UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO

CENTRO TECNOLÓGICO

COLEGIADO DE CIÊNCIA DE COMPUTAÇÃO

André Barreto Silveira

Uma Abordagem para Geração de Narrativas Interativas Utilizando Linguagem Específica de Domínio

Vitória, ES

2019

André Barreto Silveira

Uma Abordagem para Geração de Narrativas Interativas Utilizando Linguagem Específica de Domínio

Monografia apresentada ao Curso de Ciência de Computação do Departamento de Informática da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do Grau de Bacharel em Ciência de Computação.

Universidade Federal do Espírito Santo – UFES

Centro Tecnológico

Departamento de Informática

Orientador: Prof. João Paulo Almeida

Vitória, ES

2019

André Barreto Silveira
Uma Abordagem para Geração de Narrativas Interativas Utilizando Linguagem Específica de Domínio/ André Barreto Silveira. – Vitória, ES, 2019-
91 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. João Paulo Almeida

Monografia (PG) – Universidade Federal do Espírito Santo – UFES
Centro Tecnológico
Departamento de Informática, 2019.

1. Narrativa Interativa. 2. Geração de Narrativa Interativa. 3. Linguagem Específica de Domínio. 4. Sistema Inteligente. 5. Motor de jogos. I. André Barreto Silveira. II. Prof. João Paulo Almeida III. Universidade Federal do Espírito Santo. IV. Uma Abordagem para Geração de Narrativas Interativas Utilizando Linguagem Específica de Domínio

CDU 02:141:005.7

André Barreto Silveira

Uma Abordagem para Geração de Narrativas Interativas Utilizando Linguagem Específica de Domínio

Monografia apresentada ao Curso de Ciência de Computação do Departamento de Informática da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do Grau de Bacharel em Ciência de Computação.

Trabalho aprovado. Vitória, ES, 12 de julho de 2019:

Prof. João Paulo Almeida
Orientador

Prof. Vítor E. Silva Souza
Convidado 1

Profa. Roberta Lima Gomes
Convidado 2

Vitória, ES
2019

Agradecimentos

Agradeço primeiramente ao professor João Paulo, que aceitou me orientar neste projeto de interesse pessoal. Obrigado por todo o apoio e paciência ao longo de todo o processo, e também por estar sempre disposto a qualquer dúvida. Sua orientação foi essencial para a conclusão deste trabalho.

Agradeço a minha família, por me dar a oportunidade de trilhar este caminho, passando por esta universidade, estudar e ser capaz de realizar este projeto. Obrigado por todo amor, carinho e apoio.

Agradeço a todos os professores que se empenharam em ensinar, contribuindo para o meu crescimento pessoal e profissional, desde o fundamental até agora.

E agradeço aos meu amigos pela companhia e ajuda durante o curso. Um obrigado especial àqueles que me apoiaram diretamente na elaboração deste trabalho.

Resumo

Narrativas interativas possuem uma grande participação nas mídias digitais, seja por entretenimento como jogos ou filmes interativos; treinamento e avaliação de funcionários ou candidatos a cargos no mercado; ou até estudo e desenvolvimento de habilidades sociais.

A criação de narrativas interativas é consideravelmente mais exigente em comparação a uma narrativa linear, uma vez que existe a interação de um usuário externo que evidentemente deve ser capaz de alterar o desenrolar da história. Ou seja, para uma só narrativa interativa, deve-se criar múltiplos desenvolvimentos da mesma história, e idealmente sem perder a qualidade da experiência em qualquer um dos rumos.

Essa demanda e dificuldade levou a diversas pesquisas sobre geração de narrativas interativas, uma área atualmente aquecida que busca encontrar formas automáticas e inteligentes para facilitar aos autores a produção destas histórias ramificadas. O presente trabalho apresenta uma abordagem para a geração de narrativas interativas utilizando uma linguagem específica de domínio, cujo propósito é permitir um autor a criar histórias em um nível de abstração maior e específico para interatividade e ramificação de histórias.

Além disto, é definida uma estratégia completa de como utilizar esta linguagem especializada para gerar – a partir das criações de um autor na linguagem – um Diretor de Narrativa (DN), capaz de guiar a história previamente definida. Por fim, define-se a necessidade de um aplicativo independente, construído para se integrar com o DN, que realize o intermédio entre um usuário e a história codificada no DN.

A partir desta estratégia definida, está inclusa neste trabalho a implementação de todos as etapas – a criação da linguagem STGEN, específica para criação de narrativas interativas; o esquema do Diretor de Narrativa, gerado a partir de um ambiente de criação de narrativas que utiliza a STGEN e é usado por autores; o desenvolvimento de um aplicativo independente; e por fim dois exemplos de aplicação, incluindo a definição das histórias interativas no ambiente e sua execução no aplicativo.

Palavras-chaves: Narrativa Interativa, Geração de Narrativa Interativa, Linguagem Específica de Domínio, Sistema Inteligente, Motor de Jogos.

Lista de ilustrações

Códigos

Lista de abreviaturas e siglas

DN	Diretor da Narrativa
CC	Caso Contrário
DSL	Domain Specific Language
IDE	Integrated Development Environment
LSP	Language Server Protocol
AST	Abstract Syntax Tree

Sumário

1 Introdução

Narrativas são constantemente o centro das interações humanas. Seja por mídias como filmes, novelas e livros, ou por meio de escrita e discurso oral, culturalmente histórias são contadas por entretenimento ou para compartilhar experiências reais e conhecimento (RIEDL; BULITKO, 2013). Em particular, destacam-se as narrativas interativas que trazem um novo tipo de imersão. É razoável dizer que interagir diretamente com uma história e alterar seu fluxo, atuando como um personagem do mundo fictício, garante um engajamento maior dos espectadores – que deixam de ser apenas espectadores para condutores.

Narrativas interativas têm suas raízes no entretenimento, com um grande mercado voltado para jogos digitais. Além disso, pesquisas estão sendo desenvolvidas considerando o papel da narrativa interativa em outros tipos de aplicações, como educação, treinamento e propaganda (RIEDL; BULITKO, 2013). Com estas visões, segue que há uma grande demanda na criação de histórias interativas.

Em estratégias iniciais de criação de narrativa interativa, um autor escreveria exaustivamente o conteúdo, antecipando cada possível ação de um usuário. Neste caso, cada possível trajetória da história é manualmente descrita pelo autor, como em clássicos Choose-Your-Own-Adventure (CYOA), ou Escolha-Sua-Aventura (RIEDL; BULITKO, 2013). Claramente este é um processo limitado pelo crescimento exponencial da quantidade de conteúdo a ser manualmente criado de acordo com o aumento do número de opções que um usuário tem de alterar o fluxo da história. Uma forma de superar essa limitação é utilizar a estratégia de geração de narrativa interativa.

Geração de narrativa interativa é uma das abordagens no processo de criação de narrativas que busca minimizar o esforço da produção de conteúdo narrativo, automatizando esta tarefa em alguma instância. Esta estratégia pode envolver um sistema inteligente capaz de criar conteúdo de acordo com as ações do usuário, seja antecipando todas as possíveis ações e preparando as trajetórias previamente (RIEDL; YOUNG, 2006), ou adaptando a narrativa dinamicamente, reagindo ao usuário.

Atribuir a responsabilidade a uma máquina de criar enredo trouxe muitos desafios a serem superados, uma vez que histórias são um produto intelectual complexo que envolve um leque cognitivo amplo exercido pelos humanos. Este normalmente envolve conceitos como percepção de tempo e espaço, atribuição de conhecimentos a personagens, objetivos de personagens, validar planos de personagens que buscam completar seus objetivos, lidar com falhas nos planos ao atingir obstáculos, entre outros (GERVáS, 2009).

Além do desafio da geração de conteúdo, existem outras questões a serem avaliadas

no processo como um todo. Por exemplo, qual será o nível de intencionalidade do autor das histórias e como ele vai descrever isto ao sistema? Como integrar o sistema inteligente a outras aplicações? Como as histórias serão apresentadas aos usuários?

Um aspecto interessante a ser delineado é a distinção entre *fábula* (o que é contado) e *discurso* (como é contado) (GERVáS, 2009). Para produzir a *fábula* de uma história, um sistema pode conter conhecimento do mundo fictício codificado em alguma ontologia de domínio, contendo as noções básicas do mundo, ações possíveis e como elas afetam o mundo (RIEDL; BULITKO, 2013). Desta forma, o gerador de narrativa pode utilizar-se destes conceitos no processo de criação, focando no *o quê* a ser contado. Também possibilita que os autores descrevam a história de forma mais abstrata, permitindo-os a manter sua intencionalidade sem restringir muito os possíveis produtos narrativos.

Tradicionalmente a descrição de histórias é feita textualmente, logo estabelecer uma linguagem específica de domínio é uma possível estratégia para definir a interface do sistema em questão. Linguagens específicas de domínio, ou *domain-specific languages* (DSL), são linguagens criadas especificamente para um tipo de aplicação, garantindo uma maior expressividade e usabilidade (MERNIK; HEERING; SLOANE, 2005).

1.1 Motivação

Narrativas interativas é um domínio envolvendo inteligência artificial com diversas pesquisas e experimentos sendo realizados e um notável número de sistemas sendo publicados, como *Façade* (MATEAS; STERN, 2002), *Haunt II* (MAGERKO, 2005), *C-DRaGer* (SHARMA et al., 2010), *Merchant of Venice* (PORTEOUS; CAVAZZA; CHARLES, 2010) e *The Automated Story Director* (RIEDL et al., 2008).

Porém ainda existem muitas questões a serem pesquisadas e desenvolvidas. De acordo com Riedl e Bulitko (2013), “Como e quando o sistema inteligente deve intervir no mundo virtual pelo usuário? Como gerar estruturas narrativas? Como codificar a intencionalidade autoral humana e a que nível de abstração? Como ajustar as experiências narrativas?”

O uso de conhecimento estruturado para armazenar os conceitos do mundo virtual e servir como base na geração de conteúdo narrativo é uma estratégia sendo atualmente pesquisada e utilizada em sistemas e experimentos. Por exemplo, Gervás et al. (2005) utiliza ontologia de domínio para codificar histórias e gerar novas a partir de reorganização e mistura de conceitos similares, mas apresenta a limitação de pouca interatividade no decorrer da narrativa.

Sistemas inteligentes capazes de imergir usuários em experiências narrativas ainda permitindo-os guiar suas próprias experiências podem revolucionar a forma com que

sistemas computacionais são usados para entreter, educar e treinar humanos (RIEDL; BULITKO, 2013). Este tem sido um tema amplamente pesquisado nas últimas décadas, com muitas limitações a serem superadas.

Diante do exposto, este trabalho justifica-se pela necessidade de pesquisas na área de narrativa interativa e especificamente na geração destas, através de sistemas inteligentes capaz de criar conteúdo narrativo e guiar o desenrolar de histórias interativas. E também pelo grande interesse e demanda de geradores de narrativa.

1.2 Objetivos

O objetivo geral deste trabalho é desenvolver um sistema capaz de gerar e conduzir narrativas interativas em aplicações lúdicas, utilizando uma linguagem específica de domínio com interface textual para a definição do enredo principal e outros detalhes da história.

