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

# GERAÇÃO DE ENREDOS COM PLANEJAMENTO NÃO-DETERMINÍSTICO EM STORYTELLING PARA TV INTERATIVA

Fabio Araujo Guilherme da Silva

**Orientador**
Prof. Dr. Angelo Ernani Maia Ciarlini
**Co-Orientador**
Prof. Dr. Sean Wolfgand Matsui Siqueira

RIO DE JANEIRO, RJ – BRASIL
Setembro de 2010

GERAÇÃO DE ENREDOS COM PLANEJAMENTO NÃO-DETERMINÍSTICO EM STORYTELLING PARA TV INTERATIVA

Fabio Araujo Guilherme da Silva

DISSERTAÇÃO APRESENTADA COMO REQUISITO PARCIAL PARA OBTENÇÃO DO TÍTULO DE MESTRE PELO PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA DA UNIVERSIDADE FEDERAL DO ESTADO DO RIO DE JANEIRO (UNIRIO). APROVADA PELA COMISSÃO EXAMINADORA ABAIXO ASSINADA.

Aprovada por:

_____
Prof. Angelo Ernani Maia Ciarlini, D.Sc. – UNIRIO

_____
Prof. Sean Wolfgand Matsui Siqueira, D.Sc. – UNIRIO

_____
Profa. Flávia Maria Santoro, D. Sc. – UNIRIO

_____
Prof. Antonio Luz Furtado, Ph.D. – PUC-RIO

_____
Prof. Cesar Tadeu Pozzer, D.Sc. – UFSM

RIO DE JANEIRO, RJ - BRASIL
SETEMBRO DE 2010

Silva, Fabio Araujo Guilherme da.
S586 Geração de enredos com planejamento não-determinístico em storytelling para TV interativa / Fabio Araujo Guilherme da Silva, 2010.
xi, 120f.

Orientador: Angelo Ernani Maia Ciarlini.
Coorientador: Sean Wolfgand Matsui Siqueira.
Dissertação (Mestrado em Informática) – Universidade Federal do Estado do Rio de Janeiro, Rio de Janeiro, 2010.

1. Inteligência artificial. 2. Storytelling interativo. 3. Entretenimento digital. 4. Planejamento. I. Ciarlini, Angelo Ernani Maia. II. Siqueira, Sean Wolfgand Matsui. III. Universidade Federal do Estado do Rio de Janeiro (2003-). Centro de Ciências Exatas e Tecnologia. Curso de Mestrado em Informática. III. Título.

CDD – 006.3

## Agradecimentos

À Debora, amiga e namorada, por todo o apoio dado do começo ao fim dessa empreitada, me incentivando e evitando que minha vida se tornasse um verdadeiro caos durante esse longo e, muitas vezes, extenuante período.

A meus orientadores, Angelo Ciarlini e Sean Siqueira, por, fazendo jus ao título, terem prestado uma verdadeira e inestimável orientação. Demonstrando uma aptidão singular para compreender meus anseios acadêmicos, seu auxílio foi fundamental para o estabelecimento dos rumos de minha pesquisa, fazendo com que esta se tornasse verdadeiramente prazerosa, apesar dos inúmeros percalços. Sua paciência e confiança foram fatores decisivos em momentos cruciais do desenvolvimento deste trabalho.

Aos demais professores do Programa de Pós-Graduação em Informática (PPGI) da UNIRIO; os ensinamentos fornecidos durante suas aulas fizeram com que meu retorno à vida acadêmica, ocorrido após um longo hiato de 13 anos desde minha graduação, fosse tão estimulante que acabei dando prosseguimento imediato após o mestrado. Agradeço também pelas críticas e sugestões valiosas apresentadas nos seminários de acompanhamento discente.

Aos membros da banca examinadora pelas observações que tanto contribuíram para o enriquecimento da versão final desta dissertação.

Aos colegas do mestrado, em especial aqueles com quem tive mais contato durante a pesquisa, Augusto Baffa e Gustavo Valfre, por terem tornado a experiência extenuante de aulas, provas e trabalhos mais agradável.

Aos funcionários do PPGI, sempre atenciosos e prestativos, com uma menção especial à Alessandra Nascimento por sempre ter me ajudado com paciência e simpatia constantes.

À Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), pela bolsa concedida durante o curso.

Aos meus gestores e colegas na empresa Elumini IT & Business Consulting, pelo grande apoio e compreensão que permitiram que eu finalmente conseguisse realizar o sonho, mantido durante anos, de entrar para um curso de mestrado.

Por fim, a todos os que não estão aqui mencionados nominalmente, mas que também foram personagens de grande importância do enredo criado e dramatizado para permitir a conquista dessa meta tão difícil de ser alcançada, e para cuja obtenção dificilmente haverá planejador capaz de tratar devidamente tantos eventos não-determinísticos que se apresentam durante uma jornada tão árdua, mas que, sem dúvida, compensa.

SILVA, Fabio Araujo Guilherme da. **Geração de Enredos com Planejamento Não-Determinístico em Storytelling para TV Interativa**. UNIRIO, 2010. 120 páginas. Dissertação de Mestrado. Departamento de Informática Aplicada, UNIRIO.

## RESUMO

*Storytelling* interativo é um novo tipo de entretenimento digital em que os usuários interagem com uma história sendo contada. Neste contexto, algoritmos de planejamento automático são alternativas interessantes para criar tramas que são coerentes com o gênero pretendido. Entretanto, diversidade de histórias e possibilidades de interação são outros requisitos fundamentais que devem ser considerados por tais aplicações. Também é importante gerar histórias incorporando tentativas fracassadas de se atingir objetivos, uma vez que isto é essencial para que se crie tensão dramática. Neste trabalho, é proposto um modelo que leva em conta essas questões por meio de planejamento com eventos não-determinísticos, ou seja, eventos com mais de um resultado diferente possível. Para viabilizar tal modelo, foi desenvolvido um planejador que, além de lidar com o não-determinismo e o conceito de tentativas malsucedidas, utiliza técnicas de planejamento HTN, que é uma abordagem utilizada para melhorar o desempenho do processo de planejamento. O planejador implementado foi incorporado à nova versão do sistema de *storytelling* interativo Logtell.

**Palavras-chave:** Storytelling Interativo; Planejamento; Inteligência Artificial; HTN; Não-determinismo

## ABSTRACT

Interactive storytelling is a new kind of digital entertainment in which users interact with a story being told. In this context, automated planning algorithms are interesting alternatives to create plots that are coherent with the intended genre. Diversity of stories and opportunities for interaction are however other key requirements to be considered by such applications. It is also important to generate stories incorporating failed attempts to achieve goals, since this is essential to create dramatic tension. In this work, a model that takes these issues into account by means of planning with nondeterministic events, that is, events with more than one possible different outcome, is proposed. To make such a model feasible, we developed a planner that, apart from dealing with the nondeterminism and the concept of failed attempts, uses HTN planning techniques, which is an approach used to improve the performance of the planning process. The implemented planner has been incorporated to the new version of the interactive storytelling system Logtell.



**Keywords:** Interactive Storytelling; Planning; Artificial Intelligence; HTN; Nondeterminism

**Sumário**

## Lista de Figuras

## Lista de Tabelas

# 1 Introdução

## 1.1 Motivação

Com o desenvolvimento da tecnologia permitindo processamento mais rápido, maior definição gráfica e maior realismo, parece provável que o entretenimento digital venha a se tornar cada vez mais uma das opções preferenciais para o entretenimento (LOUCHART *et al*., 2006). Nesse contexto, podemos incluir os sistemas de *storytelling* interativo, que são, resumidamente, aplicações para contar histórias, que, com base no uso de mecanismos automatizados, procuram adaptar essas histórias de acordo com as interações que têm com os usuários. Tais sistemas podem ser aplicados a jogos (YOUNG, 2001), auxílio à autoria de histórias, entretenimento em geral e até a aplicações comerciais (SPIERLING *et al*., 2002). Uma das dificuldades mais comuns ao se criar um sistema de *storytelling* interativo é fornecer um bom nível de interatividade e cativar o usuário, enquanto se mantém a coerência da história que é gerada.

Um recurso valioso para o desenvolvimento de sistemas de *storytelling* interativo são os algoritmos de planejamento, pois eles podem ser usados para criar um encadeamento lógico de eventos de maneira automática. Dessa forma, proporciona-se a aplicações de *storytelling* interativo uma ferramenta que lhes permite maior grau de autonomia e dinamismo, o que acaba se refletindo em uma maior liberdade do usuário em sua interação com o sistema.

Dentre os sistemas de *storytelling* interativo usando planejamento, destaca-se o Logtell (CIARLINI *et al*., 2005) que, recentemente, incorporou a capacidade de introduzir não-determinismo na dramatização de eventos (DÓRIA *et al*., 2008). Nesse esquema, diferentes enredos são gerados por combinações de diferentes eventos determinísticos e um mesmo enredo pode ter diferentes dramatizações, pois

cada evento pode ser dramatizado de diversas formas diferentes, podendo durar mais ou menos de acordo com a conveniência do processo de *storytelling* e dando possibilidade de alguma interferência do usuário no nível de dramatização. Entretanto, a dramatização produz sempre os efeitos esperados pelo enredo. O *replay value* dos sistemas de *storytelling* interativo está fortemente associado à diversidade das histórias que eles podem gerar, o que torna desejável a geração de enredos com alternativas que possam eventualmente resultar em mais de uma história diferente. Dessa forma, é interessante que, além da possibilidade de combinação de diferentes eventos, cada evento possa ter finais diferentes, influenciados pelos usuários.

O não-determinismo em *storytelling* interativo pode ser melhorado através de um gerador de planos que considere eventos realmente não-determinísticos, ou seja, com mais de um resultado possível. Tal recurso permite a geração de enredos com alternativas de efeitos para determinados eventos, possibilitando que o processo de dramatização apresente diferentes sequências de eventos – que consideram os possíveis diferentes efeitos de cada evento – para representar aquilo que seria, em linhas gerais, um conjunto de versões para uma mesma história.

O uso de técnicas de planejamento em *storytelling* interativo permite que sequências de eventos sejam automaticamente geradas, proporcionando diversidade e coerência para as histórias e, em consequência, reforçando a experiência do usuário. Apesar de ser possível a geração de planos determinísticos variados, previamente ou em tempo real, para diferentes situações, a flexibilidade de obter histórias diversas tende a ser mais restrita se os eventos causam apenas resultados determinísticos. Além disso, para aumentar ainda mais a flexibilidade e introduzir tensão dramática na história, é importante incorporar eventos em que os personagens tentam alcançar objetivos sem sucesso.

A motivação deste trabalho é a proposição de um modelo para a geração de enredos para *storytelling* interativo que contemplem eventos não-determinísticos, incorporando, ainda, o conceito de tentativas de execução de eventos (e de eventuais insucessos nessas tentativas). O modelo precisa garantir, contudo, que a complexidade introduzida pela adoção de tais recursos não resulte em degradação na experiência do usuário.

## 1.2 Proposta

No decorrer desta dissertação, propõe-se um novo modelo para o uso do planejamento para gerar enredos com eventos não-determinísticos, de modo que o mesmo enredo possa gerar diferentes histórias dependendo dos efeitos atribuídos a esses eventos no processo de dramatização. Neste caso, a especificação do gênero literário passa a contemplar eventos com mais de um resultado possível. O modelo também lida explicitamente com metas mais fracas nas quais os personagens tentam atingir uma determinada situação, mas podem ter sucesso ou falhar. O resultado foi parcialmente incorporado ao sistema de *storytelling* interativo Logtell.

O principal problema de tal abordagem é que a complexidade tende a ser maior devido ao fato de que todas as possíveis contingências resultantes de um evento devem ser consideradas. Para garantir que o não-determinismo e o tratamento de múltiplas ramificações não afetem o desempenho global do sistema, técnicas específicas foram introduzidas para acelerar a geração de plano. A técnica principal foi a introdução de uma rede hierárquica de tarefas (HTN) (EROL *et al*., 2004) durante a fase de planejamento, que utiliza conhecimento sobre o domínio das histórias para lidar com a criação de planos não-determinísticos.

Para a validação das principais ideias do novo modelo, foi implementado um protótipo lidando com eventos não-determinísticos e incluindo o conceito de tentativas. O protótipo foi aplicado a uma adaptação do contexto de exemplo adotado nas versões anteriores do Logtell, na qual eventos não-determinísticos foram introduzidos e possibilidades de combinação dos eventos em redes de tarefas também foram especificadas.

## 1.3 Estrutura

No capítulo 2, é apresentada a fundamentação teórica desta pesquisa. O foco principal desse capítulo é o planejamento automático, sendo apresentadas as diferentes abordagens adotadas nessa área de estudo, com destaque para os conceitos relacionados a planejamento não-determinístico e planejamento HTN, especialmente importantes na implementação do modelo apresentado nesta

dissertação. Em seguida, apresenta-se uma visão geral da área de *storytelling* interativo, descrevendo-se as principais questões envolvidas e os enfoques adotados em diferentes sistemas, com especial atenção ao uso de trabalhos nessa área que façam uso de planejamento. É apresentado então, de forma mais detalhada, o sistema Logtell, de forma a auxiliar o entendimento de como e onde o modelo proposto atua.

No capítulo 3, é apresentada uma proposta de modelo para a geração de enredos não-determinísticos para *storytelling* interativo, a ser incorporado ao sistema Logtell. São também apresentados os requisitos a serem atendidos pelo modelo, e a forma como esses requisitos podem ser atendidos por ele.

O capítulo 4 descreve os principais aspectos relativos à implementação do modelo proposto no capítulo 3. Já no capítulo 5, é realizada uma análise dos testes realizados para verificar o atendimento aos requisitos desse modelo.

O capítulo 6 contém as conclusões da pesquisa com a descrição de suas principais contribuições e dos trabalhos futuros potenciais, os quais foram identificados ao longo da elaboração do modelo e dos testes realizados com o protótipo.

# 2 Fundamentação Teórica

O propósito deste capítulo é apresentar uma visão geral dos conceitos associados à pesquisa de que trata esta dissertação, permitindo uma melhor compreensão do trabalho exposto nos capítulos seguintes e do caminho percorrido até se chegar à sua conclusão.

Esta dissertação trata do uso de planejamento HTN não-determinístico aplicado a *storytelling* interativo. Por conta disso, são apresentados, inicialmente, os conceitos relativos a planejamento que são relevantes para esta pesquisa: o planejamento clássico (que serve como uma introdução ao assunto, na seção 2.1), planejamento no espaço de planos (seção 2.2), planejamento não-determinístico (seção 2.3), planejamento HTN (seção 2.4) e, por fim, o caso especial de planejamento HTN não-determinístico (seção 2.5). Em seguida, na seção 2.6, há uma explicação a respeito de sistemas de *storytelling* interativo (com ênfase no uso de planejamento em tal tipo de aplicação), incluindo alguns casos reais de implementação de tais sistemas – com destaque para o sistema Logtell, para o qual o planejador HTN não-determinístico gerado por esta pesquisa foi proposto e implementado tendo em vista a sua incorporação.

## 2.1 Planejamento Clássico

*Planejamento* é o processo de escolha e organização de ações através da antecipação (previsão) de seus efeitos. Esse processo de raciocínio tem o objetivo de satisfazer (através da execução de ações) algumas metas previamente estabelecidas. *Planejamento automático* é a subárea da Inteligência Artificial que estuda esse processo de raciocínio, usando o computador (GHALLAB *et al*., 2004). Apesar

dessa diferenciação conceitual, a literatura especializada costuma adotar o termo mais simples "planejamento" com o mesmo sentido específico de "planejamento automático".

O planejamento é, devido à grande gama de possíveis aplicações práticas, um campo de fundamental importância na pesquisa em Inteligência Artificial. De fato, técnicas de planejamento têm sido aplicadas em uma grande variedade de atividades, incluindo robótica, controle de máquinas industriais, busca de informações na web, agentes autônomos e controle de espaçonaves em missões espaciais.

A grande maioria das pesquisas envolvendo planejamento diz respeito ao chamado *planejamento clássico*. Essa abordagem exige o atendimento de uma série de suposições: é necessário que o planejador tenha conhecimento completo sobre o ambiente e que este seja determinístico, estático e de estados finitos com satisfação de metas e tempo implícito. Além disso, espera-se que os planos gerados sejam uma sequência finita de ações linearmente ordenadas (GHALLAB *et al*., 2004). A adoção de tais restrições provoca uma simplificação que se mostrou bastante útil no desenvolvimento de algoritmos de planejamento e, de fato, vários deles foram desenvolvidos com base nessas suposições.

Problemas de planejamento clássico são definidos sobre um domínio composto por um conjunto $S$ de estados possíveis, um conjunto $A$ de ações aplicáveis a esses estados e uma função de transição de estados $\gamma(s, a)$ que estabelece quais os estados atingidos pela aplicação dessas ações a esses estados – ou seja, $s' = \gamma(s, a)$ se s' é o estado atingido a partir de $s$ com a ação $a$. Podemos também, em algumas representações desse tipo de problema, adotar uma generalização das ações através de um conjunto de operadores $O$ que possuem precondições e efeitos representados por literais; nesse caso, as ações são instâncias *ground* (isto é, sem nenhum símbolo de variável) desses operadores. O objetivo desse tipo de problema é encontrar uma solução que, aplicada a partir de um estado inicial predefinido, leve a um determinado estado meta.

Uma das principais restrições do planejamento clássico diz respeito à exigência de que todas as ações sejam determinísticas, ou seja, dado um determinado estado $s$, a aplicação de uma ação $a$ produzirá como resultado um estado $s'$ único e conhecido. Nesse caso, o plano que serve como solução para um problema de planejamento será

uma sequência de ações $a_0$, $a_1$, ..., $a_n$ que gera uma trajetória de estados $s_0$, $s_1 = \gamma(s_0, a_0)$, ..., $s_{n+1} = \gamma(s_n, a_n)$, tal que cada ação $a_i$ é aplicável em $s_i$ e $s_{n+1}$ é o estado meta.

Infelizmente, poucas situações na prática oferecem condições condizentes com todas as limitações impostas pelo planejamento clássico. Para permitir o tratamento da maior parte dos problemas práticos é necessário relaxar uma ou mais das suposições do planejamento clássico. A seguir, veremos alguns casos em que o relaxamento dessas suposições permite a obtenção de planos aplicáveis a contextos mais gerais, onde não podem ser assumidas as limitações do planejamento clássico. Adicionalmente, veremos como informações adicionais sobre o domínio, não previstas no planejamento clássico, podem ser usadas para obter maior eficiência.

## 2.2 Planejamento no Espaço de Planos

O planejamento clássico exige que os planos sejam uma sequência finita de ações totalmente ordenadas. Entretanto, é possível considerar planos que se apresentem sob a forma de estruturas bem mais ricas, como, por exemplo, conjuntos parcialmente ordenados de ações (GHALLAB *et al*., 2004).

É o que ocorre, por exemplo, no planejamento no espaço de planos. No planejamento clássico, o espaço de busca consiste em um grafo cujos vértices são estados do domínio e cujas arestas são transições entre estados ou ações; um plano, nesse caso, estabelece uma sequência totalmente ordenada de ações, correspondendo a um caminho do estado inicial até o estado final. Já no planejamento no espaço de planos, os vértices são planos parcialmente especificados (definidos sob a forma de um conjunto de ações parcialmente instanciadas e um conjunto de restrições), e as arestas correspondem a operações de refinamento sobre esses planos. Essas operações de refinamento têm como objetivo atingir metas em aberto (através da adição de ações) ou da remoção de possíveis inconsistências (através da adição de restrições). Essas restrições podem indicar (NAU, 2007):

- Precedências de ações (ex.: ação *a* deve preceder ação *b*)
- Amarrações de variáveis, que podem ser:
    - Restrições de desigualdade (ex.: $v \neq c$)
    - Restrições de igualdade (ex.: $v = c$)

    - Restrições de limitação a valores do domínio (ex.: $v > 10$)
- Vínculos causais (ex.: use ação *a* para estabelecer a precondição *p* necessária para a ação *b*)

As operações de refinamento evitam a adição de restrições que não sejam estritamente necessárias para o propósito do refinamento. Tal estratégia é conhecida como o *princípio do compromisso mínimo*. O processo de planejamento consiste então em, partindo de um plano vazio, fazer refinamentos até chegar a um plano solução, onde todas as metas são satisfeitas.

O planejamento no espaço de planos difere do planejamento clássico não apenas no espaço de busca, mas na própria definição de plano – que, nesse caso, é definido como um conjunto de operadores com restrições de ordem e de amarração. Essa estrutura, por ser mais geral, pode não corresponder a uma sequência de ações, como ocorre no planejamento clássico.

## 2.3 Planejamento Não-Determinístico

Entre os relaxamentos que podem ser feitos com as suposições do planejamento clássico, a aceitação do não-determinismo, com eventos que possuam mais de um resultado possível, é um dos que mais nos aproximam de uma abordagem realística do mundo. A principal dificuldade, nesse caso, é que um plano pode resultar em inúmeras trajetórias de execução, o que exige formas eficientes de se analisar todos os resultados possíveis e a geração de planos que tenham comportamento condicional e estratégias de tentativa e erro (GHALLAB *et al*., 2004). De fato, o tratamento do não-determinismo implica em problemas de eficiência ainda mais sérios do que o planejamento clássico, pois é necessário considerar os vários efeitos possíveis de cada ação.

Quando se trata de planejamento com não-determinismo, duas abordagens se destacam: o planejamento baseado em Processos de Decisões de Markov e o planejamento baseado em verificação de modelos. A primeira delas requer probabilidades associadas aos possíveis efeitos de cada ação e a associação de custos às ações e de recompensas aos estados atingidos. Através do uso desses

valores, o problema de planejamento transforma-se em um problema de otimização, onde se tenta maximizar as recompensas, reduzindo os custos.

Já o planejamento baseado em verificação de modelos não trabalha com probabilidade. Uma ação pode ter mais de um efeito possível e não se sabe qual deles será efetivamente estabelecido quando a ação for executada. As metas, assim como eventuais condições que devam valer durante a própria trajetória de execução do plano, são expressas através de fórmulas de lógica temporal. Essas fórmulas podem assumir diferentes graus de exigência com relação à satisfação dessas metas e condições. Uma forma de lógica temporal que permite expressar esses graus de exigência é a *Computation Tree Logic* (CTL) (EMERSON, 1990), que possui operadores para indicar necessidades (ou seja, a fórmula deve ser verdadeira em todos os caminhos a partir do estado atual), possibilidades (a fórmula deve ser verdadeira em algum caminho a partir do estado atual) e condições temporais (por exemplo, "em algum momento no futuro", "em todos os estados futuros").

Os planos gerados por essas estratégias de planejamento não-determinístico são chamados de *políticas*. Diferentemente da sequência de ações que compõem um plano no planejamento clássico, uma política faz uma associação entre estados e ações, estabelecendo qual ação deve ser executada quando um determinado estado do mundo é verificado. Tal abordagem é de fundamental importância em domínios não-determinísticos, pois o estabelecimento de uma única sequência de ações que garantidamente atinja os objetivos pode ser simplesmente inviável. Além disso, políticas conseguem estabelecer comportamentos condicionais e iterativos adequados ao não-determinismo do domínio.

Uma das principais vantagens do planejamento baseado em verificação de modelos é a possibilidade de se usar fórmulas proposicionais para se representar grandes conjuntos de estados, o que permite uma representação compacta que facilita o planejamento para problemas de larga escala (CIMATTI *et al*., 2003). Nesse caso, o processo de busca é realizado através de transformações lógicas sobre essas fórmulas. BDDs (*Binary Decision Diagrams*) (BRYANT, 1992) são a forma mais comum de implementação dessa técnica (KUTER, 2005). Experimentos diversos têm demonstrado que algoritmos de planejamento baseados em verificação de modelos e BDDs conseguem lidar bem com problemas de tamanho considerável.

Um exemplo desse tipo de implementação é o MBP (*Model Based Planner*), um sistema para planejamento em domínios não-determinísticos. O sistema é composto de um *framework* para tratar de diferentes tipos de problemas em domínios não-determinísticos, algoritmos de planejamento para lidar com grandes espaços de estados e funcionalidades para validação e simulação de planos (CIMATTI *et al*., 2003).

Apesar da expressividade de CTL com relação à especificação de metas (uma das causas para que viesse a se tornar o principal formalismo usado para planejamento qualitativo baseado em verificação de modelos (PISTORE & TRAVERSO, 2001), às vezes essa linguagem mostra-se insuficiente para expressar certas características da solução desejada. É o caso das metas de alcançabilidade estendidas que, além de especificar uma condição a ser alcançada ao final da execução de uma política, também estabelecem uma condição a ser preservada (ou evitada) em todos os estados visitados durante a execução dessa política. Uma proposta para resolver esse problema é apresentada por PEREIRA (2007), sob a forma de uma nova versão da lógica temporal CTL, denominada α-CTL. Além disso, são apresentadas a lógica e a implementação de um verificador de modelos (VACTL) e de um planejador (PACTL) para tratar tais problemas com base na própria α-CTL.

## 2.4 Planejamento Baseado em Rede Hierárquica de Tarefas

Outra técnica utilizada com o intuito de facilitar a resolução de problemas práticos é o uso de conhecimento específico sobre o domínio do problema a ser resolvido. Nesse sentido, merece destaque o planejamento baseado em rede hierárquica de tarefas – ou, mais simplesmente, planejamento HTN (*Hierarchical Task Network*). Ele é, em alguns aspectos, semelhante ao planejamento clássico, pois em ambos os casos, uma sequência de ações é aplicada a partir de um estado inicial, gerando estados intermediários e conhecidos entre uma ação e outra. A maior diferença conceitual entre as duas abordagens reside naquilo para o que se planeja: enquanto no planejamento clássico as ações são escolhidas em função de

*metas* a serem atingidas, no planejamento HTN a sequência de ações é estabelecida em função de *tarefas* que devem ser executadas.

As diferentes formas como essas tarefas podem ser executadas são descritas através de *métodos*, sendo que cada um deles estabelece uma decomposição da tarefa a ele associada em um conjunto de tarefas menores, ou *subtarefas*. Através da aplicação recursiva de métodos a *tarefas não-primitivas* (que se decompõem em outras tarefas), acaba-se chegando a *tarefas primitivas* (relacionadas diretamente a operadores do domínio), que são então instanciadas em ações a serem incorporadas ao plano.

A fim de descrever essas tarefas e métodos, utiliza-se o conhecimento a respeito do domínio sobre o qual se está trabalhando. Pode-se dizer que a ideia principal do planejamento HTN, em termos conceituais, é a utilização do conhecimento sobre o domínio em questão para otimizar o processo de planejamento para um determinado problema. Isso permite a obtenção de planos de forma bem mais eficiente do que no planejamento clássico. Esse é um dos motivos pelo qual HTN é uma das técnicas de planejamento mais populares, se não a mais popular, em aplicações práticas (GHALLAB *et al*., 2004).

Um exemplo de implementação de planejamento HTN é SHOP2, um algoritmo que permite o estabelecimento de ordenação parcial entre as tarefas de um método. SHOP2 constrói planos totalmente ordenados, acrescentando cada passo do plano na mesma ordem em que eles serão executados e sendo, portanto, capaz de saber o estado em que o mundo se encontra a cada etapa do processo de planejamento (NAU *et al*., 2001). Esse conhecimento é de grande importância para a obtenção de soluções eficientemente, pois permite limitar o espaço de busca consideravelmente.

### 2.4.1 Planejamento STN

Planejamento STN (de *Simple Task Network*, ou rede simples de tarefas) é um caso específico, mais simples, de planejamento HTN. Apesar de o planejamento STN apresentar uma perda de riqueza em relação ao HTN mais genérico, a maior parte dos conceitos associados ao primeiro permanece válida para o último. Esse fato, somado à sua maior facilidade de compreensão, torna extremamente válida sua apresentação antes de se chegar ao planejamento HTN propriamente dito.

Para o planejamento STN, mantemos as mesmas definições para operadores, ações, planos e para a função de transição de estados $\gamma(s, a)$ (o resultado da aplicação de uma ação *a* a um estado *s*) que temos no planejamento clássico. Os conceitos de STN que não pertencem ao planejamento clássico, e que são utilizados tanto na definição dos problemas quanto em suas soluções, dizem respeito a *tarefas*, *métodos* e *redes de tarefas*. Tais conceitos estão intimamente relacionados à forma como um ser humano, dotado de conhecimento específico a respeito do domínio sobre o qual se efetuará o planejamento, imagina que (sub)problemas do domínio podem ser resolvidos.

#### 2.4.1.1 Tarefas e Redes de Tarefas

Problemas de planejamento HTN (incluindo o caso específico STN) são resolvidos através da execução de *tarefas*. De uma maneira informal, pode-se dizer que uma tarefa é aquilo que deve ser feito para a resolução de um determinado problema de planejamento HTN. Tarefas, em geral, podem ser decompostas em tarefas menores, ou *subtarefas*, e, nesse caso, são ditas *tarefas não-primitivas*. Tal processo de decomposição pode seguir repetindo-se recursivamente até que cheguemos a tarefas diretamente relacionadas a operadores do domínio, as quais podem ser executadas diretamente, sem a necessidade de novas decomposições. Tais tarefas são denominadas *tarefas primitivas*.

