UNIVERSIDADE FEDERAL DO ESTADO DO RIO DE JANEIRO
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

MODELO DE PLANEJAMENTO TEMPORAL NÃO DETERMINÍSTICO CONSIDERANDO PROPRIEDADES DRAMÁTICAS EM MUDANÇA CONTÍNUA PARA STORYTELLING INTERATIVO

Paulo Abelha Ferreira

Orientador

Angelo Ernani Maia Ciarlini

RIO DE JANEIRO, RJ - BRASIL

OUTUBRO DE 2013

Ferreira, Paulo Abelha.
F383 Modelo de planejamento temporal não determinístico considerando propriedades dramáticas em mudança contínua para storytelling interativo / Paulo Abelha Ferreira, 2013.
135 f. ; 30 cm

Orientador: Angelo Ernani Maia Ciarlini.
Dissertação (Mestrado em Informática) - Universidade Federal do Estado do Rio de Janeiro, Rio de Janeiro, 2013.

1. Storytelling interativo. 2. Inteligência artificial. 3. Interação homem-máquina. 4. Lógica temporal modal. 5. Geração de enredos - Informática. I. Ciarlini, Angelo Ernani Maia. II. Universidade Federal do Estado do Rio de Janeiro. Centro de Ciências Exatas e Tecnológicas. Curso de Mestrado em Informática. III. Título.

CDD – 006.3

Dedicado a Lauro de Oliveira Lima, educador e dinamizador da Ciência.

Agradecimentos

Ao Prof. Angelo Ciarlini, cuja inteligência e paixão por abordar problemas de pesquisa me inspiraram durante todo o processo. É um privilégio ter sido seu orientando e um desafio acompanhar uma mente tão ávida por encontrar soluções elegantes e eficientes aos problemas.

À Juliana, minha linda, que eu amo tanto, e que retribuiu com mais amor ainda durante essa fase da minha vida.

A minha família, por acreditar na minha paixão pela pesquisa científica, me dando todo o apoio necessário.

Ao Vinícius Gottin, por todos os debates que contribuíram para fazer do mestrado uma experiência enriquecedora. Aos amigos de mestrado, por tornarem as aulas do início do mestrado mais interessantes e o tempo livre mais divertido. Aos amigos da vida, por demonstrarem constantemente a sorte que eu tenho de estar rodeado de pessoas incríveis.

Aos professores e funcionários da UNIRIO. Em especial, ao professor Sean, pelas aulas e debates estimulantes, e à Alessandra e ao Douglas, por toda a simpatia e disponibilidade.

À CAPES, pelo financiamento dado durante o curso.

FERREIRA, Paulo Abelha. **Modelo de Planejamento Temporal Não Determinístico considerando Propriedades Dramáticas em Mudança Contínua para Storytelling Interativo**. UNIRIO, 2013. 124 páginas. Dissertação de Mestrado. Departamento de Informática Aplicada, UNIRIO.

RESUMO

A área de *Storytelling* Interativo estuda as possibilidades de gerar histórias condicionadas à intervenção do usuário. A principal técnica utilizada para a geração de histórias é o planejamento automático, que garante coerência e conformidade a especificações de gêneros. Dois requisitos podem ser destacados para as histórias: eventos não determinísticos, ou seja, com mais de um desenrolar possível; e conformidade a arcos dramáticos típicos de um gênero. Eventos não determinísticos enriquecem a história e aumentam a diversidade, e o arco dramático é uma parte fundamental da narrativa. Aqui, arcos dramáticos típicos são modelados como fórmulas de uma lógica temporal modal que fala sobre diferentes *propriedades dramáticas* (e.g. *alegria*, *suspense* e *medo*) variando continuamente em tempo contínuo e ramificado. Eventos são autômatos não determinísticos cujos estados falam do valor corrente das propriedades dramáticas como uma combinação linear de seus valores iniciais. Essa pesquisa se encontra sob um paradigma de TV Interativa, onde o tempo de resposta é crítico. Atender a ambos os requisitos aumenta a complexidade da geração ao ponto de torná-la inviável. Como estratégia para buscar eficiência, eventos passam por uma fase de preprocessamento que provê informação temporal indexada a um planejador. Propõe-se um modelo de planejamento HTN que incorpora a verificação das formulas de lógica temporal durante o planejamento. Um protótipo resultante está operacional e compatível com o LOGTELL, um sistema de Storytelling Interativo existente.

Palavras-chave: Storytelling Interativo; Arco Dramático; Planejamento Temporal; Inteligência Artificial; HTN; Não determinismo

ABSTRACT

Interactive Storytelling is an area that studies the possibilities of creating stories conditioned to user intervention. The prevailing approach to generating coherent stories belonging to a genre is automated planning. Two features of stories can be highlighted: nondeterministic events, that is, with different possible outcomes; and conformity to a genre's typical dramatic arcs. Nondeterministic events enrich the story, increasing diversity, and dramatic arcs are a fundamental part of a narrative. Here, typical dramatic arcs are modeled as temporal logic formulae over *dramatic properties* (e.g. *joy*, *suspense* and *fear*) that vary continuously through continuous and branching time. Additionally, events are modeled as nondeterministic automata whose states express the current value of dramatic properties as linear combinations of their initial value. This research is under a paradigm of Interactive TV, in which response time is critical. Attending both requisites may increase the complexity of the generation process to the point of intractability. As a strategy towards efficiency, events go through a preprocessing phase that provides indexed information to a planner. An HTN temporal planning model is proposed that deals with verification of temporal logic formulae during planning. A resulting prototype is operational and compatible with LOGTELL, an existing Interactive Storytelling system.

Keywords: Interactive Storytelling; Dramatic Arc; Temporal Planning; Artificial Intelligence; HTN; Nondeterminism

Sumário

1 Introdução 1
1.1 Motivação 1
1.2 Problema 3
1.3 Proposta 5
1.4 Estrutura 7
2 Fundamentação Teórica 8
2.1 Planejamento 9
2.1.1 Planejamento Clássico 9
2.1.2 Planejamento com Rede Hierárquica de Tarefas 12
2.1.3 Planejamento Não Determinístico 15
2.1.4 Planejamento Temporal 17
2.2 Lógica Temporal Modal 18
2.2.1 Computation Tree Logic 19
2.2.2 Uma lógica temporal para *Storytelling* Interativo 20
2.3 Storytelling Interativo 23
2.3.1 Abordagens Character-based e plot-based 24
2.3.2 Planejamento para Geração de Histórias 25
2.3.3 Controle Dramático da Narrativa 27
2.4 Logtell 29
2.5 Modelo e Implementação de Geração de Enredos com Não Determinismo 31
2.5.1 Visão Geral 31
2.5.2 Requisitos 32
2.5.3 Planejamento HTN 33
3 Modelo de Geração de Enredos com Não Determinismo e Verificação de Fórmulas Temporais 35
3.1 Visão Geral 35
3.2 Requisitos 37
3.3 Arquitetura 38
3.4 Drama Preprocessor 41
3.4.1 Visão Geral 41
3.4.2 Autômatos 41
3.4.3 Preprocessamento 43
3.5 Drama Consult 46
3.5.1 Visão Geral 46
3.5.2 Obtendo os Finais Ativos de um Evento 47

3.5.3 Obtendo Fórmulas Residuais 47
3.6 Drama Crafter 48
3.6.1 Visão Geral 48
3.6.2 Planejamento Não Determinístico com Verificação de Propriedades Dramáticas 48
3.6.3 Manutenção em Forma Normal 49
3.6.4 Fórmulas Obrigatórias e Não Obrigatórias 50
3.6.5 Anotação de Fórmulas Satisfeitas e Não Satisfeitas 51
3.6.6 Operadores de Estado e Eventos Finais no Plano 52
3.6.7 Exemplo de Planejamento HTN Não Determinístico com Verificação Temporal 53
4 Algoritmo 57
4.1 Visão Geral 57
4.2 Algoritmo de Entrada 58
4.3 Algoritmo Principal de Planejamento 59
4.4 Satisfação de Fórmulas nas Folhas 63
4.5 Tratamento de Operadores Abstratos 65
4.6 Tratamento de Operadores Compostos 66
4.7 Tratamento de Operadores Básicos 66
4.7.1 Visão Geral 66
4.7.2 Obtenção de Fórmulas Residuais 70
4.7.3 Filtragem Por Final 73
4.7.4 Verificação de Fórmulas 74
4.8 Otimização do Controle do Grau Máximo de Não Determinismo 75
5 Avaliação 77
5.1 Visão Geral 77
5.2 Avaliação do Planejamento HTN Não Determinístico 78
5.2.1 Especialização de Operadores Abstratos 78
5.2.2 Suporte a Eventos Não Determinísticos 79
5.2.3 Controle Do Nível de Não Determinismo 80
5.2.4 Tratamento de Tentativas 80
5.2.5 Conclusão 81
5.2.6 Utilizando Fórmulas Temporais 82
5.2.7 Comparação com “geração e teste” 85
5.2.8 Resumo dos Testes 85
6 Conclusão 87
6.1 Principais Contribuições 88

6.1.1 Modelo de Geração de Enredos Não Determinísticos com Verificação de Fórmulas Temporais sobre Propriedades Dramáticas Contínuas 88
6.1.2 Melhoria de Desempenho no Planejamento Não Determinístico (independente do uso de restrições temporais) 88
6.1.3 Algoritmo Temporal NDet-HTN 88
6.1.4 O Protótipo 89
6.2 Trabalhos Futuros 89
6.2.1 Integração com o LOGTELL 89
6.2.2 Gramática Completa da Lógica Temporal Proposta 89
6.2.3 Aplicação em Outros Domínios 90
6.2.4 Inferir Arcos Dramáticos Típicos para cada Capítulo 90
6.2.5 Histórias *à la carte* 90
7 Referências 92
ANEXO I – LOGTELL 99
I.2 Arquitetura 99
I.2.1 Especificação do Gênero das Histórias 101
I.2.2 Contexto Exemplo 105
I.2.3 IPG (*Interactive Plot Generator*) 107
ANEXO II – MODELO E IMPLEMENTAÇÃO DE GERAÇÃO DE ENREDOS COM NÃO DETERMINISMO 109
II.1 Arquitetura 109
II.1 Controle do Nível de Não Determinismo 112
II.2 Incorporação do Conceito de Tentativas 112
II.3 Processamento em Paralelo 113
II.4 Implementação 113
II.4.1 Mudanças no IPG 114
II.4.2 Mudanças no Gênero “Espadas & Dragões” e no Contexto 114
II.6.3 O Algoritmo NDet-HTN 117

Índice de Figuras e Tabelas

Figura 1. O problema do aumento de granularidade no processo de planejamento. 4
Figura 2. Planejador PFD (GHALLAB *et al*., 2004, p. 243, tradução nossa). 14
Figura 3. Semântica da lógica temporal (ARAÚJO, 2011). 22
Figura 4. Simplificações de fórmulas da lógica temporal (ARAÚJO, 2011) 23
Figura 5. Sistema de *Storytelling* Interativo Façade. 24
Figura 6. Dramatização do evento *Donate* no sistema LOGTELL. 31
Figura 7. Arquitetura do modelo proposto 39
Figura 8. Módulo NDet-IPG 40
Figura 9. Autômato para o evento "*Fight_to_the_death*" 41
Figura 10. Árvores de execução para o evento *fight_to_the_death*. 45
Figura 11. Exemplo de planejamento HTN temporal com falha. 54
Figura 12. Exemplo de planejamento HTN temporal com sucesso. 55
Figura 13. Algoritmo inicial para o Temporal NDet-HTN 59
Figura 14. Algoritmo principal do Temporal NDet-HTN 61
Figura 15. Verificação de fórmulas em uma folha (final do plano). 63
Figura 16. Tratamento de operadores abstratos no Temporal NDet-HTN. 66
Figura 17. Tratamento de operadores compostos no Temporal NDet-HTN 66
Figura 18. Tratamento de operadores básicos no Temporal NDet-HTN. 68
Figura 19. Algoritmo para obtenção de fórmulas residuais 71
Figura 20. Algoritmo para verificação de fórmulas. 74
Figura 21. Controle do grau de não determinismo. 76
Figura 22. Arquitetura geral do LOGTELL (DÓRIA, 2009). 100
Figura 23. Tabela com as entidades do gênero "Espadas & Dragões" do LOGTELL. 102
Figura 24. Heranças entre entidades do gênero "Espadas & Dragões". 102
Figura 25. Relacionamentos entre entidades do gênero "Espadas & Dragões". 102
Figura 26. Atributos e tipos de entidades/relações do gênero "Espadas & Dragões". . 103
Figura 27. Papéis para entidades do gênero "Espadas & Dragões". 103
Figura 28. Eventos do gênero "Espadas & Dragões" do LOGTELL. 104
Figura 29. Regras de inferência de objetivo do gênero "Espadas & Dragões". 104
Figura 30. Diagrama do do gênero "Espadas & Dragões" (SILVA, 2010). 105
Figura 31. Descrição dos personagens do contexto do gênero "Espadas & Dragões". 106
Figura 32. Descrição dos locais do contexto do gênero "Espadas & Dragões". 107
Figura 33. Módulo NDet-IPG (SILVA, 2010). 110
Figura 34. Nova arquitetura do LOGTELL 112
Figura 35.Novos eventos determinísticos para o gênero "Espadas & Dragões". 115
Figura 36. Novos eventos não determinísticos para o gênero "Espadas & Dragões". . 115
Figura 37. Hierarquia de eventos no gênero "Espadas & Dragões" (SILVA, 2010). .....
Figura 38. Eventos compostos para o gênero "Espadas & Dragões" (SILVA, 2010). .....
Figura 39. Algoritmo NDet-HTN (SILVA, 2010). .....
Figura 40. Algoritmo de atualização de estados do NDet-HTN (SILVA, 2010). .....

1 Introdução

1.1 Motivação

Essa pesquisa é motivada pela área de *Storytelling* Interativo, que abarca o estudo da criação de histórias sob a influência de um ou mais usuários. Histórias interativas são usadas em jogos, educação e entretenimento (GLASSNER, 2004), (HARTSOOK *et al*., 2011), (KLINE, 2009) e (SILVA *et al*, 2011). Essa característica interdisciplinar leva à necessidade de colocar em harmonia visões e requisitos muito distintos, que acabam por confluir através da Computação, capaz de viabilizar a geração de histórias, a dramatização e a interação com o usuário.

A área vem crescendo e já se estabelece com uma considerável gama de grupos de pesquisa pelo mundo. O principal consórcio de grupos de pesquisa na área foi o *Integrating Research in Interactive Storytelling* (IRIS), que durou de 2009 a 2012 e contou com dez parceiros de sete países diferentes. Há uma base de dados, relacionada ao IRIS, que mantém uma indexação dos artigos publicados na área. Essa base de dados contém 1.837 publicações (PETTA, 2013). Há também, atualmente um consórcio de pesquisa denominado RIDERS (2013). Além disso, a área de *Storytelling* Interativo possui, desde 2008, conferência própria, a *International Conference on Interactive Digital Storytelling* (ICIDS).

Há diversas frentes de pesquisa em SI: geração de histórias (PORTEOUS *et al*., 2010), apresentação de histórias (DIAS e MARTINHO, 2011), incorporação de interferência do usuário (LIMA *et al*., 2012) e (GILROY *et al*., 2012), autoria (MEHM *et al*., 2012), dentre outras. O foco deste trabalho é na frente de geração de histórias.

A abordagem computacional tem sido essencial para a geração de histórias (GERVÁS, 2009). Na frente de pesquisa de geração de histórias, destaca-se o uso de algoritmos de planejamento automático (BARROS e MUSSE, 2007) e (PORTEOUS *et al*., 2010). Planejamento automático (ou somente planejamento) é uma subárea de pesquisa da área de Inteligência Artificial (IA) que visa à combinação de ações para

atingir objetivos. No planejamento, as ações são escolhidas para compor um plano de acordo com precondições que devem ser satisfeitas, e, além disso, têm efeitos que mudam o ambiente. Um plano, que é um encadeamento de ações, pode ser equiparado a uma história, que pode ser vista como um encadeamento de eventos. A área de planejamento é consagrada, possuindo diversos tipos de algoritmos para os mais variados domínios. O planejamento é amplamente utilizado na área de SI, pois o garante a coerência da história a partir de especificações formais de gêneros.

Técnicas de planejamento podem ser utilizadas para aumentar a variabilidade das histórias geradas. Como boas candidatas entram as técnicas para lidar com não determinismo (ROBERTS e ISBELL, 2008), (BOSSER *et al*., 2010) e (WARE e YOUNG, 2011), onde um evento pode ter mais de um final possível. Deste modo, por exemplo, em um evento de luta até a morte em que o vilão briga com o herói, pode haver um final no qual o vilão é morto pelo herói, este assassinado pelo vilão, ou então, num trágico final, onde ambos sucumbem à batalha. Eventos não determinísticos permitem, por exemplo, que uma execução de uma mesma história seja bem diferente de outra, ao seguir por diferentes finais de cada evento encadeado. Além da característica de possuir mais de um final, é interessante que os eventos também sejam ricos em sua descrição, isto é, falem de vários acontecimentos internos a ele. No exemplo do evento de luta até a morte, pode-se ter primeiramente um diálogo entre os dois combatentes, quem sabe um ou outro golpe mais perigoso. Enfim, diferentes acontecimentos que levem a diferentes caminhos até cada um dos finais.

Uma história interessante, que mantém a atenção do público, deve possuir variação nas diferentes emoções em jogo na narrativa (ZAGALO *et al*., 2005). Seria interessante poder garantir que as histórias geradas obedecessem a certos requisitos estéticos sobre como emoções relacionadas à história deveriam variar ao longo do tempo. Essa abordagem toma inspiração na narratologia clássica, que já destacava a importância do *arco dramático* (ARISTÓTELES e BUTCHER, 2011) e (FREYTAG, 1863), que, de uma forma ou outra, representa o desenvolvimento da carga dramática ao longo do tempo. A carga dramática pode estar bem conectada às emoções do personagem principal, mas geralmente representa uma conjunção de diversas emoções em jogo na narrativa. Em narratologia, há diversas maneiras de abordar o arco dramático (BRANIGAN, 1992). Na área de *Storytelling* Interativo, o estudo da geração de histórias com conformidade a arcos dramáticos definidos explicitamente é incipiente e tem se concentrado em restrições temporais sobre emoções (ARAÚJO e CIARLINI,

2011), adaptação a arcos de tensão (BARROS e MUSSE, 2008), e verificação e geração de suspense (CHEONG, 2008) e (O'NEIL e RIEDL, 2011).

Pode-se pensar em *arcos dramáticos típicos* de um gênero, que definem como determinadas *propriedades dramáticas* (e.g. *alegria*, *suspense* e *medo*) variam ao longo do tempo. Desta forma, uma modelagem explícita de arcos dramáticos a serem cumpridos pode estar presente no gênero. Para um conto de fadas, por exemplo, poderia haver um arco dramático típico que diz que a história deve terminar sempre feliz (em todos os finais para histórias com eventos não determinísticos, que podem possuir mais de um final). Para um *thriller* de ação, pode-se querer que o nível de suspense obedeça a certos padrões de pico seguido de um relaxamento, mas que o nível de suspense nunca caísse abaixo de certo valor limítrofe.

A partir da observação da necessidade de se gerar histórias interessantes, com eventos não determinísticos ricos e conformidade a arcos dramáticos, essa pesquisa encontra duas principais frentes motivadoras:

- A variabilidade de enredos resultante da incorporação de eventos não determinísticos; e
- A qualidade de enredos resultante da garantia de conformidade a arcos dramáticos típicos e da incorporação de eventos não determinísticos ricos.

1.2 Problema

A seção 1.1 apresenta a motivação de gerar histórias com eventos não determinísticos ricos e enredos que se enquadrem em arcos dramáticos típicos pré-definidos para um gênero. Além disso, se discute que a principal técnica utilizada para lidar com geração de histórias é o planejamento, que é essencialmente um processo combinatório.

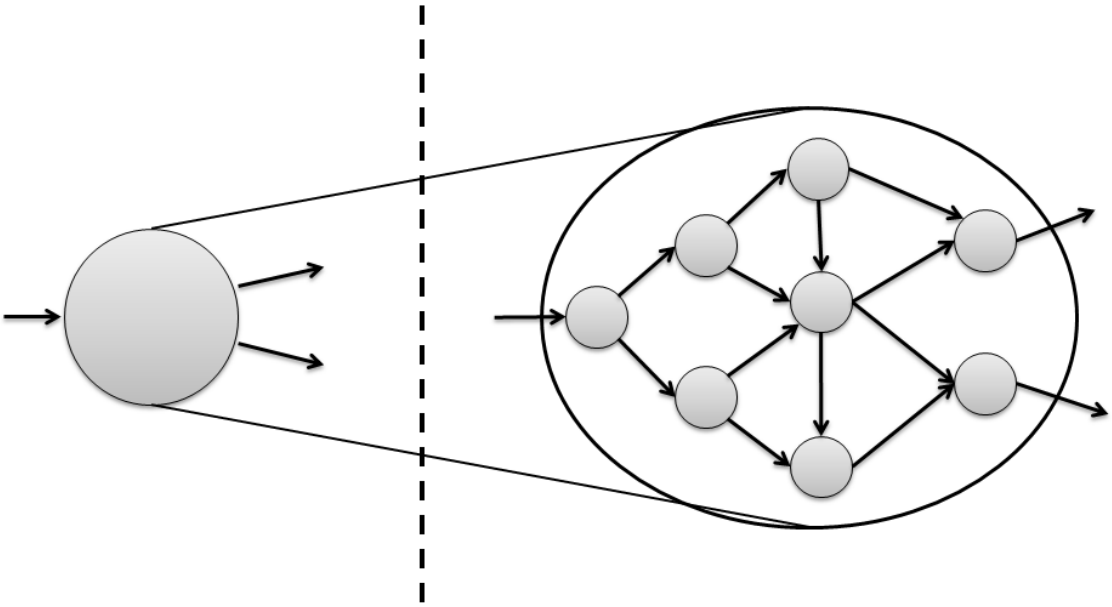

Figura 1. O problema do aumento de granularidade no processo de planejamento.

A modelagem de eventos ricos, que alterem as propriedades dramáticas ao longo do tempo em diversas transições, aliada à incorporação do não determinismo, pode levar o processo de geração a ficar intratável. Combinar eventos considerando não só precondições e efeitos, mas também as transições internas é um problema que, para não ficar intratável precisa de alguma estratégia para aumentar a eficiência.

Na Figura 1, à esquerda, há um evento sendo considerado como um elemento único, com dois finais possíveis. À direita, há as transições internas deste evento a serem consideradas. Aumentar a granularidade de um evento para o nível das suas transições internas leva a uma quantidade considerável de verificações a mais.

O processo de planejamento em *Storytelling* Interativo comumente considera os eventos como elementos "atômicos" para a composição, isto é, o processo não entra no mérito dos acontecimentos internos de cada evento. Considere um evento de perseguição, onde um personagem persegue o outro em certo local. A definição deste evento pode ser complexa, especificando, por exemplo, como a perseguição ocorre, as diferentes atitudes dos personagens em diferentes momentos da perseguição etc. Todavia, durante o planejamento, a escolha de um evento na composição não leva em conta todos esses detalhes, mas somente uma especificação que diz o que precisa estar valendo no mundo para que o evento possa ocorrer, e o que passa a valer no mundo

após o evento (no caso de um evento não determinístico, podem passar a valer coisas diferentes para cada final).

A fim de garantir que a história gerada se enquadre em um arco dramático, aproveitando a riqueza da descrição das transições internas de um evento, é interessante que as variações das propriedades dramáticas nestas transições sejam consideradas pelo processo de planejamento. Para tal, seria necessário planejar entrando no mérito de cada uma dessas variações, o que aumentaria consideravelmente a complexidade. Além do enquadramento em relação aos arcos dramáticos, também se considera o não determinismo que, por si só, leva a um aumento considerável na complexidade. O que ocorre devido ao aumento de eventos com as ramificações não determinísticas e a necessidade de garantir a coerência levando em conta a ocorrência dos diferentes finais dos eventos.

O uso de planejamento em *Storytelling* Interativo não coloca o foco no tempo do planejamento, mas sim na capacidade de se gerar planos interessantes (WARE e YOUNG, 2010), o que levou ao estudo de possíveis métricas para os planos gerados (PORTEOUS e CAVAZZA, 2009) e (RIEDL e YOUNG, 2004). Todavia, o modelo apresentado nessa pesquisa é construído para operar sob um paradigma de TV Interativa, isto é, onde há um ciclo contínuo entre a apresentação ao usuário da história e a geração desta em um servidor. Portanto, há restrições severas de tempo no processo de geração de histórias, o que leva à busca de estratégias para poder aumentar a velocidade sem perder a coerência ou a qualidade.

O processo de planejamento precisaria, antes de adicionar cada evento à história, verificar se suas variações internas contradizem o arco dramático que se deseja satisfazer. Caso se deseje que a história esteja sempre alegre, então é necessário verificar o nível de alegria em cada transição interna de cada evento a ser adicionado. Aqui nessa pesquisa, busca-se apresentar uma estratégia de planejamento temporal não determinístico para evitar a explosão combinatória do não determinismo e a verificação detalhada considerando cada transição de todos os eventos.

1.3 Proposta

Uma possibilidade para gerar enredos não lineares e que se enquadrem em certo arco dramático é uma abordagem do tipo “geração e teste”. Ou seja, gera-se o enredo e depois se verifica se este satisfaz ao arco dramático desejado. De fato, esta abordagem

se torna possível com os trabalhos de SILVA (2010) e ARAÚJO (2011). O primeiro se utilizada de planejamento com rede hierárquica de tarefas (HTN) (GHALLAB *et al.*, 2004) para, a partir da posse de conhecimento de domínio, aumentar a eficiência do planejamento, de modo a ser capaz de tratar em tempo hábil as múltiplas alternativas de história decorrentes do não determinismo. O segundo torna possível verificar se histórias não determinísticas, que já tiverem sido geradas, obedecem a restrições temporais sobre a variação de emoções através da utilização de uma lógica temporal modal que é uma extensão da *Computation Tree Logic* (CTL) (HUTH e RYAN, 2010), levando em conta tempo contínuo e variação contínua nos valores das emoções. A adaptação e união desses dois trabalhos numa abordagem de "geração e teste" são viáveis, pois o trabalho de ARAÚJO (2011) é compatível com o de SILVA (2011), tanto no modelo como no nível das estruturas de dados utilizadas. Além disso, o trabalho de ARAÚJO (2011) também seria flexível para passar a falar de propriedades dramáticas em vez de emoções.

O problema com esse tipo de abordagem "geração e teste" é que, apesar do planejamento não precisar entrar no mérito das transições de um evento, é necessário gerar uma história inteira antes de se descobrir se ela se adequa às restrições sobre a variação das emoções. Seria interessante, portanto, que fosse possível raciocinar sobre as restrições durante o processo de geração, mas sem precisar considerar as transições internas de um evento. Para contornar isso, propõe-se uma estratégia de preprocessamento dos eventos para gerar um módulo de consulta que forneça, para cada evento e certa fórmula temporal de entrada, representando uma restrição, qual fórmula temporal deve valer no futuro. Desta forma, o processo de geração pode consultar este módulo sem ter que entrar no mérito da variação das emoções dentro de um evento. O preprocessamento utilizado para o teste do protótipo dessa pesquisa foi feito a partir do protótipo desenvolvido por GOTTIN (2013) — um trabalho motivado pelas mesmas frentes. O modelo de planejamento proposto nessa pesquisa contém um algoritmo para um planejador temporal HTN que estende o planejador de SILVA (2010), adicionando a capacidade de raciocinar sobre os arcos dramáticos como fórmulas temporais sobre propriedades dramáticas que variam continuamente em tempo contínuo e ramificado.

No modelo desenvolvido por essa pesquisa, os requisitos estéticos sobre emoções, como levantados por ARAÚJO (2011), são repensados para serem arcos dramáticos típicos de um gênero que falam sobre como propriedades dramáticas devem variar ao longo do tempo. Além disso, a variação das propriedades dramáticas acontece

dentro dos eventos (modelados como autômatos), em cada estado, como uma combinação linear relativa aos valores das propriedades dramáticas no início do evento. Esta modelagem é mais expressiva do que a feita em (ARAÚJO, 2011), pois cada estado do evento pode colocar o valor de uma propriedade dramática como sendo relativo a um (ou mais) valor(es) de outra(s) propriedade(s) no estado inicial, e não somente em relação a ela mesma.

Os requisitos levantados por SILVA (2010) para o planejador continuam a ser atendidos e o algoritmo se mantém totalmente compatível. O modelo consiste de duas etapas: uma etapa de preprocessamento e uma etapa de planejamento. Esta segunda etapa é feita a partir do desenvolvimento de um algoritmo baseado no algoritmo de planejamento de SILVA (2011), mantendo-se todas as funcionalidades e aplicabilidades. O novo algoritmo lida com planejamento HTN não determinístico e temporal, considerando tempo contínuo e variação contínua das propriedades dramáticas. Para testar o modelo, foi criado um protótipo que se mostrou rápido e eficiente, confirmando a hipótese guia dessa pesquisa de que é possível gerar e verificar, em tempo real e em um mesmo processo, histórias que obedeçam a arcos dramáticos típicos em tempo tratável sob o paradigma de TV Interativa. Uma apresentação resumida da proposta dessa pesquisa pode ser encontrada em (FERREIRA *et al*., 2013).

1.4 Estrutura

Esta dissertação é composta de seis capítulos e uma seção final com as referências bibliográficas. O segundo capítulo contém a fundamentação teórica necessária à compreensão da pesquisa. O terceiro capítulo apresenta o modelo proposto e explica como este resolve o problema. O quarto capítulo detalha a implementação do modelo, com os algoritmos utilizados. O quinto capítulo contém os testes, incluindo a comparação com o trabalho de SILVA (2011), e uma discussão dos resultados. O sexto capítulo fecha a dissertação com as conclusões gerais, mostrando as principais contribuições e possíveis trabalhos futuros para dar prosseguimento à pesquisa. Por fim, há a seção de referências bibliográficas e, em seguida, os anexos I e II. O anexo I descreve em maior detalhe o sistema LOGTELL e o anexo II, o trabalho de SILVA (2010).

2 Fundamentação Teórica

Este capítulo visa apresentar a fundamentação teórica necessária à pesquisa desta dissertação. Utiliza-se uma abordagem para geração de histórias em *Storytelling Interativo* que une planejamento não determinístico com redes hierárquicas de tarefas (*hierarchical task networks* – HTN), que incorpora a verificação de fórmulas temporais modais sobre a variação contínua de propriedades dramáticas.

Na seção 2.1, é feita uma revisão da área de planejamento (GHALLAB *et al*., 2004), explicando, em particular, as abordagens mais relacionadas ao tipo de planejamento necessário para a aplicação. Começa-se com a descrição do planejamento clássico e de planejamento HTN. Em seguida, são apresentadas as principais alternativas de planejamento não determinístico.

A seção 2.2 aborda lógicas temporais modais, necessárias para se especificar restrições sobre sequências de eventos. Descreve-se a *Computation Tree Logic (CTL)*, usualmente adotada para especificar restrições sobre possíveis sequências de eventos. Detalha-se então a extensão de CTL, formulada em (ARAÚJO, 2011), especificamente para tratar de aplicações onde se deseja trabalhar com tempo contínuo em um contexto de *Storytelling Interativo*.

Na seção 2.3, é feita uma revisão da área de *Storytelling Interativo* e a seção 2.4, apresenta o sistema LOGTELL de *Storytelling Interativo*. Por fim, a seção 2.5 descreve o algoritmo de planejamento não determinístico baseado em HTN (SILVA, 2010) que foi desenvolvido no intuito de tratar eventos não determinísticos na geração de enredos. No contexto desta dissertação, esse algoritmo é modificado para incorporar a verificação de fórmulas temporais sobre propriedades dramáticas durante o próprio planejamento.

2.1 Planejamento

Planejamento é a atividade de selecionar e combinar ações para atingir objetivos. Quando se pretende ir ao supermercado fazer compras, se torna necessário planejar, mesmo que inconscientemente, o encadeamento de diversas ações para que se possa atingir o objetivo de comprar os itens que faltam na dispensa. Certamente há mais de um encadeamento satisfatório, como por exemplo: ir de carro ou de ônibus; ou começar pela fila do pão ou pelos legumes. Cada encadeamento combinando essas possibilidades pode ser pensado como um plano que satisfaz um objetivo. Este é um exemplo concreto do dia-a-dia, mas é interessante formalizar esses conceitos para que se possa automatizar o processo de planejamento.

Dentro da área de Inteligência Artificial, *planejamento automático* é uma subárea que estuda diferentes abordagens para realizar planejamentos computacionalmente. Conceitos tais como *plano*, *ação*, *objetivo*, dentre outros, são traduzidos para conceitos formais, e as diferentes abordagens e soluções podem então ser teoricamente estudadas, comparadas e implementadas computacionalmente. Daqui para frente, o termo *planejamento* é utilizado como sinônimo de *planejamento automático*, salvo diferenciação explícita. Planejamento é uma área fundamental de estudo e é utilizado nas aplicações mais importantes e diversas. Dentre elas, pode-se destacar a fabricação de circuitos integrados, robótica, busca de informações em bancos de dados e controle de naves espaciais.

2.1.1 Planejamento Clássico

O termo *planejamento clássico* é utilizado para se referir à abordagem de pesquisa que assume um conjunto de restrições a fim de tornar os problemas tratáveis. Em se tratando de *planejamento clássico*: o agente planejador deve ter acesso completo aos dados do ambiente; o ambiente deve ser determinístico (uma ação só tem um resultado possível e nada pode modificar o resultado de uma ação); o ambiente deve ser estático (nada muda enquanto o agente não agir); o número de estados em que o sistema pode estar é finito; o tempo é implícito, ou seja, as ações não têm duração e geram seus efeitos imediatamente; os planos são sequências totalmente ordenadas de ações e os objetivos a satisfazer são restritos à especificação do que deve valer no estado final a ser alcançado após a execução do plano, ou seja, não especificam restrições sobre as ações e estados intermediários.

Na definição formal de planejamento clássico, os principais conceitos são os de *domínio de planejamento*, *problema de planejamento*, *função de transição de estados*, *plano* e *objetivo*. Um domínio de planejamento D é um conjunto $D = (S,A,\gamma)$ onde S é o conjunto de estados possíveis, A é o conjunto de ações possíveis e γ é uma função de transição de estados $\gamma: S \times A \rightarrow S$, sendo $\gamma(s,a)$ o estado atingido através da aplicação da ação a em s. Um estado $s \in S$ é um conjunto de proposições que indica o que está valendo no mundo. Uma ação $a \in A$ é definida por meio de uma tripla $<precond(a), efeitos^{+}(a), efeitos^{-}(a)>$ tal que $precond(a)$ é chamado de *precondições* de a e $efeitos^{+}(a)$ e $efeitos^{-}(a)$ são chamados de *efeitos* de a, correspondendo, respectivamente, às proposições que devem passar a valer e devem deixar de valer como resultado da ação. É necessário que ambos os conjuntos de efeitos sejam disjuntos, isto é, $efeitos^{+}(a) \cap efeitos^{-}(a) = \emptyset$. Uma ação é dita *aplicável* em um estado s se $precond(a) \subseteq s$, o que significa que é possível executar a ação no estado. A função de transição de estados é definida por meio das definições das ações: $\gamma(s,a) = (s - efeitos^{-}(a) \cup efeitos^{+})$. Uma generalização útil em muitos casos é o conceito de *operador* que é uma tripla semelhante à definida acima para uma ação, mas cujos conjuntos de precondições e efeitos contêm literais em vez de proposições. Assim sendo, uma ação passa a ser uma instância *ground* de um operador, isto é, uma tripla onde não há símbolos de variável. Considere, por exemplo, as ações *a1* e *a2* e o operador *o1* a seguir:

a1: ({acessível('b1')},pegar('rb1','b1'),{segurando('rb1','b1')},{acessível('b1')})
a2: ({acessível('b1')},pegar('rb2','b1'),{segurando('rb2','b1')},{acessível('b1')})
o1: ({acessível(O)},pegar(R,O),{segurando(R,O)},{acessível(O)})

As ações *a1* e *a2* são instâncias *ground* do operador o1, que generaliza a ação de um robô representado por uma variável *R* pegar um objeto acessível representado pela variável *O*, passando a segurá-lo e, assim, tornando o objeto não mais acessível. Neste caso, há duas instâncias do operador, com as ações do robô *r1* pegar o bloco *b1* ou do robô *r2* pegar o bloco *b1*. Com essa generalização, as ações em um domínio de planejamento podem ser especificadas genericamente por um conjunto *O* de operadores.