Os objetivos específicos são:

1. Criar uma *domain-specific language* (DSL), ou linguagem específica de domínio, que permita a um especialista descrever narrativas interativas em um ambiente de criação especializado.

2. Habilitar um gerador de códigos a compilar a DSL em questão, gerando um sistema que contenha as informações da história escrita pelo autor. Este deverá ser capaz de conduzir e adaptar a narrativa de acordo com as ações de um usuário.

3. Desenvolver um aplicativo dedicado em um motor de jogos para permitir a demonstração e avaliação da viabilidade do sistema gerado no ambiente de criação de narrativas interativas.

4. Escrever uma narrativa interativa na DSL em questão, incluindo um exemplo de recriação de uma narrativa interativa, e com o código gerado, incorporá-lo no aplicativo dedicado para demostrar a aplicabilidade do trabalho.

1.3 Metodologia

Neste trabalho foram projetados uma DSL especializada para descrição de narrativas interativas, chamada STGEN, um Diretor de Narrativas (DN) capaz de conduzir e guiar uma dessas histórias e um aplicativo dedicado para testar o DN.

A DSL STGEN foi desenvolvida utilizando o *framework* Xtext juntamente com o IDE Eclipse. A DSL, entre outros aspectos, possui um gerador de código, que foi construído de forma a gerar código em C# do DN.

O DN foi elaborado em C# como um sistema com algoritmos e estrutura de dados fixos e métodos para ser integrado a outros sistemas maiores. Este foi usado como modelo na DSL STGEN durante o processo de geração de código, assim gerando DN para histórias interativas diferentes.

Foi desenvolvido um aplicativo em C# e Unity que possui o papel de servir como intermédio entre o usuário e o DN.

Por fim, foram elaboradas duas narrativas interativas no ambiente de criação e, com o DN gerado a partir delas, foram realizados testes no aplicativo dedicado.

1.4 Organização

Este trabalho é organizado da seguinte forma. A seção 2 aborda os conceitos fundamentais presentes neste trabalho, delineando tecnologias, métodos e os contextos históricos dos temas em questão. A seção 3 apresenta a estrutura do ambiente de construção de narrativas proposto, detalhando sua arquitetura geral; apresenta o projeto da linguagem STGEN; apresenta o esquema do Diretor de Narrativa (DN) que é gerado para cada narrativa escrita em STGEN, incluindo sua estrutura de dados, lógica de execução e métodos de integração com outros aplicativos; apresenta alguns métodos comuns de elaboração de história voltados a DSL STGEN; e por fim apresenta o aplicativo dedicado criado em C# e Unity que utiliza o DN. Na seção 4 é detalhada a implementação de todo o ambiente proposto. Na seção 5 são apresentados exemplos completos de criação de narrativas em STGEN e sua execução no aplicativo. Por fim, na seção 6, são expostas as conclusões obtidas com o desenvolvimento deste trabalho, incluindo a apresentação de trabalhos relacionados, uma discussão que contém a análise crítica dos produtos desenvolvidos e as propostas de trabalhos futuros.

2 Referencial Teórico

Nesta seção será inicialmente apresentada a conceituação básica sobre narrativa interativa (seção 2.1), incluindo abordagens conhecidas para a criação das mesmas (seção 2.1.1) e em seguida uma discussão sobre métodos de geração de narrativas interativas (seção 2.1.2). Após isto, são apresentadas as noções básica sobre linguagens específicas de domínio (seção 2.2). E por fim é discutido brevemente sobre motores de jogos (seção 2.3).

2.1 Narrativa interativa

"Narrativa interativa é um tipo de experiência interativa e digital na qual usuários criam ou influenciam uma narrativa dramática através de ações, seja assumindo o papel de um personagem em um mundo virtual, emitindo comandos a personagens controlados por computador, ou diretamente manipulando o estado do mundo fictício" (RIEDL; BULITKO, 2013).

Tradicionalmente, jogos digitais e outras aplicações utilizam narrativas lineares ou ramificadas (RIEDL; YOUNG, 2006). Em narrativas lineares, o enredo é fixo e cuidadosamente preparado para maximizar a experiência de seu usuário. Em narrativas interativas ramificadas, as ações do usuário impactam significantemente no desenrolar do enredo, mas de forma determinística. Ou seja, repetidas as mesmas ações, a mesma narrativa será exibida. Para jogos especificamente, estes fatos limitam a "rejogabilidade".

A produção de narrativas interativas pode envolver um sistema que auxilie no processo. O grau de inteligência deste sistema influencia nos produtos narrativos possíveis resultantes das definições feitas por um autor no ato de escrever a história. Em um lado temos um sistema simples que auxilia na interatividade da narrativa ao criar uma navegabilidade direcionada de acordo com as escolhas do usuário. Este é o clássico exemplo da produção de um "Escolha-Sua-Aventura". Por exemplo, Twine[1] é uma ferramenta que possibilita autores descreverem suas histórias interativas através de ligações de hipertexto, agilizando o processo. Apesar de simples, esta abordagem é limitada por conta da necessária criação exaustiva de conteúdo autoral.

No outro lado, observamos sistemas inteligentes com maior autonomia e capazes de criar conteúdo sem que o autor tenha descrito manualmente cada passagem. Naturalmente, há uma maior complexidade envolvida não só na criação de conteúdo em si como na forma em que o autor deve descrever suas mensagens ou experiências nas quais tem intenção de transmitir a sua audiência. O desafio central nessa linha é balancear a coerência narrativa e

[1] Ferramenta *open-source*, disponível em <https://twinery.org/>.

a autonomia do jogador (RIEDL; BULITKO, 2013). Manter coerência da história significa garantir que a progressão do enredo está bem relacionada com os eventos passados, e que as ações e eventos ocorridos têm consequências relevantes na história. O que entra em conflito com a autonomia do jogador, que é o quanto o usuário externo consegue alterar a história através de ações diretas no mundo virtual. Quanto mais liberdade o jogador tem de agir e alterar o fluxo narrativo, maior a complexidade para manter a coerência narrativa.

2.1.1 Abordagens para narrativa interativa

Na literatura, muitas abordagens foram utilizadas considerando e enfatizando diferentes aspectos. Riedl e Bulitko (2013) organizaram esses esforços em uma taxonomia de abordagens para narrativa interativa, distinguindo-as através de três dimensões: intenção autoral, autonomia dos personagens virtuais e modelo de jogador.

Intenção autoral. O quanto a intenção original do autor limita o sistema interativo? Em um extremo há sistemas com alta restrição das narrativas, preservando a intenção do autor. No outro, sistemas que assumem responsabilidade criativa e possuem autonomia para alterar a narrativa de formas talvez não previstas pelo autor. (RIEDL; BULITKO, 2013).

Como mencionado anteriormente, o lado mais prescritivo deste espectro aponta para o "Escolha-Sua-Aventura". "O Abominável Homem das Neves" (MONTGOMERY, 2006) é um livro de narrativa interativa onde o leitor é levado a fazer escolhas. Dependendo destas, ele é enviado para certas páginas, até que chegue em um dos possíveis finais. Sistemas neste lado utilizam como representação de história um *gráfico de história ramificado* (ver figura 1), onde cada nó representa um pedaço de conteúdo narrativo manualmente descrito e os arcos representam as possíveis escolhas dos usuários (RIEDL; BULITKO, 2013).

Este trabalho terá foco no outro lado deste espectro, em sistemas que assumem certa responsabilidade pela experiência narrativa do usuário e são capazes de produzir conteúdo dinamicamente. Neste sentido, autores não podem se expressar literalmente e diretamente como através de um gráfico de história ramificado. Deve haver alguma representação mais abstrata com a qual o autor vai descrever a história (RIEDL; BULITKO, 2013). *Gráfico de enredo* é uma abordagem comum que representa histórias através de um conjunto parcialmente ordenado de elementos narrativos a serem selecionados pelo sistema na composição da narrativa em tempo real (RIEDL et al., 2008). Neste esquema, os nós são elementos narrativos e os arcos representam restrições de forma que os próximos eventos só ocorrem caso satisfeito todos os eventos prévios. Veja na figura 2 um exemplo de gráfico de enredo.

Sistemas inteligentes narrativos que possuem grande autoridade criativa requerem

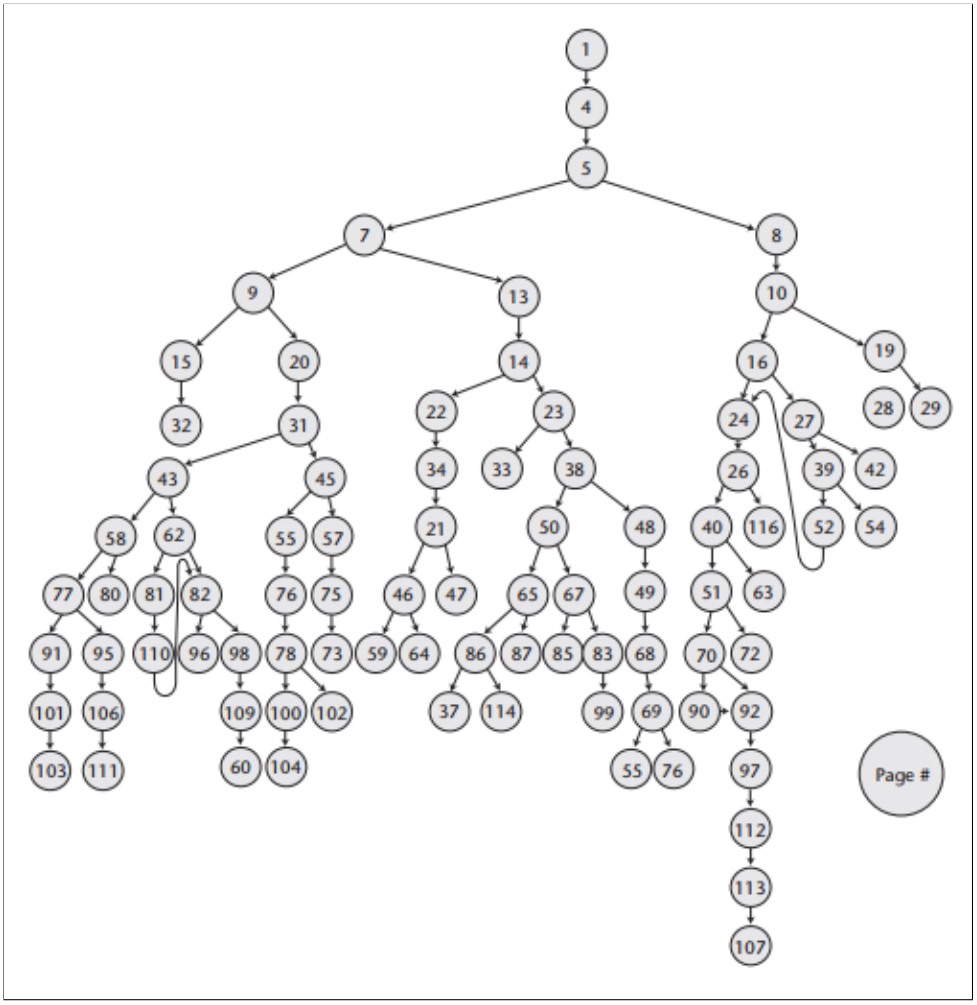

Figura 1 – Gráfico de história ramificado do livro O Abominável Homem das Neves (RIEDL; BULITKO, 2013)

conhecimento sobre o mundo fictício e sobre como conduzir histórias. Este conhecimento, ou a *teoria de domínio*, é um conjunto de dados abstratos envolvendo ações e pré/pós condições para ações (RIEDL; BULITKO, 2013). Tais informações podem ser consolidadas usando ontologias de domínio. Por exemplo, Gervás et al. (2005) propõem um sistema que utiliza uma ontologia de domínio baseado em contos de fada e, juntamente com um raciocínio baseado em casos, gera novos contos de fadas "inspirados" em contos já existentes. A figura 3 exibe uma parte desta ontologia.