De uma maneira mais formal, podemos definir uma tarefa *t* como uma expressão da forma $t(r_1, ..., r_k)$, onde *t* é um *símbolo de tarefa* e $r_1, ..., r_k$ são termos. A tarefa é primitiva se *t* for um símbolo de operador (ou seja, diretamente instanciável em uma ação do domínio); caso contrário, é não-primitiva. Além disso, dizemos que uma tarefa é *ground* se todos os seus termos são *ground* (isto é, não há nenhum símbolo de variável na expressão que representa a tarefa); senão, é considerada *unground*. Por fim, dizemos que uma ação *a cumpre* uma tarefa *t* em um estado *s* se seu nome for igual a *t* e *a* é aplicável a *s*.

Tarefas grandes, de nível mais alto, são decompostas em tarefas menores, de nível mais baixo, através de definições de *redes de tarefas*. Uma rede de tarefas é um dígrafo acíclico da forma $w = (U, E)$, onde *U* é o conjunto de vértices e *E* é o conjunto de arestas. Cada vértice *u* de *U* contém uma tarefa $t_u$, e cada aresta de *w*

representa uma ordenação parcial entre um par de vértices. Uma rede de tarefas *w* é considerada *ground* ou *primitiva* se todas as tarefas que a compõem são *ground* ou primitivas, respectivamente; de forma análoga, *w* é considerada *unground* ou *não-primitiva* conforme as condições anteriores não sejam respectivamente atendidas.

Se a ordenação definida pelas arestas de *w* for total, ou seja, existe um e apenas um sucessor possível para cada um de seus vértices (com exceção do último, para o qual não há nenhum), então se diz que *w* é *totalmente ordenada*; do contrário, dizemos que é *parcialmente ordenada*.

#### 2.4.1.2 Métodos

Tarefas nos dizem o que podemos fazer, mas são os métodos que, efetivamente, definem quais tarefas são realizadas e em que ordem durante a elaboração de um plano. Uma tarefa pode ter mais de uma forma possível de ser cumprida e, nesse caso, as definições contidas nos métodos serão usadas pelo planejador no momento em que for necessário escolher algum deles.

Formalmente, um *método* consiste em uma quádrupla do tipo

$$m = (nome(m), tarefa(m), precond(m), rede(m))$$

na qual:

- *nome*(*m*) é o *nome* do método, uma expressão sintática do tipo $n(x_1, ..., x_k)$ na qual *n* é um símbolo único para o método e os elementos $x_1, ..., x_k$ correspondem a todos os símbolos de variáveis que ocorrem em *m*;
- *tarefa*(*m*) é uma tarefa não-primitiva, à qual o *m* está associado;
- *precond*(*m*) é um conjunto de literais que são precondições para a execução de *m;*
- *rede*(*m*) é uma rede de tarefas cujas tarefas constituintes são chamadas *subtarefas* de *m*; *tarefa*(*m*) estará cumprida quando todas as suas subtarefas também estiverem.

Um método *m* é *totalmente ordenado* quando *rede*(*m*) também o for. Nesse caso, pode-se substituir a especificação em forma de dígrafo para *rede*(*m*) por uma mais simples, *subtarefas(m)*, que apresenta as subtarefas de *m* como uma lista ordenada de tarefas da forma

$$subtarefas(m) = \langle t_1, ..., t_k \rangle,$$

onde cada tarefa $t_i$ corresponde a um vértice $u_i$ de *rede*(*m*).

#### 2.4.1.3 Aplicabilidade, Relevância e Decomposição

Uma tarefa pode estar associada a mais de um método; entretanto, existem critérios para definir qual deve ser escolhido para efetuar a decomposição dessa tarefa. Esses critérios dizem respeito à *aplicabilidade* e à *relevância* de uma instância do método. Dizemos que uma instância *m* de um método é *aplicável* a um estado *s* se *s* atende às precondições de *m*, ou seja:

$$\text{precond}^{+}(m) \subseteq s \text{ e } \text{precond}^{-}(m) \cap s = \emptyset,$$

sendo $\text{precond}^{+}(m)$ e $\text{precond}^{-}(m)$, respectivamente, as precondições positivas e negativas de *m*.

Uma instância de método *m* é dita *relevante* para uma tarefa *t* se existe alguma substituição $\sigma$ tal que $\sigma(t) = \textit{tarefa}(m)$. Nesse caso, a *decomposição* de *t* por *m* sob $\sigma$, indicada por $\delta(t, m, \sigma)$ é igual a $\sigma(\textit{rede}(m))$ – ou $\sigma(\textit{tarefas}(m))$, caso *m* seja totalmente ordenado.

Sempre que um método decompõe uma tarefa, uma nova rede de tarefas é gerada. Indicamos por $\delta(w, u, m, \sigma)$ o conjunto de redes de tarefas resultante da decomposição de um vértice de tarefa *u* de uma rede de tarefas *w* por um método *m* sob uma substituição $\sigma$. Embora a definição formal desse conceito seja um tanto complicada, a ideia por ele expressa pode ser compreendida intuitivamente: substitui-se *u* em *w* por uma cópia de *subtarefas*(*m*), e todas as restrições de ordenação aplicadas a *u* passam então a valer para cada vértice da cópia de *subtarefas*(*m*).

Uma das principais dificuldades do processo de decomposição reside no fato de que pode haver mais de um candidato possível a ser a primeira tarefa do conjunto de subtarefas (de acordo com as precondições a serem atendidas), criando a necessidade de estabelecer um conjunto de redes alternativas de tarefas – uma para cada um dos possíveis candidatos.

Em um processo de planejamento STN, estaremos sempre interessados em métodos que sejam aplicáveis ao estado corrente e relevantes para uma tarefa que queiramos cumprir.

#### 2.4.1.4 Domínio, Problemas e Soluções

Para que possamos usar um planejador automático na busca de uma solução para um problema de planejamento STN, precisamos de algumas formalizações para definir como o problema será representado e o que, afinal, poderá ser considerada uma solução para ele.

Antes de definir nosso problema, precisamos descrever o mundo ao qual ele se aplica – ou, ao menos, o que nos interessa desse mundo com relação ao nosso problema. Essa descrição corresponde a um *domínio de planejamento STN*, e se apresenta sob a forma

$$D = (O, M),$$

onde $O$ é o conjunto de operadores e $M$ é o conjunto de métodos associados ao domínio em questão. Se todos os métodos de $M$ forem totalmente ordenados, $D$ é um *domínio de planejamento de ordem total*.

Podemos ter vários problemas diferentes sobre um mesmo domínio. O que vai diferenciar um do outro são dois fatores: seu estado inicial e a tarefa (ou conjunto de tarefas) que se deseja cumprir. Dessa forma, um *problema de planejamento STN* apresenta-se como uma quádrupla da forma:

$$P = (s_0, w, O, M),$$

sendo $s_0$ o estado inicial, $w$ uma rede de tarefas a que chamamos *rede inicial de tarefas*, e $O$ e $M$ são respectivamente o conjunto de operadores e o conjunto de métodos do domínio $D$ do problema. Mais uma vez, podemos estender o conceito de ordenação total para um nível de abstração mais alto: se tanto $w$ quanto $D$ forem totalmente ordenados, então se diz que $P$ é um *problema de planejamento de ordem total* (caso contrário, trata-se de um *problema de planejamento de ordem parcial*).

Tendo definido o domínio do problema e, com a ajuda desse conceito, o problema em si, falta ainda definir o que faz de um plano $\pi = \{a_1, ..., a_n\}$ uma solução para esse problema $P = (s_0, w, O, M)$ – o que nos permitiria também dizer que o plano $\pi$ *cumpre* $w$. Intuitivamente, podemos dizer que a sequência de ações expressa por $\pi$ é o resultado da decomposição sucessiva de $w$ até chegar-se a tarefas primitivas, que são então instanciadas em ações que seguem uma ordem condizente com a ordem das tarefas que as geraram.

Formalmente, para um problema de planejamento $P = (s_0, w, O, M)$, teremos que $\pi = \{a_1, ..., a_n\}$ será uma solução para $P$ se algum dos três casos a seguir ocorrer:

- Se $w$ for vazio e o plano $\pi$ também for vazio (ou seja, $n = 0$).
- Se em $w$ houver um vértice $u$ sem predecessores associado a uma tarefa primitiva $t_u$, a ação $a_1$ do plano $\pi$ for aplicável a $t_u$ em $s_0$ e, recursivamente, o plano $\pi = \{a_2, ..., a_n\}$ for uma solução para:

  $$P' = (\gamma(s_0, a_1), w - \{u\}, O, M),$$

  que é o problema de planejamento produzido pela execução da primeira ação de $\pi$ e pela remoção do vértice correspondente $u$ da rede de tarefas original $w$.
- Se em $w$ houver um vértice $u$ sem predecessores associado a uma tarefa não-primitiva $t_u$, existir uma instância $m$ de algum método em $M$ que seja relevante para $t_u$ e aplicável em $s_0$ e houver uma rede de tarefas $w'$ pertencente a $\delta(w, u, m, \sigma)$ tal que, recursivamente, o plano $\pi$ seja uma solução para:

  $$P' = (s_0, w', O, M).$$

Para apresentar os procedimentos para resolução de problemas STN, tomaremos como exemplo de domínio o "Mundo das Docas", ou *DWR* (*Dock-Worker Robots*). Em nosso problema, temos dois locais (*l1* e *l2*), dois guindastes-robôs (*crane1* e *crane2*), uma carreta-robô (*r1*), dois contêineres (*c1* e *c2*) e quatro pilhas (*p11*, *p12*, *p21* e *p22*). O estado inicial é mostrado na Figura 2.1. Nosso objetivo é, com o uso da carreta-robô *r1*, transportar os contêineres *c1* e *c2* para o local *l2*.

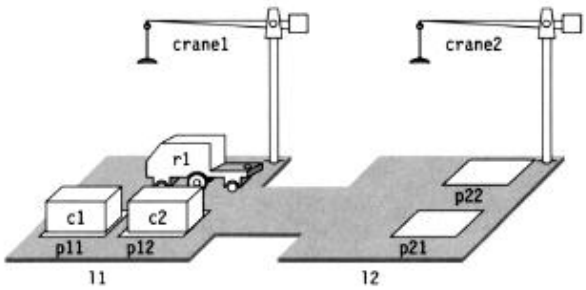


Figura 2.1: Estado inicial do problema DWR. Reproduzida de (GHALLAB *et al*., 2004) – pág. 240.

Veremos como esse problema pode ser resolvido usando um planejamento de ordem total e, em seguida, como podemos aumentar o número de soluções possíveis (possibilitando a obtenção de soluções eventualmente melhores) com o uso de métodos parcialmente ordenados.

### 2.4.2 Planejamento STN de Ordem Total

Uma forma se resolver o problema da Figura 2.1 é através do procedimento *TFD* (*Total-order Forward Decomposition*), utilizado para a resolução de problemas de ordem total. A abordagem utilizada por esse procedimento baseia-se diretamente nos três casos apresentados na definição de uma solução para um problema de planejamento STN. TFD é tanto correto quanto completo.

Consideremos, como exemplo, um conjunto de métodos que, por razões de clareza, serão apresentados em um formato um pouco diferente da quádrupla descrita anteriormente; chamaremos a esse conjunto de métodos de *M1*:

transfer2($c_1$, $c_2$, $l_1$, $l_2$, $r$) *; método para transferir $c_1$ e $c_2$*
    tarefa: transfer-two-containers($c_1$, $c_2$, $l_1$, $l_2$, $r$)
    precond: *; nenhuma*
    subtarefas: ⟨transfer-one-container($c_1$, $l_1$, $l_2$, $r$),
    transfer-one-container($c_2$, $l_1$, $l_2$, $r$)⟩
transferl($c$, $l_1$, $l_2$, $r$) ; *método para transferir c*
    tarefa: transfer-one-container($c$, $l_1$, $l_2$, $r$)
    precond: *; nenhuma*
    subtarefas: ⟨setup($c$, $r$), move-robot($r$, $l_1$, $l_2$), finish($c$, $r$)⟩
movel($r$, $l_1$, $l_2$) *; método para mover r se não estiver em $l_2$*
    tarefa: move-robot($r$, $l_1$, $l_2$)
    precond: at($r$, $l_1$)
    subtarefas: ⟨move($r$, $l_1$, $l_2$)⟩
move0($r$, $l_1$, $l_2$) *; método para não fazer nada se r já estiver em $l_2$*
    tarefa: move-robot($r$, $l_1$, $l_2$)
    precond: at($r$, $l_2$)
    subtarefas: () *; i.e., sem subtarefas*

do-setup($c$, $d$, $k$, $l_1$, $l_2$, $p$, $r$) *; método para preparar para mover um contêiner*

tarefa: setup($c$, $r$)

precond: on($c$, $d$), in($c$, $p$), belong($k$, $l_2$), attached($p$, $l_2$), at($r$, $l_1$)

subtarefas: ⟨move-robot($r$, $l_1$, $l_2$), take($k$, $l_2$, $c$, $d$, $p$), load($k$, $l_2$, $c$, $r$)⟩

unload-robot($c$, $d$, $k$, $l$, $p$, $r$) *; método para finalizar após mover um contêiner*

tarefa: finish($c$, $r$)

precond: attached($p$, $l$), loaded($r$, c), top($d$, $p$), belong($k$, $l$), at ($r$, $t$)

subtarefas: ⟨unload($k$, $l$, $c$, $r$), put($k$, $l$, $c$, $d$, $p$)⟩

Uma descrição formal desse problema de planejamento deveria, ainda, apresentar:

- uma descrição do estado inicial mostrado na Figura 2.1;
- uma tarefa inicial, diretamente associada à descrição do problema – no caso, a tarefa *transfer-two-containers(c1, c2, l1, l2, r1)*;
- o conjunto de métodos *M1*;
- um conjunto de operadores que incluiria os operadores *take*, *load*, *move*, *unload* e *put*.

A resolução desse problema acabaria por produzir a árvore de decomposição da Figura 2.2. A tarefa inicial *transfer-two-containers(c1, c2, l1, l2, r1)* é decomposta recursivamente em subtarefas, sendo que cada uma é instanciada herdando as substituições da tarefa que a originou; os métodos selecionados para a geração dessas subtarefas devem ser relevantes para a tarefa avaliada, e ter suas precondições atendidas no estado corrente. O processo de decomposição prossegue até chegar-se às tarefas primitivas, as quais podem ser instanciadas diretamente por operadores do domínio DWR (essas tarefas correspondem aos retângulos sombreados da Figura 2.2).

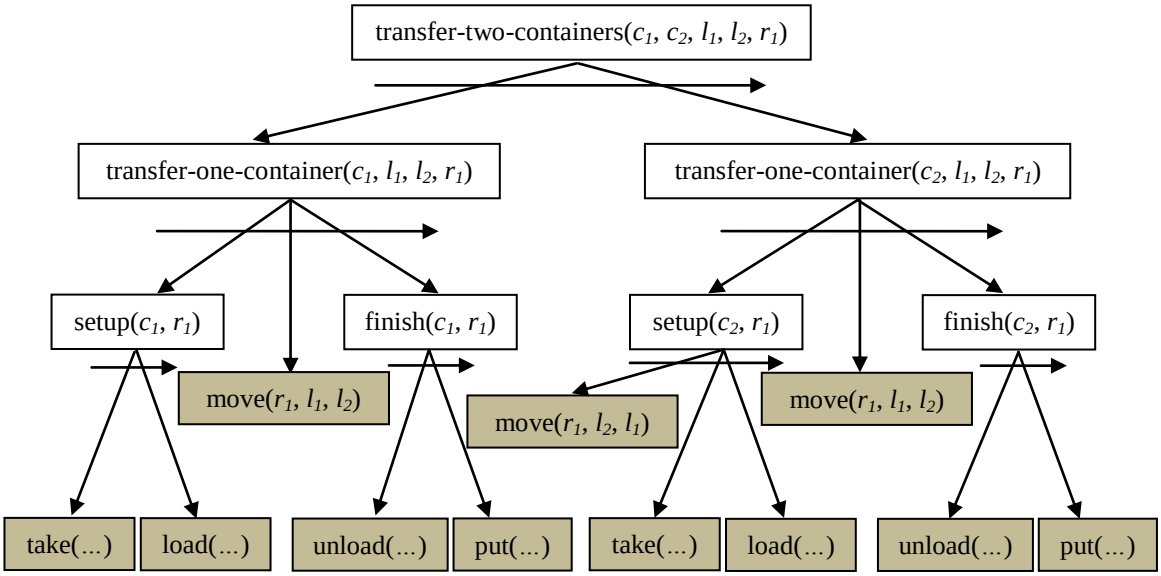


Figura 2.2: Árvore de decomposição para o problema da Figura 2.1, utilizando o conjunto M1 de métodos totalmente ordenados. Exemplo retirado de (GHALLAB *et al*., 2004).

### 2.4.3 Planejamento STN de Ordem Parcial

Suponhamos que, no problema DWR apresentado, a carreta-robô *r1* possa carregar mais de um contêiner ao mesmo tempo. Poderíamos ter uma solução melhor, como a apresentada na Figura 2.3, onde duas tarefas se intercalam de modo a fazer com que *r1* transporte os dois contêineres ao mesmo tempo.

No planejamento de ordem total, tal intercalação não é possível, pois todos os métodos são totalmente ordenados e, assim, uma subtarefa de uma tarefa não pode ser executada antes que a tarefa predecessora tenha sido totalmente cumprida. Uma opção seria alterar nosso conjunto de métodos *M1* de forma a ter um método que carregasse todos os contêineres, só então efetuasse o transporte dos mesmos e, por fim, os descarregasse ao mesmo tempo.

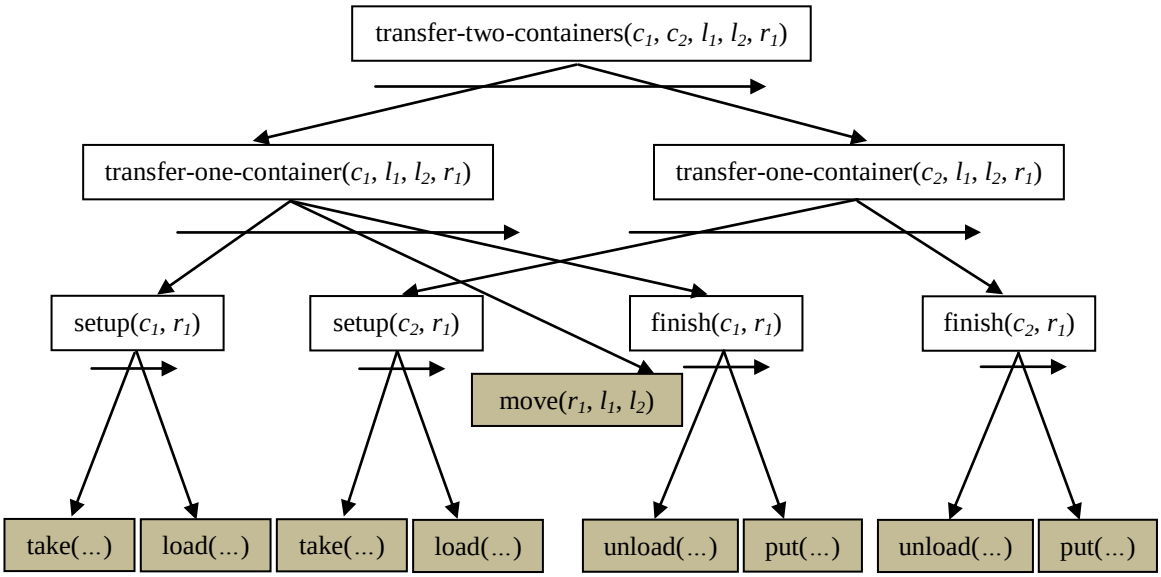


Figura 2.3: Árvore de decomposição para o problema da Figura 2.1, com a possibilidade de *r1* carregar mais de um contêiner ao mesmo tempo; o uso de ordenamento parcial permite a intercalação de tarefas. Exemplo retirado de (GHALLAB *et al*., 2004).

Outra opção mais interessante, por ser mais flexível, seria o uso de planejamento de ordem parcial, com métodos parcialmente ordenados. Isso permitiria que uma subtarefa de uma tarefa $t_1$ pudesse ser executada antes que outra tarefa $t_2$, com a qual $t_1$ não tivesse nenhuma restrição de ordenação, fosse totalmente cumprida.

Um conjunto de métodos *M2*, que permite tal comportamento, é apresentado abaixo:

transfer2($c_1$, $c_2$, $l_1$, $l_2$, $r$) *; método para transferir $c_1$ e $c_2$*

tarefa: transfer-two-containers($c_1$, $c_2$, $l_1$, $l_2$, $r$)

precond: *; nenhuma*

subtarefas: $u_1$ = transfer-one-container($c_1$, $l_1$, $l_2$, $r$),
$u_2$ = transfer-one-container($c_2$, $l_1$, $l_2$, $r$),

transferl($c$, $l_1$, $l_2$, $r$) *; método para transferir c*

tarefa: transfer-one-container($c$, $l_1$, $l_2$, $r$)

precond: *; nenhuma*

rede: $u_1$ = setup($c$, $r$), $u_2$ = move-robot($r$, $l_1$, $l_2$),
$u_3$ = finish($c$, $r$), {($u_1$, $u_2$), ($u_2$, $u_3$)}

move l($r$, $l_1$, $l_2$) *; método para mover r se não estiver em* $l_2$

tarefa: move-robot($r$, $l_1$, $l_2$)

precond: at($r$, $l_1$)

subtarefas: ⟨move($r$, $l_1$, $l_2$)⟩

move0($r$, $l_1$, $l_2$) *; método para não fazer nada se r já estiver em* $l_2$

tarefa: move-robot($r$, $l_1$, $l_2$)

precond: at($r$, $l_2$)

subtarefas: ⟨⟩ *; i.e., sem subtarefas*

do-setup($c$, $d$, $k$, $l_1$, $l_2$, $p$, $r$) *; método para preparar para mover um contêiner*

tarefa: setup(c, r)

precond: on($c$, $d$), in($c$, $p$), belong($k$, $l_2$), attached($p$, $l_2$), at($r$, $l_1$)

rede: $u_1$ = move-robot($r$, $l_1$, $l_2$), $u_2$ = take($k$, $l_2$, $c$, $d$, $p$),

$u_3$ = load($k$, $l_2$, $c$, $r$), {($u_1$, $u_2$), ($u_2$, $u_3$)}

unload-robot($c$, $d$, $k$, $l$, $p$, $r$) *; método para finalizar após mover um contêiner*

tarefa: finish($c$, $r$)

precond: attached($p$, $l$), loaded($r$, $c$), top($d$, $p$), belong($k$, $l$), at ($r$, $t$)

rede: $u_1$ = unload($k$, $l$, $c$, $r$), $u_2$ = put($k$, $l$, $c$, $d$, $p$) , {($u_1$, $u_2$)}

O procedimento *PFD* (*Partial-order Forward Decomposition*), apresentado na Figura 2.4, permite a geração de planos que se aproveitem da maior flexibilidade oferecida pelo planejamento de ordem parcial em relação ao planejamento de ordem total. Assim como TDF, PFD implementa diretamente a definição para solução de um problema de planejamento STN, testando cada uma das três situações possíveis; além disso, também é correto e completo.

PFD($s$, $w$, $O$, $M$)

se $w$ = Ø então retorne *plano vazio*

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

se $t_u$ é uma tarefa primitiva então

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

$\sigma$ é uma substituição tal que *nome*($a$) = $\sigma(t_u)$,

e *a* é aplicável a *s*}

se *válidos* = Ø então retorne *fracasso*

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

*π* ← PFD (*γ*(*s*, *a*), *σ*(*w* – {*u*}), *O*, *M*)

se *π* = *fracasso* então retorne *fracasso*

senão retorne a.*π* *(ou seja, π acrescido de a)*

senão

*válidos* ← {(*m*, *σ*) | *m* é instância "ground" de um método em *M*,

*σ* é a substituição tal que *nome*(*m*) = *σ*($t_u$),

e *m* é aplicável a *s*}

se *válidos* = Ø então retorne *fracasso*

escolha não-deterministicamente algum (*m*, *σ*) ∈ *válidos*

escolha não-deterministicamente alguma rede de tarefas *w'* ∈ δ(*w*, *u*, *m*,σ)

retorne(PFD(*s*, *w'*, *O*, *M*))

Figura 2.4: Procedimento PFD para planejamento STN de ordem parcial

### 2.4.4 Planejamento HTN

No planejamento STN, temos dois tipos de restrições para cada método: restrições de ordenação e precondições. Expressamos as restrições de ordenação através das arestas da rede de tarefas – ou, no caso de métodos totalmente ordenados, através da ordem das tarefas dentro da lista de subtarefas. Quanto às precondições, nós não temos nenhum recurso para acompanhá-las; na verdade, o que fazemos é forçá-las a ocorrer no momento adequado através da forma como construímos as redes de tarefas.

Tal abordagem provavelmente só funciona com procedimentos que realizam decomposição para a frente, como é o caso de TFD e PFD. Entretanto, pode haver casos em que se queira uma flexibilidade maior, que permita realizar o processo de decomposição de outra forma (para trás, por exemplo), e outra abordagem faz-se necessária.

O planejamento HTN é uma generalização do planejamento STN que permite maior liberdade para a construção das redes de tarefas através do uso de um mecanismo de acompanhamento que representa as restrições ainda não atendidas. Nesse caso, uma *rede de tarefas* é representada por um par $w = (U, C)$, sendo $U$ um conjunto de vértices associados a tarefas e $C$ um conjunto de restrições que devem ser atendidas por qualquer plano que seja proposto como solução para o problema de planejamento analisado. Essas restrições dizem respeito, basicamente, à ordenação entre os vértices de tarefas e ao estabelecimento de literais antes, após ou entre determinados conjuntos de vértices de tarefas.

Um método HTN consiste em uma quádrupla do tipo:

$$m = (nome(m), tarefa(m), subtarefas(m), restr(m))$$

na qual *nome*(*m*) e *tarefa*(*m*) têm o mesmo sentido visto no planejamento STN, e (*subtarefas*(*m*), *restr*(*m*)) constitui uma rede de tarefas no planejamento HTN.

Eis outro conjunto de métodos para o problema da Figura 2.1; desta vez, utilizando métodos HTN:

transfer2($c_1$, $c_2$, $l_1$, $l_2$, $r$) *; método para transferir* $c_1$ *e* $c_2$

tarefa: transfer-two-containers($c_1$, $c_2$, $l_1$, $l_2$, $r$)

subtarefas: $u_1$ = transfer-one-container($c_1$, $l_1$, $l_2$, $r$),

$u_2$ = transfer-one-container($c_2$, $l_1$, $l_2$, $r$)

restr: $u_1 \prec u_2$

transferl($c$, $l_1$, $l_2$, $r$) *; método para transferir c*

tarefa: transfer-one-container($c$, $l_1$, $l_2$, $r$)

subtarefas: $u_1$ = setup($c$, $r$), $u_2$ = move-robot($r$, $l_1$, $l_2$), $u_3$ = finish($c$, $r$)

restr: $u_1 \prec u_2$, $u_2 \prec u_3$

move l($r$, $l_1$, $l_2$) *; método para mover r se não estiver em* $l_2$

tarefa: move-robot($r$, $l_1$, $l_2$)

subtarefas: $u_1$ = move($r$, $l_1$, $l_2$)

restr: before({$u_1$}, at($r$, $l_1$))

move0($r$, $l_1$, $l_2$) *; método para não fazer nada se r já estiver em* $l_2$

tarefa: $u_0$ = move-robot($r$, $l_1$, $l_2$)

subtarefas: *; sem subtarefas*

restr: before({$u_0$}, at($r$, $l_2$))

do-setup($c$, $d$, $k$, $l_1$, $l_2$, $p$, $r$) *; método para preparar para mover um contêiner*

tarefa: setup($c$, $r$)

subtarefas: $u_1$ = move-robot($r$, $l_1$, $l_2$), $u_2$ = take($k$, $l_2$, $c$, $d$, $p$),
$u_3$ = load($k$, $l_2$, $c$, $r$), {($u_1$, $u_2$), ($u_2$, $u_3$)}

restr: $u_1 \prec u_2$, before({$u_1$}, on($c$, $d$)), before({$u_1$}, attached($p$, $l_2$)),
before({$u_1$}, in($c$, $p$)), before({$u_1$}, belong($k$, $l_2$)),
before({$u_1$}, at($r$, $l_1$))

unload-robot($c$, $d$, $k$, $l$, $p$, $r$) *; método para finalizar após mover um contêiner*

tarefa: finish($c$, $r$)

subtarefas: $u_1$ = unload($k$, $l$, $c$, $r$), $u_2$ = put($k$, $l$, $c$, $d$, $p$), {($u_1$, $u_2$)}

restr: $u_1 \prec u_2$, before({$u_1$}, attached($p$, $l$)), before({$u_1$}, loaded($r$, $c$)),
before({$u_1$}, top($d$, $p$)), before({$u_1$}, belong($k$, $l$)),
before({$u_1$}, at ($r$, $t$))

Tanto a definição de domínio quanto a de problema de planejamento HTN são idênticas àquelas feitas para planejamento STN; a única diferença reside no fato de, neste caso, o conjunto de métodos ser composto de métodos HTN.