Um problema de planejamento é uma tripla $P = (D,s_0,g)$ onde D é um domínio de planejamento, s_0 é o estado inicial do mundo e g representa as proposições que devem estar satisfeitas em um estado objetivo que se deseja atingir com um plano. Um plano é uma sequência $\pi = <a_1,a_2,...,a_k>$, com $k \geq 0$ e o estado produzido pela aplicação

de um plano π a um estado s é obtido através da definição abaixo que estende a definição da função de transição de estados:

$$\gamma(s,\pi) = \begin{cases} s, & \text{se } k = 0 \text{ (i.e. } \pi \text{ é vazio)} \\ \gamma(\gamma(s,a_1),<a_2,...,a_k>) & \text{se } k > 0 \text{ e } a_1 \text{ aplicável a } s \\ \text{indefinida} & \text{caso contrário} \end{cases}$$

Um plano π satisfaz o objetivo g se e somente se $g \subseteq \gamma(s,\pi)$. Dado um problema de planejamento clássico, este pode ser visto como uma busca. O espaço de busca é um grafo direcionado de transições entre estados, onde um nó é um estado e uma aresta, uma ação. Neste grafo, se deseja encontrar um caminho que leve de um nó representando o estado inicial até um nó representando um estado objetivo. Como o espaço de busca pode ser muito grande, normalmente o grafo de estados não é explicitamente construído, mas as possíveis transições entre estados são examinadas com diversas estratégias, podendo ir do estado inicial para o estado final ou do estado final para o inicial. Esse tipo de algoritmo é denominado *planejamento no espaço de estados*.

O *planejamento no espaço de planos* (GHALLAB *et al*., 2004) é uma variante do planejamento clássico, que trabalha com planos na forma de conjuntos parcialmente ordenados de instâncias de operadores. Ele também pode ser visto como uma busca, mas, nesse caso, o espaço de busca é um grafo direcionado onde os nós são *planos parcialmente especificados* e arestas são *refinamentos* de planos. Parte-se de um plano contendo apenas duas ações fictícias, uma sem precondições que cria o estado inicial do mundo, e uma, sem efeitos, que tem como precondição o objetivo a ser atingido. Fazem-se então refinamentos sobre os planos até que se obtenha um plano que tenha a garantia de que qualquer ordem total, compatível com a ordem parcial, leva a sequências de ações aplicáveis. Os refinamentos de planos são tipicamente a adição de ações (instâncias de operadores), instanciações de variáveis e o estabelecimento de restrições sobre a ordem das ações ou os valores das variáveis.

O planejamento clássico tem restrições para manter o problema de planejamento tratável. Todavia, em aplicações reais, uma ou mais dessas restrições normalmente precisam ser relaxadas a fim de tornar possível uma modelagem mais real do problema, o que torna a busca de uma solução mais complexa. Essa pesquisa está particularmente interessada no relaxamento da restrição de determinismo, ou seja, deseja-se assumir que

ações podem ter mais de um resultado possível. Desta maneira, tem-se uma função de transição de estados $\gamma: S \times A \rightarrow 2^S$, definindo que uma ação executada em um estado pode levar a um conjunto de estados e não somente a um estado único.

2.1.2 Planejamento com Rede Hierárquica de Tarefas

Mesmo o planejamento clássico em diversas situações torna-se não tratável por causa do aspecto intrinsecamente combinatório da busca por encadeamentos de ações válidos. O *planejamento com redes hierárquicas de tarefas (HTN)* aparece como principal solução para usar conhecimento do domínio de modo a acelerar o processo de planejamento.

Quando um especialista de uma área deseja planejar algo, ele muitas vezes possui "receitas" ou métodos de como resolver problemas parciais, não precisando ponderar a cada passo sobre todas as ações disponíveis a serem incorporadas ao plano. O especialista está interessado em como ele irá cumprir certas tarefas utilizando os métodos mais adequados a cada situação. O planejamento HTN fornece o formalismo necessário para modelar este tipo de abordagem, onde já se conhece métodos de realizar tarefas e se deseja cumpri-las. Nesse formalismo, tarefas complexas podem ser decompostas em subtarefas segundo variados *métodos*, típicos do domínio de planejamento, formando assim hierarquias de tarefas, cujas folhas correspondem a instâncias de operadores diretamente executáveis. De fato, esse tipo de planejamento é o mais utilizado e bem sucedido em aplicações reais. Provavelmente, isto se deve ao fato de que "métodos HTN provêm um meio conveniente de escrever receitas para resolver problemas que correspondem à maneira pela qual um ser humano especialista em um domínio poderia ponderar sobre como resolver um problema de planejamento" (GHALLAB *et al*., 2004, p. 229, tradução nossa).

Uma das formas mais aplicadas de planejamento HTN é o planejamento feito com *Redes Simples de Tarefas* (STN - *Simple Task Network*). O planejamento STN mantém as definições dadas pelo planejamento clássico de ação, operador, plano e função de transição de estados. Os conceitos novos são os de *tarefa*, *rede de tarefas* e *método*.

Uma tarefa pode ser *primitiva* ou *não primitiva*. Uma tarefa primitiva é uma expressão do tipo $t(R_1,...,R_k)$, onde t é um símbolo de tarefa e $R_1,...,R_k$ são termos. Um símbolo de tarefa primitiva também é um símbolo de operador, de forma que uma ação a cumpre uma tarefa t em um estado s se $nome(a) = t(R_1,...,R_k)$ e a é aplicável em s.

Uma rede simples de tarefas, ou STN, é um dígrafo acíclico onde os nós representam tarefas e as arestas, as restrições de ordem parcial entre tarefas. Finalmente, um método é uma quádrupla $m = (nome(m),tarefa(m),precond(m),rede(m))$ onde *nome(m)* é uma expressão do tipo $n(X_1,...,X_k)$ onde *n* é um símbolo de método único e $X_1,...,X_k$ são símbolos de variáveis presentes em *m*, *tarefa(m)* é uma tarefa não primitiva representando a tarefa que o método cumpre, *precond(m)* é um conjunto de literais representando as precondições do método e, por fim, *rede(m)* é uma rede de tarefas (STN) cujas tarefas são denominadas *subtarefas* de *m* e estão parcialmente ordenadas em *rede(m)*.

A escolha de um método *m* para decompor uma tarefa *t* associada a um nó *u* sem predecessor em uma rede de tarefas *w*, sob uma substituição de variáveis σ (que unifica as variáveis de *m* e *t*), leva a uma nova rede de tarefas *w'* gerada pela decomposição $\delta(w,u,m,\sigma)$. Essa decomposição visa remover *u* e todas as arestas que contém *u* da rede tarefas *w*, e adicionar as subtarefas de *m* a *w* mantendo as restrições de ordem que havia sobre *u*, só que agora relativas às subtarefas de *m*.

O processo de planejamento STN difere do planejamento clássico no sentido de que, em vez de ter como objetivo um conjunto de literais que devem valer ao final do plano, tem como objetivo uma rede de tarefas a ser decomposta. O planejamento consiste em encontrar uma sequência de ações que cumpram as tarefas em uma das ordens delimitas pelas restrições de ordem parcial. Isto é feito através da decomposição sucessiva de tarefas não primitivas através de métodos aplicáveis até que se chegue a tarefas primitivas que podem ser postas em sequência para formar um plano.

O planejamento PFD (*Partial-Order Forward Decomposition*) (GHALLAB *et al.*, 2004) é uma das alternativas mais comuns para o planejamento STN. Ele recebe esse nome porque trabalha com redes de tarefas parcialmente ordenadas e por fazer a decomposição a partir do estado inicial em direção ao estado final. A Figura 2 a seguir mostra o algoritmo.

```
PFD(s,w,O,M)
    se w=0
        retorne o plano vazio
    não deterministicamente escolha qualquer u ∈ w tal que u não tenha predecessores em w
    se t_u é uma tarefa primitiva
        ativos ← {(a,σ) |  a é uma instância ground de um operador em O,
                            σ é uma substituição tal que nome(a) = σ(t_u), e
                            a é aplicável em s}
        se ativos = ∅
            retorne falha
        não deterministicamente escolha qualquer (a,σ) ∈ ativos
        π ← PFD(γ(s,a),σ(w – {u}),O,M)
        se π = falha
            retorne falha
        senão
            retorne a.π
    senão
        ativos ← {(m,σ) | m é uma instância ground de um método em M,
                          σ é uma substituição tal que nome(m) = σ(t_u), e
                          m é aplicável em s}
        se ativos = ∅
            retorne falha
        não deterministicamente escolha qualquer par (m,σ) ∈ ativos
        não deterministicamente escolha qualquer STN w' ∈ δ(w,u,m,σ)
        retorne PFD(s,w',O,M)
```

Figura 2. Planejador PFD (GHALLAB *et al.*, 2004, p. 243, tradução nossa).

O algoritmo apresentado na Figura 2 acima consta de dois blocos principais. Primeiro, é necessário checar se a rede de tarefas está vazia (acabaram as tarefas a cumprir), em cujo caso, o plano vazio é retornado. Segundo, uma tarefa sem predecessor é escolhida como próxima tarefa a ser cumprida. Caso a tarefa seja primitiva, é necessário escolher uma ação para cumpri-la e continuar a planejar recursivamente para o resto das tarefas. Caso a tarefa não seja primitiva é necessário escolher um método para decompô-la e gerar uma nova rede de tarefas resultante desta decomposição da tarefa pelo método. Essa nova rede de tarefas, então, é utilizada na chamada recursiva para o planejador.

Há outras formas mais gerais de planejamento HTN, onde as redes de tarefas correspondem a pares *(U,C)*, onde *U* é um conjunto de tarefas e *C* corresponde a restrições sobre essas tarefas mais gerais do que as restrições de ordem entre as tarefas, como no planejamento STN. Apesar de menos geral do que outros tipos de planejamento HTN, o planejamento STN permite um planejamento bastante eficiente, como demonstrado no planejador SHOP2 (NAU *et al.*, 2003) que utiliza o algoritmo PFD descrito.

2.1.3 Planejamento Não Determinístico

A incorporação do não determinismo é um dos relaxamentos mais necessários do planejamento clássico para modelar problemas do mundo real. Ao considerar o não determinismo, a função de transição de estados passa a levar cada par *(estado,ação)* a um conjunto de estados possíveis e não somente a um estado único. Esse relaxamento leva a um aumento considerável na complexidade do processo de planejamento já que agora, para cada ação, é necessário olhar mais de uma continuação possível após sua execução.

Quando se fala em não determinismo, talvez se pense em ações incertas, como o icônico exemplo do rolar de um dado que contém probabilidades associadas a cada transição para um dos estados possíveis. De fato, essa é uma das duas principais abordagens para lidar com não determinismo e recebe o nome de planejamento baseado em processos de decisão de Markov (GHALLAB *et al*, 2004). Este tipo de planejamento, além de probabilidades associadas a cada efeito possível de uma ação, também permite modelar o custo de cada ação e a recompensa por atingir um ou outro estado objetivo. Assim, o problema de planejamento se torna um problema de maximizar recompensas e mitigar custos.

A segunda abordagem principal é o planejamento baseado na verificação de modelos (MCMILLAN, 1993). Neste tipo de planejamento, não há probabilidades associadas e cada efeito de uma ação e nem recompensas ou custos. A principal característica é a possibilidade de expressar metas que devem valer durante a execução do plano. Por exemplo, pode-se imaginar como meta a necessidade de um robô manter seu nível de bateria acima de um limite aceitável. Esse tipo de exigência sobre as metas também é um relaxamento do planejamento clássico onde tínhamos como meta somente estados objetivo a atingir. O nome que se dá para esse relaxamento é a incorporação de objetivos estendidos.

Até agora um plano foi definido como uma sequência ou ordem parcial de ações, mas o planejamento baseado em verificação de modelos (e também o baseado em processos de decisão de Markov) apresenta uma nova ideia de plano que é definido como uma política. O resultado final do processo de planejamento gera uma política, que é um conjunto de associações que mapeiam qual ação tomar dado o estado do mundo em que se está. Isto permite obter comportamentos que se mantêm iterativamente em certo ciclo ou ainda comportamentos movidos por condições em cima do estado do mundo.

Além dessa flexibilidade com a geração de políticas, o planejamento baseado em verificação de modelos permite que se utilizem técnicas para compactar a representação dos estados do mundo, aumentando assim o tamanho do problema com o qual se pode lidar (CIMATTI *et al*., 2003). A ideia da compactação é que seja possível ter fórmulas lógicas que representem diversos estados ao mesmo tempo. Uma técnica bem estabelecida é a utilização da estrutura de dados chamada *Binary Decision Diagram* (BDD) (BRYANT 1992), que representa uma função *booleana* de maneira compacta, podendo assim representar vários estados ao mesmo tempo. Inicialmente a BDD foi criada como estrutura de dados para a lógica proposicional, mas já foi feita extensão que permite a utilização de lógica de primeira ordem (GROOTE e TVERETINA, 2003). As BDDs são amplamente estudadas e encontram diversas aplicações que podem ser encontradas em (WEGENER, 2004). Em (CIMATTI *et al*., 2003), é apresentado um modelo de sistema para domínios com não determinismo chamado *Model Based Planner* (MBP). O modelo fornece a possibilidade de validação e simulação de planos e maneiras de lidar com grandes espaços de estados como é esperado de planejamento que utiliza estratégias de compactação.

O planejamento HTN não determinístico busca unir o não determinismo com a eficiência obtida com o uso de HTN.O planejador que inicialmente se destacou na área foi o ND-SHOP2 (KUTER e NAU, 2004), que é uma adaptação do algoritmo SHOP2 para torná-lo capaz de lidar com o não determinismo. O algoritmo SHOP2 recebeu em 2002 um prêmio de desempenho na competição internacional de planejamento *International Planning Competition*. Em (KUTER e NAU, 2004), os autores comparam a execução do algoritmo ND-SHOP2 com a do algoritmo MBP em dois domínios diferentes. O algoritmo MBP obteve um tempo exponencial em relação ao tamanho do problema enquanto o NDSHOP2 se manteve em tempo polinomial. Além disso, uma análise de complexidade confirma o tempo polinomial. O algoritmo YoYo (KUTER *et al*., 2009) é uma combinação do ND-SHOP2 com o MBP. O YoYo utiliza HTN para reduzir o espaço de busca, BDD para representar conjuntos e transições de estados, e tem sua corretude e completude provadas. Estudos experimentais mostraram que o YoYo é capaz de lidar com problemas em escalas proibitivas para o MBP e o ND-SHOP2 e, além disso, que também é capaz de resolver problemas de ordem cem a mil vezes maior do que ambos os algoritmos. A análise experimental com os três algoritmos ocorreu em dois domínios de forma que um favorecesse a abordagem de verificação de modelos do MBP e o outro, a abordagem HTN do ND-SHOP2. Em ambos os domínios,

o YoYo manteve seu tempo de planejamento quase sempre abaixo dos outros dois algoritmos e nunca acima de ambos para um mesmo problema. YoYo e ND-SHOP2 são bem mais eficientes que o MBP, porém não dão suporte a planejamento com objetivos estendidos.

2.1.4 Planejamento Temporal

As técnicas de planejamento apresentadas nas seções anteriores não falam de tempo em suas formalizações. Por exemplo, os estados do mundo não têm um momento que começam ou terminam; os operadores não têm uma duração; e não há como definir relações ou restrições temporais entre eles. Não é possível raciocinar temporalmente no encadeamento de ações para atingir objetivos e estes não podem falar do tempo. Portanto, pode-se dizer que o tempo sendo considerado é implícito.

O raciocínio temporal explícito é essencial para aproximar a modelagem de um problema da realidade. As aplicações de planejamento temporal são de suma importância para as operações tecnológicas da época atual. Como exemplo, pode-se citar as pesquisas nas áreas de:

- Robótica Espacial: pesquisas da NASA (*National Aeronautics and Space Administration*) para controle de robôs enviados a Marte (AI-CHANG *et al*., 2004)
- Exploração Submarina: pesquisa conjunta do MIT (*Massachusetts Institute of Technology*) com o MBARI (*Monterey Bay Aquarium Research Institute*) para controle de veículos submarinos autônomos (LI e WILLIAMS, 2008)
- Transporte Marítimo de Cargas: pesquisa de minimização de custos para as atividades de transporte de carga entre portos (TIERNEY *et al*., 2012)

O tempo é comumente modelado matematicamente através de um conjunto associado a uma operação de ordenação assimétrica e transitiva. Por exemplo, a escolha do conjunto $\mathbb{R}$ dos números reais permite tempo contínuo e infinito, e uma operação de ordenação parcial, tempo ramificado. Um instante normalmente é modelado como um elemento de $\mathbb{R}$, e um intervalo, como um par $(x,y) \in \mathbb{R}^2$, onde $x1 \leq x2$. Deste modo, de acordo com nossa escolha de modelagem, pode-se ter tempo discreto ou contínuo, finito ou infinito, linear ou ramificado. Uma explicação detalhada, com referências aos diversos tipos de modelagem matemática do tempo, pode ser encontrada em (GHALLAB *et al*, 2004).

Há dois conceitos importantes em planejamento temporal: o conceito de referências temporais e de relações temporais. O primeiro conceito diz respeito a instantes ou intervalos, e o segundo, a relações do tipo *antes*, *durante*, *sobreposto* e relações numéricas em geral. É comum a utilização de uma base de dados temporais, que armazena referências temporais e permite a inserção, consulta e checagem por consistência de relações temporais. Durante o planejamento, o planejador tenta inserir uma nova relação entre as referências temporais.

A pesquisa na área de planejamento temporal gerou diversos tipos de planejadores, tais como: de ordem parcial (PENBERTHY e WELD, 1994) e (YOUNES e SIMMONS, 2003); baseado em lógica (ALLEN, 1991); por busca no espaço de estados (DO e KAMBHAMPATI, 2003); de abordagem híbrida, combinando diferentes vantagens (CUSHING, 2012) e (COLES *et al.*, 2009). Os planejadores temporais iniciais buscavam resolver problemas temporais bastante complexos e sofriam problemas de desempenho, mas os planejadores atuais ultrapassaram alguns dos problemas de complexidade (CUSHING, 2012) e (COLES *et al.*, 2009). Para questões de análise de expressividade e discussão de complexidade, em (CUSHING, 2012), há uma extensa comparação entre planejadores clássicos e os diferentes tipos de planejadores temporais.

2.2 Lógica Temporal Modal

Lógicas temporais modais são formalismos com os quais é possível falar logicamente sobre o tempo. Elas são particularmente interessantes para sistemas de transição de estado onde há a necessidade de se raciocinar sobre estados ao longo do tempo. Além disso, a utilização de um formalismo logicamente correto pode ser necessária para sistemas críticos onde se queira ter a garantia de não invalidar restrições temporais. Uma abordagem frequente para raciocinar com lógica temporal modal é o uso de métodos de verificação de modelos (MCMILLAN, 1993). Há diversas propostas de lógicas temporais modais que variam em termos da expressividade (e.g. proposicional ou de primeira ordem) e da complexidade do tipo de representação do tempo (e.g. tempo linear ou ramificado, tempo discreto ou contínuo, tempo considerado em intervalos ou em pontos etc.) (VENEMA, 2001).

2.2.1 Computation Tree Logic

Computation Tree Logic (CTL) é uma lógica temporal modal que permite expressar fórmulas de logica proposicional quantificadas em tempo ramificado e discreto (HUTH e RYAN, 2010). CTL utiliza quantificadores de caminho para o tempo ramificado e operadores de estado para os estados ao longo de um caminho. Além disso, CTL permite a utilização dos conectivos lógicos usuais: ($\neg$,$\rightarrow$, $\wedge$,$\vee$ e $\leftrightarrow$).

Há dois quantificadores de caminho:

- ***A*** (*All*), para uma quantificação "universal": "em todos os caminhos";
- ***E*** (*Exists*), para uma quantificação "existencial": "em pelo menos um caminho.

Há quatro operadores de estado:

- ***G*** (*Globally)*, para operar "universalmente": a proposição deve valer "em todos os estados";
- ***F*** (*Finally*), para operar "existencialmente": a proposição deve valer "em pelo menos um estado";
- ***U*** (*Until*), para operar "condicionalmente": opera sobre duas proposições e serve para indicar que "a proposição antecessora é necessariamente válida até que a proposição sucessora o seja";
    - Por vezes se distingue o conceito desse operador condicional em dois tipos distintos:
        - $\rho\ U\ \mu$ (*strong until*): A proposição ρ deve valer em todos os estados até que μ valha e, além disso, μ deve valer em algum estado.
        - $\rho\ W\ \mu$ (*weak until*): A proposição ρ deve valer em todos os estados do caminho até que μ valha, mas μ não precisa valer. Ou seja, somente, e sempre que, μ é válida em algum estado, ρ deve valer em todos os estados anteriores daquele caminho.
- ***X*** (*neXt*), para operar "sobre o próximo estado": a proposição deve valer "no próximo estado".

Na definição de CTL, quantificadores de caminho e operadores de estado devem sempre aparecer em pares nas fórmulas as quais se referem, com os primeiros antecedendo aos segundos (e.g. $AG\ \rho$, $AF\ \rho$, $E\ \rho\ U\ \mu$, $EX\ \rho$ etc.). Os quantificadores e

operadores podem ser aninhados para criar fórmulas mais complexas (e.g. *EF (ρ ∧ □G μ)*) para significar que "há pelo menos um estado no futuro onde a proposição *ρ* é válida e, a partir daquele instante, a proposição μ é válida em todos os estados".

A CTL lida com lógica proposicional e tempo discreto e não é definida para lidar com fórmulas de lógica de primeira ordem ou com tempo contínuo. Há extensões de CTL que generalizam essa definição para aumentar o poder de expressão e se aproximar da modelagem de certo domínio.

2.2.2 Uma lógica temporal para *Storytelling* Interativo

Na seção anterior foi apresentada uma definição resumida de CTL, uma lógica temporal modal para tempo discreto e quantificação de fórmulas proposicionais. Todavia, certos domínios exigem a modelagem de variação contínua, que não pode ser capturada pela expressividade de uma lógica proposicional. Além disso, a variação contínua leva a um conjunto infinito de estados possíveis, aumentando consideravelmente a complexidade do problema de verificação.

Dentro do domínio de *Storytelling* Interativo, foi proposta uma lógica temporal modal em (ARAÚJO, 2011) para expressar restrições sobre valores de emoções que variam continuamente em tempo contínuo, ramificado e finito. ARAÚJO (2011) argumenta que uma lógica para tal domínio e modelagem deveria ser capaz de capturar a característica natural contínua das emoções variando ao longo do arco dramático. O tempo finito é uma restrição que modela de maneira adequada a finitude das histórias e diminui a complexidade do tratamento. Além disso, é adicionado um novo operador de estado, *end*, para falar de fórmulas que devem valer no final da história. Este novo operador é particularmente interessante para o domínio de *Storytelling* Inteativo onde é comum a necessidade de exigir certas coisas sobre os finais das histórias. Por exemplo, a exigência de todos os finais serem felizes em um conto de fadas. Finalmente, como o número de estados é infinito, o operador X (que fala de um próximo estado) perde o seu sentido e é retirado da lógica.

A verificação é feita utilizando *Constraint Logic Programming* (CLP) (MARRIOT e STUCKEY, 1998), ou *programação em lógica com restrições*; uma programação declarativa que permite o tratamento de restrições sobre variáveis. Por exemplo, utilizando CLP é possível estabelecer uma restrição $\boldsymbol{F = M \cdot A}$ (neste exemplo, representando uma lei física), tal que as variáveis mencionadas na equação deverão manter-se sob a restrição durante a execução do programa. No caso de

linguagens tradicionais de programação, os outros relacionamentos, implícitos na equação, teriam de estar explícitos no código (i.e. $\boldsymbol{M = F / A}$ e $\boldsymbol{A = F / M}$). Todavia, com a utilização de CLP, a adição de uma restrição $\boldsymbol{F = M \cdot A}$ leva à garantia dos outros relacionamentos algébricos implícitos. Ou seja, no exemplo mencionado, a atribuição de valores para quaisquer duas das variáveis garante que se encontre o valor da terceira. A técnica CLP é muito útil em problemas complexos, onde há muitas equações e variáveis envolvidas e deseja-se manter a execução correta e eficiente (e.g. Engenharia). Diversas linguagens para programação em lógica, como Prolog, possuem um *constraint solver*, que é um módulo responsável pelo gerenciamento e verificação das restrições adicionadas e removidas. Os *constraint solvers* disponíveis são normalmente capazes de lidar com números reais e racionais, mas possuem, em geral, apenas a capacidade de lidar com equações e inequações lineares.

A gramática abaixo define, recursivamente, uma *fórmula bem formada* na lógica. Nela, *a* e *b* representam combinações lineares de propriedades contínuas e α e β representam fórmulas de lógica temporal modal.

$$\alpha, \beta \rightarrow basic_constr \mid AG\ \alpha \mid AF\ \alpha \mid EG\ \alpha \mid EF\ \alpha \mid A\ \alpha\ U\ \beta \mid \\ E\ \alpha\ U\ \beta \mid A\ end\ \alpha \mid E\ end\ \alpha \mid \neg\alpha \mid \alpha \rightarrow \beta \mid \alpha\ and\ \beta \mid \alpha\ or\ \beta \mid \quad (1)$$

$$basic_constr \rightarrow a = b \mid a \neq b \mid a > b \mid a < b \mid a \geq b \mid b \leq a \quad (2)$$

A semântica para as fórmulas é semelhante à semântica usual em lógica clássica com a adição da semântica para CTL. Conectivos lógicos e quantificadores de caminho seguem a mesma definição usual, mas operadores de estado têm sua semântica adaptada para considerar intervalos de tempo e sequência infinita de estados. O operador de estado *end* é avaliado considerando que os valores das propriedades permanecem inalterados a partir do final de uma folha em diante. Essa definição permite que se raciocine sobre fórmulas que possuam fórmulas complexas quantificadas pelo operador *end*, isto é, fórmulas que exigem que algo valha em um estado futuro em relação ao último estado.

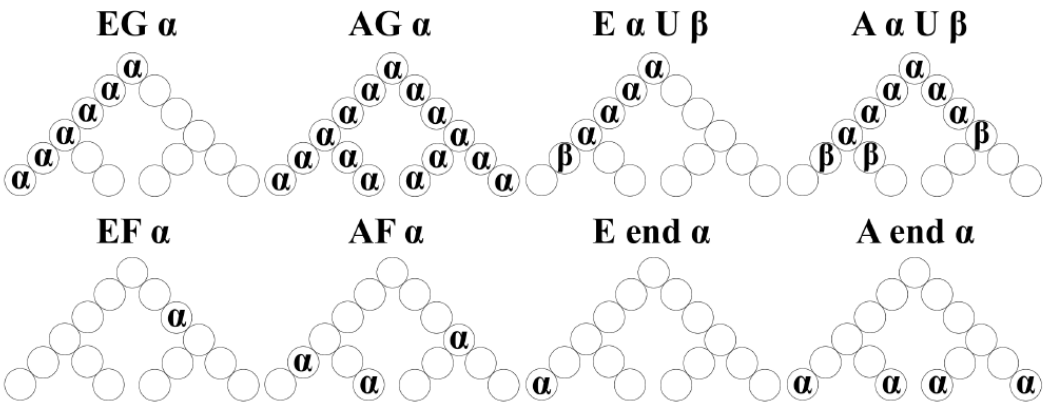

Figura 3. Semântica da lógica temporal

Além de definir a lógica temporal modal, ARAÚJO (2011) também apresenta algumas simplificações possíveis de fórmulas. Foram estudadas 64 simplificações dentre as quais 48 foram simplificadas. Para fórmulas complexas quantificadas pelo operador de estado *end*, não há necessidade de simplificação visto que basta verificar se as variáveis presentes nessas fórmulas complexas valem no estado final.

FÓRMULAS			
ORIGINAIS	EQUIVALENTES	ORIGINAIS	EQUIVALENTES
1. EF EG α	E end α	33. EG EF α	E end α
2. EF AG α	E end α	34. EG AF α	–
3. EF AF α	–	35. EG AG α	AG α
4. EF EF α	EF α	36. EG EG α	EG α
5. EF (E end α)	E end α	37. EG (E end α)	E end α
6. EF (A end α)	E end α	38. EG (A end α)	A end α
7. EF (E (α U β))	–	39. EG (E (α U β))	–
8. EF (A (α U β))	–	40. EG (A (α U β))	–
9. AF EG α	–	41. AG EF α	A end α
10. AF AG α	A end α	42. AG AF α	A end α
11. AF EF α	EF α	43. AG EG α	AG α
12. AF AF α	AF α	44. AG AG α	AG α
13. AF (E end α)	E end α	45. AG (E end α)	A end α
14. AF (A end α)	A end α	46. AG (A end α)	A end α
15. AF (E (α U β))	–	47. AG (E (α U β))	–
16. AF (A (α U β))	–	48. AG (A (α U β))	–
17. E end (EF α)	E end α	49. A end (EF α)	A end α
18. E end (AF α)	E end α	50. A end (AF α)	A end α
19. E end (EG α)	E end α	51. A end (EG α)	A end α
20. E end (AG α)	E end α	52. A end (AG α)	A end α
21. E end (E (α U β))	E end (α∧β)	53. A end (E (α U β))	A end (α∧β)
22. E end (A (α U β))	E end (α∧β)	54. A end (A (α U β))	A end (α∧β)
23. E end (E end (α))	E end α	55. A end (E end (α))	A end α
24. E end (A end (α))	E end α	56. A end (A end (α))	A end α
25. E (EF α U β)	EF (β ∧ EF α)	57. A (EF α U β)	AF (β ∧ EF α)
26. E (AF α U β)	EF (β ∧ AF α) ∧ AF α	58. A (AF α U β)	AF (β ∧ AF α)
27. E (EG α U β)	EG (α ∧ EF β)	59. A (EG α U β)	–
28. E (AG α U β)	(AG α) ∧ (EF β)	60. A (AG α U β)	(AG α) ∧ (AF β)
29. E (E end α U β)	EF (β ∧ E end α)	61. A (E end α U β)	AF (β ∧ E end α)
30. E (A end α U β)	(EF β) ∧ (A end α)	62. A (A end α U β)	AF (β ∧ A end α)
31. E (E (α U β) u θ)	–	63. A (E (α U β) u θ)	–
32. E (A (α U β) u θ)	–	64. A (A (α U β) u θ)	–

Figura 4. Simplificações de fórmulas da lógica temporal

2.3 Storytelling Interativo

Há diversas definições para Storytelling Interativo (SI), que surgem de acordo com a ênfase que se dê em cada uma das áreas nas quais se baseia ou nas quais é aplicado. De maneira geral, quando se fala em Storytelling Interativo, há concordância de que deve haver uma história sendo contada e que um ou mais espectadores devem influenciar no desenrolar da narrativa.

Um exemplo clássico da área, e que ajuda a clarificar essa definição, é o sistema de S*torytelling* Interativo (SSI) Façade (MATEAS e STERN, 2005). Nele, você é convidado para jantar no apartamento de um casal que aparenta estar vivendo uma época de conflitos na relação. A narrativa é vista por uma interface gráfica 3D ou realidade aumentada (DOW *et al*., 2008). É possível conversar através de uma interface de texto com o marido ou a esposa, e, através do mouse, interagir com alguns dos objetos de cena. As interações influenciam diretamente o desenrolar da narrativa, podendo levar desde a um jantar agradável até à expulsão da casa de seus amigos.

Figura 5. Sistema de *Storytelling* Interativo Façade.

2.3.1 Abordagens Character-based e plot-based

Duas principais abordagens se contrastam em SI: *character-based* e *plot-based*. Na primeira, a narrativa é construída através de personagens, seus objetivos e suas interações uns com os outros e com o ambiente ou com o usuário. A abordagem *character-based* (PEINADO *et al*., 2008 e PIZZI, 2011,) é inspirada em jogos eletrônicos e, como se baseia em interações diversas entre agentes, ambientes e o usuário, dá grande liberdade a este, mas dificulta em muito a garantia de coerência do enredo. A segunda abordagem, *plot-based* (GRASBON *et al*, 2001, CIARLINI *et al*, 2010), influenciada pela Narratologia, centra seus esforços na geração de uma história com um enredo coerente, embora com isso dificulte as possibilidades de interação com

o usuário, que poderiam ferir a coerência. A abordagem *plot-based* tem a sua influência inicial no trabalho seminal de PROPP (1968), que fez um extenso estudo da literatura de contos de fadas russos, encontrando um conjunto de 31 funções básicas que ocorrem de forma semelhante em diversas histórias (CIARLINI *et al*., 2005).

As duas abordagens não são, todavia, excludentes, e há algumas tentativas de juntá-las em abordagens híbridas. O sistema Façade (MATEAS e STERN, 2005), que trouxe muita visibilidade à área de SI, adota tal abordagem híbrida. No Façade, os personagens são agentes autônomos com seus objetivos, mas um gerenciador de história (*Drama manager*) garante que a trama se desenrole através de mecanismos para que objetivos de mais alto nível sejam atingidos. Outro exemplo é o S-MADE (CAI *et al*., 2007), onde há distinção entre o agente diretor e os personagens cada um tendo seus objetivos. Estes são modelados utilizando a ferramenta FCGN (*Fuzzy Cognitive Goal Net*), que diferencia o comportamento dos personagens dos planos desejados pelo diretor. A abordagem dessa pesquisa encontra-se próxima à feita no LOGTELL, ou seja, uma abordagem mista, mas com ênfase no *plot-based* na medida em que busca conformidade a arcos dramáticos típicos de um gênero.

2.3.2 Planejamento para Geração de Histórias

2.3.2.1 Visão Geral

A geração de histórias em SI é, em grande parte, baseada em planejamento (BARROS e MUSSE, 2007) e (PORTEOUS *et al*., 2010). Isto ocorre pela proximidade do conceito de um plano (como uma sequência de ações) e de uma história (como uma sequência de eventos). Além disso, o uso de planejamento visa garantir, ou aumentar as chances, de gerar uma história coerente. Essa ideia do plano equiparado a uma história parece mais inclinada à abordagem plot-based, mas abordagens character-based também se utilizam de planejamento para, por exemplo, garantir que agentes autônomos atinjam seus objetivos (PIZZI, 2011).

2.3.2.2 Ordem Parcial e HTN

A abordagem com algoritmos de planejamento centra-se principalmente em planejamento de ordem parcial e planejamento HTN. Como exemplo de uso do primeiro tipo de planejamento, temos (WARE e YOUNG, 2011), que busca gerar histórias considerando conflitos através do algoritmo CPOCL (*Conflict Partial-Order Causal Link*) (WARE e YOUNG, 2011). Os conflitos acontecem quando há vínculos causais ameaçados entre eventos do plano. Estes vínculos são gerenciados e a corretude do planejamento é mantida pela anotação de certos passos como não executados. Como

exemplo de utilização de planejamento HTN, tem-se (SILVA, 2010), que é o trabalho no qual essa pesquisa se baseia. Além deste trabalho, há a pesquisa em (HOANG *et al*., 2005), onde os autores utilizam HTNs para modelar estratégias da Inteligência Artificial de jogos. Também há a pesquisa em (KELLY *et al*., 2008), onde HTNs são utilizadas para modelar planos de ação de NPCs (*non playing characters*), que são personagens não controlados por jogadores. Além destas pesquisas, em (GÖBEL *et al*, 2003), os autores apresentam a utilização de HTN para representar ações típicas de personagens. Deste modo, dentre as abordagens com planejamento HTN, o trabalho de SILVA (2010), mesmo considerando padrões típicos de ação de personagens, é o mais próximo de uma abordagem *plot-based* no espectro que vai de uma abordagem puramente *character-based* a uma puramente *plot-based*.

O planejamento de ordem parcial tem a vantagem de gerar histórias que não precisam ser ordens totais de eventos e, portanto, têm mais flexibilidade para conciliar diferentes objetivos. As abordagens com planejamento HTN buscam eficiência quando há a possibilidade de inserir conhecimento do domínio. A limitação do planejamento HTN é justamente a exigência de conhecimento prévio de uma hierarquia de soluções típicas para a execução de tarefas do domínio. Em (SILVA *et al*., 2010), os autores apresentam uma abordagem híbrida que combina as vantagens de ambos os tipos de planejamento (ordem parcial e HTN) em um contexto de planejamento não determinístico.

2.3.2.3 Temporal

A modelagem do tempo na geração de histórias comumente faz uso de tempo implícito, isto é, a duração dos eventos não é considerada. O uso de planejamento temporal para modelar o tempo explicitamente ainda é incipiente na área de S*torytelling* Interativo. A maioria das pesquisas que lidam com o tempo se centraram em lidar com o tempo após a geração de uma história. Dentre as abordagens, há os exemplos de: controle na dramatização para que os efeitos dos eventos só passem a valer após a execução destes (VERNIERI, 2006); geração políticas para dramatização mais lenta ou mais rápida de eventos (DÓRIA *et al*., 2008); e verificação de histórias geradas para checar se estas se conformam a fórmulas temporais que falam da variação contínua de emoções ao longo de tempo contínuo (ARAÚJO, 2011). Em (CIARLINI *et al*, 2010), é apresentado a formalização do sistema LOGTELL, que utiliza uma lógica temporal modal para a inferência de objetivos a atingir pelo planejador e incorporação de sugestões de objetivos dadas pelo usuário para o futuro e para o passado da narrativa.

Entretanto, o processo de planejamento não entra no mérito da duração de transições ou variação de propriedades dentro do evento.