Autonomia dos personagens virtuais. Narrativas são compostas de personagens e a interação entre eles, sejam pessoas ou animais e objetos antropomorfos. Personagens críveis são aqueles que possuem características próprias, objetivos e reagem de acordo com seu mundo (RIEDL; BULITKO, 2013). Seguindo essa análise, sistemas podem ser focados na história ou focados na autonomia dos personagens. O primeiro tipo não admite (ou admite pouca) autonomia dos personagens, de forma que as suas interações não são consideradas essenciais para a progressão do enredo. O segundo dá relevância as interações e objetivos dos personagens. No extremo, o sistema dá completa autonomia a eles, de forma

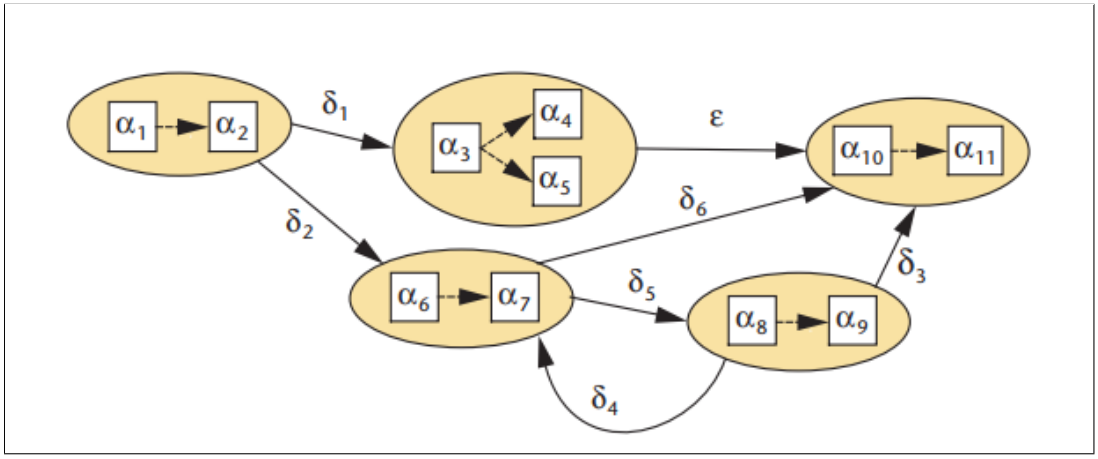

Figura 2 – Gráfico de enredo genérico intercalando conteúdo narrativo (nós circulares) e pontos de decisão (arcos externos) (RIEDL; YOUNG, 2006)

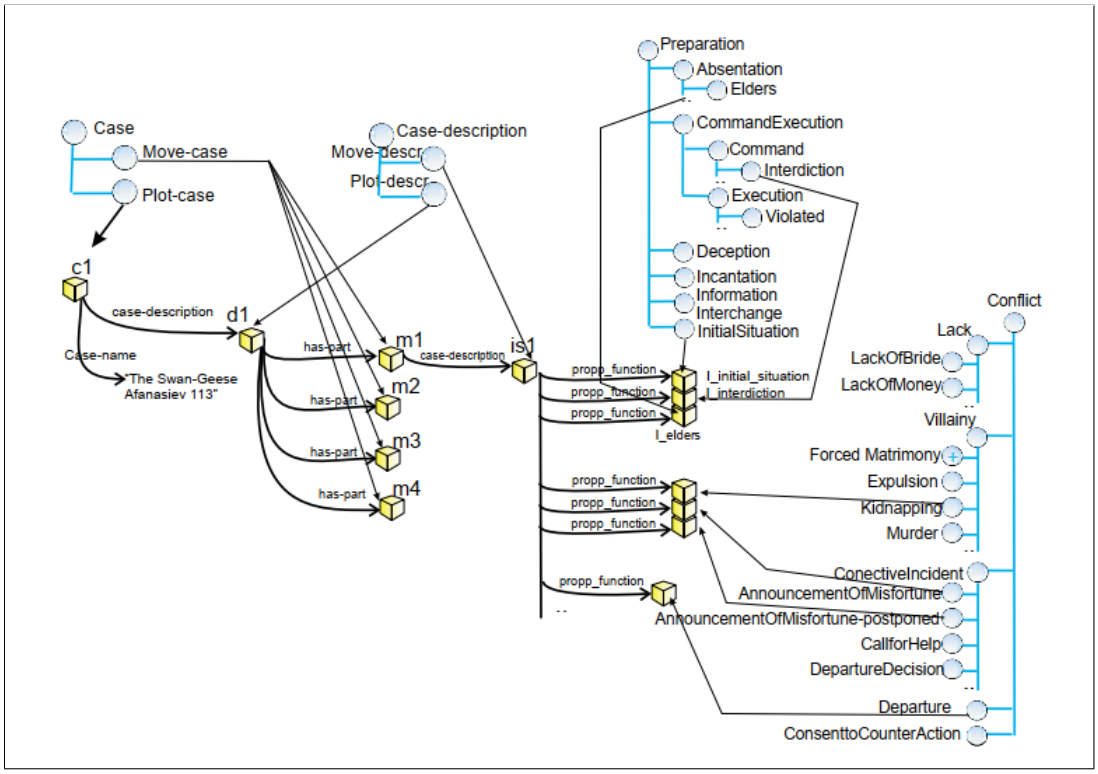

Figura 3 – Parte da ontologia utilizada em (GERVáS et al., 2005), demonstrando os conceitos (circulares) e as instâncias em uma narrativa (cubos)

que essas narrativas interativas são classificadas como *narrativas emergentes* (RIEDL; BULITKO, 2013).

Modelo de jogador. Sistemas narrativos podem tentar analisar como seu usuário interage com a história, guiando a experiência narrativa para melhor se adequar ao usuário. Este processo envolve recolher as decisões tomadas durante a narrativa e armazenar em uma representação estruturada do jogador, e com parâmetros definidos para o domínio, apresentar fluxos de história considerados mais interessantes (RIEDL; BULITKO, 2013).

Em outro estudo sistemático da literatura, Kybartas e Bidarra (2017) analisam diversos sistemas propostos de geração de narrativa interativa e os categorizam em um mapeamento bidimensional, discretizando graus de automação de *espaço* e *enredo*. Para eles, Espaço inclui os personagens, ambientes, *props* e qualquer outra coisa que existe na narrativa, também chamados de *existentes*. E Enredo é o conjunto de eventos organizados em alguma estrutura, com relações causais entre si. A figura 4 apresenta este mapeamento proposto.

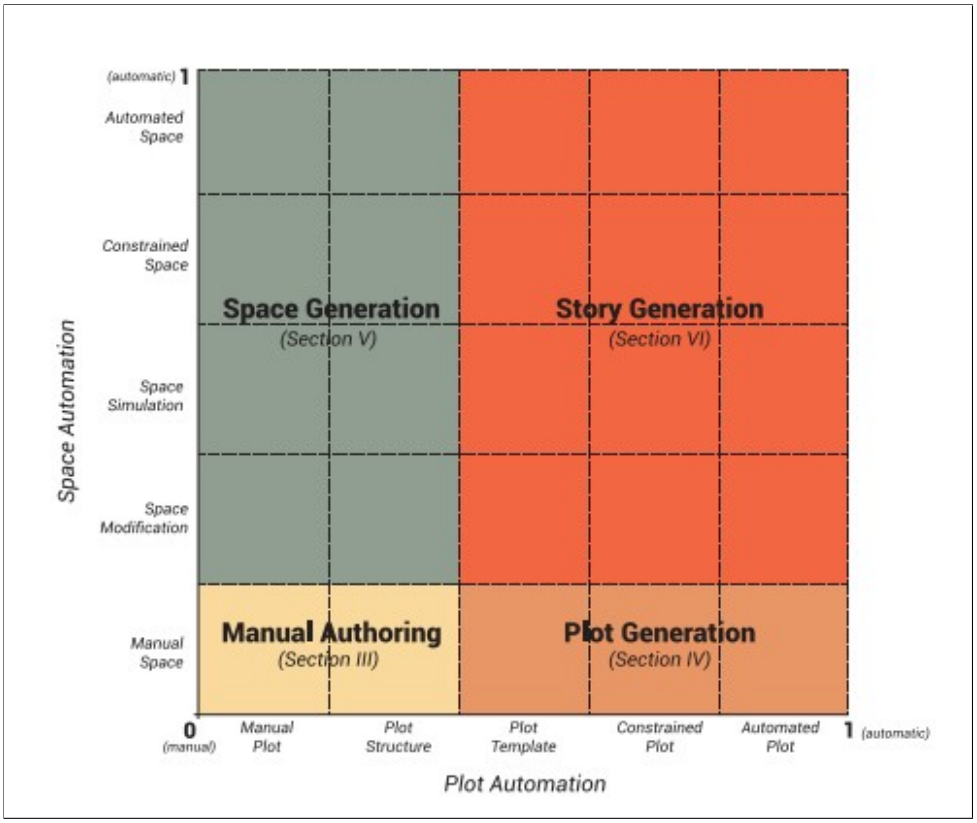

Figura 4 – Mapeamento bidimensional dos graus de automação de espaço e enredo (KYBARTAS; BIDARRA, 2017).

São delineadas regiões que categorizam os sistemas que se encaixam nelas. *Manual Authoring* representa sistemas que não realizam esforços para automatizar o processo de criação das narrativas, indo no máximo até facilitar o trabalho do autor com possíveis auxílios visuais, edição avançada, entre outros. Em *Plot Generation* se encaixam sistemas que possuem inteligência para automatizar a produção do enredo da história – seja criando modelos do enredo ou criando conteúdo restringido à definições do autor – mas que deixa toda a criação do espaço para o autor. Já os sistemas classificados como *Space Generation* fazem o contrário, automatizando a criação de existentes da história, porém não possuem inteligência para integrá-los ao enredo. Por fim, *Story Generation* é atribuído a sistemas que realizam algum grau de automação em ambos os eixos, sendo capaz de ao menos realizar modificações no espaço e ligá-las ao enredo que está sendo produzido (KYBARTAS; BIDARRA, 2017).

Estas categorizações são feitas através da discretização em cinco níveis de automação de enredo e espaço. O *enredo* é particionado nos seguintes cinto níveis.