Um plano $\pi = \langle a_1, a_2, ..., a_k \rangle$ é uma solução para um problema de planejamento $P$ se houver uma instância *ground* ($U'$, $C'$) gerada pela decomposição da rede inicial de tarefas $w$ = ($U$, $C$) com uma ordenação total $\langle u_1, u_2, ..., u_k \rangle$ dos vértices de $U'$ de forma que todas as restrições sejam atendidas.

Com relação aos procedimentos de planejamento HTN, eles devem cumprir dois objetivos básicos: instanciar operadores e decompor tarefas. Existem inúmeras formas de se cumprir esses objetivos; por conta disso, há um grande número de diferentes procedimentos HTN possíveis. Um exemplo é o procedimento *HTN-Abstrato* (GHALLAB *et al*., 2004), que é suficientemente genérico para permitir muitas dessas possibilidades. Por exemplo, ele consegue acomodar versões HTN tanto de TFD quando de PFD.

## 2.5 Planejamento HTN Não-Determinístico

Uma ideia interessante em termos de planejamento é a de unir a abordagem mais realista (por permitir o tratamento de muito mais situações reais do que o planejamento clássico permite) do não-determinismo à eficiência proporcionada pelo planejamento HTN. Em (KUTER & NAU, 2004) é apresentada uma técnica para ser usada com planejadores que utilizam encadeamento para a frente em domínios determinísticos, adaptando-os para trabalhar com não-determinismo. O objetivo é aproveitar a eficiência que determinados planejadores determinísticos têm de evitar estados que não sejam promissores (como ocorre com HTN), sem perder suas características de correção e completude. Essa técnica foi aplicada no algoritmo SHOP2, gerando uma versão não-determinística denominada ND-SHOP2. Comparações experimentais entre ND-SHOP2 e MBP mostraram que, em alguns casos, enquanto o tempo de CPU do último cresce exponencialmente em relação ao tamanho do problema, o do primeiro cresce apenas de forma polinomial.

Outro trabalho nesse sentido é o algoritmo de planejamento Yoyo (KUTER *et al*., 2009), que usa um mecanismo baseado em HTN para restringir sua busca (assim como ND-SHOP2) e representação utilizando BDD para raciocinar sobre conjuntos de estados e transições de estados. Assim, da mesma forma que o planejador MBP, Yoyo faz uso de verificação de modelos (implementada com o uso de BDDs) para tratar o não-determinismo; além disso, seus planos também são apresentados sob a forma de políticas. Yoyo é completo e correto, e foi comparado experimentalmente tanto a ND-SHOP2 quanto a MBP, tendo sempre sido muito mais rápido que pelo menos um dos dois e, quase sempre, mais rápido que ambos.

HERMANN (2008) propõe a combinação de planejamento hierárquico e planejamento sob incerteza Knightiana (ou seja, sem considerar a distribuição de probabilidades de efeitos das ações) (KNIGHT, 1921). A estratégia proposta consiste em aplicar o processo de transformação apresentado por KUTER & NAU (2004), aqui chamado de *ND-transformação*, a um algoritmo de planejamento HTN. Uma versão em linguagem Haskell (JONES & HUGHES, 2002) do planejador SHOP (NAU *et al*., 1999), chamada HSHOP, foi submetida à ND-transformação, obtendo-se assim um sistema de planejamento hierárquico para sistemas não-

determinísticos denominado ND-HSHOP. Esse planejador foi comparado experimentalmente a MBP e a HMBP (um planejador baseado em MBP, mas sem fazer uso de BDDs), tendo superado a ambos nos testes efetuados.

## 2.6 Planejamento no Contexto de *Storytelling* Interativo

### 2.6.1 *Storytelling* Interativo

S*torytelling* interativo é uma nova forma de entretenimento digital que une técnicas e ferramentas para a criação, visualização e controle de histórias interativas por meios eletrônicos (CAMANHO *et al*., 2008). Como discutiremos a seguir, trata-se de uma área na qual o uso de algoritmos de planejamento tem larga aplicação.

Existem dois aspectos importantes relacionados a esses sistemas que, via de regra, entram em conflito entre si. De um lado, temos o controle exercido pelo usuário, ou seja, a autonomia a ele concedida em sua interação com o sistema (o que lhe permite uma experiência mais rica ao dar-lhe a sensação de realmente participar da história); do outro lado, temos a coerência da narrativa, que permite ao usuário compreender o relacionamento entre os eventos apresentados durante o desenvolvimento da história (RIEDL & YOUNG, 2006). Estratégias para assegurar a coerência usualmente atingem seu objetivo privando o usuário de um maior grau de liberdade na forma como ele pode alterar os rumos da narrativa, enquanto que estratégias mais voltadas para o controle do usuário tendem a gerar ameaças à coerência da narrativa. Isto cria o problema: como permitir que o usuário tenha um grau satisfatório de controle, fornecendo-lhe uma sensação de imersão na história através da capacidade de modificar o ambiente apresentado pelo sistema de *storytelling* interativo, sem afetar a coerência da narrativa?

Diante dessa questão, existem duas abordagens principais para o desenvolvimento de *storytelling* interativo. Uma delas, inspirada em jogos eletrônicos, é baseada nos personagens (*character-based*) (CAVAZZA *et al*. 2002, CHARLES *et al*. 2001, YOUNG, 2001). Nessa abordagem, agentes autônomos, cada um com seus próprios objetivos, interagem entre si, com o ambiente e com o usuário. A história propriamente dita é o resultado dessas interações e das decisões e

atitudes tomadas pelos agentes e pelo usuário. Assim, o usuário possui um grande grau de liberdade, mas é difícil manter a coerência da narrativa.

A outra abordagem para o desenvolvimento de *storytelling* interativo é influenciada pela Literatura e pelo Cinema, e é baseada no enredo (*plot-based*). Nesse caso, há um forte controle sobre o fluxo da história narrada, evitando-se que o usuário se afaste muito da intenção original do autor; este define uma estrutura para a narrativa com pontos bem definidos, e o único efeito da interação do usuário sobre a narrativa é afetar a forma como a história alcança esses pontos predefinidos (GRASBON & BRAUN, 2001, SPIERLING *et al*., 2002). Esse último modelo é fortemente inspirado no trabalho seminal desenvolvido no início do século XX por Vladimir Propp no campo da Literatura (PROPP, 1968), que, usando como campo de estudo o universo de contos de fada russos, definiu que ações importantes dentro de uma narrativa de um gênero específico podem ser associadas a 31 funções que ocorrem em certas sequências típicas (CIARLINI *et al*., 2005).

Para garantir a coerência narrativa, alguns sistemas que utilizam uma abordagem, a princípio *character-based*, fazem uso de um gerenciador de dramas que monitora as atividades dos personagens – agentes e usuário – e os compara a um grafo do enredo (um grafo parcialmente ordenado de eventos que conduzem o rumo da história). Quando há a possibilidade de ocorrer mais de um evento, o gerenciador de drama analisa qual trará o resultado mais satisfatório e manipula sutilmente o ambiente e os agentes autônomos para induzir à ocorrência daquele evento (RIEDL & YOUNG, 2006). Nesse caso, a estratégia adotada deixa de ser puramente *character-based*, assumindo também características da abordagem *plot-based*.

Um exemplo de aplicação dessa alternativa, que integra características tanto *plot-based* quanto *character-based*, é o sistema interativo Façade (MATEAS & STERN, 2005). Nele, o usuário assume o papel do convidado de um casal de personagens (*Trip* e *Grace*, mostrados na Figura 2.5), podendo interagir através da manipulação de alguns objetos, ações sobre os personagens (como abraçar ou beijar) e conversas em linguagem natural – influenciando, dessa forma, o desenvolvimento da trama. O sistema utiliza um gerenciador de dramas que permite que os personagens sejam autônomos durante a maior parte do tempo, mas muda seus comportamentos para adiantar a trama, conciliando objetivos de nível maior, que

são essenciais à história, com objetivos menores, específicos dos personagens autônomos.

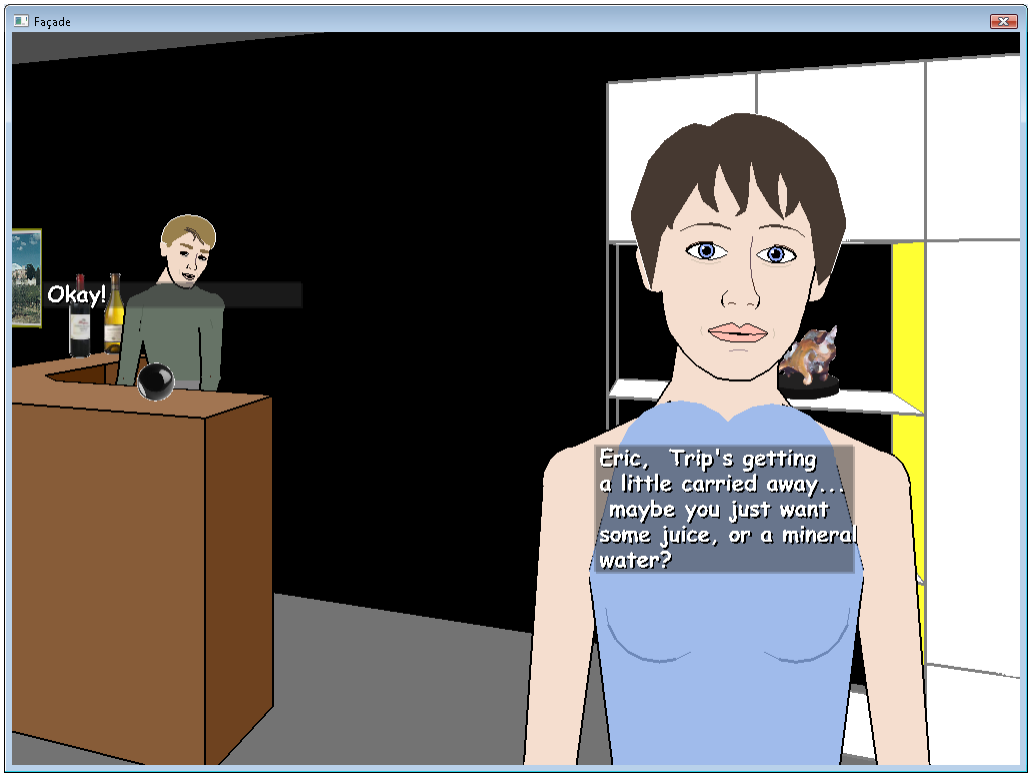


Figura 2.5: Cena do sistema interativo Façade

Outro exemplo de abordagem híbrida é o Erasmatron (CRAWFORD, 1999), que trabalha em cima do conceito de verbos e frases, tentando balancear os aspectos da trama e de personagens da narrativa. As ações são representadas por verbos, com listas de papéis que podem ser atribuídos aos personagens para formar frases.

### 2.6.2 Trabalhos de Planejamento para *Storytelling* Interativo

A principal utilidade de se usar planejamento em *storytelling* interativo é a possibilidade de se gerar sequências de eventos coerentes automaticamente. Dessa forma, proporciona-se a aplicações de *storytelling* interativo uma ferramenta que lhes permite maior grau de autonomia e dinamismo, o que acaba se refletindo em uma maior liberdade do usuário em sua interação com o sistema. Por conta disso, surgiram trabalhos na área de *storytelling* interativo nos quais o planejamento

automático assume uma posição central, destacando-se as pesquisas realizadas pelo *Liquid Narrative Group* e as relacionadas ao sistema *Logtell*. A seguir são apresentados maiores detalhes dessas pesquisas.

#### 2.6.2.1 Liquid Narrative Group

Liquid Narrative é um grupo de pesquisa do Departamento de Ciência da Computação da Universidade do Estado da Carolina do Norte (EUA) que vem trabalhando com *storytelling* interativo baseado em planejamento visando a criação de conteúdo para jogos interativos e outros ambientes virtuais.

Uma proposta feita pelo grupo para resolver o problema de balanceamento entre coerência da narrativa e controle do usuário na geração de narrativas interativas é a mediação de narrativa, uma técnica, baseada no enredo, na qual um agente-autor tem o controle sobre as ações dos personagens (RIEDL & YOUNG, 2006). O sistema gera uma narrativa linear que representa uma história ideal a ser contada, e considera então todas as maneiras pelas quais o usuário pode interagir com o mundo virtual e com os outros personagens. A história gerada inclui ações a serem realizadas pelos personagens controlados pelo sistema e ações que se espera que o personagem controlado pelo usuário realize. Para cada ação do usuário que ameace causar um desvio muito grande da narrativa linear proposta, o sistema gera dinamicamente uma história alternativa a partir do ponto de desvio.

Na mediação de narrativa, é gerado um plano que é a própria narrativa. Esse plano possui marcações indicando tanto as relações temporais entre as ações quanto as relações de causa e efeito entre elas. Ações do usuário que ameacem as relações de causa e efeito são denominadas exceções. Para cada exceção possível, o sistema de mediação de narrativa gera um plano alternativo de história a partir do ponto da exceção. O resultado é uma árvore de planos de história (chamada de árvore de mediação de narrativa) na qual cada plano representa uma história completa – seja a partir do estado inicial do mundo, seja a partir do ponto de desvio – até sua conclusão. Essa árvore é gerada antes da fase de interação com o usuário, para a qual ela serve como guia.

O usuário, no entanto, não tem ciência do plano gerado para a narrativa. Caso ele execute alguma ação que não faça parte do roteiro previamente elaborado, o sistema verifica se a ação causa uma exceção. Em caso afirmativo, a história alternativa apropriada é adotada e passa a ser executada imediatamente.

Para evitar um crescimento exagerado da árvore de mediação, o sistema intervém em algumas ações do usuário, substituindo-as por ações similares ou por uma falha, evitando, assim, os efeitos indesejados. Uma lista mantida pelo sistema indica quais são as falhas associadas a cada ação. Exceções tratadas por meio de intervenção não necessitam de um plano alternativo de história, pois a relação original de causalidade é preservada. Intervenções também são usadas em casos onde o sistema não teria como planejar uma história alternativa – por exemplo, quando o usuário destrói algum elemento ou personagem que venha a ser necessário em algum ponto futuro no desenvolvimento do enredo.

#### 2.6.2.2 Logtell e IPG

Logtell (CIARLINI *et al*., 2005) é um sistema de *storytelling* interativo que, inspirado no trabalho de Propp sobre as funções típicas de uma narrativa, é capaz de criar enredos com base em uma especificação lógica de um determinado gênero literário e na situação inicial da história e, através de animações em 3D, dramatizá-los (Figura 2.6). Basicamente, o que Logtell faz é alternar mecanismos de inferência de objetivos com planejamento para atingir esses objetivos, levando em conta intervenções dos usuários. O sistema apresenta características tanto da abordagem *plot-based* (através da especificação lógica do gênero literário) quanto *character-based* (através de um processo de inferência de objetivos a serem alcançados pelos personagens quando determinadas situações são observadas).

Desde sua versão original, Logtell tem sido aperfeiçoado por intermédio de uma série de extensões. Uma dessas extensões teve como intuito principal conciliar coerência com responsividade e, assim, melhor adaptar Logtell ao ambiente de TV Digital Interativa (CAMANHO, 2009). Nesse novo modelo, os processos de geração, interação e apresentação das histórias ocorrem em paralelo e em tempo real. Outros trabalhos recentes relacionados ao Logtell visam a criação de um nível de não-determinismo na dramatização de eventos (DÓRIA *et al*., 2008), mas mantendo o

compromisso com o enredo gerado. Nesse esquema, um evento pode ser dramatizado de diversas formas diferentes, podendo durar mais ou menos de acordo com a conveniência do processo de *storytelling* e dando possibilidade de alguma interferência do usuário no nível de dramatização, mas a dramatização produz sempre os efeitos esperados pelo enredo.


Figura 2.6: Dramatização no sistema Logtell

A arquitetura atual de Logtell adota um modelo cliente-servidor, conforme se pode ver na Figura 2.7. O lado cliente é responsável pela interação do usuário e dramatização das histórias. Já o servidor é responsável pela simulação. Para cada história sendo contada, há uma aplicação sendo executada no servidor, de forma coordenada com uma ou mais aplicações sendo executadas em diferentes clientes (CIARLINI *et al*., 2008).

Todos os módulos acessam o contexto das histórias (especificado no *Context Database*) através do *Context Control Module* (CCM), que é executado no servidor

e é composto de submódulos. Um deles é o *Context Specifier*, que especifica contextos que servem de base para a geração das histórias, incluindo a parte necessária para a dramatização; nele, o autor especifica todos os detalhes do gênero, da situação inicial e das possibilidades de dramatização de cada evento. A descrição do gênero compreende:

- Um conjunto de operações básicas parametrizadas, com pré- e pós-condições, especificando logicamente os eventos possíveis;
- Um conjunto de regras de inferência de metas, especificadas com uma lógica temporal modal, que definem situações que levam os personagens a buscar determinadas metas;
- Uma biblioteca de planos típicos, correspondendo a combinações típicas de operações para a obtenção de metas específicas, organizada de acordo com hierarquias do tipo "*part-of*" e "*is-a*";
- Descrições lógicas de situações iniciais para as histórias, introduzindo personagens e seus estados iniciais;
- Um autômato não-determinístico para cada operação, especificando formas alternativas de se dramatizar os eventos associados;
- Modelos gráficos dos atores virtuais 3D;
- Scripts e Máquina de Estados Finitos para cotrole de ações e movimentação de câmera dos atores virtuais.

Todos os dados gerados no *Context Specifier* são armazenados no banco de dados, permitindo ao usuário recuperá-los de um contexto para modificar as alternativas de geração de enredos e de dramatização (DÓRIA, 2009).

Outro submódulo do CCM é o *Policy Generator*, que cria políticas de dramatização de acordo com as especificações feitas para cada evento. Após recuperar os dados por intermédio do *Context Specifier*, este submódulo gera as políticas através de um planejador baseado em verificação de modelos, armazenando no banco de dados as políticas geradas. O submódulo *Real Time Access* provê acesso em tempo real aos demais módulos do sistema às informações do contexto relevantes para a geração de enredos e sua dramatização.

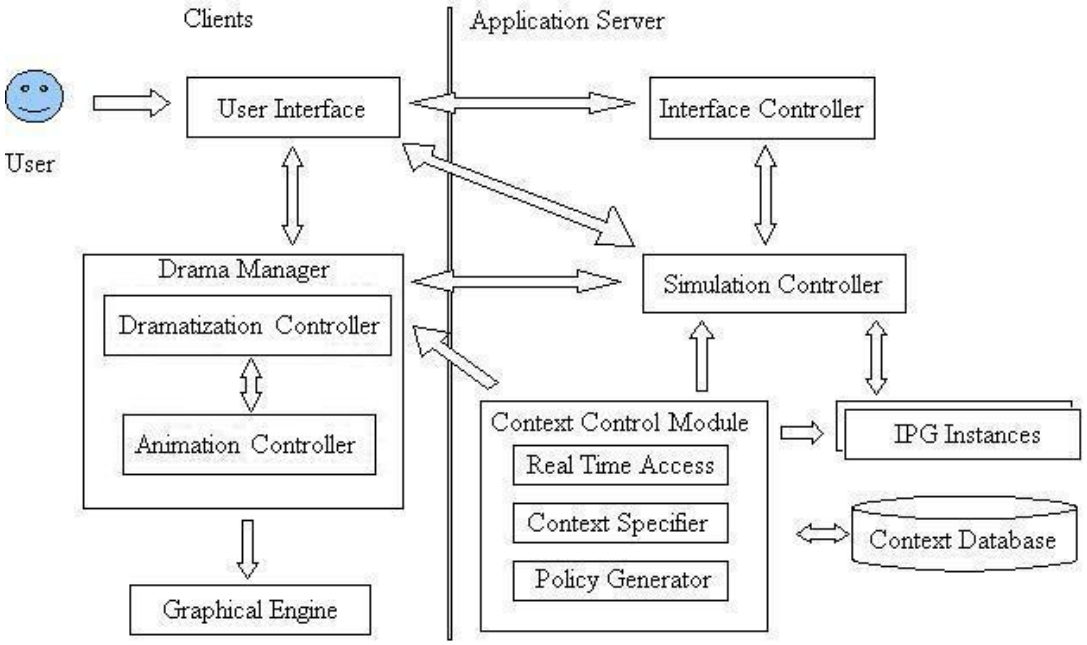


Figura 2.7: Arquitetura do Logtell (DÓRIA, 2009)

Para criar um encadeamento lógico de eventos, o Logtell faz uso de algoritmos de planejamento. Existe um subsistema específico responsável pela execução desse trabalho de geração de enredos, o *Interactive Plot Generator* (IPG). IPG (CIARLINI, 1999) é um planejador não-linear (ou seja, que planeja no espaço de planos, lidando com ordens parciais de eventos) que cria enredos como uma sequência de capítulos, sendo que cada capítulo corresponde a um ciclo no qual a interferência do usuário é incorporada, metas são inferidas e planejamento é utilizado para alcançar as metas. O IPG pode trabalhar com domínios diversos, de contos de fadas a jogos corporativos. Além disso, as técnicas usadas pelo planejador permitem a obtenção um grau interessante de diversidade: o protótipo inicial do Logtell foi usado com base em um subgênero específico de conto de fadas (“espadas e dragões”) e, mesmo nesse contexto simples, foi capaz de gerar uma quantidade considerável de enredos diferentes.

O processo começa pela inferência de metas dos personagens a partir da situação inicial. A partir daí, o IPG encadeia eventos no enredo de modo a permitir que os personagens atinjam suas metas. Quando o planejador conclui que todas as metas foram atingidas ou abandonadas, tem-se o primeiro capítulo quase concluído.

O aspecto de interatividade do Logtell começa na apresentação ao usuário do enredo gerado pelo IPG. Se o usuário não gostar da história, pode solicitar ao IPG que gere uma versão alternativa para o capítulo. Em caso de não interferência do usuário no processo, capítulos são gerados continuamente pela inferência de novas metas a partir das situações criadas no capítulo anterior. O processo segue alternando inferência de metas e planejamento até o momento em que ou o usuário decide parar ou não se infere mais nenhuma nova meta. Outra forma de interferência do usuário é forçando a ocorrência de eventos e situações nos capítulos futuros. A interação no Logtell está centrada na fase de geração dos enredos e ocorre de forma indireta com a escolha de alternativas e as solicitações de inclusão de eventos e situações no próximo capítulo. Uma vez que um capítulo é gerado e incorporado à história, o usuário não intervém diretamente na dramatização.

O cliente interage com o sistema através da *User Interface*, que informa ao *Interface Controller*, no lado do servidor, as interações desejadas. O *Drama Manager* recebe do *Simulation Controller* o próximo evento a ser dramatizado e controla as instâncias dos atores para cada personagem em um ambiente 3D em execução no motor de jogos 3D (*Graphical Engine*). Esse trabalho é realizado através de seus submódulos (*Dramatization Controller* e *Animation Controller*), da seguinte forma: o *Dramatization Controller* recupera as políticas geradas pelo *Policy Generator*, selecionando-as e executando-as de acordo com a necessidade da dramatização; além disso, recebe do *Simulation Controller* a solicitação de eventos a dramatizar e indica ao *Animation Controller* qual micro-ação deve ser animada em cada momento, de acordo com a política adotada. Por fim, trata as solicitações feitas pelo *Simulation Controller* para atrasar ou acelerar a dramatização, permitindo a sincronização da dramatização com a geração dos enredos.

No servidor, o *Interface Controller* centraliza as sugestões feitas pelos diversos clientes. Quando vários usuários compartilham a mesma história, as interações são selecionadas de acordo com o número de usuários que as solicitaram; havendo apenas um, as sugestões são automaticamente enviadas ao *Simulation Controller*.

O *Simulation Controller* possui múltiplas responsabilidades: informar ao *Drama Manager*, no lado do cliente, quais os próximos eventos a serem dramatizados; receber pedidos de interação e incorporá-los à história (desde que sejam coerentes

com a especificação do gênero literário); selecionar intervenções viáveis a serem sugeridas ao usuário; ativar o IPG, a fim de obter os próximos eventos a serem dramatizados; e sincronizar as dramatizações com a geração do enredo.

# 3 Modelo de Geração de Enredos com Não-Determinismo

Neste capítulo, é apresentada uma proposta de modelo para geração de enredos não-determinísticos para *storytelling* interativo. O modelo deverá ser incorporado ao sistema Logtell, descrito no capítulo 2, através da integração entre o planejador de ordem parcial do IPG e um novo planejador hierárquico não-determinístico.

O objetivo principal é aumentar o grau de diversidade dos enredos gerados, incorporando eventos com mais de um conjunto de efeitos possível. A seleção de qual conjunto de efeitos será efetivamente levado em conta para um determinado evento pode ser postergada até o processo de dramatização, através de um processo de escolha não-determinística - permitindo, assim, que diferentes resultados sejam alcançados pelos eventos de um capítulo. Entretanto, para que o não-determinismo não afete a coerência das histórias apresentadas, é necessário que o planejador não-determinístico considere todas as possíveis situações (a serem testadas) após cada evento e gere previamente as ramificações que garantam o cumprimento dos objetivos a serem atingidos em cada capítulo, independente do resultado dos eventos. Para aumentar o grau de variação entre os possíveis finais de um capítulo, torna-se essencial também que objetivos possam corresponder simplesmente a tentativas de estabelecer uma certa situação. Desse modo, um capítulo pode acabar com a situação tendo sido gerada ou não, o que é importante para a carga dramática da narrativa.

Com o objetivo de nortear a seleção dos eventos que virão a fazer parte do plano, adotou-se o uso de redes hierárquicas de tarefas (HTN). Uma das vantagens de tal abordagem é que o uso de HTN permite que o planejador adote como critério de validade para os planos gerados a satisfação de uma rede de tarefas – algo mais genérico do que o estabelecimento de um estado desejado, como ocorre atualmente

com o IPG. Outra vantagem é que o uso de HTN permite um direcionamento na seleção dos eventos (devido à forma como os métodos tratam a execução das tarefas às quais se aplicam) capaz de, ao considerar apenas formas promissoras (e predefinidas) de se estabelecer o plano, acelerar o processo de planejamento – um efeito mais do que desejável em uma aplicação de *storytelling* interativo, principalmente quando passamos a contar com várias alternativas que podem aumentar de forma considerável nosso espaço de busca.

O uso de redes hierárquicas de tarefas também influencia na variedade de enredos possíveis, já que pode haver mais de um método possível para realizar a mesma tarefa. Dessa forma, definições de nível mais alto relacionadas ao gênero literário podem resultar em diferentes sequências de eventos no nível mais baixo, bastando para isso que diferentes métodos sejam adotados.

No decorrer deste capítulo, serão apresentados os requisitos a serem atendidos pelo modelo e uma proposta de nova arquitetura, de modo a permitir que o Logtell incorpore a geração e o tratamento de enredos não-determinísticos à sua gama de recursos. Em seguida, serão mostradas, com maior grau de detalhe, as alterações necessárias para a adoção desse novo modelo: modificações na sintaxe e semântica da definição dos operadores do domínio (não apenas para incluir o conceito de não-determinismo, mas também para permitir o uso de planejamento HTN); inclusão de um novo módulo para coordenar a geração de capítulos que compreendam não-determinismo; alterações no planejamento de ordem parcial do IPG para que trabalhe de forma integrada com o planejador hierárquico não-determinístico, incorporando o conceito de tentativas; inclusão do planejador hierárquico não-determinístico; e, por fim, as alterações necessárias nos demais módulos do Logtell.

## 3.1 Requisitos

A fim de orientar as decisões a serem tomadas na elaboração do modelo proposto para a geração de enredos não-determinísticos, foi preciso definir uma lista de aspectos que deveriam ser levados em conta por tal modelo. A seguir, serão mostrados os requisitos que foram considerados relevantes para a composição dessa lista.

### 3.1.1 Suporte a Eventos Não-Determinísticos

Uma das principais formas de se enriquecer a experiência do usuário de um sistema de *storytelling* interativo é promover variedade não só na forma como as histórias são apresentadas, mas também no próprio desenvolvimento dessas histórias – incluindo a possibilidade de, a partir de um mesmo evento incorporado ao enredo, chegar-se a diferentes desfechos. Entretanto, embora tal liberdade possa ser considerada essencial para garantir o interesse gerado por tal tipo de aplicação, não se deve permitir que ela venha a se tornar uma ameaça à coerência da história que está sendo contada (de acordo com as regras determinadas para o seu contexto, sejam elas oriundas do gênero literário ao qual ela pertence ou, simplesmente, da imaginação do autor), pois dessa forma as expectativas do usuário podem não ser devidamente atendidas por causa da criação de uma história sem fundamento.

Um meio de se proporcionar um maior grau de interatividade é através da possibilidade de interferência dos usuários nos eventos enquanto estes estão sendo dramatizados. Um sistema com grau de liberdade suficiente para permitir alterações no enredo no último instante possível (ou seja, durante a própria dramatização), ao mesmo tempo em que concede maior poder ao usuário, também oferece a possibilidade de se gerar mais enredos diferentes a partir de um mesmo ponto da narrativa. Para isso, é necessário o estabelecimento de uma estrutura que apresente alternativas para o enredo, e que a seleção de quais alternativas serão adotadas seja postergada até o processo de dramatização, de forma automática ou através da interferência direta do usuário. Os efeitos efetivamente selecionados serão levados em conta na continuidade do enredo, garantindo, dessa forma, a variedade desejada.

