hierarquica para realizar a análise de requisitos, especialmente da dinâmica de ações.

1.4 Contribuição da tese

As principais contribuições científicas da tese são:

- Uma nova proposta de uma disciplina de projeto para a modelagem de problemas reais em planejamento automático, usando modelos hierárquicos em todo o processo de projeto. Tal disciplina é composta pelos dois níveis já mencionados: estruturação e modelagem do domínio e do problema de planejamento, e análise dinâmica deste modelo;

- Dentro deste método de projeto (especialmente no segundo nível), a definição de um conjunto mínimo de diagramas da UML que seja capaz de representar tais problemas de forma coerente e concisa;

- O desenvolvimento de um novo algoritmo que seja capaz de converter os requisitos modelados em UML, considerando todos os pontos de vista do problema representados nos diagramas, numa única rede de Petri hierárquica que reflita todos os níveis de abstração do problema, e a convergência dos pontos de vista. A partir desta rede aplicar as técnicas de análise próprias das redes de Petri, com o intuito de identificar e antecipar problemas no modelo, de modo a gerar planos de ações mais aderentes ao problema em questão.

- A definição de um processo de projeto, considerando os itens anteriores, que garanta a integridade de todos os níveis de abstração do problema, preservando a topologia hierárquica do modelo desde a etapa de modelagem até a geração dos planos de ação.

1.5 Organização do trabalho

O conteúdo do presente trabalho foi organizado da seguinte maneira:

- Capítulo 2: Neste capítulo apresenta-se a revisão da literatura relacionada ao presente trabalho, evidenciando necessidades e oportunidade de pesquisa. São apresentadas áreas como Planejamento Automático em Inteligência Artificial (com enfoque especial para o Planejamento Hierárquico), Engenharia do Conhecimento, UML, bem como redes de Petri no contexto do presente trabalho.

- Capítulo 3: Neste capítulo apresenta-se o projeto de tese. O capítulo descreve o método de projeto desenvolvido (DynA), detalhando cada ciclo de evolução do mesmo.

- Capítulo 4: Neste capítulo serão apresentados dois estudos de caso que têm o objetivo de ilustrar e avaliar empiricamente o método proposto no presente trabalho. Para tanto, foram escolhidos dois casos de uso para que se pudesse aplicar o método proposto.

- Capítulo 5: Neste capítulo conclui-se o presente trabalho reavaliando as principais contribuições e sugerindo algumas áreas onde cabem trabalhos futuros.

Capítulo 2

Revisão da Literatura

2.1 Processo de Projeto em Planejamento

Os maiores esforços das pesquisas realizadas na área de Planejamento Automático, estão no desenvolvimento de sistemas inteligentes que realizam a tarefa de planejamento. Tais sistemas são conhecidos como planejadores (planners). Os planejadores, para encontrar soluções (planos) - que são uma ordenação sobre um conjunto admissível de ações - aplicam mecanismos de busca sobre uma base de conhecimento.

Estes sistemas inteligentes recebem como entrada a descrição (representação) do conhecimento do domínio e um problema de planejamento a ser resolvido (composto de um estado inicial e os estados objetivos). Esta representação é fornecida através de uma linguagem de modelagem. Como resposta, o sistema fornece uma sequência de ações que modifica os estados do domínio de forma a atingir um determinado conjunto de objetivos [25], [LaValle 2006].

Em uma aplicação de planejamento automático, o planejador constitui geralmente a parte inteligente do sistema, e sua inteligência está diretamente ligada ao mecanismo de busca utilizado e ao modelo do conhecimento do domínio que este possui. A Figura 2.1 ilustra o modelo conceitual de uma aplicação de planejamento.

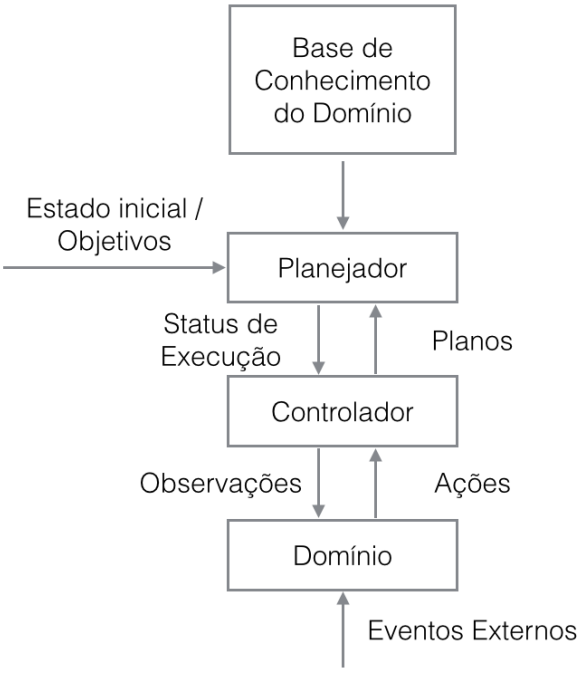

Figura 2.1: Modelo conceitual de um Sistema de Planejamento Automático. Adaptado de [25]

A Figura 2.1 mostra alguns elementos fundamentais do processo de planejamento. O Controlador representa o conjunto de sensores e atuadores que são responsáveis por observar o estado do domínio e também agir sobre o mesmo, seguindo as sequências de ações fornecidas pelo planejador. Outro elemento fundamental representado na figura é o domínio propriamente dito, onde o sistema inteligente está inserido. Na maioria dos problemas reais de planejamento, o domínio está sujeito a alterações provenientes tanto das ações executadas pelos agentes quanto dos eventos externos inerentes ao domínio. Este aspecto dinâmico do domínio aumenta ainda mais a complexidade dos problemas de planejamento.

O desenvolvimento de técnicas mais avançadas de planejamento automático tem se intensificado no decorrer na última década . Isto é motivado pelos diversos desafios tecnológicos encontrados em problemas reais existentes no mundo. Alguns trabalhos mostram aplicações de grande desafio para estes sistemas como, por exem-

plo, controle de espaçonaves e satélites [25] [Edelkamp e Jabbar 2006], manufatura avançada [Vaquero 2007] e logística de portos [Vaquero 2012].

A maioria dos trabalhos científicos relacionados à construção de planejadores se concentra na eficiência do processo de planejamento - ou seja, na rapidez com que o planejador fornece uma resposta - conforme ilustrado e discutido em [Perez e Carbonell 1994] [Zhao 2004].

Os planejadores são avaliados em geral pela sua eficiência no processo de planejamento. Porém, com as aplicações reais ficou evidente que não é possível considerar apenas a eficiência do processo de busca dos algoritmos, mas também a qualidade das respostas fornecidas, e como a aplicação foi representada [Ramchandani 1974].

Fica claro também que a confiabilidade destes sistemas não está estritamente ligada à eficiência e ao desempenho dos planejadores, mas ao modelo do conhecimento do domínio (refletindo os requisitos, parâmetros e métricas de qualidade) e, principalmente, ao ciclo de vida de projeto [Vaquero 2007].

Para que a aplicação seja modelada com sucesso, é necessário um ciclo de vida de projeto estruturado e adequado às necessidade de tal classe de aplicações [Vaquero 2007]. Infelizmente, o conceito de ciclo de vida ainda não recebe a devida atenção pela comunidade de planejamento.

Em qualquer processo de desenvolvimento de software e/ou sistema de informação, são necessárias diversas fases de projeto que são de extrema importância, principalmente para aplicações que envolvem sistemas inteligentes. Fases como, a elaboração dos processos de negócio, elicitação, análise de requisitos, especificação, modelagem, análise integrada, testes, implementação, validação, e verificação constituem, tópicos gerais de um ciclo de vida de projeto. Além das fases citadas, vale ressaltar que o projeto deve levar em consideração os diversos pontos de vista (dos usuários, desenvolvedores, analistas, especialistas, stakeholders, dentre outros) durante o processo de projeto (especialmente na fase inicial).

Ainda considerando o ciclo de vida do projeto, a elaboração da base de conhe-

cimento do domínio tem um papel fundamental nas fases iniciais de projeto. É fácil perceber que se a base de conhecimento do domínio (formada pelo conhecimento dos especialistas, desenvolvedores, usuários e stakeholders) é incorreta, a resposta fornecida por qualquer planejador também será incorreta [McCluskey 2002].

A área de pesquisa que trata dos aspectos referentes a elaboração do modelo do conhecimento é a Engenharia do Conhecimento. Esta área é responsável pela captura, construção e refinamento da base de conhecimento. Diversos conceitos provenientes desta área estão sendo gradativamente inseridos dentro da comunidade de planejamento, especialmente na modelagem de problemas reais, porém sua aplicação no contexto do Planejamento Automático é relativamente nova.

No cenário atual da área de planejamento, os conceitos de ciclo de vida de projeto ainda não estão estabelecidos, por isso ainda existem oportunidades a serem exploradas especificamente nos processos de projeto, verificação e validação do modelo. Baseado na carência de trabalhos nesta área e na demanda crescente por métodos de qualidade capazes de lidar com problemas reais, acredita-se que estudar os processos de análise e validação do modelo, pode trazer informações sobre o domínio que não foram identificadas facilmente, promovendo, consequentemente, uma melhoria na qualidade do modelo e principalmente dos planos fornecidos pelos planejadores.

2.2 Engenharia do Conhecimento em Planejamento

A Engenharia do Conhecimento estuda os princípios, métodos e ferramentas para o desenvolvimento e construção de bases de conhecimento e sistemas baseados em conhecimento [84]. Assim como na Engenharia de Software, a construção de uma base de conhecimento envolve processos como, levantamento de requisitos, especificação, modelagem, implementação e testes, para a definição da estrutura e arquitetura do modelo. Devido ao fato dos algoritmos de planejamento automático utilizarem a base de conhecimento (observadas suas especificidades) para gerar

soluções, pode-se considerá-los também como sistemas baseados em conhecimento.

A construção deste tipo de aplicação deve seguir um ciclo de vida que considera não só os princípios da Engenharia do Conhecimento, mas também da Engenharia de Requisitos, Engenharia de Software e Engenharia de Design. A maior parte dos trabalhos de Engenharia do Conhecimento encontrados na literatura dedicam-se às fases iniciais do projeto, principalmente na aquisição de conhecimento, especificação, modelagem e análise do modelo antes que este seja utilizado pelo planejador.

A elicitação do conhecimento, que é uma das primeiras etapas de projeto em planejamento automático, considera as possíveis ações e como os objetos são afetados por essas ações. O conhecimento deve ser modelado de forma adequada, para permitir tanto um raciocínio automatizado eficiente, quanto a construção de planos que sejam aplicáveis na solução do problema [McCluskey 2002].

Os processos de avaliação e análise do modelo se tornam fundamentais para a validação do modelo do domínio e do problema considerado [39]. Vaquero [Vaquero 2011], investiga a utilização de Engenharia do Conhecimento em aplicações de planejamento, tentando descobrir como esta técnica pode ser aplicada em problemas de planejamento. Dois pontos devem ser considerados: i) o algoritmo de planejamento, ou seja, o sistema de conhecimento usado como gerador de soluções; ii) a modelagem do domínio do ambiente, e do problema de planejamento. O interesse do presente trabalho está no segundo ponto, ou seja, no uso de técnicas de Engenharia do Conhecimento para desenvolver um processo de projeto para a modelagem de problemas reais em planejamento.

A engenharia do conhecimento usada em aplicações de planejamento automático ainda não atingiu a sua maturidade, portanto, existem vários pontos a serem investigados. Por causa disto, torna-se complicado definir um processo de projeto comum para planejamento automático. Entretanto, existem alguns trabalhos que apresentam uma discussão sobre as necessidades e particularidades do processo de projeto e ciclo de vida [McCluskey 2002] [Simpson 2006] [Vaquero 2007].

O processo de projeto é um fator importante para o sucesso do desenvolvimento e manutenção de aplicações reais em planejamento automático. Se este for bem estruturado aumenta a chance de se gerar um modelo apropriado para a aplicação de planejamento, além de reduzir a ocorrência de possíveis erros no futuro.

Vaquero [Vaquero 2011] apresenta 6 passos para serem seguidos no processo de projeto:

1 Especificação de Requisitos: elicitação, análise e validação dos requisitos. Neste passo pode-se usar uma abordagem semi-formal e análise de pontos de vista [Sommerville, I and Sawyer, P 2007].

2 Modelagem do Conhecimento: abstração, modelagem e reuso da definição do domínio, e o relacionamento básico com o problema de planejamento.

3 Análise do Modelo: verificação e validação do domínio do modelo e do problema de planejamento, bem como o aperfeiçoamento do modelo.

4 Implantação do Modelo para o Planejador: tradução da especificação do problema numa linguagem entendida pelos planejadores automáticos.

5 Síntese do Plano: interação com um ou mais planejadores automáticos, para criar possíveis soluções para o problema de planejamento.

6 Análise do Plano e Pós-Design: análise dos planos gerados de acordo com algumas métricas.

Dos 6 passos propostos por Vaquero [Vaquero 2011], especialmente os 1, 2 e 3 interessam no presente trabalho. Em problemas do mundo real a identificação prévia e a análise dos requisitos são passos chave para o sucesso do projeto. Estes projetos têm classes distintas de usuários, stakeholders dentre outros, os quais possuem pontos de vista diferentes [Sommerville, I and Sawyer, P 2007], e devem ser combinados de forma consistente foando nos objetivos do projetista e dos usuários.

A fase de especificação de requisitos, geralmente é dividida em duas etapas:

- aquisição de conhecimento.
- análise de requisitos para identificar possíveis conflitos no modelo, omissões ou interpretações equivocadas sobre a interação entre o sistema de planejamento e o domínio do ambiente.

Nenhum problema real é totalmente isolado do mundo e qualquer sistema de planejamento está inserido num domínio real com o qual deve interagir. Portanto, é necessário ter um modelo que represente o ambiente, e que este modelo seja independente do sistema de planejamento [McDermott 2000]. Diante disso, fica claro que dois modelos devem ser construídos: um modelo que represente o domínio do ambiente; e outro modelo para representar o problema de planejamento, e neste caso poderão existir vários modelos que representem problemas diferentes para um mesmo domínio. Tais representações devem prover semântica, o que implica que os elementos mencionados correspondem diretamente aos elementos nomeados no domínio real [McCluskey 2002].

Para a modelagem de um domínio de planejamento adequado deve-se analisar o modelo durante as fases de concepção. A análise do modelo abrange verificação, validação e aperfeiçoamento do conhecimento e refinamento de todo o modelo [McCluskey 2002]. Geralmente, a análise feita manualmente, automaticamente ou assistida por algum sistema, foca em dois aspectos principais, que são: as propriedades estáticas e as propriedades dinâmicas[Vaquero 2011]. O objetivo é encontrar erros de modelagem, inconsistências e incoerências nas propriedades. Uma vez que tais erros foram encontrados e corrigidos, pode-se poupar tempo e recurso nas fases anteriores.

Existem dois tipos de análise que podem auxiliar na validação do modelo: i) a análise estática investiga essencialmente se o modelo é consistente. Tal análise pode ser feita tanto por verificadores de sintaxe, quanto por depuradores, verificando partes do modelo, especialmente aquelas que contém um conjunto de diagramas ou qualquer outro tipo de representação. A análise estática pode ser aplicada para

verificar a definição de tipos de objetos, restrições, definições de estado, e etc. ii) a análise dinâmica verifica se o comportamento das ações modeladas é consistente com os requisitos e com o que é esperado pelos usuários. Isto envolve investigar como as ações interagem entre si e como elas são executadas. Tanto a análise estática, quanto a análise dinâmica podem ser realizadas independente do planejador automático.

Existem algumas ferramentas de Engenharia do Conhecimento que podem ser usadas para produzir modelos de domínios de planejamento. A seguir será apresentado um resumo dessas ferramentas [Shah 2013].

O itSIMPLE [Vaquero 2012] apresenta um processo de projeto disciplinado para a utilização de planejamento em problemas reais, integrando um conjunto de linguagens e ferramentas para apoiar um processo de projeto cíclico, partindo de uma representação informal para uma representação formal. No itSIMPLE, o modelo do domínio de planejamento é desenhado em UML, que é uma linguagem gráfica de fácil visualização e entendimento. O modelo, que inicialmente é desenhado em UML, também pode ser gerado em PDDL diretamente. A análise e validação atualmente são feitas usando "storyboards", que mostram como o plano evolui, para isso são usados diagramas de objetos. O itSIMPLE é desenvolvido pelo grupo de pesquisa do Design-Lab (Escola Politécnica da Universidade de São Paulo) e será utilizado ao longo da tese.

A ferramenta EUROPA (Extensible Universal Remote Operations Planning Architecture) [Barreiro 2012], é uma plataforma integrada para aplicações de planejamento e escalonamento, programação de restrições e otimização em Inteligência Artificial. O objetivo principal dessa plataforma é lidar com problemas complexos do mundo real. Permite que o usuário possa escolher entre duas linguagens diferentes de representação, NDDL e ANML [80]. Esta última tem sido usada em várias missões da NASA. EUROPA oferece suporte a modelagem, visualização de resultados e um processo de planejamento interativo.

GIPO (Graphical Interface for Planning with Objects) [Simpson 2006] [77] é

baseado em sua própria linguagem centrada em objetos, que são OCL e OCLh. Essas linguagens formais exploram a idéia de que o conjunto dos potenciais estados dos objetos é definido primeiro, antes da definição das operações [Mccluskey e Kitchin 1998]. GIPO é centrada na definição precisa do planejamento de estados como um amalgama de objetos individuais dos estados. Isto dá o conceito de "estados do mundo"como sendo feito de um conjunto de objetos de estado, satisfazendo certos tipos de restrições. Esquemas de operação são restringidos para serem consistentes com os estados, dando a oportunidade de se usar ferramentas que verificam a consistência do modelo. GIPO verifica uma lista de atributos como, a hierarquia da classe de objetos, se a descrição dos objetos de estado satisfaz os invariantes, a estrutura dos predicados e o esquema dos operadores. Tal verificação de consistência garante a prevenção de erros nas classes, em contraste com métodos ad hoc.

JABBAH [Gonzalez Ferrer 2009] é uma ferramenta integrada independente do domínio, que tem o objetivo de desenvolver o processo de transformação para ser representado num modelo HTN correspondente. O sistema lida principalmente com processos de negócio e fluxos de trabalho. Os processos são representados como gráficos de Gantt ou através de uma máquina de código aberto para a modelagem de fluxos de trabalho. A ferramenta oferece apoio a transformação de BPMN (Business Process Management Notation) para HTN-PDDL. O domínio do modelo é usado em planejadores HTN para obter uma rede de soluções de tarefas.

MARIO (Mashup Automation with Runtime Invocation and Orchestration) [Bouillet 2009] é uma ferramenta integrada para a composição de fluxo de trabalho em diferentes plataformas, tais como Web Services e Enterprise Service Bus. Esta ferramenta provê uma linguagem de representação baseada em conhecimento para a composição de problemas de planejamento de metas. Também oferece uma GUI baseada na web para sistemas de planejamento em IA. Assim o usuário pode oferecer uma composição de metas via software, visualizar e gerar fluxos com parâmetros para implantar em outras plataformas.

PDDL Studio [Plch 2012] é um editor de PDDL recente que permite aos usuá-

rios escrever e editar o domínio e o problema de planejamento. O principal objetivo e oferecer aos engenheiros do conhecimento a funcionalidade de editar e inspecionar o código PDDL, independente de como esse código foi criado. A ferramenta apoia o usuário a identificar erros de sintaxe. PDDL Studio não requer que o usuário desenhe qualquer tipo de diagrama.

VIZ [Vodrazka e Chrpa 2010], é uma ferramenta de engenharia de conhecimento inspirada no GIPO e no itSIMPLE. Ela compartilha algumas características desses dois sistemas com a adição de uma GUI simples e amigável, que permite ao usuário sem muita experiência desenvolver modelos em PDDL. Esta ferramenta utiliza um processo intuitivo de modelagem, que usa diagramas transparentes para produzir os modelos em PDDL.

KEWI [24] [23], oferece um modelo conceitual usado para representar conhecimento declarativo e procedural. O modelo consiste em três níveis: uma ontologia, um modelo básico de ações e métodos mais complexos. É este modelo conceitual estruturado que facilita a engenharia do conhecimento. O objetivo é avaliar a utilização de um modelo central de conhecimento numa gama de funções relacionadas com planejamento automático, onde as partes do modelo são automaticamente montadas, usando por exemplo a PDDL para uso operacional.

GIPO, itSIMPLE e KEWI objetivam um processo de projeto baseado em modelos, nos quais os objetos são agrupados em classes e subclasses. Apesar de usarem linguagens de representação diferentes, a filosofia de projeto é bastante parecida.

2.2.1 Critérios de Avaliação para o Desenvolvimento de Aplicações de Planejamento.

Além desse resumo, Shah et. al. [Shah 2013] apresentam alguns critérios de avaliação aplicados às principais abordagens para o desenvolvimento de aplicações de planejamento, que serão apresentados na tabela 2.1.

Tabela 2.1: Critérios de Avaliação para o Desenvolvimento de Aplicações de Planejamento.

Critérios de Avaliação	
Critério	*Características a serem verificadas*
Operacionalidade	Quão eficiente são os modelos produzidos? O método é capaz de melhorar o desempenho dos planejadores para os modelos e problemas gerados?
Colaboração	O método/ferramenta incentiva o trabalho em equipe? O método é adequado para ser utilizado em equipe ou foca na utilização de um único usuário?
Manutenção	É fácil rever o modelo e fazer alterações? Existe algum tipo de documentação que é gerada automaticamente? A ferramenta induz o usuário a produzir documentação?
Experiência	O método/ferramenta é indicado para usuários inexperientes? O usuário precisa ter um bom conhecimento de PDDL? A ferramenta é capaz de apoiar o usuário e "esconder" detalhes de baixo nível no desenvolvimento do modelo?
Depuração	O método/ferramenta apoia a depuração do modelo? O tempo de depuração pode ser reduzido pela ferramenta? Existe algum mecanismo para garantir a qualidade geral do modelo?
Apoio aos usuários	Existem manuais disponíveis para o uso do método/ferramenta? É fácil receber suporte dos desenvolvedores? Existe alguma comunidade ativa usando a ferramenta?

Utilizando esses 7 critérios de avaliação Shah et. al. [Shah 2013] analisam alguns métodos, incluindo a ferramenta itSIMPLE. Esta avaliação é de extrema importância e nos guiará no desenvolvimento deste projeto. Sobre a versão atual do itSIMPLE [Shah 2013] apontam:

Operacionalidade: A partir da experiência deles usando o itSIMPLE, eles afirmam que o modelos gerados podem sim aumentar o desempenho dos planejadores. Isto provavelmente está relacionado a uma descrição de domínio menos restritiva do que a oferecida por editores PDDL. A qualidade dos modelos gerados não depende da experiência, o itSIMPLE guia os usuários no processo de projeto.

Colaboração: O itSIMPLE não foi desenvolvido para o trabalho em equipe. Isto significa que, o modelo é desenvolvido por um único usuário. Entretanto, é possível importar modelos (projetos) desenvolvidos por usuários diferentes. Isto ajuda a troca de idéias e comentários entre os usuários. Além disso, os diagramas gerados em UML são úteis no compartilhamento e discussão de questões sobre o modelo entre os especialistas.

Manutenção: A ferramenta itSIMPLE foi desenvolvida para apoiar um ciclo de projeto disciplinado. Os diagramas UML também podem ser usados como documentação. A partir deste ponto de vista fica fácil manter o modelo gerado mesmo por pessoas que não foram envolvidas no processo de projeto. Entretanto, se um modelo gerado no itSIMPLE é modificado usando uma ferramenta diferente (ou mesmo o editor de texto), não é possível importar o modelo de volta.

Experiência: O usuário típico do itSIMPLE não precisa ser um expert em PDDL. Entretanto, deve ter alguma experiência básica em engenharia de software, especialmente em UML.

Eficiência: A maior parte do tempo é gasta na modelagem das classes de objetos e definindo as interações entre estes nos diagramas UML. Após isso, é necessário pouco tempo para depuração. Esse método é mais lento do que o uso direto da PDDL, na geração do modelo inicial.

Depuração: Mesmo que o itSIMPLE forneça uma análise dinâmica, através das redes de Petri criadas a partir do modelo UML, a depuração inicia através de testes dinâmicos feitos ao executar os planejadores com alguns problemas-modelo. Embora os diagramas UML ajudem no desenvolvimento e manutenção, alguns erros são provenientes da falta de um mecanismo de planejamento para resolver um determinado problema que, na maioria dos casos, alerta o usuário sobre a presença de erros no modelo.