1. *Manual*: Sistemas de enredo manual não automatizam a produção de enredo. Toda estrutura e conteúdo do enredo são manualmente criados pelo autor.
2. *Plot Structure*: O sistema provê alguma estrutura para facilitar a criação do enredo, mas o autor permanece manualmente definindo o conteúdo.
3. *Plot Template*: Sistemas deste tipo provem um modelo completo do enredo, mas não atribuem o conteúdo do espaço ao modelo definido, sendo tarefa do autor realizar a atribuição dos existentes aos eventos.
4. *Constrained Plot*: Neste caso, o sistema gera todo o enredo, incluindo atribuir os existentes aos eventos. O sistema segue restrições do autor ou de alguma outra estrutura, mantendo assim a influência direta do autor.
5. *Automated Plot*: Sistemas com enredo automáticos minimizam ao máximo o trabalho do autor. Seguindo algum conhecimento de domínio, produzem toda o enredo sem qualquer influência de um autor.

E o *espaço* é particionado nos seguintes cinco níveis.

1. *Manual Space*: Sem qualquer auxílio inteligente, o autor deve produzir todo o conteúdo utilizado na história manualmente.
2. *Space Modification*: Atribuído a sistemas que são capazes de realizar modificação nos existentes com objetivo de auxiliar a geração do enredo. Os existentes devem ser previamente criados pelo autor.
3. *Space Simulation*: Em simulação do espaço, novo conteúdo pode ser criado a partir de outros previamente definidos.
4. *Constrained Space*: Sistemas de geração de espaço restringido é capaz de criar novos conteúdos do espaço seguindo restrições definidas pelo autor.
5. *Automated Space*: Similar ao enredo automático, sistemas com espaço automatizado não requerem envolvimento do autor e geram conteúdo de existentes seguindo apenas conhecimento do domínio.

Estas definições são interessantes para realizar uma categorização de sistemas de forma objetiva, considerando múltiplos aspectos no âmbito de geração de narrativas. Em relação ao presente trabalho, será desenvolvido um sistema que explora, deste espaço, os conceitos de *Plot Structure* a *Manual Space*.

2.1.2 Gerador de narrativa interativa

"Geração de narrativa é um processo que envolve seleção, ordenação e apresentação durante o decorrer do conteúdo narrativo" (RIEDL; YOUNG, 2006).

Em criações artísticas, como pinturas, filmes ou a produção de narrativas, pode-se avaliar quanto criativo é o produto gerado. A criatividade, porém, envolve uma série de preceitos – a ideia que é algo novo ou que é diferente de outros exemplos – e também depende de ao menos uma pessoa como audiência, na qual tais preceitos podem variar, relativo a conhecimentos prévios ou a área na qual a produção é inserida (GERVáS, 2009). No escopo deste trabalho o conceito de criatividade será limitado. Um sistema considerado criativo é aquele capaz de gerar novas narrativas a partir de conceitos abstratos, criando conteúdo não manualmente descrito pelo autor da história.

Para que um sistema criativo seja capaz de produzir novos conteúdos narrativos, este deve conter conhecimentos sobre um mundo de domínio no qual ele vai utilizar como base (RIEDL; BULITKO, 2013). Estas representações devem ser feitas abstratas o suficiente para garantir que tais conceitos representem mais do que um contexto. Desta forma, é possível produzir múltiplos variantes de uma mesma narrativa ao realizar alterações no domínio (PORTEOUS; CAVAZZA; CHARLES, 2010).

Quando se trata de representação de narrativas, várias propostas de sistemas utilizam suas próprias representações internas, definidas pelos desenvolvedores do sistemas. Isto implica em grandes variações no que se refere a como é feita estas representações. Esta discrepância dificulta uma possível troca de informações entre os esforços na área de geração de narrativas (CONCEPCIóN; GERVáS; MéNDEZ, 2017).

Por conta disso, Concepción, Gervás e Méndez (2017) propuseram um modelo comum para representação de narrativas. Este modelo, apresentado na figura 5, cria uma representação geral de narrativas para tentar englobar de forma suficiente todos os aspectos que sistemas geradores de narrativa podem utilizar.

O modelo define, assim como em (KYBARTAS; BIDARRA, 2017), que a história é composta por um enredo e um espaço. O enredo é definido por um conjunto de *cenas* que são fundamentalmente *eventos* que ocorrem em um determinado ambiente e em um determinado tempo da história. O espaço da história é formado pelo conjunto de *existentes* – personagens e objetos, – e um conjunto de cenários. Em destaque os personagens são definidos com diversas características, uma vez que são quase sempre o centro das narrativas.

Resumindo as discussões feitas até aqui, um sistema gerador de narrativas interativas criativo deve conter ao menos os seguintes módulos:

1. Base de conhecimento do domínio da história, envolvendo conceitos narrativos básicos

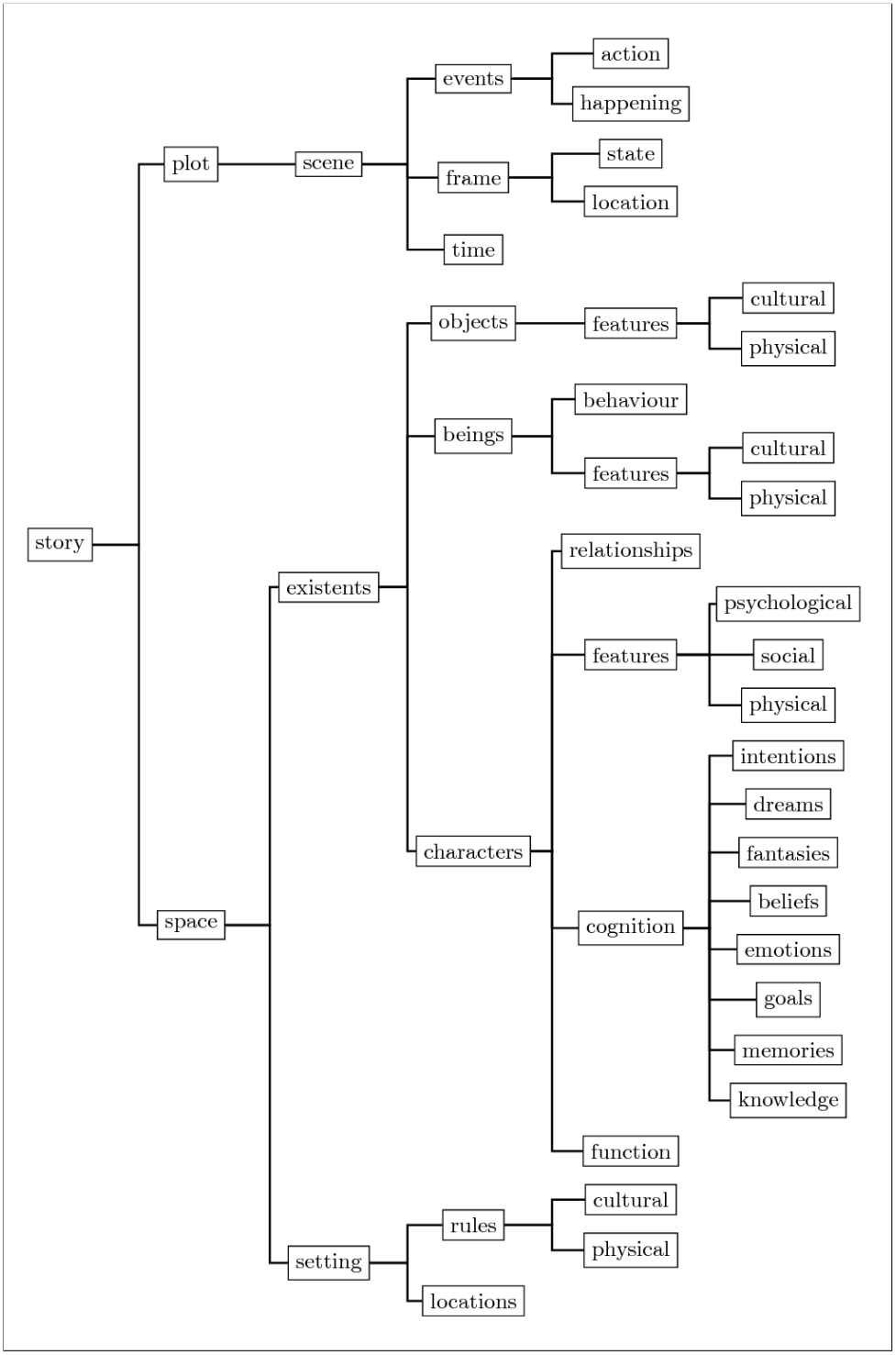

Figura 5 – Diagrama com estrutura hierárquica do modelo comum de narrativas proposto por (CONCEPCIóN; GERVáS; MéNDEZ, 2017).

e/ou de um domínio específico;

2. Representação computacional de narrativas – como gráficos de enredo –, permitindo algoritmos selecionarem caminhos no enredo com base em critérios pré-estabelecidos em tempo de projeto ou implantação;

3. Algoritmo que controla e guia o desenrolar da narrativa, realizando adaptações na história de acordo com as mudanças que ocorrem no mundo virtual;

4. Saída formalizada que conte a história, seja através de um discurso em texto, ou simplesmente eventos formalizados em mundos virtuais.

5. Interface para um autor descrever sua narrativa interativa – o conjunto de várias narrativas possíveis.

2.2 Linguagens específicas de domínio

Normalmente, autores escreveriam suas histórias em linguagem natural, uma vez que sua audiência são outros humanos e o meio no qual a história será contada é direto (como livros, teatros, etc). Porém nesta proposta as histórias descritas pelos autores terão um intermediário, o sistema responsável pela condução da narrativa. Apesar dos esforços e avanços no processamento de linguagem natural, ainda não está perto de viável um sistema processar relatos contendo noções complexas através de textos (ou voz) em linguagem natural.

Ainda que o sistema possua responsabilidade e autonomia criativa nesta proposta, é necessário que um humano descreva o enredo principal da história, incluindo personagens, eventos factíveis, entre outros. Esta etapa pode ser vista como a preparação do gerador de narrativas. A preparação deve incorporar os conceitos do domínio que são utilizados como fundamento pelo sistema inteligente. Dado o âmbito desta abordagem, uma linguagem específica de domínio é uma possível estratégia.

Linguagens específicas de domínio, ou *domain-specific languages* (DSL), são linguagens feitas sob medida para algum domínio de aplicação, envolvendo notações e construtos específicos do domínio (MERNIK; HEERING; SLOANE, 2005). DSLs são menos genéricas que linguagens de propósito geral (como C, Java e C#), mas possuem maior expressividade em seu domínio limitado. De acordo com Mernik, Heering e Sloane (2005), DSLs provêm um conjunto significativo de benefícios como:

- Notações mais apropriadas ao domínio;
- Construtos e abstrações são diretamente mapeados às representações básicas da linguagem;
- Facilidade em análise, verificação e otimização nos termos da linguagem;
- Compreensão facilitada e, dependendo do caso, um menor nível de proficiência é exigido dos usuários, abrindo o escopo para uma maior comunidade.

Por conta de tais vantagens, e considerando que a escrita textual é quase inerente às produções narrativas, será utilizado neste trabalho uma DSL como interface na qual os autores criarão histórias.