Vale ressaltar que, para não permitir que tal grau de liberdade se torne uma ameaça à coerência do enredo gerado, é necessário que a estrutura que o representa seja montada de forma que todos os encadeamentos possíveis de seus eventos passem somente por estados que não comprometam a lógica do contexto formalmente especificado.

### 3.1.2 Controle do Nível de Não-Determinismo

A inclusão de eventos com efeitos não-determinísticos, aliada ao compromisso de se garantir a coerência dos enredos gerados, resulta em um maior nível de

complexidade durante a etapa de planejamento. A geração de planos sob tais condições pode, caso haja um número muito grande de alternativas encadeadas, provocar uma explosão combinatória. Além disso, existe a necessidade de se garantir um desenvolvimento aceitável do enredo (dentro das especificações do gênero literário escolhido), o que, embora limite o número de alternativas possíveis, acaba tornando o processo de planejamento mais custoso. A existência de tais fatores pode resultar em uma grande demanda de tempo de processamento.

Para evitar o impacto negativo de possíveis interrupções no fluxo da narrativa, devemos apresentar um meio de controlar o nível de não-determinismo desejado, através do estabelecimento de um valor máximo para o número de ramificações que podem ocorrer no plano gerado. Tal valor pode ser verificado pelo planejador durante o processo de geração do plano e, caso seja ultrapassado, deve resultar em falha no plano que está sendo desenvolvido, forçando assim a busca por outro plano que atenda às condições estabelecidas (incluindo, naturalmente, o nível máximo de não-determinismo).

Ao se estabelecer um número máximo de ramificações, impede-se que a geração de um novo capítulo (resultado da busca pela satisfação de condições estabelecidas pelas regras de inferência de objetivos e pelas sugestões introduzidas, conjugadas com as regras do gênero literário) incorra em um tempo de processamento muito alto. Soluções que ultrapassem o limite estabelecido seriam descartadas em prol de alguma outra que se mantenha dentro dele e, caso não seja possível encontrar nenhuma, a sugestão introduzida seria simplesmente descartada.

### 3.1.3 Incorporação do Conceito de Tentativas

O IPG funciona através de inferência de objetivos (com base no estado atual) para, em seguida, gerar um plano de ordem parcial que permita o estabelecimento desses objetivos. Se deseja-se que, através de eventos não-determinísticos, objetivos sejam certamente atingidos, mas produzindo situações diferentes, é necessário incorporar mecanismos em que o alcance de um objetivo não seja necessariamente o estabelecimento de fatos em determinados momentos.

Se, por um lado, a adoção de eventos não-determinísticos permite um aumento no grau de diversidade das histórias, por outro, temos uma dificuldade maior para

garantir que se alcancem determinadas condições. Como, para esses eventos não-determinísticos, não há meios de se estabelecer quais efeitos se tornarão efetivamente válidos durante o planejamento, surge a dificuldade de se garantir que algum plano consiga cumprir o objetivo estabelecido, pois todos os possíveis encadeamentos de eventos devem levar a um estado que o atenda. Além disso, é desejável que haja um mecanismo que permita aumentar a variedade dos enredos gerados, através do possível estabelecimento de novas condições, mas sem que isso comprometa a geração dos planos (no caso de não ser possível estabelecer essas condições sem causar falha no planejamento).

Com o fim de atender a esses requisitos, pode-se adotar o conceito de *tentativa*. Neste caso, além das tarefas que devem ser cumpridas (sem direito a falhas), o IPG poderia incorporar condições que se deve tentar cumprir – ou seja, espera-se que o planejador procure estabelecê-las no plano, mas, caso não seja possível, isso não resultaria em falha. Dessa forma, é possível criar condições para que se diversifiquem os enredos, sem que para isso se ameace a possibilidade de geração dos planos.

### 3.1.4 Processamento em Paralelo das Diversas Alternativas

Embora seja possível minimizar a possibilidade de interrupções na continuidade da apresentação da história através do controle do nível de não-determinismo, deve-se procurar outros recursos que ajudem a manter o fluxo da narrativa – caso contrário, pode haver a necessidade de um controle muito estrito sobre o número de alternativas a serem aceitas, o que enfraqueceria o benefício obtido pela introdução do não-determinismo. Como os planos gerados podem possuir mais de um estado final possível, pode-se acelerar o planejamento através do processamento em paralelo de novos planos (que seriam opções para o capítulo seguinte) – assumindo, como estado inicial para cada um deles, os diferentes estados finais alternativos alcançáveis pelo capítulo atual. Mesmo que, em uma implementação específica, os recursos tecnológicos (tanto em termos de *hardware* quanto de *software*) disponíveis dificultem um paralelismo real na execução do planejamento, ainda assim se faz necessário um mecanismo de controle que simule tal comportamento, de modo a tornar possível a apresentação, de modo simultâneo, de todas as novas

alternativas geradas – caso contrário, ter-se-ia apenas uma visão parcial dos possíveis caminhos de execução do plano.

A fim de melhorar ainda mais o desempenho, deve-se monitorar o andamento da dramatização da narrativa e, conforme as alternativas existentes no enredo forem selecionadas, planos ainda em processamento cujo estado inicial não fosse mais alcançável pela trama em andamento seriam descartados.

### 3.1.5 Eficiência na Geração de Enredos

Um sistema de *storytelling* interativo, principalmente se considerado no contexto de TV Digital Interativa, deve satisfazer ao usuário com relação a seu tempo de resposta. Seria extremamente frustrante ter que ficar aguardando pelo sistema enquanto uma parte do enredo é gerada, por exemplo. Tal requisito torna-se ainda mais crítico quando os enredos parciais gerados possuem vários estados finais distintos (cada um deles atingido através de um dos possíveis caminhos de execução do plano, conforme os efeitos não-determinísticos levados em consideração), e é necessário planejar a partir de cada um deles, separadamente – pois todos têm a possibilidade de serem alcançados no decorrer da dramatização.

A adoção do não-determinismo, ao mesmo tempo em que permite incrementar o nível de diversidade dos enredos apresentados, cria uma demanda maior de processamento para planejar considerando as diferentes ramificações encontradas durante o planejamento. Para contrabalançar esse efeito, é necessária a adoção de recursos que melhorem o desempenho da aplicação, de modo a encontrarmos um equilíbrio entre diversidade e eficiência. Uma das formas de se alcançar tal objetivo é através do controle do nível de não-determinismo, citado anteriormente. Outra forma de tentar garantir a continuidade do fluxo, também já citada, é através do processamento em paralelo das diversas alternativas. Por fim, a utilização de conhecimento do domínio, como em métodos de planejamento HTN, parece importante para garantir um bom desempenho.

## 3.2 Proposta de Nova Arquitetura

A nova arquitetura proposta prevê a alteração da arquitetura atual do Logtell através da inclusão de um novo planejador, não-determinístico e baseado em redes hierárquicas de tarefas (HTN), que passará a trabalhar conjuntamente com o IPG, que também deverá ser alterado. A integração do IPG com esse novo planejador passa a constituir um novo módulo, denominado *Chapter Simulator*.

Para coordenar as atividades entre o *Chapter Simulator* e os demais módulos do Logtell, propõe-se a criação de um módulo de controle, o *Chapter Controller*, que passa a ser responsável pela geração do enredo, capítulo a capítulo, incluindo o aspecto do não-determinismo e o processamento em paralelo das diversas alternativas existentes a cada momento. Juntos, o *Chapter Controller* e o *Chapter Simulator* compõem um módulo de nível mais alto, o *NDet-IPG* – que passa, então, a exercer o papel antes cumprido apenas pelo IPG na arquitetura do Logtell, sendo, com seus submódulos e divisão de tarefas, o responsável pela geração do plano a ser usado como insumo para o processo de dramatização.

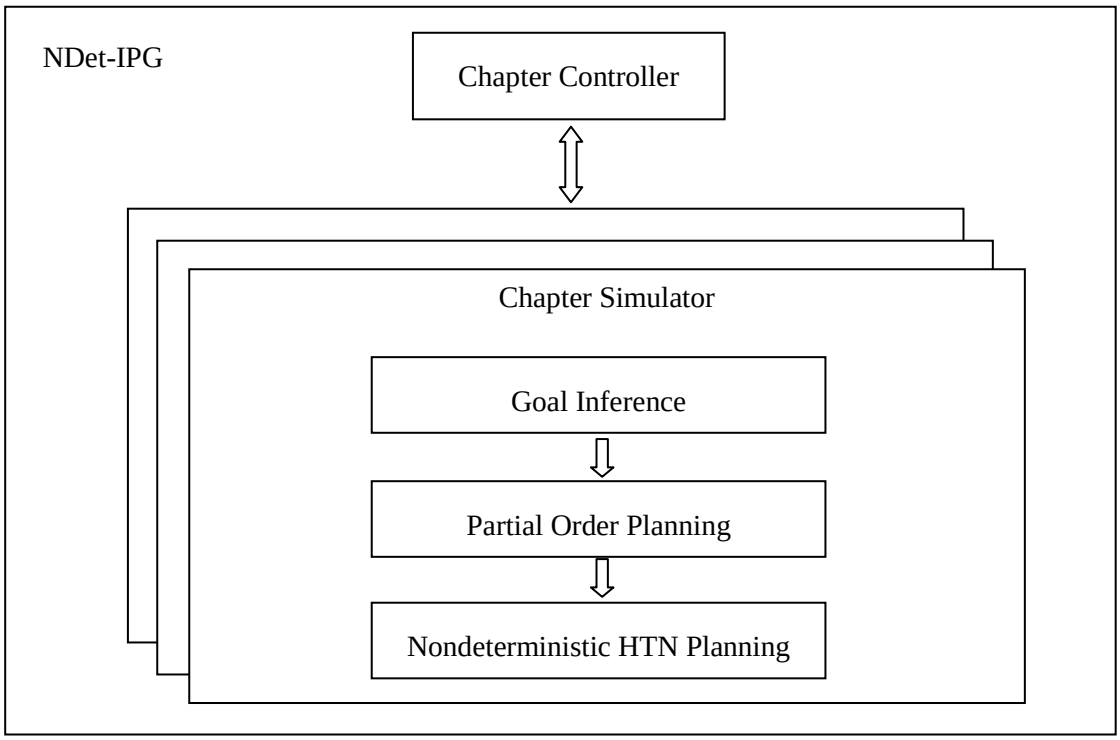


Figura 3.1: Esquema funcional do módulo NDet-IPG

Conforme ilustrado na Figura 3.1, o *Chapter Controller* deverá, a princípio, funcionar como um agente intermediário para as entradas necessárias ao *Chapter Simulator* - recebendo os operadores e situações a inserir e acionando-o em seguida, com a passagem desses parâmetros. O *Chapter Simulator* utiliza as especificações recebidas do *Chapter Controller* para, utilizando as regras de inferência de objetivos e as técnicas de planejamento de ordem parcial que atualmente são parte do IPG, gerar uma rede hierárquica de tarefas inicial (ou seja, um conjunto de tarefas a serem cumpridas, juntamente com possíveis restrições de ordem parcial) a ser repassada ao planejador hierárquico não-determinístico. Para tal, as regras de inferência deverão passar a gerar planos menos detalhados do que atualmente. A ideia é que em vez de conterem apenas ações específicas do domínio, contenham essencialmente umas poucas operações de nível mais alto, correspondentes a tarefas HTN a serem cumpridas (o que ocorrerá pela existência de tarefas HTN que tenham como efeitos os objetivos inferidos). Dessa forma, a etapa de planejamento com ordem parcial deverá ser bem mais rápida. De posse dessa rede de tarefas inicial, o planejador hierárquico não-determinístico tentará gerar um plano que, para todos os caminhos possíveis, atenda à rede de tarefas recebida como entrada. Em caso de falha, descartam-se as sugestões que haviam sido passadas inicialmente ao *Chapter Controller* e prossegue-se com o planejamento normalmente sem elas; eventualmente, novas sugestões podem ser enviadas, reiniciando o processo.

Caso o planejador hierárquico não-determinístico consiga gerar um plano que cumpra a rede de tarefas estabelecida, esse resultado será retornado ao *Chapter Controller*, que deverá, então, repassá-lo aos módulos do Logtell responsáveis pela dramatização dos eventos, passando, a partir daí, a gerenciar as intervenções do usuário e a coordenar a geração de novos planos a partir dos diferentes estados finais alcançados por esse plano não-determinístico.

A Figura 3.2 mostra como fica a arquitetura do Logtell dentro desse novo modelo, incluindo a adoção do novo módulo de planejamento NDet-IPG.

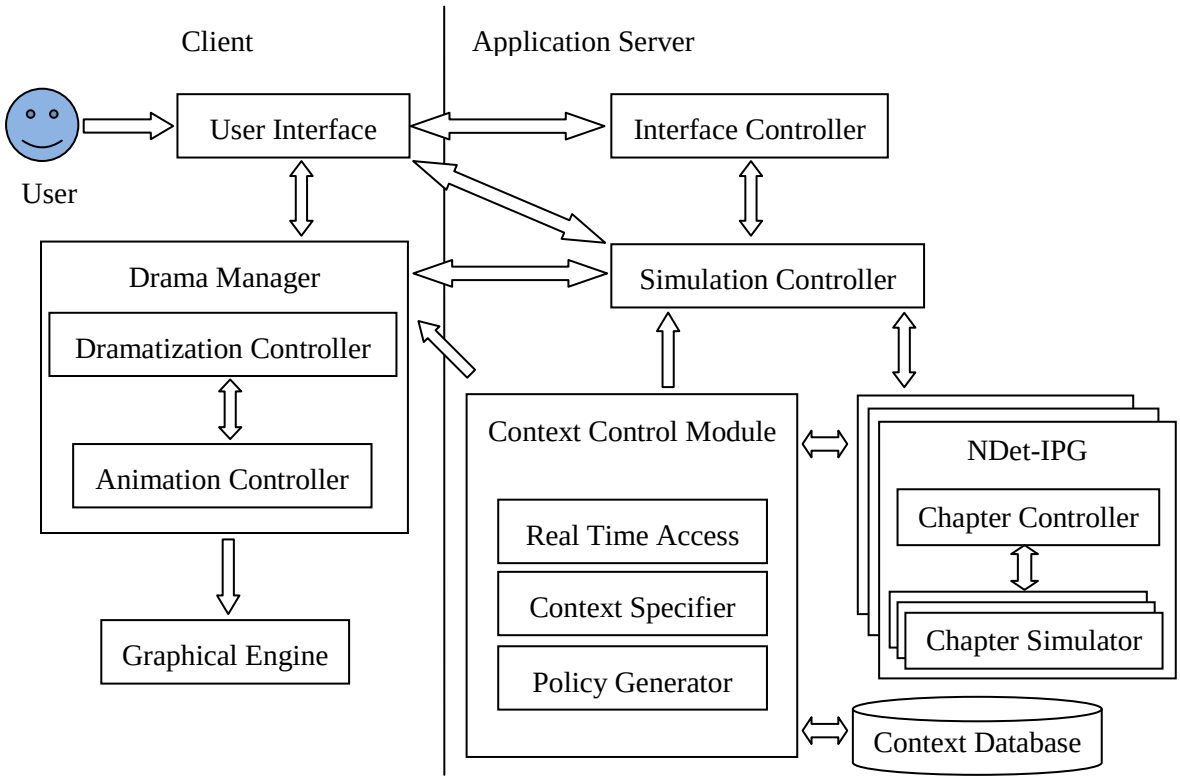


Figura 3.2: Nova arquitetura do Logtell, contemplando o novo módulo composto pelo planejador NDet-IPG

## 3.3 Modificações na Definição de Operadores

Atualmente, as definições de gênero literário utilizadas pelo IPG estabelecem, para cada operador, um conjunto de efeitos determinísticos que especificam os fatos que se tornam válidos (ou deixam de ser válidos) quando uma instância do operador é executada sob a forma de um evento. Para que possamos trabalhar com efeitos não-determinísticos, é necessário alterar a sintaxe dessas definições, a fim de que possam também incluir, além desses efeitos que sempre devem ser levados em conta, os possíveis efeitos não-determinísticos dos eventos.

Por exemplo, no contexto baseado no subgênero "Espadas e Dragões" utilizado na versão atual do Logtell, existe o operador *kidnap(VIL, VIC)*, que estabelece que um personagem (*VIL*), cumprindo o papel de vilão, rapta um personagem que exerce o papel de vítima (*VIC*), conduzindo este último ao seu covil. Este operador possui como efeitos, todos determinísticos, além do estabelecimento da condição da vítima estar raptada pelo vilão, o fato de que ambos os personagens passam a estar no local

onde o vilão reside. Uma forma de acrescentar carga dramática ao enredo seria substituir tal operador por uma versão não-determinística, *try_to_kidnap(VIL, VIC)*, que poderia apresentar como efeito, além do mesmo resultado de sua contraparte determinística, também uma alternativa em que o vilão, no afã de capturar uma vítima que talvez apresentasse alguma resistência à sua captura, acabasse matando-a no processo. Dessa forma, um mesmo operador poderia conduzir a dois estados divergentes (vítima raptada ou vítima morta), que acabariam por gerar alternativas diferentes no desenvolvimento da história.

Esses efeitos não-determinísticos são descritos como um conjunto de conjuntos de efeitos, em que cada um desses conjuntos de efeitos é uma alternativa a ser escolhida para o processo de dramatização – durante o planejamento, não se sabe qual desses conjuntos será efetivamente integrado ao enredo. Operadores totalmente determinísticos caracterizam-se por não ter nenhum conjunto de efeitos não-determinísticos.

Além disso, a descrição dos operadores requer meios de indicar a hierarquia entre eles. O planejamento HTN se desenvolve através da decomposição de determinados operadores, compostos, em suboperadores, que podem também ser compostos – seguindo, assim, o processo sucessivamente. Dessa forma, os operadores agora incluem, em sua definição, os respectivos suboperadores (caso existam), assim como as restrições de ordenação relativas a eles. Por exemplo, além dos operadores *fight* (que indica uma luta entre dois personagens) e *kill* (que indica que um personagem mata outro), existentes na versão anterior do Logtell, podemos incluir um operador composto *defeat* contendo, como suboperadores, os operadores *fight* e *kill*, com uma restrição de ordem impondo que o operador *fight* deve vir antes do operador *kill*.

Para implementar o conceito de método do planejamento HTN, usamos uma estrutura de herança de operadores. Determinados operadores podem ser considerados generalizações de operadores mais específicos. Esses operadores de nível mais baixo podem corresponder a eventos do domínio ou podem ser também generalizações, permitindo que haja uma herança em cascata.

Em nossa formalização do novo modelo, chamaremos de *operadores complexos* aqueles que são genéricos e/ou possuem suboperadores (correspondendo, assim, às

tarefas não primitivas do planejamento HTN). Operadores de nível mais simples, que não são herdados por nenhum outro nem possuem suboperadores, serão chamados de *operadores básicos* (correspondendo às tarefas primitivas do planejamento HTN). Vale ressaltar que, para efeito do planejamento HTN não-determinístico, só estamos considerando os efeitos (determinísticos ou não) quando se trata da definição dos operadores básicos. Os efeitos especificados para operadores complexos são usados apenas para a inclusão desse tipo de operador na rede inicial de tarefas, com planejamento de ordem parcial. Após isso, durante o planejamento HTN não-determinístico, trabalha-se com os estados que são atingíveis através da execução de instâncias dos operadores básicos, as quais são diretamente dramatizáveis.

## 3.4 Controle da Geração dos Capítulos: Chapter Controller

A fim de controlar as alternativas de continuidade do capítulo corrente, contamos agora com um novo módulo, o *Chapter Controller*. Suas funções são, essencialmente:

- Receber do *Simulation Controller* as sugestões relativas a estados a serem atingidos e/ou eventos a serem incorporados ao enredo, e repassá-las ao *Chapter Simulator*;
- Repassar os planos não-determinísticos gerados pelo *Chapter Simulator* ao *Simulation Controller* (para que, a partir daí, passem a ser dramatizados para o usuário);
- Antecipar o planejamento dos próximos capítulos, gerando vários planos em paralelo, tendo como estado inicial, cada um dos possíveis estados finais do último capítulo gerado.

Caso receba, do *Simulation Controller*, alguma mensagem de replanejamento derivada de alguma intervenção do usuário (inserção de eventos ou situações), o *Chapter Controller* deve interromper as execuções em paralelo responsáveis pela geração do próximo capítulo, e reiniciar o processo com a inclusão das sugestões introduzidas. Por outro lado, se o usuário intervier na dramatização (causando a seleção de efeitos específicos para eventos não-determinísticos), o *Chapter*

*Controller* deve interromper todas as execuções em paralelo que não tenham como estado inicial algum dos estados finais alcançáveis somente pela ramificação escolhida.

## 3.5 Geração dos Capítulos: Chapter Simulator

### 3.5.1 Inferência de Metas e Planejamento de Ordem Parcial

A tarefa de gerar os planos a serem usados no processo de dramatização cabe ao *Chapter Simulator*. Este novo módulo agrega as funcionalidades do planejador de ordem parcial IPG, combinadas com o novo planejador hierárquico não-determinístico. Toda a parte relativa ao planejamento com não-determinismo está concentrada neste novo planejador, especificamente desenvolvido para esse fim; entretanto, o novo modelo também requer algumas modificações no planejamento de ordem parcial.

Uma delas está na interpretação das definições do gênero literário – mais especificamente, no que diz respeito aos operadores. Agora, cada operador pode possuir, além de um conjunto de efeitos determinísticos, conjuntos alternativos de efeitos não-determinísticos. O *Chapter Simulator*, ao receber uma entrada do *Chapter Controller* especificando um estado meta a ser atingido, irá usar uma versão adaptada do IPG para tentar montar uma rede de tarefas que permita atingir aquele estado. Entretanto, o planejamento de ordem parcial do IPG permanece determinístico e, para tal, deve adotar um comportamento pessimista: para efeito de estabelecimento de precondições (e do estado meta), deverão ser considerados apenas os efeitos determinísticos de cada operador; por outro lado, para efeito de possíveis quebras de estabelecimento de precondições (e do estado meta), deverão ser levados em conta todos os efeitos, determinísticos ou não.

Para aumentar o potencial de variação dos enredos gerados, é interessante que as redes de tarefas a serem cumpridas possuam operadores complexos tanto quanto possível, pois assim permite-se que um número maior de diferentes instanciações desses operadores possa ser utilizado ao se gerar enredos alternativos. Com este fim, podem-se usar os custos – valores que os operadores utilizados pelo IPG possuem, atualmente, como um de seus atributos. Dessa forma, os operadores de nível de

generalização mais alto e os que possuem suboperadores são especificados com custo mais baixo, tendo, assim, prioridade durante o processo de planejamento de ordem parcial.

Além disso, espera-se manter algum controle sobre o tamanho dos planos gerados, através da adoção de um limite no número de operadores. Redes de tarefas a serem passadas ao planejador não-determinístico devem ser geradas sob a observação desse critério.

Outro recurso incorporado ao *Chapter Simulator*, mais uma vez com o intuito de enriquecer a gama de possibilidades de geração de enredos, é a adoção do conceito de *tentativas*. Passam a fazer parte da semântica dos operadores o uso de precondições do tipo *tried*(*LITERAL*, *N*), onde *LITERAL* corresponde a algum literal – ou seja, um fato ou a negação de um fato - do domínio que se deve tentar estabelecer. *N* é um número inteiro que corresponde ao nível de tolerância de possível falha. De forma simplificada, foi estabelecido que *N* é tal que, quanto maior o seu valor, maior será o número de eventos encadeados necessários ao estabelecimento do *LITERAL* que podem deixar de ocorrer por não terem suas precondições satisfeitas. O processo funciona de tal forma que eventos inseridos para o estabelecimento de uma precondição do tipo *tried*(*LITERAL*, *N*) são "tentativas" de execução de eventos que ocorrerão apenas se suas precondições forem válidas. Consequentemente, pode-se dizer que, para valores maiores de *N*, é menor o grau de exigência quanto ao estabelecimento de *LITERAL* e a desistência do estabelecimento de *LITERAL* pode ocorrer mais cedo.

De uma maneira mais formal, quando *N* = 1, tal precondição será atendida se houver um evento que seguramente ocorrerá antes do evento que tem essa precondição e que tenha *LITERAL* definido como um de seus efeitos – determinísticos ou não. Obviamente, o estabelecimento de LITERAL por um efeito, determinístico ou não, não deve ser bloqueado por algum evento que possa ocorrer entre o momento do estabelecimento e o momento em que a precondição deveria valer. Para valores de *N* maiores que 1, adota-se o conceito de tentativa para as precondições do evento que atenda a *tried*(*LITERAL*, *N*), mas agora com um nível de tolerância reduzido de uma unidade. Dessa forma, uma condição do tipo *tried*(*LITERAL*, *N*) requer o cumprimento de condições do tipo *tried*(*PRECOND*, *N*-

*1*) para cada precondição *PRECOND* do evento que estabelece *tried*(*LITERAL, N*). Um evento *OP* nessas condições deve ser considerado uma tentativa, indicada por *attempt*(*OP*, *N-1*), que só será incorporada ao plano final se todas as suas precondições forem atendidas. De acordo com a semântica apresentada, um evento do tipo *attempt*(*OP*, 0) tem que acontecer, ou seja, requer o estabelecimento determinístico de todas as suas precondições. Da mesma forma, operadores do tipo *attempt*(*OP*, *N*) só estabelecem implicitamente efeitos do tipo *tried*(*LITERAL, N+1*), independente de esses efeitos serem determinísticos ou não. Operadores podem especificar explicitamente, dentre seus efeitos determinísticos, o estabelecimento de uma tentativa *tried*(*LITERAL, N*). Isso é útil para expressar que um operador complexo pode ser usado para representar certa tentativa, com um grau específico de tolerância. Se um operador complexo é inserido em um plano como uma tentativa com nível de tolerância *N*, seu efeitos do tipo *tried*(*LITERAL, M*) passam a ser considerados como *tried*(*LITERAL, M+N+1*).

Por fim, vale ressaltar que, para que o tratamento do conceito de tentativa se mostre viável para a utilização do planejamento de ordem parcial, efeitos explicitamente definidos como do tipo *tried* só devem ser permitidos dentre os efeitos determinísticos.

### 3.5.2 Planejamento Hierárquico Não-Determinístico

A fim de atender ao requisito de diversidade de enredos através do uso de operadores não-determinísticos, foi necessário o desenvolvimento de um novo planejador, que recebe como entrada uma especificação de rede de tarefas e deve gerar como resultado uma árvore de decisão percorrendo todos os caminhos encontrados durante o processo de planejamento. Nesta árvore, os nós correspondem a eventos (não-determinísticos) e as arestas a condições (disjuntas) de teste do estado atingido após a última ação, que determinam o próximo evento. Tal árvore deverá forçosamente ter cumprido todo o HTN em cada estado final por ela alcançável. O uso de um parâmetro estabelecendo um limite máximo para o número de ramificações ajuda a evitar que o tamanho do plano se torne impraticável.

O planejamento ocorre através de um processo contínuo de decomposição ou especialização de operadores complexos. Generalizações são substituídas por

especializações cujas precondições estejam sendo atendidas (ou seja, funcionam como os métodos HTN ao serem selecionados para o cumprimento de uma tarefa) e composições, por seus suboperadores (que herdam as restrições de ordem do operador complexo original).

Eventos são incorporados ao plano na ordem em que devem ser executados (a cada passo, é considerado um operador que não tenha antecessores, ou cujos antecessores já tenham sido incorporados ao plano). Dessa forma, para cada ramo da árvore de decisão representativa do plano não-determinístico, o planejador é capaz de materializar o estado corrente de forma a avaliar o atendimento de precondições – seja para a seleção de um operador adequado que especialize um operador genérico, seja para considerar a viabilidade de inclusão de um operador básico ao plano.

Vale ressaltar que o modelo considera a possibilidade de se conjugar generalização com composição. Quando um operador é especialização de outro operador com suboperadores, ele herda esses suboperadores (juntamente com suas restrições parciais de ordem). Há, ainda, flexibilidade para que uma especialização acrescente novas restrições de ordem aos suboperadores do operador complexo original. Além disso, é possível que um operador que especialize outro tenha, ele próprio, seus suboperadores, e que possua restrições de ordem em sua definição que conjuguem seus suboperadores específicos com aqueles herdados do operador complexo. Consegue-se assim uma expressividade maior do que a obtida na definição tradicionalmente adotada por planejadores HTN para as tarefas e métodos.

Como o planejador de ordem parcial pode inserir eventos que serão apenas tentativas, o planejador não-determinístico deve analisar a incorporação ou não do evento em cada ramo da árvore, conforme suas precondições sejam satisfeitas. Quando as precondições do evento do tipo *attempt*(*OP*, _) são satisfeitas, o evento é incorporado normalmente. Quando, dentre os eventos sem predecessores a serem selecionados para o cumprimento do HTN, só há eventos do tipo *attempt*(*OP*, _) com precondições que não são satisfeitas pelo estado corrente, esses eventos são descartados, mas considera-se que a parte da rede de tarefas correspondente a eles foi cumprida.