A única abordagem, além da apresentada nessa pesquisa, que lida com planejamento temporal durante a geração, é a apresentada em (PORTEOUS *et al*., 2011). Nesta abordagem, os autores utilizam o planejador temporal de ordem parcial CRIKEY (COLES *et al*., 2009) juntamente a uma estratégia de otimização que divide a narrativa em subpartes a serem planejadas. Nesta abordagem, um autor cria previamente restrições que definem uma ordem parcial temporal entre predicados do gênero. Essa informação é mantida separada da especificação do universo da narrativa e seus personagens. As restrições permitidas são: *sometime-before* e *sometime*. Se um predicado pode ocorrer em qualquer ponto da narrativa, ele é representado utilizando o operador *sometime*. Caso contrário, se há uma relação temporal necessária, é utilizado o operador *sometime-before*, que um predicado deve ocorrer em algum ponto anterior a outro. As restrições temporais representam situações-chave da narrativa e são utilizadas como objetivos pelo planejador temporal. O processo de geração de uma narrativa, então, consiste em: utilizar um conjunto de restrições temporais para dividir a narrativa em subproblemas a atingir; planejar cada subproblema com o CRICKEY; e compor as subnarrativas geradas em uma narrativa única e em ordem total. De fato, conforme o planejador CRIKEY resolve cada subproblema, as ações são enviadas a um módulo de visualização, que, por sua vez, faz proveito das especificações temporais das ações para dramatizá-las. Os resultados da abordagem são discutidos em termos do considerável aumento na diversidade das histórias geradas com a adição do tempo explícito. Quanto ao tempo de geração, há outro artigo sobre a mesma abordagem (PORTEOUS *et al*., 2010) que apresenta tempos médios com um teto de *1.5 s* para um universo narrativo com pouco mais de 40 eventos. Os tempos foram medidos em uma máquina com 2.26 GHZ e 4GB de memória RAM.

2.3.3 Controle Dramático da Narrativa

O controle dramático em *Storytelling* Interativo consiste na capacidade de se obter garantias de atendimento a requisitos estéticos definidos *a priori* por um ou mais autores, considerando a interação com o usuário. Em outras palavras, se deseja certo nível de controle sobre a dramaticidade da narrativa a ser apresentada aos usuários de acordo com suas interações, mas levando em conta os requisitos estéticos.

O interesse em controlar a dramaticidade da narrativa se inspira na teoria narratológica (BRANIGAN, 1992). As abordagens se centram mais na geração de suspense. A proposta dessa pesquisa se inspira na satisfação de arcos dramáticos ligados a emoções em geral e não particularmente ao suspense. Diversas abordagens apresentam foco no arco dramático, seja no processo autoral, (CATALDI *et al*, 2011) e (MURRAY *et al*., 2011), ou na verificação após a geração, (MAGERKO, 2007) e (ARAÚJO, 2011).

As principais abordagens que incorporam o controle do arco dramático ao processo de geração são: (BARROS e MUSSE, 2008), (CHEONG, 2007) e (BARBER e KUDENKO, 2007). Essas abordagens fazem uso explícito da definição do arco dramático, onde este é definido por uma função, ou por algum valor mensurável que se pode comparar e que muda ao longo da narrativa.

Em (BARROS e MUSSE, 2008), os autores modelam eventos para poderem revelar pistas sobre a narrativa que se desenrola conforme o usuário escolhe uma ação para o protagonista. Antes da geração da narrativa, um autor deve definir um arco de tensão como uma função do tempo (discretizado). Durante a geração da narrativa, cada evento escolhido aumenta ou diminui a tensão e, conforme o usuário interage, os eventos são escolhidos de acordo com sua adequação ao arco definido.

Em (CHEONG, 2007), os autores apresentam a geração de uma *fábula* (ordem parcial de eventos) através de um planejador de ordem parcial. Após a geração de uma fábula, ocorre uma etapa de composição de um *sjuzhet*[1] (ordem total de eventos) com alto nível de suspense. Os autores definem heurísticas para se encontrar um alto nível de suspense. Uma dessas heurísticas é, por exemplo, comparar o número de possiblidades de fracasso e sucesso para o protagonista atingir seus objetivos. A ideia é que quanto maior a dificuldade do protagonista, maior será o suspense. Para entender isso, basta lembrar-se de as cenas onde o James Bond se encontra completamente sem saída, enquanto o vilão lentamente conta o seu plano.

Em (BARBER e KUDENKO, 2007), os autores apresentam um processo de geração de narrativa que se baseia em modelo de usuário e na definição de dilemas típicos para os personagens que conduzem a dramaticidade da história. Exemplos de dilemas típicos incluem: o *Betrayal*, onde um personagem deve decidir por uma ação que lhe causará um bem enorme, mas um terrível mal a um amigo; e o *Sacrifice*, onde

[1] O termo *sjuzhet* tem sua origem com o formalismo Russo no estudo da estrutura narrativa (PROPP, 2003). Pode-se entender a *fábula* como o agregado de eventos em sua relação lógica, causal e cronológica, e o *sjuzhet*, como sendo os eventos na forma em que estão descritos na mídia que os apresenta (texto do livro, cenas do filme etc.) (MEISER, 2003).

um personagem tem a opção de realizar uma ação que lhe causará um mal terrível, mas um bem enorme a um amigo, dentre outros dilemas. Cada dilema é apresentado ao planejador como um objetivo a atingir, e cada plano gerado para atingir o objetivo é uma história. O usuário então interage com essa história e pode interferir em certos eventos fazendo com que eles aconteçam ou não. A coerência é garantida através da geração de diversos outros planos possíveis em paralelo e o usuário é compelido a um leque menor de escolhas.

2.4 Logtell

LOGTELL (CIARLINI *et al*., 2010) é um sistema de *Storytelling* Interativo que utiliza planejamento para gerar histórias e dramatizá-las em um ambiente de animações 3D. Nesse sistema, histórias são geradas capítulo a capítulo e a interação com o usuário pode ocorrer de duas maneiras: o usuário pede a geração de uma versão alternativa para o capítulo que está assistindo ou sugere a ocorrência de eventos e situações para o próximo capítulo. Um capítulo no LOGTELL é uma sequencia de eventos.

O objetivo do LOGTELL é ser capaz de gerar histórias diferentes e coerentes dentro de um mesmo *gênero* literário através de diferentes estágios de simulação e interferência do usuário. A fim de garantir a coerência, há um modelo formal em lógica para capturar as especificações de um gênero. A diversidade das histórias é garantida pelo processo de planejamento, que pode gerar diferentes combinações e encadeamentos de eventos. Além disso, há *regras de inferência de objetivo* que visam expressar *funções narrativas*, no sentido do estudo seminal de Propp (PROPP, 1968), e capturar objetivos de personagens que cumprem certo papel narrativo. Estas regras, expressas em lógica temporal modal, buscam afirmar que, se certas condições existem no momento, então, no futuro, tais outras condições deverão ser encontradas. Por exemplo, uma regra típica do gênero "Espadas & Dragões", inspirado em fantasia medieval, onde pode-se encontrar dragões, magos, princesas e cavaleiros exercendo os papéis de vilões, vítimas e heróis. Neste gênero ocorrem sequestros, lutas, encantamentos e o amor e a ambição movem os interesses dos personagens. Este gênero relaciona os papéis narrativos de *vilão* e *vítima*, seria: "*se a vítima estiver viva, e ainda não estiver raptada, então o vilão tentará raptá-la.*".

O LOGTELL trabalha com uma especificação de um gênero, que compreende a parte formal e a parte necessária à dramatização e controles de cena. Para esclarecer como tal especificação é feita, esta seção apresenta exemplos concretos do gênero "Espadas & Dragões". Além do gênero, é necessário definir um *contexto*, que é uma instanciação do gênero, e especifica os personagens, locais e suas relações existentes no estado inicial. A definição de um gênero é composta de:

- Descrição lógica das entidades, relações e papéis narrativos que podem existir no gênero:
    - Entidades: *criatura*, *pessoa*, *cavaleiro*, *dragão*, *princesa*, *local*, etc.;
    - Herança entre entidades: um *cavaleiro* é uma *pessoa*, uma *pessoa* é uma *criatura*, um *dragão* é uma *criatura* etc.;
    - Relações: A relação *lar* entre uma *criatura* e um *local*, a relação *conhecido* entre uma *criatura* e outra, a relação *casado* entre uma *pessoa* e outra etc.;
    - Atributos: Uma *criatura* tem *natureza*, uma *criatura* pode estar *viva*, um *local* tem uma *proteção*, *conhecidos* tem *afeição* etc.;
    - Tipos de atributos: *Vivo* é do tipo *booleano*, *proteção* é do tipo *composto* e é uma dupla com a *natureza* do lugar e o *nível* de proteção etc.;
    - Papéis narrativos: Um *cavaleiro* pode exercer o papel de *herói*, tanto um *cavaleiro* como uma *princesa* podem exercer o papel de *vítima*, tanto um *cavaleiro* como um *dragão* podem exercer o papel de *vilão* etc.
- Um conjunto de operadores com precondições efeitos;
- Um conjunto de regras de inferência de objetivos;
- Uma biblioteca de planos típicos que contém combinações típicas de operadores para atingir certos objetivos. A biblioteca é organizada de acordo com hierarquias do tipo "*is-a*" e "part-of";
- Um autômato não determinístico associado a cada operador a fim de especificar as formas possíveis de dramatização do evento;
- Modelos gráficos dos locais e atores virtuais 3D;

- Scripts e Máquina de Estados Finitos para controlar a movimentação das câmeras e dos atores no ambiente virtual 3D.

Figura 6. Dramatização do evento *Donate* no sistema LOGTELL.

O LOGTELL é voltado para um ambiente de TV Interativa. Nesse contexto, a diversidade das histórias e a capacidade de planejar rapidamente a sua continuação, levando em conta as interações com o usuário são requisitos para a geração de enredos com base em planejamento. Nesta seção, apresenta-se inicialmente uma descrição geral do LOGTELL, contendo o necessário à compreensão da pesquisa. Maiores detalhes sobre outros aspectos do LOGTELL, como a formalização do conceito de gênero, a interação em ambiente cliente-servidor para TV interativa, e a dramatização devem ser buscados, respectivamente, em (CIARLINI *et al*., 2010), (CIARLINI *et al*., 2008) e (DÓRIA *et al*., 2008). Um detalhamento do LOGTELL pode ser encontrado no Anexo I.

2.5 Modelo e Implementação de Geração de Enredos com Não Determinismo

2.5.1 Visão Geral

O modelo de planejamento proposto nessa dissertação se baseia no modelo proposto em (SILVA, 2010), compatível com o LOGTELL. Essa pesquisa teve o

cuidado de manter o novo modelo totalmente compatível com o modelo de SILVA (2010). Desta forma, para o entendimento completo do modelo que é proposto nessa pesquisa, é necessário explicar o modelo proposto em (SILVA, 2010).

Segundo SILVA (2010), o objetivo principal do modelo de geração de enredos com não determinismo (nesta seção, referenciado simplesmente como modelo) é "aumentar o grau de diversidade dos enredos gerados, incorporando eventos com mais de um conjunto de efeitos possíveis". A ideia geral é que se possa gerar enredos que não sejam simplesmente uma sequência de eventos, mas sim uma *árvore de contingências*, onde os nós são eventos, as arestas representam as diferentes possibilidades não determinísticas e as contingências são as precondições associadas a cada evento, que permitem o enredo seguir por uma ou outra ramificação.

A incorporação de não determinismo no processo de planejamento não pode causar incoerência nos enredos gerados. Para garantir isso, o processo deve considerar todos os possíveis estados após cada evento, gerando ramificações que garantam o cumprimento dos objetivos a atingir em cada caminho do enredo gerado. Além disso, também é interessante que se possam ter objetivos que não necessariamente precisam ser atingidos, mas cuja tentativa de alcançá-los cause carga dramática. Por exemplo, pode haver uma *tentativa* de o vilão sequestrar a vítima. É de se esperar que o planejador tente encontrar maneiras de encadear eventos que, mesmo que não alcancem o sequestro, permaneçam no plano. Assim, o modelo propõe o conceito de *tentativa* no processo de planejamento, que é detalhado abaixo.

A incorporação de não determinismo aumenta o espaço de busca do processo de planejamento e, consequentemente, o tempo de execução. Portanto, para tornar viável tal incorporação, o modelo adota planejamento HTN. Este tipo de planejamento não só aumenta a eficiência como também é adequado ao domínio de *Storytelling* Interativo ao permitir que se definam eventos em um nível mais alto que se decomponham em ordens parciais de eventos em um nível mais baixo (equiparando eventos e suas diferentes decomposições a tarefas, métodos e subtarefas em HTN).

2.5.2 Requisitos

O modelo proposto nessa pesquisa buscou continuar atendendo a cada um dos cinco requisitos levantados por SILVA (2010) e que serviram de guia para a construção do modelo. Os cinco requisitos são:

1. Suporte a eventos não determinísticos: Planejar levando em conta eventos com mais de um final possível sem com isso comprometer a coerência;
2. Controle do nível de não determinismo: Oferecer uma maneira de controlar o número de alternativas encadeadas no plano a fim de tornar o problema de planejamento mais tratável;
3. Incorporação do conceito de tentativas: Oferecer um mecanismo para que se possam ter objetivos no planejamento que não necessariamente precisem ser atingidos e possibilitar que eventos que busquem atingi-los permaneçam no plano como tentativas.
4. Processamento em paralelo das diversas alternativas: Possibilitar que mais de um planejamento possa estar ocorrendo ao mesmo tempo a fim de planejar continuações de uma história;
5. Eficiência na geração de enredos.

2.5.3 Planejamento HTN

SILVA (2010) modelo diversas versões não determinísticas para os operados determinísticos do LOGTELL. Por exemplo, o operador *kidnap(VIL,VIC)*, pode ser substituído por sua versão não determinística *try_to_kidnap(VIC,VIL)*. A versão determinística estabelece que um personagem *VIL* rapta um personagem *VIC*. A versão não determinística tem o mesmo final possível, mas adiciona ao evento outro final onde o vilão mate sem querer a vítima no processo de rapto. Com a versão não determinística, gera-se a possibilidade do planejamento continuar por um caminho onde a vítima é raptada e outro onde ela morre, e, assim, possivelmente gerando narrativas bem diversas. Operadores não determinísticos possuem, então, dois conjuntos:

- Um conjunto de literais representando os efeitos determinísticos;
- Um conjunto de conjuntos de literais representando os diferentes efeitos não determinísticos para cada final.

Adicionalmente à definição de operadores não determinísticos, o modelo de SILVA (2010) também prevê a definição de hierarquia e composição entre operadores. Portanto, os operadores passam a ser divididos em duas classes diferentes: *básicos* e *complexos*. Operadores básicos, correspondentes a tarefas primitivas em HTN, são aqueles associados a eventos dramatizáveis, e podem ser determinísticos ou não determinísticos. Um capítulo enviado para dramatização contém somente eventos básicos. Operadores complexos, correspondentes a tarefas não primitivas em HTN, se

dividem em duas subclasses: *abstratos* e *compostos*. Os operadores abstratos podem ter uma ou mais especializações (também chamadas aqui de filhos), que são outros operadores de qualquer classe, e capturam o conceito de método em HTN. Cada especialização pode ser pensada como um *método* de resolver o operador sendo especializado. Operadores compostos contêm uma sub-HTN (também chamada aqui de suboperadores) em forma de uma ordem parcial de outros operadores de qualquer classe. Operadores abstratos também podem ser compostos. Nesse caso, as suas especializações herdam a HTN do pai e podem adicionar novos operadores e restrições de ordem.

Por fim, os efeitos de operadores complexos só são utilizados na fase de planejamento de ordem parcial. Durante o planejamento HTN, só se considera estados gerados por instâncias de operadores básicos.

Como exemplo, podem-se citar dois operadores do LOGTELL: *fight(CH1,CH2)*, onde um personagem luta com o outro, e *kill(CH1,CH2)*. No novo modelo, estes operadores são básicos e determinísticos e, além disso, pode-se incluir um operador composto *defeat(CH1,CH2)*, que tenha como suboperadores *fight(CH1,CH2)* e *kill(CH1,CH2)* e uma restrição de ordem dizendo que *fight(CH1,CH2)* venha antes de *kill(CH1,CH2)*. Este operador composto *defeat(CH1,CH2)* visa capturar uma visão de mais alto nível onde um personagem irá derrotar o outro lutando com ele e depois matando-o. Além disso, pode-se incluir um novo operador abstrato *combat(CH1,CH2)*, que tenha duas especializações: *defeat(CH1,CH2)* e *fight_to_the_death(CH1,CH2)*. Nesse caso, um combate entre dois personagens pode ser pensado como podendo ocorrer respectivamente de duas maneiras distintas: um derrotando o outro; ou uma luta até a morte onde o primeiro pode morrer, o segundo pode morrer, ou ambos podem morrer, sucumbindo à batalha.

3 Modelo de Geração de Enredos com Não Determinismo e Verificação de Fórmulas Temporais

3.1 Visão Geral

O objetivo principal do modelo apresentado nessa dissertação é fornecer à área de *Storytelling* Interativo uma alternativa eficiente para gerar enredos que levem em conta não determinismo e restrições temporais (arcos dramáticos típicos) sobre a variação de propriedades dramáticas. Desta forma, busca-se que as histórias geradas tenham mais chances de cativar e surpreender os usuários através de um aumento na diversidade e na garantia de cumprimento de arcos dramáticos.

A fim de incorporar o não determinismo enquanto se mantém a eficiência e a variabilidade, o modelo utiliza a estratégia de planejamento híbrido, com um planejador de ordem parcial e um planejador HTN não determinístico. Este, claro, é modificado para tratar a verificação, em tempo de planejamento, das restrições temporais sobre as propriedades dramáticas. A possibilidade de integração entre os dois tipos de planejamento é comprovada pelo trabalho de (SILVA, 2010) e serve de base para o modelo aqui apresentado. Mantêm-se similares as definições de gênero, contexto, operadores e geração capítulo-a-capítulo do LOGTELL (seção 2.4) e do modelo de planejamento HTN não determinístico (seção 2.5).

O gênero passa a possuir um conjunto de fórmulas temporais típicas que torna possível o seu preprocessamento para utilização pelo planejador. Essas fórmulas são expressas através da lógica temporal modal desenvolvida em (ARAÚJO, 2011) e explicada na seção 2.2.2.

O LOGTELL utiliza uma modelagem de eventos tal que, além do operador básico que o representa, cada evento básico tem associado a si um autômato que contém as informações para a dramatização. O LOGTELL gera políticas em cima desses autômatos para criar um mecanismo de controle sobre o tempo de dramatização em caso

de atraso na geração do capítulo ou de haver necessidade de acelerar a dramatização para aumentar o interesse da plateia (DÓRIA, 2009). As políticas de controle do tempo de dramatização não estão diretamente relacionadas ao trabalho desta dissertação, embora sejam compatíveis com ele. Desse modo, a definição de autômato associado a um evento básico foi alterada com foco na inclusão de informações sobre propriedades dramáticas.

Cada evento básico passa a ter associado a si um autômato não determinístico, que contém informações sobre a variação das propriedades dramáticas para cada transição entre os estados e também informações para a dramatização. Assim, obtém-se uma descrição rica para a dramatização e para as diferentes curvas possíveis de variação das propriedades dramáticas. A estrutura de autômato utilizada é detalhada na seção 3.4.2. O modelo é inspirado em geração capítulo-a-capítulo, portanto, os arcos dramáticos típicos têm ser fornecidos para cada capítulo.

O modelo visa lidar com tempo contínuo e variação contínua (de propriedades dramáticas) na composição de uma estrutura não linear (árvore) de eventos complexos (cada um representado por um autômato não determinístico). Sozinha, a mudança contínua leva a um espaço de busca infinito, o que demanda tratamento especial.

Deseja-se incorporar a verificação temporal em tempo de planejamento. Todavia, os autômatos podem conter diversos caminhos que modificam as propriedades dramáticas e que devem ser levados em conta pelo planejador. Para adicionar um evento básico ao plano, precisamos saber se a fórmula temporal é válida em seu autômato, e também qual a fórmula temporal residual para cada final (a fim de continuar o planejamento a partir do final). Desta forma, planejar no nível de granularidade dos estados do autômato leva o processo de planejamento a entrar, potencialmente, no mérito de cada variação temporal em cada possível transição entre os estados de um autômato. Isto aumenta consideravelmente o risco de tornar o planejamento inviável.

Para os algoritmos clássicos de planejamento HTN, incluindo aqui o algoritmo NDet-HTN de SILVA (2010), o tempo para decidir sobre a adição de um evento a um plano envolve somente a verificação da satisfabilidade de precondições no estado corrente do mundo. Seria interessante poder manter um tempo próximo, mesmo com a existência do raciocínio temporal sobre as propriedades dramáticas. Para isso, o modelo prevê uma etapa de preprocessamento a fim de gerar uma estrutura indexada que permite, para cada tripla <*evento, estado final do evento, fórmula temporal de entrada*>, obter um final, e uma fórmula temporal de entrada, obter uma fórmula

temporal residual condicionada a restrições sobre os valores de entrada das propriedades dramáticas. Desta forma, a adição de um evento ao plano passa a ser condicionada também a uma consulta a essa estrutura, que é indexada, e, portanto, não leva a um aumento considerável no tempo.

O processo de planejamento passa a receber uma fórmula temporal típica de entrada que deve ser válida no plano final gerado. Essa fórmula vai sendo modificada através da obtenção das fórmulas residuais conforme se adicionam eventos básicos ao plano. Um evento que não satisfaça a fórmula temporal corrente deve ser descartado, levando o planejamento a voltar a um ponto de escolha onde possa vir a obter um evento diferente para adicionar. Além disso, fórmulas sob o quantificador existencial de caminho só precisam valer em um dos caminhos e, portanto, quando satisfeitas por um destes não precisam ser verificadas em outro. O modelo de planejamento prevê essa eliminação de tais fórmulas quando já satisfeitas por um caminho.

3.2 Requisitos

Todos os cinco requisitos mencionados na seção 2.5.2 e levantados por SILVA (2010) continuam a ser atendidos pelo modelo e sua arquitetura. Estes são os de: suporte a eventos não determinísticos; controle do nível de não determinismo; incorporação do conceito de tentativas; processamento em paralelo das diversas alternativas; e eficiência na geração de enredos. Os cinco requisitos já existentes para o modelo em (SILVA, 2010) continuam a ser atendidos, pois a estrutura geral do planejamento HTN não determinístico foi mantida igual, com a entrada e a saída do planejador compatíveis.

A esses requisitos, adiciona-se um sexto requisito: conformidade a arcos dramáticos típicos de um gênero. Conforme a pesquisa na área de *Storytelling* Interativo aponta (BARROS e MUSSE, 2008), (CHEONG, 2007) e (BARBER e KUDENKO, 2007), há a necessidade de abordar a questão do arco dramático para envolver, surpreender e manter o interesse do usuário. Por exemplo, podem-se querer histórias com finais felizes, com um alto nível de medo, com picos de suspense seguidos de relaxamento etc.

Essa pesquisa assume a definição de *arcos dramáticos típicos* de um gênero, que expressam variações de *propriedades dramáticas* (e.g. *alegria*, *suspense*, *medo*, *romance*) ao longo do tempo. Esses arcos são modelados como fórmulas temporais a serem cumpridas pelo planejador HTN. Por exemplo, para o gênero de contos de fadas,

poderia haver uma fórmula que exija que todo final do capítulo tenha de terminar com um alto nível de alegria. Isto poderia ser expresso com a fórmula *A end ({alegria ≥ 75})*, onde *75* representa um nível alto em relação os parâmetros definidos para o gênero. Além disso, um gênero romântico poderia exigir que o romance, a partir de certo momento em que a história fique triste, se mantenha sempre acima de certo valor. Isto poderia ser expresso através da fórmula:

$$AF (\{alegria < 30\} \text{ and } AG \{romance \geq 60\})$$

Diversas outras relações ricas podem ser pensadas já que a lógica é expressiva e a modelagem permite que se fale, em uma mesma fórmula, de várias propriedades dramáticas, relacionando-as.

A geração de histórias interessantes, com eventos não determinísticos ricos e conformidade a arcos dramáticos é capaz de aumentar a variabilidade de enredos com a incorporação de eventos não determinísticos, e, além disso, aumentar também a qualidade de enredos resultante da garantia de conformidade a arcos dramáticos típicos e com a incorporação dos próprios eventos não determinísticos com rica descrição.

3.3 Arquitetura

A arquitetura se divide em duas partes: o módulo de preprocessamento, *Drama Preprocessor* e o módulo de geração de capítulos, *NDet-IPG*. O *Drama Preprocessor* é responsável por preprocessar, anteriormente ao processo de geração, todos os arcos dramáticos típicos dos gêneros existentes e seus eventos e gravar na base de dados *Drama Database*, as estruturas de consulta de fórmulas temporais residuais a serem carregadas, na hora da geração, pelo *NDet-IPG*. Este último é o módulo responsável pela geração dos capítulos.

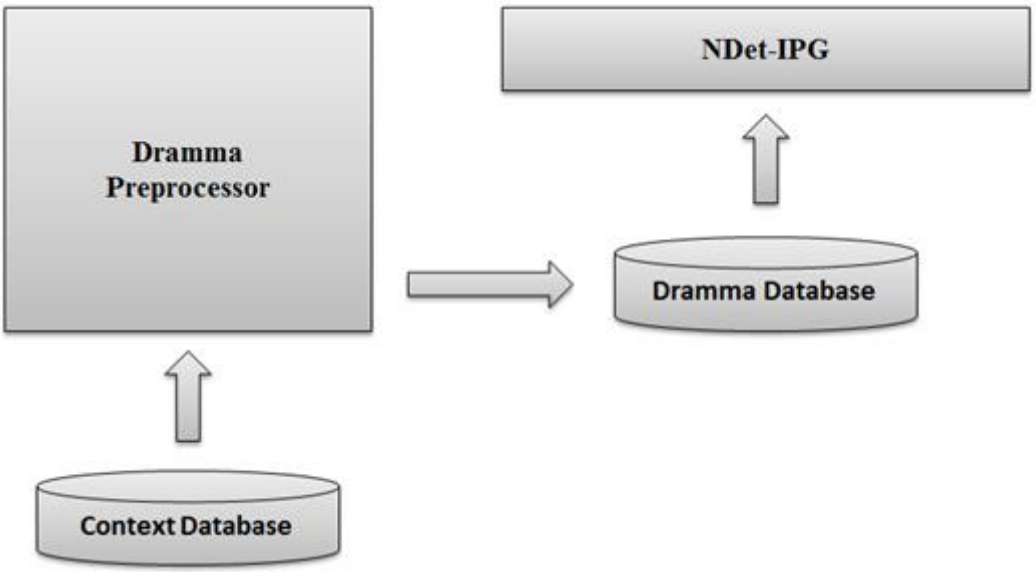

Figura 7. Arquitetura do modelo proposto

A base de dados *Context Database* na Figura 7 tem função semelhante à base de dados homônima na arquitetura do LOGTELL, mas aqui ela é pensada para guardar a especificação de diversos gêneros e contextos com os seus autômatos segundo a nova especificação (Seção 3.4.2). Desta forma, o preprocessamento pode gerar estruturas de consulta para todos os gêneros e contextos disponíveis na *Context Database*. Estas estruturas de consulta geradas são armazenadas na base de dados *Drama Database*. O módulo *NDet-IPG*, responsável pela geração de capítulos, carrega da *Drama Database* uma estrutura de consulta para o gênero (indicado no início do processo de geração). É importante notar que o carregamento de uma estrutura de consulta só precisa ser feito uma vez e esta fica, então, disponível durante todo o processo de geração para a utilização pelo *NDet-IPG*.

No módulo *NDet-IPG* (Figura 8), o planejamento de enredos é feito de forma conjunta entre o planejador de ordem parcial e o planejador HTN não determinístico em um submódulo denominado *Chapter Simulator*, responsável pela geração dos capítulos. Como pode haver mais de um planejamento ocorrendo em paralelo, há diversas instâncias de *Chapter Simulator* e estas são controladas pelo submódulo *Chapter Controller*.

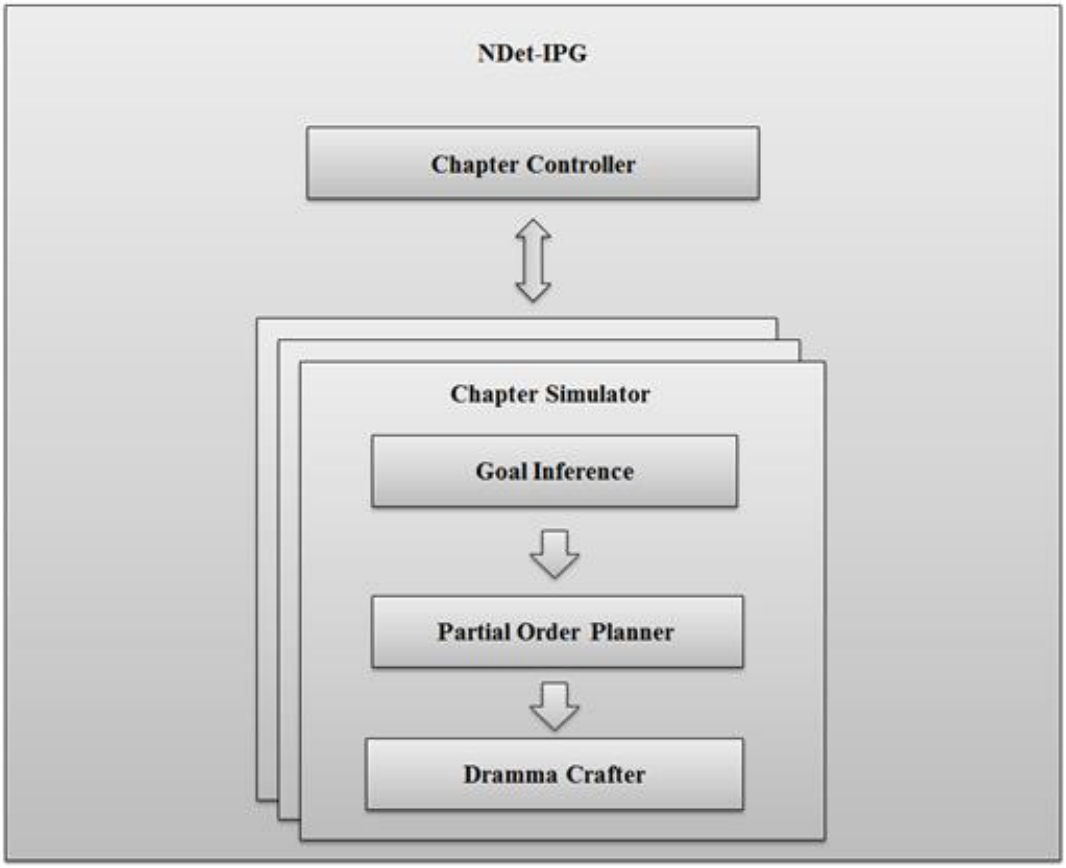

Figura 8. Módulo NDet-IPG

A geração de um capítulo é feita por cada *Chapter Simulator* através de três etapas. A primeira etapa, feita pelo submódulo *Goal Inference*, envolve a inferência de objetivos baseado nas regras de inferência de objetivos e no estado corrente da história (seção 2.4). Além disso, a primeira etapa também prevê a escolha ou inferência da fórmula temporal típica que terá de ser satisfeita pelo capítulo. O foco dessa pesquisa não é no processo de inferência ou escolha das fórmulas temporais típicas, mas o apresenta como uma possível frente de trabalho futuro (seção 6.2.4). A segunda etapa é onde entra o planejador de ordem parcial *Partial Order Planner* para gerar um plano que atinja os objetivos inferidos na etapa anterior. Esse plano é gerado como uma ordem parcial equivalente a uma HTN. O *Partial Order Planner* não leva em conta a fórmula temporal típica, que só será considerada pelo *Drama Crafter*. Por último, na terceira etapa, o *Drama Crafter* recebe a HTN gerada pelo *Partial Order Planner* e a fórmula temporal típica inferida (ou escolhida) pelo submódulo *Goal Inference* e conclui a geração do capítulo.

3.4 Drama Preprocessor

3.4.1 Visão Geral

O módulo *Drama Preprocessor* é responsável por gerar estruturas de consulta de fórmulas temporais residuais para todos os gêneros e contextos disponíveis considerando os arcos dramáticos típicos de cada gênero bem como os autômatos dos eventos. O preprocessamento é feito anteriormente à geração de enredos, de forma que tais estruturas de consulta estejam completas e disponíveis durante uma geração.

3.4.2 Autômatos

Cada evento básico tem associado a si um autômato que indica como as propriedades dramáticas variam dentro do evento. Os estados do autômato representam

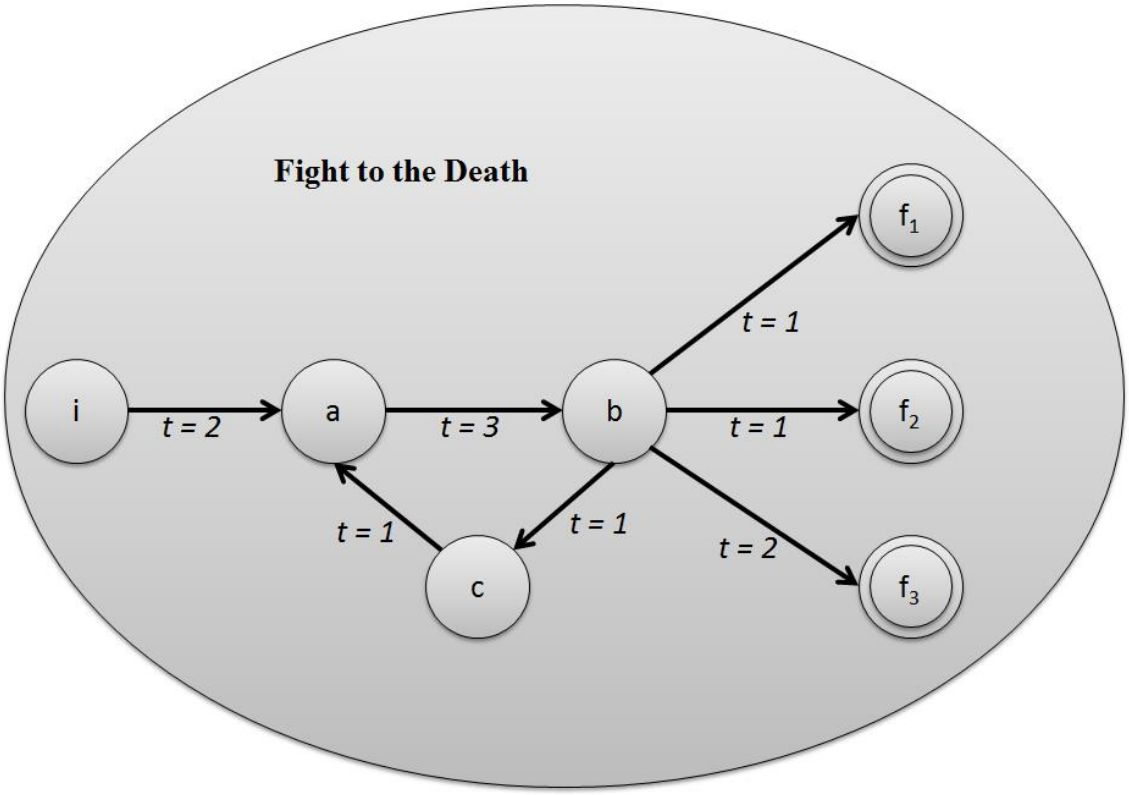

Figura 9. Autômato para o evento "*Fight_to_the_death*"

os estados parciais do mundo durante a execução do evento e as transições representam ações. Cada ação é uma sequência de *microações* levando de um estado ao outro. Uma microação é uma estrutura diretamente dramatizável que contém o seu tempo de duração e outros parâmetros de execução para um dramatizador.

A Figura 9, apresenta o autômato para o evento "*Fight_to_the_death*", com estado inicial i e três estados finais f1,f2 e f3. Neste evento dois personagens entram em

uma batalha que pode culminar com a morte do primeiro (f1), a morte do segundo (f2) ou a morte de ambos (f3). As setas representam as ações que levam de um estado a outro (com os tempos de execução abaixo). Cada ação tem uma duração e uma precondição (as precondições não estão consideradas na figura).

O autômato é definido de maneira semelhante à definição clássica de autômato finito determinístico em (HOPCROFT *et al*., 2006). Assim o autômato *e* aqui é um sistema $e = <nome(e),S,L,M,T,\delta,s_0,F>$, onde:

- *nome(e)* é a assinatura do autômato, que é igual à assinatura do operador do evento ao qual está associado (e.g. *fight_to_the_death(CH1,CH2)*), note que a assinatura possui parâmetros que tonam-se instanciados quando a instância do operador é definida);
- *S* é um conjunto (não vazio) de estados, onde cada estado $s \in S$ contém a definição dos valores das propriedades dramáticas em *s*, especificadas como combinações lineares dos valores das propriedades dramáticas no estado inicial, conforme explicado a seguir:
    - Uma propriedade dramática *p* no estado inicial s_0 de um autômato *e* pode assumir qualquer valor real;
    - O valor da propriedade dramática *p* em um estado $s \neq s_0$ de um autômato *e* com estado inicial s_0 é dado por uma combinação linear da forma $V(p,s) = a_0 \cdot V(p,s_0) + a_1 \cdot V(p_1,s_0) + a_2 \cdot V(p2,s_0) + ... + a_n \cdot V(p_n,s_0) + c$, onde:
        - $V(p,s)$ indica o valor da propriedade *p* no estado *s*;
        - $p, p_1, p_2 ... p_n$ são as diferentes propriedades do gênero;
        - $a_1, a_2, ... a_n$ são valores reais;
        - *c* é um valor real.
    - É importante notar que:
        - O valor de uma propriedade dramática num dado estado é relativo somente aos valores iniciais de propriedades dramáticas no autômato;
        - Uma propriedade dramática pode ter seu valor relacionado ao valor inicial de outra(s) propriedade(s) dramática(s);
        - O valor de uma propriedade dramática num certo estado pode ser igual a uma constante.