2.3 Motor de jogos

Narrativas interativas são criadas em abundância para jogos digitais, uma vez que dar ao jogador o poder de conduzir a história no caminho que mais lhe interesse é muito positivo em termos de qualidade de experiência. Além disso, consequência significativas de acordo com as ações, levando a caminhos e finais alternativos de uma mesma história garante um maior valor de "rejogabilidade".

Considerando que uma das maiores contribuições de um sistema inteligente criativo é possibilitar experiências inovadoras e um engajamento de usuários mais profundo em jogos digitais, será abordada uma estratégia para integração do sistema proposto com um motor de jogos.

Um motor de jogos, também conhecido como *engine*, é um *software* composto por uma coleção de módulos que servem como base na criação de uma grande variedade de jogos. O motor inclui módulos como: módulo para lidar com entrada, envolvendo abstrações para teclado, *mouse*, *game-pads*, entre outros; módulo para exibir saída, como renderização 3D, exibir 2D e sons; e módulo para simular física ou dinâmicas genéricas (LEWIS; JACOBSON, 2002). A figura 6 apresenta a arquitetura básica de um motor de jogos.

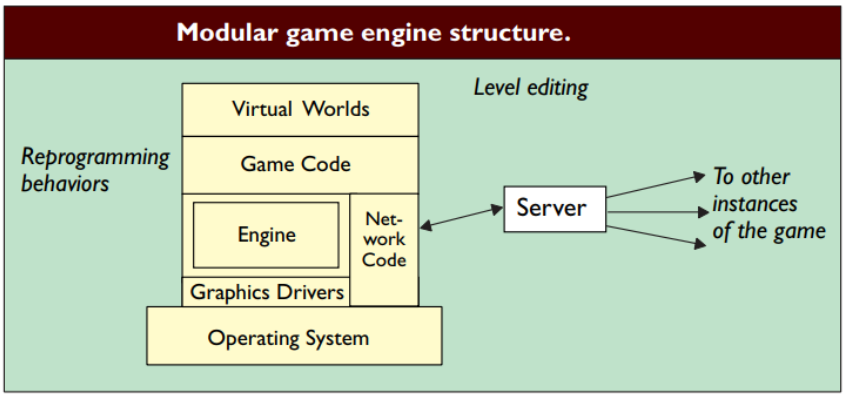

Figura 6 – Estrutura modular de motor de jogos (LEWIS; JACOBSON, 2002)

No topo estão os *virtual worlds*, ou mundos virtuais, que compõem os cenários e estabelecem a forma de interação do jogador. *Game code* é a lógica do jogo, composta por um conjunto de mecânicas básicas e lógicas para controle de exibição do conteúdo, definidos pelo desenvolvedor do jogo. O módulo *Engine* é o núcleo da estrutura, abstraindo toda a complexidade recorrente e necessária para a execução de um jogo, como lidar com *drivers* gráficos e de som, simulação de física, entre outros.

Quanto à integração entre o sistema narrativo e um motor de jogos, Young (2003) utiliza um controle direto envolvendo os componentes básicos da *engine*. No caso, é apresentado um módulo do controlador da narrativa que tem autoridade de realizar chamadas de funções do motor diretamente.

Nesta abordagem, porém, a integração é considerada de forma a possivelmente abranger múltiplos motores de jogos. A estratégia envolve limitar a influência do sistema narrativo a camada de lógica do jogo, disponibilizando uma interface de comunicação, na qual serão expostas o estado da história e ações possíveis. Desta forma, o sistema narrativo previamente moldado de acordo com um autor expõe o estado do mundo a aplicação que utiliza o motor de jogos, assim como as possíveis ações do jogador.

O desenvolver de uma narrativa neste esquema proposto envolve repetidamente a exposição do estado da história, processar a ação do jogador, alterar o estado da história e novamente expor o estado, encerrando quando a narrativa chegar explicitamente a um fim. Para validar a integração, será desenvolvido uma aplicação simples que seja capaz de demonstrar este fluxo, utilizando o motor de jogos Unity[2]. O sistema narrativo será um módulo do jogo escrito em C#, compilado a partir das definições feitas pelo autor na DSL.

[2] Motor de jogos multi-plataforma, disponível em <https://unity.com/>

3 Arquitetura do Sistema

O sistema proposto neste trabalho constitui-se de um ambiente de produção especializado na construção de narrativas interativas, que inclui a geração de um artefato capaz de gerenciar e dirigir toda e qualquer história produzida neste ambiente. A figura 7 apresenta a estrutura geral deste ambiente.

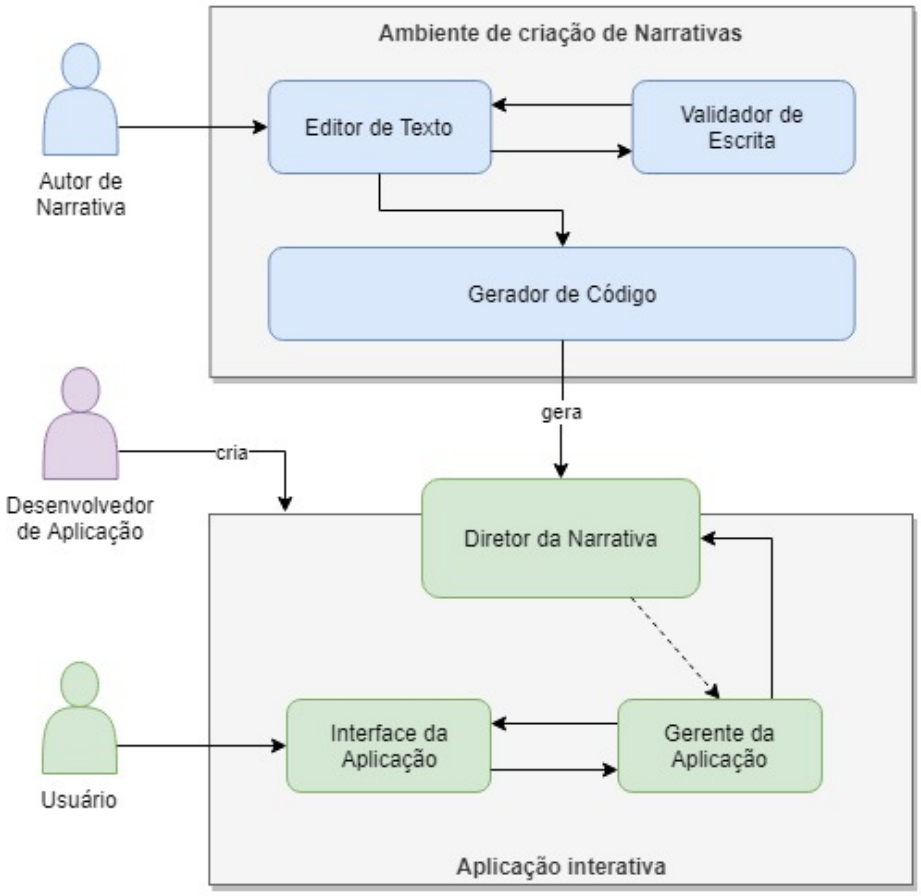

Figura 7 – Estrutura geral do ambiente de criação de narrativas interativas, incluindo todos os atores envolvidos desde a elaboração da história até a interação com um usuário.

O ambiente determina a existência de ferramentas que possibilitem uma escrita especializada para a criação de narrativas interativas. Para tal, é estabelecida uma linguagem específica de domínio (DSL) – incluindo sintaxe, validação de escrita e geração de código – que permite a elaboração de histórias da forma pretendida. Esta linguagem foi nomeada como STGEN e é abordada em mais detalhes na seção 3.1.

No ambiente de criação, um autor de histórias interativas, e especialista da linguagem, é auxiliado e guiado a descrever sua história uma vez que será limitado a escrevê-la

no formato da DSL STGEN. Sendo escrita uma narrativa neste formato, gera-se o código do Diretor da Narrativa (DN), que contém toda a estrutura de dados e procedimentos necessários para conduzir a narrativa interativa. O DN é especificado na seção 3.2.

Uma linguagem nova implica em uma nova forma de descrever histórias. Novos métodos e padrões de escrita requerem uma especialização a mais para um autor utilizar esta ferramenta. Uma breve explicação de como escrever em STGEN, assim como alguns padrões básicos, estão descritos na seção 3.3.

Requer-se ainda a existência de um aplicativo que fará o intermédio entre um usuário e o DN. O aplicativo é responsável pela forma de interação entre o usuário e a história e deve ser desenvolvida de forma apropriada aos paradigmas da linguagem STGEN. O aplicativo é descrito na seção 3.4.

Por fim, com um aplicativo interativo especializado desenvolvido e um DN gerado a partir de uma narrativa descrita em STGEN, é possível um usuário vivenciar a história produzida.

3.1 A linguagem específica de domínio STGEN

A linguagem específica de domínio (DSL) STGEN é a linguagem que apoia o ambiente de criação de narrativas e é a forma na qual autores vão descrever histórias. Sua estrutura sintática e semântica é inspirada no modelo comum de descrição de narrativas, proposto em (CONCEPCIóN; GERVáS; MéNDEZ, 2017). Esta seção vai apresentar o projeto desta linguagem.

3.1.1 Sintaxe

A figura 8 mostra um fragmento simplificado do metamodelo que representa a sintaxe, contendo os construtos fundamentais da linguagem para construção de uma narrativa interativa.

A definição de uma história é mapeada pelo recipiente superior da sintaxe, representado pelo objeto *Story*, que é composto por um *Plot*, um *Space* e um *StoryData*. O objeto *StoryData* representa dados gerais da história, sendo composto por uma coleção de atributos (*Attribute*).

O *Space* representa o espaço de conteúdo que a história pode fazer uso, isto é, tudo aquilo que existe no mundo desta narrativa. Ele é composto por um conjunto de *Existent*, ou existentes, que podem ser *Actors* ou *Objects* – atores ou objetos, respectivamente. Atores e objetos são existentes similares porém atores podem executar ações que alteram o estado da história, enquanto objetos não. Além disso, deve haver ao menos um ator que seja demarcado como o "*player*", que representa o usuário que irá interagir com a

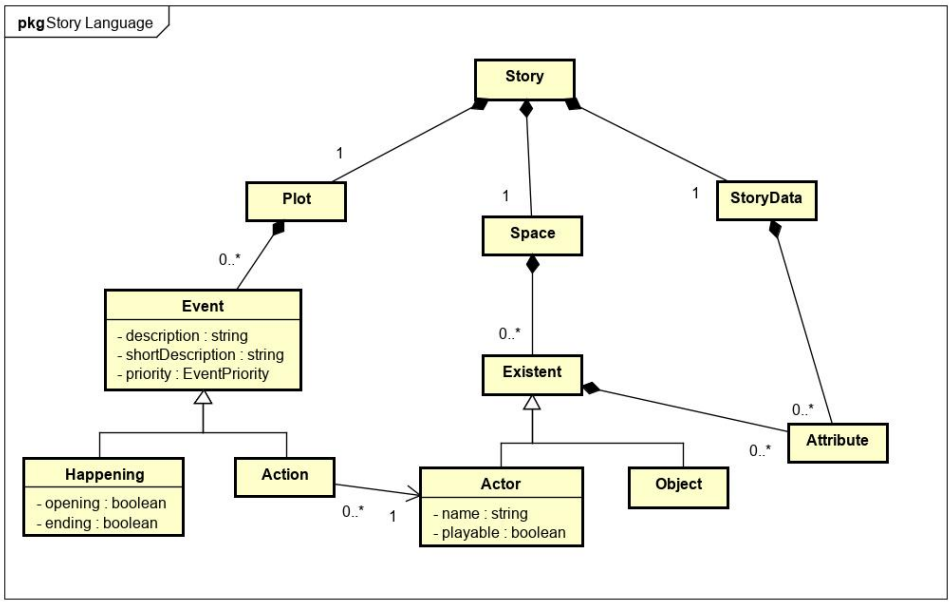

Figura 8 – Diagrama revelando um fragmento da linguagem STGEN, que destaca estrutura fundamental da sintaxe.

narrativa. Existentes possuem um conjunto de atributos que os constituem.