Apoio aos usuários: itSIMPLE oferece uma documentação completa que inclui uma descrição da ferramenta, um tutorial e uma seção de FAQ online. É fácil encontrar informações e soluções para a maioria dos problemas mais comuns.

A partir dessa avaliação, e com as propostas já definidas para o presente trabalho, foram definidas as alterações mais urgentes e relevantes tanto no processo de projeto quando no método usado no itSIMPLE. Essas mudanças serão discutidas posteriormente não seção que apresenta a proposta de projeto.

2.2.2 Planejamento Hierárquico

O planejamento hierárquico, também conhecido como Rede Hierárquica de Tarefas (*Hierarchical Task Network* - HTN) [25], provê uma forma particular de especificação de regras de controle de busca, utilizando a natureza hierárquica inerente a muitos problemas do mundo real. A principal diferença com o planejamento clássico está relacionada com o tipo de meta a ser atingida: neste caso, o objetivo é realizar um conjunto de tarefas através de sua decomposição em sub-tarefas, recursivamente, até que seja possível realizá-las diretamente através de ações primitivas. Cada decomposição possível representa uma ramificação na árvore de busca do problema. O projetista do domínio deve especificar diferentes métodos de decomposição para cada tarefa composta, junto com suas pré-condições para que sejam utilizadas durante o processamento. Esses conjuntos de tarefas são representados através de redes de tarefas, representando restrições sobre sua ordem de execução, que pode

ser tanto uma ordem total quanto uma ordem parcial. Esta última permite maior expressividade, mas, em contrapartida, aumenta a complexidade do problema de planejamento.

Assim como no planejamento clássico, cada estado do mundo é representado por um conjunto de átomos e cada ação corresponde a uma transição de estados determinística. A entrada para o sistema de planejamento inclui um conjunto de operadores similares aos do planejamento clássico e também um conjunto de métodos, cada um deles com uma especificação de como decompor alguma tarefa em um conjunto de sub-tarefas (tarefas menores) [25]. O planejador HTN irá aplicar um operador somente se for orientado, por um método, a fazê-lo. Portanto, cada método é parte da especificação de uma função de ramificação.

Também conhecido como planejamento prático, o planejamento hierárquico é a técnica mais utilizada para solucionar problemas reais, nas mais diversas áreas do conhecimento, como otimização de navegação de robôs [6], geração automatizada de cursos baseada em tarefas pedagógicas [Ullrich 2005], dentre outras.

O poder de expressão do planejamento hierárquico é maior que o do planejamento clássico, permitindo que todo problema clássico seja representado por um hierárquico, onde o contrário não é verdadeiro [35]. A função do planejador hierárquico é encontrar métodos para cada tarefa, decompondo-as de modo que as pré-condições sejam respeitadas. Isso reduz drasticamente o espaço de busca, por utilizar sequências de ações como escolhas de possíveis componentes do plano, ao invés de escolher ações individualmente. Basicamente permite maior controle do espaço de busca [25].

O planejamento hierárquico tem sido largamente utilizado em aplicações do mundo real e isto deve-se ao fato de que os métodos HTN oferecem um meio conveniente de representar soluções de problemas que correspondem ao modo com que os especialistas de domínio humano pensam ao planejar [25].

Existem diferentes formalismos para o planejamento hierárquico [Yang 1990]

[35], pois as contribuições partiram de mais de um planejador, dentre eles pode-se citar: *Nets Of Action Hierarchies* (NOAH) [Sacerdoti 1975], utilizando a linguagem *Semantics Of User Programs* (SOUP), com busca no espaço de planos e e geração de planos em ordem parcial; NONLIN [Tate 1977] com vínculos causais; SIPE-2 [Wilkins 1990], permite ordens arbitrárias de ações; O-Plan [Tate e Currie 1991], que oferece maior modularidade e flexibilidade; Universal Method Composition Planner (UMCP) [34], com um sólido formalismo.

Existe também uma abordagem híbrida [33]. Além da decomposição hierárquica, outras abordagens diferentes de controle incluem poda do espaço de busca utilizando lógica temporal, onde pode-se citar: TLPlan [Bacchus e Kabanza 2000], (*Linear Temporal Logic*) (LTL) [Pnueli 1977], TALplanner [Doherty e Kvarnstrom 1999] e (*Temporal Action Logics*) (TAL) [Doherty 1998].

Planejadores Hierárquicos - Família SHOP

A família *Simple Hierarchical Ordered Planner* (SHOP) [Nau 1999], [Nau 2005] de planejadores permite planejamento hierárquico utilizando HTN, em ordem total (SHOP) e em ordem parcial (SHOP2) [Nau 2003]. Dentre suas propriedades interessantes está o fato de que a busca é feita progressivamente, de modo que o planejador conheça sempre o estado atual. Isto oferece maior poder de expressão, tais como: axiomas, manipulação simbólica e numérica e a execução de procedimentos externos durante o planejamento.

JSHOP2 [Okhtay e Nau 2004] é uma implementação em Java do planejador SHOP2 (sendo este último escrito em Common Lisp). Adicionalmente, o JSHOP2 possui uma fase de compilação, que, a partir da descrição de um domínio, gera código Java de um planejador dependente do domínio. Este planejador gerado é utilizado para resolver problemas específicos deste domínio e se aproveita desta técnica na geração do código para torná-lo mais eficiente quanto às suas estruturas internas de representação. Resultados experimentais sugerem que o ganho obtido pela técnica

de compilação otimiza o tempo de resposta do planejador [Okhtay e Nau 2004].

Os planejadores SHOP têm sido utilizados em aplicações diversas [Nau 2005], como planejamento de evacuação, combate a incêndio em florestas, integração de sistemas de software, controle de veículos autônomos e seleção de materiais para manufatura, além de diversar aplicações para web semântica [Wu 2003], [Dan Wu Evren Sirin e Parsia. [Sirin 2004], [1].

Vantagens e Desvantagens

Comparados aos planejadores clássicos, a principal vantagem da abordagem HTN é a sua sofisticada representação de conhecimento e raciocínio sob ações compostas [Georgievski e Aiello 2015] [Georgievski e Aiello 2014]. Eles podem representar e solucionar uma variedade de problemas de planejamento; com um bom conjunto de métodos de decomposição, eles podem solucionar problemas de planejamento clássico com maior eficiência. A principal desvantagem de planejadores HTN é que os projetistas necessitam especificar um conjunto de métodos. O planejamento clássico necessita apenas do conjunto de operadores de planejamento.

Comparando HTNs com regras de controle [Bacchus e Kabanza 2000], é difícil afirmar que tipo de técnica de controle é mais efetiva. Planejadores hierárquicos têm sido mais utilizados em aplicações do mundo real, mas isto deve-se parcialmente ao fato de serem mais antigos. HTNs oferecem ao planejador informações sobre quais opções considerar, e regras de controle sobre quais opções não considerar. Os dois tipos de informação são úteis em diferentes situações e combiná-las pode ser vantajoso [83].

2.3 UML

Para representar um problema de planejamento é necessário usar uma linguagem que faça essa tarefa de forma que, os planejadores automáticos sejam capazes

de interpretar o problema e oferecer um conjunto de ações executáveis.

A maioria dos planejadores automáticos usam a linguagem PDDL como entrada para os problemas de planejamento. Entretanto, a linguagem PDDL não é de fácil entendimento. Para compreendê-la é necessário um conhecimento prévio de linguagens declarativas. Esta característica dificulta o uso da PDDL. Pensando nisso Vaquero [Vaquero 2007] desenvolveu a ferramenta, itSIMPLE, que permite aos usuários definir os problemas de planejamento usando diagramas UML (Linguagem de Modelagem Unificada). Uma vez que o problema está modelado em UML, este é traduzido pela ferramenta para a linguagem PDDL e em seguida fornecido a uma série de planejadores automáticos.

Em planejamento, a UML pode ser utilizada para modelar os problemas de forma mais simples, por ser uma ferramenta de modelagem visual. Analisando a literatura percebe-se que o uso da UML está cada vez mais difundido, não só para a modelagem de software, mas para a modelagem de processos e de sistemas reais [8].

Além do fato de ser uma linguagem visual, o que facilita a modelagem dos sistemas, o uso da UML permite agregar ao projeto atributos de modelagem e análise que certamente auxiliarão na construção de um modelo mais adequado à realidade. Um exemplo desses atributos é a linguagem OCL, que será explicada na próxima seção.

A UML é uma linguagem sólida para modelar sistemas, e tem sido usada com sucesso em diferentes projetos, vários deles ligados ao desenvolvimento de software, mesmo que sua utilidade não esteja restrita a este tipo de sistemas. A UML oferece um amplo conjunto de notações e diagramas que permitem representar os diferentes aspectos de um sistema em desenvolvimento, mas carece da possibilidade de simulação e verificação. Isto é devido a sua natureza semi-formal.

A UML não é uma metodologia de desenvolvimento, ou seja, não oferece um método para projetar sistemas, porém serve de ajuda para modelar, visualizar e documentar sistemas. Junto com uma notação gráfica, a UML também especifica

significados, isto é, semântica. Atualmente, é aceita como uma linguagem padrão para modelagem de projeto de software, e apresenta diversos diagramas focando em aspectos distintos do software mas, ao mesmo tempo, provê descrições redundantes dos mesmos aspectos do sistema [8].

Sendo assim são necessárias técnicas de verificação e validação para assegurar a consistência das descrições. Porém, verificação e validação de sistemas complexos desenvolvidos de acordo com UML não é um processo simples, devido à complexidade dos sistemas além de que existem vários diagramas em UML que podem ser usados para modelar o comportamento e a estrutura do sistema.

2.3.1 Diagramas

A especificação UML 2.4.1 define dois grupos de diagramas: diagramas estruturais e diagramas comportamentais. Os diagramas estruturais mostram a estrutura estática do sistema e suas partes em níveis de abstrações diferentes e como eles se relacionam. Os elementos num diagrama estrutural representam os conceitos significativos de um sistema, e podem incluir abstrações do mundo real e conceitos de implementação. Os diagramas comportamentais mostram o comportamento dinâmico do sistema, que pode ser descrito como uma série de alterações no sistema durante um intervalo de tempo [8] [Guedes 2009]. Na figura 2.2 pode-se observar a nova hierarquia dos diagramas UML desde a versão 2.4.

2.3.2 Uso da UML em Planejamento Automático

Em planejamento automático a UML pode ser usada para representar relacionamentos entre os planejadores, o domínio e o problema de planejamento. Em planejamento, existem duas classes diferente de requisitos: a) requisitos do domínio e requisitos do problema. Por causa disto, a fase de projeto pode ser separada em dois estágios: 1) modelagem do domínio, onde todos as características essenciais do domínio são consideradas e as otimizações são deixadas de lado; 2) inclusão dos aspectos

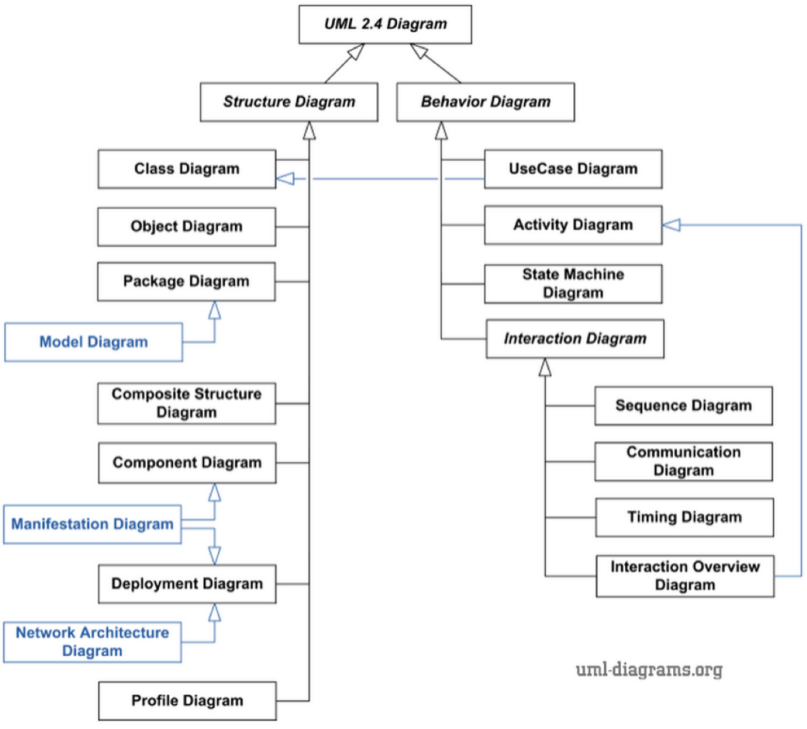

Figura 2.2: Hierarquia dos diagramas UML na versão 2.4. Figura retirada de uml.diagramas.org.

de otimização que estão relacionados ao problema. Essa é uma das abordagens proposta pelo laboratório de pesquisa ao qual essa tese faz parte (Design-Lab/USP), uma abordagem um pouco diferente será apresentada neste projeto [Vaquero 2007].

Além do fato de ser uma linguagem visual, o que facilita a modelagem dos sistemas, o uso da UML permite agregar ao projeto atributos de modelagem e análise que certamente auxiliam na construção de um modelo mais adequado à realidade. Um exemplo desses atributos é a linguagem OCL.

Um diagrama UML - Diagrama de Classes ou de Estados, por exemplo - não possui o formalismo necessário para prover todos os aspectos relevantes do modelo, ou eventualmente de uma especificação. Existe a necessidade de expressar restrições adicionais sobre os objetos no modelo. Tais restrições são geralmente descritas em linguagem natural, e a prática mostra que o uso de linguagem natural sempre

causa ambiguidade. Com o objetivo de evitar tais ambiguidades foi desenvolvida a linguagem formal OCL - Object Constraint Language - para especificar restrições [OMG 2003].

2.3.3 UML na Modelagem de Problemas Estado-Transição

Antes de listar os diagramas UML usados na modelagem de problemas estado-transição, vale apresentar uma definição breve do que compreende tais problemas. Segundo [Lam e Shankar 1988], um sistema estado-transição é especificado como um conjunto finito de variáveis de estado $v = (v1, v2, ...)$, um conjunto finito de eventos, $e1, e2, ...$ e uma condição inicial do sistema. Para cada variável de estado, existe um domínio específico de possíveis valores. O estado do sistema é representado por um conjunto de valores que podem ser assumidos pelas variáveis de estado. O espaço de estados S de um sistema de transição é o produto cartesiano dos estados das variáveis de domínio. Um evento pode ocorrer somente quando os valores em v satisfazem as condições rotulados nos eventos. A ocorrência de um evento atualiza os valores das variáveis de estado em uma ação atômica.

Dito isto, foram selecionados os diagramas UML que melhor representam esta classe de problemas. Na classe dos diagramas estruturais, interessam para o presente trabalho o diagrama de Pacotes e o diagrama de Classe. O diagrama de Pacotes mostra os pacotes e seus relacionamentos. O diagrama de Classe é uma estrutura estática que descreve o sistema no nível de seus classificadores (classes, interfaces, e etc.). Este diagrama é capaz de representar alguns dos classificadores do sistema, subsistemas ou componentes, os diferentes relacionamentos entre estes, seus atributos, operações e restrições (em OCL) [Guedes 2009].

Na classe dos diagramas estruturais, interessa para este projeto apenas o diagrama de Estados e suas variações, mas o diagrama de Atividades também foi pesquisado com o intuito de entender qual a maior diferença entre este e o diagrama de Estados.

O diagrama de Estados é usado para modelar o comportamento discreto por transições de estados finitos. Este também pode ser usado para representar protocolos de uso de apenas uma parte do sistema. Estes dois tipos de diagramas de Estado são chamados de diagrama de Estados Comportamental e diagrama de Estados de Protocolo [OMG 2011] [Guedes 2009].

O diagrama de Atividades representa sequências e condições para coordenar comportamentos de baixo nível, em vez de classificadores que possuem esses comportamentos. Estes são comumente chamados de fluxo de controle e fluxo de objetos [OMG 2011].

2.3.4 OCL - Object Constraint Language

A OCL é uma linguagem de especificação de restrições que favorece a modelagem de negócios e sistemas, e é muito similar a lógica de primeira ordem. As restrições são geralmente expressões booleanas, que podem ser utilizadas com conectores lógicos, que devem ser verdadeiros durante todo o ciclo de vida do objeto no sistema. As expressões de restrições podem complementar a descrição das características e dos comportamentos dos objetos no sistema. Além disso, é possível representar expressões sobre classes, associações, estados, ações, diagramas, dentre outros.

A OCL pode ser utilizada para diferentes propósitos como, por exemplo:

- especificar invariantes sobre as classes e os elementos do modelo;
- especificar pré e pós-condições das ações/operadores realizadas pelos objetos;
- descrever restrições de estados do domínio;
- ser utilizada como linguagem de navegação (acesso) entre classes;
- especificar restrições sobre a ocorrência de operadores.

A figura 2.3 fornece um exemplo de Diagrama de Classe, no qual algumas expressões serão apresentadas com o intuito de exemplificar a utilização da OCL em um Diagrama de Classes.

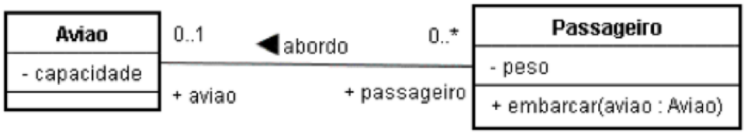

Figura 2.3: Exemplo de Diagrama de Classe. Figura retirada de [Vaquero 2007].

Além das restrições introduzidas pela multiplicidade entre as classes Avião e Passageiro é possível adicionar a seguinte restrição utilizando a OCL: em todo e qualquer estado do domínio a capacidade de peso do avião deve ser sempre maior ou igual a somatória dos pesos dos passageiros que estão à bordo. Esta invariante do domínio pode ser declarada da seguinte forma em OCL [Vaquero 2007]:

context Aviao inv: self.capacidade >= self.passageiro.peso -> sum()

Qualquer que seja o avião a somatória dos pesos de todos os passageiros que estão a bordo deve ser menor ou igual a capacidade do avião. Observação: a palavra reservada self representa um elemento da classe Avião.

Utilizando o mesmo exemplo da Figura 2.4 é possível declarar pré e pós-condições da ação embarcar. Uma pré-condição seria afirmar que para o passageiro embarcar no avião seria preciso verificar se seu peso não excede a capacidade de carga do avião. Uma pós-condição para esta ação, caso a pré-condição seja estabelecida, seria afirmar que o passageiro estará à bordo do avião. Estas condições podem ser declaradas em OCL da seguinte maneira [Vaquero 2007]:

context Passageiro : : embarcar (aviao : Aviao) pre: aviao.capacidade >= aviao.passageiro.peso -> sum() + self.peso post: self.abordo = aviao

A OCL mostra-se muito útil na fase de modelagem na área de Planejamento Automático já que esta é uma linguagem formal de especificação de restrições. Res-

trições como invariantes são muito úteis durante o processo de planejamento pois evita que os planejadores realizem buscas exaustivas em caminhos que não chegarão a soluções plausíveis. As restrições podem, de fato, enriquecer o modelo do domínio fazendo com que as técnicas de planejamento possam usufruir desta informação.

2.4 Redes de Petri

2.4.1 Conceitos Básicos

O conceito de Redes de Petri (RdP) foi introduzido por Carl Adam Petri em sua tese de doutorado (1962). A tese apresentava as RdPs como um formalismo matemático para descrever relações entre condições e eventos, no estudo de protocolos de comunicação entre componentes assíncronos em computadores. Embora ocorresse uma ampla divulgação acadêmica deste conceito ao longo de três décadas, o potencial só foi reconhecido na metade da década de oitenta. Nesta época, a teoria foi usada para implementações práticas nas áreas de informática e manutenção devido à disponibilidade de recursos mais poderosos de hardware e software.

Um sistema modelado em rede de Petri é um formalismo matemático que possui uma representação tanto algébrica quanto gráfica oferecendo uma abordagem uniforme para modelagem, análise e projeto de sistemas a eventos discretos [Murata 1989]. A RdP é, de fato, uma representação fundamental e extremamente efetiva para a modelagem de sistemas. A sua base teórica permite o desenvolvimento de poderosas ferramentas de análise e síntese de estratégias de controle.

Em geral, utiliza-se Redes de Petri para modelagem conceitual da estrutura dinâmica do sistema. Sua aplicação tem se estendido a uma grande variedade de sistemas como, por exemplo, sistemas de comunicação, sistemas de software, simulação, e processos seqüenciais em sistemas flexíveis de manufatura [Vaquero 2011].

De um modo geral, as Redes de Petri são grafos compostos de: lugares (P) que graficamente são representados por círculos; transições (T) que graficamente

são representadas por barras; arcos orientados que conectam lugares a transições e transições a lugares (os arcos nunca conectam lugares a lugares ou transições a transições) e estes podem receber pesos (1,2,3,....); e marcas que são utilizadas para definir o estado da rede. A RdP é considerada um multi-grafo (já que é permitido múltiplos arcos direcionados saírem de um nó e irem até outros seguindo as restrições de ligação) direcionado (pois os arcos possuem orientações definidas) bipartido (já que os nós são particionados em dois conjuntos, lugares e transições, e os arcos conectam necessariamente elementos de conjuntos distintos) [Murata 1989]. Um exemplo simples da representação gráfica de RdP é ilustrado na figura 2.4 onde todos os arcos possuem peso 1.

A execução da Rede de Petri é representada pelos disparos das transições presentes na rede. Estes disparos são definidos pela distribuição das marcas, que localizam-se nos lugares e condicionam o disparo das transições. A condição necessária para que haja um disparo de uma transição é a existência de marcas em todos os lugares de entrada (pré-condições). Ocorrendo o disparo, as marcas são removidas dos lugares de entrada e são criadas novas marcas nos lugares de saída (pós-condições) daquela transição [Murata 1989]. A figura 2.5 ilustra o disparo de uma transição onde (a) representa o momento antes do disparo e (b) representa o momento após o disparo da transição.

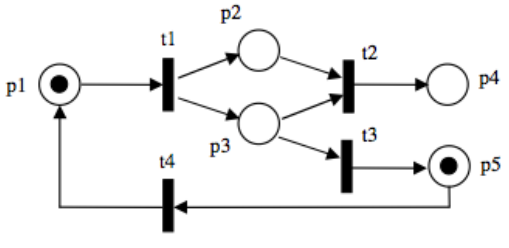

Figura 2.4: Exemplo de rede de Petri.

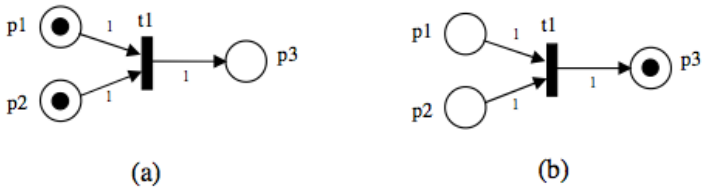

Figura 2.5: Disparo de uma transição em redes de Petri.

2.4.2 Definição Formal

Segundo Murata [Murata 1989], uma rede de Petri é uma 5-tupla, $R = (P, T, F, W, M0)$ onde:

1. $P = p1, p2, ...pn$ é um conjunto finito de lugares, $n \geq 0$.
2. $T = t1, t2, ...tm$ é um conjunto finito de transições, $m \geq 0$.
3. $F \subset (PXT) \cup (TXP)$ é o conjunto de arcos (relação de fluxo).
4. $W : F \rightarrow 1, 2, 3, ...$ é a função peso.
5. $M0 : P \rightarrow 0, 1, 2, 3, ...$ é a marcação inicial da rede.

Uma Rede de Petri com uma marcação inicial é representada por $(N, M0)$. A marcação da uma Rede de Petri (ou seja, o estado da rede) muda de acordo com as seguintes regras de disparo das transições:

- Uma transição t é dada como habilitada se cada lugar de entrada p de t está marcado com pelo menos $w(p, t)$ marcas, onde $w(p, t)$ é o peso do arco de p para t.
- Uma transição habilitada pode ou não ser disparada (dependendo se o evento realmente ocorre ou não).
- Um disparo de uma transição t remove $w(p, t)$ marcas de cada lugar de entrada p de t, e adiciona $w(t, p)$ marcas para cada lugar de saída p de t, onde $w(t, p)$ é o peso do arco de t para p.