# 4 Implementação

Este capítulo descreve os principais aspectos relativos à implementação do modelo proposto no capítulo 3. Com base na versão atual do Logtell, foi desenvolvido um protótipo capaz de gerar enredos com eventos não-determinísticos. Esse protótipo também é capaz de, em conformidade com as ideias expostas na apresentação da proposta de modelo, lidar com o planejamento em paralelo a partir dos diferentes estados finais alcançáveis por um plano não-determinístico. Apesar de o IPG ter sido desenvolvido utilizando o implementador SICStus Prolog, optou-se, a princípio, pelo uso do implementador SWI-Prolog para o módulo responsável pelo não-determinismo, pois este apresenta recursos nativos que facilitam o uso de *threads* (para o processamento em paralelo dos múltiplos estados). Entretanto, a constatação, durante os testes iniciais, da existência de inconsistências no tratamento de *constraints* fez com que fosse adotado o SICStus. Com isso, o processamento em paralelo está sendo simulado através do tratamento sequencial dos vários estados finais gerados.

O protótipo não contempla as alterações no *Simulation Controller* e na interface com o usuário. Assim, ainda não é possível ver a dramatização dos eventos sob essa nova abordagem, nem receber intervenções do usuário. Para validar o atendimento aos requisitos apresentados no capítulo anterior e testar a forma como o sistema reage às intervenções do usuário, abstraiu-se a existência desses módulos externos ao planejamento; as intervenções do usuário foram simuladas diretamente através de código de programação, de modo a permitir testar as reações do sistema a esses eventos. No decorrer deste capítulo, serão apresentados os novos módulos incorporados à arquitetura do Logtell. Primeiro, o *Chapter Simulator*, responsável

pela geração dos planos não-determinísticos, Em seguida, o *Chapter Controller*, cujo objetivo é controlar as alternativas de continuidade do capítulo corrente.

## 4.1 Chapter Simulator

O *Chapter Simulator* é um novo módulo incorporado ao Logtell, responsável por gerar os planos a serem utilizados no processo de dramatização. Tal módulo integra o planejador de ordem parcial IPG ao novo planejador hierárquico não-determinístico, denominado NDet-HTN. Esta seção apresenta o algoritmo utilizado para a implementação do NDet-HTN e, particularmente, a solução adotada para a representação dos estados durante o planejamento. Também são apresentadas as modificações que se fizeram necessárias no planejamento de ordem parcial.

### 4.1.1 Algoritmo de Planejamento Hierárquico Não-Determinístico

A Figura 4.1 mostra o algoritmo utilizado pelo planejador HTN não-determinístico incorporado ao Logtell, que produz como resultado um plano que cumpre uma rede de tarefas. Apesar de inspirado no algoritmo de planejamento PFD (principalmente no tratamento dado à ordem em que as tarefas são selecionadas, conforme as ordenações parciais definidas), ele possui uma lógica própria, principalmente para permitir o tratamento do não-determinismo. Os dados de entrada são:

- *estado_inicial*: conjunto de fatos positivos que descrevem o estado a partir do qual o enredo deverá ser gerado;
- *estado_atual*: estrutura que contém os conjuntos $fatos^+$ e $fatos^-$ de fatos adicionados e removidos em relação ao estado inicial, respectivamente;
- *w*: uma rede de tarefas, na qual cada vértice *u* é associado a uma tarefa $t_u$, da mesma forma que no algoritmo PFD (apresentado no capítulo 2);
- *O*: corresponde ao conjunto de operadores do domínio, incluindo suas definições de especialização e composição;
- *grau_ndet*: número inteiro positivo que corresponde ao grau máximo de não-determinismo.

São retornados os seguintes valores:

- $\pi$: plano gerado pelo algoritmo;
- *estados_finais*: estados finais alcançáveis pelo plano, considerando todas as ramificações de $\pi$.

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*, $\sigma$) | *a* é instância "ground" de um operador em *O*,

$\sigma$ 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*, $\sigma$) ∈ *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*

$\pi'$ ← 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

$w' \leftarrow \sigma(w - \{u\})$ *// 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*

$\pi''$ ← NDet-HTN(*estado_inicial, estado_det, w', O, grau_ndet, estados_finais*)

$\pi' \leftarrow$ [*VERDADE*, $\pi''$]

Senão

$\pi' \leftarrow \emptyset$

Para cada efeito não-determinístico *efeito_ndet* de *a*

*estado_ndet* ← aplicação de *efeito_ndet* sobre *estado_det*

$\pi''$ ← NDet-HTN(*estado_inicial, estado_ndet, w', O, grau_ndet, estados_finais*)

Se $\pi'' = FRACASSO$ então retorne *FRACASSO*

Senão $\pi' \leftarrow \pi' \cup \{[efeito\_ndet, \pi'']\}$

Fim-para

*estados_finais* ← {estados finais alcançáveis por $\pi'$}

Se tamanho(*estados_finais*) > *grau_ndet* então

Retorne *FRACASSO*

Senão *// Se w' = HTN_VAZIO*

*// Incorporará subplano vazio ao plano, fechando aquele ramo*

$\pi' \leftarrow \emptyset$

Se $t'_u$ não tem efeitos não-determinísticos então

*estados_finais* ← {*estado_det*}

Senão

*estados_finais* ← Ø

Para cada efeito não-determinístico *efeito_ndet* de *a*

*estados_finais* ← *estados_finais* ∪ {*efeito_ndet*}

Fim-para

Se tamanho(*estados_finais*) > *grau_ndet* então retorne *FRACASSO*

Senão se $t'_u$ é um operador genérico então *// Substitui por especialização*

*especializações* ← {*e* | *e* é instância "ground" (usando σ) de uma especialização de *a* em *O*, e *e* é aplicável a *estado_atual*}

Escolha não-deterministicamente algum *e* ∈ *especializações*

*w'* ← substituição de *u* por *e* em *σ*(*w*)

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

Senão se $t'_u$ é um operador composto então *// Substitui por suboperadores*

*w'*← δ(*w*, *u*, *a*, *σ*) *// Nova rede de tarefas criada pela substituição de u pelos suboperadores de a (usando a substituição σ)*

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

Senão retorne *FRACASSO*

Se *π' = FRACASSO* então retorne *FRACASSO*

*π* ← [*a*, *π'*]

Retorne *π*

Fim

Figura 4.1: Algoritmo NDet-HTN para planejamento hierárquico não-determinístico

Para a execução do processo de planejamento, deve ser feita a seguinte chamada inicial ao planejador:

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

Para representar os planos não-determinísticos que passam a ser tratados pelo Logtell, tornou-se necessário o estabelecimento de uma estrutura de dados

adequada. A solução adotada foi a geração de planos sob a forma de uma árvore de decisão, em que cada nó corresponde a uma ação e cujos ramos correspondem à execução a ser selecionada para cada conjunto de efeitos não-determinísticos da ação. Ações determinísticas possuem apenas um ramo e levam a outro único nó (associado a outra ação); caso haja efeitos não-determinísticos, os ramos correspondentes levam a novos nós que corresponderão às ações a serem executadas de acordo com cada condição (efeito não-determinístico) assumida. O planejador sempre adota uma ordenação total possível de acordo com a ordenação parcial da rede de tarefas, e as ações sem sucessores no plano final são associadas a nós terminais, ou seja, aqueles que não possuem ramos.

Uma das necessidades deste novo planejador é tratar adequadamente as atualizações de estado decorrentes da aplicação dos efeitos de um operador instanciado em ação, de forma a tratar seus possíveis efeitos não-determinísticos. Assim, cada vez que um operador básico é selecionado e instanciado, o planejador não-determinístico atualiza o estado atual, inicialmente, apenas com seus efeitos determinísticos e, a partir deste novo estado atual, cria uma nova ramificação para cada um dos seus efeitos não-determinísticos. Cada um desses ramos tem seu próprio estado atual atualizado com seus respectivos efeitos não-determinísticos, produzindo novos estados atuais que serão usados para planejar para a frente usando a rede de tarefas restante (sem o operador básico cujos efeitos não-determinísticos geraram a ramificação no plano). Assim, cada vez que um operador não-determinístico é adicionado ao plano, o planejador acrescenta novas ramificações, cada uma com sua própria sequência de ações e de estados. Quanto maior for o número de ações não-determinísticas, maior será o número de estados finais gerados pelo plano.

Os conceitos de subtarefas e métodos HTN foram implementados usando uma estrutura de herança e a composição dos operadores. Além dos operadores do domínio (que geram os efeitos no estado atual e são instanciados diretamente em ações), a especificação do domínio inclui também o conceito de operadores complexos. Um operador dito complexo pode ser uma generalização de um ou mais operadores específicos e/ou uma composição de suboperadores parcialmente ordenados. Especializações relacionam um operador pai a um ou mais operadores

filhos; se o operador pai é composto, seus filhos também herdarão sua composição (suboperadores e restrições de ordem). Conforme o modelo proposto, operadores filhos podem também adicionar novas restrições de ordenação e até mesmo incluir novos operadores entre os suboperadores herdados.

O algoritmo trata tais casos da seguinte forma: primeiramente, ele analisa se o operador é básico; se for, ele realiza diretamente as modificações necessárias no estado atual, no plano e na rede de tarefas. Caso seja complexo, ele vai primeiro verificar se corresponde a uma generalização, buscando então uma especialização e fazendo as modificações necessárias na rede de tarefas para substituir o operador genérico pela especialização selecionada. Por fim, caso não seja uma generalização, ele deve ser uma composição, e a rede de tarefas é então alterada de forma a, analogamente ao que é feito no PFD quando uma tarefa é substituída por suas subtarefas, substituir o operador por seus suboperadores na rede de tarefas. Em todos os casos, o planejador é então chamado recursivamente, com sua rede de tarefas atualizada.

Durante o processo de planejamento, operadores complexos são substituídos por especializações ou suboperadores, os quais podem ser não apenas operadores do domínio, mas também novas generalizações ou composições – causando, dessa forma, herança e composição em cascata. Os operadores complexos que são generalizações ou composições devem ser recursivamente especializados ou decompostos até que operadores do domínio sejam alcançados e incorporados ao plano. Quando um operador genérico tem mais de uma especialização, suas precondições são avaliadas pelo planejador para que uma delas seja escolhida, da mesma forma como um método HTN é escolhido para realizar uma tarefa.

Outro conceito importante tratado pelo algoritmo é o de tentativas. Operadores do tipo *attempt*($OP(a_1, a_2, ..., a_n)$, $N$), para $N > 0$, equivalem a uma tentativa de incorporar o operador $OP(a_1, a_2, ..., a_n)$ ao plano. Caso as precondições do operador sejam atendidas, o algoritmo prossegue normalmente tratando-o sem considerar o modificador *attempt*. Caso as precondições não sejam atendidas e as restrições parciais de ordenação não permitam que ocorra algum outro evento antes, o operador simplesmente é descartado e removido da rede de tarefas como se tivesse sido cumprido normalmente. O valor de $N$, herdado do planejador de ordem parcial

e representando o nível de tolerância a falha para a ocorrência do evento associado ao operador $OP(a_1, a_2, ..., a_n)$, é ignorado no processo de planejamento hierárquico não-determinístico, a não ser no caso especial em que $N = 0$, quando equivale a um evento que deve, forçosamente, ocorrer, ou seja, $attempt(OP(a_1, a_2, ..., a_n)) = OP(a_1, a_2, ..., a_n)$.

### 4.1.2 Representação dos Estados

Durante um processo de planejamento não-determinístico é preciso raciocinar sobre vários estados que podem ser alcançados ao longo da execução. Cada ação incorporada ao plano provoca uma modificação no estado atual; essas alterações devem ser constantemente reconhecidas pelo planejador, para permitir que as ações corretas possam ser selecionadas. Para evitar-se a necessidade de "materialização" do estado atual a cada ação incorporada – ou seja, a descrição completa do novo estado, contemplando os efeitos causados pela ação –, adotou-se como estratégia a descrição de estados por meio da indicação das alterações em relação ao estado inicial. O objetivo principal dessa escolha é evitar possíveis degradações de desempenho causadas pelo trânsito em memória de uma estrutura de dados que contenha todos os literais que descrevam o estado em um determinado instante.

Os efeitos estabelecidos pelos operadores (e, consequentemente, pelas ações que os instanciam) podem contemplar *efeitos positivos* (que indicam fatos que devem ser verdadeiros no novo estado) e *efeitos negativos* (que indicam fatos que devem ser falsos no novo estado). As alterações em relação ao estado inicial são mantidas em dois conjuntos: um de fatos acrescentados ($fatos^+$) e outro de fatos removidos ($fatos^-$) em relação ao estado inicial. Cada nova ação incluída no plano tem seus efeitos (já devidamente instanciados) comparados a esses conjuntos e ao estado inicial.

A lógica por trás das atualizações desses dois conjuntos faz com que o conjunto $fatos^+$ possa conter apenas fatos que não façam parte do estado inicial, e que o conjunto $fatos^-$ possa conter apenas fatos que façam parte do estado inicial. Dessa forma, efeitos positivos podem apenas ter seus fatos adicionados a $fatos^+$ ou removidos de $fatos^-$, assim como efeitos negativos têm seus fatos apenas adicionados a $fatos^-$ ou removidos de $fatos^+$. Tanto o estado inicial quanto os dois conjuntos de fatos são analisados a cada vez que uma nova ação é incorporada ao

plano, de forma a determinar se cada novo efeito aplicado ao estado atual será acrescentado a algum dos dois conjuntos, removido de algum deles ou se nada deverá ser feito. O algoritmo responsável por essa tarefa é mostrado na Figura 4.2 e recebe os seguintes parâmetros:

- *estado_inicial*: conjunto de fatos positivos que descrevem o estado a partir do qual o enredo deverá ser gerado;
- *estado_atual*: estrutura que contém os conjuntos $fatos^+$ e $fatos^-$;
- *efeitos*: estrutura que contém um conjunto de efeitos positivos e um conjunto de efeitos negativos.

```
// 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 4.2: Algoritmo para atualização dos estados

### 4.1.3 Inferência de Metas e Planejamento de Ordem Parcial

A fim de permitir a integração do novo planejador ao IPG, este último também teve que ser modificado. O IPG implementa um planejador de ordem parcial, onde a obtenção de novos planos a partir de um plano que não é solução (ou seja, onde não é verdade que todas as precondições de todos os eventos estarão necessariamente satisfeitas no momento da execução) segue o algoritmo da Figura 4.3. O algoritmo recebe como entrada um plano parcial *P* com alguma precondição ainda não estabelecida para algum de seus eventos e gera, se possível, um plano sucessor de *P* que garanta o estabelecimento de tal precondição. Para isso, são utilizados os conceitos de *user*, *establisher* e *clobberer*. Um evento *u* é considerado um *user* de um literal *pre* se *pre* for uma precondição de *u*. Já um evento *est* é dito ser um *establisher* de *pre* para *u* se *est* pode ocorrer antes de *u* e, além disso, possui alguma pós-condição *pos* que possa ser unificada com *pre*. Por outro lado, um evento *clob* é considerado um *clobberer* para o estabelecimento da precondição *pre* do *user u* se *clob* possuir algum efeito que possa ser unificado com a negação de *pre*.

**successor(P,U,PRE,SUCC)**

**Entrada**

**P**: plano anterior com precondição não resolvida.
**U**: operador de P com precondição que não é necessariamente válida.
**PRE**: precondição de **U** a ser tornada válida

**Saída**

**SUCC**: plano sucessor de **P**. Valores diferentes são retornados por "backtracking"

1. Definir candidatos **CAND** a sucessores do plano **P** nos quais existe um *establisher* **EST** de **PRE** para **U**.
   1.1. Examinar eventos **EST** em **P** que podem ocorrer antes de **U** e que possuem

uma pós-condição **POS** que pode ser unificada com **PRE**. Definir **CAND** como a extensão de **P** onde **EST** estabelece **PRE** para **U**.

1.2. Examinar novos eventos **EST** na biblioteca que podem ser inseridos para estabelecer **PRE** obrigatoriamente antes de **U**. Definir **CAND** como a extensão de **P** onde **EST** estabelece **PRE** para **U**.

2. Para cada par <**EST**,**CAND**>:

2.1. Obter a lista de eventos (*clobberers*) **LCLOB** que geram conflito para o estabelecimento da precondição **PRE** do *user* **U** pelo *establisher* **EST**.

2.2. Obter todas as combinações possíveis de conjunto de restrições a serem adicionadas a **CAND** para que nenhum *clobberer* esteja em conflito com o estabelecimento de **PRE** por **EST** para **U**. Para cada *clobberer* **CLOB** que estabelece uma pós-condição **negation(Q)**, tal que o literal **Q "possibly_codesignates"** com **PRE**, existem as seguintes alternativas de resolução de conflito:

- Separação de variáveis: fazer com que os argumentos de **Q** e **PRE** sejam distintos. Cada argumento em **Q** e **PRE** especificado por variáveis ou constantes **X** e **Y**, respectivamente, gera uma possibilidade de resolução de conflito adicionando-se a restrição **dif(X,Y)**. Obviamente, isso só funciona com argumentos onde **X** não é igual a **Y** (mesma variável ou mesma constante).
- Forçar que **CLOB** ocorra antes de **EST**. Isso só é possível se já não estiver especificado que a etiqueta de tempo de **EST** precede a de **CLOB**.
- Forçar que **CLOB** ocorra após **U**. Isso só é possível se já não estiver especificado que a etiqueta de tempo de **CLOB** precede a de **U**.

2.3. Para cada conjunto de restrições, gerar um sucessor **SUCC** a ser retornado por "backtracking".

Figura 4.3: Algoritmo *successor* do IPG (CIARLINI, 1999)

As alterações introduzidas no algoritmo *successor* têm como objetivo possibilitar o tratamento adequado de "tentativas" e a limitação do número de eventos no plano. Essas modificações estão listadas a seguir:

- Passa a ser permitido o uso de precondições do tipo *tried*(*COND*, *N*), onde *COND* é um literal e *N* um inteiro (o limite de tolerância para a tentativa). Uma precondição da forma *tried*(*COND*, 0) é equivalente a uma precondição *COND* que deve obrigatoriamente ser estabelecida;
- Eventos podem ter a forma *attempt*(*OP*, *N*), indicando que o evento é uma tentativa de execução de *OP* que aceita que eventos até o nível *N* de antecedência de *OP* na cadeia de precondições falhe. Operadores anteriores a esses devem necessariamente ocorrer. Um evento do tipo *attempt*(*OP*, 0) é equivalente a um evento *OP* de ocorrência obrigatória, ou seja, não é uma tentativa;
- A indicação de quem pode estabelecer uma precondição (item 1.1) foi alterada, levando em conta que uma precondição pode ser do tipo *tried*(*COND*, *N*):
    - Se *PRE* não é do tipo *tried*(*COND*, *N*), considera-se o estabelecimento de *PRE* levando em conta apenas os efeitos determinísticos dos eventos que já estão no plano;
    - Se *PRE* é do tipo *tried*(*COND*, *N*), com $N = 0$, trata-se o estabelecimento de *PRE* como o estabelecimento de *COND*, da mesma forma que no item anterior;
    - Se *PRE* é do tipo *tried*(*COND*, *N*), com $N > 0$, considera-se o estabelecimento de *PRE* através de:
        - Efeitos determinísticos de eventos que não são tentativas e que estabelecem *COND*;
        - Efeitos determinísticos de eventos que não são tentativas e que estabelecem explicitamente *tried*(*COND*, *N1*), com $N1 <= N$;
        - Efeitos não determinísticos de eventos que não são tentativas, se $N = 1$;
        - Efeitos determinísticos ou não-determinísticos de eventos do tipo *attempt*(*OP*, *N1*) que estabelecem *COND*, com $N1 < N$;

        - Efeitos determinísticos de eventos do tipo *attempt*(*OP*, *N1*) que estabelecem explicitamente *tried*(*COND*, *M*), com $M + N1 < N$.
- A indicação dos novos eventos que podem ser inseridos no plano para estabelecer precondições (item 1.2) também foi alterada, da seguinte forma:
    - Para precondições que não são do tipo *tried*(*COND*, *N*), considera-se apenas a inclusão de eventos onde *PRE* pode ser estabelecida pelos efeitos determinísticos. Novo evento *EST* é uma instância de *OP*;
    - Para precondições que são do tipo *tried*(*COND*, 1), considera-se a inclusão de eventos onde *COND* pode ser estabelecida pelos efeitos determinísticos ou não-determinísticos. Novo evento *EST* é uma instância de *OP;*
    - Para precondições que são do tipo *tried*(*COND*, *N*), com $N > 1$, considera-se a inclusão de eventos onde *COND* pode ser estabelecida pelos efeitos determinísticos ou não-determinísticos de um operador *OP*. Novo evento *EST* é uma instância de *attempt*(*OP*, *N1*), com $N1 = N - 1$;
    - A inclusão de novos eventos é condicionada ao fato do plano corrente não exceder o limite para a introdução de novos eventos.
- O item 2.1 também foi modificado para considerar bloqueios, tendo em vista a introdução do conceito de tentativas:
    - É preciso considerar todos os efeitos determinísticos ou não-determinísticos que podem bloquear o estabelecimento da precondição. Examinam-se as listas de efeitos determinísticos e não-determinísticos de todos os eventos, sejam eles tentativas ou de ocorrência obrigatória. Efeitos definidos explicitamente como *tried*(¬*COND*, *N*) – ou seja, tentativas de se estabelecer a negação de *COND*, seja qual for o limite de tolerância – devem também ser considerados para o caso de bloqueio do estabelecimento de *COND;*
    - Caso *PRE* seja do tipo *tried*(*COND*, *N*), analisa-se o bloqueio no estabelecimento de *COND*.

Adicionalmente, o teste da validade da precondição de um evento no momento de sua execução passa a considerar que:

- Há um evento que necessariamente estabelece a precondição, de acordo com o item 1.1, mas exigindo que *EST* esteja antes de *U* e o efeito de *EST* já esteja instanciado com a precondição *PRE*;
- Não há outro evento que pode bloquear o estabelecimento da condição, de acordo com as definições em 2.1.

## 4.2 Chapter Controller

De acordo com o modelo proposto, cada vez que um plano parcial é concluído pelo planejador, ele é passado para o *Simulation Controller* para que os processos de interação do usuário e dramatização possam ser executados. Em seguida, cada um dos possíveis vários estados finais fará com que uma instância específica do módulo NDet-IPG seja criada, sendo usado como estado inicial para os processos de inferência de metas, planejamento de ordem parcial (que irá produzir a rede de tarefas inicial para cada instância, respectivamente) e planejamento não determinístico.

A princípio, para permitir o planejamento em paralelo a partir de cada estado final possível, foi utilizado o implementador SWI-Prolog e seu mecanismo de *threads*. Havia uma *thread* para cada instância do *Chapter Simulator* (realizando a geração dos planos em paralelo), além de uma *thread* para o *Chapter Controller* (que iniciava as outras *threads* e recebia suas respostas). A comunicação entre as *threads* era realizada por meio do mecanismo de filas de mensagem que também faz parte do SWI-Prolog. Entretanto, a detecção de inconsistências na forma como o SWI-Prolog trata *constraints* tornou necessária uma abordagem alternativa. A escolha natural foi a utilização do *SICStus Prolog* por já ter sido utilizado para o desenvolvimento do IPG.

Como o SICStus não apresenta recursos nativos para o tratamento de *threads*, foi necessário adotar uma alternativa. A solução adotada foi fazer com que o *Chapter Controller* receba uma lista de combinações de estados iniciais e listas de tarefas, e que, de posse desses argumentos de entrada, faça chamadas sequenciais ao NDet-HTN, passando, em cada uma delas, uma dessas combinações como argumento de entrada. O módulo, ao fim, associa um ID a cada estado final gerado e apresenta os

planos gerados, assim como os estados finais alcançados (com seus respectivos IDs), de forma a permitir a escolha de algum deles através de uma entrada do usuário.

# 5 Aplicação do Protótipo

Para avaliar a implementação do modelo proposto, foram realizados testes baseados no mesmo contexto do subgênero dos contos de fada que já foi utilizado em outros trabalhos envolvendo o Logtell (CIARLINI *et al*., 2009). Entretanto, para permitir a verificação do atendimento aos novos requisitos apresentados no capítulo 3, algumas alterações se fizeram necessárias nas definições dos eventos que servem de base para o processo de planejamento. Neste capítulo, será apresentada a modelagem de um exemplo para avaliar o atendimento aos requisitos listados. Na seção 5.2, é feita uma avaliação do planejamento HTN não-determinístico implementado, destacando-se a análise do suporte a eventos não-determinísticos e do controle do nível de não-determinismo. Na seção 5.3, é avaliada a incorporação do conceito de tentativas, tanto no que diz respeito ao seu tratamento pelo planejamento de ordem parcial, quanto pelo seu detalhamento através do planejamento não-determinístico HTN. Por fim, na seção 5.4, os aspectos de eficiência da abordagem envolvendo a integração dos dois planejadores são avaliados, abordando inclusive o processamento em paralelo de diversas alternativas de enredo.

## 5.1 Descrição do Contexto Exemplo

O contexto para a geração de histórias é formado pela especificação dos predicados que permitem a descrição dos estados do "mundo", dos operadores básicos que modificam os estados, de um conjunto de regras de inferência de objetivos e de operadores genéricos e compostos, que permitem o uso de

planejamento HTN. Adicionalmente, o contexto é complementado pela especificação do estado inicial das histórias, a partir do qual elas são criadas.

### 5.1.1 Descrição de Estados

O contexto originalmente usado pelo Logtell visa a criação de histórias do gênero Espadas & Dragões, apresentando princesas, cavaleiros, dragões e magos, desempenhando papéis como vítimas, heróis e vilões. Esses personagens podem realizar ações tais como atacar, raptar, lutar, matar e casar-se, alterando, assim, o estado da história.

As configurações iniciais, assim como os demais estados gerados pelo desenrolar do enredo, são representadas por conjuntos de fatos que descrevem a situação de personagens e lugares, além de seus relacionamentos. Tais fatos são expressos através de cláusulas em Prolog; por exemplo, para indicar que *Marian se encontra no White Palace*, usa-se a cláusula *current_place('Marian', 'White_Palace')*. Os fatos também são usados para indicar as diferentes classes de personagens, utilizando herança para indicar que, por exemplo, tanto dragões quanto pessoas são criaturas, e tanto cavaleiros quanto princesas são pessoas. A Figura 5.1 mostra um diagrama de Entidade-Relacionamento com o esquema representando o mini-mundo onde se desenrolam as tramas desse exemplo.

Os eventos gerados através da execução dos operadores do planejador podem, conforme a história se desenvolve, modificar a configuração inicial através da exclusão de fatos que deixaram de ser válidos e da adição de novos fatos. A Tabela 5.1, baseada em (CAMANHO, 2009) (mas com o acréscimo de novos predicados), apresenta os predicados Prolog usados para representar as situações no contexto usado como exemplo.

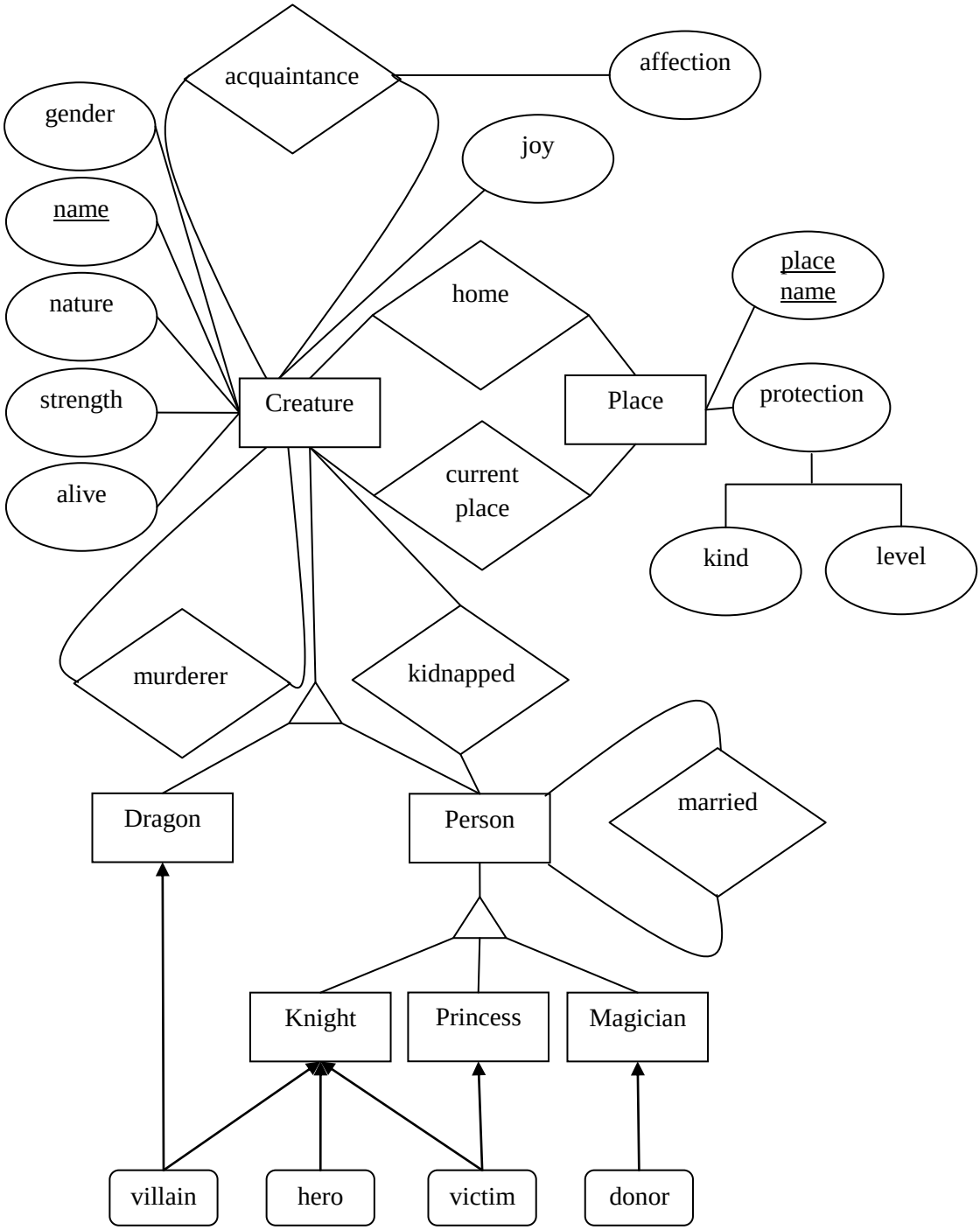


Figura 5.1: Diagrama de Entidade-Relacionamento do Contexto Exemplo

Na Figura 5.1, os retângulos de contornos arredondados que contêm os nomes *villain*, *hero*, *victim* e *donor* representam os papéis que podem ser exercidos pelos tipos de personagens aos quais estão ligados. Tais papéis correspondem a predicados unários na Tabela 5.1. Os demais predicados unários correspondem às classes de

entidades: *creature*, *dragon*, *person*, *knight*, *princess*, *magician* e *place.* Já os relacionamentos *acquaintance*, *current_place*, *home*, *kidnapped*, *married* e *murderer* correspondem a predicados binários na mesma tabela. Da mesma forma, os atributos *gender*, *nature*, *strength* e *joy* também correspondem a predicados binários da tabela. O predicado *alive*, apesar de receber apenas um parâmetro (o nome do personagem que se afirma estar vivo), pode ser considerado também um predicado binário que associa, implicitamente, o valor *True* ao atributo *alive* do personagem em questão. O atributo *affection*, associado ao relacionamento entre criaturas *acquaintance*, é indicado por um predicado ternário, da mesma forma que o atributo *protection*, que associa dois valores (*kind* e *level*) a uma entidade do tipo *place*.

| **Predicado** | **Descrição** |
|---|---|
| *acquaintance*([*CH1*, *CH2*]) | Indica que o personagem *CH1*conhece o personagem *CH2* |
| *affection*([*CH1*, *CH2*], *L*) | Indica que o nível de afeição do personagem *CH1* pelo personagem *CH2* é *L* |
| *alive*(*CH*) | Indica que o personagem *CH* está vivo |
| *creature*(*CH*) | Indica que o personagem *CH* é uma criatura |
| *current_place*(*CH*, *PL*) | Indica que o local atual do personagem *CH1* é *PL* |
| *donor*(*CH*) | Indica que o personagem *CH* é um doador (de força) |
| *dragon*(*CH*) | Indica que o personagem *CH* é um dragão |
| *gender*(*CH*, *G*) | Indica se o personagem *CH* é do sexo masculino ou feminino, de acordo com o valor de *G* |
| *hero*(*CH*) | Indica que o personagem *CH* é um herói |
| *home*(*CH*, *PL*) | Indica que o local inicial do personagem *CH* é *PL* |
| *joy*(*CH*, *L*) | Indica que o nível de felicidade (ou tristeza, se o valor for negativo) do personagem *CH* é *L* |
| *kidnapped*(*CH1*, *CH2*) | Indica que o personagem *CH1* está sendo mantido cativo pelo personagem *CH2* |
| *knight*(*CH*) | Indica que o personagem *CH* é um cavaleiro |
| *magician*(*CH*) | Indica que o personagem *CH* é um mago |
| *married*(*CH1*, *CH2*) | Indica que o personagem *CH 1*é casado com o personagem *CH2* |
| *murderer*(*CH1*, *CH2*) | Indica que o personagem *CH 1*matou o personagem *CH2* |

| | |
|---|---|
| *nature*(*CH*, *K*) | Indica a natureza do personagem *CH* (se é bom, mau, ou neutro), de acordo com o valor de *K* |
| *person*(*CH*) | Indica que o personagem *CH* é uma pessoa |
| *place*(*PL*) | Indica a existência de um local *PL* |
| *princess*(*CH*) | Indica que o personagem *CH* é uma princesa |
| *protection*(*CH*, [*K*, *L*]) | Indica que a natureza e o nível de proteção do local *PL* são, respectivamente, *K* e *L* |
| *strength*(*CH*, *L*) | Indica que a força do personagem *CH* é *L* |
| *victim*(*CH*) | Indica que o personagem *CH* é uma vítima |
| *villain*(*CH*) | Indica que o personagem *CH* é um vilão |

Tabela 5.1: Predicados usados no contexto exemplo do Logtell e suas descrições

### 5.1.2 Operadores Básicos

Os eventos inseridos na história são instâncias dos operadores definidos para o contexto; tais operadores são padrões de eventos típicos do gênero (contendo variáveis). Pré- e pós-condições são especificadas usando os predicados de descrição de situações do contexto. A Tabela 5.2 apresenta uma lista dos possíveis eventos determinísticos que podem ocorrer no contexto exemplo.

| **Evento** | **Descrição** |
|---|---|
| *attack*(*CH, PL*) | Indica que o personagem *CH* ataca as defesas do local *PL* |
| *bewitch*(*CH1, CH2*) | Indica que o personagem *CH1* enfeitiça o personagem *CH2* |
| *commit_suicide*(*CH*) | Indica que o personagem *CH* se suicida |
| *do_evil_laugh*(*CH*) | Indica que o personagem *CH* emite uma risada diabólica |
| *donate*(*CH1, CH2*) | Indica que o personagem *CH1* aumenta a força do personagem *CH2* |
| *fight*(*CH1, CH2*) | Indica que o personagem *CH1* ataca o personagem *CH2* |
| *free*(*CH1, CH2*) | Indica que o personagem *CH1* liberta o personagem *CH2* |
| *go*(*CH, PL*) | Indica que o personagem *CH* vai ao local *PL* |
| *remove_skin*(*CH1, CH2*) | Indica que o personagem *CH1* esfola (remove a pele) do personagem *CH2* |

| | |
|---|---|
| *join_convent*(*CH*) | Indica que o personagem *CH* entra para um convento |
| *kidnap*(*CH1*, *CH2*) | Indica que o personagem *CH1* sequestra o personagem *CH2* |
| *kill*(*CH1*, *CH2*) | Indica que o personagem *CH1* mata o personagem *CH2* |
| *make_funeral*(*CH1*, *CH2*) | Indica que o personagem *CH1* realiza o funeral do personagem *CH2* |
| *marry*(*CH1*, *CH2*) | Indica que o personagem *CH1* casa-se com o personagem *CH2* |
| *nil* | Indica que nada deve ser feito |
| *reduce_protection*(*CH*, *PL*) | Indica que o personagem *CH* reduz a proteção do local *PL* (ex: dispensa os guardas) |
| *roar*(*CH*) | Indica que o personagem *CH* emite um rugido |
| *spread_horror*(*CH*) | Indica que o personagem *CH* sai disseminando terror pelo território ao seu redor |

Tabela 5.2: Lista de eventos determinísticos possíveis no contexto exemplo

Por exemplo, o predicado *go*(*CH*, *PL*) indica o deslocamento do personagem *CH* para o local *PL*. Para que tal evento possa ocorrer, é preciso que o personagem *CH* esteja vivo e não esteja sendo mantido cativo; além disso, segundo as regras de nosso contexto, *CH* também não pode se deslocar de seu local atual caso seja ele que mantenha alguém cativo – considerando que não seria sensato baixar a guarda sobre sua vítima em um mundo com heróis à solta, ansiosos por praticar um ato de heroísmo. Por fim, para evitar que o planejador avalie eventos desnecessários em que um personagem se desloca para o próprio local em que já se encontra, existe mais uma precondição que exige que o local *PL* seja diferente da localização atual *PL0* de *CH*. Como resultado de tal evento, a localidade atual de *CH* passa a ser *PL*, e não mais *PL0*. Tais pré- e pós-condições podem ser expressas da seguinte forma (sendo *P* o conjunto de pré-condições e *Q* o conjunto de pós-condições):

*go*(*CH*, *PL*) /* *CH é um personagem e PL é um lugar* */*
*P* = {*alive*(*CH*) ∧ /* *CH está vivo* */*
¬*kidnapped*(_, *CH*) ∧ /* *Ninguém está raptado por CH* */*
¬*kidnapped*(*CH*, _) ∧ /* *CH não está raptado por ninguém* */*
*current_place*(*CH*, *PL0*) ∧ /* *A localização atual de CH é PL0* */*

*PL0* ≠ *PL*)} /* *Os locais PL0 e PL são diferentes* */

*Q* = {¬*current_place*(*CH*, *PL0*) ∧ /* *A localização atual de CH não é PL0* */

*current_place*(*CH*, *PL*)} /* *A localização atual de CH é PL* */

Tais cláusulas utilizam a notação de variável anônima do Prolog ("_") quando não estamos interessados no valor que aquela variável possa assumir, ou seja, quando apenas queremos saber se ela pode ser unificada com algum valor, mas não nos interessa o valor em si. No caso acima, queremos saber se CH está raptado por alguém ou se, ao invés disso, é o raptor de alguém, não nos interessando quem possa vir a ser o outro parâmetro do predicado *kidnapped*.

Vale ressaltar o uso do operador *nil*, que indica que nenhuma ação ocorre. Tal operador, ainda que, à primeira vista, pareça desnecessário, serve para manter a consistência da estrutura do plano, sendo usado para indicar situações especiais, tais como uma falha resultante de uma tentativa ou em casos em que, para algum operador genérico, existe algum caso (dependendo do estado encontrado no momento) em que nada realmente deve ser feito (para os demais casos, seriam utilizadas especializações com operadores convencionais). Um exemplo da aplicação de tal recurso, utilizado em nosso contexto, é o operador genérico *perform_unhappy_ending*, que possui como precondição básica a morte do herói que tenta resgatar a vítima e/ou a morte da própria vítima (impossibilitando um final feliz em que os dois se casem). Tal operador possui especializações para cada combinação possível de personagens dentro dessas condições. Por exemplo, caso o herói seja morto pelo vilão, e a vítima esteja viva, ela cometerá suicídio (evento *commit_suicide*); caso o herói mate o vilão, mas a vítima já esteja morta, o evento selecionado será *make_funeral*, indicando que o herói realiza o funeral da vítima. Entretanto, caso tanto o herói quanto o vilão e a vítima estejam mortos no fim da história, nada mais há a ser feito; uma especialização associando tal precondição ao operador *nil* é então usada para permitir que todos os estados possíveis (incluindo aqueles que não conduzem a uma cadeia específica de eventos) tenham uma especialização que possa ser escolhida pelo planejador.

Para permitir uma avaliação do tratamento de efeitos não-determinísticos, foram acrescentados alguns eventos que possuem mais de um conjunto possível de pós-condições. A Tabela 5.3 apresenta esses eventos:

| Evento | Descrição |
|---|---|
| *try_to_kidnap*(*CH1*, *CH2*) | Indica que o personagem *CH1* tenta raptar o personagem *CH2* (uma vítima), podendo resultar em um rapto bem-sucedido ou na morte da vítima |
| *fight_to_death*(*CH1*, *CH2*) | Indica que os personagens *CH1* e *CH2* lutam entre si, podendo resultar na morte de um, de outro, ou de ambos |

Tabela 5.3: Lista de eventos não-determinísticos possíveis no contexto exemplo

Os conjuntos de precondições e pós-condições são definidos de forma análoga ao que é feito para os eventos determinísticos. A diferença é que, no caso da existência de efeitos não-determinísticos, o conjunto de pós-condições do evento é representado por uma conjunção entre seus efeitos determinísticos e uma disjunção entre cada um de seus conjuntos de pós-condições não-determinísticas. Consideremos, por exemplo, o evento *try_to_kidnap*. Suas precondições (representadas pelo conjunto *P*) e pós-condições (representadas pelo conjunto *Q*) são apresentadas logo abaixo:

*try_to_kidnap*(*VIL*, *VIC*) /* *VIL é um vilão e VIC é uma vítima* */
*P* = {*alive*(*VIC*), *alive*(*VIL*) ∧ /* *VIC e VIL estão vivos* */
*nature*(*VIC*, *K1*) ∧ /* *A natureza de VIC é K1* */
¬*kidnapped*(*VIC*, _) ∧ /**VIC não está raptado por ninguém* */
*strength*(*VIC*, *VIC_S*) ∧ /* *A força de VIC é VIC_S* */
*current_place*(*VIC*, *PL*) ∧ /* *A localização atual de VIC é PL* */
*protection*(*PL*, [*K2*, *LP*]) ∧ /* *PL tem proteção do tipo K2 e nível LP* */
*strength*(*VIL*, *VIL_S*) ∧ /* *A força de VIL é VIL_S* */
*current_place*(*VIL*, *PL*) ∧ /* *A localização atual de VIL é PL* */
*PL* ≠ *home*(*VIL*) ∧ /* *PL é diferente do local inicial de VIL* */

*VIL_S* > *VIC_S* + *LP* * *K1* * *K2* ∧ /* *VIL_S é maior que VIC_S+LP*K1*K2* */

*joy*(*VIC*, *VIC_J*) ∧ /* *O nível de alegria de VIC é VIC_J* */

*joy*(*VIL*, *VIL_J*) /* *O nível de alegria de VIL é VIL_J* */

*Q* = {¬*current_place*(*VIC*, *PL*) ∧ /* *A localização atual de VIC não é PL* */

¬*current_place*(*VIL*, *PL*) ∧ /* *A localização atual de VIL não é PL* */

*current_place*(*VIC*, *home*(*VIL*)) ∧ /* *A localização atual de VIC é o local inicial de VIL* */

*current_place*(*VIL*, *home*(*VIL*)) ∧ /* *A localização atual de VIL é o local inicial de VIL* */

¬*joy*(*VIL*, *VIL_J*)) ∧ /* *O nível de felicidade de VIL não é VIL_J* */

*joy*(*VIL*, *VIL_J* + 20) ∧ /* *O nível de felicidade de VIL é VIL_J+20*

/* *Efeitos não-determinísticos* */

((*kidnapped*(*VIC*, *VIL*) ∧ /* *VIC está raptado por VIL* */

¬*joy*(*VIC*, *VIC_J*) ∧ /* *O nível de felicidade de VIC não é VIC_J* */

*joy*(*VIC*, -50.0)) /* *O nível de felicidade de VIC é -50.0* */

∨ /* *Alternativa entre os efeitos* */

(¬*alive*(*VIC*)) ∧ /* *VIC não está vivo* */

*murderer*(*VIL*, *VIC*)))} /* *VIL é o assassino de VIC* */

### 5.1.3 Operadores Genéricos e Compostos

Para implementar o uso de planejamento HTN e permitir que um mesmo evento, genérico, possa ser especializado de diferentes formas (diversificando, dessa forma, a criação dos enredos), a modelagem dos operadores passa agora a usar extensivamente a definição de hierarquias de composição e de especialização de operadores. Tais hierarquias já eram usadas dentro do IPG em uma possibilidade de uso alternativa que previa o reconhecimento de planos típicos a serem adaptados. No entanto, como o uso do IPG no Logtell ficou centrado na geração de enredos, o contexto de exemplo do Logtell continha apenas operadores básicos, diretamente

dramatizáveis. Agora com o uso de métodos baseados em HTN, decidiu-se usar a mesma modelagem de operadores genéricos (que podem ser especializados) e compostos, para representar as tarefas e alternativas de execução delas em um domínio.

Um operador genérico pode ter uma ou mais especializações, e, durante o planejamento, uma delas será selecionada para ser incorporada ao enredo. Ao se solicitar uma alternativa ao planejador, outra especialização pode ser escolhida, aumentando assim a variedade das histórias geradas. Além disso, um operador que especializa outro pode, ao mesmo tempo, ser também uma generalização, sendo ele próprio especializado por um ou mais operadores.

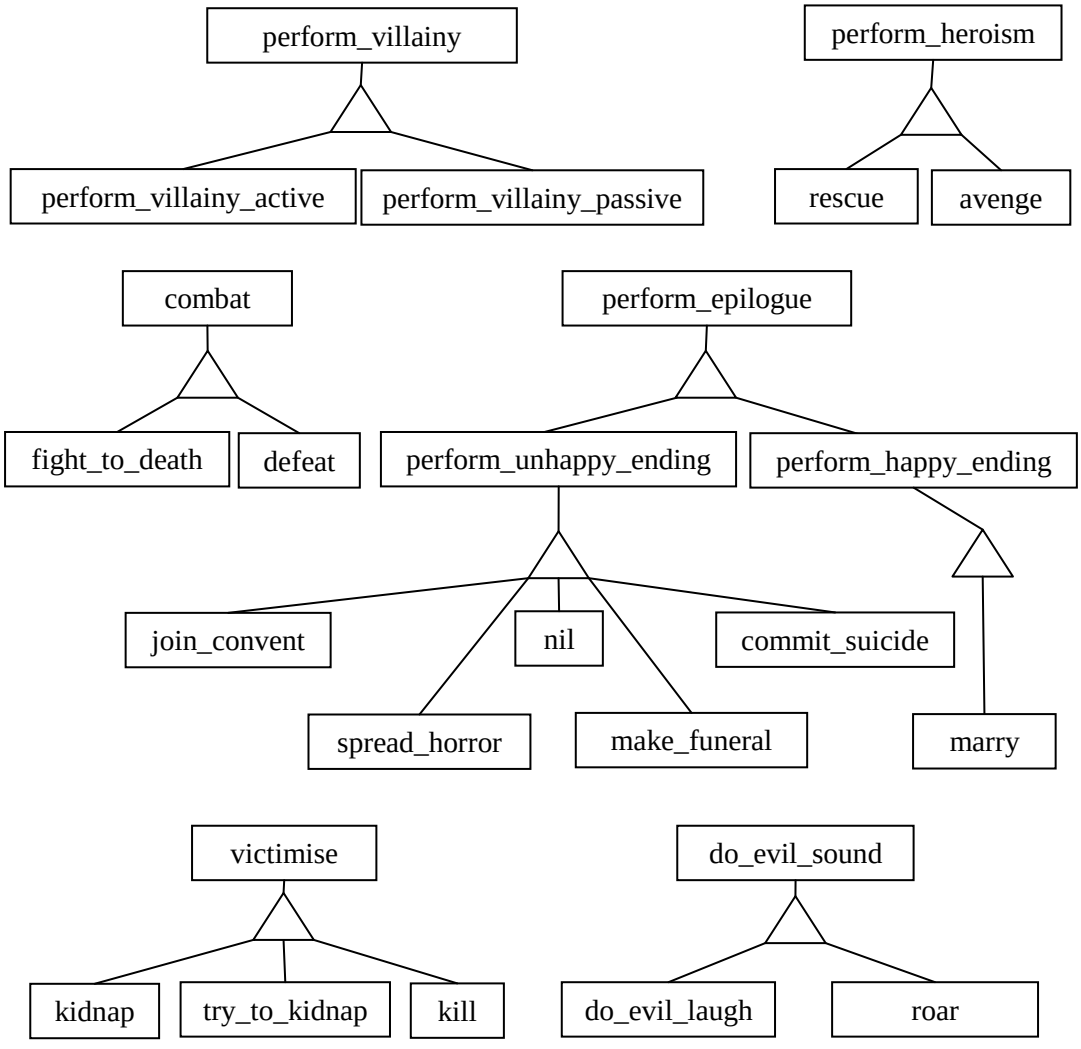


Figura 5.2: Especializações do contexto exemplo

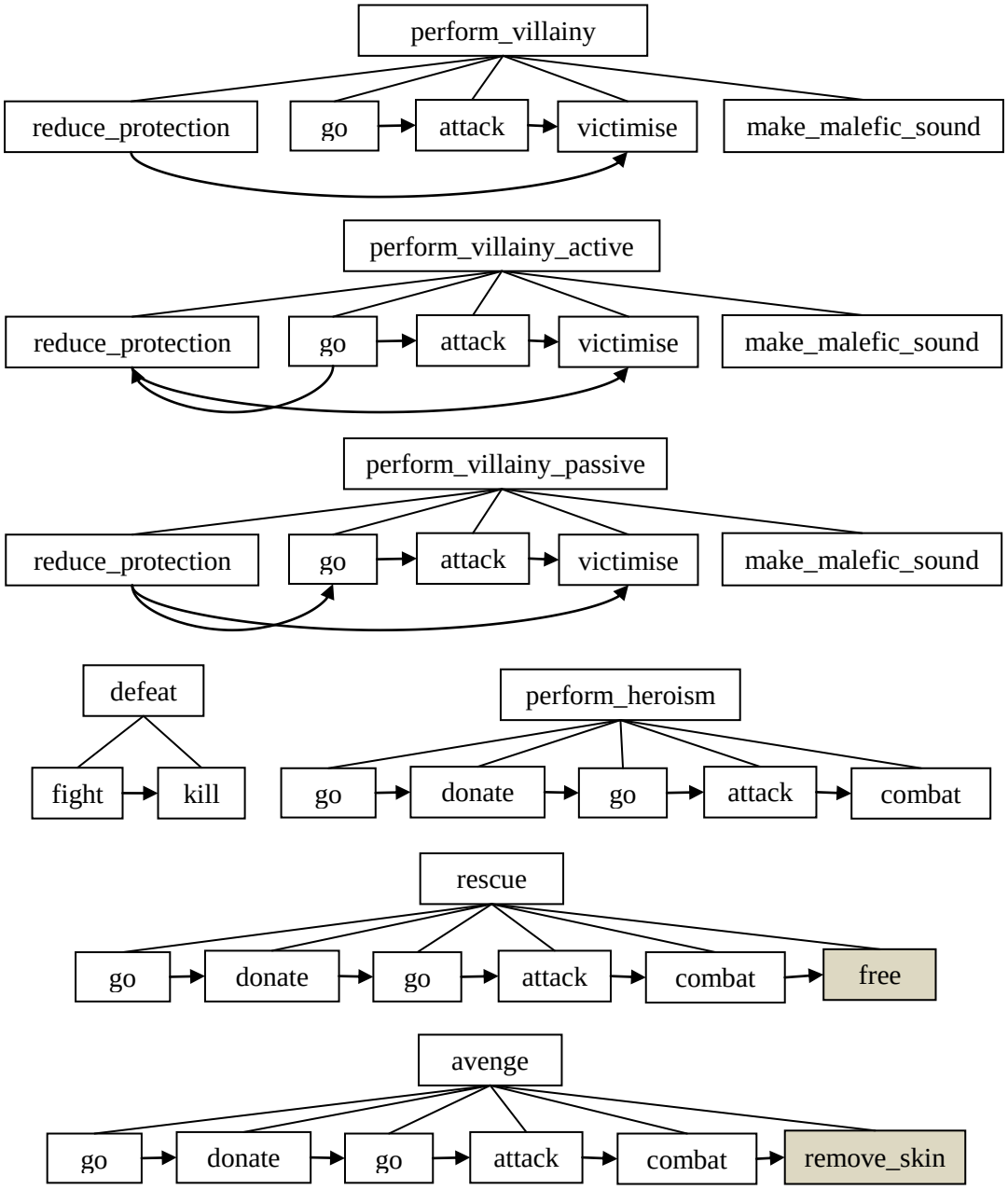


Figura 5.3: Composições do contexto exemplo - as caixas acinzentadas correspondem a tentativas (*attempt*) de executar os operadores nelas contidos

Operadores também podem ser compostos, o que significa que se decompõem em um ou mais suboperadores. Durante o processo de planejamento, um operador composto é substituído por seus suboperadores, que herdam as mesmas restrições de ordem que o operador original possuía. Generalização e composição podem ser

combinados: um operador que especializa outro, composto, herda os suboperadores daquele, podendo incluir novos operadores e/ou restrições de ordem à composição.

De forma a avaliar o uso de generalizações e composições, outros operadores foram incluídos ao cenário original. A Figura 5.2 (generalizações e suas especializações) e a Figura 5.3 (composições e seus suboperadores) mostram as estruturas relativas a esses novos operadores, ditos complexos. Com o objetivo de simplificar as figuras, os argumentos não são mostrados. Pelas figuras podemos ver, por exemplo, que *perform_villainy* pode ser especializado de duas formas distintas: *perform_villainy_active* e *perform_villainy_passive*. Sendo *perform_villainy* composto, ambas as especializações herdam seus suboperadores, incluindo suas restrições de ordem. Entretanto, cada uma possui sua própria restrição de ordem específica, acrescentada àquelas herdadas: enquanto, no caso de *perform_villainy_active*, o evento *go* (que, nesse caso, estabelece a ida do vilão para o local onde se encontra vítima) deve vir antes do evento *reduce_protection* (indicando que o vilão se antecipa a uma ação em que a vítima se desproteja, como se ficasse de tocaia aguardando uma oportunidade), no caso de *perform_villainy_passive* a ordem desses dois eventos é invertida, indicando que o vilão aguarda o surgimento de uma oportunidade para só então se deslocar ao local da vítima. Tais restrições de ordem específicas são possíveis, já que elas não entram em conflito com aquelas que foram herdadas, pois não há nenhuma restrição de ordem entre *go* e *reduce_protection* no operador complexo *perform_villainy*.

### 5.1.4 Regras de Inferência de Objetivos

As regras de inferência de objetivos usadas pelo IPG, especificadas num formalismo de lógica temporal modal (CIARLINI *et al*., 2000), definem os objetivos que os personagens de determinadas classes (herói, vilão, vítima) passam a perseguir quando certas situações se observam. Essas regras usam meta-predicados para falar sobre a ocorrência de um evento em um determinado tempo ou sobre a veracidade de um fato (ou sua negação) em determinado momento. O formalismo incorpora o tratamento das ordens parciais de eventos, uma vez que a veracidade de um fato pode depender ou não da ordem dos eventos (CAMANHO, 2009).

Para o contexto exemplo, foram usadas regras que gerassem operações de alto nível, genéricas, que pudessem ser transformadas em mais de um plano pelo planejador HTN. Uma dessas regras de inferência de objetivos diz que "se existe uma vítima viva, que não está raptada, e há um vilão, este vai realizar uma vilania". Outra regra, quase que complementar, diz que "se um vilão fez algo de mal a uma vítima e há um herói que gosta dela, este vai querer realizar um ato de heroísmo para combater tal vilania". É importante ressaltar que as regras não ditam diretamente as reações dos personagens. As regras apenas indicam objetivos a serem perseguidos. Os eventos que vão conseguir realizar os objetivos são preenchidos pelos algoritmos de planejamento – em um primeiro momento, através das tarefas de alto nível especificadas pelo IPG e, posteriormente, por seu detalhamento através do planejador NDet-HTN.

A Tabela 5.4 a seguir apresenta uma lista com as regras que foram modeladas para o contexto exemplo.

| **Regra** | **Operação Genérica** |
|---|---|
| A existência de uma vítima viva e livre no início da história vai despertar no vilão o desejo de fazer uma vilania contra ela | *perform_villainy* |
| Se o vilão tiver realizado alguma vilania com a vítima, o herói tentará realizar um ato heroico | *perform_heroism* |
| Se o herói é morto pelo vilão, ou vice-versa, ou ambos terminam se matando, os protagonistas que tiverem sobrevivido vão realizar ações de acordo com o que foi especificado para cada uma dessas situações, concluindo a história de forma feliz ou não | *perform_epilogue* |

Tabela 5.4: Lista de regras no exemplo do Logtell

A primeira regra listada acima é formalizada por meio da regra em lógica temporal a seguir, onde o metapredicado *e*(*T*, *LIT*) é usado para especificar que um literal *LIT* é estabelecido no tempo *T*, e o metapredicado *o*(*T*, *EV*), que o evento *EV* ocorre no tempo *T*. A constante *i* é usada para falar do tempo inicial da história. As

variáveis que não aparecem quantificadas são universalmente quantificadas para a fórmula toda.

$$e(i, victim(VIC)) \wedge e(i, villain(VIL)) \wedge e(i, alive(VIC)) \wedge e(i, alive(VIL)) \wedge$$
$$e(i, not(kidnapped(VIC, \_))) \rightarrow \exists T\ o(T, perform\_villainy(VIL,VIC))$$

As regras listadas acima permitem a geração de enredos onde a etapa de planejamento de ordem parcial se restringe à verificação da validade das precondições dos operadores genéricos inferidos (eventualmente inserindo eventos que as tornem válidas). No caso mais geral, usam-se também regras que levam à inferência de situações que devem valer no futuro. Nesse caso, as situações são inseridas no plano como eventos fictícios que têm apenas a situação indicada como precondição e nenhum efeito.

Para testar o caso em que o planejamento de ordem parcial encadeia mais eventos a serem decompostos pelo planejamento HTN, e também a avaliação do conceito de tentativas, um conjunto alternativo de regras de inferência de objetivos é usado, onde as duas primeiras regras são substituídas pelas regras a seguir:

- Se vítima está viva e livre no início da história, então vilão **vai tentar** atingir estado onde ela está raptada por ele.
- Se vítima está raptada e herói tem nível alto de afeição por ela, ele **vai tentar** atingir estado onde ela está livre.

A primeira regra listada acima é formalizada por meio da regra em lógica temporal a seguir, onde o metapredicado $h(T, LIT)$ é usado para especificar que um literal *LIT* é válido no tempo *T*.

$$e(i, victim(VIC)) \wedge e(i, villain(VIL)) \wedge e(i, alive(VIC)) \wedge e(i, alive(VIL)) \wedge$$
$$e(i, not(kidnapped(VIC, \_))) \rightarrow \exists T\ h(T, tried(kidnapped(VIL, VIC), 2))$$

Note-se que no lugar do literal *LIT* aparece a indicação de que terá sido tentado o estabelecimento do fato da vítima estar raptada pelo vilão com nível de tolerância à falha igual a 2, ou seja, pode deixar de ser estabelecida porque o predicado que a

geraria é não-determinístico ou até porque ele é planejado como uma tentativa que não chega a ocorrer por causa de falha em suas precondições.

### 5.1.5 Configuração Inicial do Contexto

Para os testes apresentados neste capítulo, foi preciso definir uma configuração inicial, a partir da qual as histórias são desenvolvidas. Nessa configuração, criou-se um mundo onde existe uma princesa - *Marian*, um feiticeiro - *Turjan*, um dragão - *Draco*, e três cavaleiros - *Brian*, *Hoel* e *Lucius*. *Brian*, *Hoel* e *Marian* são de natureza boa, enquanto que *Turjan* é neutro e *Draco* e *Lucius* são maus. Todos estes personagens estão vivos quando a história se inicia. *Brian*, *Hoel* e *Lucius* são mais fracos do que *Draco*; *Marian* é mais fraca ainda, e *Turjan* é tão forte quanto o dragão. *Brian*, *Hoel* e *Lucius* têm forte afeto por *Marian*, o qual não é retribuído. Além disso, cada um dos personagens tem um papel, informação que é utilizada na geração das histórias: *Brian* e *Hoel* são heróis, *Draco* e *Lucius* são vilões, *Marian* é uma vítima e *Turjan* é um "doador". O doador é um papel especial que, neste gênero, é alguém capaz de doar poder ou então de enfeitiçar outro personagem, trocando sua natureza. A existência de mais de um vilão e de mais de um herói permite, a princípio, uma modelagem de operadores e de regras de inferência que levem a maior interação entre personagens que desempenhem o mesmo papel. Os heróis poderiam, por exemplo, colaborar para libertar a princesa ou, pelo contrário, entrarem em disputa pela primazia da realização do ato heroico. Tais interações não foram tratadas no contexto aqui apresentado. A diversidade de personagens com o mesmo papel serve, neste caso, apenas à diversidade de enredos gerados.

Outros elementos importantes do contexto, além dos personagens e seus papéis e relações, são as localidades. Cada localidade pode estar protegida ou não. Locais protegidos possuem guardas que protegem personagens que são da mesma natureza da proteção exercida no local. Existem sete localidades: os castelos Cinza (*Gray Castle*), Negro (*Black Castle*) e Vermelho (*Red Castle*), o Palácio Branco (*White Palace*), a Floresta Verde (*Green Forest*), a Igreja (*Church*) e o Cemitério (*Cemetery*). O Castelo Cinza é de natureza boa e desprotegido, sendo a residência de *Brian* e *Hoel*. A residência de *Marian*, o Palácio Branco, é um local de natureza boa e está fortemente protegido. *Turjan* vive na Floresta Verde, que é neutra como ele, e

que tem um certo nível de proteção. *Draco* e *Lucius* vivem em suas fortalezas, respectivamente os castelos Vermelho e Negro, ambos também contando com alguma proteção. A Igreja é um local de natureza boa, mas desprotegido. Por fim, o Cemitério é um local neutro e desprotegido. O nível de proteção de um local desempenha papel fundamental na hora da construção da história, dado que os personagens só podem tomar certas ações, tais como sequestrar outro personagem ou lutar com ele, se esse outro personagem estiver em um local suficientemente desprotegido.

## 5.2 Avaliação do Planejamento HTN Não-Determinístico

Para avaliar o planejamento HTN não-determinístico, foram feitas chamadas ao planejador NDet-HTN diretamente passando, como argumentos, o estado inicial (sob a forma de listas de fatos acrescentados e removidos com relação à configuração inicial), a rede de tarefas a ser cumprida e o grau máximo de não-determinismo. Nos testes a seguir, serão mostrados os argumentos passados para o planejador e, em seguida, o resultado gerado pelo planejamento. Um teste bem simples é verificar o comportamento do planejador com relação à especialização do operador *make_malefic_sound*. Isso pode ser feito passando para o NDet-HTN os seguintes argumentos (usando a notação Prolog de colchetes para representar listas):

*Fatos*$^{+}$*:* []
*Fatos*$^{-}$*:* []
*Tarefas:* [(t1, do_malefic_sound(_))]
*Restrições de Ordem:* []
*Grau de Não-Determinismo:* 1

Ou seja, o estado inicial é formado por duas listas vazias, indicando não ter modificações em relação ao contexto original, sendo, portanto, o próprio. Da mesma forma, havendo apenas uma tarefa na lista de tarefas, não há nenhuma restrição de ordem, fazendo com que a lista dessas restrições também seja vazia. Cada tarefa é representada sob a forma (*rótulo*, *nome_do_operador*($arg_1$, ..., $arg_n$)), onde *rótulo* é

utilizado para identificar a tarefa na lista de restrições de ordenação, *nome_do_operador* equivale a algum dos operadores definidos para o contexto e $arg_1$, ..., $arg_n$ são seus argumentos, instanciados ou não. O operador *do_malefic_sound* possui como único argumento o nome do personagem – que, no, caso, deve ser um vilão – que produz o som maléfico. Para permitir que o planejador escolha o vilão, foi usada uma variável anônima do Prolog, pois não estamos interessados no valor com o qual ela será unificada – o vilão escolhido já aparecerá no próprio plano. Esses argumentos produziram, em duas chamadas ao planejador (utilizando o recurso de *backtracking* do Prolog), os seguintes resultados:

```
==========================================================================
PLAN:
==========================================================================

      Initial State
        []
        []

      HTN
        [(t1, do_malefic_sound(Draco))],[]

      Plan
        → roar(Draco)

      Final States
    ------------------------------------------------------------------------------
        FINAL STATE ID: 1

        Facts +
          []
        Facts -
          []
    ------------------------------------------------------------------------------

    true ;

==========================================================================
PLAN:
==========================================================================

      Initial State
        []
        []

      HTN
        [ (t1,do_malefic_sound(Lucius))],[]
```

```
  Plan
    → do_evil_laugh(Lucius)

  Final States
--------------------------------------------------------------------------------
    FINAL STATE ID: 1

    Facts +
      []
    Facts -
      []
--------------------------------------------------------------------------------

true ;
false.
```

A saída do planejador apresenta, antes de cada plano, suas variáveis de entrada: o estado inicial (através das listas de fatos acrescentados e removidos) e a rede de tarefas inicial. Em seguida, é apresentado o plano, que deve ser interpretado da seguinte forma: as setas (→) indicam os eventos a serem executados, em sequência; quando um evento possui efeitos não-determinísticos, é exibido, abaixo dele, um nó para cada um desses efeitos, com a descrição dos fatos que distinguem aquele efeito dos demais, precedido por uma interrogação entre colchetes ([?]). Cada um desses nós corresponde, portanto, a uma escolha, e, abaixo de cada escolha, existe um subplano que possui a mesma estrutura do plano como um todo. Cada caminho percorrido através dessas escolhas, até chegar-se às folhas da árvore correspondente ao plano gerado, corresponde a uma possível execução do plano. Após cada plano, são apresentados todos os estados finais alcançáveis por ele – neste caso, como não há nenhum evento não determinístico, apenas um para cada plano.

Na verdade, as únicas variações possíveis nesse caso foram as encontradas pelo planejador, ou seja, uma especialização de *do_malefic_sound* para cada um dos vilões (*Draco* e *Lucius*). As regras estabelecidas para os operadores *roar* e *do_evil_laugh* exigem que o vilão seja, respectivamente, um dragão (*dragon*) e uma pessoa (*person*); *Draco* atende à primeira condição e, conforme se pode ver na Figura 5.1, *Lucius*, que é um cavaleiro (*knight*), atende à segunda. A saída gerada pelo planejador permite observar que as chamadas causaram a geração de dois planos seguidos, conforme se pode ver no valor *true* que é retornado pelo planejador

a cada execução bem-sucedida. O valor *false* mostrado ao final indica que não foi possível gerar mais nenhum plano.

A seguir são apresentados testes mais específicos que visaram avaliar o funcionamento do suporte a eventos não-determinísticos e o controle do nível de não-determinismo.

### 5.2.1 Suporte a Eventos Não-Determinísticos

Para comprovar o suporte a eventos não-determinísticos, foi feito um teste utilizando um operador não-determinístico, *fight_to_death*:

*Fatos*$^{+}$*:*
[current_place('Marian','Red_Castle'),current_place('Hoel','Red_Castle'),
protection('Red_Castle',[-1.0,0.0]),strength('Hoel',70.0),
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])]
*Tarefas:*
[(t1, fight_to_death ('Hoel','Draco')),(t2, perform_epilogue)]
*Restrições de Ordem:* [(t1,t2)]
*Grau de Não-Determinismo:* 3

Os fatos acrescentados e removidos em relação à configuração inicial do contexto servem para estabelecer as condições necessárias ao evento *fight_to_death*(*Hoel*, *Draco*). O resultado gerado é apresentado abaixo:

```
==========================================================
PLAN:
==========================================================

    Initial State
      [current_place(Marian,Red_Castle),current_place(Hoel,Red_Castle),
      protection(Red_Castle,[-1.0,0.0]),strength(Hoel,70.0),
      affection([Marian,Hoel],100.0),kidnapped(Marian,Draco)]
```

[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
[ (t1,fight_to_death(Hoel,Draco)), (t2,perform_epilogue)],[ (t1,t2)]

Plan
→ fight_to_death(Hoel,Draco)
[?][strength(Hoel,25.0),not(alive(Draco)),murderer(Hoel,Draco)]
→ marry(Hoel,Marian)
[?][strength(Draco,0.0),not(alive(Hoel)),murderer(Draco,Hoel)]
→ commit_suicide(Marian)
[?][not(alive(Hoel)),murderer(Draco,Hoel),not(alive(Draco)),
murderer(Hoel,Draco)]
→ join_convent(Marian)

Final States
---
FINAL STATE ID: 1

Facts +
[protection(White_Palace,[1.0,30.0]),current_place(Marian,Red_Castle),
joy(Draco,20.0), kidnapped(Marian,Draco),joy(Marian,-50.0)]
Facts -
[protection(White_Palace,[1.0,70.0]),current_place(Marian,White_Palace),
joy(Draco,0.0), joy(Marian,50.0)]
---
FINAL STATE ID: 1

Facts +
[protection(Red_Castle,[-1.0,0.0]),affection([Marian,Hoel],100.0),
kidnapped(Marian,Draco),strength(Hoel,25.0),murderer(Hoel,Draco),
current_place(Hoel,Church),current_place(Marian,Church),
married(Hoel,Marian),married(Marian,Hoel),joy(Hoel,100.0),joy(Marian,100.0)
]
Facts -
[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]),strength(Draco,45.0),alive(Draco),joy(Hoel,15.0),joy(Marian,50.0)]
---
FINAL STATE ID: 2

Facts +
[current_place(Marian,Red_Castle),current_place(Hoel,Red_Castle),
protection(Red_Castle,[-1.0,0.0]),affection([Marian,Hoel],100.0),
kidnapped(Marian,Draco),strength(Draco,0.0),murderer(Draco,Hoel),
murderer(Marian,Marian)]
Facts -
[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]),strength(Draco,45.0),alive(Hoel),alive(Marian)]
---

```
FINAL STATE ID: 3

Facts +
    [current_place(Marian,Red_Castle),current_place(Hoel,Red_Castle),
    protection(Red_Castle,[-1.0,0.0]),affection([Marian,Hoel],100.0),
    murderer(Draco,Hoel),murderer(Hoel,Draco)]
Facts -
    [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]),strength(Draco,45.0),alive(Hoel),alive(Draco)]
--------------------------------------------------------------------------------
```

O plano retornado considera os três possíveis efeitos para o evento não-determinístico *fight_to_death*(*Hoel*, *Draco*): um em que morre *Draco*, outro em que morre *Hoel* e, por fim, uma terceira alternativa em que ambos morrem. Caso o herói mate o vilão (e sobreviva), ele libertará a princesa e eles se casarão; senão, caso o dragão permaneça vivo, a princesa, em um ato de desespero, cometerá suicídio. Por fim, caso ambos, *Draco* e *Hoel*, morram na luta, a princesa conseguirá escapar (por não mais estar sob as garras de seu captor); entretanto, devido ao choque causado pela sequência de eventos que culminou na morte de seu heroico pretendente, ela decide tornar-se freira e entra para um convento.

### 5.2.2 Controle do Nível de Não-Determinismo

Uma das formas de validar o tratamento do limite do grau de não-determinismo é através de um operador genérico que tem especializações com números diferentes de efeitos não-determinísticos (incluindo zero, caso haja apenas efeitos determinísticos). Em nosso contexto, temos, como exemplo, o operador genérico *combat*, que possui uma especialização determinística (*defeat*, uma simples composição dos operadores básicos *fight* e *kill*, em que sempre se sabe qual personagem morrerá) e outra não determinística (*fight_to_death*, que indica uma luta até a morte entre dois personagens e possui três resultados possíveis: a morte de um, do outro ou de ambos). Planos que usem o evento *defeat* produzirão menos ramificações do que aqueles que utilizem, em seu lugar, o evento *fight_to_death*. Assim, um mesmo operador genérico (*combat*) pode, então, contribuir para o plano com quantidades variáveis de estados finais, dependendo do valor passado ao planejador como limite para o número de estados finais.

Para demonstrar que o planejador consegue limitar o nível de não-determinismo dentro de um valor especificado, foi realizado um teste utilizando os argumentos a seguir:

*Fatos$^{+}$:*

[current_place('Draco','White_Palace'),protection('White_Palace',[1.0,30.0])]

*Fatos$^{-}$:*

[current_place('Draco','Red_Castle'), protection('White_Palace',[1.0,70.0])]

*Tarefas:*

[(t1, victimise('Draco', 'Marian'))]

*Restrições de Ordem:* []

*Grau de Não-Determinismo:* 2

O resultado gerado pelo planejador, com o uso de *backtracking* para que todos os resultados possíveis fossem encontrados, é exibido a seguir:

```
==========================================================================
PLAN:
==========================================================================

    Initial State
      [current_place(Draco,White_Palace),protection(White_Palace,[1.0,0.0])]
      [current_place(Draco,Red_Castle),protection(White_Palace,[1.0,70.0])]

    HTN
      [ (t1,victimise(Draco,Marian))],[]

    Plan
      → try_to_kidnap(Draco,Marian)

    Final States
  ------------------------------------------------------------------------------
    FINAL STATE ID: 1

    Facts +
        [protection(White_Palace,[1.0,0.0]),        current_place(Marian,Red_Castle),
        joy(Draco,20.0), kidnapped(Marian,Draco),joy(Marian,-50.0)]
    Facts -
        [protection(White_Palace,[1.0,70.0]),     current_place(Marian,White_Palace),
        joy(Draco,0.0), joy(Marian,50.0)]
  ------------------------------------------------------------------------------
```

FINAL STATE ID: 2

Facts +
[protection(White_Palace,[1.0,0.0]), current_place(Marian,Red_Castle), joy(Draco,20.0), murderer(Draco,Marian)]
Facts -
[protection(White_Palace,[1.0,70.0]), current_place(Marian,White_Palace), joy(Draco,0.0), alive(Marian)]

---

true ;

==========

PLAN:

==========

Initial State
[current_place(Draco,White_Palace),protection(White_Palace,[1.0,0.0])]
[current_place(Draco,Red_Castle),protection(White_Palace,[1.0,70.0])]

HTN
[ (t1,victimise(Draco,Marian))],[]

Plan
→ kidnap(Draco,Marian)

Final States

---

FINAL STATE ID: 1

Facts +
[protection(White_Palace,[1.0,0.0]),kidnapped(Marian,Draco), current_place(Marian,Red_Castle), joy(Marian,-50.0), joy(Draco,20.0)]
Facts -
[protection(White_Palace,[1.0,70.0]),current_place(Marian,White_Palace), joy(Marian,50.0), joy(Draco,0.0)]

---

true ;

==========

PLAN:

==========

Initial State
[current_place(Draco,White_Palace),protection(White_Palace,[1.0,0.0])]
[current_place(Draco,Red_Castle),protection(White_Palace,[1.0,70.0])]

HTN
[ (t1,victimise(Draco,Marian))],[]

Plan

```
   → kill(Draco,Marian)

  Final States
--------------------------------------------------------------------------------
    FINAL STATE ID: 1

    Facts +
        [current_place(Draco,White_Palace),protection(White_Palace,[1.0,0.0]),
        murderer(Draco,Marian)]
    Facts -
        [current_place(Draco,Red_Castle),protection(White_Palace,[1.0,70.0]),
        alive(Marian)]

--------------------------------------------------------------------------------

true ;
false.
```

Mais uma vez, foi utilizada uma rede de tarefas simples, composta apenas por uma tarefa – no caso, o operador genérico *victimise*, com seus argumentos já instanciados para os personagens *Draco* e *Marian*. Tal operador pode ser especializado por três diferentes operadores básicos: o operador não-determinístico *try_to_kidnap* e os operadores determinísticos *kidnap* e *kill*. As chamadas sucessivas ao planejador, utilizando o valor 2 como grau máximo de não-determinismo, geraram três planos diferentes, cada um deles utilizando uma das especializações de *victimise* (já que o estado inicial passado para o planejador atendia às precondições de todos eles). O primeiro deles conta com a especialização não-determinística *try_to_kidnap*, possuindo dois estados finais. Os dois últimos fazem uso de cada uma das outras especializações, determinísticas, e possuem um estado final cada um. Todos os planos gerados atendem à restrição no grau de não-determinismo. Outro teste foi feito, estabelecendo como grau de não-determinismo o valor 1. Conforme o esperado, apenas os dois últimos planos do teste anterior foram retornados, pois o primeiro deles não atendia a essa nova restrição.

## 5.3 Tratando o Conceito de Tentativas

Para validar o tratamento do conceito de tentativas no planejamento HTN, podemos considerar a estrutura relativa ao operador complexo *perform_heroism*. Este operador representa um evento em que o herói tenta dar uma resposta a um ato de vilania. O feito malicioso perpetrado pelo vilão é representado pelo operador

complexo *perform_villainy*, uma composição que possui, entre seus suboperadores, o operador genérico *victimise*, que pode ser especializado pelo operador não-determinístico *try_to_kidnap*. Este último pode resultar tanto em um rapto bem-sucedido (em que a vítima torna-se prisioneira do vilão), quanto na morte da vítima. Cada um desses efeitos vai provocar uma reação diferente do herói: seu ato heroico consistirá de um resgate, no primeiro caso, e de uma vingança, no segundo (o planejador se decidirá entre um caso ou outro de acordo com o atendimento das respectivas precondições).

Em ambos os casos, a composição associada a *perform_heroism* inclui o suboperador genérico *combat*, que pode ser especializado pelo operador não-determinístico *fight_to_death*. Este último pode causar a morte de qualquer um dos contendores, bem como de ambos. Como tanto o operador *rescue* quanto o operador *avenge* preveem uma ação do herói após a luta (resgatar a vítima no primeiro caso e, em um ato de violência desenfreada, esfolar o vilão no segundo), faz-se necessário o uso do modificador *attempt* para indicar que tais ações finais podem ter alguma de suas precondições não atendida – no caso, o fato de o herói estar vivo.

A fim de confirmar o tratamento correto do conceito de tentativas, primeiramente foram passados os seguintes argumentos para o planejador:

*Fatos*$^{+}$*:*
[kidnapped('Marian','Draco'),current_place('Marian','Red_Castle'),
strength('Brian',120.0),current_place('Brian','Red_Castle'),
protection('Red_Castle',[-1.0,0.0])]
*Fatos*$^{-}$*:*
[current_place('Marian','White_Palace'),current_place('Brian','Gray_Castle'),
strength('Brian',20.0),protection('Red_Castle',[-1.0,20.0])]
*Tarefas:*
[ (f1,combat('Brian', 'Draco')), (f2,free('Brian', 'Marian'))]
*Restrições de Ordem:* [(f1,f2)]
*Grau de Não-Determinismo:* 4

Este teste simula uma situação em que *Maria*n é mantida prisioneira por *Draco* em seu covil, o Castelo Vermelho, e *Brian* está em vias de tentar resgatá-la (o estado inicial estabelece as precondições necessárias para isso). Para tal, o cavaleiro deverá matar o dragão para, em seguida, libertar a princesa. Com este intuito, a rede de tarefas possui dois componentes a serem executados em sequência. O primeiro deles é o operador genérico *combat*, que pode ser especializado pelo operador básico não-determinístico *fight_to_death* (que pode apresentar três resultados possíveis: a morte de qualquer um dos dois personagens envolvidos na luta, ou de ambos) ou pelo operador composto determinístico *defeat* (que estabelece deterministicamente que um personagem mata o outro). O segundo operador da rede de tarefas, *free*, estabelece a libertação da vítima pelo herói, um evento que depende essencialmente da sobrevivência deste último ao combate contra o vilão.

Como o evento *fight_to_death(Brian, Draco)* pode resultar na morte de *Brian*, sua escolha não permitiria garantir a libertação da princesa pelo cavaleiro. Tal incerteza inviabilizaria a geração de um plano que assegure a ocorrência do evento *free(Brian,Marian)*, necessário ao cumprimento da rede de tarefas. A especialização *defeat(Brian, Draco)*, no entanto, garantiria a sobrevivência de *Brian* e o sucesso na geração do plano. De fato, ao executarmos o planejador sob essas condições, obtivemos o seguinte resultado:

```
================================================================
PLAN:
================================================================

    Initial State
      [kidnapped(Marian,Draco),current_place(Marian,Red_Castle),
      strength(Brian,120.0),current_place(Brian,Red_Castle),protection(Red_Castle,[-
      1.0,0.0])]
      [current_place(Marian,White_Palace),current_place(Brian,Gray_Castle),
      strength(Brian,20.0),protection(Red_Castle,[-1.0,20.0])]

    HTN
      [ (f1,combat(Brian,Draco)), (f2,free(Brian,Marian))],[ (f1,f2)]

    Plan
      → fight(Brian,Draco)
      → kill(Brian,Draco)
      → free(Brian,Marian)
```

```
  Final States
--------------------------------------------------------------------------------
    FINAL STATE ID: 1

    Facts +
        [current_place(Marian,Red_Castle),current_place(Brian,Red_Castle),
        protection(Red_Castle,[-1.0,0.0]),strength(Brian,75.0),strength(Draco,0.0),
        murderer(Brian,Draco),affection([Marian,Brian],100.0),joy(Brian,50.0),
        joy(Marian,70.0)]
    Facts -
        [current_place(Marian,White_Palace),current_place(Brian,Gray_Castle),
        strength(Brian,20.0),protection(Red_Castle,[-1.0,20.0]),strength(Draco,45.0),
        alive(Draco),affection([Marian,Brian],0.0),joy(Brian,20.0),joy(Marian,50.0)]
--------------------------------------------------------------------------------

true ;
false.
```

Como se pode ver, o planejador encontrou apenas um plano, utilizando a especialização determinística de *combat*; ao tentarmos gerar outra solução por *backtracking*, não obtivemos sucesso. Seria interessante uma versão da história em que fosse aceitável a morte do herói, mesmo que tal efeito impeça o resgate da princesa, pois assim ganharíamos tanto na variedade das histórias geradas, quanto em tensão dramática. Para permitir o atendimento a tais condições, outro teste similar foi feito, mas incorporando o conceito de tentativa à libertação da princesa; a rede de tarefas foi alterada através da substituição de *free(Brian,Marian)* por *attempt(free(Brian,Marian),1)*:

```
================================================================
PLAN:
================================================================

    Initial State
      [kidnapped(Marian,Draco),current_place(Marian,Red_Castle),
      strength(Brian,120.0),current_place(Brian,Red_Castle),protection(Red_Castle,[-
      1.0,0.0])]
      [current_place(Marian,White_Palace),current_place(Brian,Gray_Castle),
      strength(Brian,20.0),protection(Red_Castle,[-1.0,20.0])]

    HTN
      [ (f1,combat(Brian,Draco)), (f2,attempt(free(Brian,Marian),1))],[ (f1,f2)]