- L é um conjunto de literais que é um subconjunto do conjunto de literais do gênero;
- M é um conjunto de microações que é um subconjunto do conjunto de microações do gênero, onde cada microação $m \in M$ é uma dupla $m = <t,info_Dramatization>$, onde t é um tempo estimado de dramatização de m em um dramatizador e *info_Dramatization* contém as informações necessárias para a dramatização de m por um dramatizador. As microações também possuem parâmetros que indicam como a dramatização deve ocorrer;
- T é um conjunto de transições, onde uma transição $t \in A$ é uma dupla $t = <precond(t),microactions(t)>$, onde:
    - *precond(t)*, também chamada de precondição de t, é uma conjunção de literais pertencentes a L definida em lógica de primeira ordem, onde as variáveis que não são parâmetros do autômato são assumidas como existencialmente quantificadas; essa precondição indicará, em relação ao valor das propriedades de contexto, se a ação pode ou não acontecer;
        - *microactions(t)* é uma sequência de microações $<m_1,m_2,...,m_i>$, tal que $m_1, m_2, ..., m_i \in M$;
- δ é uma função de transição de estados $\delta : S \times A \rightarrow S$ que indica qual estado é atingível através da aplicação de uma ação a partir de um estado; note que o autômato torna-se não determinístico ao se assumir que qualquer transição pode acontecer a partir de um estado, desde que suas precondições sejam válidas; eventualmente, transições podem ser agrupadas em ações não determinísticas, mas, para fins dessa dissertação isto não é relevante (ou seja, todas as transições que saem de um estado podem ser consideradas como componentes de uma única ação);
- s_0 é o estado inicial $s_0 \in S$ do autômato. Só há um estado inicial;
- F é um conjunto contendo todos os estados finais do autômato, tal que $f \in S$ é um estado final do autômato.

3.4.3 Preprocessamento

A partir da definição do conjunto de arcos dramáticos típicos e dos autômatos para representar os eventos básicos de um gênero, a etapa de preprocessamento consiste

em um algoritmo para percorrer todos os autômatos, verificando as fórmulas temporais referentes aos arcos dramáticos, e obtendo fórmulas temporais residuais, que são então novamente aplicadas a todos os autômatos.

Esta etapa se divide em duas subetapas. A primeira consiste em obter as *árvores de execução* de cada autômato, e a segunda, em percorrer estas árvores realizando uma verificação abstrata das fórmulas temporais a fim de obter a fórmula residual para cada final do autômato.

A obtenção de árvores de execução é feita através do percorrimento do autômato considerando um tempo mínimo e um tempo máximo para limitar execuções do estado inicial a um estado final. Desta forma, obtém-se uma árvore onde os nós representam estados do autômato e as arestas, ações do autômato levando de um estado a outro. Na árvore de execução, a raiz representa o estado inicial do autômato e as folhas, os estados finais. É importante notar que um mesmo estado final do autômato pode vir a ser representado por mais de uma folha caso haja mais do que um caminho do estado inicial a esse estado final dentro dos limites de tempo mínimo e máximo. Cada folha também tem anotada em si uma fórmula de lógica de primeira ordem que é a conjunção simplificada de todas as fórmulas das ações representadas pelas arestas no caminho da raiz até a folha. Finalmente, ciclos e laços levam à repetição de nós na árvore que representam os mesmos estados do autômato.

Na Figura 10 abaixo, à esquerda, há uma árvore de execução resultante do processo de geração considerando um tempo máximo de 7 unidades e, à direita, com tempo máximo de 12 unidades. Para este exemplo, não foram consideradas as precondições das ações.

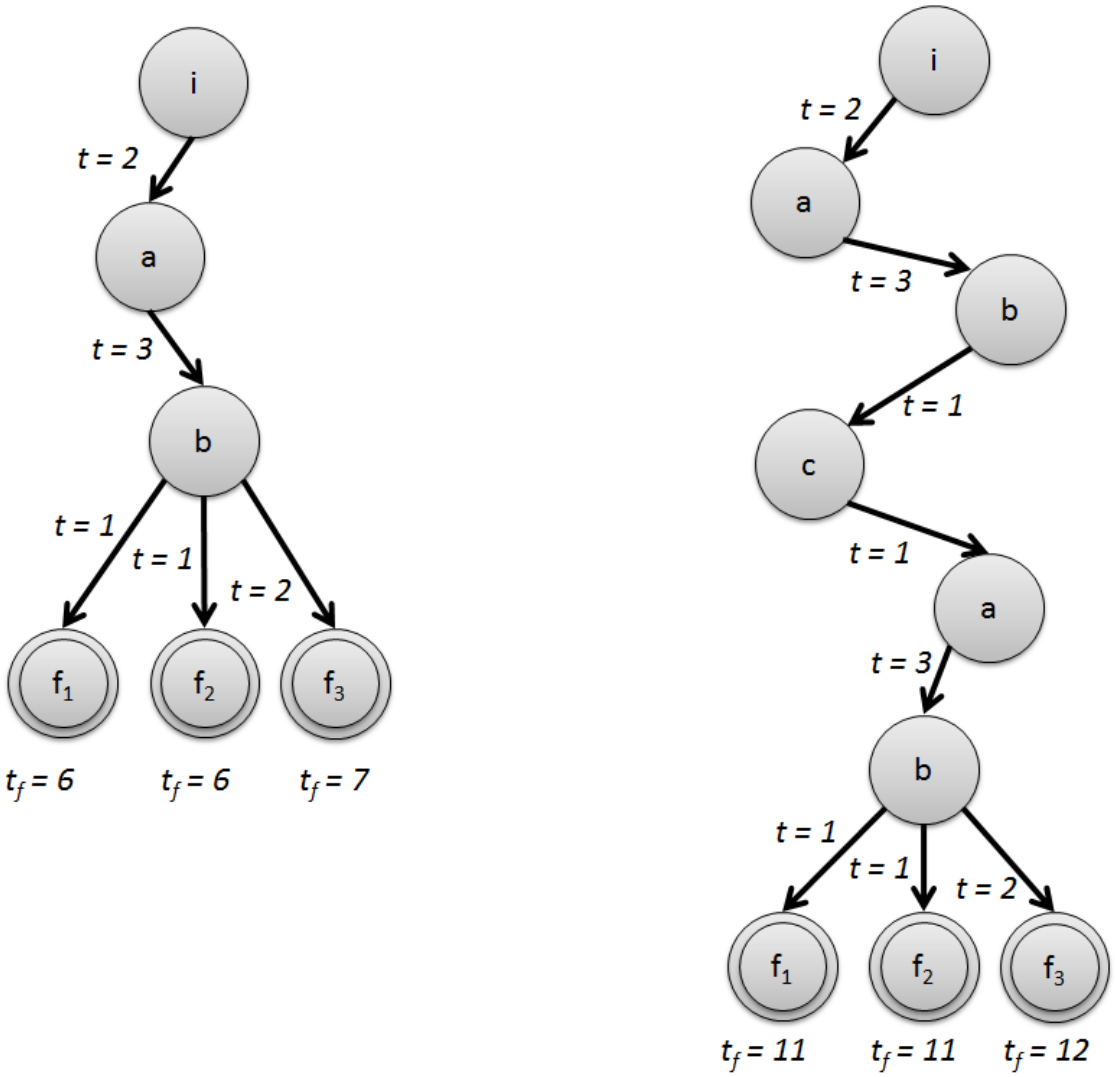

Figura 10. Árvores de execução para o evento *fight_to_the_death*.

A partir da obtenção da árvore de execução para um autômato, é necessário percorrê-la considerando primeiramente uma fórmula temporal típica. A técnica utilizada para tal percorrimento é CLP (Seção 2.1.5). A técnica é utilizada como em (ARAÚJO, 2011), só que realizando uma verificação abstrata, isto é, sem conhecimento do valor concreto das variáveis nas fórmulas temporais. A cada transição da árvore, obtêm-se as alternativas possíveis de satisfação da fórmula considerando faixas de valores possíveis para as variáveis. Cada alternativa, então, é avaliada separadamente a partir dali até que se chegue a uma folha. Além disso, a fórmula corrente no nó e alternativa sendo considerado pode desdobrar-se em outras fórmulas, que devem então ser avaliadas separadamente, e, cujos resultados, devem ser avaliados após o processo estar finalizado. Uma fórmula temporal típica de entrada para uma árvore de execução

resultará, possivelmente, em fórmulas residuais diferentes nas folhas. Estas fórmulas residuais devem ser realimentadas para o algoritmo a fim de que se tenha um mapeamento para qualquer fórmula de entrada que seja uma fórmula temporal típica ou uma fórmula residual resultante de consulta.

O resultado do preprocessamento dos autômatos de um gênero é uma estrutura de consulta que contém mapeamentos de uma tripla da forma $(event,ending,\alpha_{entry})$ para uma tripla da forma $(constraints_ending,precond_ending,\alpha_{resid})$, onde:

- *event* é uma assinatura de um operador;
- *ending* é um identificador de um estado final do autômato do evento;
- α_{entry} é uma fórmula temporal de entrada;
- *constraints_ending* é um conjunto de restrições sobre os valores de entrada das propriedades dramáticas que precisa ser válido para que se possa considerar a fórmula residual;
- *precond_ending* é uma fórmula de lógica de primeira ordem composta das precondições das arestas (representando ações do autômato) em cada caminho que leva do nó raiz até um nó folha que represente o final identificado por *ending*; essa fórmula precisa estar válida para que o final seja atingido; e
- α_{resid} é uma fórmula residual.

Essa estrutura é gravada, associada ao gênero, na base de dados *Drama Database* e será utilizada para compor uma estrutura indexada em memória, a ser gerada e utilizada pelo módulo *Drama Consult*, capaz de fornecer duas consultas ao planejador: os finais de um evento que estão ativos, isto é, que são atingíveis considerando o estado corrente do mundo; e a fórmula residual para certo evento, um final deste evento e uma fórmula de entrada.

3.5 Drama Consult

3.5.1 Visão Geral

Este módulo é responsável por gerar e manter em memória a estrutura indexada, de certo gênero, que será utilizada pelo planejador para consultar as fórmulas residuais. Além disso, o módulo também deve ser capaz de responder quais finais estão ativos para certo evento levando em conta o estado corrente do mundo. As estruturas de dados necessárias para isso devem ser geradas a partir de uma consulta à base de dados *Drama Database*, que contém os resultados do preprocessamento.

3.5.2 Obtendo os Finais Ativos de um Evento

O autômato associado a um evento básico tem precondições para as ações que fazem a transição de um estado ao outro. Assim, a árvore de execução contém em suas arestas essas precondições de modo que, a partir de um caminho do nó raiz até um nó folha, pode-se compor uma conjunção das precondições. Tal conjunção representa o que deve ser válido no nó raiz para que o nó folha seja atingível. Como há possivelmente mais de um nó folha por final do autômato, pode-se compor uma disjunção dos caminhos do nó raiz até todos os nós folha para um final a fim de obter uma fórmula que represente o que deve ser válido no nó raiz para que o estado final do evento seja atingível. A partir do estado atual do mundo, é possível, então, saber, para certo evento, quais estados finais são atingíveis. Deste modo, o planejador pode realizar uma consulta para saber para quais estados finais do evento corrente é possível continuar planejando. Isso é importante em termos de atender corretamente a restrições sobre caminhos que envolvem quantificações existenciais e universais.

3.5.3 Obtendo Fórmulas Residuais

O módulo *Drama Consult* mantém em memória uma estrutura indexada, onde uma chave é uma dupla $<event,\alpha_{entry}>$ e um registro, uma dupla da forma $<situation_constraints,\alpha_{resid}>$. A estrutura mapeia uma chave para um ou mais registros. A consulta por uma fórmula residual é feita passando uma tripla $<event,\alpha_{entry},P_{current}>$, onde $P_{current}$ é uma lista contendo os valores correntes das propriedades dramáticas. A partir dessa tripla, e consultando o estado corrente da história, o módulo pode, com a chave $(event,\alpha_{entry})$ obter os registros e testar as restrições contidas no em *situation_constraints* de modo a descobrir a fórmula α_{resid} que deve ser imposta aos eventos que ocorrem a partir de cada um dos finais. A fórmula α_{resid} contém conjunções e disjunções de pares $<ending,\alpha_{ending}>$, onde ending especifica um dos finais do evento e α_{ending} uma fórmula a ser verificada a partir daquele final. Note que α_{resid} pode conter mais de uma referência a certo final e que pode acontecer de não haver nada a ser verificado a partir de certo final.

3.6 Drama Crafter

3.6.1 Visão Geral

A fim de se incorporar o não determinismo, é necessária uma definição para que um operador possa ter efeitos não determinísticos. A definição aqui utilizada é a mesma de (SILVA, 2010), conforme explicado na seção 2.5.3.

Este módulo é responsável por compor um capítulo que obedeça a uma fórmula temporal típica a partir de um esboço de enredo em forma de HTN. A técnica empregada é planejamento HTN não determinístico com capacidade de verificar fórmulas de lógica temporal modal a partir da utilização de uma estrutura de consulta de fórmulas residuais. Para fins de fluidez de explicação, toma-se a liberdade de falar em "fórmula universal" para se referir a fórmulas universalmente quantificadas nos caminhos e "fórmula existencial" para se referir a fórmulas existencialmente quantificadas nos caminhos. Nesta seção é apresentada uma explicação geral dos principais atributos do modelo de planejamento para lidar com a verificação de fórmulas temporais. O algoritmo que implementa esses atributos é apresentado no próximo capítulo.

3.6.2 Planejamento Não Determinístico com Verificação de Propriedades Dramáticas

O planejamento HTN puro, sem levar em conta a verificação de fórmulas temporais, é feito segundo as definições apresentadas nas seções 2.1.2 e 2.5.3. Esta seção foca no tratamento de fórmulas temporais.

Antes de iniciar o planejamento, faz-se uma normalização da fórmula temporal de entrada. Esta fórmula, já normalizada, pode corresponder a múltiplos elementos agregados por meio de conjunções e disjunções, onde cada elemento contém uma única quantificação mais geral (nos caminhos e nos estados ao longo do caminho). A partir da fórmula normalizada gera-se um conjunto de fórmulas para iniciar o planejamento Neste conjunto, as fórmulas cuja não satisfação invalidaria toda a fórmula de entrada são marcadas como obrigatórias. O planejamento segue decompondo e especializando as tarefas da HTN de entrada até que se chegue a um evento básico, que é onde as fórmulas temporais são consideradas.

A verificação de fórmulas temporais é feita somente nos eventos básicos, que são os únicos que modificam o valor de propriedades dramáticas. O tratamento de

eventos básicos é feito consultando-se a fórmula residual para o evento em relação a todos os seus finais. Isto é, a estrutura retornada é uma fórmula, possivelmente complexa, que indica o que deve valer, em relação a todos os finais do evento, para que a fórmula de entrada seja satisfeita A consulta por fórmula residual é feita para cada um dos elementos da fórmula corrente de entrada que está sendo examinada.

A obtenção da fórmula residual pode vir a satisfazer um dos elementos da fórmula de entrada (ou a fórmula inteira). A consulta por fórmulas residuais retorna duas estruturas: uma lista de pares <*elemento,residual*>, contendo o elemento para o qual a consulta foi feita e a sua fórmula residual *residual*; e uma substituição, que indica quais fórmulas foram satisfeitas pelo evento e quais falharam. É importante notar que uma fórmula obrigatória não pode falhar durante a análise da consulta. A fórmula *residual* é uma fórmula que contém indicações em cada um de seus elementos sobre em qual final aquele elemento deve valer (seção 3.5.3).

Após obter a fórmula residual, itera-se sobre todos os finais do evento, planejando para cada um deles. Todavia, antes de planejar para um final do evento, é necessário "filtrar" a estrutura que retornou da consulta por fórmulas residuais a fim de obter somente as fórmulas que dizem respeito ao final corrente. Além disso, atualiza-se o valor das propriedades dramáticas de acordo com o evento e o final corrente.

Quando o planejamento volta para um final, é necessário verificar se a fórmula foi satisfeita, ou se falhou. Isto é possível, pois o planejamento retorna uma substituição indicando quais fórmulas foram satisfeitas e quais falharam na chamada recursiva. Tal substituição é aplicada à estrutura corrente de pares <*elemento,residual*> a fim de obter quais elementos foram satisfeitos e quais falharam. Se nenhuma fórmula obrigatória falhou, essa estrutura é, então, atualizada e o planejamento prossegue até que se tenha planejado para todos os finais. Quando acaba o planejamento para todos os finais, os elementos (da fórmula de entrada para o evento) que foram satisfeitos e os que falharam estão acumulados em uma substituição, que é retornada para cima.

3.6.3 Manutenção em Forma Normal

As fórmulas temporais estão sempre mantidas em uma forma normal a fim de permitir uma análise mais eficiente de seus elementos. Nessa forma normal, as implicações são removidas e as negações são recursivamente "empurradas para dentro" com a utilização de as equivalências: $\{\neg AG\alpha \equiv EF\neg\alpha,\ \neg AF\alpha \equiv EG\neg\alpha,\ \neg EG\alpha \equiv AF\neg\alpha, \neg EF\alpha \equiv AG\neg\alpha,\ \neg\{a > b\} \equiv \{a \leq b\}, \neg\{a < b\} \equiv \{a \geq b\}, \neg\{a = b\} \equiv \{a \neq b\}\}$.

Em uma forma normalizada α_{norm}, faz-se referência a um *elemento* de α_{norm} como sendo uma subfórmula de α_{norm} que só contenha um quantificador mais externo. Por exemplo, considere a fórmula normalizada α_{norm}:

$$((AG\ (\alpha\ and\ EF\ \beta))\ or\ AF\ \gamma)\ and\ EG\ \zeta$$

Neste caso, há três elementos de α_{norm}, que são: $AG\ (\alpha\ and\ EF\ \beta)$; $AF\ \gamma$; e $EG\ \zeta$.

3.6.4 Fórmulas Obrigatórias e Não Obrigatórias

A cada nova consulta para gerar fórmulas residuais a satisfazer, o planejamento marca certos elementos dessas fórmulas residuais como fórmulas *obrigatórias*. As fórmulas obrigatórias são as que necessariamente devem valer durante o planejamento. Quando uma fórmula obrigatória "falha", ou seja, não é satisfeita em algum ponto, ela pode levar o planejamento a falhar no ponto e realizar *backtracking*. Fórmulas que não sejam obrigatórias nunca podem levar o planejamento a falhar. É importante notar que uma fórmula obrigatória pode estar associada por uma disjunção a outra fórmula não obrigatória e, portanto, pode falhar e não necessariamente levar o planejamento a falhar.

Um elemento $\alpha_{elemento}$ de uma fórmula $\alpha_{residual}$ que seja fórmula residual de uma fórmula $\alpha_{original}$ é obrigatório *sse* $\alpha_{original}$ é obrigatória e $\alpha_{elemento}$ é consequência lógica de $\alpha_{residual}$. A definição é recursiva e encontra ponto fixo no início do planejamento, onde todo elemento $\alpha_{elemento_inicial}$ da fórmula temporal inicial $\alpha_{inicial}$ que é consequência lógica de $\alpha_{inicial}$ é marcado como obrigatório. Deste modo, a obrigatoriedade de fórmulas pode ser ou não "herdada" de uma fórmula original por elementos de sua fórmula residual. Além disso, mesmo que em certo passo do planejamento não haja fórmulas obrigatórias a satisfazer, o planejamento pode falhar. É importante notar que, quando foi mencionada a obtenção de fórmulas temporais que são consequência lógica de outras, a consequência lógica é obtida através da análise do elemento em relação às disjunções e conjunções em que ele se encontra em sua fórmula.

Fórmulas podem vir a ser anotadas como obrigatórias tanto no início como durante o processo de planejamento. Por exemplo, se há como entrada para o planejador a fórmula $(AG\ \alpha)\ and\ (EF\ \beta)$, então ambos os elementos da fórmula são marcados como fórmulas obrigatórias, gerando o conjunto $\{obrig(AG\ \alpha), obrig(EF\ \beta)\}$. Todavia, ao consultar um evento com dois finais para a fórmula $EF\ \beta$, marcada como obrigatória, se for obtida como fórmula residual $(f_1, EF\ \beta)\ or\ (f_2, EF\ \beta)$, indicando que $EF\ \beta$ deve ser satisfeita por um final *ou* por outro, então, para cada um dos finais, a fórmula deixa de ser obrigatória. A intuição por trás desta eliminação da indicação de obrigatoriedade é

que no momento em que o planejamento seguir pelo final $final_1$, ele pode não satisfazer $EF\ \beta$ sem falhar porque ainda há chance de satisfazer $EF\ \beta$ pelo final $final_2$. Em outro exemplo, se há como entrada para o planejador a fórmula $AG\ \alpha\ or\ EF\ \beta$, então nenhum dos dois elementos será uma fórmula obrigatória, pois nenhum deles é consequência lógica da fórmula de entrada. Todavia, quando retornar ao início, o planejador terá de ter satisfeito pelo menos uma das duas fórmulas. Isto é garantido, pois o planejador retorna a indicação de quais fórmulas foram satisfeitas.

3.6.5 Anotação de Fórmulas Satisfeitas e Não Satisfeitas

É interessante que se possa evitar a verificação redundante de fórmulas temporais. Para isso, é necessário algum esquema de anotação de fórmulas que já estão satisfeitas e não precisam mais ser verificadas. Por exemplo, se, em um ramo do planejamento, um evento já satisfizesse uma fórmula $EF\ \beta$, não haveria a necessidade de verificá-la no ramo irmão. Esta fórmula poderia ser eliminada a partir do momento em que um ramo já a satisfez.

O modelo lida com essa anotação através de uma *substituição*, que é repassada na volta das chamadas recursivas ao tratamento de operadores básicos, e indica quais fórmulas foram satisfeitas e quais fórmulas falharam. Uma substituição é um conjunto τ cujos elementos são substituições da forma $\alpha \rightarrow true$ ou $\alpha \rightarrow false$, onde α é uma fórmula temporal. Assim, supondo que $EF\ \beta$ foi resolvido por um evento, no primeiro final f_1, teríamos uma substituição $\tau = \{EF\ \beta \rightarrow true\}$. Como para o evento que recebeu esta substituição, havia a fórmula a resolver $(f_1, EF\ \beta)\ or\ (f_2, EF\ \beta)$, é possível simplificar, para o final f_1, $EF\ \beta$ para *true* e, então, levar a disjunção a virar *true*, eliminando a necessidade de um ramo irmão verificar $EF\ \beta$.

Além de evitar uma possível verificação redundante das fórmulas residuais, a anotação de fórmulas satisfeitas e não satisfeitas serve para controlar a satisfação de fórmulas obrigatórias. Se uma fórmula obrigatória tem alguns dos elementos de sua fórmula residual substituídos por *false*, então, possivelmente ela poderá ser simplificada para *false*, ou seja, "falhar". Por exemplo, suponha uma única fórmula obrigatória de entrada $EF\ (\alpha\ and\ AG\ \beta)$, que tem como fórmula residual, no primeiro evento, $EF\ (\alpha\ and\ AG\ \beta)\ or\ AG\ \beta$. Além disso, assuma que o planejamento tenha construído um plano e retornado ao ponto de início com a substituição $\tau = \{EF\ (\alpha\ and\ AG\ \beta) \rightarrow false, AG\ \beta \rightarrow false\}$. Neste caso, a aplicação da substituição τ na fórmula residual $EF\ (\alpha\ and\ AG\ \beta)\ or\ AG\ \beta$ gera a fórmula *false or false*, simplificada para *false*, o que faz com que $EF\ (\alpha$

and AG β) não esteja satisfeita e, como é obrigatória, também leve o planejamento a "falhar" no ponto.

3.6.6 Operadores de Estado e Eventos Finais no Plano

Os operadores de estado só precisam ser considerados quando o planejamento está encadeando eventos em um final do plano. Em eventos internos, o trabalho de verificação de operadores de estado está implícito no preprocessamento feito, que é a fase que lida no nível dos estados propriamente ditos.

Quando se obtém a fórmula residual para um evento em um final do plano, é necessário verificar se está satisfeita. Como a CTL é pensada originalmente para árvores infinitas, é necessário uma adaptação para o caso de um plano finito. Assim, toma-se a interpretação semântica de que se os valores das propriedades dramáticas se mantêm iguais a partir do final do plano *ad infinitum*.

Considere os elementos de fórmula residual obtida na consulta para certo final um evento em um final do plano. Um elemento com operador de estado *G* significa o evento não contradisse o operador de estado e, portanto, a fórmula de entrada está satisfeita. Um elemento com operador de estado *F* significa o evento não satisfez o operador de estado e, portanto, a fórmula de entrada não está satisfeita. Um elemento com operador de estado *U* significa o evento não contradisse o operador de estado e, portanto, a fórmula de entrada não está satisfeita. Finalmente, é só nesse ponto que fórmulas sob o operador de estado *end* podem ser verificadas. Um elemento com operador de estado *end* leva à necessidade de se verificar a fórmula sob o operador. Essa fórmula pode ser complexa, e, portanto, precisa ter suas subfórmulas analisadas de modo que todas as restrições básicas sobre propriedades dramáticas satisfeitas de acordo com o valor final das propriedades dramáticas.

Por exemplo, considere a fórmula residual complexa obtida em uma consulta para um final de um evento encadeado em um final do plano:

A end ({alegria >50} or EF ({suspense>30} and AF {medo>20}) or AG {medo>30})

Neste caso, os valores das propriedades dramática no final do plano em questão teriam de satisfazer a restrição complexa:

{alegria >50} or ({suspense>30} and {medo>20}) or {medo>30}

3.6.7 Exemplo de Planejamento HTN Não Determinístico com Verificação Temporal

A fim de esclarecer o processo de planejamento, esta seção apresenta uma visão geral de como funciona o processo de planejamento HTN para satisfazer fórmulas temporais. Esta seção não entra em detalhes do algoritmo, já que este é apresentado em detalhes no capítulo 0. O objetivo desta seção é exemplificar, através de uma execução-exemplo concreta, algumas das possibilidades do modelo de planejamento no que concerne à satisfação de fórmulas temporais.

O enredo-exemplo consistirá de uma simples aventura, na qual um herói e um vilão vão a um deserto para resolver um conflito de longa data. Durante essa aventura, se deseja que o medo da história nunca fique muito alto e deve haver a possibilidade de acontecer um pico de suspense a partir do qual, nos finais, o suspense volta a um nível baixo.

A HTN de entrada será simples e composta de três operadores (já instanciados):

U = {(1,go('Brian','Desert')), (2,go('Draco','Desert')), (3,confront ('Brian','Draco')}

C = {(1,3),(2,3)])

O evento *confront(CH1,CH2)* é abstrato e pode se especializar em um dos dois operadores básicos:

(4,fight_to_the_death(CH1,CH2))

(5,parley_for_truce(CH1,CH2))

A fórmula temporal de entrada será:

AG {tristeza<70} and EF ({suspense>60} and A end {suspense <50})

O primeiro passo é normalizar a fórmula temporal de entrada. Neste caso, ela já se encontra na forma normal desejada. Em seguida, é necessário obter os elementos obrigatórios da fórmula temporal de entrada. No caso da fórmula acima, ambos os elementos são consequência lógica da fórmula de entrada e, portanto, marcados como obrigatórios.

O planejamento segue buscando tarefas sem predecessores e decompondo-as para chegar a eventos básicos a incorporar ao plano. As fórmulas temporais só são tratadas nos eventos básicos. A cada final de um evento básico, o planejador gera fórmulas residuais a satisfazer e recebe, como retorno recursivo, uma indicação de quais

fórmulas foram satisfeitas e quais não. Cada fórmula estabelece com sua fórmula residual uma relação de dependência, que deve ser avaliada quando o planejamento retorna da recursão.

No exemplo apresentado, o evento *go('Brian','Desert')* indica que o herói *Brian* se locomove até o local *Desert*. O evento *go('Brian','Desert')* é similar, mas para o vilão *Draco*. O operador *confront(CH1,CH2)* representa um evento abstrato simbolizando um confronto entre dois personagens, que pode se resolver de maneiras distintas. Estas estão representadas pelas duas especializações: *fight_to_the_death(CH1,CH2)* e *parley_for_truce(CH1,CH2)*.

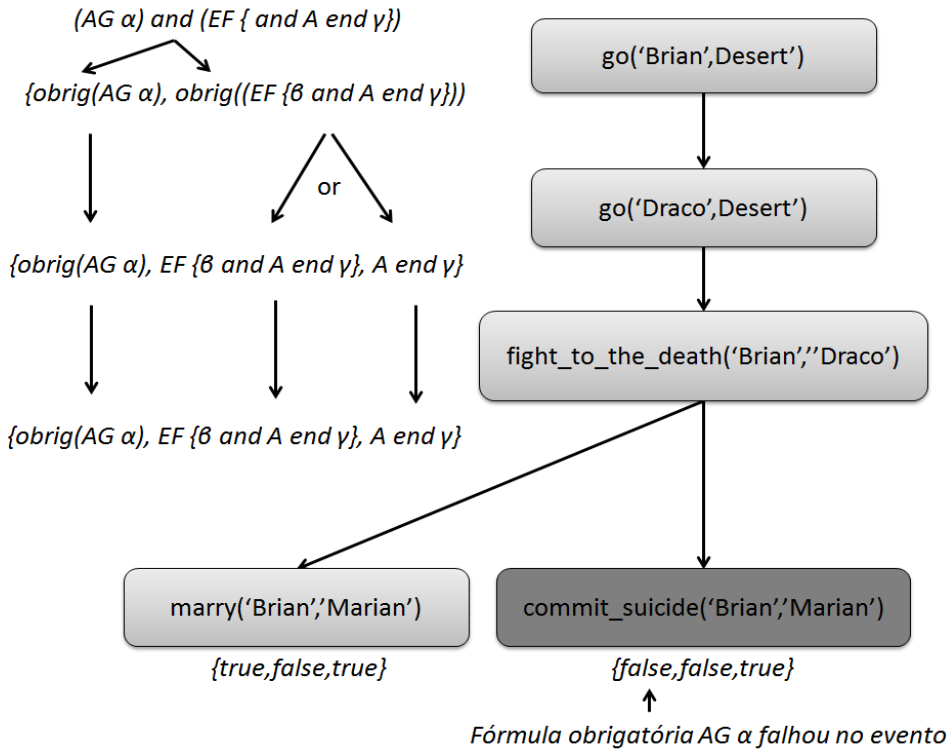

Figura 11. Exemplo de planejamento HTN temporal com falha.

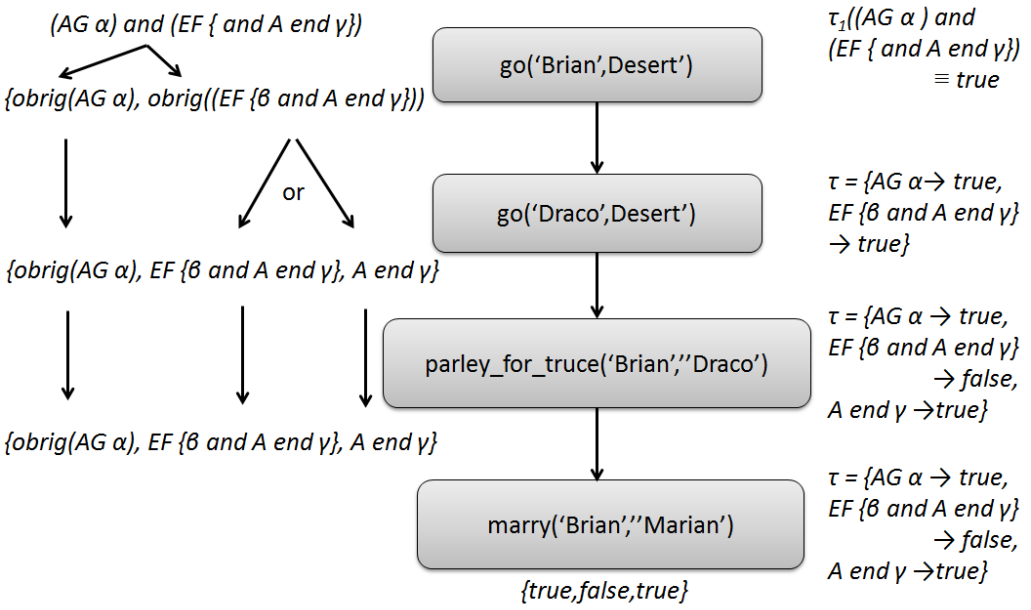

Figura 12. Exemplo de planejamento HTN temporal com sucesso.

Em *fight_to_the_death(CH1,CH2)*, os personagens lutam até a morte e a batalha pode culminar com a morte de um, de outro, ou de ambos. São *3* finais diferentes já previamente analisados pelo processamento.

Na segunda especialização, os personagens entram em um acordo de trégua. A utilização do evento abstrato *confront(CH1,CH2)* serve para exemplificar uma das maneiras pela qual o planejador pode vir a escolher uma ou outra especialização de acordo com a fórmula temporal que se deseja estabelecer. Nesse caso, vê-se que o planejador irá escolher um capítulo menos melancólico através da especialização *parley_for_truce(CH1,CH2)* que levará a um final que não leva a fórmula temporal *AG {tristeza <80}* a falha.

A Figura 11 e a Figura 12 apresentam uma indicação do processo de planejamento para a fórmula temporal e a HTN de exemplo. As restrições sobre as propriedades dramáticas são simbolizadas pelas letras α, β e γ, respectivamente representando *{tristeza<70}*, *{suspense>60}* e *{suspense <50}*. Deste modo, a figura fica mais sucinta e direta.

A leitura do processo deve ser feita começando na esquerda em cima, descendo, e voltando pela direita para cima. Os círculos representam eventos básicos incorporados ao plano. As setas conectando um círculo ao outro simbolizam os finais do evento e sua continuação por outro evento. As setas conectando fórmulas indicam a relação de

dependência residual a ser avaliada quando o planejamento retorna. No caso da fórmula inicial, as setas indicam que os elementos foram marcados como obrigatórios devido ao *and* unindo-os em uma conjunção. A verificação na folha (após o final de um evento terminal no plano) sempre retorna *true* ou *false* para qualquer fórmula que receber como entrada. Deste modo, qualquer fórmula que não tiver sido satisfeita ou falhado por um evento, terá sua situação conhecida quando chegar a uma folha. As setas para cima, à direita, mostram a substituição τ que é retornada para que o nível de cima possa avaliar quais fórmulas foram satisfeitas e quais não.

Na Figura 11, o planejamento escolheu os eventos *go('Brian','Desert')*, *go('Draco','Desert'))* e *fight_to_the_death('Brian','Draco')*. Este último foi escolhido como possível especialização do evento *confront ('Brian','Draco')*. Após o evento *fight_to_the_death('Brian','Draco')*, é encaixado para o primeiro final, o evento *marry('Brian','Marian')*, mas ao incorporar o evento para o segundo final (onde o herói é assassinado pelo vilão), *commit_suicide('Marian')*, o planejamento falha. O evento *commit_suicide('Marian')* não satisfaz a fórmula obrigatória *AG {tristeza<70}*. Isto leva a um *backtracking* do processo de planejamento e à escolha do evento *parley_for_truce('Brian','Draco')* no lugar de *fight_to_death('Brian','Draco')*. Esta nova escolha leva à incorporação, após o único final de *parley_for_truce('Brian','Draco')* (Figura 12), do evento *marry('Brian','Marian')*. Este último satisfaz a fórmula obrigatória *AG {tristeza<70}* e a fórmula *A end {suspense <50}*, gerando uma substituição que é retornada e avaliada em cada nível na volta da recursão e, eventualmente, leva à satisfação da fórmula original de entrada *AG {tristeza<70} and EF ({suspense>60} and A end {suspense <50})*.

No exemplo mostrado, o planejamento, a fim de manter o nível de *tristeza* abaixo do limiar exigido pela fórmula *AG {tristeza <70}*, escolheu uma especialização a cumprir que fornece ao capítulo, um tom muito diferente do que se outra fórmula temporal estivesse sendo exigida. Deste modo, pode-se ver o potencial de fórmulas temporais para guiar o planejamento e obter capítulos que obedeçam a arcos dramáticos desejados.

4 Algoritmo

4.1 Visão Geral

Este capítulo apresenta o algoritmo e os aspectos relevantes da implementação do protótipo desenvolvido para validar o modelo de planejamento HTN não determinístico com restrições temporais sobre propriedades contínuas. O planejador Temporal NDet-HTN fruto da pesquisa é baseado no planejador NDet-HTN e foi implementado no mesmo ambiente (SILVA, 2010). Assim, o Temporal NDet-HTN foi implementado em SICStus Prolog (SICSTUS, 2013) e também é capaz de lidar com planejamento em paralelo a partir dos diferentes estados finais alcançados por um plano.

O algoritmo assume a disponibilidade dos dados de preprocessamento das fórmulas, conforme mencionado no capítulo anterior. A implementação do preprocessamento e da consulta aos resultados desse preprocessamento estão descritas em (GOTTIN, 2013), o qual implementa esses módulos em SICStus Prolog, utilizando CLP.