2.4.3 Principais Propriedades

A RdP pode apresentar diversas propriedades [Murata 1989]. As propriedades de uma Rede de Petri (que interessam para o presente trabalho), representadas por um conjunto P de lugares, um conjunto T de transições, um conjunto de arcos F (relação de fluxo) que associam os elementos de P e T, e $M0$ a marcação inicial da rede. As propriedades podem ser de dois tipos:

- Dinâmicas: dependem da marcação inicial $M0$. Para esse tipo de análise é possível o uso de grafos para analisar as redes;
- Estruturais: depende da estrutura topológica da rede e não de sua marcação inicial $M0$;

A seguir serão apresentadas as propriedades que interessam para RdPs aplicadas na análise de problemas de planejamento:

Alcançabilidade (Reachability): sendo a marcação inicial $M0$ de uma rede N, uma marcação Mn é alcançável a partir de $M0$ se existe uma seqüência de disparos que transformam $M0$ em Mn. Esta propriedade é bastante utilizada para verificação das propriedades dinâmicas das redes onde são analisados todos os conjuntos de marcações Mn para alcançar um determinado estado de marcação desejado da rede. Existem algoritmos que respondem à seguinte pergunta: Mn é alcançável a partir de $M0$?

Limite de Marcas (Boundness): uma rede$N = (P, T, F, W, M0)$ é k- limitada se o número de marcas em cada lugar p jamais excede k, considerando todos os estados da rede alcançáveis a partir de $M0$. Análises podem ser efetuadas em relação ao número de marcas em uma determinada marcação da rede.

Com estas propriedades é possível efetuar análises, verificações e validações de redes, de forma sistemática, através de ferramentas e algoritmos.

Vivacidade: uma rede $N = (P, T, F, W, M0)$ é viva se, em alguma marcação Mn alcançável a partir de $M0$, qualquer transição poderá ser disparada em Mn ou

em marcações alcançáveis a partir de Mn. O conceito de vivacidade está relacionado com a total ausência de deadlocks na rede. Entretanto, essa propriedade é válida apenas em alguns caso. Por exemplo, considere uma linha de montagem na qual o processo é seqüencial. Neste caso, a rede terá um deadlock - que indicará o final do processo - aceitável. Deadlocks não são aceitáveis em processos cíclicos.

Reversibilidade: seja $R(M0)$ o conjunto de estados de uma rede PN alcançáveis a partir da marcação $M0$. Esta rede será reversível se $M0$ for alcançável por qualquer Mn pertencente a $R(M0)$, ou seja, é sempre possível retornar ao estado inicial da rede.

Cobertura: uma marcação M na rede PN é dita coberta se existe M em $R(M0)$ tal que $M'(p) \geq M(p)$. Isto ocorre para todo p pertencente ao conjunto P, onde $M(p)$ é o número de marcas no lugar p com marcação M.

Paralelismo estrutural: é quando há na rede transições paralelas ou concorrentes sem um lugar de entrada em comum;

Conflito estrutural: é quando as transições com lugares de entrada em comum estão em disputa não determinística de recursos (marcas);

Assincronismo: é quando um lugar possui mais de uma marca disponível, mas somente uma pode ser disparada pela transição.

Hierarquia: é a substituição simbólica de uma região da rede por apenas um lugar, ou transição.

2.4.4 Rede GHENeSys

No trabalho apresentado por Miralles [Miralles 2012] a rede GHENeSys é proposta e detalhada. [Miralles 2012] justifica a concepção da rede GHENeSys com base nas ideias apresentadas em [Miyagi 1988], [Silva e Miyagi 1995], [Silva e Miyagi 1996], [Silva 1998] e finalmente formalizada em [Gonzalez 2001]. Como resultado da formalização feita em [Gonzalez 2001] se obteve uma RdP estendida com conceitos de

orientação a objetos e dotada de mecanismos de abstração e de síntese, através da definição de hierarquia.

Ao contrário de outras redes propostas com características semelhantes, a rede GHENeSys, desde a sua criação, procura introduzir o conceito de objetos no seu modelo semântico e não apenas na implementação da rede. Assim, se obtém um esquema mais propício para aplicação ao processo de projeto em geral e não apenas para a modelagem de sistemas a eventos discretos. A proposta da rede é ter uma representação integrada e formal (com modelo semântico bem definido) para as extensões das RdPs [17].

2.4.5 Definição Original da Rede GHENeSys

A definição atual da rede GHENeSys, tem as características de uma rede lugar/transição estendida, de modo que os sistemas que podem ser modelados utilizando este formalismo restringem-se a sistemas que possam ser representados por uma rede lugar/transição. Porém a estrutura hierárquica e orientada a objetos da rede GHENeSys facilita a modelagem de sistemas de maior porte e a verificação de propriedades sobre esses modelos, assim como a reutilização de elementos extraídos de modelos existente. Isto possibilita que alguns sistemas de grande porte que teriam um modelo complexo, utilizando as redes clássicas, sejam modelados com maior facilidade.

Na definição da rede GHENeSys destacam-se três elementos conceitualmente importantes para obter a modelagem de um amplo escopo de sistemas:

- **Pseudo-lugares:** estes elementos permitem modelar a transferência de informação entre diferentes partes do sistema, permitindo assim a modularização e estruturação do processo de síntese da rede; permitem também representar a interação com o ambiente que envolve o sistema, dado que não existe de fato sistema isolado na natureza [Foyo 2010]. Estes elementos também são utili-

zados para representar informações de eventos observáveis, previsíveis, porém não controláveis [Lin e Wonham 1990], [Ramos e Silva 1998].

- **Hierarquia:** Os elementos chamados *macro* são a base da hierarquia na rede GHENeSys, eles têm a capacidade de representar uma sub-rede. A forma como estão definidos os elementos *macro** permite a preservação das propriedades estruturais e comportamentais da rede global, o que permite a verificação de propriedades globais tendo a certeza que o comportamento das sub-redes não terá influência nessas propriedades. Assim, a rede pode ser construída mesclando elementos simples e *macros* nos diversos níveis de abstração, o que denota sua capacidade de mesclar as abordagens top-down e bottom-up.

- **Abordagem orientada a objetos:** os elementos da rede são definidos como objetos ativos, e as marcas são definidas como objetos passivos (não contém métodos). A rede é formada pela agregação dos objetos que a compoem, e também constitui um objeto. As sub-redes, representadas por elementos *macros*, que tenham estruturas conhecidas e frequentemente utilizadas nos modelos - tais como buffers FIFO, LIFO, entre outros - podem ser substituídas por métodos de classes. Desta forma, é possível a reutilização destas estruturas em outros modelos. Isto permite atenuar o problema da explosão de estados, o que é comum na modelagem de sistemas reais de médio e grande porte. Na figura 2.7 é mostrada a estrutura hierárquica da rede GHENeSys e a figura 2.8 apresenta a estrutura de classes e atributos da rede GHENeSys.

Resumindo, a abordagem orientada a objetos, além de fornecer uma estrutura hierárquica disciplinada, permite a reutilização de modelos *bottom up*, enquanto a modelagem conceitual é feita por refinamentos (*top-down*). Portanto é possível obter uma fusão dos métodos *top-down* e *bottom-up* utilizando o esquema orientado a objetos.

*Na definição original os elementos macro são definidos como elementos *próprios* resgatando as definições formais associadas ao teorema da estruturação de Yourdon e Constatine

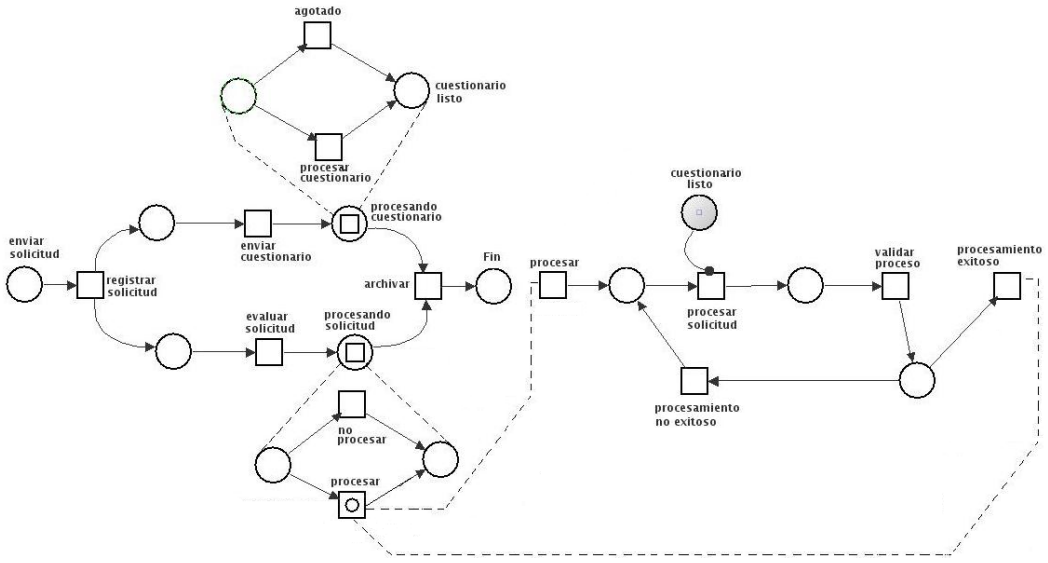

Figura 2.6: Exemplo da hierarquia na rede GHENeSys. Figura retirada de [Miralles 2012].

Além disso, a disciplina de construção dos modelos, unida à prova da conservação de propriedades durante o processo de refinamento, facilita de forma considerável a validação destes modelos, e a definição da rede GHENeSys garante que isto possa ser feito em qualquer nível de abstração.

2.4.6 Redes de Petri em Planejamento Automático

Algumas características das Redes de Petri são muito atrativas para área do Planejamento Automático, tanto para a modelagem e verificação de domínio quanto para o desenvolvimento de técnicas de planejamento. Neste projeto, o formalismo das RdP é aplicado na análise dinâmica dos domínios de planejamento [Vaquero 2011]. Alguns fatores das RdP que fundamentam esta utilização em planejamento são:

- Representa a dinâmica e a estrutura de um sistema segundo o nível de detalhamento desejado (sendo uma rede hierárquica);
- Identifica estados e ações de modo claro e explícito. E tem a capacidade de

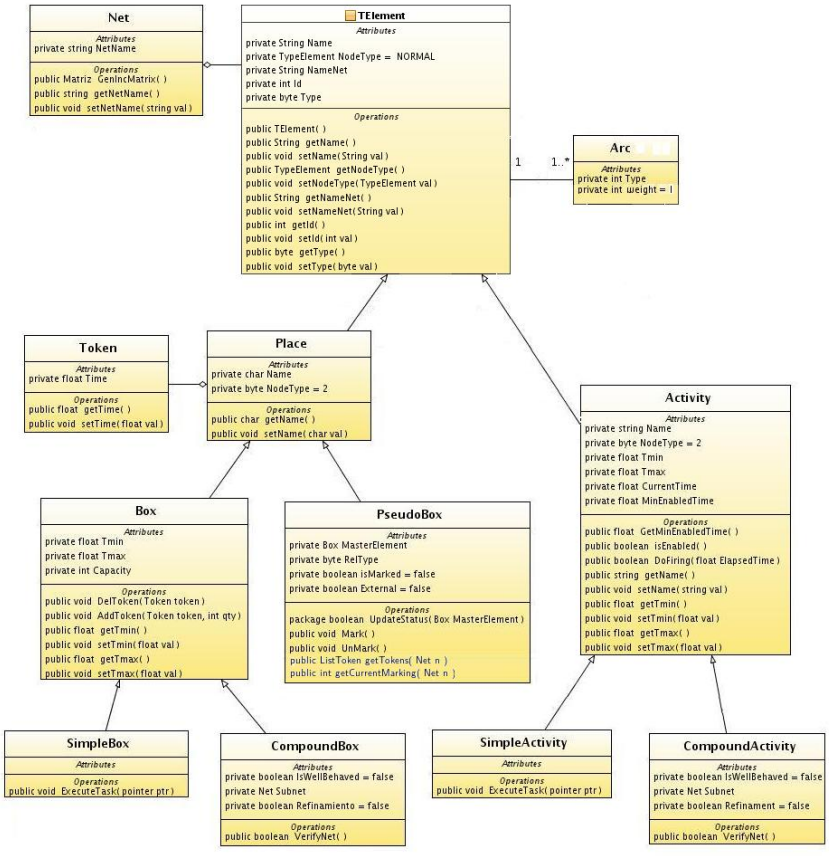

Figura 2.7: Diagrama de classes da rede GHENeSys.Figura retirada de [Miralles 2012].

representar de forma natural as características dos sistemas a eventos discretos SEDs (sincronização, assincronismo, concorrência, causalidade, conflito, compartilhamento de recursos, etc.) muito importante na formulação de um plano;

- Os conceitos envolvidos na representação de um plano são perfeitamente expressos em RdP. Por exemplo, uma característica fundamental existente em um plano é a seqüência de ações (ou eventualmente uma rede de planos alternativos), e esta característica é claramente representada pelas RdP. Podemos citar outras características de um plano como o paralelismo, ou a sincronização de ações;

- Oferece um formalismo gráfico que permite a documentação, visualização, verificação e monitoramento do sistema, facilitando assim o diálogo entre o modelo e projetista que analisa o comportamento dinâmico do mesmo;

- Pode prover uma visão unificada e geral do comportamento dinâmico do modelo do domínio de planejamento;

- Possui uma semântica formal e precisa que permite que o mesmo modelo possa ser utilizado, por exemplo, para análise de propriedades, avaliação do desempenho, bem como para a construção de simuladores a eventos discretos e controladores (para implementar ou gerar códigos para controle de sistemas). A rede também pode servir para verificar comportamentos indesejáveis como bloqueio, limitação, entre outros;

- Incorpora conceitos de modelagem por refinamento (top-down) e do tipo composição modular (bottom-up) através de técnicas como: modularização, reutilização, refinamento;

- Redes de Petri possuem uma ótima capacidade de lidar com recursos. Muitos domínios reais de planejamento possuem o conceito de recursos e, de fato, a habilidade de lidar com recursos é bastante procurada pela comunidade de Inteligência Artificial que está muito alicerçada nas representações lógicas;

- Suas propriedades podem ser utilizadas para realizar análises e verificações sobre o domínio de planejamento;

- Muitos domínios de planejamento são modelados utilizando redes hierárquicas de ações (ou tarefas) e as RdP possibilitam estas representações;

- Algumas propriedades e informações dos modelos em RdP podem contribuir para o processo de planejamento (ex.: alcançabilidade), ou seja, podem ajudar o planejador a evitar buscas indesejáveis;

2.4.7 UML e Redes de Petri na Modelagem de Sistemas

Independentemente de como se faz a especificação final, o sucesso de qualquer projeto depende em grande parte da correta e completa definição dos requisitos, e a especificação dos requisitos serve como um padrão para testar se as fases de projeto e implementação do processo de desenvolvimento de sistemas estão corretas. Esta é uma fase essencial do processo de engenharia de requisitos, que compreende a representação completa do comportamento do sistema, tanto em termos de requisitos funcionais quanto de requisitos não funcionais. A identificação inadequada dos requisitos é uma das principais causas de falhas nos sistemas, e portanto, a eliminação de erros durante esta fase torna-se um processo de vital importância durante o ciclo de desenvolvimento dos projetos.

Muitos pesquisadores têm usado técnicas de modelagem semi formais e formais para capturar os requisitos do sistema seguida de uma fase de análise e verificação desses requisitos que acaba por herdar a excessiva flexibilidade destes métodos.

Os métodos semi-formais em geral se caracterizam por sua simplicidade e flexibilidade na representação dos requisitos, enquanto os métodos formais, ao contrário, se caracterizam por possuir uma forte base formal para simulação e análise. São várias as técnicas e métodos para a análise e especificação dos requisitos propostas na literatura, mas, nenhuma delas garante a completa e correta definição e representação dos requisitos. Existem três desafios fundamentais na análise de requisitos que ainda não foram resolvidos:

- A detecção e análise de requisitos imprecisos e incompletos [Liu e Yen 1996];
- A detecção de inconsistências e o estabelecimento de métodos para gerenciá-las;
- A criação de um processo sistemático que leve dos requisitos especificados informalmente, mesmo que imprecisos, a uma especificação formal.

A estes desafios podemos acrescentar ainda a volatilidade dos requisitos, isto é, ao fato de que requisitos importantes na fase de eliciação e análise podem perder importância durante o próprio processo de projeto, ou ainda, podem praticamente desaparecer ou se fundir com outros.

Um grande desafio seria a detecção prévia, durante a fase de análise de requisitos, desta volatilidade [Silva 2004]. Outro desafio importante é garantir o mapeamento entre os requisitos e os atributos do sistema de modo que, por um lado, todos os requisitos tenham sido atendidos, e por outro que a sua manutenção seja orientada [Vaquero 2011].

Existe a proposta de converter os requisitos semi-formais em uma representação formal, como é o caso de [Baresi e Pezze 2001]. Entre as representações formais mais difundidos estão as Redes de Petri, amplamente usadas na representação e validação dos requisitos e até referendadas pela OMG. As Redes de Petri possuem importantes propriedades que possibilitam a verificação de requisitos.

Muitos pesquisadores tem usado técnicas de modelagem semi-formais como UML para capturar os requisitos do sistemas antes de passar para a fase de análise e verificação dos mesmos [Baresi e Pezze 2001], [Denaro e Pezze 2004], [19], [Salmon 2014]. Por outro lado, as redes de Petri são eficientes na modelagem, análise e simulação de sistemas. Devido a seu aspecto formal, podem ser aplicados algoritmos de análise sobre a rede para verificar e validar propriedades do sistema.

Existem atualmente várias propostas para lidar com a UML, usando a metodologia formal das redes de Petri e suas extensões como uma via para simular e verificar os modelos. Em [Zhao 2004], são discutidas algumas técnicas de transformação de grafos, que podem ser usadas para converter diagramas UML em redes de Petri. .

Os diagramas UML estão inter-relacionados, e suas relações podem refletir na semântica dos diagramas. Portanto, para transformar modelos UML, em uma rede de Petri deve-se levar em consideração tanto a estrutura estática e dinâmica

dos diagramas, como as relações entre eles. Em [Zhao 2004], estas relações foram classificadas em três níveis: as relações entre os mesmos diagramas da UML de diferentes contextos; as relações entre vários diagramas de um mesmo ponto de vista; e as relações entre vários diagramas com pontos de vista diferentes do sistema. Este terceiro nível descreve as relações entre os diagramas de comportamento estáticos e os diagramas de comportamento dinâmicas. Esta terceira abordagem é o foco do presente trabalho.

2.5 Trabalhos Relacionados

Redes de Petri têm uma tradição bem estabelecida como ferramenta de modelagem e validação, e seu uso em planejamento automático já foi sugerido em [Edelkamp e Jabbar 2006], [Hickmott 2007], [Ziparo 2008]. Nesta busca por trabalhos relacionados, a pesquisa foi limitada a utilização de redes de Petri no projeto e validação de modelos que representam sistemas reais. Alguns trabalhos fazem o uso de redes de Petri em Planejamento automático, como é o caso de [Edelkamp e Jabbar 2006], [Hickmott 2007], [Ziparo 2008].

Edelkamp e Jabbar [Edelkamp e Jabbar 2006], trabalham aplicando planejamento via heurística para detectar deadlocks em redes de Petri. As heurísticas são bastante simples, mas com abordagem em model checking verificam estimativas que podem levar a uma redução drástica no tamanho do espaço de estado. Ele opera traduzindo o problema de detecção de deadlocks em um problema de métrica de planejamento, utilizando métodos de heurísticas de busca.

Bonet et. al. [Bonet 2007], investiga "Directed Unfolding", que explora informações específicas do problema na forma de uma função heurística para guiar o desdobramento da rede até a marcação desejada. O desdobramento de uma rede de Petri revela todas as possíveis execuções parcialmente ordenadas, sem a intercalação combinatória de eventos independentes. Os autores acreditam que a razão pela qual tal estratégia não foi considerada antes pode ser porque Directed Unfolding normal-

mente é usado para provar a ausência de deadlocks. Inspirados pelas heurísticas de busca em Inteligência Artificial, particularmente na área de planejamento automático, Directed Unfolding explora informações específicas do problema na forma de uma função heurística que guia o processo de busca até a marcação desejada.

Ziparo et. al. [Ziparo 2008], desenvolveram um framework (dependente de domínio) baseado em redes de Petri para descrever o comportamento de agentes robóticos. O formalismo Petri Net Plan (PNP), permite criar uma descrição em alto nível de interações complexas das ação que são necessárias na programação de robôs cognitivos, que habitam um ambiente parcialmente observável e imprevisível.

No presente trabalho propomos uma disciplina de projeto em planejamento automático, baseada no uso de modelos hierárquicos, que utiliza a UML como ferramenta de modelagem. Os diagramas gerados para representar a aplicação de planejamento são traduzidos numa rede de Petri Hierárquica. Esta rede é analisada com o intuito de validar o modelo desenhado. Uma vez que a rede esteja corretamente modelada, ela é traduzida para HTN e então apresentada ao planejador JSHOP2. Note que o uso de redes de Petri (combinado com a UML), no presente trabalho, fica restrito a etapa de modelagem da aplicação de planejamento. O objetivo é utilizar a UML, uma ferramenta de alto poder expressivo, para representar problemas de engenharia e então validar esses modelos usando redes de Petri. A proposta inicial trabalha com redes hierárquicas, apresentadas anteriormente.

No próximo capítulo será apresentada a disciplina de projeto desenvolvida, apresentando o processo de projeto e o método de modelagem usando a UML, bem como o algoritmo de tradução dos diagramas UML para uma rede hierárquica.

Capítulo 3

O Processo de Projeto DynA (Dynamic Analyser)

3.1 Apresentação

A aplicação das técnicas de planejamento inteligente para sistemas reais requer um processo diferente daquele aplicado na solução de problemas acadêmicos. Sistemas reais devem seguir um processo de projeto bastante disciplinado, fundamentado na Engenharia do Conhecimento, cuja etapa inicial é composta da eliciação e análise de requisitos. Atualmente existem poucos trabalhos de pesquisa sobre projeto de sistemas reais na área de planejamento automático. Além disso, após pesquisa bibliográfica, não foram encontrados trabalhos que tenham foco no uso de UML e Redes de Petri para o projeto e validação de um modelo convergente e rico que possa melhorar a qualidade das respostas dos planejadores. O presente trabalho parte do pressuposto de que este é um tema emergente, motivado pela tendência recente de se utilizar planejamento em aplicações reais. Tais aplicações são extremamente sensíveis ao conhecimento modelado (tanto do domínio quanto do problema em si) e aos processos de validação, que podem economizar tempo e custo no ciclo de vida de projeto.

Com base na revisão da literatura apresentada no Capítulo 2, o presente capítulo apresentará um processo de projeto desenvolvido para a modelagem de problemas reais em planejamento automático. Este processo combina a modelagem visual (usando UML) com o formalismo matemático (proveniente das Redes de Petri). O objetivo é oferecer um método que guie o usuário no projeto do modelo. Os resultados gerados no decorrer deste trabalho mostraram a necessidade de criar um método diferente do apresentado no framework itSIMPLE. Isto ocorreu por causa das limitações encontradas no framework e também por causa da evolução do projeto inicial, que acabou levando para um processo bastante distinto, como será mostrado no decorrer deste capítulo. O objetivo é promover um ciclo de vida de projeto onde se pode analisar, validar e avaliar o modelo do sistema e do domínio (bem como da interação entre eles) através de propriedades estabelecidas pela teoria de Rede de Petri.

Mais especificamente, o processo de projeto pressupõe a existência do domínio de discurso ou domínio de conhecimento do problema (composto de todo o conhecimento adquirido sobre um tipo específico de problema e dos casos já solucionados); do domínio de trabalho, ou ambiente de trabalho (composto dos elementos e especificações (restrições)) impostos pelo ambiente onde será implementado o planejador; e pelo problema de planejamento propriamente dito. O projeto associado a problemas de planejamento automático (e escalonamento) consiste em levantar os requisitos (ou reutilizar) e modelar o domínio de trabalho, modelar o problema de planejamento e verificar se ambos formam um modelo consistente e convergente (isto é, logicamente aceitável, sem contradições e que resulte em uma rede (ou grafo) simples e conectada) harmonizando todos os pontos de vista [91]. Outras propriedades também podem ser analisadas como vivacidade, invariantes de lugar e transição, equidade (fairness), dentre outras. Como resultado temos a validação dos modelos de referência do domínio de trabalho e do problema de planejamento e da Rede de Petri que os unifica.