O objeto *Plot*, ou enredo, é um conjunto de possíveis *Events*, ou eventos, que podem ocorrer ao longo da narrativa. Eventos podem ser do tipo *Happening* – um acontecimento qualquer na história – ou *Action* – uma ação que deve ser realizada por um ator. Devem haver dois eventos específicos no enredo: uma abertura e ao menos um encerramento, demarcados com *opening* e *ending*, respectivamente.

Com a sintaxe destacada até aqui, já podemos descrever uma história mínima, isto é, uma narrativa escrita em STGEN utilizando apenas o mínimo exigido pela linguagem. Esta composição mínima está descrita no código 3.1.

Este exemplo simples não pode ainda ser classificado como uma narrativa interativa, uma vez que não existe possibilidade de interação. A interatividade com a história, assim como a forma com que as ações do jogador afetam a narrativa, são descritos na linguagem através da construção completa de ações e eventos, incluindo suas condições e alterações respectivas. A figura 9 mostra outro fragmento da STGEN, destacando estes aspectos.

Eventos ditam o desenvolvimento do enredo e alteram o estado do mundo da narrativa através de modificações diretas ao atributos da história ou de atributos de existentes. Essas modificações são mapeadas pelo objeto *Change*. Um evento pode ter nenhuma ou várias modificações, sendo que cada uma dela afeta um atributo de um existente ou da história.

Para que um evento possa acontecer, deve-se satisfazer todas as sua condições. Cada condição de um evento é descrita por um *Condition*, que pode ser de três tipos: *OrderCondi-*

Código 3.1 – Composição mínima de história na sintaxe STGEN.

```
Story data
{
}

Space
{
  player actor Jogador {
    name: "Jogador"
  }

}

Plot
{
  opening event Abertura {
    description: "Era uma vez..."
  }

  ending event Encerramento {
    description: "... e viveram felizes para sempre."
  }
}
```

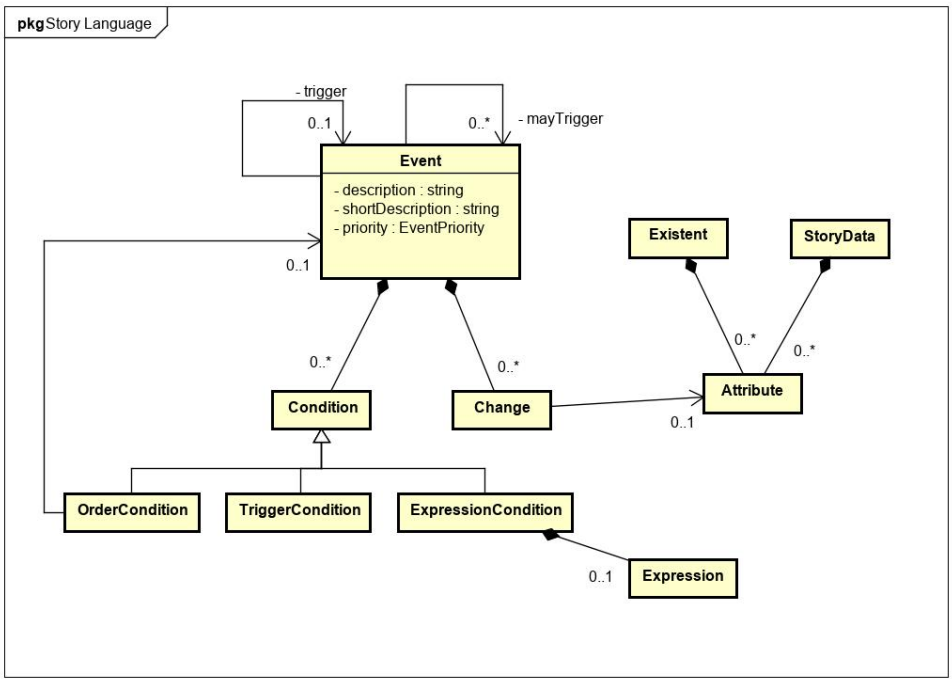

Figura 9 – Diagrama contendo fragmento da linguagem STGEN, destacando a sintaxe dos eventos e suas relações com outros elementos da lingagem.

tion, *TriggerCondition* ou *ExpressionCondition*. *OrderCondition* é uma condição temporal que verifica se um dado outro evento já aconteceu ou não na história. *TriggerCondition* significa que o evento só pode ocorrer caso seja explicitamente desencadeado a partir de outro evento. E o *ExpressionCondition* representa uma expressão boleana que faz uso dos atributos dos existentes da história. Cada uma dessas condições pode ser vista como uma função boleana, e para verificar se um evento pode acontecer em um determinado

momento da história, verifica-se se cada uma das condições do evento são verdadeiras. A lógica de execução de eventos e progresso da história é abordada em mais detalhes na seção 3.2.

Além de modificações e condições, eventos podem possuir desencadeadores – a saber, *Trigger* e *MayTrigger*. *Trigger* faz referência a um outro evento e que garantidamente este evento será efetivado na história caso o primeiro aconteça, isto é, um evento desencadeia outro assim que ocorre na narrativa. *MayTrigger*, por sua vez, faz referência a um conjunto de outros eventos e, quando o primeiro ocorre, o conjunto em questão é avaliado e estes podem ser desencadeados também. Neste caso não é certo que os eventos ocorrerão, pois suas condições devem ser satisfeitas.

O código 3.2 contém a escrita de um evento fazendo uso dos parâmetros descritos acima.

Código 3.2 – Exemplo de um evento complexo em STGEN.

```
1  event PonteBamba {
2      description: "A ponte é fraca e começa a balançar freneticamente!"
3      short-description: "Ponte balança preocupantemente."
4
5      triggers: ReagirPonteBambaInstintivamente
6      may-trigger: PonteCai, PonteEstabilizou
7
8      require: Jogador.peso > 80
9      require: after SubiuNaPonte
10     require: be-triggered
11
12     change: Story.podeAcabar = true
13     change: Jogador.medo += 10
14 }
```

Nas linhas 2 e 3 estão definidas as descrições do evento. Nas linhas 5 e 6 estão os desencadeadores *Trigger* e *MayTrigger*, respectivamente. Nas linhas 8 a 10 estão exemplos dos três tipos de condição de eventos – na 8 usa-se um *ExpressionCondition*, avaliando o estado atual da história; na linha 9 há um *OrderCondition* onde é verificado se o evento "SubiuNaPonte" já ocorreu; e na linha 10 o *TriggerCondition* que estabelece que esse evento só pode ocorrer caso seja desencadeado explicitamente por outro evento (que inclua em suas cláusulas *triggers* ou *may-trigger* esse evento). E nas linha 12 e 13 estão duas modificações, sendo a primeira uma modificação que altera um atributo boleano da história e o segundo adiciona 10 unidades em um atributo que representa o pavor do ator do Jogador.

Eventos podem ter prioridades diferentes, podendo ser atribuídos a eles as seguintes prioridades de relevância crescente: *normal*, *high* ou *veryHigh*. A influência dessas prioridades altera a lógica de seleção de eventos do Diretor da Narrativa e está bem detalhada na seção 3.2. Mas resumidamente, quando o DN está a selecionar um evento, verifica-se primeiro a maior prioridade dentre eles. Se houver alguma prioridade elevada, o algoritmo

vai selecionar apenas eventos deste nível, e tentará selecionar o máximo deles que puder – pois são importantes e devem ocorrer, – desde que permaneçam válidos durante a aplicação dos mesmos. O código 3.3 mostra como os eventos são marcados desta forma.

Código 3.3 – Eventos com prioridades diferentes em STGEN.

```
event EventoNormal {
    description: "Aconteceu algo normal."
}

event OutroEventoNormal {
    priority: normal
    description: "Como no dia-a-dia."
}

event EventoImportante {
    priority: high
    description: "Aconteceu algo importante!"
    // ...
}

event EventoMuitoImportante {
    priority: veryHigh
    description: "Aconteceu algo muito importante! Uau!"
    // ...
}
```

Em contraste com os Eventos, a composição de Existentes na história é mais simples. Os existentes em STGEN – atores ou objetos – são fundamentalmente recipientes de atributos, utilizados para organizar semanticamente o estado da narrativa. Atributos na linugagem podem ser de dois tipos: *fact* que indica valor boleano ou *quantity*, um valor inteiro. O código 3.4 apresenta um exemplo de escrita de existentes em STGEN.

Código 3.4 – Exemplo de existentes em STGEN contendo um ator e um objeto.

```
player actor Jogador {
    name: "John Doe"
    quantity comida = 0
    quantity fome = 10 [decrements by 1 every player action]
    quantity saude = 10

    fact muitaFome = (Jogador.comida <= 2)
    fact estaSofrendo = (Jogador.saude <= 2)
}

object Tocha {
    fact acesa = false
    quantity integridade = 100
}
```

A forma com que os atores e objetos são descritos revelam um pouco sobre o caminho que a história pode trilhar. O ator *Jogador* possui atributos quantitativos de comida e fome, indicando provavelmente uma preocupação na história relativa sobrevivência. Junto ao atributo de fome, está descrito "*[decrements by 1 every player action]*", uma notação de configuração de atributos, usada para facilitar sua alteração situacional. E também

dois atributos do tipo *fact*, a saber *muitaFome* e *estaSofrendo*, que usam uma expressão para determinar seu valor boleano.

O objecto Tocha é mais simples, contendo apenas informação se está acesa ou não e sobre sua integridade geral, que pode diminuir ao decorrer da narrativa. A sintaxe na composição de atributos de existentes está descrita na figura 10.