    Plan
      → fight_to_death(Brian,Draco)
      [?][strength(Brian,75.0),not(alive(Draco)),murderer(Brian,Draco)]
```

→ free(Brian,Marian)
[?][strength(Draco,0.0),not(alive(Brian)),murderer(Draco,Brian)]
[?][not(alive(Brian)),murderer(Draco,Brian),not(alive(Draco)),
murderer(Brian,Draco)]

Final States

---

FINAL STATE ID: 1

Facts +
[protection(White_Palace,[1.0,30.0]),affection([Marian,Draco],-100.0),
current_place(Marian,Red_Castle),current_place(Brian,Red_Castle),
protection(Red_Castle,[-1.0,0.0]),affection([Draco,Brian],-100.0),joy(Draco,-
5.0),strength(Brian,75.0),murderer(Brian,Draco),affection([Marian,Brian],100.0
), joy(Brian,50.0),joy(Marian,70.0)]
Facts -
[protection(White_Palace,[1.0,70.0]),affection([Marian,Draco],0.0),
joy(Marian,50.0),current_place(Marian,White_Palace),joy(Draco,0.0),
current_place(Brian,Gray_Castle),strength(Brian,20.0),protection(Red_Castle,[-
1.0,20.0]),affection([Draco,Brian],0.0),strength(Draco,45.0),alive(Draco),
affection([Marian,Brian],0.0),joy(Brian,20.0)]

---

FINAL STATE ID: 2

Facts +
[protection(White_Palace,[1.0,30.0]),affection([Marian,Draco],-100.0),
kidnapped(Marian,Draco),current_place(Marian,Red_Castle),joy(Marian,-
50.0),current_place(Brian,Red_Castle),protection(Red_Castle,[-
1.0,0.0]),affection([Draco,Brian],-100.0),joy(Draco,-
5.0),strength(Draco,0.0),murderer(Draco,Brian)]
Facts -
[protection(White_Palace,[1.0,70.0]),affection([Marian,Draco],0.0),
joy(Marian,50.0),current_place(Marian,White_Palace),joy(Draco,0.0),
current_place(Brian,Gray_Castle),strength(Brian,20.0),protection(Red_Castle,[-
1.0,20.0]),affection([Draco,Brian],0.0),strength(Draco,45.0),alive(Brian)]

---

FINAL STATE ID: 3

Facts +
[protection(White_Palace,[1.0,30.0]),affection([Marian,Draco],-100.0),
kidnapped(Marian,Draco),current_place(Marian,Red_Castle),joy(Marian,-50.0),
current_place(Brian,Red_Castle),protection(Red_Castle,[-1.0,0.0]),
affection([Draco,Brian],-100.0),joy(Draco,-5.0),murderer(Draco,Brian),
murderer(Brian,Draco)]
Facts -
[protection(White_Palace,[1.0,70.0]),affection([Marian,Draco],0.0),
joy(Marian,50.0),current_place(Marian,White_Palace),joy(Draco,0.0),
current_place(Brian,Gray_Castle),strength(Brian,20.0),protection(Red_Castle,[-
1.0,20.0]),affection([Draco,Brian],0.0),strength(Draco,45.0),alive(Brian),
alive(Draco)]

---

Dessa vez, o primeiro plano retornado contempla o evento *fight_to_ death(Brian, Draco)*. Apenas o primeiro resultado avaliado para este evento permite que *Brian* liberte *Marian*, o que faz com que o planejador incorpore o evento *free(Brian, Marian)* a esta ramificação do plano. As demais, em que *Brian* está morto, terminam sem nenhum outro evento, comprovando que o planejador tratou de forma correta o conceito de tentativa, incorporando o evento tentado quando possível e simplesmente descartando-o quando as precondições não puderam ser atendidas. A execução de novas chamadas por *backtracking* retornou também o plano encontrado no teste anterior, sem o uso do modificador *attempt*, e mais nenhum plano após esses dois, o que também era esperado.

## 5.4 Avaliação de Eficiência e do Tratamento de Alternativas

A eficiência do processo de planejamento é, como explicado anteriormente, requisito de fundamental importância para que o sistema de *storytelling* seja capaz de gerar e dramatizar enredos em paralelo. O não-determinismo cria complexidade maior para se gerar um plano, pois é preciso considerar os vários possíveis efeitos de cada evento. Por outro lado, a introdução de conhecimento específico do domínio para detalhar os planos reduz o espaço de busca, provendo maior eficiência.

Como o próximo capítulo precisa ser gerado enquanto o capítulo corrente está sendo dramatizado, torna-se importante avaliar também a viabilidade da geração de várias alternativas para o próximo capítulo, compatíveis com os possíveis finais do capítulo corrente que é apresentado.

Para a realização desses testes, foi utilizado um notebook ASUS UL30A 64 (1.30 GHz) com 4 GB de RAM e Windows 7 Home Premium. Como a versão do SICStus Prolog no qual o IPG foi desenvolvido (3.7.1) é incompatível com o Windows 7, os testes tanto com o IPG quanto com o NDet-HTN foram feitos em uma máquina virtual Windows XP SP3 criada no Oracle VM VirtualBox 3.2.8, configurada para usar 192 MB de RAM. Os tempos médios mencionados foram extraídos de 10 execuções de cada teste em questão. Convém ressaltar que nenhuma execução dos testes já apresentados chegou a 1s, sendo que, na maioria das vezes, o tempo esteve abaixo de 0,4s.

A seguir é apresentada a análise da eficiência do novo processo de planejamento e, em seguida, é analisada a viabilidade do tratamento da geração de várias alternativas de capítulos em paralelo.

### 5.4.1 Eficiência na Geração de Enredos

A introdução de efeitos não-determinísticos fizera com que o planejador de ordem parcial tivesse que examinar tanto listas de efeitos determinísticos quanto listas de efeitos não-determinísticos para descobrir eventos que estabelecem condições e eventos que bloqueiam o estabelecimento de condições. No entanto, o número de efeitos médio dos eventos não chegou a ser aumentado significativamente e isto não chega a impactar o tempo de geração do HTN inicial a ser detalhado.

Na versão atual do Logtell, os maiores capítulos gerados para o contexto exemplo contêm, no máximo, 6 eventos básicos. No entanto, o IPG não impunha limite máximo no número de eventos novos que cada capítulo poderia ter. Com isso, planos com muito mais eventos chegavam a ser examinados (e eventualmente descartados). Tendo em vista a ideia de que o planejamento de ordem parcial agora deve apenas gerar um esboço do plano, usando operadores genéricos e compostos, eventualmente complementados por uns poucos eventos básicos que habilitem precondições, tornou-se viável limitar o número máximo de eventos em um capítulo, o que reduz consideravelmente o espaço de busca. Ao se limitar, por exemplo, o número máximo de eventos em cada capítulo, por exemplo, a 4, consegue-se gerar o HTN inicial em poucos segundos, em tempo bem menor que o tempo médio demandado pelo IPG para a geração de capítulos completos. Isto ocorre apesar de ter que se considerar os efeitos não-determinísticos. Tal resultado pode ser explicado pelo fato de que o planejamento de ordem parcial continuou sendo um "planejamento determinístico" que, no entanto, incorpora a ideia de tentativas. Note-se que a limitação no número de eventos introduzidos pelo planejamento de ordem parcial não impacta o nível de detalhamento dos capítulos, pois, após a decomposição via HTN, pode-se gerar facilmente capítulos com mais de uma dezena de eventos básicos.

Uma vez que o planejamento de ordem parcial deixou de ser o gargalo no desempenho, o desempenho global passou a depender essencialmente do desempenho do planejamento HTN não-determinístico. Para testar a eficiência do planejamento HTN, foi feita uma chamada ao planejador HTN com os argumentos a seguir:

*Fatos*$^{+}$*:* []

*Fatos*$^{-}$*:* []

*Tarefas:*

[(t1, perform_villainy('Draco','Marian')), (t2, perform_heroism('Hoel', 'Marian')), (t3, perform_epilogue)]

*Restrições de Ordem:* [(t1,t2), (t2,t3)]

*Grau de Não-Determinismo:* 8

A rede de tarefas utilizada neste exemplo simula uma sequência com os três operadores complexos criados para simular o desenvolvimento de uma história completa dentro do contexto utilizado para os testes. Tal sequência pode ser gerada sob a forma de capítulos (um para cada um dos operadores) pelo IPG, através das regras de inferência alternativas. Nesse caso, o NDet-HTN planejaria apenas um capítulo por vez, ou seja, apenas uma dessas tarefas por vez. A intenção do teste foi avaliar como o planejador se comporta mesmo com uma rede de tarefas mais complexa do que seria esperado para a geração de um único capítulo do Logtell. O plano gerado, mostrado a seguir, consumiu um tempo médio de 4,97s, com tempo mínimo de 4,19s e tempo máximo de 6,10s:

```
==========================================================================
PLAN:
==========================================================================

Initial State
    []
    []

HTN
    [(t1,perform_villainy(Draco,Marian)),(t2,perform_heroism(Hoel,Marian)),
    (t3,perform_epilogue)],
```

[ (t1,t2), (t2,t3)]

Plan
→ go(Draco,White_Palace)
→ reduce_protection(Marian,White_Palace)
→ attack(Draco,White_Palace)
→ try_to_kidnap(Draco,Marian)
[?][kidnapped(Marian,Draco),not(joy(Marian,25.0)),joy(Marian,-50.0)]
→ roar(Draco)
→ go(Hoel,Green_Forest)
→ donate(Turjan,Hoel)
→ go(Hoel,Red_Castle)
→ attack(Hoel,Red_Castle)
→ fight_to_death(Hoel,Draco)
[?][strength(Hoel,70.0),not(alive(Draco)),murderer(Hoel,Draco)]
→ free(Hoel,Marian)
→ marry(Hoel,Marian)
[?][strength(Draco,0.0),not(alive(Hoel)),murderer(Draco,Hoel)]
→ commit_suicide(Marian)
[?][not(alive(Hoel)),murderer(Draco,Hoel),not(alive(Draco)),murderer(Hoel,Draco)]
→ join_convent(Marian)
[?][not(alive(Marian)),murderer(Draco,Marian)]
→ roar(Draco)
→ go(Hoel,Green_Forest)
→ donate(Turjan,Hoel)
→ go(Hoel,Red_Castle)
→ attack(Hoel,Red_Castle)
→ fight_to_death(Hoel,Draco)
[?][strength(Hoel,70.0),not(alive(Draco)),murderer(Hoel,Draco)]
→ remove_skin(Hoel,Draco)
→ make_funeral(Hoel,Marian)
[?][strength(Draco,0.0),not(alive(Hoel)),murderer(Draco,Hoel)]
→ spread_horror(Draco)
[?][not(alive(Hoel)),murderer(Draco,Hoel),not(alive(Draco)),murderer(Hoel,Draco)]

Como podemos ver, o plano gerado possui seis diferentes caminhos possíveis, cada um levando a um estado final distinto. Isso se deve ao fato de haver dois eventos não-determinísticos, um com dois e outro com três resultados possíveis.

Pode-se confirmar o uso do conceito de tentativas pelo fato de o plano contemplar os casos em que o herói é morto pelo vilão. Por exemplo, caso a vítima tenha sido morta pelo vilão, o herói tentará vingar sua morte, matando o vilão e, em seguida, esfolando-o. Podemos ver no plano que a tentativa de matar o vilão pode, ou não, ser bem-sucedida, já que o evento *fight_to_death* possui três alternativas distintas, podendo resultar na morte de qualquer um dos dois personagens envolvidos, ou de ambos. Caso o herói mate o vilão e permaneça vivo, ele o esfolará

e realizará o funeral da princesa; caso ele seja morto pelo dragão, no entanto, isso não será possível. Sem o uso do conceito de tentativas, o planejador não conseguiria cumprir a tarefa que lhe foi passada, correspondente à realização do ato heroico. Entretanto, como se pode ver no plano gerado, a história segue seu rumo mesmo no caso de o herói não conseguir realizar o seu intento. Caso o dragão permaneça vivo, ele resolve, tendo já matado sua vítima e seu oponente, sair para praticar novas vilanias. Por fim, caso ambos, *Draco* e *Hoel*, morram na luta (e considerando que *Marian* já está morta), nada mais acontecerá (para atender a essa condição, é feito o uso do operador *nil*, conforme visto anteriormente).

### 5.4.2 Processamento em Paralelo das Diversas Alternativas de Capítulos

Para permitir testar a geração e o controle das múltiplas alternativas de capítulos (assim como a viabilidade em termos de tempo de computação), foi feita uma chamada ao *Chapter Controller* passando, como entrada, cinco combinações de estados iniciais e listas de tarefas utilizadas nos exemplos anteriores e, como grau máximo de não-determinismo (nesse caso, um mesmo valor a ser aplicado a todos os planos gerados) o valor 3. As redes de tarefas processadas simultaneamente foram:

```
([(t1, do_malefic_sound(_))], [])
([(t1, fight_to_death('Hoel','Draco')),(t2, perform_epilogue)], [(t1,t2)])
([(t1,victimise('Draco', 'Marian'))],[])
([(f1,combat('Brian', 'Draco')), (f2,free(Brian,Marian))], [(f1,f2)])
([(f1,combat('Brian', 'Draco')), (f2,attempt(free(Brian,Marian),1))], [(f1,f2)])
```

Para cada uma delas, assim como nos testes feitos anteriormente com essas mesmas redes de tarefas, foi passado um estado inicial que permitisse o cumprimento do HTN. Mesmo gerando cinco planos, e todos eles sequencialmente e não em paralelo, o tempo médio de resposta foi de 0,671s (com tempo mínimo de 0,660s e tempo máximo de 0,681s).

Vale ressaltar que, limitando-se o grau de não-determinismo, consegue-se garantir a geração do próximo capítulo para os vários finais que o capítulo corrente

tenha. Além disso, uma vez que o paralelismo seja possível, esse grau de não-determinismo pode ser ainda maior.

# 6 Conclusão

A pesquisa descrita nesta dissertação apresentou um novo modelo para a geração de enredos para *storytelling* interativo com o uso de eventos não-determinísticos. O modelo proposto procura enriquecer a experiência do usuário em termos de variedade dos enredos e de possibilidades de interação, porém sem comprometer a coerência lógica das narrativas, a ser garantida pela inferência de objetivos e pela aplicação de planejamento automático para atingi-los. O modelo foi proposto de forma a ser incorporado ao sistema de *storytelling* para TV interativa Logtell, atendendo a um conjunto de requisitos. No que tange à variedade dos enredos, a possibilidade de inclusão de eventos com resultados diferentes faz com que cada dramatização de um mesmo capítulo possa ser completamente diferente de outra. Adicionalmente, a incorporação do conceito de tentativa mostra-se interessante para reforçar a carga dramática do enredo. No que tange à interação, ao gerar capítulos que levam em conta as possibilidades não-determinísticas, torna-se viável dar a liberdade ao usuário de intervir no capítulo corrente, sem que isso comprometa a continuidade da história.

O uso de planejamento HTN tornou possível a geração automática de enredos não-determinísticos sem que a complexidade resultante da existência de múltiplos caminhos alternativos afetasse o desempenho do processo de geração dos enredos. O mecanismo de planejamento conjuga uma fase de planejamento de ordem parcial para o estabelecimento de um esquema básico para atingir os objetivos, seguido de um detalhamento do plano por meio de uma fase de planejamento HTN. A fase inicial é importante para garantir a incorporação de eventos e situações explicitamente sugeridas pelos usuários e para permitir que o enredo não fique preso à combinação de alternativas previamente pensadas. Criaram-se, contudo, limites no

nível de detalhamento desse esquema inicial, de modo a acelerar o desempenho. Na fase final de planejamento HTN, é possível gerar planos detalhados de forma muito mais rápida do que seria possível se fosse utilizado apenas o planejamento de ordem parcial.

Para permitir que tais conceitos fossem validados, foi implementado um protótipo, a ser integrado ao Logtell, que foi testado com base em uma versão modificada do contexto exemplo de histórias usado para validar a primeira versão do Logtell, correspondente ao gênero "Espadas e Dragões".

Os resultados obtidos mostram a viabilidade da geração de enredos não-determinísticos para *storytelling* interativo, bem como da possibilidade de se fornecer meios ao usuário de interagir de forma mais direta com a narrativa, no próprio nível de dramatização.

Durante o desenvolvimento da dissertação, foi elaborado um artigo relatando os resultados parciais obtidos, tendo sido aceito pelo *12th edition of the Ibero-American Conference on Artificial Intelligence – IBERAMIA 2010* (SILVA *et al*., 2010).

A seguir, a seção 6.1 faz um relato mais detalhado das principais contribuições decorrentes da pesquisa desenvolvida. Por fim, a seção 6.2 destaca as principais possibilidades de trabalhos futuros potenciais que podem ser feitos usando como base o modelo proposto e o planejador implementado.

## 6.1 Principais Contribuições

- **Modelo de Geração de Enredos com Não-Determinismo**

A integração entre inferência de objetivos, incorporação de intervenções dos usuários, planejamento de ordem parcial e planejamento HTN não-determinístico tornou o Logtell mais versátil ao trazer alguns benefícios ao seu processo de geração de enredos, entre os quais se destacam:

  - **Suporte a eventos não-determinísticos.** A adição de eventos não-determinísticos aos enredos gerados pelo Logtell permite oferecer ao usuário um nível maior de interatividade com o sistema, assim como incrementar a tensão dramática e a diversidade das histórias geradas,

aumentando, assim, o *replay value* do sistema. Para melhor usufruir de tal benefício, o contexto utilizado no protótipo privilegia eventos não-determinísticos no momento de selecionar especializações de um evento genérico.

- **Controle do nível de não-determinismo.** Os ganhos trazidos pela adoção do não-determinismo poderiam ser postos a perder caso tal recurso trouxesse, como consequência, a geração de planos demasiadamente complexos que consumissem muito tempo de processamento. Para evitar a ocorrência desse efeito colateral indesejável que, certamente, diminuiria o grau de satisfação do usuário, foi adotado um mecanismo que permite o controle do nível de não-determinismo dos enredos gerados, de forma a impedir que sequências de eventos não-determinísticos causem uma explosão combinatória no número de ramificações.
- **Incorporação do conceito de tentativas.** No modelo proposto, pode-se considerar que a adoção do conceito de tentativas se relaciona com a adoção do não-determinismo de duas formas: se, por um lado, ambos os conceitos contribuem para incrementar tanto a diversidade das histórias quanto a tensão dramática, por outro, a possibilidade de que um evento venha a falhar facilita a criação de composições com eventos não-determinísticos entre seus suboperadores. Isso se deve ao fato de que, desse modo, eventos cujas precondições devem ser estabelecidas por um evento não-determinístico que ocorre anteriormente (e que produz alguma ramificação que não atende a essas precondições) podem ser modelados como "tentativas" e, assim, o processo de planejamento pode gerar alternativas que, de outra forma, não seriam possíveis devido à falha causada por tal ramificação.
- **Processamento em paralelo das diversas alternativas.** Os diferentes estados finais atingíveis pelo enredo de um capítulo tornam-se estados iniciais a partir dos quais o planejamento deve ser efetuado em paralelo, de forma a adiantar o planejamento do próximo capítulo. Foram feitos testes em que se efetuou o planejamento a partir de vários estados

iniciais distintos de uma vez e, embora o processamento não tenha sido verdadeiramente em paralelo, os resultados encontrados foram, mesmo assim, satisfatórios.

- **Eficiência na geração de enredos.** O uso de planejamento HTN mostrou-se de extrema valia para impedir que o tratamento de múltiplas alternativas geradas pelo não-determinismo provocasse algum impacto no tempo de processamento. Mesmo realizando testes com planos mais complexos do que os que são esperados para a composição de um capítulo, o tempo de resposta foi satisfatório.

- **Implementação do protótipo**

O protótipo criado para a avaliação do modelo proposto permitiu verificar como o Logtell pode funcionar com a adição dos novos recursos avaliados. É extremamente desejável manter tais recursos em versões futuras do sistema, pois os requisitos atendidos por eles podem ser considerados extensões daqueles já oferecidos atualmente pelo Logtell. Dessa forma, uma nova versão do sistema deverá usar esse protótipo como base para a sua elaboração.

- **Algoritmo de planejamento não-determinístico eficiente e flexível**

Os testes efetuados com o protótipo comprovaram que a combinação do algoritmo de planejamento não-determinístico NDet-HTN com uma adaptação do planejador de ordem parcial do IPG é eficiente, gerando planos complexos em pouco tempo. Embora não seja o primeiro planejador não-determinístico a mostrar eficiência através do uso de HTN, NDet-HTN diferencia-se não só por incorporar o conceito de tentativas, mas também por sempre garantir o cumprimento do HTN. A combinação com o planejamento de ordem parcial, por outro lado, garante que planos não fiquem restritos à utilização de métodos previamente pensados.

O algoritmo é genérico e não está limitado ao uso em *storytelling* interativo. A princípio, pode ser aplicado a qualquer outro domínio, desde que se elabore um contexto com atributos, relacionamentos, eventos, pré- e pós-condições e eventuais generalizações e composições apropriados. Por exemplo, em (VALFRE, 2010) o planejador é usado para a composição de serviços web.

## 6.2 Trabalhos Futuros

- **Modificações nos outros módulos do Logtell**

Para permitir que o usuário interaja de forma a aproveitar os novos recursos acrescentados, torna-se necessária a realização de modificações em outros módulos. Segue uma breve apresentação dos principais aspectos a se considerar:

  - **Simulation Controller**

Precisa passar a tratar o novo formato do plano, que deixa de ser uma sequência de eventos e passa a ser uma árvore de decisão. Além disso, necessita de adaptações que permitam passar de forma adequada ao *NDet-IPG* as novas sugestões feitas pelo usuário durante a dramatização.

  - **Interface (User Interface e Interface Controller)**

Esses dois módulos, diretamente relacionados à apresentação da trama ao usuário, requerem modificações que permitam a interferência no processo de dramatização e, assim, explorar o não-determinismo introduzido nos enredos. Tais intervenções poderiam se dar de várias formas, e em vários níveis – variando, por exemplo, de uma simples seleção de preferências com relação ao enredo (torcer por um determinado personagem, por exemplo) até uma intervenção mais direta (determinando quais objetivos devem ser alcançados, por exemplo).

  - **Modo passo a passo**

É necessário alterar o módulo de apresentação passo a passo, de modo a permitir uma melhor apresentação dos vários caminhos possíveis de serem percorridos pelo desenvolvimento do enredo.

- **Tratamento do conceito de emoções**

A adoção do não-determinismo permite ao usuário interferir na dramatização da trama, por intermédio de escolhas que direcionem o plano gerado através de alguma de suas ramificações. Uma das possibilidades para a realização dessa interferência seria através do uso de atributos emocionais relacionados aos personagens em particular e à trama como um todo.

A experiência do usuário pode ser incrementada através da modelagem de emoções para guiar o comportamento dos personagens, tanto na seleção de objetivos quanto na relação com outros personagens da trama, obtendo-se assim uma maior

riqueza dramática. Emoções podem ser associadas não apenas a personagens, mas a eventos, que poderiam ter diferentes níveis de atributos emocionais que permitissem classificá-los (e ao enredo gerado como um todo) em termos de características tais como alegria/tristeza, aventura, romance, mistério, violência etc. Tais atributos emocionais poderiam, então, servir como parâmetros de entrada para a condução de um enredo não-determinístico, de forma que os usuários pudessem selecionar os níveis desejados para cada tipo de emoção e o sistema, então, escolhesse o melhor plano possível de forma a atender a essas seleções.

FURTADO (2009) apresenta um método para determinar o comportamento dos personagens que participam de uma história através de um processo de tomada de decisão em três passos: seleção de objetivos, seleção de planos e comprometimento. Os critérios de seleção refletem preferências individuais originadas, respectivamente, de pulsões, atitudes e emoções associadas a cada personagem (de acordo com sua "personalidade"). Relações entre personagens também são levadas em consideração, permitindo que eles interajam entre si de forma positiva (através de colaboração) ou de forma negativa (através de frustração de objetivos). Esse trabalho pode ser usado como ponto de partida para a introdução de modelos de emoções.

- **Aplicação do planejador a outros tipos de aplicações**

Conforme dito anteriormente, o planejador NDet-HTN permite o uso de outros domínios além de *storytelling* interativo, já existindo inclusive um exemplo de uso para a composição de serviços web. Seria interessante explorar outras formas de uso do planejador, explorando domínios como, por exemplo, alguns jogos simples (por exemplo, *Tic-Tac-Toe*, o popular "jogo da velha"). Mesmo sem sair da área de *storytelling* interativo, pode-se pensar em simulações para treinamento empresarial e jogos de negócios, por exemplo.

# 7 Referências Bibliográficas


BERTOLI, P., CIMATTI, A., ROVERI, M., TRAVERSO, P., 2001, "Planning in Nondeterministic Domains under Partial Observability via Symbolic Model Checking". In: *IJCAI 2001*, pp. 473-478.

BURCH, J. R., CLARKE, E. M., MCMILLAN, K. L., DILL, D. L., HWANG, L. J., 1992, "Symbolic Model Checking: 1020 States and Beyond". *Information and Computation*, 98(2), pp. 142-170, June.

CAMANHO, M. M., 2009, *Conciliando Coerência e Responsividade em Storytelling Interativo*. Dissertação de M.Sc., Departamento de Informática, Universidade Federal do Estado do Rio de Janeiro, Rio de Janeiro.

CAMANHO, M. M., CIARLINI, A. E. M., FURTADO, A. L., POZZER, C. T., FEIJÓ, B., 2009, "A Model for Interactive TV Storytelling". In: *Anais do VIII Simpósio Brasileiro de Jogos e Entretenimento Digital (SBGAMES 2009)*, Rio de Janeiro, outubro.

CAMANHO, M. M., CIARLINI, A. E. M., FURTADO, A. L., POZZER, C. T., FEIJÓ, B., 2008, "Conciliating Coherence and High Responsiveness in Interactive Storytelling". In: *Proceedings of the 3rd ACM International Conference on Digital Interactive Media in Entertainment and Arts (DIMEA 2008)*, Atenas.

CAVAZZA, M., CHARLES, F., MEAD, S., 2002, "Character-Based Interactive Storytelling". *IEEE Intelligent Systems*, special issue on AI in Interactive Entertainment, 17(4), pp. 17-24.

CHARLES, F., CAVAZZA, M., MEAD, S., 2001, *Character-Driven Story Generation in Interactive Storytelling*. Relatório Técnico, VSMM, Berkeley.

CIARLINI, A. E. M., CAMANHO, M. M., DÓRIA, T. R., FURTADO, A. L., POZZER, C. T., FEIJÓ, B., 2008, "Planning and Interaction Levels for TV Storytelling". *1st Joint International Conference on Interactive Digital Storytelling*, Erfurt, Germany, 26-29 November.

CIARLINI, A. E. M., 1999, *Geração Interativa de Enredos*. Tese de D.Sc., Departamento de Informática, PUC-Rio, Rio de Janeiro.

CIARLINI, A. E. M., CASANOVA, M. A., FURTADO, A. L., VELOSO, P. A., 2009, "Modeling Interactive Storytelling Genres as Application Domains". *Journal of Intelligent Information Systems* (2009) :1-35, November 17.

CIARLINI, A. E. M., POZZER, C. T., FURTADO, A. L., FEIJÓ, B., 2005, "A Logic-Based Tool for Interactive Generation and Dramatization of Stories". In: *Proceedings of the ACM SIGCHI International Conference on Advances in Computer Entertainment Technology (ACE 2005)*, Valencia, pp.133-140, June.

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

CRAWFORD, C., 1999, "Assumptions Underlying the Erasmatron Storytelling System". In: *Working Notes of the 1999 AAAI Spring Symposium on Narrative Intelligence*, AAAI Press.

DÓRIA, T. R., 2009, *Dramatização Não-Determinística de Enredos em Storytelling para TV Interativa*. Dissertação de M.Sc., Departamento de Informática, Universidade Federal do Estado do Rio de Janeiro, Rio de Janeiro.

DÓRIA, T. R., CIARLINI, A. E. M., ANDREATTA, A., 2008, "A Nondeterministic Model for Controlling the Dramatization of Interactive Stories". In: *Proceedings of the ACM MM2008 - 2nd ACM Workshop on Story Representation, Mechanism and Context - SRMC08*, Vancouver, Canada.

EMERSON, E. A., 1990, "Temporal and Modal Logic". In: van Leeuwen, J. (ed.), *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics*, Elsevier Science Publishers B. V., Amsterdam, The Netherlands, pp. 996-1072.

EROL, K., HENDLER, J., NAU, D. S., 1994, "UMCP: A Sound and Complete Procedure for Hierachical Task-Network Planning". In: *Proceedings of the International Conference on AI Planning Systems (AIPS)*, pp. 249-254.

FURTADO, A. L., 2009, *A Decision-Making Process for Digital Storytelling*. MCC 21.

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

GRASBON, D., BRAUN, N., 2001, "A Morphological Approach to Interactive Storytelling". In: *Proceedings of CAST01, Living in Mixed Realities*. Special issue of Netzspannung.org/journal, the Magazine for Media Production and Inter-media Research, pp. 337-340, Sankt Augustin, Germany.

HOPCROFT, J. E., MOTWANI, R., ULLMAN, J. D., 2001, *Introduction to Automata Theory, Languages and Computation*. 2 ed., Boston, Addison-Wesley.

KNIGHT, F. H., 1921, *Risk, Uncertainty and Profit*. Hart, Schaffner, and Marx.

KUTER, U., NAU, D. S., 2004, "Forward-Chaining Planning in Nondeterministic Domains". In: *AAAI-2004*.

LOUCHART, S., AYLETT R., ENZ S., DIAS, J., 2006, "Understanding Emotions in Drama, a Step Towards Interactive Narratives". In: Tim Kovacs and James A. R. Marshall (eds.), *Proceedings of AISB'06: Adaptation in Artificial and Biological Systems*, April 2006, pp. 38-44.

MATEAS, M., STERN, A., 2005, "Structuring Content in the Façade Interactive Drama Architecture". In: *Proceedings of the Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE 2005)*, Marina del Rey, June.

NAU, D., MUÑOZ-AVILA, H., CAO, Y., LOTEM, A., MITCHELL, S., 2001, "Total-Order Planning with Partially Ordered Subtasks". In: *IJCAI-2001*, Seattle, August.

PAIVA, A., MACHADO, I., PRADA, R., 2001, "Heroes, Villains, Magicians, ...: Dramatis Personae in a Virtual Story Creation Environment". In: *Proc. Intelligent User Interfaces*.

PEREIRA, S., 2007, *Planejamento sob Incerteza para Metas de Alcançabilidade Estendidas*. Tese de D.Sc., IME-USP.

PISTORE, M., TRAVERSO, P., 2001, "Planning as Model Checking for Extended Goals in Non-Deterministic Domains". In: *17th International Joint Conference on Artificial Intelligence*, pp. 479-484, Washington.

POZZER, C. T., 2005, *Um Sistema para Geração, Interação e Visualização 3D de Histórias para TV Interativa*. Tese de D.Sc., Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro.

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

RIEDL, M. O., STERN, A., DINI, D., ALDERMAN, J., 2008, "Dynamic Experience Management in Virtual Worlds for Entertainment, Education, and Training". *International Transactions on Systems Science and Applications*, Special Issue on Agent 110 Based Systems for Human Learning, v. 4, n. 2.

RIEDL, M, YOUNG, R. M., 2006, "From Linear Story Generation to Branching Story Graphs". *IEEE Computer Graphics and Applications*, v. 26, n. 3, pp. 23-31, May/June, doi:10.1109/MCG.2006.56.

RUSSEL, S. J., NORVIG, P., 2002, *Artificial Intelligence: A Modern Approach*. 2 ed., Upper Saddle River, New Jersey: Prentice-Hall.

SANTOS, G. L., 2004, *Máquinas de Estados Hieráquicas em Jogos Eletrônicos*. Dissertação de M.Sc., Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro.

SILVA, F. A. G., CIARLINI, A. E. M., SIQUEIRA, S. W. M., 2010, "Nondeterministic Planning for Generating Interactive Plots". *Ibero-American Conference on Artificial Intelligence*, 162, Bahía Blanca, Argentina, 1-5 November.

SPIERLING, U., BRAUN, N., IURGEL, I., GRASBON, D., 2002, **"**Setting the Scene: Playing Digital Director in Interactive Storytelling and Creation". *Computers & Graphics*, v. 26, n.1, pp. 31-44.

TANAKA, K., 1997, *An Introduction to Fuzzy Logic for Practical Applications*. Springer-Verlag, New York.

VALFRE, G. B., 2010, *Composição e Monitoração Automáticas e Contínuas de Serviços Web Não-Determinísticos*. Dissertação de M.Sc., Departamento de Informática, Universidade Federal do Estado do Rio de Janeiro, Rio de Janeiro.

YOUNG, R. M., 2001, "An Overview of the Mimesis Architecture: Integrating Narrative Control into a Gaming Environment". In: *Working notes of the AAAI Spring Symposium on Artificial Intelligence and Interactive Entertainment*, AAAI Press, pp. 78-81, Stanford, CA, March.