Durante a atividade de modelagem, ajustes podem ser feitos nos modelos de

referência, após os resultados adquiridos através da análise via RdPs. Este trabalho contribuirá também para que exista uma disciplina de projeto em planejamento de sistemas do mundo real, otimizando as soluções produzidas com relação aos critérios de convergência e inclusão das características dos diversos pontos de vista de um projeto.

3.2 O Método DynA

Antes de apresentar o método DynA, é importante delimitar o escopo de aplicação e como o conhecimento referente a essa classe de problemas será tratado. Como foi apresentado no capítulo 2, este trabalho visa a modelagem, análise e validação de modelos que representem um fluxo de processo de problemas reais. Como por exemplo, manufatura, logística e controle de tráfego aéreo.

O conceito de planejamento automático e suas aplicações se encaixam perfeitamente nessa classe de problemas. Dito isto, vale ressaltar que as aplicações de planejamento têm duas classes diferentes de requisitos, que são: requisitos de domínio e requisitos de problema [94]. Neste trabalha, é proposto uma evolução desta suposição, baseada em [91]. A proposta é dividir o processo de projeto considerando dois aspectos: 1) os aspectos do domínio de discurso (work domain), onde todas as características essenciais são consideradas (tais como: nome, restrições, operações, ações gerais e descrições do ambiente que são críticas para o sistema); e 2) os aspectos do problema de planejamento, onde o estado inicial, o estado final e um conjunto de objetos que compreendem a instância de um problema, como é mostrado na figura 3.1. Baseado nesta hipótese de independência, o processo de projeto é feito.

Uma vez que o projeto e o escopo do problema foram contextualizados, é possível estruturar o processo de forma clara apontando as principais diferenças entre a versão clássica da ferramenta itSIMPLE e o proposto neste trabalho. A figura 3.2 apresenta o método clássico usado no itSIMPLE, anterior a este trabalho.

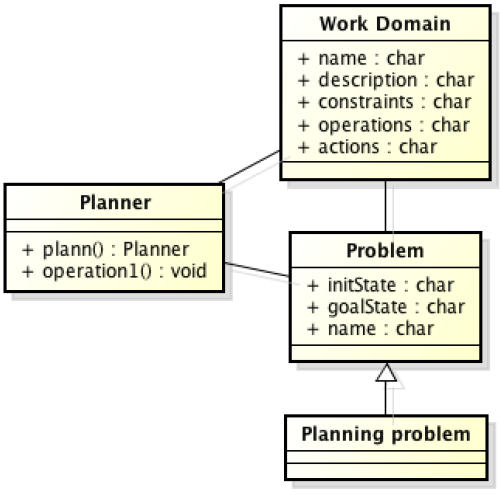

Figura 3.1: Esquema geral para aplicações de planejamento.

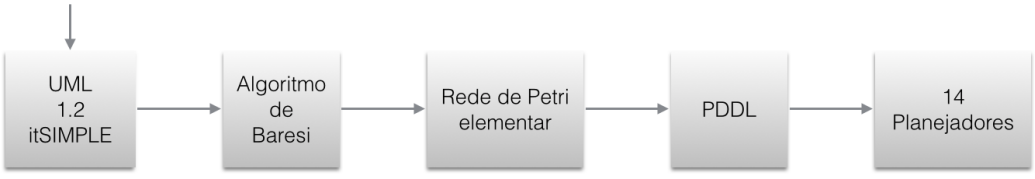

Figura 3.2: Processo de projeto clássico do itSIMPLE.

As etapas apresentados na figura 3.2 refletem o uso do itSIMPLE na sua forma clássica. Uma revisão desses passos e os estudos de caso realizados motivaram alterações no processo de projeto original usado no itSIMPLE, que serão mostradas detalhadamente mais adiante. No entanto, uma breve inspeção na figura 3.2 mostra que, conceitualmente - independentemente da evolução da versão da UML ou das redes de Petri*, ou até mesmo a inclusão de novas propostas de planejadores, os pontos cruciais são:

- a transferência entre UML e Redes de Petri, o que remete à disciplina no uso

*As redes de Petri vêm sendo revistas de 2004, com a composição de um padrão ISO/IEC 15.909 que separa as redes clássicas e de alto nível como proposta básica, estabelece um padrão de linguagem de transferência baseada em XML, e as possíveis extensões, incluindo-se aí a inserção de hierarquia e tempo

dos diagramas para evitar redundância, ao algorítimo utilizado - no caso o algoritmo de Baresi - e à possibilidade de perda semântica no processo;

- as limitações da PDDL como linguagem de especificação, para, por exemplo incluir abstração, hierarquia e tempo (neste trabalho não incluiremos o problema do tempo);
- a existência de novos planejadores que possam de fato usar estes parâmetros novos, hierarquia e propriedades (como invariantes, simetria, etc.) na elaboração de planos melhores.

Para compor uma nova disciplina de projeto é necessário abordar todos estes itens, entretanto, a contribuição original deste trabalho se refere apenas aos dois primeiros, excetuando o problema do tempo. Assim, no framework DynA escolhemos uma nova linguagem de especificação que atende aos objetivos e um novo planejador que atende aos mesmos critérios.

A partir disto, é possível apresentar a arquitetura do framework DynA de modo mais coerente. A próxima seção detalha o método proposto no presente trabalho.

3.2.1 Detalhando o Método DynA

O presente trabalho visa auxiliar os usuários na etapa de análise de requisitos e design de modelos em planejamento automático. Os domínios de planejamento almejados são aqueles em que o processo de planejamento é recursivo e sua sucessiva aplicação pode gerar melhorias no modelo. Podemos encontrar exemplos de domínios com estas características de planejamento em problemas de manufatura, localização de robôs autônomos, atividades portuárias, logística, dentre outros [Vaquero 2007].

Neste trabalho pretende-se utilizar esta recursividade do processo de projeto para que conhecimentos mais gerais sejam capturados e utilizados de modo que os planos possam ser produzidos com alta qualidade em relação aos critérios de qualidade definidos pelos diferentes pontos de vista de um projeto.

O objetivo foi desenvolver um método que guie o projeto do sistema e contemple (i) a fase de eliciação e documentação dos requisitos, tendo uma adaptação da UML 2.4 [OMG 2011] como base de representação, especialmente os Diagramas de Estado Comportamental (UML 2.4), Classe e Objetos; (ii) a transferência e síntese destes diagramas para uma Rede de Petri Hierárquica, que será analisada no sistema GHENeSys. Para tal, serão levados em consideração os trabalhos e métodos existentes na literatura (apresentados no Capítulo 2) e, principalmente, a plataforma GHENeSys também desenvolvida no labaratório de pesquisa do qual este trabalho faz parte (Design-Lab). Por questões de incompatibilidade da versão da UML e do tipo de planejamento adotado (planejamento hierárquico) não foi possível usar a ferramenta itSIMPLE para testar o método proposto.

Com o objetivo de fornecer uma visão geral do método, a Figura 3.3 ilustra o papel desta proposta no processo de construção e melhoria de modelos do conhecimento do domínio de trabalho e do problema de planejamento. Nesta figura pode-se observar que o método visa contribuir no processo de projeto [Tonaco-Basbaum 2013] [88]. O GHENeSys será usado para realizar a análise da Rede de Petri que representará o modelo unificado. Os principais passos que compreendem o processo de projeto apresentado neste capítulo são apresentados na figura 3.3.

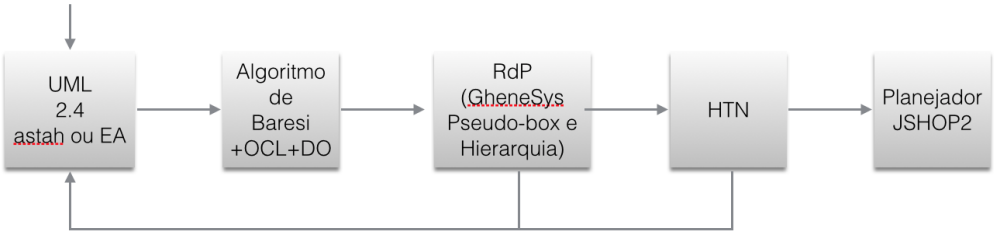

Figura 3.3: Etapas de projeto do método DynA.

O usuário desenvolve o modelo usando a UML como linguagem de representação (este processo será explicado com mais detalhes nas seções seguintes. O modelo UML é convertido numa Rede de Petri. Para realizar a análise da Rede de Petri é usado o framework GHENeSys. A partir do resultado desta análise, o usuário decide

alterar o modelo UML, ou não. O objetivo é resolver os problemas encontrados no modelo (caso haja algum problema).

A análise dos requisitos é feita depois que o Diagrama de Estados Comportamental foi sintetizado em uma rede de Petri Hierárquica. A primeira etapa detecta basicamente contradições e conflitos entre os diversos requisitos ou os diferentes pontos de vista refletidos nos diagramas. A segunda fase detecta inconsistências mais profundas, que se escondem na perspectiva dinâmica do plano que está sendo especificado, bem como informações adicionais ao modelo que podem ser interpretadas pelos planejadores de modo a melhorar o desempenho do processo de planejamento automático. Entre estas informações adicionais inclui-se novas restrições, invariantes, estratégias parciais de solução, características e soluções específicas que podem contribuir para uma melhor resposta do planejador, seja com relação ao tempo de processamento, ou com a qualidade dos planos gerados.

Os domínios mais complexos são difíceis de modelar sem o uso de abstração. Nestes domínios, uma decomposição hierárquica do problema baseada numa estrutura topológica pode levar a um melhor desempenho dos planejadores. Estudos de casos anteriores, usando domínios complexos como experimento, mostraram que a adoção de estruturas hierárquicas resulta em modelos mais fiéis ao problema real.

Em problemas padrões de planejamento, como por exemplo problemas de logística, a abstração topológica do problema real é parte inerente da definição do domínio de discurso. Num problema de logísitica grande, vários pacotes devem ser entregues saindo de um local inicial para vários outros locais de destino. O problema de logísitica tem um mapa das cidades conectadas por rotas de avião. Transportes dentro das cidades podem ser feitos por meio de um caminhão (existe um caminhão em cada cidade). As cidades podem ser abstraídas e tratadas como caixas pretas. Dentro das cidades, um caminhão pode ir de um lugar para vários outros, [9]. Neste contexto, removendo o conhecimento humano pode-se obter um modelo abstrato do mundo real.

Por estas razões, após vários estudos de caso, optou-se pelo uso de modelos hierárquicos para a representação de problemas do mundo real, especialmente no contexto de planejamento automático.

3.3 Modelagem do Problema Usando UML

Uma das principais características da UML é a possibilidade de modelar e analisar problemas a partir dos pontos de vista oferecidos pelos diversos diagramas da linguagem. Se o artefato (ou processo, ou ainda o plano) em desenvolvimento deve satisfazer a várias classes de atores, a importância dos pontos de vista é clara, dado que o artefato final (unificado) precisa atender a todos os pontos de vista ao mesmo tempo, ou a algum resumo negociado destes pontos de vista. Entretanto, mesmo que este não seja o caso e que se identifique somente uma classe de atores, a modelagem baseada num único ponto de vista é perigosa e pode levar ao retrabalho. Nesta situação uma abordagem abstrata da classe de atores pode levar a um modelo onde ainda existem várias possibilidades de solução que devem ser eliminadas até que reste apenas uma. Este resultado é amplamente conhecido na Engenharia Concorrente e na Engenharia de Design.

Na versão 4.0 do framework itSIMPLE [Vaquero 2012] o usuário pode modelar a aplicação de planejamento usando o Diagrama de Casos de Uso, o Diagrama de Classes, o Diagrama de Estados e o Diagrama de Objetos. Entretanto, estudos de caso realizados com problemas reais, apresentados no ROADEF Challenge - como controle de tráfego aéreo [Palpant 2009] e sequenciamento de carros [Nguyen 2005] - mostraram que existe uma deficiência de informação no modelo desenvolvido usando o método de projeto que o itSIMPLE adota em sua versão 4.0. Portanto, a partir desses estudos de caso, pôde-se comprovar e justificar a necessidade de uma nova disciplina de projeto para guiar o desenvolvimento dos diagramas UML.

Diante disso surgiram algumas questões: ”Qual o conjunto mínimo de diagramas UML necessário para representar problemas de planejamento?”; ”Qual disciplina

de projeto adotar para o desenvolvimento de tais diagramas?". A resposta para estas questões, tem o intuito de agrupar diagramas com visões diferentes e complementares, além de otimizar o processo de projeto. Assim, propõe-se um conjunto mínimo capaz de representar problemas de planejamento de maneira otimizada.

Para isto, alguns testes foram feitos no decorrer do desenvolvimento deste trabalho. Mas um primeiro passo se manteve presente e necessário desde o princípio. As aplicações de planejamento devem ser divididas em duas partes, que são: domínio de trabalho (ou de aplicação) e problema de planejamento [91]. Assim, identifica-se inicialmente quais requisitos podem ser representados em cada um desses universos e depois disso sua convergência.

A necessidade de alterar o conjunto mínimo de diagramas UML foi justificada pela deficiência que o modelo de tradução dos diagramas para Redes de Petri apresenta no itSIMPLE 4.0. A versão anterior a este projeto considerava apenas o diagrama de Estados na tradução para uma rede de Petri elementar, e acreditava-se que o problema no algoritmo de tradução era a falta de informação vinda do modelo UML. Diante deste raciocínio propôs-se a inclusão de novos diagramas.

O itSIMPLE na sua versão anterior ao presente projeto, baseia-se na especificação da versão 1.2 da UML. Nesta versão os diagramas de Estados são desenhados de forma separada para representar o fluxo de vida de cada objeto ativo, gerando uma representação fragmentada dos aspectos dinâmicos do sistema. Essa abordagem se mostrou ineficiente no processo de tradução do modelo UML para uma Rede de Petri, pois um modelo UML fragmentado gera uma representação em redes de Petri igualmente fragmentada, o que é inviável para a análise dinâmica do sistema como um todo.

Uma pesquisa sobre a versão mais recente da UML (UML 2.4) mostrou que existe uma maneira de representar o sistema todo usando uma das variações do diagrama de Estados (diagrama de Estados Comportamental) que contempla o uso de estruturas hierárquicas para definir o modelo. Este pode ser usado para modelar

o comportamento discreto através de transições de estados finitos, tal característica se encaixa muito bem na maior parte das aplicações de planejamento. Com isto, percebeu-se que o diagrama de Estados Comportamental seria suficiente para representar os aspectos dinâmicos do sistema. Tal constatação reduziu significativamente o conjunto de diagramas UML usado para representar problemas reais em planejamento, e foi base para uma nova proposta de disciplina de projeto que se será apresentada a seguir. Portanto, o conjunto mínimo de diagramas UML é composto por: Diagrama de Pacotes, Diagrama de Classes, Diagrama de Estados Comportamental e Diagrama de Objetos. A seguir será apresentada detalhadamente a disciplina de projeto que deve ser usada para gerar o modelo UML. A figura 3.4 mostra o fluxo das etapas definidas na modelagem dos diagramas UML.

Pré-condição de modelagem: o sistema deve possuir componentes distintos que se inter-relacionem.

1 Definir Componentes e relações;

2 Desenvolver o diagrama de Classe.

2.1 Se necessário, divida o diagrama de Classe em módulos (Pacotes) para facilitar a modelagem do sistema.

3 Identificar no diagrama de Classe os objetos dinâmicos do sistema.

3.1 Usar OCL para especificar restrições.

4 Desenvolver o diagrama de Estados Comportamental, representando estados compostos (ou super estados) que darão origem a estados internos. Defina todos os níveis de abstração do sistema, formando assim um diagrama hierárquico.

4.1 Se existe um estado composto (ou super estado), então, refinar o estado em outros diagramas.

4.2 Usar OCL para especificar restrições no diagrama de Estados.

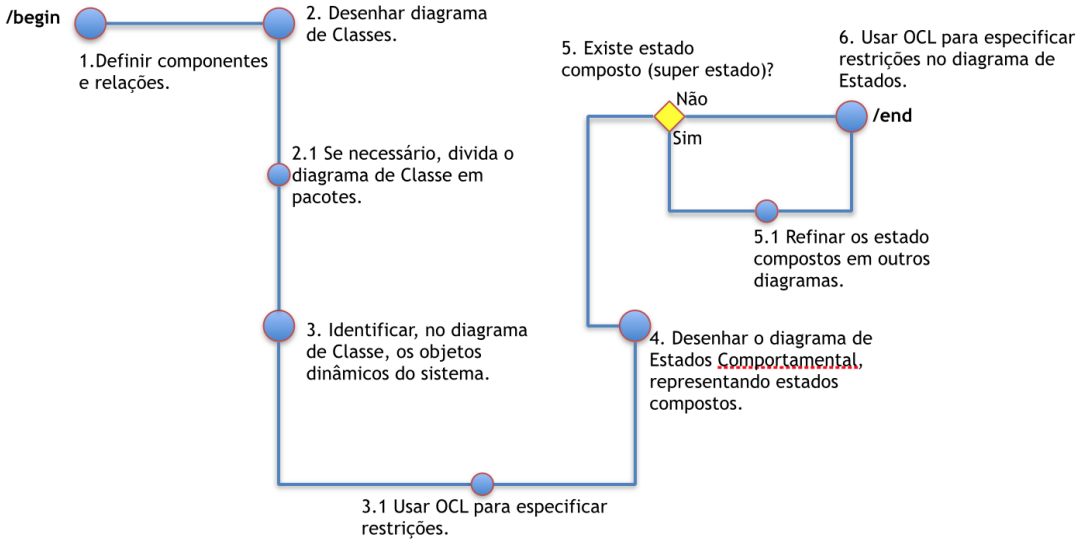

Figura 3.4: Fluxo das etapas definidas no processo de projeto do modelo UML.

Uma vez que o modelo UML está concluído, deve-se traduzí-lo para uma rede de Petri hierárquica que represente de forma única o sistema.

Com esta abordagem elimina-se o risco de perda dos pontos de vista individuais, o que tornava a identificação de erros específicos em cada diagrama complicada após a etapa de modelagem em UML. As alterações apresentadas neste trabalho, apesar de sutis com relação ao processo de modelagem adotado nas primeiras versões do itSIMPLE, são suficientes para gerar um modelo conciso e fiel ao problema que se pretende modelar, além de oferecer uma disciplina de projeto que guie o usuário na modelagem do problema. Mais adiante serão apresentados estudos de casos que comprovam a eficiência deste método.

3.4 Análise de Requisitos Usando Redes de Petri

O uso de Redes de Petri é bastante promissor e pode possibilitar a redução do ciclo de melhoria dos planos até atingir um estado aceitável. Além disso, pode-se realizar a verificação e a validação do modelo desenvolvido para representar o

sistema. A figura 3.5 mostra o ciclo de vida do planejamento automático. Nele é possível observar em qual etapa do processo de projeto as Redes de Petri serão utilizadas.

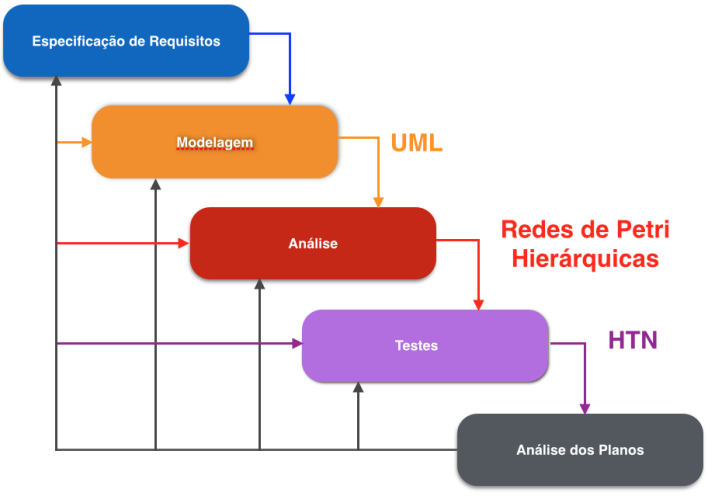

Figura 3.5: Ciclo de Vida do Planejamento Automático. Figura adaptada de [Vaquero 2007].

Muitos dos trabalhos pesquisados até o momento, usam redes de Petri para melhorar o desempenho do planejador automático, como é o caso de Hickmott et. al. [Hickmott 2007], que propõem o uso de Redes de Petri para diminuir o espaço de estados de busca do planejador automático. Entretanto, pouco se tem encontrado, até o presente momento, sobre o uso desta técnica na análise de requisitos em problemas de planejamento. Isso ocorre porque grande parte das aplicações geradas na área de planejamento automático são aplicações de Inteligência Artificial, nas quais o problema de planejamento é sintetizado apenas para testar a capacidade do planejador. Recentemente, tem surgido o interesse em problemas do mundo real, o que muda significativamente o cenário da pesquisa em planejamento [Vodrazka e Chrpa 2010].

Diante disso, torna-se necessário realizar a análise de requisitos dado que sistemas reais são complexos e requerem um processo diferenciado dos problemas-modelo. Uma boa ferramenta para a validação dos requisitos é a Rede de Petri

[Santos e Silva 2005], pois permite uma representação geral do sistema além de ser um bom formalismo para sistemas reais, porque representa a dinâmica das ações.

No itSIMPLE a tradução do modelo UML para Redes de Petri era feita através do diagrama de Estados. Tal processo de tradução é apresentado em [Vaquero 2007]. Na figura 3.6 pode-se observar um exemplo simples de como essa tradução ocorre.

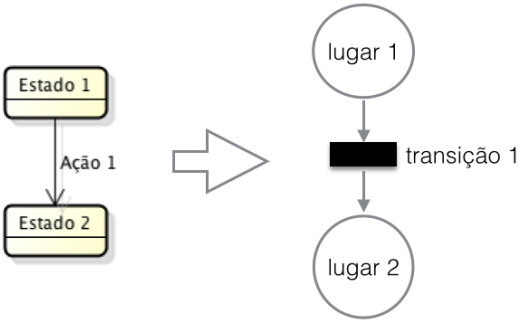

Figura 3.6: Tradução do Diagrama de Estados em Rede de Petri. Figura adaptada de [Vaquero 2007].

Nesta abordagem, que segue o algoritmo de Baresi [Baresi e Pezze 2001], uma Rede de Petri é gerada para cada diagrama de Estados do modelo. Entretanto, essa abordagem não permite realizar a análise do sistema como um todo, então encontra-se os pontos de interação das redes de Petri para juntá-las numa única rede. As redes derivadas a partir da abordagem inicial do itSIMPLE, apenas mostram o ciclo de vida e as transições indicam os métodos definidos no diagrama de Classe. Além disso, não apresentam a comunicação dos objetos entre si e como seria a interação caso existisse. Esta representação usando o diagrama de Estados é fragmentada, usando apenas o algoritmo de Baresi não é possível representar na rede as restrições definidas em OCL.

Os estudos de caso mostraram que esse método também possuía algumas limitações e um novo algoritmo de tradução usando o diagrama de Estados Comportamental foi testado e mostrou resultados satisfatórios. Esse algoritmo é uma variação do algoritmo de Baresi, onde as restrições em OCL também são traduzidas em lu-

gares na rede de Petri e os objetos instanciados no diagrama de Objetos indicam a multiplicidade do problema. A figura 3.7 apresenta o fluxo de atividades para o processo de tradução do modelo UML em redes de Petri.

Nesta primeira parte do algoritmo, considere apenas o diagrama de Estados Comportamental:

- Os estados do diagrama são modelados como lugares na rede de Petri.
- As ações no diagrama são modeladas como transições na rede de Petri.
- Se os estados do diagrama possuem restrições formuladas em OCL, representando pré-condições, ou pós-condições, estas serão modeladas como um lugar na rede que representa o respectivo estado do diagrama.
- As relações entre estados e ações no diagrama serão modeladas como arcos entre os lugares e as transições correspondentes na rede de Petri.
- Os estados compostos (ou super estados) no diagrama serão modelados como macro elementos na rede de Petri e então refinados até que atinjam o nível mais elementar do sistema.

Nesta última parte do algoritmo, considere apenas o diagrama de Objetos:

- Os objetos instanciados no diagrama indicarão a multiplicidade do processo na rede de Petri.

A idéia é usar apenas os diagramas de Estados Comportamental e o diagrama de Objetos na tradução de UML para rede de Petri. O objetivo é traduzir o diagrama de Estados Comportamental para uma rede hierárquica. Tal rede poderá ser convertida numa rede colorida, caso surja a necessidade, embora este processo não faça parte do escopo do presente trabalho.