O módulo *Drama Crafter* é o módulo responsável pela composição de um capítulo a partir de um esboço de plano a ser detalhado, uma fórmula temporal típica e de valores inicias para as propriedades dramáticas. Como explicado anteriormente, o esboço de plano é uma ordem parcial de eventos equivalente a uma rede hierárquica de tarefas (HTN), a qual é gerada pelo planejamento de ordem parcial.

Além das modificações que permitiram a incorporação do tratamento das fórmulas temporais sobre propriedades dramáticas, foi implementada uma otimização no mecanismo de controle de não determinismo que teve impacto importante para melhorar o desempenho do algoritmo, conforme descrito na seção 5.2.3.

O algoritmo Temporal NDet-HTN mantém a mesma lógica do NDet-HTN no que concerne à decomposição e especialização de tarefas e construção do plano. A mudança principal é no tratamento de eventos básicos que agora modificam o valor das

propriedades dramáticas cuja variação pode satisfazer ou invalidar fórmulas temporais modais.

A gramática utilizada para expressar restrições sobre as propriedades dramáticas é a gramática definida em (ARAÚJO, 2011). Algumas fórmulas dessa gramática, no entanto, não são passíveis de serem expressas sobre valores de intervalos contínuos diretamente e não são permitidas:

- Negações de fórmulas $A\ \alpha\ U\ \beta$ e $E\ \alpha\ U\ \beta$;
- Fórmulas do tipo $AG\ \alpha$ e $EF\ \alpha$, quando α é uma disjunção de fórmulas (básicas ou quantificadas) ou uma fórmula do tipo $\beta \neq \gamma$.

ARAÚJO (2011) aplica uma técnica verificação por meio de negação por falha para esses casos, a qual não é adotada neste trabalho por afetar o desempenho.

4.2 Algoritmo de Entrada

O pseudocódigo do algoritmo inicial *ObterPlano* é apresentado na Figura 13. Como os enredos são gerados por meio de capítulos, o estado corrente para o qual é feito o planejamento de um capítulo difere do estado inicial da história. O estado inicial da história e as estruturas geradas pelo preprocessamento estão implícitos ao longo do algoritmo, sendo acessados via um banco de dados. Os dados de entrada e de retorno desse algoritmo são:

Dados de entrada:

- w: uma rede de tarefas (HTN) na mesma forma mostrada na seção 2.1.2;
- α: uma fórmula temporal típica;
- S_{corr}: uma dupla $<Fatos^{+},Fatos^{-}>$. $Fatos^{+}$ é um conjunto de átomos "ground" representando os fatos adicionados no mundo em relação ao estado inicial. $Fatos^{-}$ é um conjunto de átomos "ground" representando os fatos removidos do mundo em relação ao estado inicial;
- O: o conjunto de operadores do gênero, incluindo definições de especialização e decomposição;
- P: o valor inicial das propriedades dramáticas
- NE_{max}: O grau máximo de não determinismo permitido (número máximo de finais possíveis para o plano).

- Dados de retorno:
    - π: o plano, representando um capítulo, com estrutura de árvore, onde cada nó é um evento básico e cada aresta, uma possibilidade não determinística. Cada aresta tem associada a ela, um conjunto de fatos que determina as precondições necessárias para seguir por aquela ramificação.

1. **ObterPlano***(w,α,S_{corr},O,P,NE_{max})*
2. α_{norm} ← Normalizar*(α)*
3. *F* ← GeraConjuntoFormulas(α_{norm})
4. *(π,τ,N_{finais})* ← Planejador(*w,F,*S_{curr},*O,P,*NE_{max},*1*)
5. *sat* ← {aplicação de *τ* sobre *α*}
6. *sat* ← Simplifica(*sat*)
7. **Se** *sat = false*
8. **Retorne** *Falha*
9. **Retorne** *π*

Figura 13. Algoritmo inicial para o Temporal NDet-HTN

O algoritmo inicial *ObterPlano* (Figura 13) normaliza a fórmula para o início do planejamento (linha 2); gera o conjunto de fórmulas a verificar a partir da fórmula de entrada (linha 3); faz chamada para obtenção o plano (linha 4); verifica se a fórmula enviada ao planejador foi satisfeita (linhas 5 a 8); e retorna o plano obtido (linha 9).

A função *Normalizar(α)* normaliza a fórmula temporal *α* em uma forma normal sem negações e implicações (seção 3.6.3). A função *GeraConjuntoFormulas(*α_{norm}*)* gera um conjunto *F* que contém todos os elementos de α_{norm}, sendo que os elementos que são consequência lógica de α_{norm} são anotados como obrigatórios (seção 3.6.4). O procedimento *Planejador(w,*α_{norm}*,*S_{curr}*,O,*P_{inic}*,*NE_{max}*,1)* retorna uma tripla <*π,τ,*N_{finais}>, onde *π* é o plano, *τ* é uma substituição e N_{finais} é o número de finais alcançados pelo plano. A substituição *τ* é utilizada para verificar se a fórmula inicial enviada ao planejador foi satisfeita (linhas 5 a 8). O parâmetro de retorno N_{finais} não é utilizado no nível do algoritmo *ObterPlano*.

4.3 Algoritmo Principal de Planejamento

O algoritmo principal é o *Planejador* (Figura 14), cujos dados de entrada são os mesmo do *ObterPlano*, mas com a adição de um dado de entrada, NE_{in}, para o número

corrente de finais que o plano alcança (começa com o valor *1*) e de dois dados de retorno, que são:

- τ: uma substituição de fórmulas por valores *true* ou *false*, indicando quais fórmulas foram satisfeitas e quais falharam;
- $N_{endings}$: o número corrente de estados finais que o plano alcança.

O *Planejador* é um algoritmo recursivo cujo objetivo é gerar um plano (árvore de contingências) que obedece à HTN e satisfaz a fórmula temporal de entrada. Desse modo, o algoritmo consome as tarefas (operadores) da HTN, decompondo-os e especializando-os em suboperadores. Quando se depara com operadores básicos, o algoritmo leva em conta a satisfação da fórmula temporal corrente e trata o não determinismo, planejando para cada possibilidade de continuação a partir do operador básico em questão.

1. **Planejador**(w,F,S_{corr},O,P,NE_{max},NE_{in})
2. **Se** w é a HTN vazia
3. $\tau_{folha} \leftarrow$ VerificaFormulaNaFolha(F,P)
4. **Se** $\tau_{folha} = Falha$
5. **Retorne** (*Falha*,$\emptyset$,*0*)
6. **Retorne** (*Plano_Vazio*,τ_{folha},*0*)
7. **Escolha não deterministicamente** {$u \in w \mid u$ não tem predecessor em w}
8. /* *Impõe explicitamente restrição de ordem para que seja mantida caso o*
9. *operador associado ao vértice seja complexo* */
10. **Para cada** {$i \in w \mid i \neq u$}
11. Adicione, se não existente, uma ordem de restrição (u,i) a w
12. /* t'_u *é a tarefa, sem o modificador "attempt" (caso tenha), associada ao*
13. *vértice "u"* */
14. **Se** u for do tipo "*attempt*"
15. $t'_u \leftarrow$ *tarefa que a tentativa procura estabelecer*
16. *grau_tentativa* $\leftarrow$ {grau de tolerância da tentativa t'_u }
17. **Senão**
18. $t'_u \leftarrow t_u$
19. *grau_tentativa* $\leftarrow 0$
20. *válidos* $\leftarrow$ {(a,σ) | a é instância "ground" de um operador em O.
21. σ é uma substituição tal que $\sigma(t'_u) = nome(a)$}
22. **Se** *válidos* = $\emptyset$
23. **Retorne** (*Falha*,$\emptyset$,*0*)
24. **Escolha não deterministicamente** {(a,σ) $\leftarrow$ *válidos*}
25. **Se** a não é aplicável a S_{curr}
26. **Se** *grau_tentativa* > *0*
27. $w' \leftarrow w - \{u\}$
28. (π',τ,NE_{out}) $\leftarrow$ **Planejador**(w',F,S_{corr},O,P,NE_{max},NE_{in})
29. **Senão**
30. **Retorne** (*Falha*,$\emptyset$,*0*)
31. **Se** t'_u é um operador básico
32. (π,τ,NE_{out}) $\leftarrow$ **ExpOpBasico**((a,σ),t'_u,S_{corr},w,F,P,NE_{max},NE_{in})
33. **Senão Se** t'_u é um operador abstrato
34. (π,τ,NE_{out}) $\leftarrow$ **ExpOpAbstrato**((a,σ),t'_u,S_{corr},w,F,P,NE_{max},NE_{in})
35. **Senão Se** t'_u é um operador composto
36. (π,τ,NE_{out}) $\leftarrow$ **ExpOpComposto**((a,σ),t'_u,S_{corr},w,F,P,NE_{max},NE_{in})
37. **Retorne** (π,τ,NE_{out})

Figura 14. Algoritmo principal do Temporal NDet-HTN

A Figura 14 apresenta o algoritmo principal do planejador Temporal NDet-HTN. Como primeiro passo (linhas 2 a 6), tem-se uma condição de parada da recursão do planejador, que verifica se o planejamento chegou a uma HTN vazia (indicando uma possível incorporação de um final ao plano). Se este for o caso, é necessário verificar as fórmulas para a folha e, nesse ponto, a verificação leva em conta os operadores de estado (Seção 3.6.6). A verificação é feita através da chamada

VerificaFormulaNaFolha(F,P), que retorna *Falha* caso o conjunto de fórmulas *F* não possa ser satisfeito na folha considerando o valor *P* corrente das propriedades dramáticas, ou, caso contrário, uma substituição τ que indica quais fórmulas foram ou não satisfeitas. O pseudocódigo do procedimento *VerificaFormulaNaFolha(F,P)* é apresentado na seção 4.4. Caso não seja possível satisfazer a fórmula temporal na folha, o algoritmo *Planejador* retorna *Falha* e, caso contrário, retorna o plano vazio para ser concatenado ao final do plano em questão, a substituição τ obtida, e o valor *0* (número de finais alcançados a partir da folha).

Quando a HTN não é vazia, o terceiro passo (linhas 7 a 11) do algoritmo é obter uma tarefa da HTN para cumprir. O algoritmo escolhe não deterministicamente uma tarefa sem predecessores. É importante notar que este ponto de obtenção de tarefas sem predecessor, e todos os outros onde há uma escolha não determinística, é um ponto de *backtracking*. Quando o planejamento falha em um ponto, ele retorna a recursão para o último ponto de backtracking onde ainda havia uma escolha a ser feita. Além de escolher uma tarefa sem predecessor, também é necessário, para o caso de operadores complexos, adicionar restrições de ordem de modo a garantir que a tarefa será de fato a próxima a ser decomposta. Como quarto passo (linhas 10 a 19), o algoritmo verifica se a tarefa a ser cumprida é uma tentativa e, se for o caso, obtém a tarefa sem a indicação de "*attempt*" e também o grau da tentativa. Para tarefas que não são tentativas, o algoritmo continua com a tarefa a cumprir e o grau de tentativa é *0*.

Após a obtenção da tarefa a cumprir e a consideração sobre uma possível tentativa, o algoritmo deve obter uma ação para cumprir a tarefa. O quinto passo (linhas 20 a 24) obtém uma instância de um operador (ação) cujo nome seja unificável com o nome da tarefa a cumprir. Essa unificação é feita através de σ, que corresponde a substituições de variáveis para instanciar os operadores e obter as ações. Mais de uma ação pode ser candidata a cumprir a tarefa, e, portanto, o conjunto *válidos* é retornado, guardando cada ação e a respectiva substituição que foi aplicada. Se não há ações capazes de cumprir a tarefa, o algoritmo retorna *Falha* e, caso contrário, escolhe não deterministicamente um elemento de *válidos* com a ação a considerar. As instanciações feitas por σ são propagadas no plano já gerado e na HTN restante.

Após a obtenção da ação para cumprir a tarefa corrente, como sexto passo (linhas 25 a 30) o algoritmo verifica se a ação é aplicável no estado corrente. Caso a não seja aplicável, a tarefa pode ser uma tentativa, e, portanto, passível de ser desconsiderada. Se a tarefa não for aplicável e não for uma tentativa (fato indicado pelo

valor *0* da variável *grau_tentativa*), então o algoritmo retorna *Falha*. Se o algoritmo está lidando com uma tentativa, esta é desconsiderada e o planejamento segue recursivamente.

Como último passo (linhas 31 a 36), o algoritmo trata os diferentes tipos de operadores (básico, abstrato e composto), chamando, para cada tipo, um procedimento diferente. O algoritmo obtém como retorno, em qualquer um dos casos, uma tripla$<\pi,\tau,NE_{out}>$, onde, para o evento associado ao operador corrente: π é um plano; τ é uma substituição de fórmulas por valores *true* ou *false*, indicando quais fórmulas falharam e quais foram satisfeitas; e NE_{out} é o número de estados fiais alcançados pelo subplano. Como último passo (linha 37), o algoritmo retorna a tripla $<\pi,\tau,NE_{out}>$.

4.4 Satisfação de Fórmulas nas Folhas

Quando o algoritmo *Planejador* (Figura 14) chega a uma HTN vazia (linhas 2 a 6), é feita uma chamada ao procedimento *VerificaFormulaNaFolha(F,P)* (Figura 15).

```
1. VerificaFormulaNaFolha(F,P)
2.      τ ← Ø
4.      Para cada α ∈ F
5.              /* α_resid só pode ser true ou false*/
6.              α_resid ← ResultadoNaFolha(α,P)
7.              α_resid ← Simplifica(α_resid)
8.              Se α_resid = false
9.                      Se α é obrigatória
10.                             Retorne Falha
12.                     τ ← τ ∪ {α → false}
13.             Senão
14.                     τ ← τ ∪ {α → true}
15.     Retorne τ
```

Figura 15. Verificação de fórmulas em uma folha (final do plano).

O primeiro passo (linha 2) da verificação de fórmulas no final do plano é inicializar a substituição τ que será retornada com $\emptyset$. O segundo passo (linhas 4 a 14) é um *loop* em cima de todas as fórmulas α pertinentes ao conjunto F de fórmulas a verificar. Dentro do *loop*, o primeiro passo (linha 6) é obter, para a fórmula na folha, o resultado e o segundo passo (linha 7) é simplificar a fórmula, pois ela pode estar, por exemplo, como uma conjunção *true or true*, que deveria ser simplificada simplesmente para *true*. Quando se chega à folha do plano, é possível saber se qualquer fórmula tem

valor *true* ou *false*. O valor é obtido através das tabelas I e II abaixo. Além disso, nenhuma fórmula obrigatória pode falhar (ter valor *false*) e somente fórmulas existenciais podem ser ditas satisfeitas. A indicação de fórmulas universais que foram satisfeitas só pode ser feita após a avaliação de todos os filhos de um evento, ou seja, na parte do planejamento que lida com eventos básicos. Se o procedimento não falhar, após o término do *loop*, todas as fórmulas (com exceção de fórmulas universais com valor *true*) $\alpha \in F$ estão indicadas como satisfeitas ou não através da substituição τ.

Tabela 1. Tabela para verificação de fórmulas no final do plano.

Fórmula na Folha (α)	**ResultadoNaFolha(α,P)**
AG β; *EG β*	*true*
AF β; *EF β;*	*false*
A β U γ; *E β U γ*	*false*
A end β; *E end β*	*/* É necessário verificar a Tabela 2 */* ResultOpEnd(*β*,*P*)

Tabela 2. Tabela para verificação de fórmulas no final do plano.

Fórmula(α)	**ResultOpEnd(α,*P*)**
β and γ	ResultOpEnd*(β,P) and* ResultOpEnd*(γ,P)*
β or γ	*ResultOpEnd(β,P) or ResultOpEnd(γ,P)*
AG β; EG β; AF β; EF β; A end β; E end β	*ResultOpEnd(β,P)*
A β U γ; E β U γ	*ResultOpEnd(γ,P)*
{a > b}; {a < b}; {a ≥ b}; {a ≤ b}; {a ≡ b}; {a ≠ b}	*A restrição deve estar satisfeita segundo P.*

A e a são utilizadas para obter o valor (*true* ou *false*) de fórmulas no final do plano (em uma folha). A coluna da esquerda é a fórmula cujo valor se deseja saber e a coluna da direita é uma fórmula que indica o valor. É importante notar que há uma recursão para obter o valor das fórmulas, onde um valor obtido na coluna da direita da tabela pode ter de ser usado recursivamente na coluna da esquerda até que se chegue a um valor *true* ou *false*. Além disso, se a fórmula tiver um operador de estado *end*, a tabela II

tem de ser usado para continuar obtendo o valor. Como exemplo, considere as fórmulas exemplo e valor das propriedades contínuas:

EF ({alegria > 10} and A end ({alegria >10} and AF {medo >20})) (I)

E end ({alegria > 10} and {medo ≥ 25}) (II)

P = {(alegria = 15), (medo = 25)}

Neste caso, primeiramente a fórmula é avaliada segundo a , que retorna imediamente o valor *false* para a fórmula I não importando o valor de nenhuma propriedades contínua. O fato de, por exemplo, *alegria = 15* satisfazer a primeira restrição *{alegria > 10}* da fórmula I nem chega a ser avaliado, assim como todas as outras fórmulas internas. Para a fórmula II, a retorna o valor *ResultOpEnd(β,P)*, a ser avaliado recursivamente através da , onde *β* é a conjunção *{alegria > 10} and {medo ≥ 25}*. A nos indica que devemos obter o valor da fórmula *{alegria > 10}* e depois da fórmula *{medo ≥ 25}* e, então, juntá-los em uma conjunção. Restrições são tratadas na última linha da e devem ser avaliadas segundo o valor das propriedades dramáticas em *P*. Nesse caso, ambas as restrições obterão valor *true* e o resultado na folha final retornado será *true and true*, que no algoritmo *VerificaFormulaNaFolha* (Figura 15), na linha 7, é simplificado para *true* somente.

4.5 Tratamento de Operadores Abstratos

O tratamento de operadores abstratos é feito através da escolha não determinística de uma especialização. Se não houver especializações, o planejamento falha neste ponto. Após a escolha da especialização, o operador abstrato é substituído por ela na HTN, mantendo-se as restrições de ordem que se aplicavam sobre ele. Um operador abstrato que também é composto e cuja especialização também é um operador composto, leva à herança da sua sub-HTN pela especialização, que pode, por sua vez, adicionar novas ordens de restrição à sub-HTN do operador que especializa. Com a substituição do operador, o planejamento segue recursivamente para obter o plano. O pseudocódigo do algoritmo para o algoritmo é apresentado na figura Figura 16 abaixo. O único parâmetro que segue modificado em relação aos parâmetros que vieram do *Planejador* é a nova HTN.

```
1. ExpOpAbstrato((a,σ),u,S_corr,w,F,P,NE_max,NE_in)
2.     especializações ← {e | e é instância "ground" (sob σ) de uma
3.                         especialização de a em O e e é aplicável a S_corr}
4.     Se especializações = Ø
5.         Retorne Falha
6.     Escolha Não Deterministicamente {e ∈ especializações}
7.     w' ← {substituição de a por e em σ(w)}
8.     Retorne Planejador(w',F,S_corr,O,P,NE_max,NE_in)
```

Figura 16. Tratamento de operadores abstratos no Temporal NDet-HTN.

4.6 Tratamento de Operadores Compostos

O tratamento de operadores compostos é feito através da substituição do operador composto pela sua sub-HTN. Essa decomposição, representada é feita de maneira semelhante à explicada na seção 2.1.2.

A decomposição $\delta(\sigma(w),\sigma(u),\sigma(a))$ gera uma nova HTN, primeiro retirando o nó de tarefa para *a*, e, depois, forçando que os nós de tarefa para todas as subtarefas da sub-HTN herdem as restrições de ordem de *a*. Além disso, através da substituição σ, a decomposição retorna a nova HTN *w'* já com variáveis instanciadas. O pseudocódigo do algoritmo é apresentado na figura Figura 17 abaixo. O único parâmetro que segue modificado em relação aos parâmetros que vieram do *Planejador* é a nova HTN.

```
1. ExpOpComposto((a,σ),u,S_corr,w,F,P,NE_max,NE_in)
2.     w' ← δ(σ(w),σ(u),σ(a))
3.     Retorne Planejador(w',F,S_corr,O,P,NE_max,NE_in)
```

Figura 17. Tratamento de operadores compostos no Temporal NDet-HTN

4.7 Tratamento de Operadores Básicos

4.7.1 Visão Geral

O tratamento de operadores básicos determinísticos e não determinísticos é feito pelo mesmo algoritmo da Figura 18. De modo geral, percorrem-se os finais, obtendo o conjunto de fórmulas a verificar na chamada recursiva para um final e verificando se as fórmulas foram satisfeitas quando a recursão volta. Após o término do planejamento de

todos os filhos, as fórmulas universais que não falharam são explicitamente indicadas como resolvidas na substituição de retorno.

O algoritmo será explicado de maneira "*top down*". Isto é, primeiro se explica por alto todos os passos e depois, há uma seção para cada um dos principais tratamentos do algoritmo: obtenção de fórmulas residuais que cada subplano terá de satisfazer; filtragem da subfórmula para obter somente as que concernem ao final corrente para o qual se irá planejar; e a verificação da satisfação das fórmulas quando o planejamento retorna.

1. **ExpOpBasico**((*a*,σ),*u*,S_{corr},*w*,*F*,*P*,NF_{max},NF_{in})
2. $w' \leftarrow \sigma(w - \{u\})$
3. *finais* ← ObtemFinaisAtivos(*a*,S_{corr})
4. *num_finais* ← {número de finais em *finais*}
5. **Se** *num_finais = 0*
6. **Retorne** *Falha*
7. $NF_{corr} \leftarrow NF_{in}$ + *num*_finais - *1*
8. **Se** $NF_{corr} > NF_{max}$
9. **Retorne** (*Falha*,*Ø*,*0*)
10. *L_π'* ← *Ø /* Lista de contingências é inicializada */*
11. $<L_{resid\text{-}pairs},\tau_{pai}>$ ← FormulasResiduais($\sigma(a)$,*F*,S_{corr},*P*)
12. */* Se alguma fórmula obrigatória falhou */*
13. **Se** τ_{pai} = *Falha*
14. **Retorne** (*Falha*,*Ø*,*0*)
15. S_{det} ← {aplicação dos efeitos determinísticos de $\sigma(a)$ em S_{corr}}
16. **Para cada** *final* **em** *finais*
17. EF_{ndet} ← {efeitos não determinísticos de $\sigma(a)$ para o *final*}
18. S_{ndet} ← {aplicação de EF_{ndet} em S_{det}}
19. F_{resid} ← FiltraPorFinal($L_{resid\text{-}pairs}$, *final*)
20. P_{sub} ← AtualizaValorPropriedades($\sigma(a)$,s_{end},P)
21. $<\pi_{sub},\tau_{resid},NE_{sub}>$ ← **Planejador**(*w*,F_{resid},S_{init},S_{ndet},*O*,P_{sub},NE_{max},NE_{corr})
22. **Se** π_{sub} = *Falha*
23. **Retorne** (*Falha*,*Ø*,*0*)
24. $<L_{resid\text{-}pairs},\tau_{filho}>$ ← VerificaFormula(*final*,$L_{resid\text{-}pairs}$,τ_{resid})
25. */* Se alguma fórmula obrigatória falhou */*
26. **Se** τ_{filho} = *Falha*
27. **Retorne** (*Falha*,*Ø*,*0*)
28. $\tau_{pai} \leftarrow \tau_{pai} \cup \tau_{filho}$
29. $NE_{corr} \leftarrow NE_{corr} + NE_{sub}$ - 1
30. $L_\pi' \leftarrow L_\pi' \cup \{<EF_{ndet},\pi_{sub}>\}$
31. $\pi \leftarrow \sigma(a).L_\pi'$
32. **Return** (π,τ_{pai},NE_{corr})

Figura 18. Tratamento de operadores básicos no Temporal NDet-HTN.

O primeiro passo do algoritmo (linha 2) é remover da HTN a tarefa que o planejador tentará cumprir. O segundo passo (linha 3) é a obtenção de finais ativos para o algoritmo, isto é, os finais que são alcançáveis no autômato associado a *a* segundo o estado corrente da história S_{corr}. É importante notar que há a necessidade de haver pelo menos um final ativo, caso contrário o planejamento retorna *Falha*. Como terceiro passo (linhas 4 a 9), o algoritmo verifica se a inserção da ação *a* no plano irá aumentar o nível de não determinismo ao ponto de ultrapassar o grau máximo de não determinismo indicado pelo parâmetro de entrada NF_{max}. Esta verificação é feita atualizando o número corrente de estados finais alcançados pelo plano (NF_{corr}) como a soma do número de

finais que o plano alcança (NF_{in}) com o número de finais (*num_finais*) do evento subtraído de *1*, onde esse *1* simboliza a ligação já existente com o pai do evento. Se NF_{corr} ultrapassa NF_{max}, o planejamento retorna *Falha*. O terceiro passo do algoritmo (linha 10) é a atualização do estado corrente da história com a aplicação dos efeitos determinísticos do operador. Em seguida, no quarto passo (linha 11), o algoritmo inicializa o plano π a ser retornado.

O quinto passo do algoritmo (linha 12) é a obtenção das fórmulas residuais através da função *FormulasResiduais(*$\sigma(a)$*,F,*S_{corr}*,P)*, que recebe como entrada uma ação *a*, um conjunto de fórmulas *F*, o estado corrente da história S_{corr} e o valor corrente das propriedades dramáticas *P*, retornando uma dupla $<L_{resid\text{-}pairs},\tau_{pai}>$, onde $L_{resid\text{-}pairs}$ é um conjunto de pares $<\alpha,\alpha_{resid}>$, guardando a fórmula residual de cada fórmula do conjunto *F*, e τ_{pai} é uma substituição que anota quais fórmulas foram resolvidas e quais falharam. O pseudocódigo e uma explicação detalhada desse procedimento são dados na seção 4.7.2. O próximo passo do algoritmo (linhas 14 e 15) é verificar se alguma fórmula obrigatória em *F* falhou, o que é feito através da verificação da substituição de retorno do procedimento *FormulasResiduais(*$\sigma(a)$*,F,*S_{corr}*,P)*.

O sétimo passo do algoritmo (linhas 16 a 30) é um *loop* que irá planejar para cada final tentando obter um subplano para cada filho de forma a satisfazer as fórmulas residuais obtidas. O primeiro passo do loop (linhas 17 e 18) é atualizar o estado da história com os efeitos não determinísticos do evento. O segundo passo (linha 19) é filtrar as fórmulas residuais para obter somente aquelas que se referem ao final para o qual se irá planejar. A explicação do procedimento de filtragem é dada na seção 3.6.2. Como terceiro passo (linha 20), o algoritmo atualiza o valor corrente das propriedades dramáticas de acordo com a combinação linear (seção 3.4.2) presente no estado final do autômato referente ao final para o qual está sendo feito o planejamento. O quarto passo do algoritmo (linha 21) é uma chamada ao algoritmo *Planejador* que irá obter um subplano a partir do final que está sendo considerado. Se o planejador falha (linhas 22 e 23), o algoritmo retorna *Falha*. Caso contrário, o próximo passo é verificar se as fórmulas que deveriam ser satisfeitas pelo subplano, realmente foram satisfeitas (linhas 24 a 27). Essa verificação faz uma chamada ao procedimento *VerificaFormula(F,*$L_{resid\text{-}pairs}$*,*τ_{resid}*)*, que percorre o conjunto $L_{resid\text{-}pairs}$ e utiliza a substituição τ_{resid} retornada pelo subplano para saber se todas as fórmulas obrigatórias em *F* estão satisfeitas, e, também, possivelmente obter fórmulas que foram satisfeitas para o pai. O retorno da função *VerificaFormula(F,*$L_{resid\text{-}pairs}$*,*τ_{resid}*)* é uma dupla $<L_{resid\text{-}pairs},\tau_{filho}>$, onde $L_{resid\text{-}pairs}$ é o

$L_{resid\text{-}pairs}$ que foi passado como parâmetro, mas agora atualizado de acordo com as fórmulas satisfeitas, e τ_{filho} é a substituição que contém quais fórmulas foram satisfeitas ou não pelo final (filho) que foi planejado. O pseudocódigo do procedimento *VerificaFormula*($L_{resid\text{-}pairs}$,τ_{resid}) juntamente a uma explicação detalhada são dados na seção 4.7.4. Finalmente, se a verificação falhou, o algoritmo retorna *Falha*; a verificação só falha se uma fórmula que seja obrigatória falhar.

O sexto passo do *loop* (linha 28) é atualizar a substituição τ_{pai} que será retornada para quem chamou a expansão do operador básico. A atualização une a substituição τ_{pai} com a substituição τ_{filho} obtida pelo planejamento para o final corrente no *looop*. O sétimo passo do *loop* (linha 29) atualiza o número de finais corrente alcançáveis pelo plano (NE_{corr}). Como oitavo, e último, passo do *loop* (linha 30), o algoritmo atualiza a estrutura de plano corrente com a contingência (efeitos não determinísticos do operador para o final) e o subplano retornado.

Finalmente, como último passo (linhas 31 e 32), o algoritmo concatena a ação instanciada $\sigma(a)$ com a estrutura de contingências do plano que trata cada final $L.\pi'$ a fim de formar um plano π que é retornado na tripla $<\pi,\tau_{pai},NE_{corr}>$.

4.7.2 Obtenção de Fórmulas Residuais

A obtenção de fórmulas residuais é uma parte essencial do algoritmo de tratamento de operadores básicos e feito pelo procedimento *FormulasResiduais*($\sigma(a)$,*F*,*Scorr*,*P*). Este procedimento serve para obter as fórmulas que devem valer após certo evento para que certo conjunto de fórmulas seja satisfeito, levando em conta o estado corrente da história e das propriedades dramáticas. Além disso, o procedimento também retorna quais fórmulas foram satisfeitas pelo próprio evento. O pseudocódigo para o procedimento *FormulasResiduais*($\sigma(a)$,*F*,*Scorr*,*P*) é apresentado na Figura 19 abaixo.

```
1.  FormulasResiduais(a,F,S_corr,P)
2.      Se F = Ø
3.          Retorne <Ø,Ø>
4.      L_resid-pairs ← Ø
5.      τ ← Ø
6.      Para cada α ∈ F
7.          α_resid ← ConsultaFormulaResidual(a,S_corr,P,α)
8.          Se α_resid = false
9.              Se α é obrigatório
10.                 Retorne (Ø,Falha)
11.             τ ← τ ∪ {α → false}
12.         Senão Se α_resid = true
13.             τ ← τ ∪ {α → true}
14.         Senão Se α é obrigatório
15.             MarcaObrigatorios(α_resid)
16.         Se α_resid ≠ false E α_resid ≠ true
17.             L_resid-pairs ← L_resid-pairs ∪ {<α,α_resid>}
18.     Retorne <L_resid-pairs,τ>
```

Figura 19. Algoritmo para obtenção de fórmulas residuais.

O algoritmo *FormulasResiduais(a,F,S_{corr},P)* (Figura 19) recebe como entrada uma ação *a*, um conjunto de fórmulas a serem satisfeitas *F*, o estado corrente da história S_{corr}, e o valor corrente das propriedades dramáticas *P*. O retorno do algoritmo é um par de conjuntos $<L_{resid\text{-}pairs},\tau>$, onde $L_{resid\text{-}pairs}$ é um conjunto de pares $<\alpha,\alpha_{resid}>$ e τ é um conjunto de substituições do tipo $\alpha \rightarrow true$ ou $\alpha \rightarrow false$. Um par (α,α_{resid}) mapeia uma fórmula α à sua fórmula residual α_{resid}.

O primeiro passo do algoritmo (linhas 2 e 3) é verificar se há fórmulas para verificar, e, caso não haja, retorna um par de conjuntos vazios. O segundo passo (linhas 4 e 5) é inicializar as duas estruturas de retorno $L_{resid\text{-}pairs}$ e τ. Como terceiro passo (linhas 6 a 19), o algoritmo tem um *loop* que constrói as estruturas de retorno.

O primeiro passo do *loop* é obter a fórmula residual α_{resid} através do procedimento *ConsultaFormulaResidual(a,S_{corr},P,α).* Esse procedimento obtém uma fórmula em uma linguagem especial que permite a indicação do final para o qual a fórmula deve valer. Como exemplo, suponha uma consulta *ConsultaFormulaResidual(α,S_{corr},P,α)* para a seguinte fórmula de entrada:

$$EF\ (\beta\ and\ AG\ \gamma)$$

Neste caso, suponha ainda que estamos consultando para um evento que tem dois finais: f_1 e f_2, onde por f_1, β vale e por f_2 não. Assim, para o final f_1 poderíamos satisfazer

AG γ e teríamos a fórmula de entrada satisfeita. Se para o final f_1 a fórmula não fosse satisfeita, então, para o final f_2, ainda seria necessário satisfazer *EF (β and AG γ)*. Nesse exemplo, se desconsideram detalhes do estado corrente e dos valores de propriedades dramáticas que levaram a consulta a retornar tal valor. Deste modo, para a consulta com a fórmula de exemplo acima, poderíamos ter a seguinte fórmula residual:

$$((f_1,EF\ (\beta\ and\ AG\ \gamma))\ or\ (f_1,AG\ \gamma))\ or\ (f2,EF\ (\beta\ and\ AG\ \gamma)))$$

A fórmula acima contém a informação de o que é preciso satisfazer, em relação aos finais f_1 e f_2 do evento, a fim de que a fórmula de entrada *EF (β and AG γ)* seja satisfeita. Repare que se pode, pelo final f_1, satisfazer tanto a fórmula de entrada como também a fórmula *AG γ*, que surgiu porque houve a possibilidade de *β* valer em um caminho para o final f_1. Todavia, se o subplano pelo final f_1 não satisfizer nenhuma das duas fórmulas, o subplano através do final f_2 terá de satisfazer a fórmula de entrada a fim de que a fórmula residual seja satisfeita.

Imagine outro exemplo de consulta, desta vez com a fórmula de entrada:

$$AF\ (\beta\ and\ EG\ \gamma)$$

Poderíamos ter uma consulta que retornasse a fórmula residual:

$$((f_1,AF\ (\beta\ and\ AG\ \gamma))\ or\ (f_1,EG\ \gamma))\ and\ (f_2,AF\ (\beta\ and\ AG\ \gamma))$$

Este caso acima poderia ocorrer caso o evento satisfizesse *β*, por exemplo, em um estado inicial, que é ancestral dos estados finais f_1 e f_2, e *γ* valha durante todo o evento a partir do ponto onde vale *β* para o caminho até o final f_1, mas que *γ* não valha no caminho até f_2. Repare que pelo final f_1 a fórmula que tem de valer é "mais fraca", pois basta encontrar *EG γ* e a disjunção está satisfeita. No caso de seguir pelo final f_2, ainda seria necessário satisfazer a fórmula de entrada *AF (β and EG γ)*. Todavia, o *and* une as fórmulas para os dois finais, formando uma conjunção, indicando que as fórmulas para ambos os finais devem ser satisfeitas para que a fórmula de entrada esteja satisfeita.

A consulta por uma fórmula residual também pode retornar *false* ou *true*. Imagine uma fórmula *AG β*, desse modo, basta que o evento não possa satisfazer *β* para que a fórmula residual seja *false*. Neste caso, o retorno não é uma conjunção de duplas com os finais e a fórmula *false*, mas uma forma simplificada para *false* somente. Além disso, uma fórmula do tipo *EF α* pode ser satisfeita, bastando para isso que o evento possa ter um ponto onde *α* valha. Neste caso, a fórmula residual retornada é *true*.

É importante notar que aqui se toma a liberdade de falar em satisfação e simplificação de fórmulas que não são exatamente fórmulas da linguagem CTL estendida. Afinal, as fórmulas residuais são disjunções e conjunções em cima de pares *<final,formula>*. Todavia, de maneira informal diz-se que um par pode ser simplificado para *true*, caso sua fórmula também o possa, e o mesmo vale para *false*. Deste modo, se aplicarmos uma substituição τ = *{AF {alegria > 10} → verdadeiro}* na fórmula $(f_1$*,AF {alegria > 10}) or (*f_2*,AF {medo > 20})* a fórmula se transforma em $(f_1$*,true) or (*f_2*,AF {medo > 20})*, que pode ser simplificada para *true* somente.

Após a consulta pela fórmula residual, o algoritmo segue para criar a substituição τ e a lista de pares $L_{resid\text{-}pairs}$ que serão retornados. Primeiro, é necessário verificar se alguma fórmula obrigatória falhou (linhas 8 a 10), e, caso alguma tenha falhado, retornar *Falha*. Em seguida (linha 11), qualquer fórmula α com residual *false* leva à adição, em τ, de uma nova substituição $\alpha \rightarrow$ *false* a fim de indicar que α falhou. Uma fórmula que foi satisfeita (linhas 12 e 13) leva à adição, em τ, de uma substituição $\alpha \rightarrow$ *true* a fim de indicar o fato. Em seguida (linhas 14 a 15), o algoritmo marca fórmulas como obrigatórias, através do procedimento *MarcaObrigatorios(*α_{resid}*)* (seção 3.6.4). Um elemento α_{elem} de α_{resid} é marcado como obrigatório em *MarcaObrigatorios(*α_{resid}*) sse* α_{elem} é consequência lógica de α_{resid}. Como próximo passo (linhas 16 e 17), fórmulas que não tenham fórmulas residuais *true* ou *false* são adicionadas ao conjunto $L_{resid\text{-}pairs}$. Finalmente, o algoritmo retorna o par de conjuntos $<L_{resid\text{-}pairs},\tau>$.