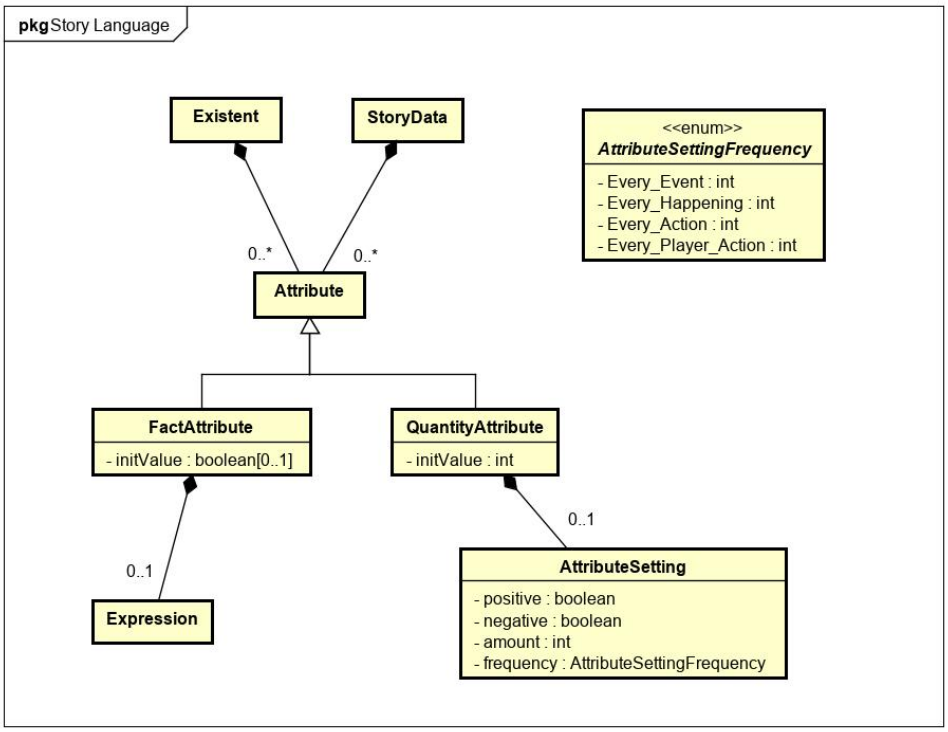

Figura 10 – Diagrama contendo fragmento da linguagem STGEN, destacando a sintaxe da descrição dos atributos.

Atributos do tipo *FactAttribute* podem ter ou um valor inicial *true* ou *false*, ou pode ter uma expressão que define seu valor boleano. Um atributo do tipo *FactAttribute* não pode ter ao mesmo tempo um valor inicial e uma expressão.

Atributos do tipo *QuantityAttribute* possuem um valor inicial e opcionalmente um *AttributeSetting*. O *AttributeSetting* representa uma configuração para automatizar um contador condicional no atributo quantitativo. Como mostrado no código 3.3, o atributo *fome* tem a configuração *[decrements by 1 every player action]*, que mapeado representa um AttributeSetting marcado para negativo, com quantidade 1 e frequência *Every_Player_Action*.

A sintaxe da linguagem STGEN está descrita por completa, incluindo detalhes e exemplos de uso, no apêndice A.

3.1.2 Validação

A sintaxe da DSL STGEN limita significantemente a forma com que um autor descreve uma narrativa, com o intuito de possibilitar a geração de código, ao impedir que ocorram disposições de palavras indevidas. Porém existem situações que não são tratadas na sintaxe em si, e que são checadas na etapa de validação da escrita. A seguir estão descritas as verificações realizadas pelo Validador da linguagem.

Deve existir um e somente um ator *player*. Qualquer ator pode ser marcado como o ator que representa o jogador. Então verifica-se dentre os atores se existe um *player* e se não existe qualquer outro além deste.

Existentes, Eventos e Atributos devem ter nomes únicos no seu contexto. Para os existentes, sejam atores ou objetos, é verificado se qualquer um não tem nome igual a qualquer outro. A mesma verificação é feita para o eventos. No caso dos atributos, verifica-se caso todos tenham nomes diferentes no contexto do recipiente – um *Existent* ou *StoryData* – no qual ele pertence. Além disso, um atributo não deve ter nome igual ao existente que o contém, caso seu recipiente seja um existente.

Existentes, Eventos e Atributos não podem ser nomeados com palavras reservadas. A princípio qualquer nome pode ser escolhido para estes elementos. Então é verificado se o nome não conflita com palavras reservadas da sintaxe. Além disso, existem nomes reservados usados no código final gerado do DN. Estes são, a saber, para os existentes, a palavra *player*; para os eventos, as palavras *opening*, *ending*, *happenings*, *actions* e *events*; e para os atributos, a palavra *___isOver*.

Deve existir um e somente um evento *opening*. Qualquer evento pode ser marcado como o evento que inicia a história. Então verifica-se dentre os eventos se existe um do tipo *opening* e se não existe qualquer outro além deste.

Deve existir ao menos um evento *ending*. Qualquer evento pode ser definido como um evento que encerra a história, mas apenas pela sintaxe, isto não é obrigatório. Então é verificado dentre os eventos se existe pelo menos um marcado como *ending*.

É incoerente definir ações do ator marcado como *player* com prioridades elevadas, como *high* ou *veryHigh*. Isto porque as ações do ator do jogador são apenas executadas pelo jogador, e nunca pelo DN. Esta verificação dispara apenas um aviso.

É incoerente definir o evento *opening* com prioridades elevadas, como *high* ou *veryHigh*, assim como ter condições. Isto porque o evento de abertura é sempre executado no início, independente de condições ou qualquer prioridade. Esta verificação dispara apenas um aviso.

É incoerente ter modificações ou desencadeadores em um evento marcado como *ending*. Isto porque após um evento de encerramento a história é encerrada,

ou seja, modificações são irrelevantes e desencadeadores não serão ativados. Esta verificação dispara apenas um aviso.

Desencadeadores de eventos não devem ter ciclos. Um evento pode possuir um desencadeamento no *Trigger* e vários outros no *MayTrigger*. Verifica-se caso um possível caminho de desencadeamento de eventos não leva de volta ao primeiro, pois isso poderia tornar-se um laço infinito.

O desencadeador *MayTrigger* de um evento não deve conter eventos duplicados. O mesmo evento ocorrendo mais de uma vez em um *MayTrigger* levaria a desencadeamento repetido do mesmo evento sempre que um fosse válido, uma vez que são idênticos. Então verifica-se todo desencadeador deste tipo para evitar tal situação.

É inconsistente existir um evento que possui o *TriggerCondition* mas nenhum evento o desencadeia. Isto porque, caso este cenário aconteça, significa que o evento nunca ocorrerá na história, uma vez que está condicionado que o evento em questão deve ser desencadeado. Esta verificação dispara apenas um aviso.

É incoerente um evento possuir um *OrderCondition* que requer que o próprio evento já tenha ocorrido. Pois, claramente, esta condição nunca será satisfeita.

E em relação as expressões de condições ou de modificações:

- Expressões de E ou OU lógico não podem ter operandos do tipo inteiro ou *QuantityAttribute*;
- Em expressões de comparação, os operandos devem ter o mesmo tipo lógico. Isto é, ambos são valores inteiros ou *QuantityAttributes*, ou ambos são valores boleanos, *FactAttributes* ou são outra expressão boleana.
- Em expressões de modificação, atributos do tipo *FactAttribute* só podem ser atribuídos, e nunca incrementados, e apenas com valores boleanos.
- Em expressões de modificação, atributos do tipo *QuantityAttribute* só podem ser modificados usando valores inteiros.

Se em qualquer uma dessas verificações for detectado um caso indevido, são alertados erros ou apenas avisos condizentes, dependendo do quão crucial é a violação. Enquanto houver algum erro, o ambiente não é capaz de gerar código.

3.1.3 Geração de código

O código gerado pelo ambiente de criação constitui o Diretor de Narrativas (DN), o produto que pode gerenciar e conduzir a narrativa previamente descrita. Este módulo de geração de código é o que mapeia cada elemento descrito pelo autor em um bloco de

código do DN. Como previamente definido, este módulo gera todo o código do DN em C#.

O processo de geração envolve uma quantidade extensa de código estático que constrói a estrutura básica e processos do DN. Estes códigos serão quase todos omitidos aqui, e discutidos indiretamente na seção 3.2, no detalhamento da estrutura de dados e lógica do DN.

Os principais processos do gerador de código do ambiente é a geração dos existentes e seus atributos e a geração dos eventos e suas configurações. Cada existente declarado pelo autor no ambiente é compilado para uma classe em C#, que então pode ser instanciada e gerenciada pelo DN. O mesmo vale para os eventos da história.

Por exemplo, o código do ator descrito em STGEN em 3.4 é mapeado para C# no código 3.5.

Código 3.5 – Exemplo de ator gerado em C# a partir de definição em STGEN.

```
1  public class Jogador : StoryActor
2  {
3    public Jogador()
4    {
5      displayName = "John Doe";
6      actions = new List<Plot.ActionEvent>() {  };
7    }
8
9    private  int comida = 0;
10   public  int Comida
11   {
12       get { return comida; }
13       set { comida = value; }
14   }
15   private  int fome = 10;
16   public  int Fome
17   {
18       get { return fome; }
19       set { fome = value; }
20   }
21   private  int saude = 10;
22   public  int Saude
23   {
24       get { return saude; }
25       set { saude = value; }
26   }
27   public  bool MuitaFome
28   {
29     get { return Story.space.jogador.Fome <=2; }
30   }
31   public  bool EstaSofrendo
32   {
33     get { return Story.space.jogador.Saude <=2; }
34   }
35 }
```

A classe gerada não é complexa, contendo apenas um conjunto de campos e propriedades que representam os atributos do ator; o nome definido ao ator e um conjunto de ações que ele pode realizar. As variáveis nas linhas 5 e 6 são declaradas na classe superior da qual o ator Jogador herda. A classe *StoryActor* é abstrata e existe, juntamente com a

classe *StoryObject*, para diferenciar programaticamente os tipos de existentes da história. O código 3.6 é o fragmento gerado que contém as definições dos tipos de existentes.

Código 3.6 – Código estático que define os tipos de existentes.

```
1  public abstract class StoryExistent
2  {
3  }
4
5  public abstract class StoryActor : StoryExistent
6  {
7      public string displayName;
8      public List<Plot.ActionEvent> actions;
9
10     public List<Plot.ActionEvent> GetPossibleActions()
11     {
12         return actions.Where(a => a.PreconditionsMet()).ToList();
13     }
14 }
15
16 public abstract class StoryObject : StoryExistent
17 {
18     public abstract IDictionary<string, string> GetAttributes();
19 }
```

No caso dos eventos, para exemplificar, gera-se o código 3.7 a partir do evento descrito em STGEN no código 3.2.

Nas linhas 3 a 10 estão os dados básicos do evento, nome, descrições e indicadores boleanos para indicar a prioridade do evento. Nas linhas 12 a 17 estão, como o nome sugere, a solidificação de referências a outros eventos, constituindo os desencadeadores do evento. Depois, de 19 a 26 está o método que indica, quando chamado, se o evento é válido ou não no momento. Por último, na linha 28 está o método que aplica as modificações do evento.

Assim como no exemplo anterior, existem variáveis que estão definidas nas classes superiores deste evento gerado. E também existem duas classes abstratas para diferenciar o tipos de eventos, a saber, *HappeningEvent* e *ActionEvent*. O código 3.8 contém em C# as definições geradas destes tipos de eventos.

3.2 Diretor da Narrativa

Após escrita uma narrativa em STGEN, é gerado o código em C# do Diretor da Narrativa (DN). Este contém de forma estruturada todos os existentes, eventos e regras descritas pelo autor, e uma lógica de execução pré-definida. A figura 11 apresenta a estrutura geral do DN.