Uma vez que a rede de Petri esteja definida, verifica-se a existência de conflitos mais diretos e triviais, como a presença de deadlocks, por exemplo. Após isso realiza-se uma análise semântica que tem o objetivo de verificar se a rede representa de fato

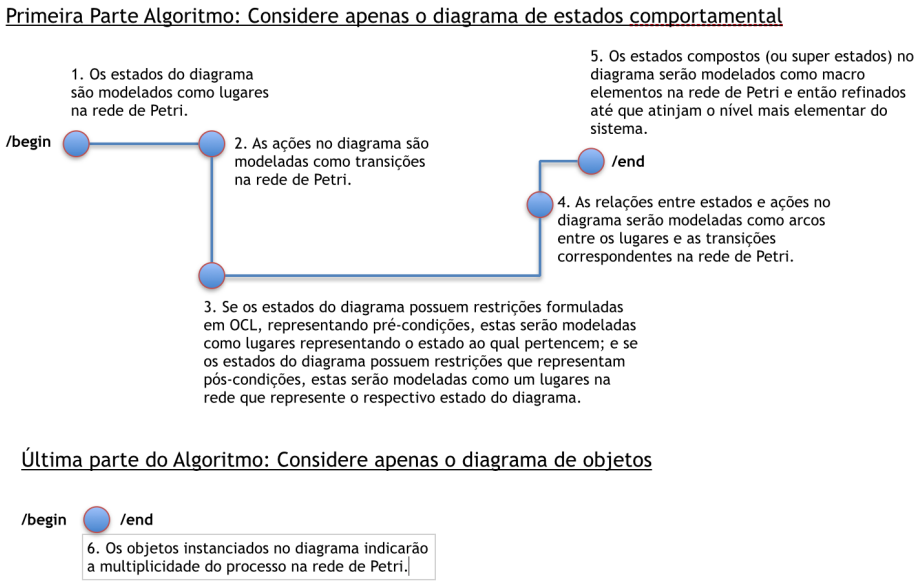

Figura 3.7: Fluxo de atividades para o processo de tradução do modelo UML em redes de Petri

o problema de planejamento. Esta análise semântica pode ser realizada observando as propriedades comportamentais e estruturais da rede, explicadas no capítulo 2. Além da identificação das propriedades citadas anteriormente, ainda deve-se realizar a análise de invariantes, análise da matriz de incidência e equação de estados, e a análise do grafo de cobertura da rede.

Redes de Petri são ferramentas gráficas e matemáticas que representam atividades dinâmicas e concorrentes de sistemas [Murata 1989]. Isso justifica o uso dos diagramas citados. O processo de projeto neste caso tem duas partes importantes: de um lado a fase de eliciação e documentação dos requisitos, tendo a UML 2.4 [OMG 2011], [Sommerville, I and Sawyer, P 2007], como base de representação; do outro lado temos a transferência e síntese destes diagramas em uma rede de Petri (sistema GHENeSys). O uso do GHENeSys, nesta etapa do projeto, ainda é justificado por outros dois fatores importantes: representar problemas complexos em Redes de Petri lugar/transição é bastante complicado, já que a rede pode crescer exponencialmente, e não existe a possibilidade de representar multiplicidade na rede,

por isso é necessário utilizar uma rede de alto nível para representar o modelo.

Após os resultados da análise da Rede de Petri o usuário pode interpretar as informações vindas do GHENeSys, e fazer as devidas críticas e alterações. Com um modelo mais aderente ao problema real, espera-se melhorar a performance do processo de planejamento automático, seja no quesito tempo de processamento quanto qualidade dos planos gerados.

3.4.1 Discussões

O método proposto no presente trabalho aborda problemas que seguem os conceitos de estado transição [Baresi e Pezze 2001], [Bonet 2007], já bastante discutidos na comunidade de planejamento automático, e que possuem características dinâmicas. Esses conceitos são tratados usando Redes de Petri para representar e analisar o modelo.

O primeiro estudo de caso, usando um problema de complexidade relativa, mostrou a impossibilidade de reproduzir o processo de projeto, apresentado neste trabalho, no itSIMPLE. Isto ocorreu por várias razões que devem ser apontadas:

- Incompatibilidade de versões da UML. Esse ponto é determinante, pois sem a representação hierárquica no modelo UML todo o restante do processo fica comprometido. O primeiro estudo de caso, que será apresentado no próximo capítulo, foi feito com o intuito de forçar a hierarquia no modelo UML com o intuito de testar o processo de projeto no itSIMPLE. Porém, os resultados atingidos não foram satisfatórios.

- Um modelo UML sem a correta representação hierárquica interfere no algoritmo de tradução de UML para redes de Petri.

- A PDDL não suporta modelos hierárquicos [McCluskey e Simpson 2003] e o itSIMPLE usa planejadores que aceitam exclusivamente a linguagem PDDL.

Alguns autores tentaram desenvolver extensões da PDDL com o objetivo de adicionar esruturas hierárquicas no modelo [9]. Mas, para problemas complexos usar uma linguagem que seja naturalmente hierárquica se mostrou mais eficiente. Por isso escolheu-se usar HTN [Nau 2005] no lugar da PDDL.

- O JSHOP2 não faz parte do conjunto de planejadores acoplados ao itSIMPLE, pois este aceita apenas a linguagem HTN.

O próximo capítulo justifica o uso do processo de projeto fora do itSIMPLE e exemplifica a proposta apresentada neste trabalho. Os estudos de casos foram escolhidos e desenvolvidos com o intuito de ilustrar os pontos discutidos neste capítulo e verificar as hipóteses feitas.

Capítulo 4

Estudos de Caso

Os resultados apresentados nesta seção espelham os diversos estudos de caso que foram realizados no decorrer do desenvolvimento deste projeto. Foram escolhidos dois domínios para ilustrar o método proposto: um de complexidade intermediária, proposto pelo ROADEF 2005 [Nguyen 2005]; e outro de alta complexidade, que trata a logística de portos nas plataformas do Pré-Sal [Vaquero 2012].

4.1 ROADEF 2005 - Sequenciamento de Carros

Este problema consiste no trabalho diário de uma fábrica que tem duas funções principais: atribuir um dia de produção a cada veículo encomendado, de acordo com a capacidade da linha de produção; e programar a ordem em que os carros devem ser colocados na linha de produção, enquanto satisfaz o maior número de requisitos dos processos de carroçaria, pintura e montagem.

É objetivo deste trabalho modelar o planejamento da fabricação dos carros, bem como mostrar a modelagem do processo de fabricação. Os carros são de diferentes tipos. Na modelagem deste sistema um dos requisitos fundamentais é garantir que a linha de montagem não possa ter mais de um carro em cada processo. Na especificação do problema não existem restrições para a montagem da carroçaria,

portanto foram considerados os detalhes da modelagem deste processo. A seguir serão apresentados o modelo UML, a rede de Petri e a análise da mesma, que foi feita no GHENeSys. Este estudo de caso teve o intuito de testar a nova disciplina de projeto no itSIMPLE. Portanto, implementou-se um algoritmo de tradução no itSIMPLE, seguindo a proposta apresentada na seção anterior.

4.1.1 Modelo UML

Como dito anteriormente, o itSIMPLE usa a versão 1.4 da UML e este trabalho é baseado nos conceitos apresentados na especificação 2.4 da UML. Como no itSIMPLE não é possível usar os novos elementos do diagrama de Estados, foi necessário forçar o conceito de hierarquia (especialmente os estados compostos). Esta abordagem foi escolhida para testar o processo de tradução do itSIMPLE (de UML para PDDL), que já é bem conhecido e solidificado.

As figuras 41. e 4.2 mostram o diagrama de Classe e o diagrama de Estados Comportamental (neste exemplo foi usado super estados para modelar o conceito de hierarquia que foi mostrado anteriormente), respectivamente. Isto foi feito porque o itSIMPLE não possui os elementos necessários para representar os estados compostos, para que o método apresentado neste pudesse ser testado nesta ferramenta, a estrutura hierárquica do modelo foi feita através do uso de super estados.

Com esses dois diagramas o usuário pode desenhar o domínio de discurso e o problema de planejamento como foi explicado anteriormente. Entretanto, para completar o modelo incluindo a instância do problema é necessário desenhar outros dois diagramas (figuras 4.3 e 4.4). Em outras palavras, um diagrama de Objetos para mostrar o estado inicial e outro com o estado meta. Com estes diagramas tem-se a representação completa do sistema real. Observe que desenhar o sistema num único diagrama de Estados Comportamental é uma abordagem mais intuitiva e o processo de tradução de UML para redes de Petri acontece diretamente.

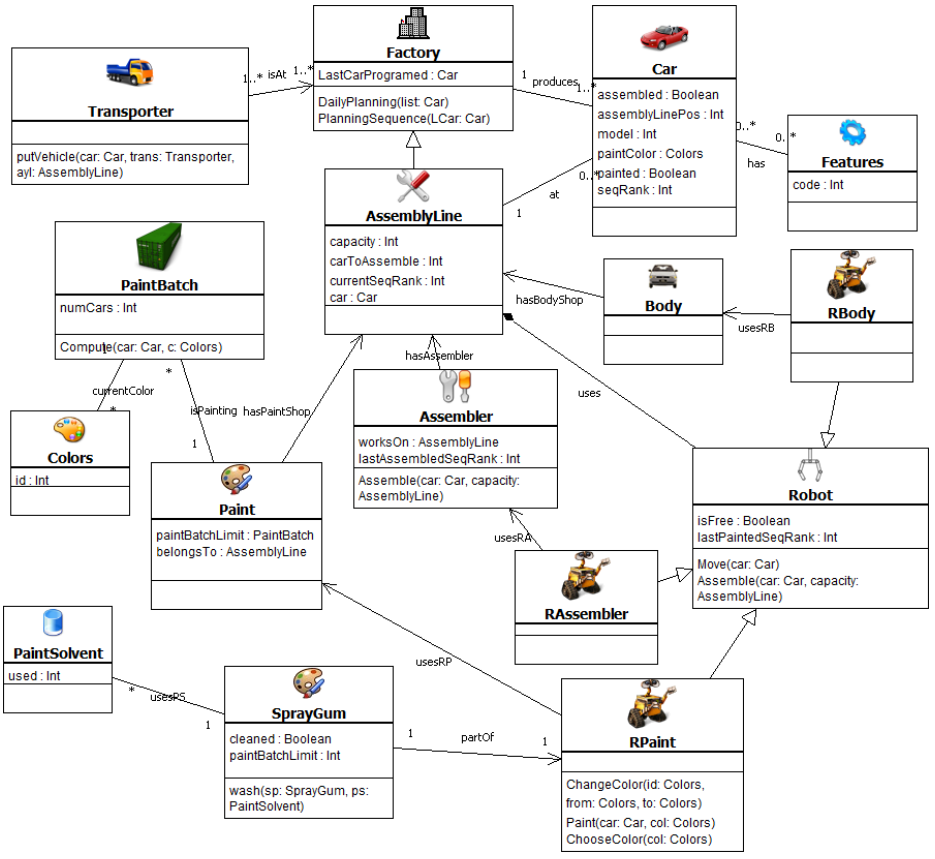

Figura 4.1: Diagrama de Classe para o desafio ROADEF 2005.

4.1.2 Transformando os Diagramas UML em Redes de Petri

Neste exemplo, capacidade da linha de produção é três. Os carros são programados numa ordem particular, de acordo com as restrições da linha de produção. Seguindo o algoritmo de tradução de UML para redes de Petri, um lugar deverá ser associado à linha de montagem, outro para a pintura e outro para a montagem da carroçaria. Os tokens nestes lugares representam a presença de carros. Entretanto, os carros têm características diferentes.

Seguindo o algoritmo de tradução proposto no capítulo anterior (usando apenas a parte referente ao domínio, isto é, o diagrama de Classe e o Diagrama de Estados Comportamental), a rede de Petri foi gerada pelo itSIMPLE. A figura 4.5 apresenta a rede de Petri gerada e fica claro observar que a falta de suporte a hi-

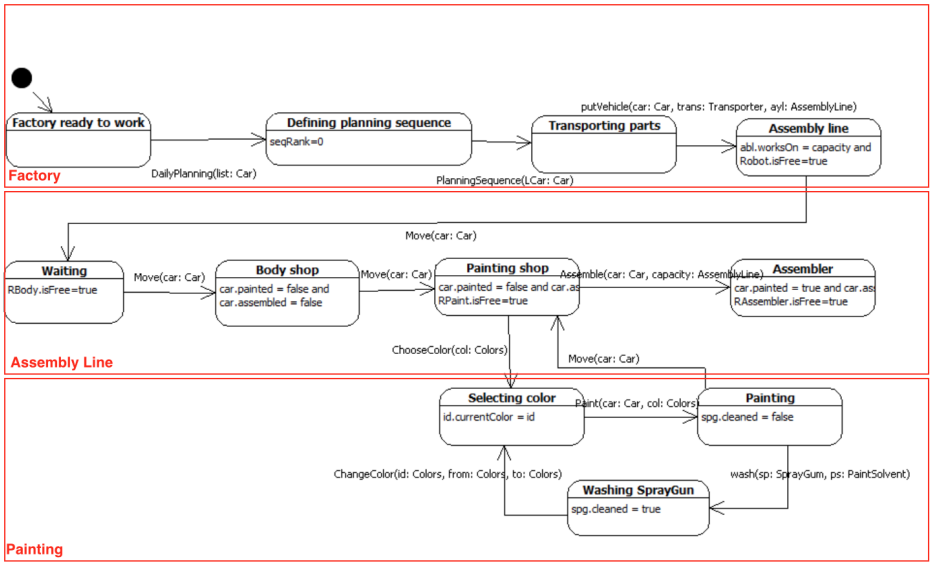

Figura 4.2: Diagrama de Estados Comportamental para o desafio ROADEF 2005.

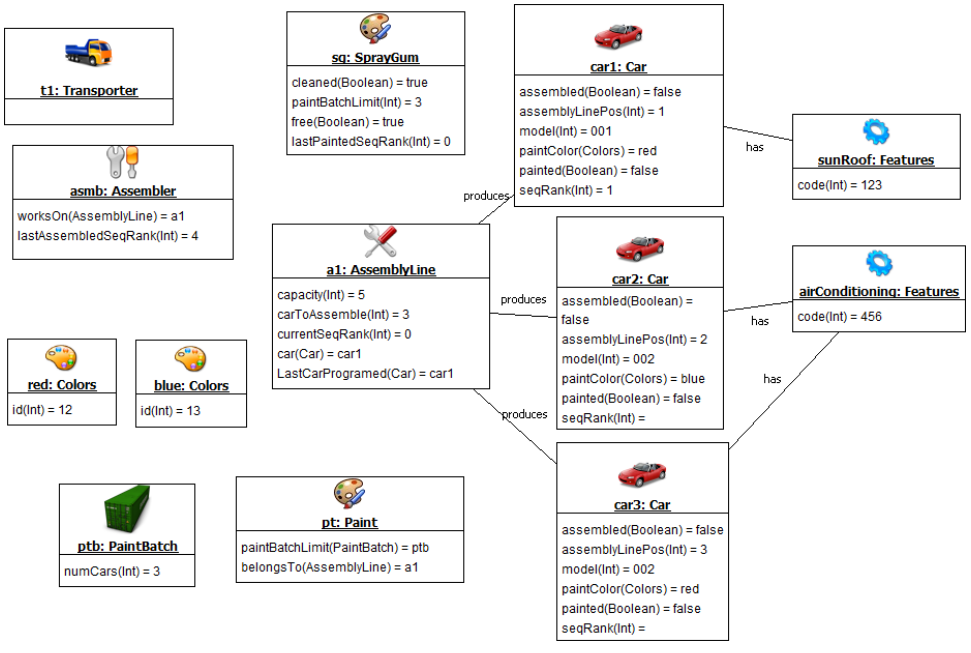

Figura 4.3: Estado inicial representado no Diagrama de Objetos para o desafio ROADEF 2005.

erárquia no modelo UML, interfere no processo de tradução. Observe a diferença entre as redes de Petri da figura 4.5 e 4.6. A última foi gerada fora do itSIMPLE

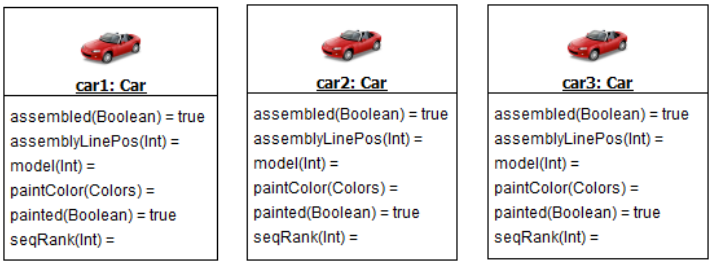

Figura 4.4: Estado meta representado no Diagrama de Objetos para o desafio ROADEF 2005.

considerando o processo de projeto em sua essência. Por causa disso não foi possível executar a análise de requisitos da rede gerada pelo itSIMPLE e isto justifica testar o método fora do framework.

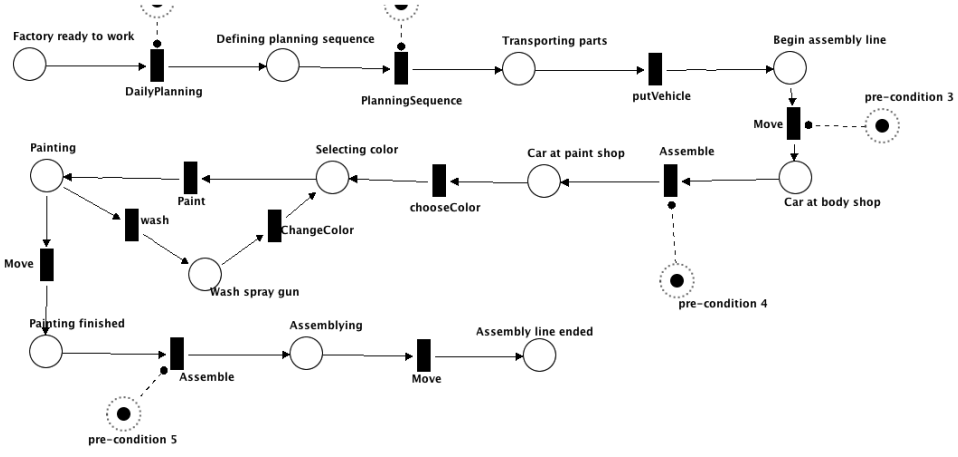

Figura 4.5: Rede de Petri gerada pelo itSIMPLE para o ROADEF 2005.

Se compararmos a rede de Petri gerada pelo itSIMPLE, usando todos os conceitos e atributos propostos pela ferramenta em sua versão 4.0, a diferença entre as redes é ainda maior. Isto ocorre porque no itSIMPLE o usuário deve gerar um diagrama de Estados para cada objeto dinâmico do sistema, e o algoritmo de tradução de UML para redes de Petri (algoritmo de Baresi) gera uma rede para cada diagrama de Estado. Desta forma a representação fica fragmentada impossibilitando a análise de propriedades da rede de Petri. Estas redes de Petri, mesmo se comparadas

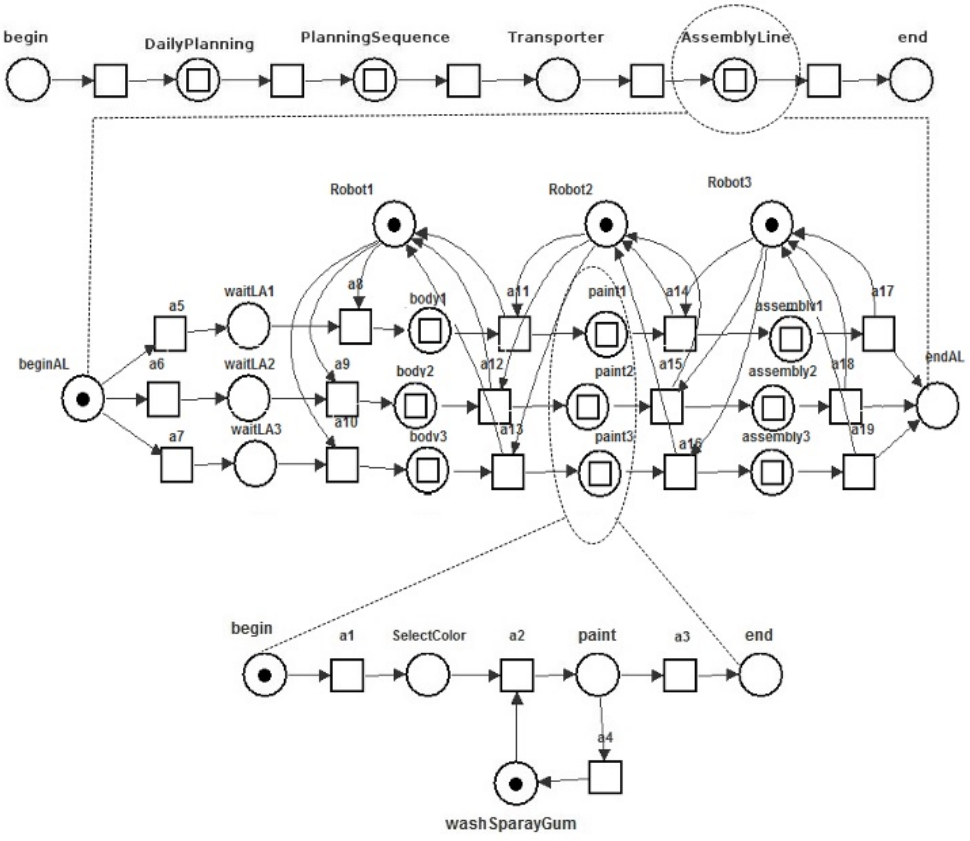

Figura 4.6: Rede de Petri, para o ROADEF 2005, gerada usando o algoritmo de tradução proposto no capítulo anterior.

à rede gerada pela rede da figura 4.5, são extremamente diferentes. Isto acontece porque o processo de projeto apresentada neste trabalho parte do pré-suposto de que o modelo deve representar o sistema como um todo desde a etapa inicial que comtempla o projeto dos diagramas UML. A figura 4.7 ilustra o que acabou de ser explicado.

Se compararmos as redes geradas pelo itSIMPLE 4.0 com a rede de Petri hierárquica - usando o método proposto em sua essência - a diferença é ainda maior. E fica claro observar que o processo de projeto da versão antiga do itSIMPLE possue lacunas no que se refere a elecitação do conhecimento para definição de todos os aspectos do sistema modelado.

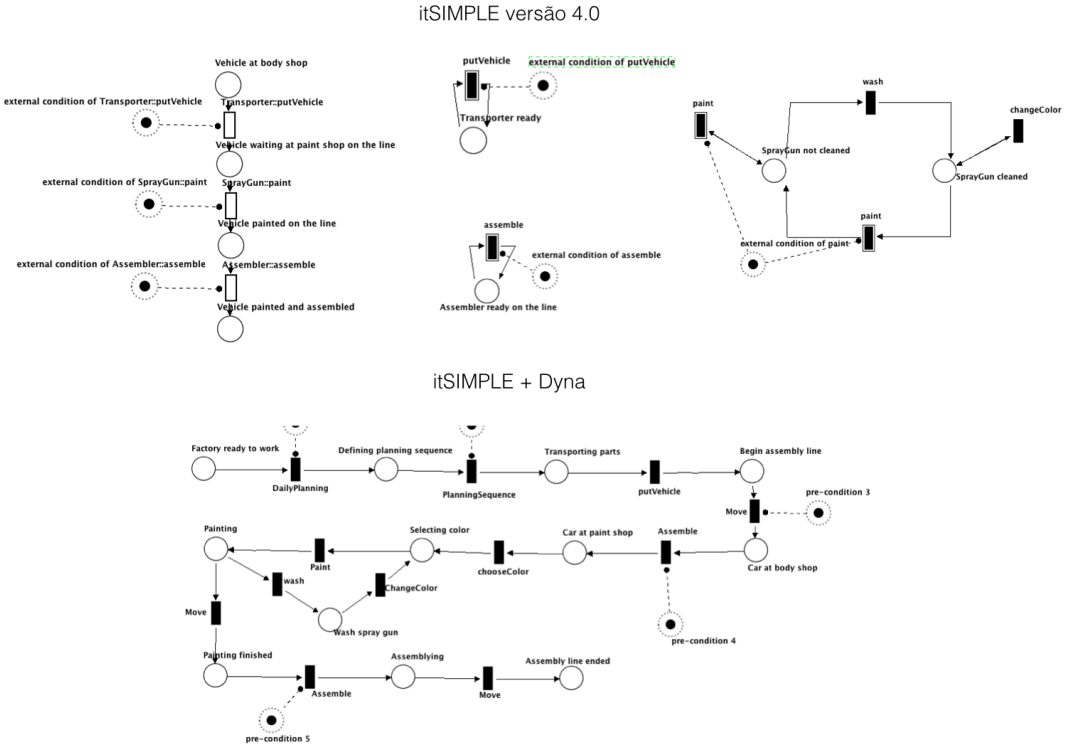

Figura 4.7: Rede de Petri para o ROADEF 2005: versão itSIMPLE 4.0 versus versão itSIMPLE + Dyna.