4.7.3 Filtragem Por Final

A consulta por fórmulas residuais retorna um conjunto $L_{resid\text{-}pairs}$ que contém fórmulas residuais a serem satisfeitas em todos os finais. Todavia, o planejamento é feito final a final, e, assim, é necessário filtrar desse conjunto $L_{resid\text{-}pairs}$, as fórmuals que são referentes somente ao final para o qual se deseja planejar.

A filtragem é um procedimento simples que retorna, para certo final *f*, as fórmulas α_i que estão em pares (f, α_i) como parte dos segundos argumentos dos pares pertinentes a $L_{resid\text{-}pairs}$. Por exemplo, considere o conjunto $L_{resid\text{-}pairs}$:

$$\{ \quad (\alpha_1, ((f_1,EF\ (\beta\ and\ AG\ \gamma))\ or\ (f_1,AG\ \gamma))\ or\ (f2,EF\ (\beta\ and\ AG\ \gamma)))),$$

$$(\alpha_2, ((f_1,AF\ (\beta\ and\ AG\ \gamma))\ or\ (f_1, EG\ \gamma))\ and\ (f_2,AF\ (\beta\ and\ AG\ \gamma)))\ \}$$

Neste caso, uma filtragem $FiltraPorFinal(L_{resid\text{-}pairs},f_1)$ retornaria o seguinte conjunto de fórmulas F_{resid}:

{ EF (β and AG γ), AG γ, AF (β and AG γ)), EG γ }

4.7.4 Verificação de Fórmulas

Após o planejamento por um final, o tratamento de eventos básicos precisa verificar se as fórmulas que precisavam são satisfeitas pelo subplano, realmente o são. Isto é feito através de uma chamada ao procedimento *VerificaFormula*($L_{resid\text{-}pairs}$,τ_{resid}), que visa descobrir quais fórmuals de $L_{resid\text{-}pairs}$ foram satisfeitas e quais falharam, considerando a substituição τ_{resid} retornada pelo subplano. O retorno do procedimento é um par ($L_{resid\text{-}pairs}$,τ), onde: $L_{resid\text{-}pairs}$ é referente ao conjunto $L_{resid\text{-}pairs}$ de entrada, mas possivelmente com fórmulas simplificadas; e τ é uma substituição indicando quais fórmuals foram satisfeitas e quais falharam em $L_{resid\text{-}pairs}$. O pseudocódigo para o procedimento é mostrado na Figura 20 abaixo.

```
1. VerificaFormula(final,L_resid-pairs,τ_resid)
2.     Se L_resid-pairs = Ø
3.         Retorne <Ø,Ø>
4.     τ ← Ø
6.     Para cada par_resid <α,α_resid> em L_resid-pairs
7.         α_resid ← {aplicação de τ_resid sobre as fórmulas para final em α_resid}
8.         α_resid ← Simplifica(α_resid)
9.         Se α_resid = false
10.            Se α é "obrigatório"
11.                Retorne <Ø,Falha>
12.            τ ← τ ∪ {α → false}
13.            Remova par_resid de L_resid-pairs
14.        Senão Se α_resid = true
15.            τ ← τ ∪ {α → true}
16.            Remova par_resid de L_resid-pairs
17.    Retorne <L_resid-pairs,τ>
```

Figura 20. Algoritmo para verificação de fórmulas.

O primeiro passo do algoritmo da Figura 20 é retornar uma estrutura vazia caso não haja fórmulas residuais a considerar. Como segundo passo (linha 4), inicializa-se a substituição τ a retornar com vazia. O terceiro passo (linhas 5 a 16) é um *loop*, operando sobre cada fórmula do conjunto *F*, e onde se constrói a substituição τ a retornar, e, possivelmente, simplifica-se $L_{resid\text{-}pairs}$.

No *loop*, o primeiro passo (linha 6) é obter o par referente a fórmula corrente sendo verificada. O Segundo passo (linha 7) é aplicar a substituição τ_{resid}, referente ao subplano, à fórmula residual α_{resid} do par corrente. Esta aplicação leva em conta somente

as fórmulas para o final passado como parâmetro de entrada. O terceiro passo (linha 8) é simplificar α_{resid}. Esta simplificação pode levar α_{resid} a *true* ou *false* e também a uma fórmula que ainda não pode ser tida como satisfeita ou não satisfeita. No caso em que o a satisfação da fórmula não pode ainda ser definida, nada é feito em relação a essa fórmula e o *loop* continua para a próxima fórmula em *F*.

Como quarto passo (linhas 9 a 11), o algoritmo retorna *Falha* caso encontre alguma fórmula obrigatória que não tenha sido satisfeita, ou seja, cuja fórmula residual seja *false*. O quinto passo (linha 12), é adicionar a τ a substituição $\alpha \rightarrow$ *false* a fim de indicar que a fórmula α falhou. Além disso, o sexto passo (linha 13) remove de $L_{resid\text{-}pairs}$ o par referente à fórmula que falhou. Como sétimo passo (linha 14 a 16), o algoritmo lida com fórmulas que foram satisfeitas, adicionado a substituição $\alpha \rightarrow$ *true* para a fórmula α satisfeita e removendo de $L_{resid\text{-}pairs}$ o par referente a α. Quando acaba de percorrer os pares em $L_{resid\text{-}pairs}$, o algoritmo retorna o par construído de conjuntos ($L_{resid\text{-}pairs}$,τ).

4.8 Otimização do Controle do Grau Máximo de Não Determinismo

O algoritmo Temporal NDet-HTN otimiza o NDet-HTN em um aspecto que é o controle do nível de não determinismo. Tal controle era feito no NDet-HTN pela contagem do número de estados finais atingidos a cada vez que o processo de planejamento retornava para um estado. Se este número fosse maior do que o grau máximo permitido, o planejamento falhava e continuava por *backtracking*. Todavia, para fins de controle do nível de não determinismo, não é estritamente necessário esperar o retorno do planejamento para um evento a fim de saber se este pode ser adicionado ao plano. Se for conhecido o número mínimo de estados finais atingidos pelo plano até o momento e também o número de ramificações que o evento corrente irá criar (o número de estados finais do evento), então é possível saber, antes de adicionar o evento ao plano, se o grau máximo de não determinismo está sendo ultrapassado. Por exemplo, com um grau máximo de não determinismo *grau_ndet*, se o plano até o momento já tem no mínimo *n* finais alcançáveis e o evento a ser adicionado tem *m* finais, então antes de adicionar o evento deve-se verificar se $n + m - 1 \leq grau_ndet$.

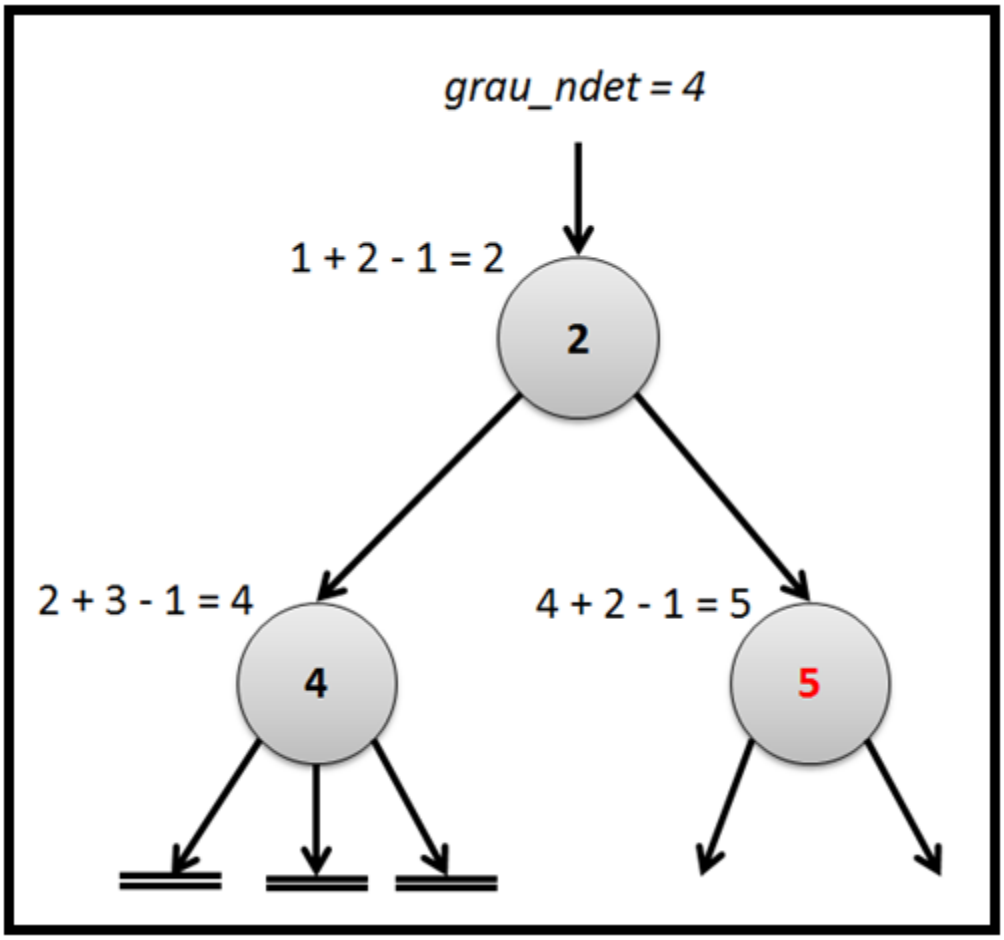

Figura 21. Controle do grau de não determinismo.

Na Figura 21 temos um exemplo de controle do grau de não determinismo, onde o grau máximo de não determinismo é *4* e as equações indicam que a soma do número mínimo de estados finais que o plano alcança com o número de estados finais do evento menos um é igual ao grau mínimo de não determinismo corrente. No evento mais à direita, o grau máximo é ultrapassado e o evento não é adicionado.

5 Avaliação

5.1 Visão Geral

A avaliação do protótipo estendeu o gênero definido por SILVA (2010) a fim de adicionar os autômatos associados a cada evento básico. Em cima destes autômatos foi feita uma etapa de preprocessamento, que gerou uma estrutura de consulta (GOTTIN, 2013) para o planejador Temporal NDet-HTN. Na estrutura de preprocessamento gerada, nenhum dos autômatos considera as precondições de ações, mas somente o tempo de execução de cada microação. Desta forma, foi possível testar o planejamento com diferentes fórmulas temporais de entrada.

A codificação do algoritmo Temporal NDet-HTN é feita na linguagem Prolog, utilizando a sintaxe do SICStus Prolog, assim como o NDet-HTN. Houve diversas modificações técnicas na codificação que levaram a uma melhoria considerável no tempo de execução em relação ao NDet-HTN. Dentre elas, o uso extensivo de listas ordenadas para aumentar a eficiência de procedimentos de subtração e união de listas, e verificação de pertinência; listas de diferença; e a utilização de "cuts" (evitando *backtrackings* desnecessários). Com base nessas modificações, gerou-se versão do NDET-HTN otimizada, que chamaremos daqui por diante de NDET-HTN*. Em cima do NDET-HTN* foram incorporadas as modificações para o tratamento das fórmulas temporais sobre propriedades dramáticas contínuas, obtendo-se o Temporal NDet-HTN. Desta forma, além de se poder comparar o Temporal NDet-HTN com o NDet-HTN, também se pode comparar o mesmo à versão otimizada desse algoritmo, NDET-HTN*, de modo a se ter ideia exata do impacto do processamento das fórmulas temporais no processo.

Os testes foram feitos em uma máquina Intel i5-2400 64 (1.30 GHz) com 6 GB de RAM e Windows 7 Professional. Quando se menciona *tempo médio*, faz-se referência a um tempo como média aritmética de 10 execuções. A versão do SICStus Prolog utilizada foi a 4.7.1, que inclui possibilidade de utilização de *Constraint Logic Programming* (CLP).

No caso de testes somente do planejamento não determinístico (sem a parte de planejamento temporal), as melhorias na implementação fizeram com que os tempos do NDet-HTN* ficassem bem mais baixos dos que os do NDet-HTN. Por exemplo, o teste para gerar uma história completa, composta de 3 capítulos e 21 eventos, mostrou um tempo de geração de *0,847 s* para o NDet-HTN e de *0,012 s* para o NDet-HTN*.

A primeira parte da avaliação diz respeito à comparação do algoritmo original NDet-HTN com o Temporal NDet-HTN, sem forçar a resolução de fórmulas temporais, ou seja, a fórmula que é passada para ser satisfeita é *true*. A ideia é garantir que, em todos os testes propostos por (SILVA, 2010) para o NDet-HTN, o Temporal NDet-HTN também funciona. A segunda parte da avaliação consiste de mostrar o funcionamento do planejamento temporal, com as diferentes respostas e tempo de planejamento.

O gênero utilizado para a avaliação é uma extensão do que foi proposto em (SILVA, 2010) e, assim, também é uma extensão do gênero "Espadas & Dragões". O contexto utilizado é o mesmo que em (SILVA, 2011). Nas avaliações, as chamadas feitas ao Temporal NDet-HTN foram feitas passando como argumento o estado corrente, a HTN, e o grau máximo de não determinismo.

5.2 Avaliação do Planejamento HTN Não Determinístico

5.2.1 Especialização de Operadores Abstratos

O primeiro teste consiste em verificar a capacidade do de buscar especializações para um operador, no caso, *do_malefic_sound(VIL)*, em que um vilão produz um som maléfico, e que pode ser especializado em *roar(VIL)* (caso o vilão seja um dragão) ou em *do_malefic_sound(VIL)* (caso o vilão seja uma pessoa).

Parâmetros de entrada:

- Estado corrente:
  - Fatos+: [];
  - Fatos−: [].
- HTN: U = [(t1,do_malefic_sound(VIL))] e C = [].
- Grau máximo de não determinismo: 1.

Saída:

- Tempo Médio de Execução:
  - NDet-HTN: *0,001 s*
  - Temporal NDet-HTN: *0,001 s*

- Reposta obtida: a especialização *roar('Draco')* e, com a utilização de backtracking, a especialização *do_evil_laugh('Lucius')*.

5.2.2 Suporte a Eventos Não Determinísticos

A fim de testar o suporte a eventos não determinísticos, o teste foi feito com o operador *fight_to_the_death(CH1,CH2)* (equivalente ao operador *fight_to_death(CH1,CH2)* em (SILVA, 2010)).

Parâmetros de entrada:

- Estado corrente:
  - Fatos+: [*current_place('Marian','Red_Castle'), current_place('Hoel','Red_Castle'),strength('Hoel',70.0), protection('Red_Castle',[-1.0,0.0]), affection(['Marian','Hoel'],100.0), kidnapped('Marian','Draco'*)];
  - Fatos⁻: [*current_place('Marian','White_Palace'), current_place('Hoel','Gray_Castle'),strength('Hoel',15.0), affection(['Marian','Hoel'],0.0),protection('Red_Castle',[-1.0,20.0])]*
- HTN:
  - U = [(t1, fight_to_the_death ('Hoel','Draco')),(t2, perform_epilogue)]
  - C = (t1,t2)
- Grau máximo de não determinismo: 1.

Saída:

- Tempo médio de execução:
  - NDet-HTN: *0,005 s*
  - Temporal NDet-HTN: *0,001 s*
- Resposta obtida: três especializações de *perform_epilogue* para serem incorporadas após *fight_to_the_death ('Hoel','Draco')*, para cada um dos três finais:
  - Após o final em que o herói mata o vilão, foi incorporada a especialização *marry('Hoel','Marian')*;
  - Após o final em que o vilão mata o herói, foi incorporada a especialização *commit_suicide('Marian')*;
  - Após o final em que ambos sucumbem à batalha, foi incorporada a especialização *join_convent('Marian')*.

5.2.3 Controle Do Nível de Não Determinismo

A fim de testar o controle do nível de não determinismo, utiliza-se um operador abstrato que tem especializações com número diferente de efeitos não determinísticos. Um exemplo de operador assim é o *combat* que possui uma especialização determinística *defeat* (composto pelos operadores básicos determinísticos *fight* seguido de *kill*) e uma não determinística (operador básico não determinístico *fight_to_the_death*). Deste modo, o planejamento pode vir a escolher uma ou outra especialização de acordo com o grau máximo de não determinismo permitido.

Foram realizados dois testes para planejamento do evento *victimise*. O primeiro com grau máximo de não determinismo igual a 2 e o segundo, 3.

O segundo teste envolve a comparação de eficiência no controle do grau de não determinismo, onde o algoritmo NDet-HTN não foi capaz de encontrar um plano possível encontrado pelo algoritmo Temporal NDet-HTN. O NDet-HTN encontrou um plano para um grau máximo de 2 e outro plano para um grau máximo de 6 (o plano da figura acima). O Temporal NDet-HTN encontrou os mesmos dois planos, mas, além deles, um outro plano com grau máximo de não determinismo 4.

Para encontrar o plano com grau máximo de não determinismo 2, O NDet-HTN obteve um tempo médio de *0,867 s*, e o Temporal NDet-HTN, um tempo médio de *0,012 s*. Assim, este último apresentou um tempo de execução cerca de *70* vezes maior. No caso do planejamento com grau máximo de não determinismo 4, O Temporal NDet-HTN obteve um tempo médio de *0.012 s*.

5.2.4 Tratamento de Tentativas

O teste para o conceito de tentativas foi feito com base no operador *perform_heroism*, que representa as diferentes reações que um herói pode ter a um ato de vilania. Este operador é um operador composto e sua sub-HTN é formada pela sequência de operadores: *<go,donate,go,attack,combat>*. O último operador da sequência, *combat*, tem duas especializações conforme supramencionado.

A rede de tarefas passada como parâmetro para o teste consiste da sequência *<combat,free>*. O segundo operador *free* representa o evento onde o herói liberta a princesa, e, para isso, o herói precisa estar vivo. Entretanto, o primeiro operador *combat* pode ser especializado por *fight_to_the_death*, um evento não determinístico onde o herói pode vir a morrer em um dos finais. Sendo assim, a escolha da especialização *fight_to_the_death* inviabiliza o encadeamento do segundo evento, *free*, ao não garantir

que o herói estará vivo antes deste. Uma maneira que poderia fazer com que *fight-_to_the_death* pudesse ocorrer é indicar o evento *free* como uma tentativa (que, assim, não precisa necessariamente ocorrer).

Para testar o conceito de tentativas foram feitos dois testes. O primeiro, com a HTN correspondendo à sequência <*combat,free*> e o segundo, à sequência <*combat,attempt(free,1)*>.

Para a HTN como a sequência <*combat,free*>, o Temporal NDet-HTN retornou o plano <*fight_kill,free*> O tempo médio para a obtenção deste plano sequência para o Temporal NDet-HTN foi de *0,003* s, e o NDet-HTN, de *0,017 s*. Para a sequência <*combat,attempt(free,1)*>, o Temporal NDet-HTN retornou tanto o plano <*fight_kill,free*> como também o plano da figura acima, incluindo o operador *fight_to_the_death*. O tempo médio para a obtenção de cada plano para o Temporal NDet-HTN foi de *0,001* s, e o NDet-HTN, de *0,009 s*.

5.2.5 Conclusão

Com base nos testes feitos, foi possível verificar que o tratamento do não determinismo não é afetado, em termos funcionais, pela incorporação do tratamento de fórmulas temporais quando não há fórmulas temporais impostas.

Além dos testes acima que foram replicados em cima dos testes em (SILVA, 2011), também se testou o planejamento de ambos os planejadores (Temporal NDet-HTN e NDet-HTN*) no que concerne à eficiência de planejamento HTN não determinístico (sem levar em conta o planejamento temporal), comparando-a com a eficiência do NDet-HTN original.

O teste do planejamento sem levar em conta o planejamento temporal é importante, pois demonstra que o algoritmo NDet-HTN, com a codificação melhorada, é capaz de realizar o planejamento de maneira muito mais eficiente. Este fato aumenta a confiança na escalabilidade de um planejamento HTN temporal que tenha como base um planejamento HTN eficientente.

É importante notar, que além da melhoria na codificação, foi feita uma melhoria no algoritmo na parte de controle do grau máximo de não determinismo. Neste aspecto, a melhoria de tempo foi considerável.

O primeiro teste realizado envolve o planejamento de uma história completa do LOGTELL que consiste de três capítulos. A HTN passada aos planejadores, representando a história completa, é a sequência de operadores complexos

<*perform_villainy,perform_heroism,perform_epilogue*>. O plano final gerado é uma árvore com 21 eventos e um grau de não determinismo igual a 6.

Nos testes realizados, o planejador NDet-HTN obteve um tempo médio de *0,867 s*, e o NDet-HTN*, um tempo médio de *0,012 s*. Assim, este último apresentou um tempo de execução cerca de *70* vezes menor. O Temporal Ndet-HTN, sem restrições temporais (ou seja, impondo a fórmula *true*), levou o tempo médio de *0,013s*, muito próximo do tempo do NDet-HTN*.

5.2.6 Utilizando Fórmulas Temporais

5.2.6.1 Visão Geral

Para os testes de planejamento temporal, foram utilizados os resultados de preprocessamento gerados pela implementação do modelo em (GOTTIN, 2013). Cada consulta por uma fórmula residual leva um tempo da ordem de *0.01 s.*

As fórmulas temporais utilizadas são *EF {alegria ≥ 70}*, *AG {suspense > 40}*, *AG {suspense < 80}* e *EF {alegria ≥ 60 and EG {alegria ≥ 35}}* e os valores de todas as quatro propriedades dramáticas escolhidas (*alegria*, *antecipação*, *medo* e *suspense*) começa em *50*. Estas fórmulas representam semanticamente, respectivamente, que: o capítulo deve ter um momento muito alegre; o capítulo deve manter o suspense acima de um nível aceitável; o capítulo não deve ultrapassar um nível de suspense considerado muito alto; e o capítulo deve ter um pico de alegria a partir do qual esta se manterá sempre acima de certo nível, considerado o mínimo aceitável. As fórmulas utilizadas são para fins de teste e não incorporam nenhum tipo de pesquisa autoral sobre quais seriam boas fórmulas temporais para arcos dramáticos típicos. Isto é mencionado na seção sobre trabalhos futuros.

O Temporal NDet-HTN tem duas frentes principais ao lidar com fórmulas temporais que garantem um aumento na eficiência. Primeiro, o algoritmo pode lidar com uma fórmula complexa do tipo *EF {alegria ≥ 70} and AG {suspense > 40}* sem precisar realizar backtracking, isto é, sem precisar realizar o mesmo planejamento mais de uma vez. Segundo, o algoritmo é capaz de eliminar da fórmula complexa que está sendo tratada, fórmulas existenciais que já tenham sido satisfeitas em um ramo. Assim, com uma fórmula complexa *EF {alegria ≥ 70} or AG {suspense > 40}*, caso um ramo do plano satisfaça *EF {alegria ≥ 70}*, a fórmula complexa é simplificada para *true* nos outros ramos. Os testes realizados se guiam por essas duas frentes.

Todos os testes realizados abaixo foram feitos recebendo como parâmetro uma HTN, correspondendo a uma história completa do contexto exemplo, representada pela sequência: <*perform_villainy,perform_heroism,perform_epilogue*>. Desta forma, pode-se testar o planejamento temporal com um razoável número de eventos.

5.2.6.2 Testes Básicos

O primeiro teste feito foi passando uma fórmula temporal *true* para medir um tempo base para o planejador. O tempo base obtido foi de *0,012 s*. Com este tempo base em mente, é possível raciocinar sobre os outros tempos obtidos para fórmulas básicas e complexas.

O segundo teste feito foi feito com uma fórmula básica, no caso, *EF {alegria ≥ 70}*. O tempo obtido foi de *0,216 s*, demonstrando que a adição de consultas por fórmulas residuais aumentou o tempo consideravelmente. Todavia, este tempo é muito razoável considerando que se planejou uma história completa.

O terceiro teste feito foi feito com a fórmula básica *AG {suspense > 40}*. Neste caso, o planejador retornou uma história completa diferente da retornada para o teste anterior, que não satisfaz essa fórmula básica. A história retornada é um a história completa com 4 finais. O tempo obtido foi de *0,158 s*.

O quarto teste realizado foi com a fórmula *AG {suspense < 80}* para o qual o planejador não encontrou um plano. O tempo de planejamento foi de *2,15 s*. Este teste é importante, pois mostra que é necessário decidir o que fazer quando o planejador não puder satisfazer a fórmula temporal. Estratégias para isso são mencionadas na seção sobre trabalhos futuros.

5.2.6.3 Fórmulas Complexas

O primeiro teste realizado foi com a fórmula complexa *EF {alegria ≥ 70} and AG {suspense > 40}* e o tempo obtido foi de *0,347 s*. O segundo teste realizado foi com a fórmula *AG {suspense > 40} or AG {suspense < 80}* e o tempo obtido foi de *0,398 s*. É importante notar que não há plano que satisfaça a fórmula *AG {suspense < 80}*. Portanto, foi a fórmula *AG {suspense > 40}* na disjunção *AG {suspense > 40} or AG {suspense < 80}* que levou o planejamento a encontrar o plano. O terceiro teste realizado foi com a fórmula *(EF {alegria ≥ 70} or AG {suspense < 80}) and AG {suspense > 40}*. O tempo obtido foi de *0,731 s*.

5.2.6.4 Eliminação de Fórmulas Existenciais

O primeiro teste realizado foi com a fórmula *EF ({alegria ≥ 60} and EG {alegria ≥ 35})*. Neste teste, o planejador encontrou em um ramo, um evento, *donate*, que tem o

pico de alegria (*{alegria ≥ 60}*), assim obtendo a fórmula residual *EG {alegria ≥ 35}*. Esta fórmula residual continuou por esse ramo do planejamento até ser dada como satisfeita no final de um evento folha no plano. Tendo a fórmula *EG {alegria ≥ 35}* satisfeita, quando o planejador voltou recursivamente para o evento *donate*, que havia a obtido como residual, ele pôde obter a fórmula *EF {alegria ≥ 60 and EG {alegria ≥ 35}}* como satisfeita. Desta forma, o outro ramo do planejamento não precisou lidar com esta fórmula. O tempo obtido foi de *0,067 s*.

O segundo teste realizado foi com a fórmula *EF {alegria ≥ 70}* or *AG {suspense > 40}* e o tempo obtido foi de *0,456 s*, devido à eliminação da fórmula *EF {alegria ≥ 70}*, quando esta está satisfeita por um dos ramos do plano.

5.2.6.5 Escalabilidade

O planejamento temporal adiciona uma complexidade considerável no planejamento HTN não determinístico. Desta forma, se torna interessante testar a escalabilidade do algoritmo Temporal NDet-HTN para fórmulas complexas grandes.

Primeiro é interessante testar a escalabilidade considerando a eliminação de fórmulas existenciais. Para simular isso se testou o algoritmo com uma disjunção composta pela fórmula *EF {alegria ≥ 70}* seguida de uma disjunção crescente da mesma fórmula *AG {suspense<80}*. Quando a fórmula está satisfeita, a disjunção inteira está satisfeita. Todavia, o planejador terá que consultar, em cada evento, para todas as fórmulas da disjunção, até que um primeiro evento a satisfaça. A tabela abaixo mostra o tempo (em segundos) para o número de fórmulas na disjunção.

Tabela 3. Teste de escalabilidade de disjunções.

1	2	4	10	20
0,478 s	*0,723 s*	*1,241 s*	*2,776 s*	*5,358 s*

O próximo teste foi feito para testar a consulta de um grande número de fórmulas. Para isso, testou-se com um número crescente da mesma fórmula *AG {suspense > 40}* em uma conjunção. A tabela abaixo mostra o tempo (em segundos) para o número de fórmulas na conjunção.

Tabela 4. Teste de escalabilidade de conjunções.

1	2	4	10	20
0,154 s	*0,287 s*	*0,546 s*	*1,356 s*	*2,680 s*

5.2.7 Comparação com "geração e teste"

A abordagem proposta nessa pesquisa visa contornar a necessidade de uma abordagem "geração e teste", onde um plano teria que ser gerado para, então, ser testado em relação às fórmulas temporais. Isso levaria à necessidade de completar um processo de planejamento inteiro antes de saber se o plano se conforma a certa fórmula temporal.

A comparação feita utiliza os resultados de ARAÚJO (2011) levando em consideração a geração de planos pelo algoritmo NDet-HTN*. Para o teste, buscou-se um teste de ÁRAUJO (2011) que equivalesse a um evento dessa pesquisa.

Para obter o primeiro tempo de comparação, referente à abordagem "geração e teste", foram gerados três planos diferentes por *backtracking*, e à soma dos três tempos médios da geração desses planos, foi somado três vezes o tempo médio de verificação do trabalho de ARAÚJO (2011). Para obter o segundo tempo de comparação, referente ao trabalho dessa pesquisa, foram somados os tempos médios de geração para os mesmos planos por *backtraking* (e fórmula equivalente) com o uso do Temporal NDet-HTN.

Os tempos médios para gerar os três planos diferentes por backtracking foram: *0,014*, *0,012* e *0,012 s*. O tempo médio obtido por ARAÚJO (2011) foi considerado para uma história com 24 eventos (próxima a história completa aqui utilizada, com 21 eventos) e a fórmula:

AF ((JOY>FEAR) and (ANTICIPATION>TRUST) and AG (JOY=<100.0 and JOY>= -100.0) and A (end (ANTICIPATION=TRUST and FEAR =\= JOY)))

O tempo médio relatado por ARAÚJO (2011) foi de *0,155* s. Deste modo, para obter um plano que satisfizesse essa fórmula acima, pode-se supor que o tempo total seria de *0,014 + 0,012 + 0,012 +* (3**0,492* s), equivalendo a 3 planejamentos pelo NDet-HTN* e 3 verificações complexas de ARAÚJO (2011). O tempo médio total, então, seria de *1,514 s*. Para o algoritmo Temporal NDet-HTN e a fórmula complexa *(EF {alegria ≥ 70} or AG {suspense < 80}) and AG {suspense > 40}*, o tempo médio obtido foi de *0,731* para encontrar um plano que satisfaça a fórmula, superando o tempo de "geração e teste".

5.2.8 Resumo dos Testes

Nesta seção apresentamos duas tabelas para ajudar a fornecer um resumo dos testes.

Tabela 5. Comparação de tempo médio entre os algoritmos para 3 capítulos e 21 eventos.

Planejador	Tempo (s)
NDet-HTN	0,867
NDet-HTN*	0,012
Temporal NDet-HTN	0,012

Tabela 6. Tempos médios do Temporal NDet-HTN para diferentes fórmulas.

Fórmula	Tempo (s)
true	0,012
EF {alegria ≥ 70}	0,216
AG {suspense > 40}	0,158
AG {suspense < 80}	2,15
EF {alegria ≥ 70} and AG {suspense > 40}	0,347
AG {suspense > 40} or AG {suspense < 80}	0,398
(EF {alegria ≥ 70} or AG {suspense < 80}) and AG {suspense > 40}	0,731
EF (alegria ≥ 60 and EG {alegria ≥ 35})	0,067
EF {alegria ≥ 70} or *AG {suspense > 40}*	0,456

6 Conclusão

Esta pesquisa apresenta um novo modelo de geração de enredos para a área de *Storytelling* Interativo. O modelo visa à geração eficiente de enredos com eventos não determinísticos e conformidade a arcos dramáticos típicos de um gênero, assumindo a variação contínua de propriedades dramáticas. Desta forma, aumenta-se a variedade de enredos e a qualidade dramática, com a esperança de contribuir para enriquecer os enredos de forma a cativar e surpreender o usuário. O modelo dá continuidade ao modelo proposto por SILVA (2010) e se inspira na proposta de ARAÚJO (2011), e assim mantém-se compatível com o sistema de *Storytelling* Interativo LOGTELL para TV Interativa.

O ganho de eficiência através da utilização de planejamento HTN, e de uma etapa de preprocessamento, permitiu que a complexidade adicionada pela verificação temporal de variáveis contínuas não inviabilizasse a geração de enredos. O planejamento HTN parte de um esboço de capítulo gerado por um planejador de ordem parcial para, então, detalhá-lo e fornecer um capítulo completo em forma de uma árvore de contingências. Este planejador de ordem parcial é essencial, pois permite a incorporação das sugestões do usuário e a geração de enredos que não são simplesmente uma composição de alternativas previamente descritas. Assim sendo, o papel do planejador HTN é gerar enredos detalhados uma vez que um esboço tenha sido feito.

A validação do modelo foi feita através da codificação de um protótipo, a ser integrado ao LOGTELL, utilizando uma extensão do gênero apresentado em (SILVA, 2010), que, por sua vez, é uma extensão do gênero “Espadas & Dragões”, utilizado para validar a primeira versão do LOGTELL. Há uma apresentação resumida dessa pesquisa em (FERREIRA *et al*., 2013).

6.1 Principais Contribuições

6.1.1 Modelo de Geração de Enredos Não Determinísticos com Verificação de Fórmulas Temporais sobre Propriedades Dramáticas Contínuas

Neste trabalho foi feito todo um estudo de como o planejamento não determinístico poderia ser compatibilizado com a verificação de propriedades dramáticas contínuas, durante a geração e dramatização do enredo. Foi definido esquema de especificação de autômatos para os eventos e de variação das propriedades contínuas. Além disso, foi feito estudo de como um preprocessamento da variação abstrata dessas propriedades poderia ser utilizado para tornar viável essa combinação.

O modelo apresentado mantém o atendimento a todos os requisitos atendidos pelo modelo de (SILVA, 2010) com a adição da verificação, em tempo de planejamento, de restrições temporais sobre a mudança contínua de propriedades dramáticas. Apesar desta adição, que poderia levar a uma possível explosão combinatória, o modelo manteve-se viável em termos de tempo de execução ao utilizar uma etapa de preprocessamento e um algoritmo eficiente de planejamento HTN não determinístico. Com isso, consegue-se gerar enredos em tempo hábil, fornecendo-se meios para garantir a variação das propriedades dramáticas de acordo com as especificidades do gênero ou o desejo do autor.

6.1.2 Melhoria de Desempenho no Planejamento Não Determinístico (independente do uso de restrições temporais)

Como subproduto desta dissertação, foi feito melhoramento considerável no desempenho do algoritmo NDet-HTN antes da incorporação do tratamento de restrições temporais sobre propriedades dramáticas. Em particular, a nova implementação de NDet-HTN, o NDet-HTN*, melhora o controle do nível de não determinismo com a identificação imediata de quando um plano excederá um limite fornecido. O algoritmo modificado superou o algoritmo de (SILVA, 2010) em todos os testes em larga margem quando fórmulas temporais não são usadas. Nesse caso, histórias com muito mais eventos e maior grau de não determinismo poderão ser criadas.

6.1.3 Algoritmo Temporal NDet-HTN

O algoritmo, conforme codificado no protótipo, mostrou-se eficiente e gerou enredos com e sem fórmulas temporais em tempos de execução viáveis para a área de *Storyteling* Interativo. No entanto, o Temporal NDet-HTN é genérico para utilização em outros domínios desde que seja possível um esquema de consulta de fórmulas temporais

residuais para os operadores. O algoritmo NDet-HTN de (SILVA, 2010) foi utilizado em (VALFRE, 2010) para composição de serviços WEB. O Temporal NDet-HTN seria, por exemplo, aplicável no mesmo contexto para modelar composição de serviços WEB com restrições temporais sobre propriedades contínuas. Em aplicações de Engenharia, onde restrições temporais sobre a combinação de diversos procedimentos são comuns, há também aplicações claras do algoritmo.

6.1.4 O Protótipo

O protótipo implementado foi inspirado pelo Logtell e é compatível com o sistema. Sua incorporação ao Logtell permitirá a geração de enredos complexos e com padrões dramáticos que aumentem a qualidade da história.

Além disso, o protótipo implementado serve de base para a realização de estudos adicionais de como a geração de enredos pode ser melhorada ao se combinar o não determinismo com a verificação de restrições sobre propriedades dramáticas contínuas.

6.2 Trabalhos Futuros

6.2.1 Integração com o LOGTELL

Para que o modelo aqui apresentado seja incorporado ao LOGTELL são necessárias algumas modificações nos módulos:

- **Drama Manager**: deve passar a lidar com os autômatos no novo formato.
- **Interface (User Interface e Interface Controller)**: deve ser capaz de lidar com interferências que aproveitem a nova modelagem dos autômatos. A intervenção pode se dar de diversas maneiras dentro de um espectro que vai desde preferências em relação ao enredo (faixa etária, preferência por uma história mais alegre ou mais tensa, torcer por um personagem etc.) até intervenções mais diretas (objetivos a alcançar).
    - **Modo Passo a Passo**: maneiras de se mostrar os diferentes caminhos possíveis dentro de um mesmo capítulo (e enredo).