Diante disso justifica-se a necessidade de testar o processo de projeto fora do itSIMPLE, dado que o mesmo não suporta as alterações fundamentais propostas no presente trabalho. O próximo estudo de caso foi feito usando ferramentas alternativas que permitissem manter a integridade do método proposto.

4.2 Operações de Navios em Plataformas de Petróleo e Portos

Descrição Geral do Problema

Após a descoberta em 2007 de um campo petrolífero entre 2000 a 3000 metros de profundidade, o governo brasileiro vem investindo em tecnologias avançadas e de infraestrutura na extração em águas profundas de petróleo e gás natural. Novas

descobertas no que é chamado pré-sal trouxeram ainda mais desafios na exploração de águas profundas.

De forma geral, o problema pode ser descrito como a necessidade de fornecer o transporte de mercadorias e de ferramentas de dois portos localizados a uns 200 quilômetros um do outro para plataformas no oceano localizadas em faixas de 100 km a 300 km de distância a partir desses portos. Estas faixas foram divididas em duas partes: Rio de Janeiro e Santos, respectivamente. Cada faixa tem um porto (porto P1 no Rio de Janeiro e porto P2 em Santos), onde as atividades de carregamento de artigos de carga ocorrem para apoiar a extração de petróleo em águas profundas. Ambas as faixas contêm um conjunto de plataformas: seis plataformas (F1, ... ,F6) na faixa do Rio de Janeiro, e quatro (G1, ...,G4) na faixa de Santos. Existe uma área de espera em cada faixa: a do Rio de Janeiro (chamada A1) está localizada a 120 km (distância radial) do porto P1 e a de Santos (chamado de A2) está localizado a 100 km do porto P2. A descrição detalhada do problema encontra-se em [31].

Especificações Gerais do Sistema

O problema geral a ser resolvido baseia-se no transporte e entrega de itens de carga solicitados em diferentes locais, considerando uma série de restrições e atributos tais como [31]:

- Portos disponíveis;
- Plataformas;
- Capacidade dos navios;
- Pesos dos itens de carga;
- Consumo de combustível;
- Estações de reabastecimento disponíveis no oceano;
- Duração diferente de operações e Custos.

Dado um conjunto de itens de carga, o problema é encontrar um plano viável que garanta a sua entrega, respeitando as limitações e exigências das capacidades do navio. O objetivo é minimizar a quantidade total de combustível consumido, o tamanho das filas de espera nos portos, o número de navios utilizados, e o custo de ancoragem.

Especificamente, o problema que pretende-se resolver é o planejamento de saída e chegada dos navios nas plataformas do pré-sal, considerando uma série de restrições e elementos com base em um problema real da Petrobrás - a Companhia Brasileira de Petróleo. Assim, a proposta é utilizar o método aqui apresentado para a modelagem deste sistema.

4.2.1 Modelo UML

A seguir serão apresentados os diagramas UML (figuras 4.8 a 4.12) que foram desenhados para representar o problema da Petrobrás. Estes diagramas seguem a proposta do capítulo 3 e foram desenhados no astah Community seguindo a especificação 2.4 da UML.

Para completar o modelo UML as próximas figuras apresentam os diagramas de Objeto para o estado inicial e final, respectivamente. No estado inicial os navios estão todos ancorados na área de espera de Santos e a meta é entregar itens de carga em uma plataforma de Santos e em duas do Rio de Janeiro.

4.2.2 Transformando os Diagramas UML em Redes de Petri

Os diagramas de Estados Comportamentais foram traduzidos em redes de Petri hierárquicas GHENeSys [Foyo 2009]. As figuras 4.14 e 4.15 mostram as redes de Petri que representam a visão geral do problema e a operação dos navios, respectivamente. Ambas foram geradas usando o ambiente GHENeSys [Salmon 2011]. A seguir é apresentado o algoritmo de tradução (de UML para redes de Petri), definido no

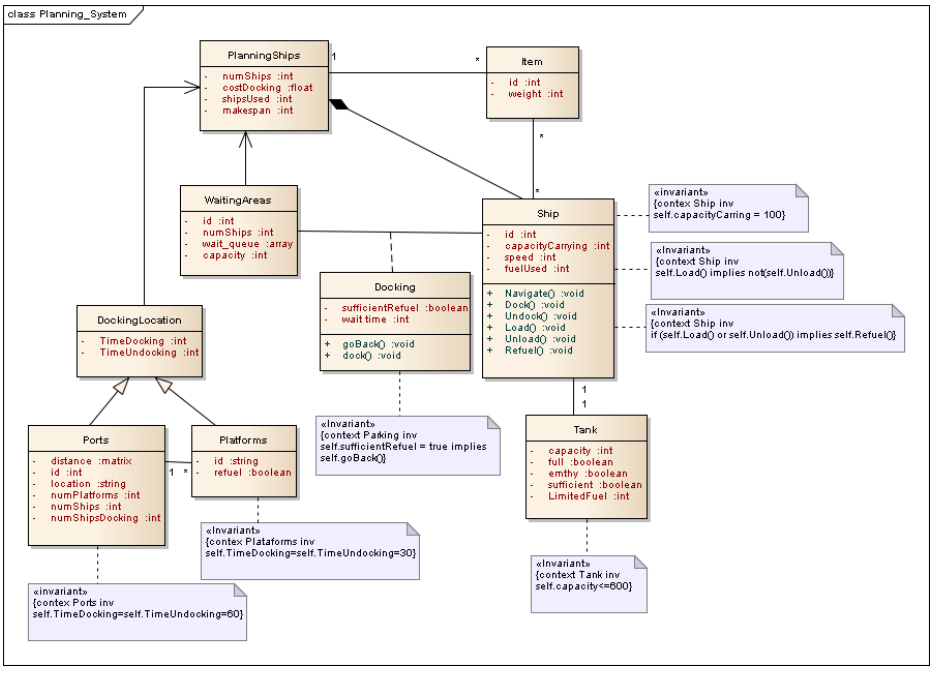

Figura 4.8: Diagrama de Classe para o problema da Petrobrás.

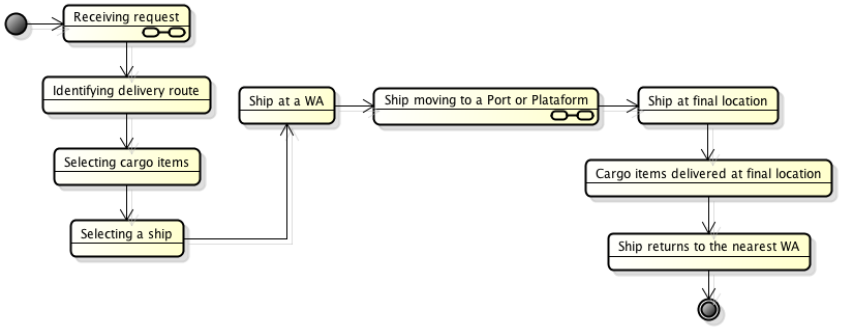

Figura 4.9: Diagrama de Estados Comportamental para o problema da Petrobrás.

capítulo 3. O intuito é guiar o leitor no processo de tradução que o estudo de caso seguiu.

O processo de verificação foi feito usando invariantes de lugar e de transição para validar as propriedades do sistema. O processo utilizado para a validação foi proposto e apresentado em Salmon et. al. [69].

Para verificar a precisão do modelo primeiramente foram computados os in-

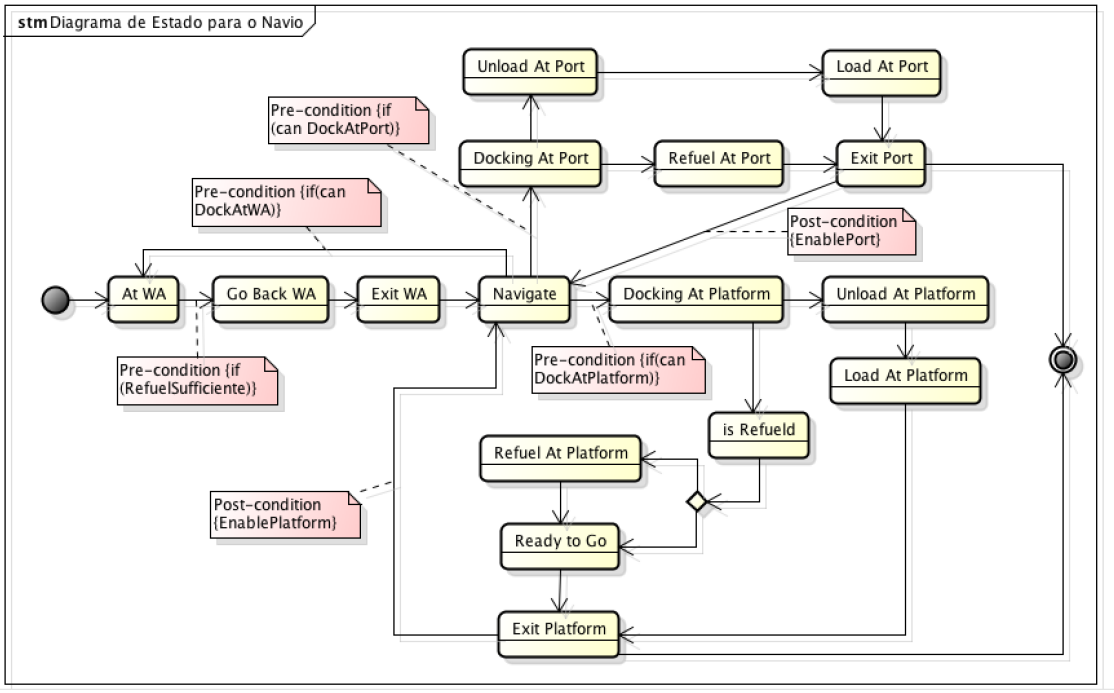

Figura 4.10: Diagrama de Estados Comportamental para operações realizadas pelos navios.

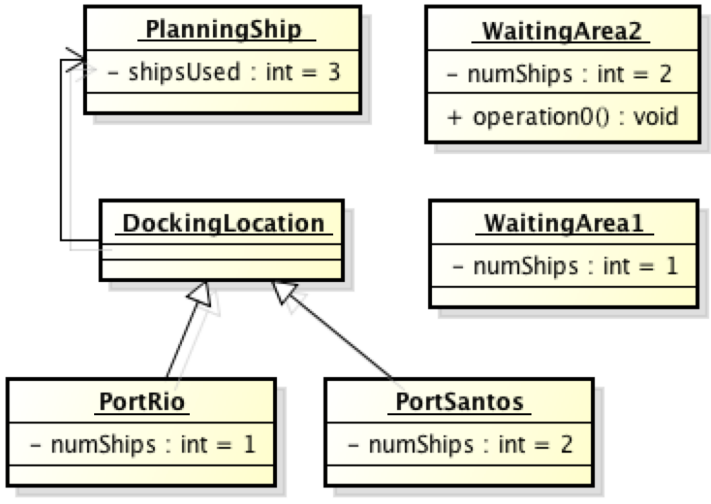

Figura 4.11: Diagrama de Objeto (estado inicial) para os navios.

variantes, então foi verificado se o conjunto de lugares, de cada inequação que gera os invariantes, pertence a algum vetor de solução dos invariantes de lugar da rede.

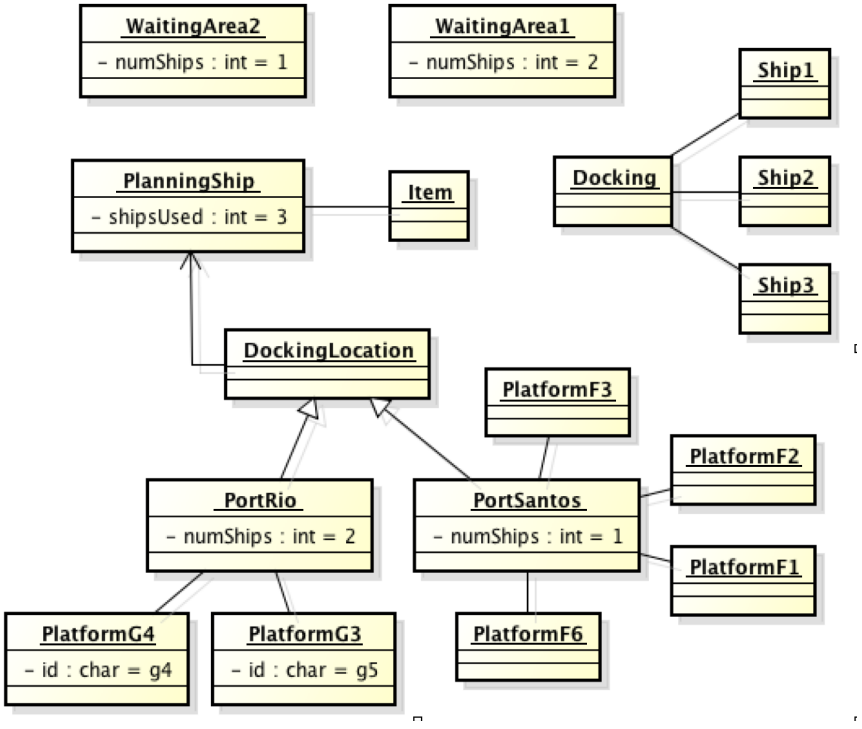

Figura 4.12: Diagrama de Objeto (estado final) para os navios.

O GHENeSys oferece um conjunto de invariantes que representam a solução básica, isto é, o ambiente gera um conjunto com todas as possibilidades de invariantes [Salmon 2011]. Assim, cada invariante da rede é gerado na forma de um conjunto linear de todas as combinações destes vetores.

Usando o conjunto de invariantes gerado pelo GHENeSys, obtém-se os invariantes apresentados nas figuras. Este conjunto de invariantes demonstra que o modelo é adequado para representação a especificação e os requisitos definidos em [Vaquero 2012]. Assim, é possível verificar que o modelo representa com precisão os requisitos desejados.

De acordo com o processo de projeto apresentado no capítulo anterior, o próximo passo é traduzir o modelo UML/redes de Petri para a linguagem HTN - linguagem aceita pelo planejador JSHOP2 para gerar o plano de ação. É importante ressaltar que esta última etapa do processo de projeto foi feita de maneira intuitiva seguindo as semelhanças entre as estruras hierárquicas tanto dos diagramas UML e

```
INÍCIO //considerar apenas diagrama de Estados Comportamental
      VARIÁVEIS
      ESTADOS DO DIAGRAMA;
      AÇÕES DO DIAGRAMA;
REL_ESTADOS_ACOES;
            SELECIONE ESTADOS DO DIAGRAMA
                  MODELE ESTADOS COMO LUGARES // na rede de Petri
            SELECIONE AÇÕES DO DIAGRAMA
                  MODELE AÇÕES DO DIAGRAMA COMO TRANSIÇÕES // na rede de Petri
            SE ESTADOS DIAGRAMA POSSUEM RESTRIÇÕES FORMULADAS EM OCL ENTÃO
                  SE ESTADOS DIAGRAMA POSSUEM PRÉ-CONDIÇÕES ENTÃO
                        MODELE PRÉ-CONDIÇÕES COMO LUGARES
            REPRESENTE ESTADO COM PRÉ-CONDIÇÃO
       FIM-SE;
            SE ESTADOS DIAGRAMA POSSUEM PÓS-CONDIÇÕES ENTÃO
                  MODELE PÓS-CONDIÇÕES COMO LUGARES
        REPRESENTE ESTADO COM PÓS-CONDIÇÃO
      FIM-SE;
    FIM-SE;
REL_ESTADOS_ACOES <- RELAÇÃO(ESTADOS DO DIAGRAMA E AÇÕES DO DIAGRAMA)
MODELE REL_ESTADOS_ACOES COMO ARCOS // para lugares
MODELE REL_ESTADOS_ACOES COMO TRANSIÇÕES  // na rede de Petri
 FIM-SE;
            ENQUANTO ESTADOS COMPOSTOS = VERDADEIRO FAÇA
                  PARA CADA ESTADO COMPOSTO CRIE UM MACRO ELEMENTOS  //na rede de Petri
            REFINE O MACRO ELEMENTO NUMA REDE DE PETRI
            FIM-PARA;
     FIM-ENQUANTO;
FIM;
```

Figura 4.13: Pseudo código para o algoritmo de tradução de UML para rede de Petri.

das redes de Petri, quanto do modelo visual usado para representar a estrutura dos problemas escritos em HTN (ver figura 2.2 do capítulo Revisão da Literatura).

O problema da Petrobrás foi testado com diferentes instâncias. Para ilustrar os resultados obtidos foi escolhida a instância de problema que trata a operação de 3 navios em 6 plataformas diferentes (2 no porto de Santos e 4 no porto do Rio de Janeiro). Para facilitar o entendimento do plano gerado, este foi traduzido (e dividido) em 3 diagramas de Estado (um para representar as ações de cada navio). As próximas figuras (figuras 4.20 a 4.22) mostram o plano de ação para os navios 1, 2 e 3, respectivamente.

Os estados representam as ações que foram definidas no domínio e as caixas de texto representam métodos auxiliares definidos para apoiar o processamento dos métodos principais. Como foi mostrado nas figuras 4.10 e 4.16 (que compreendem

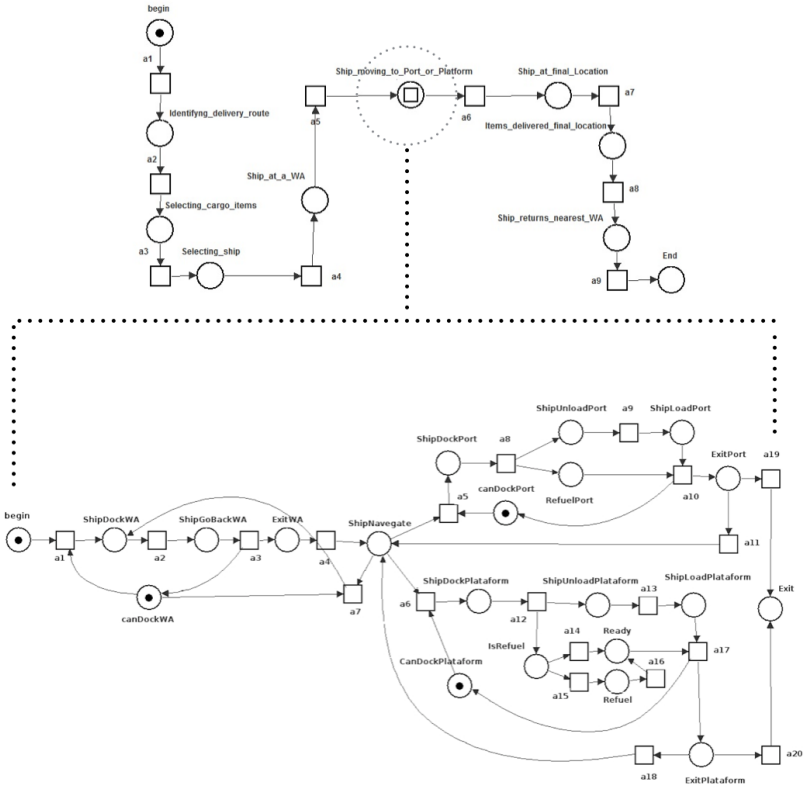

Figura 4.14: Rede hierárquica para o problema da Petrobrás.

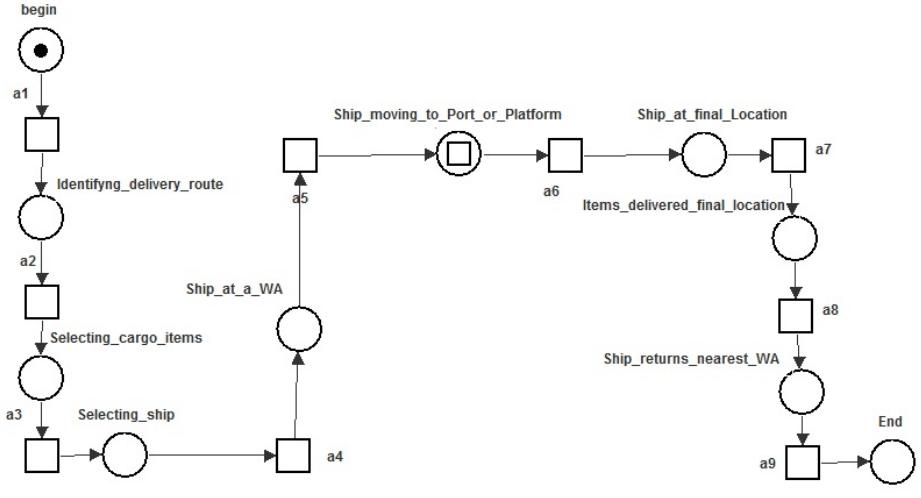

Figura 4.15: Nível mais alto de abstração da rede de Petri hierárquica.

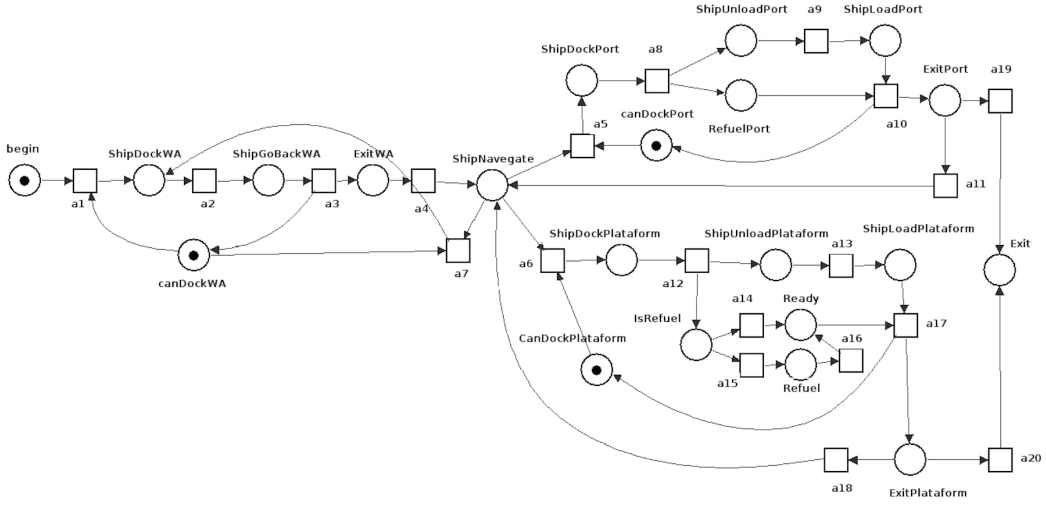

Figura 4.16: Rede secundária para a operação dos navios no problema da Petrobrás.