6.2.2 Gramática Completa da Lógica Temporal Proposta

Conforme supramencionado, o algoritmo não lida com a gramática completa da lógica apresentada em (ARAÚJO, 2011). As fórmulas temporais com as quais o algoritmo não lida são:

- Negações de fórmulas $A\ \alpha\ U\ \beta$ e $E\ \alpha\ U\ \beta$;

- Fórmulas do tipo $AG\ \alpha$ e $EG\ \alpha$, quando α é uma disjunção de fórmulas (básicas ou quantificadas) ou uma fórmula do tipo $\beta \neq \gamma$.

É necessário modificar o algoritmo para torná-lo capaz de lidar com a gramática completa. Isto é interessante, pois o torna mais abrangente e capaz de lidar com um conjunto mais expressivo de fórmulas temporais. Em (ARAÚJO, 2011), a negação por falha é usada nos dois casos, mas ela embute um mecanismo ineficiente de geração e teste. A incorporação de mecanismos similares é possibilidade a ser considerada, mas ela não pode afetar o desempenho.

6.2.3 Aplicação em Outros Domínios

O algoritmo de planejamento resultante dessa pesquisa se inspira na área de *Storytelling* Interativo para lidar de maneira eficiente com variação temporal contínua de propriedades dramáticas. Todavia, o algoritmo é genérico para domínios modeláveis em termos de problemas de planejamento HTN com objetivos estendidos sobre variação contínua de variáveis ao longo do tempo. Seria interessante pesquisar as aplicabilidades do algoritmo em diversos domínios onde se deseje eficiência e conformidade a fórmulas temporais que sejam conhecidas anteriormente ao planejamento.

6.2.4 Inferir Arcos Dramáticos Típicos para cada Capítulo

As fórmulas temporais, representando os arcos dramáticos típicos, foram assumidas como dadas para o teste da geração de um capítulo. Entretanto, uma frente possível de pesquisa é a inferência dos arcos dramáticos para cada capítulo. Pode-se pensar em um arco dramático geral que o autor queira atingir para a história, e, a partir dele, inferir os subarcos dramáticos de cada capítulo.

6.2.5 Histórias *à la carte*

Poderiam ser oferecida algumas histórias, dentre as quais os usuários pudessem escolher quais assistir de acordo com o seu desejo por um ou outro padrão de arco dramático. Desta forma, se um usuário estiver com vontade de assistir a uma história mais alegre, ele poderia restringir as histórias a serem geradas através da seleção de certa *classe dramática*. Esta, por sua vez, seria uma classe que agrupa diferentes arcos dramáticos típicos. Por exemplo, o usuário poderia escolher a classe dramática *alegre* dentro do gênero "Espadas & Dragões". Assim, as histórias geradas estariam coerentes em relação ao gênero e em relação à classe escolhida pelo usuário. Além disso, o usuário poderia escolher a classe, mas não o gênero. Neste caso, o sistema poderia

sugerir diferentes gêneros que possuam arcos dramáticos daquela classe para que o usuário escolha. Afinal, diferentes gêneros poderiam compartilhar o mesmo arco dramático típico.

7 Referências

AI-CHANG, M., BRESINA, J.L., CHAREST, L., *et al*., 2004, "MAPGEN: Mixed-initiative planning and scheduling for the mars exploration rover mission", *IEEE Intelligent Systems*, v. 19, n. 1 (Jan), pp. 8-12.

ALLEN, J.F., 1991, "Planning as Temporal Reasoning". In: *proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning*, pp. 3-14, Cambridge, Apr.

ARAÚJO, E.T., CIARLINI, A.E.M., 2011, "Verification of Temporal Constraints in Continuous Time on Nondeterministic Stories". In: *Proceedings of the 10th International Conference on Entertainment Computing*, pp. 28-34, Vancouver, Oct.

ARAÚJO, E.T., 2011, *Verificação de Restrições Temporais com Tempo Contínuo em Histórias Não-Determinísticas*. Dissertação de M.Sc., Departamento de Informática, Universidade Federal do Estado do Rio de Janeiro, RJ, Brasil.

ARISTÓTELES., BUTCHER, S.H., 2011, *The Poetics*. 1 ed. Seattle, CreateSpace Independent Publishing Platform.

BARBER, H., KUDENKO, D., 2007, "Dynamic Generation of Dilemma-based Interactive Narratives". In: *Proceedings of the Third Artificial Intelligence and Interactive Digital Entertainment Conference*, pp. 2-7, Stanford, Jun.

BARROS, L.M., MUSSE, S.R., 2008, "Towards consistency in interactive storytelling: Tension arcs and dead-ends", *Computers in Entertainment*, v. 6, n. 3 (Oct), article n. 43.

BARROS, L.M., MUSSE, S.R., 2007, "Planning algorithms for interactive storytelling", *Computers in Entertainment*, v. 5, n. 1 (Jan), article n. 2.

BOSSER, A., CAVAZZA, M., CHAMPAGNAT, R., 2010, "Linear Logic for Non-Linear Storytelling". In: *Proceedings of the 19th European Conference on Artificial Intelligence*, pp. 713-718, Lisboa, Aug.

BRANIGAN, E., 1992, *Narrative Comprehension and Film*. 1 ed. New York, Routledge.

BRYANT, R.E., 1992, "Symbolic Boolean Manipulation with Ordered Binary Decision Diagrams", *ACM Computing Surveys*, v. 24, n. 3 (Sep), pp. 293-318.

CAI, Y., SHEN, Z., MIAO, C., *et al*., 2007, "S-MADE: Interactive Storytelling Architecture through Goal Execution and Decomposition". In: *Proceedings of the AAAI Fall Symposium*, pp. 17-20, Menlo Park, Sep.

CATALDI, M., DAMIANO, R., LOMBARDO, V., *et al*., 2011, "Representing Dramatic Features of Stories through an Ontological Model". In: *Proceedings of the Fourth International Conference on Interactive Digital Storytelling*, pp. 122-127, Vancouver, Nov.

CHEONG, Y.G., YOUNG, R.M., 2008, "Narrative Generation for Suspense: Modeling and Evaluation". In: *Proceedings of the 1st Joint International Conference on Interactive Digital Storytelling*, pp. 144-155, Erfurt, Nov.

CHEONG, Y.G., 2007, *A Computational Model Of Narrative Generation For Suspense*. Ph.D. thesis, Department of Computer Science, North Carolina State University, Raleigh, North Carolina, USA.

CIARLINI, A.E.M, CASANOVA, M.A., FURTADO, A.L., *et al*., 2010, "Modeling interactive storytelling genres as application domains", *Journal of Intelligent Information Systems*, v. 35, n. 3 (Dec), pp. 347-381.

CIARLINI, A.E.M., CAMANHO, M.M., DÓRIA, T.R., *et al*., 2008, "Planning and Interaction Levels for TV Storytelling". In: *Proceedings of the 1st Joint International Conference on Interactive Digital Storytelling*, pp. 198-209, Erfurt, Nov.

CIARLINI, A.E.M., POZZER, C.T., FURTADO, A.L., *et al*., 2005, "A logic-based tool for interactive generation and dramatization of stories". In: *Proceedings of the 2nd International Conference on Advances in Computer Entertainment Technology*, pp. 133-140, Valencia, Jun.

CIARLINI, A.E.M., 1999, *Geração Interativa De Enredos*. Tese de D.Sc., Departamento de Informática, Pontifícia Universidade Católica, RJ, Brasil.

CIMATTI, A., PISTORE, M., ROVERI, M., *et al*., 2003, "Weak, Strong, and Strong Cyclic Planning via Symbolic Model Checking", *Artificial Intelligence*, v. 147, n 1-2 (Jul), pp. 35-84.

COLES, A., FOX, M., HALSEY, K., *et al*., 2009. "Managing concurrency in temporal planning using planner-scheduler interaction", *Artificial Intelligence*, v. 73, n 1 (Jan), pp. 1-44.

CUSHING, W., 2012, *When is Temporal Planning Really Temporal?*. Ph.D. thesis, School of Computing, Arizona State University, Phoenix, USA.

DIAS, R., MARTINHO, C., 2011, "Adapting content presentation and control to player personality in videogames". In: *Proceedings of the 8th International Conference on Advances in Computer Entertainment Technology*, pp. 216-227, Lisbon, Nov.

DO, M.B., KAMBHAMPATI, S., 2003, "Sapa: A Domain-Independent Heuristic Metric Temporal Planner", *Journal of Artificial Intelligence Research*, v. 20, n. 1 (Dec), pp. 155-194.

DÓRIA, T.R., CIARLINI, A.E.M., ANDREATTA, A., 2008, "A nondeterministic model for controlling the dramatization of interactive stories". In: *Proceedings of the 2nd ACM Workshop on Story Representation, Mechanism and Context*, pp. 21-26, Vancouver, Oct.

DOW, S., MACINTYRE, B., MATEAS, M., 2008, "Styles of play in immersive and interactive story: case studies from a gallery installation of AR Façade". In: *Proceedings of the 5th International Conference on Advances in Computer Entertainment Technology*, pp. 373-380, Yokohama, Dec.

FERREIRA, P.A., GOTTIN, V.M., CIARLINI, A.E.M., *et al*., 2013, "A Nondeterministic Temporal Planning Model for Generating Narratives with Continuous Change in Interactive Storytelling". In: *Proceedings of the 9th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*. No prelo.

FREYTAG, G., MACEWAN, E.J., 1900, Technique of the Drama: An Exposition of Dramatic Composition and Art. 3 ed. Chicago, Scott, Foresman and Company.

GERVÁS, P., 2009, "Computational Approaches to Storytelling and Creativity", *AI Magazine*, v. 30, n. 3 (Sep), pp. 49-62.

GHALLAB, M., NAU, D., TRAVERSO, P., 2004. Automated Planning: Theory and Practice. 1 ed. Amsterdam, Morgan Kaufmann Publishers.

GILROY, S., PORTEOUS, J., CHARLES, F., *et al*., 2012, "Exploring Passive User Interaction for Adaptive Narratives". In: *Proceedings of the Proceedings of the 17th International Conference on Intelligent User Interfaces*, pp. 119-128, New York, Feb.

GLASSNER, A.S., 2004, *Interactive Storytelling: techniques for 21st century fiction*. 1 ed. Natick, AK Peters.

GOTTIN, V.M., 2013, *Verificação Abstrata de Propriedades Dramáticas Contínuas em Eventos Não Determinísticos*. Dissertação de M.Sc., Departamento de Informática, Universidade Federal do Estado do Rio de Janeiro, RJ, Brasil.

GRASBON, D., BRAUN, N., 2001, "A Morphological Approach to Interactive Storytelling". In: Proceedings of the Conference on artistic, cultural and scientific aspects of experimental media spaces, pp. 337-340, Sankt Augustin, Sep.

GROOTE, J.F., TVERETINA, O., 2003, "Binary decision diagrams for first-order predicate logic", *Journal of Logic and Algebraic Programming*, v. 57, n. 1-2 (Sep), pp. 1-22.

HARTSOOK, K., ZOOK, A., DAS, S., *et al*., 2011, "Toward Supporting Stories with Procedurally Generated Game Worlds". In: *Proceedings of the IEEE Conference on Computational Intelligence and Games*, pp. 297-304, New York, Sep.

HOPCROFT, J.E. ULLMAN, J.D., MONTWANI, R., 2006, Introduction to Automata Theory, Languages, and Computation. 3 ed. New Jersey, Prentice Hall.

HUTH, M., RYAN, M., 2004. *Logic in Computer Science: Modelling and Reasoning about Systems*. 2 ed. Cambridge, Cambridge University Press.

KLINE, D., 2009, "Bringing Interactive Storytelling to Industry: Designing a Reactive Narrative Encounter System". In: *Proceedings of the 5th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, pp. 216-227, Palo Alto, Oct.

KUTER, U., NAU, D., PISTORE, M., *et* al., 2009, "Task Decomposition on Abstract States for Planning under Nondeterminism", *Artificial Intelligence*, v. 173, n. 5-6 (Apr), pp. 669-695.

KUTER, U., NAU, D., 2004, "Forward-chaining planning in nondeterministic domains". In: *Proceedings of the 19th national conference on Artifical intelligence*, pp. 513-518, San Jose, Jul.

LI, H.X., WILLIAMS, B.C., 2008, "Generative Planning for Hybrid Systems Based on Flow Tubes". In: *Proceedings of the 18th International Conference on Automated Planning and Scheduling*, pp. 206-213, Sidney, Sep.

LIMA, E.S., FEIJO, B., POZZER, C.T., *et al*., 2012, "Social Interaction for Interactive Storytelling". In: *Proceedings of the 11th International Conference on Entertainment Computing*, pp. 1-15, Bremen, Sep.

MAGERKO, B., 2007, "Measuring Dramatic Believability". In: *Proceedings of the AAAI Fall Symposium*, pp. 79-82, Menlo Park, Nov.

MARRIOT, K., STUCKEY, P.J., 1998, *Programming with Constraints.* 1 ed. Cambridge, The MIT Press.

MATEAS, M., STERN, A., 2005, "The Interactive Drama Façade". In: *Proceedings od the 1st AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, pp. 153-154, Marina del Rey, Jun.

MCMILLAN, K.L., 1993, *Symbolic Model Checking*. 1 ed. Dordrecht, Kluwer Academic.

MEHM, F., GÖBEL, S., STEINMEZ, R., 2012, "Authoring of Serious Adventure Games in StoryTec". In: *Proceedings of the 7th International Conference on E-Learning and Games for Training, Education, Health and Sports"*, pp. 144-154, Darmstadt, Sep.

MEISER, J.C., 2003, *Computing Action: A Narratological Approach*. 1 ed. Walter De Gruyter Inc.

MURRAY, J.H., SI, M., THUE, D., *et al*., 2011, "Why Paris Needs Hector and Lancelot Needs Mordred: Using Traditional Narrative Roles and Functions for Dramatic Compression in Interactive Narrative". In: *Proceedings of the Fourth International Conference on Interactive Digital Storytelling*, pp. 13-24, Vancouver, Nov.

NAU, D., AU, T-C., ILGHAMI, O. *et al*., 2003, "SHOP2: An HTN Planning System", *Journal of Artificial Intelligence Research*, v. 20, n. 1 (Dec), pp. 379-404.

O'NEIL, B., RIEDL, M., 2011, "Toward a computational framework of suspense and dramatic arc". In: *Proceedings of the International Conference on Affective Computing and Intelligent Interaction*, pp. 246-255, Memphis, Oct.

PEINADO, F., CAVAZZA, M., PIZZI, D., 2008, "Revisiting Character-Based Affective Storytelling under a Narrative BDI Framework". In: *Proceedings of the 1st Joint International Conference on Interactive Digital Storytelling*, pp. 83-88, Erfurt, Nov.

PENBERTHY, J.S., WELD, D.S, 1994, "Temporal planning with continuous change". In: *Proceedings of the 12th national conference on Artificial intelligence*, pp. 1010-1015, Seattle, Jul.

PETTA, P. *IRIS publication database on Interactive Storytelling*. Disponível em: <http://iris.ofai.at:7777/iris_db/index.php/>. Acesso em: 21 setembro 2013.

PIZZI, D., 2011, *Emotional Planning for Character-based Interactive Storytelling*. Ph.D. dissertation, School of Computing, Teesside University, Middlesborough, UK.

PORTEOUS, J., TEUTENBERG, J., CHARLES, F., *et al*., 2011, "Controlling Narrative Time in Interactive Storytelling", In: *Proceedings of the 10th International Conference on Autonomous Agents and Multiagent Systems*, pp. 449-456, Taipei, May.

PORTEOUS, J., CAVAZZA, M., CHARLES, F., 2010, "Applying planning to interactive storytelling: Narrative control using state constraints", *ACM Transactions on Intelligent Systems and Technology*, v. 1, n. 2 (Dec), pp. 10:1-10:21.

PORTEOUS, J., CAVAZZA, M., 2009, "Controlling Narrative Generation with Planning Trajectories: The Role of Constraints". In: *Proceedings of the 2nd Joint International Conference on Interactive Digital Storytelling*, pp. 234-245, Guimarães, Dec.

PROPP, V., 2003, *Morphology of the Folktale*. 2 ed. Austin, University of Texas Press.

ROBERTS, D.L., ISBELL, C.L., 2008, "A Survey and Qualitative Analysis of Recent Advances in Drama Management", *International Transactions on Systems Science and Applications*, v. 4, n. 1 (May), pp. 61-75.

RIEDL, M., YOUNG, R.M., 2004, "An Intent-Driven Planner for Multi-Agent Story Generation". In: *Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multiagent Systems*, pp. 186-193, New York, Jul.

RIDERS. Research in Interactive Drama Environments, Role-play and Storytelling. Disponível em: < http://www.riders-project.net/>. Acesso em: 21 setembro 2013.

SICSTUS. SICStus Prolog. Disponível em: < http://sicstus.sics.se/ >. Acesso em: 26 setembro 2013.

SILVA, M.A.R., LIMA, D.C., SILVA, J.C.A., *et al*., 2011, "A Narrative Game as an Educational Entertainment Resource to Teach Words to Children with Learning Deficits: A Feasibility Study". In: *Proceedings of the 10th International Conference on Entertainment Computing*, pp. 222-227, New York, Oct.

SILVA, F.A.G., CIARLINI, A. E. M., SIQUEIRA, S.W.M., 2010, "Nondeterministic Planning for Generating Interactive Plots". In: *Proceedings of the 12th Ibero-American Conference on Artificial Intelligence*, pp. 133-143, Bahía Blanca, Nov.

SILVA, F.A.G., 2010, *Geração de Enredos com Planejamento Não-Determinístico em Storytelling para TV* Interativa. Dissertação de M.Sc., Departamento de Informática, Universidade Federal do Estado do Rio de Janeiro, RJ, Brasil.

TIERNEY, K., COLES, A.J., COLES, A., *et al*., 2012 "Automated Planning for Liner Shipping Fleet Repositioning". In: *Proceedings of the 22th International Conference on Automated Planning and Scheduling*, pp. 156-170, Atibaia, Jun.

VENEMA, Y., 2001, "Temporal Logic". In: GLOBE, L. (eds), The Blackwell Guide to Philosophical Logic, 2 ed., chapter 10, Malden, USA, Blackwell Publishers.

VERNIERI, T., 2006, *A Web Services Approach to Generating and Using Plans in Configurable Execution Environments*. Ph.D. thesis, Department of Computer Science, North Carolina State University, Raleigh, North Carolina, USA.

WARE, S.G., YOUNG, R.M., 2011, "CPOCL: A Narrative Planner Supporting Conflict". In: *Proceedings of the 7th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, pp. 97-102, Stanford, Oct.

WARE, S.G., YOUNG, R.M., 2010, "Modeling Narrative Conflict to Generate Interesting Stories". In: *Proceedings of the 6th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, pp. 277-298, Stanford, Oct.

WEGENER, I., 2004, "BDDs — design, analysis, complexity, and applications", *Discrete Applied Mathematics*, v. 138, n. 1-2 (Mar), pp. 229-251.

YOUNES, H.L.S., SIMMONS, R.G., 2003, "VHPOP: Versatile Heuristic Partial Order Planner", *Journal of Artificial Intelligence Research*, v. 20, n. 1 (Dec), pp. 405-430.

ZAGALO, N., BARKER, A., BRANCO, V., 2004, "Story reaction structures to emotion detection". In: *Proceedings of the 1st ACM workshop on Story representation, mechanism and context*, pp. 33-38, New York, Oct.

ANEXO I – LOGTELL

I.2 Arquitetura

O LOGTELL teve como base inicial o IPG (*Interactive Plot Generator*) (CIARLINI, 1999) que é o módulo usado para gerar capítulos da história. Utilizando o IPG foi criada a primeira versão do sistema de modo a permitir dramatização da história em um ambiente de animações 3D (POZZER, 2005). Em seguida, nova versão foi criada para possibilitar uma arquitetura cliente-servidor, pensada para TV Interativa (CIARLINI *et al.*, 2008), onde os processos de geração e dramatização dos capítulos ocorre de forma paralela e contínua. Além disso, de modo a possibilitar maior variedade na dramatização e maior controle sobre o tempo de dramatização de cada evento, foi proposta extensão do sistema. Foi proposta a modelagem de cada evento como um autômato não determinístico que coordena a execução de microações por meio de políticas que visam acelerar ou retardar a sua dramatização, conforme a conveniência (DÓRIA, 2009). Inicialmente, o modelo previa um único estado final para cada evento de modo a se garantir a coerência com enredo gerado. No entanto, como será descrito mais à frente, o uso de um planejador não determinístico na geração do enredo elimina essa restrição. Na figura abaixo, temos uma visão geral da arquitetura do LOGTELL.

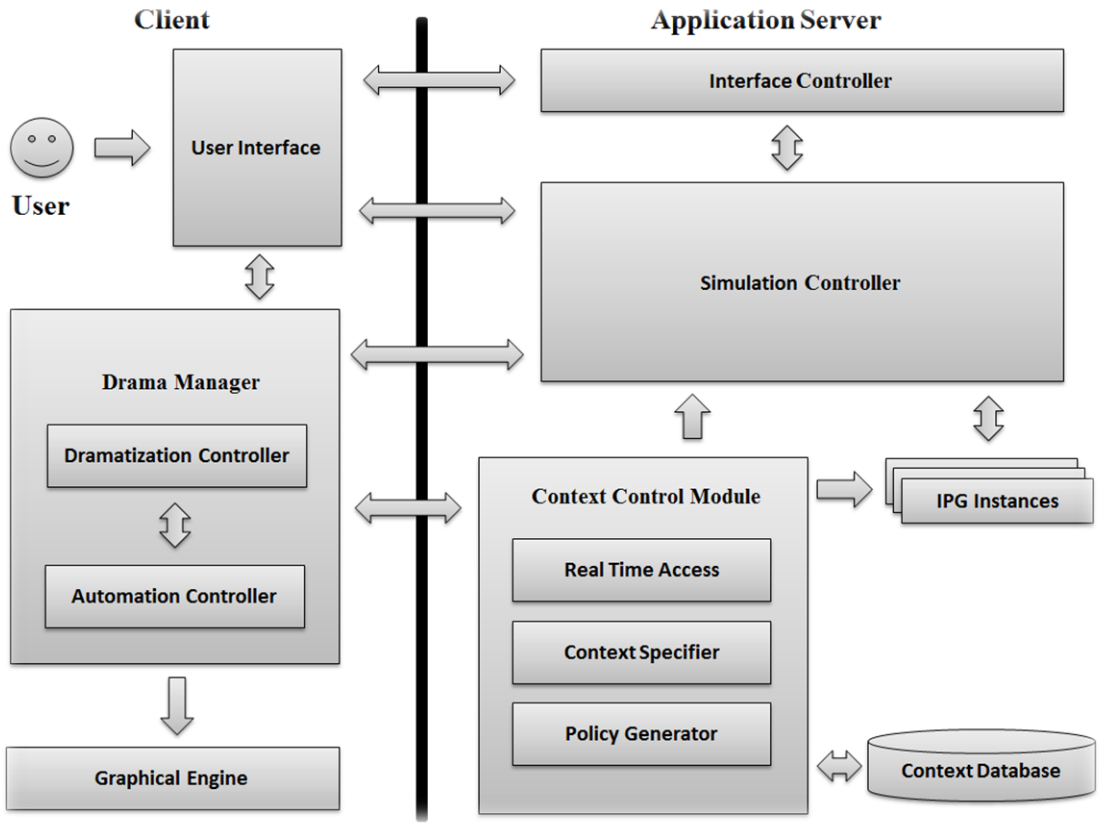

Figura 22. Arquitetura geral do LOGTELL (DÓRIA, 2009).

Os módulos e suas reponsabilidades são:

- *User Interface*:
  - Informar as interações com o usuário ao *Interface Controller*;
- *Drama Manager*:
  - *Dramatizaion Controller*:
    - Pedir políticas ao *Policy Generator* e gerenciar a execução de uma ou outra de acordo com as necessidades da dramatização, permitindo a sincronização da dramatização com a geração e enredos;
    - Enviar, momento a momento, uma microação ao *Animation* Controller a fim de dramatiza-la.
  - *Animation Controller*: Enviar microações à *Graphical Engine* para dramatização.
- *Graphical Engine*: Motor gráfico 3D capaz de dramatizar microações parametrizadas;
- *Interface Controller*:

    - Centralizar as interações vindas de todos os clientes. Caso vários usuários compartilhem a mesma história, as interações são escolhidas segundo o número de clientes que as fizeram. No caso de haver comente um cliente, a interação segue direto para o *Simulation Controller*.
- *Simulation Controller*:
    - Informar ao Drama Manager os próximos eventos a serem dramatizados;
    - Garantir a incorporar das requisições do usuário à história;
    - Controlar as diversas instâncias do IPG para a geração dos enredos;
    - Selecionar histórias viáveis e interessantes para enriquecer a experiência e a interação com o usuário;
    - Controlar o tempo de simulação.
- *Context Control Module:*
    - *Real-Time Access:* Prover acesso em tempo real aos módulos *Context Specifier* e *Policy Generator;*
    - *Context Specifier*: Viabilizar a definição de gêneros e contextos e armazená-los no *Context Database*;
    - *Policy Generator*: Criar políticas de dramatização levando em consideração os autômatos de cada evento;
- *IPG Instance*: Gerar enredos.

I.2.1 Especificação do Gênero das Histórias

As relações utilizadas para representar entidades, heranças entre entidades, atributos e papéis, no gênero exemplo de “Espadas & Dragões”, estão listados nas figuras abaixo. Após as figuras, há um diagrama de entidade-relacionamento completo para uma visão geral.

Entidade	Descrição
creature(CH)	*CH* é uma criatura
dragon(CH)	*CH* é um dragão
knight(CH)	*CH* é um cavaleiro
magician(CH)	*CH* é um mago
person(CH)	*CH* é uma pessoa
place(PL)	*PL* é um local
princess(CH)	*CH* é uma princesa

Figura 23. Entidades do gênero "Espadas & Dragões" do LOGTELL.

Herança entre Entidades	Descrição
is_a(dragon,creature)	Um dragão é uma criatura
is_a(person,creature)	Uma pessoa é uma criatura
is_a(knight,person)	Um cavaleiro é uma pessoa
is_a(magician,person)	Um mago é uma pessoa
is_a(princess,person)	Uma princesa é uma pessoa

Figura 24. Heranças entre entidades do gênero "Espadas & Dragões".

Relacionamentos	Entidades	Descrição
acquaintance(CRE1,CRE2)	*creature* com *creature*	A criatura *CRE1* conhece a criatura *CRE2*
current_place(CRE,PL)	*creature* com *place*	A criatura *CRE* se encontra no local *PL*
home(CRE,PL)	*creature* com *place*	O lar da criatura *CRE* é *PL*
kidnapped(PER,CRE)	*person* com *creature*	A pessoa *PER* está sequestrada pela criatura *CRE*
married(PER1, PER2)	*person* com *person*	A pessoa *PER1* está casada com a pessoa PER2
murderer(CRE1, CRE2)	*creature* com *creature*	A criatura *CRE1* é a assassina da criatura *CRE2*

Figura 25. Relacionamentos entre entidades do gênero "Espadas & Dragões".

Atributo	Tipo	Entidade/Relação	Descrição
affection(CH1,CH2,N)	*real*	*acquaintance*	O personagem *CH1* conhece o personagem *CH2* com nível de afeição *N*
alive(CRE)	*boolean*	*creature*	A criatura *CRE* está viva
gender(PER,G)	*integer*	*person*	O gênero da pessoa *PER* é *G*
alegria(PER,N)	*real*	*person*	A pessoa *PER* está com um nível de alegria *N*
nature(CRE,KIND)	*integer*	*creature*	A criatura *CRE* tem natureza *KIND* (má, neutra ou boa)
protection(PL,(K,N))	*(integer,real)*	*place*	O local *PL* tem uma proteção de natureza *K* e nível *N*
strength(CRE,N)	*real*	*creature*	A criatura *CRE* possui nível de força *N*

Figura 26. Atributos e tipos de entidades/relações do gênero "Espadas & Dragões".

Papel	Entidades	Descrição
hero(KNI)	*knight*	O cavaleiro *KNI* é um herói
victim(PRI)	*princess*	A princesa *PRI* é uma vítima
protagonist(CH)	*princess*, *knight*	A princesa ou o cavaleiro *CH* é um protagonista
villain(DRA)	*dragon*	O dragão *DRA* é um vilão
donor(MAG)	*magician*	O mago *MAG* é um doador (de força)

Figura 27. Papéis para entidades do gênero "Espadas & Dragões".

Evento	Descrição
attack	Um personagem ataca as defesas de um local
bewitch	Um personagem lança um feitiço sobre outro
donate	Um personagem doa força a outro
fight	Um personagem luta com outro
free	Um personagem liberta outro
go	Um personagem vai a um local
kidnap	Um personagem sequestra outro
kill	Um personagem mata outro
marry	Um personagem casa com outro
reduce_protection	Um personagem reduz a proteção de um local

Figura 28. Eventos do gênero "Espadas & Dragões" do LOGTELL.

Regra de inferência de Objetivo
Uma vítima em um local de mesma natureza que a sua irá reduzir a proteção deste local
O herói mais forte vai querer se tornar mais forte que o vilão
Se uma vítima estiver em um local desprotegido, o vilão irá sequestrá-la
Se uma vítima for sequestrada, um herói tentará libertá-la
Se uma vítima for morta, um herói tentará vinga-la
Se dois personagens têm um nível de afeição alto, eles irão se casar

Figura 29. Regras de inferência de objetivo do gênero "Espadas & Dragões".

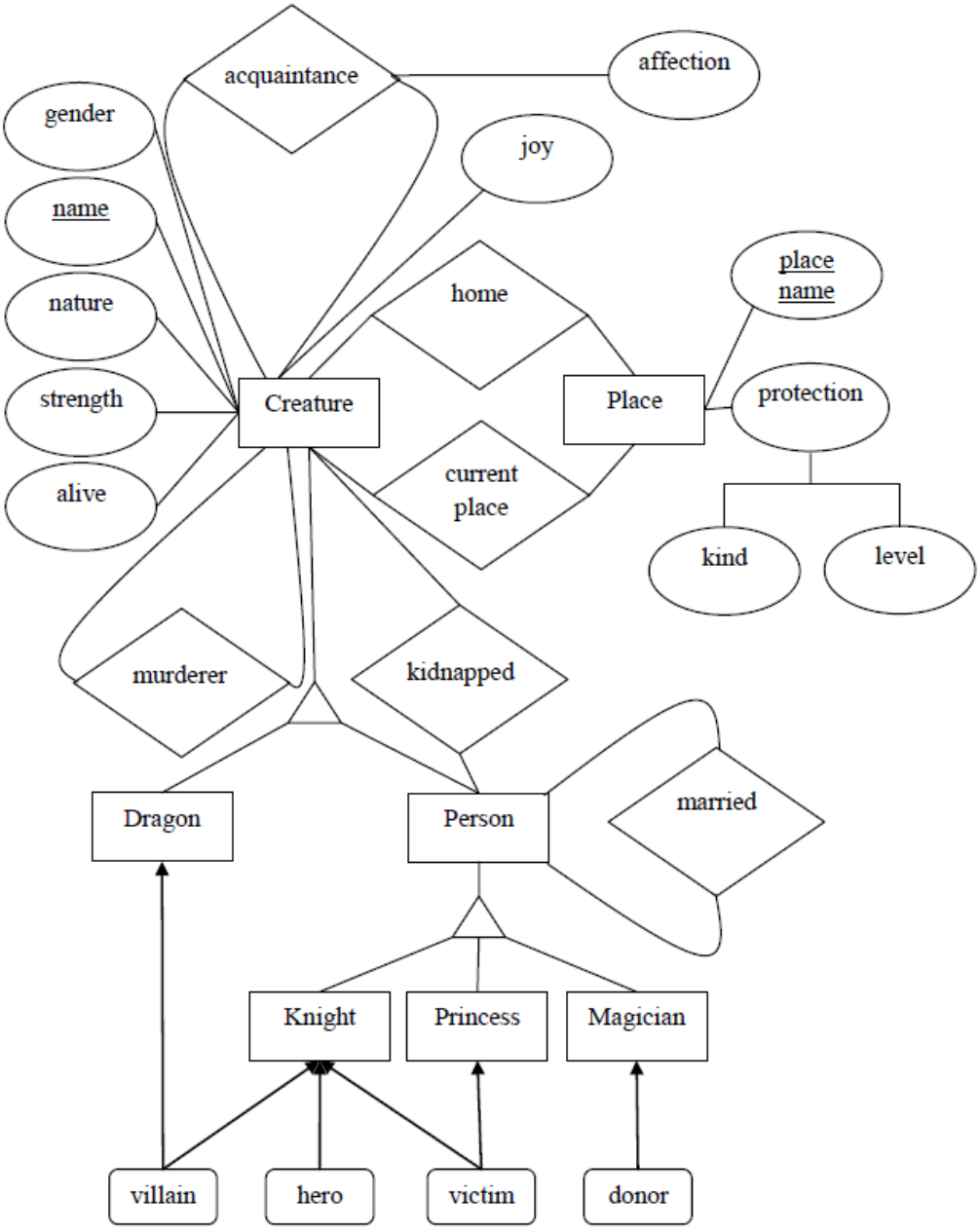

Figura 30. Diagrama do do gênero "Espadas & Dragões" (SILVA, 2010).

I.2.2 Contexto Exemplo

Como instância do gênero apresentado na seção anterior, o LOGTELL tem um contexto exemplo com o estado inicial dos personagens, locais e das suas relações. O contexto é inserido no programa através de fatos que são instâncias das relações possíveis no gênero. Nesta seção, ao invés de uma listagem dos fatos, o contexto é

apresentado do ponto de vista de uma descrição, centrada nos personagens e nos lugares.

Personagem	Descrição
Marian	A belíssima princesa do palácio *White Palace*, que, apesar de sua natureza bondosa, tem suas desavenças com o rei, figura paterna dominadora. *Marian* deseja viver uma vida de aventuras, mesmo que (ou motivada por) incorra no risco de virar uma vítima. Apesar de possuir muitos pretendentes, não é particularmente inclinada para nenhum; é uma donzela difícil de ser conquistada.
Brian	O cavaleiro mais forte do castelo *Grey Castle*, de natureza bondosa, apaixonado pela *Marian* e ambicioso para se tornar um herói.
Hoel	Outro cavaleiro de natureza bondosa do castelo *Grey Castle*. Não tão forte quanto *Brian*, mas nem por isso menos ambicioso. Alcançar o status e o poder de um herói é o que norteia os seus treinamentos diários. Possui em seu coração uma paixão de longa data pela princesa *Marian*.
Turjan	Um homem isolado, de poucas palavras, e de natureza neutra. Vive na floresta *Green Forest* e lá desenvolve suas atividades de mago. Ele pode usar seus poderes mágicos para doar força a um herói, mas também diminui-la. Todo herói sabe que é melhor visita-lo num dia de bom humor.
Draco	*Draco*, o temido dragão do castelo *Red Castle*, de natureza maldosa e o grande vilão do reino. Seus poderes só são equiparados pelos de *Turjan*. Qualquer herói que deseje enfrenta-lo precisará da ajuda do inconstante mago.

Figura 31. Descrição dos personagens do contexto do gênero "Espadas & Dragões".

Local	Descrição
White Palace	O palácio do reino, liderado por um rei respeitado e sério. A vida no palácio é segura e pacata, isto é, se *Draco* não resolver aparecer por lá.
Red Castle	Pouco se sabe desse terrível castelo. Ninguém que já tenha ido lá voltou para contar a história.
Grey Castle	Um castelo fortificado e erguido para treinamento dos cavaleiros do reino. Se você gosta de uma boa luta, lá é o seu lugar.
Green Forest	Uma floresta misteriosa, que muitos afirmam mudar de configuração a cada vez que alguém a visita. Há muitos boatos, o certo é que lá onde vive *Turjan*, o mago do reino, desenvolvendo sua arte da magia.
Church	A igreja do reino e o lugar onde todos os homens desejariam estar segurando a mão de *Marian*. Um local sagrado que certamente pertence aos sonhos de *Brian* e *Hoel* quando pensam na bela princesa.

Figura 32. Descrição dos locais do contexto do gênero "Espadas & Dragões".

I.2.3 IPG (*Interactive Plot Generator*)

O IPG é uma ferramenta para a geração de enredos que se utiliza de um planejador de ordem parcial (*planejamento no espaço de planos*) que gera os enredos como uma sequência de capítulos a partir da modelagem de eventos como operadores de planejamento clássico. Cada capítulo é gerado dentro de um ciclo onde as interferências do usuário são incorporadas e, juntamente às regras de inferência de objetivos, levam o planejador a gerar uma ordem parcial de eventos. O IPG se mostrou capaz de gerar uma diversidade expressiva de diferentes enredos para o gênero "Espadas & Dragões", incorporando interações com os usuários, mas sempre garantindo a coerência lógica.

A geração de enredos começa com a situação inicial do mundo definida pelo contexto. Então, a partir da avaliação das regras de inferência de objetivos usando lógica temporal, objetivos são inferidos para serem atingidos no próximo capítulo. O IPG utiliza planejamento no espaço de planos para compor uma ordem parcial de eventos que parta do estado inicial e atinja os objetivos. Uma vez que os objetivos sejam atingidos pelo enredo até o momento, a sequência de estados pelas quais o enredo pode passar criam novas situações que podem levar a novos objetivos a serem atingidos. A interação com o usuário também pode levar a novos objetivos. Desse modo, o novo ciclo se inicia com a inferência de novos objetivos e a incorporação dos objetivos resultantes da interação com o usuário. O planejamento é então aplicado novamente e

assim por diante os capítulos se sucedem. O processo para quando não há novos objetivos a inferir nem interações com o usuário que levem à continuação da história.

Como o planejador examina diversos planos para cada capítulo, um tipo de interação básica disponibilizada pelo IPG é permitir que o usuário solicite uma outra solução para um capítulo passado e a história recomece a partir desse ponto. Alternativamente, também é possível retornar a um ponto anterior, sem modificar o capítulo para o qual se retorna, com a finalidade de interagir de modo a modificar os capítulos seguintes.

ANEXO II – MODELO E IMPLEMENTAÇÃO DE GERAÇÃO DE ENREDOS COM NÃO DETERMINISMO

II.1 Arquitetura

Nesta seção é apresentada a arquitetura do modelo e explica-se como ela atende aos cinco requisitos de: suporte a eventos não determinísticos; controle do nível de não determinismo; incorporação do conceito de tentativas; processamento em paralelo das diversas alternativas; e eficiência na geração de enredos. A arquitetura buscou ser compatível com a arquitetura geral do LOGTELL e, nela, substitui as instâncias do planejador de ordem parcial (*IPG Instances*) pelo módulo *NDet-IPG*. Neste módulo, o planejamento de enredos passa a ser feito de forma conjunta entre o planejador de ordem parcial e o planejador HTN não determinístico em um submódulo denominado *Chapter Simulator*, responsável pela geração dos capítulos. Como pode haver mais de um planejamento ocorrendo em paralelo, há diversas instâncias de *Chapter Simulator* e estas são controladas pelo submódulo *Chapter Controller*.

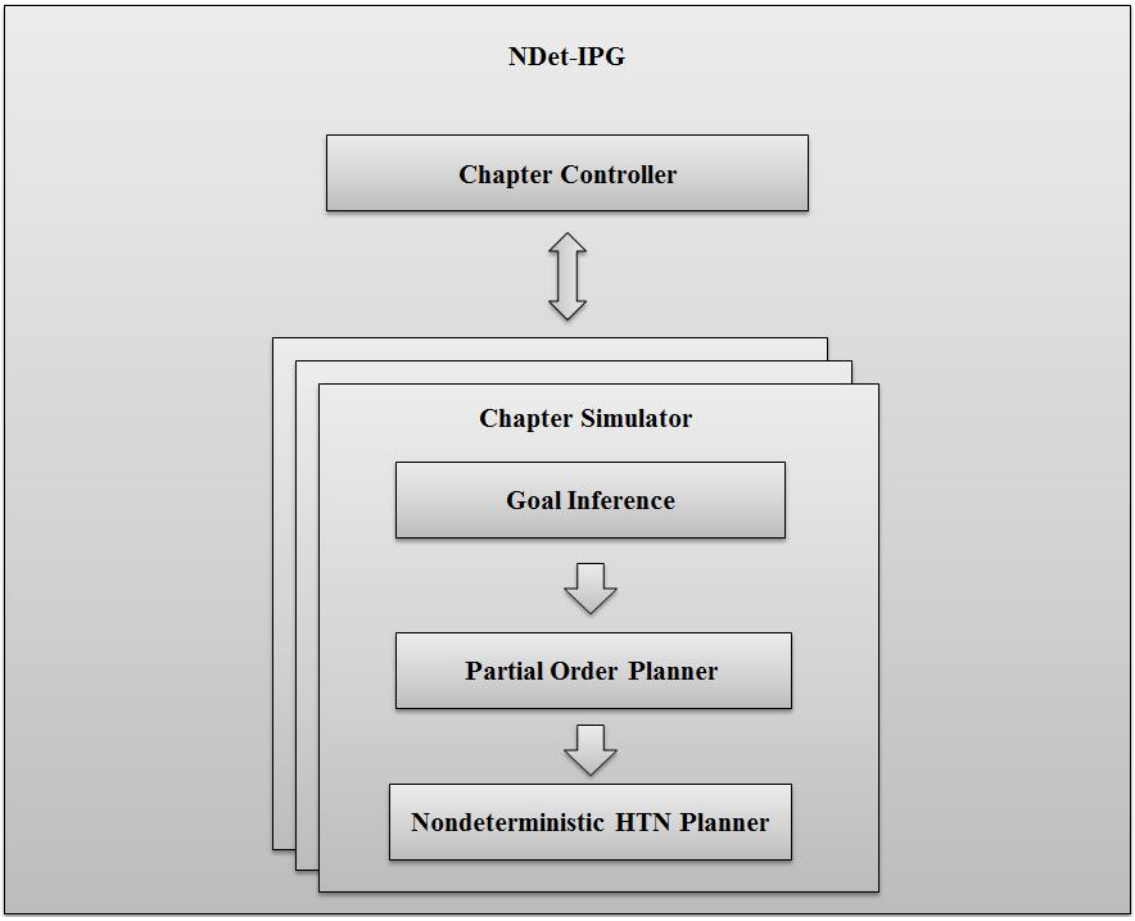

Figura 33. Módulo NDet-IPG (SILVA, 2010).

A integração entre o planejamento de ordem parcial e o planejamento HTN é importante para garantir a diversidade dos enredos e a compatibilidade com o esquema de geração de enredos e interação do LOGTELL, na medida em que se garante também uma maior eficiência, capaz de tornar o tratamento do não determinismo viável. Como apresentado anteriormente, No LOGTELL ocorrem inferência de objetivos e interações fortes onde o usuário insere eventos e situações explicitamente a serem incorporadas em um próximo capítulo. Tais elementos são inseridos no enredo e o planejamento de ordem parcial é usado para tornar esse enredo coerente novamente. Com o planejamento de ordem parcial, consegue-se combinar eventos mesmo que não se tenha soluções previamente prontas para atender parte dos objetivos, o que torna o processo mais flexível do que quando se utiliza apenas planejamento HTN. A ideia central é permitir a combinação de eventos básicos e principalmente de eventos mais abstratos, que precisam de decomposição, de modo a gerar uma rede de tarefas a ser cumprida. Os eventos dessa rede de tarefas passaram a poder ser não determinísticos e foi incorporado

ao planejamento de ordem parcial tanto o tratamento desse não determinismo quanto do conceito de tentativa.

Desta forma, o processo de planejamento pode seguir o fluxo de incorporação de interferências fortes dos usuários, inferência de objetivos, planejamento de uma ordem parcial para atingi-los e detalhamento desta ordem parcial em um capítulo em forma de árvore. Este processo pode ser dividido em duas partes: primeiro a geração de um esboço do plano com planejamento de ordem parcial e depois o seu detalhamento usando planejamento HTN não determinístico. A fim de que se tenha uma boa integração, as regras de inferência de objetivos têm de ser genéricas o suficiente para que o planejador de ordem parcial gere enredos bem menos detalhados. Assim, é necessário definir eventos de alto nível correspondendo a tarefas HTN não básicas cujos objetivos casem com os objetivos das regras de inferência de objetivos. Espera-se assim que a fase de planejamento de ordem parcial se torne bem mais rápida.

O ciclo que envolve o módulo *NDet-Partial-Order Planner* começa com a inferência de objetivos a partir do estado inicial, a geração de um esboço e o detalhamento para gerar o primeiro capítulo. A partir do primeiro capítulo, o ciclo continua levando em conta interferências do usuário recebidas pelo *Chapter Controller* e distribuídas, para execução em paralelo, entre as instâncias de *Chapter Simulator* conforme necessário. Assim que um capítulo é gerado, para cada final possível deste, o *Chapter Controller* cria uma instância de *Chapter Simulator* para gerar possíveis continuações. Com esse novo esquema de geração de enredos, incorporando o tratamento do não determinismo, tornou-se possível que novos tipos de interação com usuário sejam incorporados ao LOGTELL, fazendo com que o usuário influencie diretamente o capítulo que está sendo apresentado e não apenas capítulos futuros. Caso o caminho que está sendo seguido na dramatização de um capítulo impossibilite que se alcance um dos possíveis finais, para a qual uma das instâncias está planejando a continuação, a execução dessa instância pode ser interrompida.

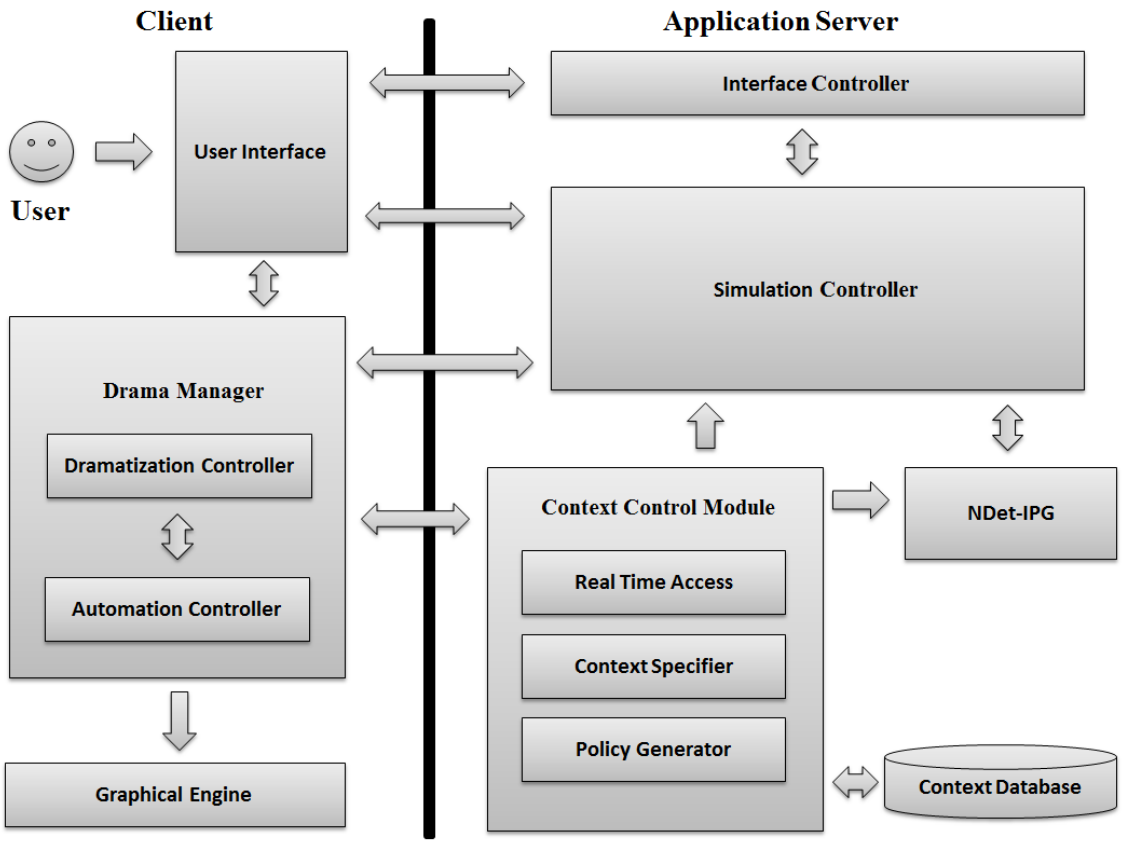

Figura 34. Nova arquitetura do LOGTELL

II.1 Controle do Nível de Não Determinismo

O não determinismo pode levar a uma expansão muito grande do número de estados em um plano. Por questão de eficiência, e para evitar uma explosão combinatória, é interessante que se possa controlar o nível de não determinismo. Isto pode ser feito através do estabelecimento de um grau máximo de não determinismo para representar o número máximo permitido de estados finais para um plano. Desta forma, durante o processo de planejamento, se descartaria planos que ultrapassem esse limite.

II.2 Incorporação do Conceito de Tentativas

Para incorporar o conceito de tentativas adotou-se uma estratégia de satisfação de objetivos estendidos (i.e. que devem ser satisfeito ao longo de um plano e não somente no final deste). Portanto, os objetivos inferidos pelas regras de inferência de objetivos passaram a poder mencionar *objetivos fracos* de tentativa de estabelecimento de certo fato. Adicionalmente, as precondições dos eventos passaram a poder ter a especificação de que o estabelecimento de um fato tenha sido tentado. Tais extensões

demandaram que tratamento específico fosse incorporado tanto ao planejador de ordem parcial quanto ao planejador HTN não determinístico.

As precondições dos operadores e os objetivos inferidos a partir das regras passaram a conter expressões do tipo *tried(L,N)*, onde *L* é um literal e *N* é um número que indica o nível de tolerância para eventos que visem estabelecer *tried(L,N)*. N define o quão antes na cadeia de eventos pode haver uma falha que impeça o real estabelecimento de *L*. Eventos inseridos em um plano em razão de um objetivo fraco são marcados como tentativas (*attempts*) e serão incorporados ao plano final apenas se suas precondições forem satisfeitas.

II.3 Processamento em Paralelo

Além do planejamento HTN como recurso à busca de eficiência na geração, pode-se pensar em paralelizá-la. A utilização da estratégia de paralelização, ao aumentar a eficiência, libera o planejamento HTN para permitir graus maiores de não determinismo, gerando histórias mais complexas.

O processamento em paralelo pode ser feito considerando como estado inicial cada um dos estados finais do plano gerado anteriormente. Deste modo, é possível observar o andamento do enredo e caso este enverede por um caminho que torne impossível alcançar certos estados finais, os planos sendo gerados em paralelo para estes finais deveriam ser descartados.

II.4 Implementação

O modelo apresentado na seção anterior foi incorporado ao ambiente do LOGTELL, estendendo o contexto original para o gênero “Espadas & Dragões”. O desenvolvimento de novos eventos, operadores e personagens buscou a criação de um novo contexto rico o suficiente para o teste do protótipo.

Para a explicação da implementação do algoritmo de planejamento HTN não determinístico NDet-HTN, é necessária uma apresentação do novo contexto criado. Todavia, primeiramente é explicado as mudanças no IPG para a incorporação de tentativas e, depois, o novo contexto para então apresentar o algoritmo NDet-HTN e os testes realizados.

II.4.1 Mudanças no IPG

O IPG precisou ser modificado para dar se integrar ao novo módulo *NDet-IPG* de planejamento híbrido. O IPG passa a tratar efeitos não determinísticos para estabelecer objetivos fracos (tentativas) e também para serem considerados como possíveis interferências no estabelecimento de precondições. Adicionalmente, como cada tentativa tem um grau de tolerância, também foi criado um mecanismo para o cálculo e propagação desse grau para os eventos a serem encadeados. Este mecanismo visa garantir que a parte inicial da sequência de eventos encadeada para estabelecer um objetivo fraco seja considerada, mas que eventos dentro do grau de tolerância fiquem indicados como tentativas. Esta indicação permite ao planejador HTN saber que as tentativas correspondem a tarefas que não precisam, necessariamente, ocorrer.

Além das duas modificações mencionadas, uma terceira modificação foi permitir ao IPG lidar com eventos básicos e complexos em um mesmo nível. Isto garante flexibilidade para atender objetivos diversos, ao mesmo tempo em que possivelmente diminui o trabalho de decomposição a ser feito pelo planejador HTN. Os eventos complexos passaram a ter efeitos que explicitamente indicam que uma tentativa, com certo grau de tolerância, ocorrerá se o evento ocorrer. Esta indicação poupa o IPG de ter de buscar por sequências de eventos para atingir uma tentativa.

II.4.2 Mudanças no Gênero "Espadas & Dragões" e no Contexto

O LOGTELL originalmente utiliza um o gênero "Espadas & Dragões". Há adição no gênero de alguns novos operadores complexos e básicos para representar eventos de nível mais alto e eventos básicos determinísticos e não determinísticos. Estes operadores foram adicionados aos que já existiam no LOGTELL Além disso, foi adicionada uma nova entidade, *nun*, que é apresentada após os novos operadores.

Novo Evento Determinístico	Descrição
commit_suicide(CH)	O personagem *CH* se suicida.
do_evil_laugh(CH)	O personagem *CH* emite uma risada diabólica.
remove_skin(CH1,CH2)	O personagem *CH1* esfola o personagem *CH2*
join_convent(CH)	O personagem *CH* adere a um convento
make_funeral(CH1,CH2)	O personagem *CH1* realiza o funeral do

	personagem *CH2*
roar(CH)	O personagem *CH* emite um rugido
spread_horror(CH)	O personagem *CH* dissemina medo pelo território à sua volta

Figura 35.Novos eventos determinísticos para o gênero "Espadas & Dragões".

Novo Evento Não Determinístico	Descrição
try_to_kidnap(CH1,CH2)	O personagem *CH1* tenta raptar o personagem *CH2*. O evento possui dois finais possíveis. No primeiro, *CH1* consegue raptar *CH2*. No segundo, o personagem *CH2* acaba morrendo durante o rapto.
fight_to_death(CH1,CH2)	O personagem *CH1* luta com o personagem *CH2* até a morte. O evento possui três finais possíveis, podendo resultar na morte de um, de outro, ou de ambos.

Figura 36. Novos eventos não determinísticos para o gênero "Espadas & Dragões".

Além dos novos eventos básicos, o autor também criou uma série de eventos complexos (i.e. abstratos, compostos ou tanto abstrato como composto) a fim de se enriquecer os enredos e testar o seu algoritmo *NDet-HTN* de planejamento HTN não determinístico. Estes novos eventos complexos se constroem tanto em cima dos novos eventos básicos como dos eventos que já havia no LOGTELL. O planejador IPG já fazia uso de hierarquias para um uso alternativo à geração de enredos que era o reconhecimento de planos típicos (CIARLINI *et al*., 2010). Todavia, o gênero de exemplo do LOGTELL continha somente eventos básicos, diretamente dramatizáveis, e, assim, se fez necessária toda uma nova formulação de eventos complexos.

Nas duas figuras 21 e 22 acima, há diferentes operadores complexos e suas

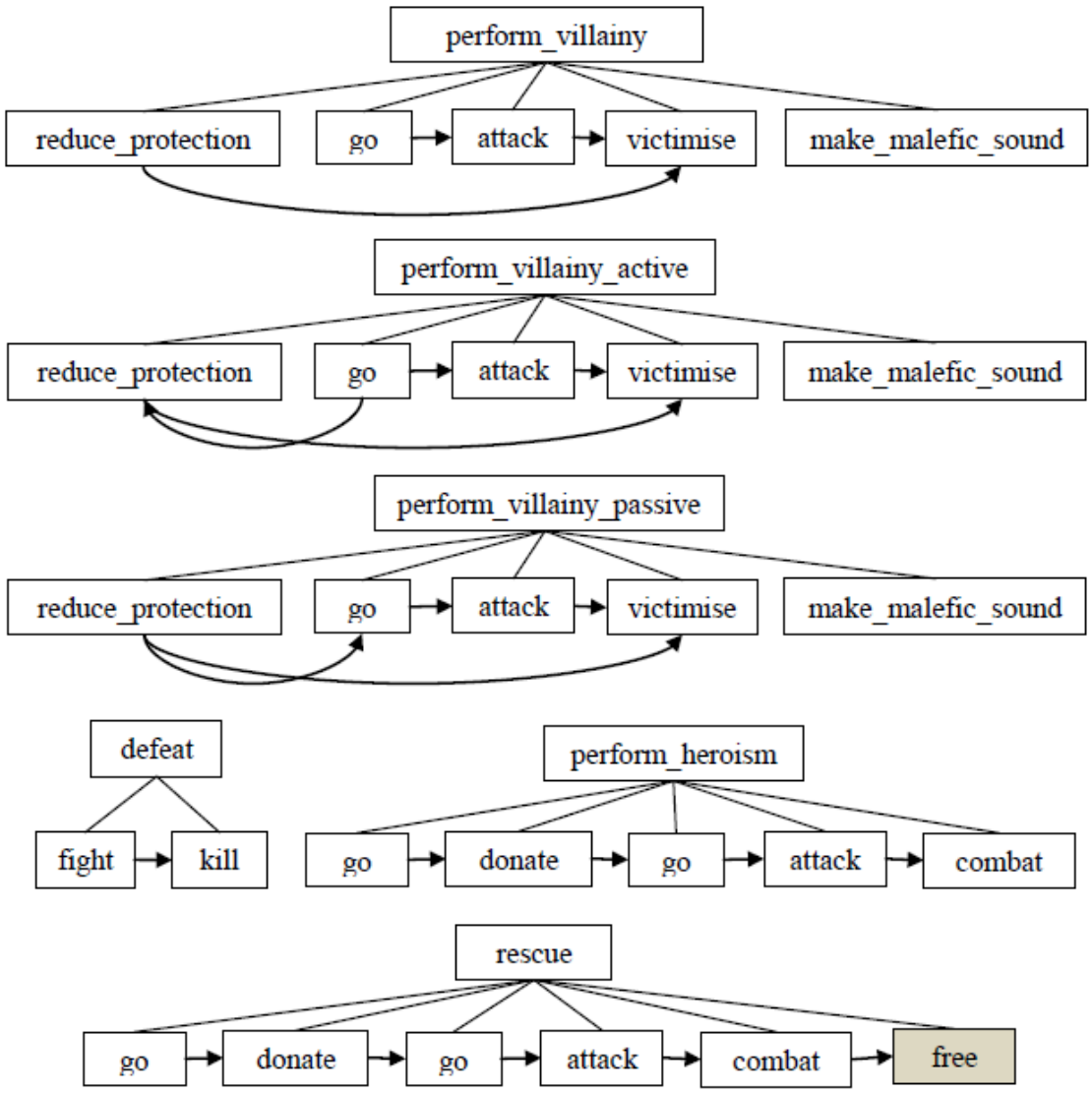

Figura 37. Hierarquia de eventos no gênero "Espadas & Dragões" (SILVA, 2010).

Figura 38. Eventos compostos para o gênero "Espadas & Dragões" (SILVA, 2010).

especializações ou suboperadores. Por exemplo, o operador *perform_heroism*, que pode ser especializado em *rescue* ou *avenge*, sendo que ambos são, por sua vez, operadores compostos. O operador *rescue* herda a sub-HTN de *perform_heroism*, e adiciona a ela o operador *free*, neste caso, colocado explicitamente como uma tentativa *attempt(free(HER,VIC),1)*. O operador *perform_vilainy* tem como um de seus filhos o operador *perform_villainy_active*, que herda a sub-HTN do pai, mas adiciona a ela uma restrição de ordem entre os suboperadores *go* e *reduce_protection*. Ambos os operadores *perform_heroism* e *perform_villainy* são tanto abstratos como compostos. O operador *defeat* é um exemplo de operador composto, mas não abstrato e o operador *victimise*, de operador abstrato, mas não composto.

Em relação ao novo contexto, o autor adicionou um novo vilão, *Lucius*, apaixonado pela princesa *Marian*, e um novo local, *Black Castle*, para ser o lar de *Lucius*. Desta maneira, se torna possível obter novos enredos onde há um vilão diferente do dragão *Draco*.

II.6.3 O Algoritmo NDet-HTN

Com a apresentação da nova modelagem de operadores para se adequar a uma modelagem HTN e o gênero e contexto de exemplo, o algoritmo HTN não determinístico NDet-HTN pode ser agora apresentado. Este algoritmo é inspirado no algoritmo PFD (Figura 2) apesar de que este retorna uma ordem total de ações, e o NDet-HTN, uma árvore de contingências. A inspiração se dá principalmente na maneira pela qual o NDet-HTN decide a ordem em que se seleciona a próxima tarefa a ser tratada em uma ordem parcial de tarefas a ser tratada. Todavia, o NDet-HTN possui uma estrutura própria com o seu tratamento de não determinismo.

Os dados de entrada para o algoritmo são o estado inicial do mundo, o estado atual do mundo, a rede de tarefas, o conjunto de operadores do domínio e o grau máximo de não determinismo permitido. O retorno do algoritmo é um plano e os estados finais alcançados por este plano. No que concerne ao pseudocódigo que será apresentado, os dados de entrada e de retorno são:

- Dados de entrada:
    - *estado_inicial*: um conjunto de átomos “ground” que descrevem o contexto inicial;
    - *estado_atual*: uma dupla $<Fatos^{+},Fatos^{-}>$. $Fatos^{+}$ é um conjunto de átomos “ground” representando os fatos adicionados no

mundo em relação ao estado inicial. *Fatos*$^{-}$ é um conjunto de átomos "ground" representando os fatos removidos do mundo em relação ao estado inicial;

- *w*: uma rede de tarefas (HTN)
- *O*: o conjunto de operadores do gênero, incluindo definições de especialização e decomposição;
- *grau_ndet*: O grau máximo de não determinismo permitido (número máximo de finais possíveis para o plano).

- Dados de retorno:
  - π: o plano, representando um capítulo, com estrutura de árvore, onde cada nó é um evento básico e cada aresta, uma possibilidade não determinística. Cada aresta tem associada a ela, um conjunto de fatos que determina as precondições necessárias para seguir por aquela ramificação;
  - *estados_finais*: os estados finais do plano. É importante notar que os estados finais não são as folhas, mas sim os finais possíveis para cada folha (que pode representar um evento determinístico ou não).

NDet-HTN(*estado_inicial*, *estado_atual*, *w*, *O*, *grau_ndet*, *estados_finais*)

Se *grau_ndet* <= 0 então retorne *PLANO_VAZIO* *// Restrição de valor*

Se *w* = Ø então retorne *PLANO_VAZIO*

$fatos^+$ ← {fatos positivos de *estado_ atual*}

$fatos^-$ ← {fatos negativos de *estado_ atual*}

estado_atual ← *estado_inicial* ∪ $fatos^+$ - $fatos^-$

Escolha não-deterministicamente qualquer $u \in w$ sem predecessores em *w*

// Impõe explicitamente restrição de ordem para que seja mantida caso o operador associado ao vértice seja complexo

Para cada vértice $i \neq u$ em *w*

Acrescente, caso ainda não exista, a restrição de ordem (*u*, *i*) a *w*

Fim-para

// t'_u = tarefa, sem o modificador "attempt"(caso tenha), associada ao vértice "u"

// N = tolerância a falhas (zero se não for tentativa)

Se *u* for do tipo "*attempt*" então

t'_u ← tarefa que a tentativa procura estabelecer

N ← grau de tolerância a falhas da tentativa

Senão

$t'_u \leftarrow t_u$

N ← 0

válidos ← {(*a*, σ) | *a* é instância "ground" de um operador em *O*,

σ é uma substituição tal que *nome*(*a*) = $\sigma(t'_u)$}

Se *válidos* = Ø então retorne *FRACASSO* *// Deve tratar todas as tarefas*

Escolha não-deterministicamente algum (*a*, σ) ∈ *válidos*

Se *a* não é aplicável a *estado_atual*

Se *N* > 0 então *// Trata-se de uma tentativa (no caso, fracassada)*

w' ← *w* – {*u*} *// Remove o vértice que resultou em fracasso*

```
            π' ← NDet-HTN(estado_inicial, estado_atual, w', O, grau_ndet,
        estados_finais)
        Senão
            Retorne FRACASSO  // Tarefa não aplicável, e não é uma tentativa
// A tarefa se aplica ao estado atual (sendo tentativa ou não, vai ser cumprida)
Se t'u é um operador básico então
    w' ← σ(w – {u})              // Remove tarefa do HTN
    estado_det ← aplicação de efeitos determinísticos de a sobre estado_atual
    Se w' ≠ HTN_VAZIO então
        Se t'u não tem efeitos não-determinísticos então
            // Monta subplano
            π'' ← NDet-HTN(estado_inicial, estado_det, w', O, grau_ndet,
            estados_finais)
            π' ← [VERDADE, π'']
        Senão
            π' ← Ø
            Para cada efeito não-determinístico efeito_ndet de a
                estado_ndet ← aplicação de efeito_ndet sobre estado_det
                π'' ← NDet-HTN(estado_inicial, estado_ndet, w', O,
                grau_ndet, estados_finais)
                Se π'' = FRACASSO então retorne FRACASSO
                Senão  π' ← π' ∪ {[efeito_ndet , π'']}
            Fim-para
            estados_finais ← {estados finais alcançáveis por π'}
            Se tamanho(estados_finais) > grau_ndet então
                Retorne FRACASSO
    Senão  // Se w' = HTN_VAZIO
        // Incorporará subplano vazio ao plano, fechando aquele ramo
        π' ← Ø
        Se t'u não tem efeitos não-determinísticos então
            estados_finais ← {estado_det}
```

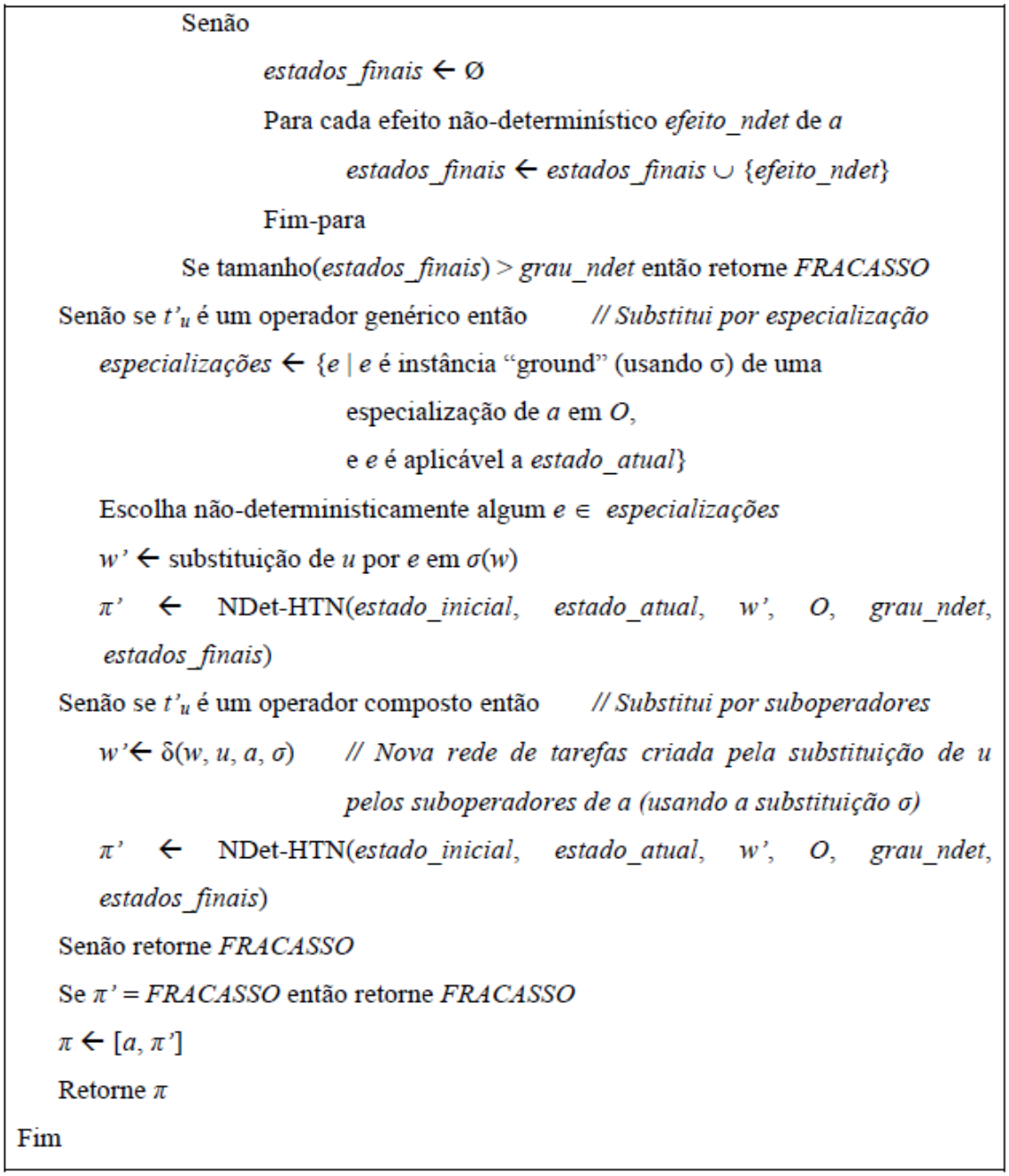

Figura 39. Algoritmo NDet-HTN (SILVA, 2010).

O algoritmo é recursivo, com chamada inicial:

NDet-HTN(estado_inicial,Ø,htn_inicial,O,grau_ndet,Ø)

O plano retornado pelo algoritmo é uma dupla <*action,subtree*>, onde *action* é uma instância "ground" de um operador representando um evento, e *subtree* é uma lista de duplas da forma <*contigency,subplan*>, onde *contingency* é um conjunto de literais e *subplan* é definido recursivamente com a mesma estrutura de um plano. Eventos finais de um plano tem uma lista vazia como *subtree*. Eventos determinísticos tem o conjunto

vazio como *contigency*, e eventos não determinísticos, têm os efeitos de cada final como *contingency* para a subárvore por aquele ramo.

O algoritmo possui pontos de backtracking, presentes toda vez em que se faz uma escolha não determinística. Cada escolha em um ponto destes representa alternativas que podem levar a diferentes planos que resolvem o problema de planejamento. Especificamente, temos três destes pontos de backtracking no algoritmo:

- A escolha de uma das tarefas sem predecessor na HTN, que pode levar a diferentes ordens de ancestralidade entre eventos no plano final;
- A escolha de um par em *validos*, que representa uma instância de operador (evento) que visa cumprir a tarefa corrente e pode levar a presença de diferentes eventos no plano; e
- A escolha de uma especialização para um operador abstrato, que pode levar a planos diversos para cada desdobramento de especializações em outros operadores básicos ou não.

De maneira mais detalhada, o plano final gerado é uma árvore onde os nós representam instâncias de operadores, ou seja, eventos básicos determinísticos ou não, e as arestas representam as ramificações para cada final do evento. Cada nó tem associado tem associado a ele um conjunto de precondições que são as precondições "ground" do operador. Desta forma, é possível, em tempo de dramatização, ou execução do plano, decidir por qual caminho seguir. Todavia, é importante notar que todo caminho no plano final é válido para se atingir os objetivos a partir da raiz e do estado inicial do contexto.

O não determinismo exige que o algoritmo trate de maneira consistente a atualização do estado do mundo para seguir com o planejamento por cada uma das alternativas referentes aos diferentes finais de um evento não determinístico. Deste modo, quando o planejador se depara com um evento básico, ele atualiza o estado do mundo com seus efeitos determinísticos e, se o evento for não determinístico, cria uma ramificação para cada final a partir desse novo estado atualizado. No que concerne à continuação do planejamento, cada um dos ramos passa a ter o seu próprio estado atual em função dos diferentes efeitos não determinísticos do evento.

É importante lembrar que as especializações capturam o conceito de método do planejamento HTN. Os métodos de cumprir uma tarefa são as diferentes especializações, que podem ser um operador básico ou outro operador complexo que implicará em mais decomposições. Somente especializações aplicáveis no estado atual podem ser escolhidas, isto é, somente aquelas cujas precondições são satisfeitas pelo estado atual.

A HTN de entrada para o algoritmo pode vir com operadores marcados como tentativas, já que o NDet-HTN pode estar funcionando em vínculo com um planejador de ordem parcial que incorpore esse conceito. Operadores marcados como *attempt(OP,N)* com $N = 0$ indicam que *OP* deve ser aplicável no estado atual, caso contrário, o planejamento nesse ponto deve falhar. Quando $N > 0$, e o operador *OP* é aplicável ao estado atual, ele pode ser considerado para o planejamento e a marcação *attempt* é, então, ignorada pelo processo de planejamento. Todavia, se $N > 0$ e o

```
// Atualiza lista de efeitos adicionados e removidos do estado inicial
Atualiza_Estado(estado_inicial, estado_atual, efeitos)
        fatos+ ← {fatos positivos de estado_ atual}
        fatos- ← {fatos negativos de estado_ atual}
        efeitos+ ← {efeitos positivos de efeitos}
        efeitos- ← {efeitos negativos de efeitos}
        // Acrescenta efeitos positivos que ainda não façam parte do estado atual
        Para cada efeito em efeitos+ faça:
            Se efeito ∈ fatos- então remova efeito de fatos-
            Senão
                    Se efeito ∉ estado_inicial então adicione efeito a fatos+
        Fim-para
        // Remove efeitos negativos que ainda façam parte do estado atual
        Para cada efeito em efeitos- faça:
            Se efeito ∈ fatos+ então remova efeito de fatos+
            Senão
                    Se efeito ∈ estado_inicial então adicione efeito a fatos-
        Fim-para
Fim
```

Figura 40. Algoritmo de atualização de estados do NDet-HTN (SILVA, 2010).

operador *OP* não é aplicável ao estado atual, então, ele é removido do processo de planejamento, que continua como se a tentativa nunca estivesse estado na rede de tarefas.

O controle do estado corrente no algoritmo é mantido na forma compactada supramencionada de uma dupla com os fatos acrescentados e removidos do estado inicial. O objetivo principal dessa representação é evitar que se tenha uma estrutura de dados imensa representando o estado corrente do mundo, que precisaria ficar transitando em memória entre as diferentes funções do algoritmo. É necessária uma lógica para garantir a consistência da representação compacta, que é garantida pelo algoritmo de atualização de estados (Figura 40). Este algoritmo recebe como entrada o estado atual, o estado corrente em forma de dupla e os efeitos do operador também em forma de dupla contendo os $efeitos^{+}$ e $efeitos^{-}$.