Incidence Matrix

	a1	a2	a3	a4	a7	a5	a8	a6	a9	a10	a11	a12	a13	a15	a14	a16	a17	a18	a19	a20
begin	-1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
canDockWA	-1.0	0.0	1.0	0.0	-1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
ShipDockWA	1.0	-1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
ShipGoBackWA	0.0	1.0	-1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
ExitWA	0.0	0.0	1.0	-1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
ShipNavegate	0.0	0.0	0.0	1.0	-1.0	-1.0	0.0	-1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
canDockPort	0.0	0.0	0.0	0.0	0.0	-1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
ShipDockPort	0.0	0.0	0.0	0.0	0.0	1.0	-1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
ShipUnloadPort	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	-1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
RefuelPort	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	-1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
ShipLoadPort	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	-1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
ExitPort	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	-1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	-1.0	0.0
CanDockPlataform	0.0	0.0	0.0	0.0	0.0	0.0	0.0	-1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
ShipDockPlataform	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	-1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
ShipUnloadPlataform	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	-1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
IsRefuel	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	-1.0	-1.0	0.0	0.0	0.0	0.0	0.0
Refuel	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	-1.0	0.0	0.0	0.0	0.0
ShipLoadPlataform	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	-1.0	0.0	0.0	0.0
Ready	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	-1.0	0.0	0.0	0.0
ExitPlataform	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	-1.0	0.0	-1.0
Exit	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0

Figura 4.17: Matriz de incidência.

o diagrama de Estados Comportamental e a rede de Petri para o navio, respectivamente), os navios devem estar inicialmente em alguma área de espera. Outro exemplo de requisito é: os navios devem sempre executar a ação de descarga antes de começar a ação de carga. Estes são alguns dos requisitos, que estão descritos na documentação do problema da Petrobrás, que foram verificados e representados

Place Invariants

begin	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	1.0	1.0
canDockWA	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0
ShipDockWA	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0	1.0	1.0	1.0	1.0
ShipGoBackWA	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0	1.0	1.0	1.0	1.0
ExitWA	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	1.0	1.0
ShipNavegate	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	1.0	1.0
canDockPort	0.0	1.0	1.0	1.0	1.0	0.0	0.0	1.0	1.0	1.0	1.0	0.0	0.0	1.0	1.0	1.0	1.0	0.0	0.0	0.0	0.0
ShipDockPort	0.0	1.0	1.0	1.0	1.0	0.0	0.0	1.0	1.0	1.0	1.0	0.0	0.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
ShipUnloadPort	0.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0	1.0	0.0	1.0
RefuelPort	0.0	1.0	1.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	1.0	0.0	1.0	0.0
ShipLoadPort	0.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0	1.0	0.0	1.0
ExitPort	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	1.0	1.0
CanDockPlataform	0.0	0.0	0.0	0.0	0.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.0	0.0	0.0	0.0
ShipDockPlataform	0.0	0.0	0.0	0.0	0.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
ShipUnloadPlataform	0.0	0.0	0.0	0.0	0.0	1.0	1.0	1.0	1.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0
IsRefuel	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	1.0	1.0	1.0	1.0	0.0	0.0	1.0	1.0
Refuel	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	1.0	1.0	1.0	1.0	0.0	0.0	1.0	1.0
ShipLoadPlataform	0.0	0.0	0.0	0.0	0.0	1.0	1.0	1.0	1.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0
Ready	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	1.0	1.0	1.0	1.0	0.0	0.0	1.0	1.0
ExitPlataform	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	1.0	1.0
Exit	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	1.0	1.0

Figura 4.18: Invariantes de lugar.

Transition Invariants

a1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
a2	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0
a3	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0
a4	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0
a7	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0
a5	0.0	1.0	1.0	0.0	0.0	1.0	1.0	0.0	0.0	1.0	1.0
a8	0.0	1.0	1.0	0.0	0.0	1.0	1.0	0.0	0.0	1.0	1.0
a6	0.0	0.0	0.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
a9	0.0	1.0	1.0	0.0	0.0	1.0	1.0	0.0	0.0	1.0	1.0
a10	0.0	1.0	1.0	0.0	0.0	1.0	1.0	0.0	0.0	1.0	1.0
a11	0.0	1.0	1.0	0.0	0.0	1.0	1.0	0.0	0.0	1.0	1.0
a12	0.0	0.0	0.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
a13	0.0	0.0	0.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
a15	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	1.0	1.0
a14	0.0	0.0	0.0	1.0	1.0	1.0	1.0	0.0	0.0	0.0	0.0
a16	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	1.0	1.0
a17	0.0	0.0	0.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
a18	0.0	0.0	0.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
a19	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
a20	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Figura 4.19: Invariantes de transição.

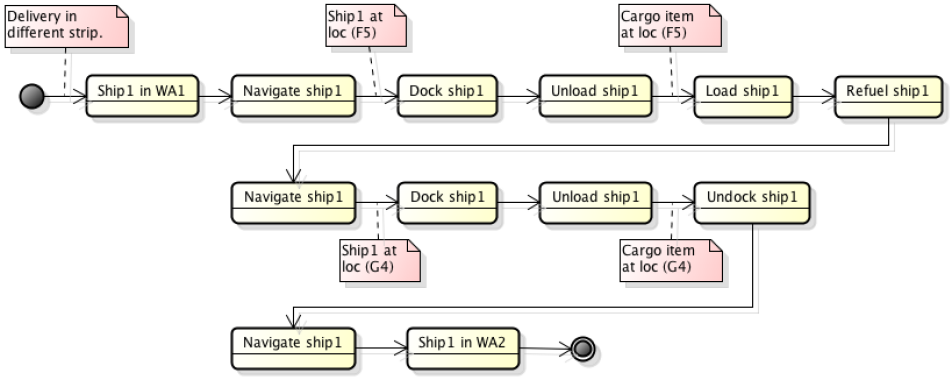

Figura 4.20: Diagrama de estado que representa o plano de ação para o navio 1.

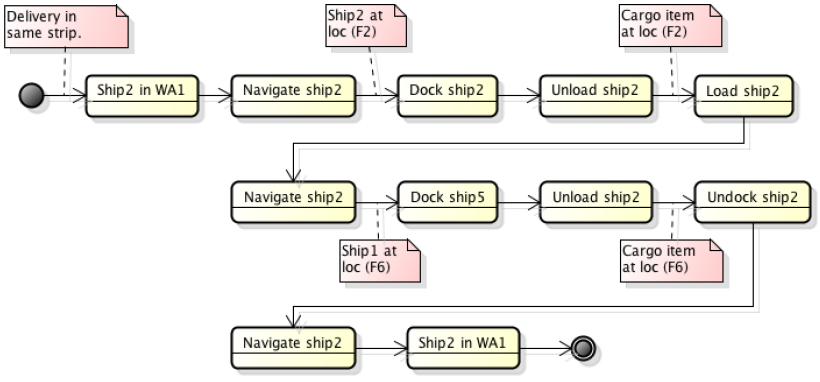

Figura 4.21: Diagrama de estado que representa o plano de ação para o navio 2.

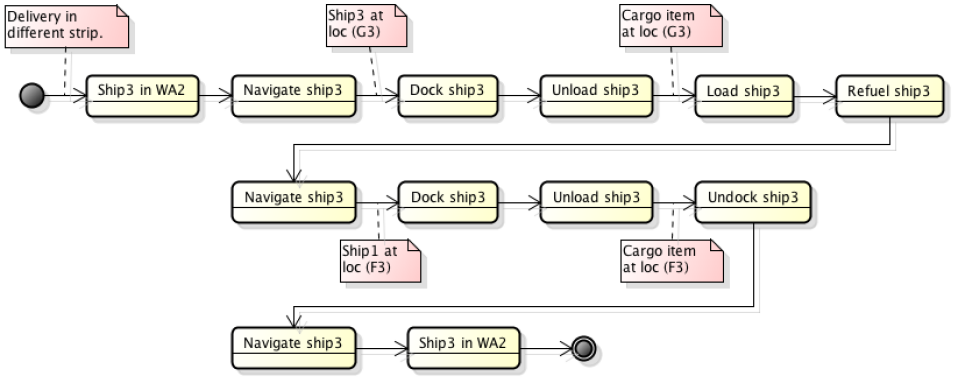

Figura 4.22: Diagrama de estado que representa o plano de ação para o navio 3.

atráves do processo de projeto apresentado neste trabalho.

4.3 Resultados e Discussões

Os estudos de caso apresentados neste trabalho tiveram o intuito de testar o processo de projeto desenvolvido e proposto. A idéia inicial era desenvolver um processo de projeto focado na modelagem, análise e validação de problemas reais. Após revisão de literatura ficou claro que utilizar estruturas hierárquicas para modelar problemas complexos de grande porte era a solução mais apropriada. As versões mais recentes da UML (de 2.2 em diante) oferecem diagramas que são capazes de representar os diversos níveis de abstrações que um problema complexo pode possuir. O uso de redes de Petri de alto nível também havia sido levantado na hipótese inicial, e a evolução do projeto (baseada nos estudos de caso) apontava para as redes de Petri hierárquicas, o que fechava o conjunto de "linguagens"capazes de lidar com abstração e representar hierarquia de forma apropriada para domínios de planejamento. Para fechar o ciclo de projeto era necessário utilizar um planejador (e consequentemente uma linguagem) que fosse capaz de processar todos os níveis de abstração que os modelos apresentassem. Para executar essa tarefa foi escolhido o planejador JSHOP2, que faz parte da família SHOP e foi proposto inicialmente em 1999, por [Nau 1999].

Diante das características principais do método proposto, percebeu-se (além da necessidade de reestruturação do itSIMPLE) que, seria inviável testar o processo de projeto usando o itSIMPLE. Não só por causa da versão antiga da UML e do uso de redes de Petri elementar, mas também pela incapacidade da PDDL de processar estruturas hierárquicas e o itSIMPLE usa apenas planejadores baseados em PDDL. Assim, tornou-se necessário o uso de ferramentas alternativas como o editor UML astah (ou o Enterprise Achitecture), e o planejador JSHOP2.

A literatura mostra que no ano de 2003 muito se falou sobre planejamento hierárquico e seu uso para aplicações de grande porte [Nau 2003], [Dan Wu Evren Sirin e Parsia. 200

[9], [McCluskey e Simpson 2003], [McCluskey 2003]. No ICAPS de 2003 a linguagem HTN foi discutida em diferentes artigos por sua capacidade de lidar e representar problemas complexos onde o uso de abstração se torna necessário para expressar adequadamente todos os níveis de hierarquia que compõem o mesmo. Já nesta conferência foi discutida a limitação da linguagem PDDL para representar problemas com tais características [McCluskey 2003]. Mais recentemente, voltou-se a falar sobre o uso deste tipo de planejamento para a solução de problemas do mundo real [Georgievski e Aiello 2014],[Georgievski e Aiello 2015], [24].

Através dos estudos de caso apresentados foi possível justificar os argumentos levantados. O primeiro estudo de caso foi pensado com o intuito de testar o processo de projeto no itSIMPLE, salvo as devidas adaptações que seriam necessárias. A idéia era tentar identificar os principais pontos de reestruturação no ferramenta. Apesar de já terem sido mencionados vale listá-los de maneira mais organizada.

1 Atualualizar a versão da UML na ferramenta itSIMPLE. E isto implica na atualização do algoritmo que traduz o modelo UML para a linguagem que o planejador compreende. A versão clássica do itSIMPLE trabalha apenas com a PDDL;

2 Implementar em sua essência o algoritmo de tradução de UML para redes de Petri, apresentado no capítulo anterior;

3 Atualizar a versão da rede de Petri, usando agora as redes de Petri Hierárquicas conforme definido em [Miralles 2012].

4 Utilizar o planejamento hierárquico para processar os modelos. No presente trabalho adotou-se a linguagem HTN e o planejador JSHOP2.

O segundo estudo de caso teve o objetivo de testar o processo de projeto em sua totalidade e com isso podemos observar o potencial do método apresentado neste trabalho, bem como do uso de planejamento hierárquico na solução de problemas do

mundo real. Uma das vantagens que o método oferece é o uso da linguagem UML para o desenho do modelo, por ser uma linguagem visual e semi-formal não exigindo, do usuário, o conhecimento específico de linguagens mais complexas (como a PDDL, a HTN).

Ainda seguindo a idéia de restruturação do itSIMPLE, no mesmo laboratório de pesquisa ao qual esse projeto faz parte (Design-Lab) outra proposta projeto que também propõe algo parecido. Silva [Silva 2014] sugere o uso do método GORE (KAOS) para Engenharia de Requisitos em planejamento automático. Esta proposta parte do pressuposto que a utilização de linguagens formais desde a primeira etapa de projeto gera modelos mais aderentes aos requisitos reais. Tal projeto encontra-se em fase de desenvolvimento e tem o potencial de agregar novas características e funcionalidades ao framework itSIMPLE, ampliando sua capacidade de abordar problemas complexos através de propostas diferentes.

Capítulo 5

Conclusões e Trabalhos Futuros

5.1 Contribuições

As contribuições desta tese foram baseadas nas seguintes premissas:

- Foi apresentada uma visão geral das ferramentas de engenharia de conhecimento existentes para planejamento automático em função das fases de um processo de projeto hipotético para aplicações reais. O itSIMPLE foi apresentado à comunidade há 10 anos (2005) na competição ICKEPS, e ficou na época em segundo lugar tendo o GIPO - a referência mundial na época - como vencedor. Em 2009 o itSiMPLE passou a ser a referência e venceu a competição, enquanto o GIPO era descontinuado por decisão dos seus criadores. Estes passaram a usar o itSIMPLE como base da pesquisa. Até hoje o itSIMPLE ainda é bastante citado na literatura da área.

- A pesquisa em Engenharia do Conhecimento para sistemas inteligentes e planejamento automático combina contribuições teóricas, métodos baseados em lógica, com a manipulação de grande quantidade de dados (se o tratamento de sistemas reais for incluido nos objetivos), ao lado de métodos heurísticos eventualmente extraídos de experiência prática ou validados por ela. Assim, sistemas como itSIMPLE desempenham um papel importante neste processo.

- A perspectiva do uso de planejamento automático em problemas reais, tanto no interesse acadêmico como para obter soluçõs mais flexíveis para problemas complexos e proibitivos computacionalmente, leva a inseir métodos para ampliar o desempenho destes sistemas, juntamente com a análise conceitual dos processos (base pra a contribuição deste trabalho).

Com base nestas premissas foi feita uma análise crítica do sistema itSIMPLE, tanto das vantagens como das deficiências e especialmente da dependência deste sistema da base utilizada para a representação de requisitos: a UML. A diferença entre as versões 1.x e 2.x é marcante, onde esta última abre a perspectiva do desenvolvimento baseado em modelos e introduz novos diagramas e nova interpretação para os antigos. Isso aproxima os problemas de planning e scheduling, unifica a notação e coloca alguns desafios para se obter um processo de projeto realmente eficiente.

Baseado nisso as seguintes contribuições foram delineadas como objetivo do trabalho:

- uma evolução da suposição de que as aplicações de planejamento têm duas classes diferentes de requisitos, baseada em [91]: o work domain, onde todas as características essenciais do entorno são consideradas (tais como: nome, restrições, operações ações gerais e descrições do ambiente que são críticas para o sistema); e os aspectos do problema de planejamento, onde são definidos o estado inicial, o estado final e um conjunto de objetos que compreendem a instância de um problema, como foi mostrado no capítulo 3, e o conjunto das ações admissíveis.

- A definição de um conjunto mínimo de diagramas da UML, reduzindo o escopo apresentado na versão clássica do itSIMPLE. Tal conjunto foi testado empiricamente em diversos estudos de caso e apresentou resultados positivos em todos eles, especialmente nos dois estudos de caso apresentados no capítulo anterior.

- A definição de uma disciplina de projeto para guiar o usuário no levantamento de requisitos e modelagem dos diagramas UML. Esta disciplina encontra-se detalha e exemplifica no decorrer do presente trabalho.

- Foi desenvolvido um algoritmo de tradução dos diagramas UML para redes de Petri Hierárquicas, que é uma evolução do algoritmo de Baresi [Baresi e Pezze 2001], considerando as restrições em OCL, a instância de problema definida no diagrama de Objetos garantindo a multiplicidade da rede de Petri, e a representação de simetria nos processos. Este algoritmo foi implementado na ferramenta itSIMPLE, não obtivemos bons resultados porque a versão da UML no itSIMPLE não suporta hierarquia (como foi mostrado do estudo de caso ROADEF 2005 do capítulo anterior).

- A tradução do modelo para HTN, de modo que este possa ser testado no planejador JSHOP2. Esse processo foi feito manualmente nos estudos de caso, mas deverá ser implementado na próxima versão do itSIMPLE.

- A tradução do plano de ação em diagramas de Estados, facilitanto o entendimento do plano. Uma opção igualmente viável é traduzir o planos para diagramas de Atividades.

- O desenvolvimento de dois estudos de caso que tiveram o objetivo de: exemplificar as limitações do itSIMPLE, bem como a incapacidade de testar o método proposto na ferramenta; e a aplicação do processo de projeto em sua essência sugerindo um novo ciclo de vida no projeto de problemas reais em planejamento automático.

- Levantou-se uma série de limitações no framework itSIMPLE, sugerindo como solucionar cada uma delas. Tais melhorias visam o aperfeiçoamento do processo de desenvolvimento adotado atualmente, bem como a validação do modelo a partir de um processo bem definido e estruturado. Isto não só organiza o processo de projeto como otimiza a resposta do planejador, antecipando para

a etapa de modelagem/análise a solução de possíveis problemas e erros no modelo.

Como já foi enfatizado, estas contribuições, pequenas se comparadas em separado, levam a uma revisão do papel da Engenharia do Conhecimento em processos de planejamento, enquanto abre novas perspectivas de métodos que estão sendo criados e testados tanto na USP como na Univ. de Huddersfield - UK, e em outros centros. Por exemplo, baseado neste trabalho, está sendo criada uma nova versão do itSiMPLE, o ItSIMPLE-SE que insere o tempo nos diagramas e busca a unificação entre os processos dedicados a planning e aqueles dedicados a scheduling. Este é apenas um dos possíveis trabalhos futuros que se abrem.

5.2 Trabalhos Futuros

Ao longo deste trabalho comentou-se brevemente sobre os aspectos da pesquisa que permanecem como trabalhos futuros. Nesta seção, serão reforçadas tais perspectivas que apontam para novas pesquisas e que surgiram a partir do trabalho desenvolvido neste projeto de tese.

5.2.1 Modelos Baseados em Tempo

No presente trabalho não foram considerados problemas de planejamento que envolvem tempo (ou tempo e recursos), uma característica comum em aplicações reais. Portanto um trabalho futuro é incluir o aspecto temporal no projeto, especificamente na modelagem e na análise do modelo. O tempo é também um parâmetro fundamental de qualidade a ser avaliado nos dois níveis já definidos: do problema de planejamento automatizado e no *work domain. Como uma evolução natural das linguagens de design (UML, redes de Petri e HTN), um modelo baseado em tempo proporcionaria um desafio interessante para a eliciação do conhecimento, modelagem e análise do modelo. A UML oferece atualmente suporte para os modelos baseados*

em tempo através do diagrama de Tempo, bem como as redes de Petri temporais. Entretanto, a linguagem HTN apresenta uma limitação no tratamento de tempo contínuo e deve ser melhor investigada para entendermos tais limitações (por exemplo, como as funções de tratamento de tempo se comportariam em problemas que possuem tempo contínuo). As redes de Petri por sua vez podem tratar tanto o tempo como intervalo discreto, time slice, (que seria o modelo Ramchandani) como o tempo contínuo, (que seria o modelo de Merlin). Assim, a mesma disciplina de projeto poderia ser aplicada, dependendo apenas de uma linguagem de transferência para os planejadores, e de uma nova leva de planejadores que consideram tempo contínuo.

5.2.2 Interpretação dos Planos Oferecidos pela Família SHOP2

Por causa das limitações, já mencionadas anteriormente, no framework itSIMPLE foi necessário aplicar o processo de projeto apresentado fora da sua estrutura básica, com isto perdeu-se algumas das funcionalidade. Por exemplo, foi utilizado o planejador JSHOP2, que não pertence ao conjunto de planejadores acoplados ao itSIMPLE, e assim perdeu-se a possibilidade de avaliar a qualidade do plano, um dos grandes destaques do processo de projeto original do itSIMPLE. O principal motivo é que o itSIMPLE em sua essência é um framework desenvolvido a partir dos conceitos de Engenharia de Conhecimento e as facilidades que ele oferece não são integralmente aproveitadas pelo planejador JSHOP2. No JSHOP2 os planos são oferecidos num arquivo log de difícil entendimento. Para o presente trabalho o plano foi dividido por navio e apresentado em diagramas de Estados. A proposta é criar um formalismo que seja capaz de interpretar esse arquivo de log e apresentá-lo usando alguma linguagem visual que facilite o entendimento do usuário final.

5.2.3 Mapear Semelhança entre as Estruturas Topológicas

A idéia do processo de projeto é preservar a estrutura hierárquica do modelo do começo ao fim. Ao propor e aplicar as ferramentas definidas percebeu-se semelhanças

entre suas estruturas topológicas. As semelhanças (e equivalências) encontradas entre UML e redes de Petri (de modo geral e mantendo a estrutura hierárquica) já foram mapeadas e formalizadas no algoritmo de tradução. A proposta em aberto seria identificar as correlações entre UML/redes de Petri e HTN para propor um formalismo de tradução do modelo visual para a linguagem HTN. Nos estudos de caso realizados este processo de tradução foi feito empiricamente e apresentou resultados satisfatórios comprovados no plano de ação gerado pelo JSHOP2.

5.2.4 Implementação do Método Proposto

Implementar o método proposto em sua totalidade na nova versão do framework itSIMPLE, que será chamado de itSIMPLE-SE. Está nova versão, seguindo o sugerido neste projeto de tese, deverá comtemplar a versão 2.5 da UML, (com o conjunto mínimo de diagramas definido no capítulo 3), o algoritmo de tradução para redes de Petri hierárquica e temporal, a comunicção com o framework GHENeSys (onde é feita a análise de propriedades da rede baseado no padrão ISO/IEC 15.909), o algoritmo de tradução do modelo visual (UML/rede de Petri) para a linguagem HTN, e talvez a inclusão de uma nova versão do planejador JSHOP2 com a tradução dos planos de ação oferecidos pelo mesmo. Este novo sistema será implementado em Python e deverá reutilizar parte do código gerado neste trabalho (que está em Java).

5.3 Considerações Finais

Esta de tese descreveu um processo de projeto, que tem o objetivo de oferecer uma disciplina capaz de guiar o usuário durante o ciclo de vida de projeto de problemas do mundo real para os quais se aplicam métodos de planejamento automático. Foi mostrado que problemas reais possuem características abstratas que são melhor representadas através do uso de hierarquia. Para tanto, definiu-se um ciclo de vida

que garante a integridade da estrutura hierárquica do modelo durante todo o ciclo de vida do projeto. Tal formato proporciona o desenvolvimento de um modelo mais aderente ao problema real e melhora a qualidade do plano de ações, minimizando a dependência de esforços de pós-projeto. Todo o processo de projeto oferece um método dependente de domínio, onde o conhecimento específico pode ser incluído nos diagramas e refletido até o fim mantendo a flexibilidade do processo, mas pode também ser ampliado para incluir problemas independentes de domínio.

Referências Bibliográficas

[1]AU, T.; KUTER, U.; NAU, D. Web service composition with volatile information. In: IN YOLANDA GIL ENRICO MOTTA, V. R. B.; MARK A. MUSEN, e. (Ed.). International Semantic Web Conference. *[S.l.]: Lecture Notes in Computer Science, 2005. v. 3729.*

[Bacchus e Kabanza 2000]BACCHUS, F.; KABANZA, F. Using temporal logics to express search control knowledge for planning. *2000.*

[Baresi e Pezze 2001]BARESI, L.; PEZZE, M. Improving uml with petri nets, electronic notes in theoretical computer science 44. 2001.

[Baresi e Pezze 2001]BARESI, L.; PEZZE, M. On formalizing uml with high-level petri nets. Concurrent object-oriented programming and petri nets: advances in petri nets, *Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2001.*

[Barreiro 2012]BARREIRO, J. e. a. Europa: A platform for ai planning, scheduling, constraint programming, and optimization. In Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS-12) - The 4th International Competition on Knowledge Engineering for Planning and Scheduling., *2012.*

[6]BELKER, T.; HAMMEL, M.; HERTZBERG, J. Learning to optimize mobile robot navigation based on htn plans. In ICRA, IEEE., *2003.*

[Bonet 2007]BONET, B. et al. Planning via petri net unfolding: Generalisation and improvements. ICAPS, *2007.*

[8]BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. UML Guia do Usuário. *[S.l.]: Editora Campus, 2000.*

[9]BOTEA, A.; MULLER, M.; SCHAEFFER, J. Extending pddl for hierarchical planning and topological abstraction. ICAPS 2003*, 2003.*

[Bouillet 2009]BOUILLET, E. et al. Mario: middleware for assembly and deployment of multi platform flow based applications. In Proceedings of the 10th ACM/IFIP/USENIX International Conference on Middleware, Middleware 09.*, 2009.*

[Dan Wu Evren Sirin e Parsia. 2003]DAN WU EVREN SIRIN, J. A. H. D. S. N.; PARSIA., B. Automatic web services composition using shop2. In: WWW. [S.l.], 2003.

[Denaro e Pezze 2004]DENARO, G.; PEZZE, M. Petri nets and software engineering. Spring Verlag Berlin Heidelberg*, 2004.*

[Doherty 1998]DOHERTY, P. et al. TAL: Temporal Action Logics language specification and tutorial. *1998.*

[Doherty e Kvarnstrom 1999]DOHERTY, P.; KVARNSTROM, J. TAL planner: An empirical investigation of a temporal logic based forward chaining planner. *1999.*

[15]EDELKAMP, S.; FRANK, J.; KELLERSHOFF, M. Knowledge engineering through simulation. Proceedings of the International Knowledge Engineering Competition.*, 2007.*

[Edelkamp e Jabbar 2006]EDELKAMP, S.; JABBAR, S. Action planning for directed model checking of petri nets. Electronic Notes in Theoretical Computer Science.*, v. 149, n. 2, 2006.*

[17]FOYO, P. M.; ; SILVA, J. R. Some issues in real time systems verification using time petri nets. Journal of the Brazilian Society of Mechanical Sciences and Engineering.*, v. 33, 2011.*

[Foyo 2009]FOYO, P. M. G. Verificação formal de sistemas discretos distribuídos. Tese (Doutorado) - Escola Politécnica da Universidade de São Paulo.*, 2009.*

[19]FOYO, P. M. G.; SALMON, A. Z. O.; SILVA, J. R. Requirements analysis of automated projects using uml and petri nets. Proceedings of COBEM 2011*, 2011.*

[Foyo 2010]FOYO, P. M. G. del. Verificação Formal de Sistemas Discretos Distribuídos. *Tese (Doutorado) — Escola Politécnica da Universidade de São Paulo, 2010.*

[Georgievski e Aiello 2014]GEORGIEVSKI, I.; AIELLO, M. An overview of hierarchical task network planning. ArXiv preprint. Mar 2014.

[Georgievski e Aiello 2015]GEORGIEVSKI, I.; AIELLO, M. Htn planning: Overview, comparison, and beyond. Artificial Intelligence., *v. 222, p. 124–156, 2015.*

[23]GERHARD, W.; CHRPA, L.; MCCLUSKEY, T. Creating planning domain models in kewi. Proceedings of the 5th Workshop on Knowledge Engineering for Planning and Scheduling., *2014.*

[24]GERHARD, W.; CHRPA, L.; MCCLUSKEY, T. Kewi: A knowledge engineering tool for modelling ai planning tasks. In: 6th International Conference on Knowledge Engineering and Ontology Development., *2014.*

[25]GHALLAB, M.; NAU, D.; TRAVERSO, P. Automated Planning: Theory and Practice. *San Francisco, CA.: Morgan Kaufman., 2004.*

[Gonzalez 2001]GONZALEZ, P. M. GHENeSys : Uma rede Estendida Orientada a Objetos para Projeto de Sistemas Discretos. *Tese (Doutorado) — Universidade de São Paulo, 2001.*

[Gonzalez Ferrer 2009]GONZALEZ FERRER, A. et al. Jabbah: A java application framework for the translation between business process models and htn. In Working notes of the 19th International Conference on Automated Planning and Scheduling (ICAPS-09). Proceedings of the 3rd International Competition on Knowledge Engineering for Planning and Scheduling (ICKEPS)., *2009.*

[Guedes 2009]GUEDES, G. UML 2: Uma abordagem prática. *[S.l.]: Novatec, 2009.*

[Hickmott 2007]HICKMOTT, S. et al. Planning via petri net unfolding. IJACAI, *2007.*

[Hoffmann 2003]HOFFMANN, J. Utilizing problem structure in planning a local search approach. Lecture Notes in Computer Science, *v. 285, 2003.*

[31]IGREJA, H.; SILVA, J. R.; TONIDANDEL, F. Planning ship operations on petroleum platforms and ports. ICKEPS Challenge Domain: ICAPS, the 22nd International Conference on Automated Planning and Scheduling. Atibaia, São Paulo, Brazil., *2012.*

[Jain 2003]JAIN, A. et al. Roams: Planetary surface rover simulation environment. In: Proceedings of the International Symposium on Artificial Intelligence Robotics and Automation in Space, Nara, Japan. *[S.l.: s.n.], 2003.*

[33]KAMBHAMPATI, S.; DATTATRAYA, A.; SRIVASTAVA, B. Hybrid planning for partially hierarchical domains. *1998.*

[34]KUTLUHAN, E.; HENDLER, J.; NAU, D. UMCP: A sound and complete procedure for hierarchical task network planning. *1994.*

[35]KUTLUHAN, E.; HENDLER, J.; NAU, J. Semantics for HTN planning. *[S.l.], 1994.*

[Lam e Shankar 1988]LAM, S.; SHANKAR, A. A relational notation for state transition systems. Elsevier Science., *1988.*

[LaValle 2006]LAVALLE, S. M. Planning Algorithms. *[S.l.]: Cambridge University Press, 2006.*

[Lin e Wonham 1990]LIN, F.; WONHAM, W. Decentralized control and coordination of discrete event systems with partial observation. Automatic Control, IEEE Transactions., *v. 35, 1990.*

[39]LITTLE, J.; SELENSKY, E.; BECK, J. C. Process design for efficient scheduling. In: Proceedings of the 12th IFAC Symposium on Information Control Problems in Manufacturing. *[S.l.: s.n.], 2006. v. 12.*

[Liu e Yen 1996]LIU, X.; YEN, J. An analytic framework for specifying and analyzing imprecise requirements. International Conference of Software Engineering., *1996.*

[McCluskey 2002]MCCLUSKEY, T. Knowledge engineering: issues for the ai planning community. The AIPS-2002 Workshop on Knowledge Engineering Tools and Techniques for AI Planning., *p. unpublished, 2002.*

[McCluskey 2003]MCCLUSKEY, T. L. Pddl: A language with a purpose? ICAPS 2003 Proceedings., *2003.*

[McCluskey 2003]MCCLUSKEY, T. L. et al. Knowledge engineering for planning roadmap. 2003.

[Mccluskey e Kitchin 1998]MCCLUSKEY, T. L.; KITCHIN, D. A tool supported approach to engineering htn planning models. In Proceedings of 10th IEEE International Conference on Tools with Artificial Intelligence., *1998.*

[McCluskey e Simpson 2003]MCCLUSKEY, T. L.; SIMPSON, R. M. Gipo ii: Htn planning in a tool-supported knowledge engineering environment. ICAPS 2003 Proceedings., *2003.*

[McDermott 2000]MCDERMOTT, D. The 1998 ai planning competition. AI Magazine., *2000.*

[Miralles 2012]MIRALLES, J. S. P. GHENeSys , uma Rede Unificada e de Alto Nível. *Tese (Doutorado) — São Paulo, 2012.*

[Miyagi 1988]MIYAGI, P. E. Control System Design, Analysis and Implementation of Discrete Event Production Systems by using Mark Flow Graph. *Tese (Doutorado) — Tokyo Institute of Technology, Tokyo, 1988.*

[Murata 1989]MURATA, T. Petri nets: Properties, analysis and applications. In: Proceedings of the IEEE. *[S.l.: s.n.], 1989. v. 77.*

[Nau 1999]NAU, D. et al. SHOP: Simple Hierarchical Ordered Planner. *1999. 986-975 p.*

[Nau 2003]NAU, D. e. a. SHOP2: An HTN planning system. *2003.*

[Nau 2005]NAU, D. e. a. Applications of SHOP and SHOP2. *2005.*

[Nguyen 2005]NGUYEN, A. Challenge ROADEF 2005 - Car Sequencing Problem. 2005.

[Okhtay e Nau 2004]OKHTAY, O.; NAU, D. A general approach to synthesize problem specific planners. 2004.

[OMG 2001]OMG. OMG Unified Modeling Language Specification, Version 1.4. *[S.l.], 2001.*

[OMG 2003]OMG. OMG Unified Modeling Language Specification - Object Constraint Language, Version 2.0. *[S.l.], 2003.*

[OMG 2011]OMG. OMG Unified Modeling Language (TM) (OMG UML), Superstructure. Version 2.4.1. *[S.l.], 2011.*

[Palpant 2009]PALPANT, M. et al. ROADEF 2009 Challenge : Disruption Management for Commercial Aviation. 2009.

[Perez e Carbonell 1994]PEREZ, M.; CARBONELL, J. Control knowledge to improve plan quality. In: Proceedings of the Second International Conference on Artificial Intelligence Planning Systems. *[S.l.: s.n.], 1994.*

[Perez 2006]PEREZ, O. et al. Planning process from a user perspective. In: Proceedings of the 16th International Conference on Automated Planning and Scheduling (ICAPS 2006) Workshop on Plan Analysis and Management. Cumbria, UK. *[S.l.: s.n.], 2006.*

[Plch 2012]PLCH, T. et al. Inspect, edit and debug pddl documents: Simply and efficiently with pddl studio. ICAPS12 System Demonstration 4., *2012.*

[Pnueli 1977]PNUELI, A. The temporal logic of programs. *1977.*

[Ramchandani 1974]RAMCHANDANI, C. Analisys of Asynchronous Concurrent Systems by Petri Nets. *[S.l.], 1974.*

[Ramos e Silva 1998]RAMOS, R.; SILVA, J. A formal model for integrated complex dynamic systems. In: 5th IFAC Workshop on Intelligent Manufacturing Systems - IMS 98., *1998.*

[Reddy 2008]REDDY, S. et al. Planning and Monitoring Solar Array Operations on the ISS. In: Proceedings of the 18th International Conference on Automated Planning and Scheduling (ICAPS 2008) Scheduling and Planning Application Workshop. Sydney, Australia. *[S.l.: s.n.], 2008.*

[Sacerdoti 1975]SACERDOTI, E. The nonlinear nature of plans. *1975.*

[Salmon 2011]SALMON, A. O. et al. Towards a unified view of modelingand design with ghenesys. Proceedings of COBEM2011., *2011.*

[Salmon 2014]SALMON, A. Z. O. Uso de redes de petri para modelar e analisar sistemas multiagentes. Departamento de Engenharia Mecatrônica e de Sistemas Mecânicos, São Paulo, Brasil. May 2014.

[69]SALMON, A. Z. O.; DEL FOYO, P. M. G.; R., S. J. Verification of automated systems using invariants. Anais do XX Congresso Brasileiro de Anais do XX Congresso Brasileiro de Anais do XX Congresso Brasileiro de Automação., *2014.*

[Santos e Silva 2005]SANTOS, E.; SILVA, J. R. Applying petri nets to requirements validation. ABCM Symposium Series in Mechatronics., *2005.*

[Shah 2013]SHAH, M. et al. Knowledge engineering tools in planning: State of the art and future challenges. In Proceedings of 4th Workshop on Knowledge Engineering for Planning and Scheduling - KEPS. ICAPS, Rome, Italy., *2013.*

[Silva 2014]SILVA, J. M. Metodologia GORE (KAOS) para Engenharia de Requisitos. *[S.l.], 2014.*

[Silva 1998]SILVA, J. R. Interactive design of integrated systems. In: MATOS, L. C.; AFSARMANESH, H. (Ed.). Intelligent Systems for Manufacturing. *[S.l.]: Kluwer Academic Pub., 1998.*

[Silva 2004]SILVA, J. R. Applying petri nets to requirements validation. IFAC Symposium on Information Control Problems in Manufacturing., *v. 1, 2004.*

[Silva e Miyagi 1995]SILVA, J. R.; MIYAGI, P. E. Pfs/mfg: A high level net for the modeling of discrete manufacturing systems. In: MATOS, L. C.; AFSARMANESH, H. (Ed.). Balanced Automation Systems, Arquitectures and Design Methods. *[S.l.]: Chapman and Hall, 1995.*

[Silva e Miyagi 1996]SILVA, J. R.; MIYAGI, P. E. A formal approach to pfs/mfg: a petri net representation of discrete manufacturing systems. In: Studies in Informatics and Control. *Romenia: IC Publications, 1996.*

[77]SIMPSON, R.; KITCHIN, D.; MCCLUSKEY, T. L. Planning domain definition using gipo. Knowledge Engineering Review., *2007.*

[Simpson 2006]SIMPSON, R. M. Gipo graphical interface for planning with objects. Engineering*, 2006.*

[Sirin 2004]SIRIN, E. et al. HTN planning for web service composition using SHOP2. *2004.*

[80]SMITH, D.; FRANK, J.; CUSHING, W. The anml language. In: Proceedings of the 18th International Conference on Automated Planning and Scheduling (ICAPS 2008). Workshop on Knowledge Engineering for Planning and Scheduling. Sydney, Australia. *[S.l.: s.n.], 2008.*

[Sohrabi e McIlraith 2008]SOHRABI, S.; MCILRAITH, S. On planning with preferences in htn. Proc. of the 12th Int l Workshop on Non-Monotonic Reasoning (NMR)l Workshop on Non-Monotonic Reasoning (NMR)*, 2008.*

[Sommerville, I and Sawyer, P 2007]Sommerville, I and Sawyer, P. Requirements Engineering: A Good Practice Guide. *[S.l.]: Wiley, 2007.*

[83]SON, C.; BARAL, T.; MCILRAITH, S. Domain dependent knowledge in answer set planning. 2002.

[84]STUDER, R.; BENJAMINS, V.; FENSEL, D. Knowledge engineering: Principles and methods. Data and Knowledge Engineering*, v. 25, 1998.*

[Tate 1977]TATE, A. Generating project networks. *1977.*

[Tate e Currie 1991]TATE, A.; CURRIE, W. O plan: The open planning architecture. *1991.*

[Tonaco-Basbaum 2013]TONACO-BASBAUM, R. Requirements and work domain analysis in automated planning systems. Doctoral Consortium. The 23rd International Conference on Automated Planning and Scheduling. Rome, Italy.*, 2013.*

[88]TONACO-BASBAUM, R.; VAQUERO, T.; SILVA, J. R. Requirement analysis method for real world systems in automated planning. Knowledge Engineering for Planning and Scheduling (KEPS). The 23rd International Conference on Automated Planning and Scheduling. Rome, Italy*, 2013.*

[Ullrich 2005]ULLRICH, C. Course generation based on htn planning. In: _____. [S.l.]: In Mathias Bauer, Boris Brandherm, Johannes Furnkranz, Gunter Grieser, Andreas Hotho, Andreas Jedlitschka, and Alexander Kroner, editors, LWA., 2005.

[Upal 2005]UPAL, M. Learning to Improve Plan Quality. Computational Intelligence*, v. 21, n. 4, 2005.*

[91]VAQUERO, T.; BECK J C MCCLUSKEY, T. L.; SILVA, J. R. Knowledge engineering for planning and scheduling: Tools and methods. JAIR*, 2013.*

[Vaquero 2012]VAQUERO, T. et al. Planning and Scheduling Ship Operations on Petroleum Ports and Platforms. Proceedings of the Scheduling and Planning Applications Workshop*, 2012.*

[Vaquero 2007]VAQUERO, T. et al. itSIMPLE 2.0 : An Integrated Tool for Designing Planning Domains. 2007.

[94]VAQUERO, T.; TONIDANDEL, F.; SILVA, J. R. The itSIMPLE tool for Modelling Planning Domains. In: Proceedings of the First International Competition on Knowledge Engineering for AI Planning, Monterey, Califormia, USA. *[S.l.: s.n.], 2005.*

[Vaquero 2007]VAQUERO, T. S. itSIMPLE : Ambiente Integrado de Modelagem e Análise de Domínios de Planejamento Automático. *DissertaÃ§Ã£o (Dissertação de Mestrado) — Universidade de São Paulo Paulo, 2007.*

[Vaquero 2011]VAQUERO, T. S. Pós-design para Problemas de Planejamento Automático : uma abordagem combinando diagnóstico, realidade virtual e reutilização de rationales. *Tese (Tese de doutorado) — Universidade de São Paulo, 2011.*

[Vaquero 2007]VAQUERO, T. S. et al. itsimple2.0: An integrated tool for designing planning environments. In: Proceedings of the 17th International Conference on Automated Planning and Scheduling (ICAPS 2007). Providence, Rhode Island, USA. *[S.l.: s.n.], 2007.*

[Vaquero 2009]VAQUERO, T. S. et al. Planning and Scheduling of Crude Oil Distribution in a Petroleum Plant. Artificial Intelligence*, 2009.*

[Vaquero 2012]VAQUERO, T. S. et al. itsimple4.0: Enhancing the modeling experience of planning problems. Proceedings of 22nd International Conference on Automated Planning and Sheduling. South America., *2012.*

[Vaquero 2006]VAQUERO, T. S. et al. On the use of UML.P for modeling a real application as a planning problem. In: Proceedings of the 16th International Conference on Automated Planning and Scheduling (ICAPS). *[S.l.: s.n.], 2006.*

[Vodrazka e Chrpa 2010]VODRAZKA, J.; CHRPA, L. Visual design of planning domains. in keps 2010: Workshop on knowledge engineering for planning and scheduling. 2010.

[Wilkins 1990]WILKINS, D. Can AI planners solve practical problems? *1990.*

[Wu 2003]WU, D. et al. Automating daml s web services composition using shop2. In: IN DIETER FENSEL, K. P. S.; JOHN MYLOPOULOS, e. (Ed.). International Semantic Web Conference. *[S.l.]: Lecture Notes in Computer Science, 2003. v. 2870.*

[Yang 1990]YANG, Q. Formalizing planning knowledge for hierarchical planning. *1990.*

[Zhao 2004]ZHAO, Y. et al. Towards formal verification of uml diagrams based on graph transformation. Proceedings of the IEEE International Conference on E-Commerce Technology for Dynamic E-Business, IEEE Computer Socity., *2004.*

[Zimmerman e Kambhampati 2003]ZIMMERMAN, T.; KAMBHAMPATI, S. Learning-assisted automated planning: looking back, taking stock, going forward. AI Magazine, *American Association for Artificial Intelligence, Menlo Park, CA, USA, v. 24, n. 2, 2003. ISSN 0738-4602.*

[Ziparo 2008]ZIPARO, V. A. et al. Petri net plans: A formal model for representation and execution of multi robot plans. ReCALL, *2008.*

Apêndice A

Apêndice A - Trecho de código da classe Java para o algoritmo de Tradução de UML para Redes de Petri.

```
     /***
* itSIMPLE: Integrated Tool Software Interface for Modeling PLanning
   Environments
*
* Copyright (C) 2007, 2008, Universidade de Sao Paulo
*

* This file is part of itSIMPLE and was updated in december/2015 by
   Rosimarci Tonaco-Basbaum to include the new translation process.
*
* itSIMPLE is free software: you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation, either version 3 of the License, or
```

```
* (at your option) any later version. Other licenses might be available
* upon written agreement.
*
* itSIMPLE is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with itSIMPLE. If not, see <http://www.gnu.org/licenses/>.
*
* Authors: Tiago S. Vaquero,
*       Fernando Sette,
*       Victor Romero,
        Rosimarci Tonaco-Basbaum.
**/

package src.languages.petrinets;


import java.io.IOException;
import java.util.ArrayList;
import java.util.HashSet;
import java.util.Iterator;
import java.util.List;
import java.util.LinkedList;

import org.jaxen.JaxenException;
import org.jaxen.XPath;
import org.jaxen.jdom.JDOMXPath;
import org.jdom.Attribute;
```

```
import org.jdom.Element;
import org.jdom.JDOMException;
import org.jdom.Parent;

import src.gui.ItSIMPLE;
import src.languages.xml.XMLUtilities;

public class toPNML {

   private static Element project = null;
   private static final int MODULE_OFFSET = 10;
   private static int maxY;
   private static int currentY;

   @SuppressWarnings("unchecked")
   public static Element buildPetriNet(Element project)
        {
      System.out.println("executando buildPetriNet...");
      toPNML.project = project;
      Element pnmlNodes = null;
      Element pnml = null;

      try {
pnmlNodes =
    (Element)XMLUtilities.readFromFile("resources/settings/commonData.xml").getRootEle
getChild("pnmlNodes").clone();

// ...bloco de código suprimido para não extender o tamanho do texto da
    tese.

        /**
```

```
        * faz a traducao de Diagrama de estado para PN
        *
        * @param stateMachineDiagram elemento XML do diagrama de estados
        * @param allModules
        * @return
        */


------>>>            // verificando se a acao tem precondicao
Element precondition = action.getChild("precondition");
if(precondition != null)
 // se a precondicao existe
 if(!precondition.getText().isEmpty())
 {
System.out.println("transicao_id = " +
    transition.getAttributeValue("id"));
preconditionID++;
 // acrescentar um lugar equivalente
 Element place = (Element)pnmlNodes.getChild("place").clone();

// set place id
place.setAttribute("id","p" + placeNumberID);
System.out.println("acrescentando precond >> " +
    place.getAttributeValue("id"));
//3.3.2 set place name
place.getChild("name").getChild("text").setText("pre-condition " +
    preconditionID);

//3.3.3 set place graphics
//set place position considering the size of the place image
```

```
place.getChild("graphics").getChild("position").setAttribute("x",
    Integer.toString(x-19));
place.getChild("graphics").getChild("position").setAttribute("y",
    Integer.toString(y-90));
 if(y > maxY){
 maxY = y;
     }

 //set toolspecific
place.getChild("toolspecific").setAttribute("version",version);
    place.getChild("toolspecific").getChild("type").setText("pseudoBox");
place.getChild("toolspecific").addContent(new
    Element("action").setText(action.getAttributeValue("id")));
place.getChild("toolspecific").getChild("state").detach();
//setText(state.getAttributeValue("id"));
place.getChild("toolspecific").getChild("class").detach();
//.setText(stateMachineDiagram.getChildText("class"));
//set marking
place.getChild("initialMarking").getChild("text").setText("1");

//3.3.4 add place
module.addContent(place);
placeNumberID++;

 ------>>>            // criando o arco entre a precondicao e a sua
     transicao(acao)
Element arc1 = (Element)pnmlNodes.getChild("arc").clone();
arc1.setAttribute("id","arc" + Integer.toString(arcNumberID));
arc1.setAttribute("source", place.getAttributeValue("id"));
arc1.setAttribute("target", transition.getAttributeValue("id"));
arc1.getChild("type").setAttribute("value", "read");
```

```
arcNumberID++;
arcList.add(arc1);
    }
   //5.4 add arcs

            Element sourcePlace = null;

            try {
               XPath path = new JDOMXPath("place[toolspecific/class='" +
                   classID +
                "' and toolspecific/state='" +
                    sourceState.getAttributeValue("id") + "']");
               sourcePlace = (Element) path.selectSingleNode(module);
            } catch (JaxenException e) {
               e.printStackTrace();
            }

            Element targetPlace = null;
            try {
               XPath path = new JDOMXPath("place[toolspecific/class='" +
                   classID +
                "' and toolspecific/state='" +
                    targetState.getAttributeValue("id") + "']");
               targetPlace = (Element) path.selectSingleNode(module);
            } catch (JaxenException e) {
               e.printStackTrace();
            }

            Element arc1 = (Element)pnmlNodes.getChild("arc").clone();
            arc1.setAttribute("id","arc" +
                Integer.toString(arcNumberID));
```

```
arc1.setAttribute("source",sourcePlace.getAttributeValue("id"));
arc1.setAttribute("target",transition.getAttributeValue("id"));
arc1.getChild("type").setAttribute("value", "normal");

arcNumberID = arcNumberID+1;
Element arc2 = (Element)pnmlNodes.getChild("arc").clone();
arc2.setAttribute("id","arc" +
    Integer.toString(arcNumberID));
arc2.setAttribute("source",transition.getAttributeValue("id"));
arc2.setAttribute("target",targetPlace.getAttributeValue("id"));
arc2.getChild("type").setAttribute("value", "normal");

if(actionPointsSize > 0){
   if(actionPointsSize % 2 == 0){
      for(int i = 0; i < actionPointsSize; i++){
         Element point = actionPoints.get(i);
         Element position =
             (Element)pnmlNodes.getChild("position").clone();
         int positionX =
             Integer.parseInt(point.getAttributeValue("x"));
         int positionY = currentY +
             Integer.parseInt(point.getAttributeValue("y"));
         if(positionY > maxY){
            maxY = positionY;
         }
         position.setAttribute("x",
             Integer.toString(positionX));
         position.setAttribute("y",
             Integer.toString(positionY));
         if(i < actionPointsSize/2){
            arc1.getChild("graphics").addContent(position);
```

```
                    }
                    else{
                        arc2.getChild("graphics").addContent(position);
                    }
                }
            }
        }

        //5.5 set arc graphics
        arcList.add(arc1);
        arcList.add(arc2);
        arcNumberID++;
    }
  }  // fim do for para traducao das acoes -> transicoes

                    // criar o retorno das precondicoes
                    for(Iterator<Element> arcIter = arcList.iterator();
                        arcIter.hasNext();){
    Element currentArc = arcIter.next();
    module.addContent(currentArc);
  }
  currentY = maxY + MODULE_OFFSET;
}

return module;
}
}
```