O Diretor da Narrativa disponibiliza uma interface de comunicação através do acesso a métodos e propriedades da história. Essa interface, na figura, é representada pelo objeto *Story*. O *Selector* é o módulo responsável pela decisão dos próximos eventos que

Código 3.7 – Exemplo de evento gerado em C# a partir de definição em STGEN.

```
1  public class PonteBamba : HappeningEvent
2  {
3      public PonteBamba()
4      {
5          name = "PonteBamba";
6          description = "A ponte é fraca e começa a balançar freneticamente! Será
               que vai aguentar o peso?";
7          shortDescription = "Ponte balança preocupantemente.";
8          priority = EventPriority.Normal;
9      }
10
11     public override void SetReferences()
12     {
13         eventToTrigger = Story.plot.reagirPonteBambaInstintivamente;
14
15         eventsThatMayTrigger = new List<Plot.Event>() { Story.plot.ponteCai,
               Story.plot.ponteEstabilizou,  };
16     }
17
18     public override bool PreconditionsMet(bool isTriggerCheck = false)
19     {
20         if(!isTriggerCheck) return false;
21
22         return ((Story.space.jogador.Peso > 80) &&
23         (Story.HasEventOcurred(Story.plot.jogadorSubiuNaPonte)) &&
24         (true) && true);
25     }
26
27     public override void ApplyChanges()
28     {
29         Story.PodeAcabar = true;
30         Story.space.jogador.Medo += 10;
31     }
32 }
```

vão ocorrer na narrativa. Ele contém vários métodos pré-definidos e uma lógica estática, estabelecida no projeto da *framework*. Os objetos *Plot* e *Space* são os conjuntos de dados acumulados pela descrição do autor na DSL STGEN. Estes contém todos os eventos e existentes, respectivamente, descritos pelo autor da narrativa, incluindo suas regras e atributos.

3.2.1 Estrutura dos dados

Resumidamente, o que diferencia uma narrativa de outra é o conteúdo que a compõe e a ordem na qual este é apresentado. No processo de criação de narrativas STGEN, o grande variante entre os produtos gerados está na coleção de dados da história – eventos, atores, etc. Por tanto, essas informações devem ser armazenadas de forma clara e eficiente. A figura 12 apresenta a organização hierárquica dos dados de um narrativa gerada.

O *Plot* e o *Space* gerados são classes que funcionam como recipientes, agrupando dados que os pertencem de forma semântica. Qualquer evento da narrativa faz parte do enredo e por isto estão dentro do *Plot*, e o mesmo vale para os existentes – atores e objetos – com o *Space*. Ambos são semelhantes em sua organização interna, possuindo coleções de

Código 3.8 – Código estático que define os tipos de eventos.

```
1  public abstract class Event
2  {
3    public string name;
4    public string description;
5    public string shortDescription;
6
7    public EventPriority priority;
8
9    public Plot.Event eventToTrigger;
10   public List<Plot.Event> eventsThatMayTrigger;
11
12   public abstract bool PreconditionsMet(bool isTriggerCheck = false);
13   public abstract void ApplyChanges();
14   public abstract void SetReferences();
15
16     public override bool Equals(object obj)
17     {
18         if (obj == null || GetType() != obj.GetType())
19         {
20             return false;
21         }
22
23         return (obj as Event).name == this.name;
24     }
25
26     public override int GetHashCode()
27     {
28         return base.GetHashCode();
29     }
30 }
31
32 public abstract class HappeningEvent : Event
33 {
34 }
35
36 public abstract class ActionEvent : Event
37 {
38     public ActionType type;
39     public Space.StoryActor actor;
40 }
```

dados e referências a essas coleções, individualmente e em grupos semânticos.

O *Plot* armazena todos os eventos definidos pelo autor. Para cada Acontecimento ou Ação definida durante a descrição da história, um novo objeto é gerado no enredo do DN. Isto está representado na figura com a sequência de *Actions* e *Happenings*. Todos podem ser acessados individualmente ou através de coleções que os agrupam. Existem as coleções *Events*, armazenando todos os eventos; *Actions* que detém todos os eventos do tipo ação; e *Happenings*, contendo todos os eventos do tipo acontecimento. Além desses, existem os eventos especiais – a abertura e os encerramentos possíveis da história. Para estes existem referências que os apontam, sendo elas o *Opening* e a coleção *Endings*, respectivamente.

O *Space* possui estrutura similar, contendo todos os existentes definidos. Nele existe um objeto para cada ator e objeto descrito em STGEN, representado pela sequência de *Actor* e *Object* na figura. Também possui coleções que referenciam estes objetos, a saber, *Existents*, armazenando todos os existentes; *Actors* que contém todos os atores; e *Objects*,

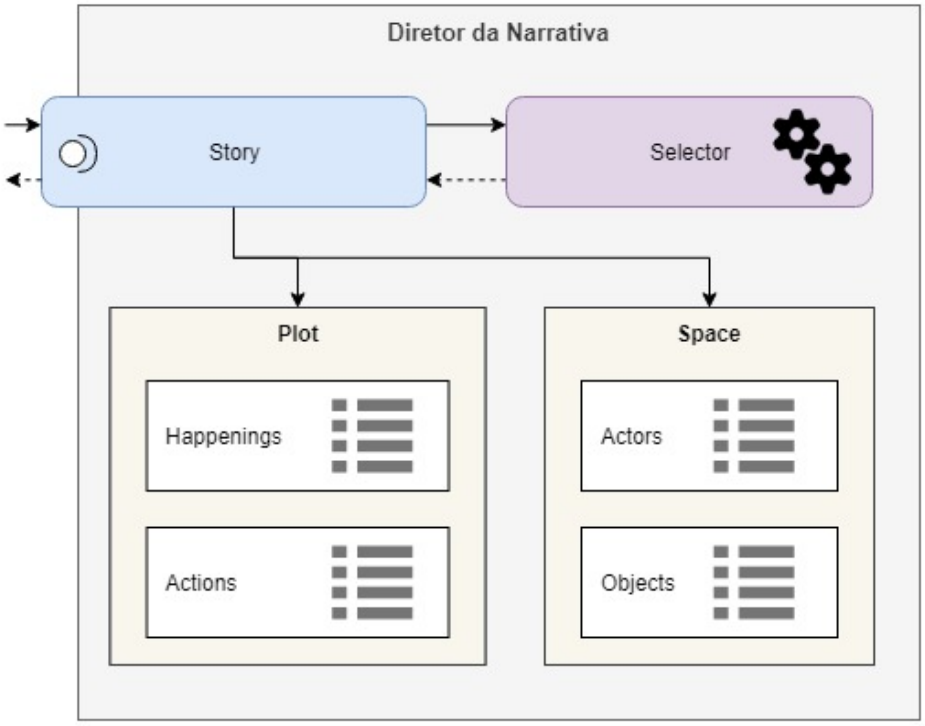

Figura 11 – Estrutura geral do Diretor de Narrativa.

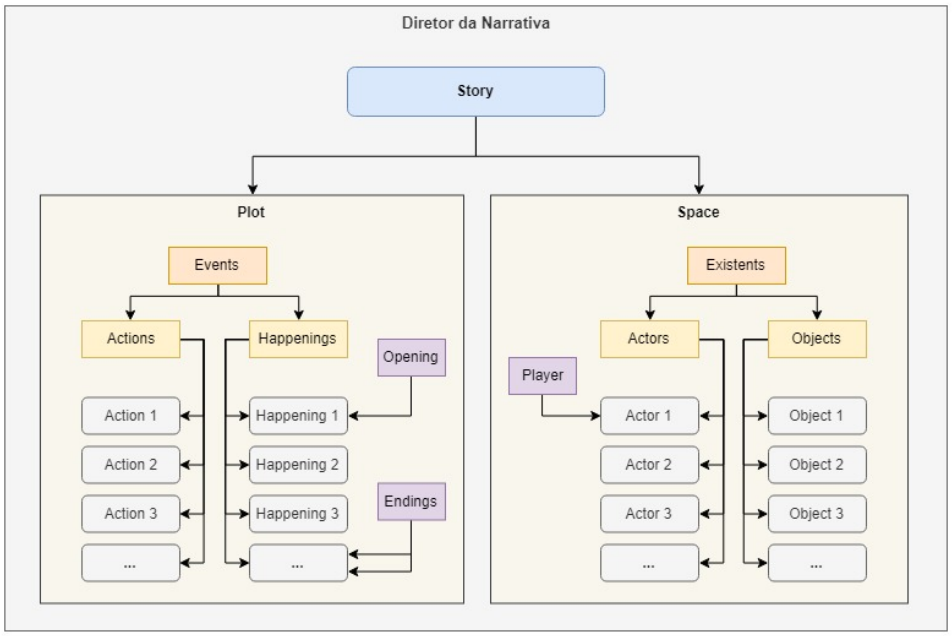

Figura 12 – Organização hierárquica dos dados de uma narrativa no Diretor da Narrativa.

o conjunto de todos os objetos. Em especial no *Space* está a referência nomeada *Player*, que aponta para o ator que foi definido como o "jogador" da história.

O DN é composto por um conjunto de classes fixas – que formam a base das especificações do sistema, como apresentadas nos códigos 3.6 e 3.8 – e classes geradas de acordo com as definições em STGEN. Além destas, existe uma classe estática (a classe

Story), que serve como recipiente de todas as outras e como interface para os aplicativos. O código 3.9 apresenta uma parte da classe *Story* e seus membros.

Código 3.9 – Fragmento da classe *Story* contendo alguns de seus membros.

```
1  public static class Story
2  {
3      public static Space space;
4      public static Plot plot;
5      private static Selector selector;
6
7      private static int ato = 1;
8      public static int Ato
9      {
10         get { return ato; }
11         set { ato = value; }
12     }
13     private static int relogio = 0;
14     public static int Relogio
15     {
16         get { return relogio; }
17         set { relogio = value; }
18     }
19
20     // More story attributes...
21     // ...
22
23     public static bool __isOver;
24     public static List<Plot.Event> pastEvents;
25
26
27     public static void Init() { /* ... */ }
28
29     // More Methods...
30     // ...
31 }
```

Os campos *space*, *plot* e *selector* fazem referência a objetos instanciados das classes respectivas uma vez que o método *Init* é invocado. Diferentemente dos primeiros, a classe do *selector* é fixa, pois contém apenas os procedimentos de seleção dos eventos. Neste exemplo, em STGEN foram definidos dois atributos em *Story data*, resultando nestes dois pares de campo e propriedade, a saber, *ato* e *Ato* e *relogio* e *Relogio*.

As classes *Space* e *Plot* possuem estrutura similar, contendo campos com referências a objetos que são instanciados a partir de classes – neste caso, classes geradas de acordo com a história criada em STGEN. Por exemplo, o código 3.10 contém uma parte da classe *Plot*.

Neste exemplo, foram definidos apenas cinco eventos em STGEN – três acontecimento e duas ações. Para cada evento definido na descrição da história, é gerado uma classe que o representa, e também um campo que conterá sua instância durante a execução da narrativa. Isso permite um fácil controle do estado de cada evento (ou existente, no caso do *Space*), uma vez que altera-se apenas a instância e isto é um comportamento esperado. Se houvessem mais acontecimento, existiram mais campos que os representam depois da linha 10. O mesmo vale pra ações, cujos campos viriam depois da linha 14.

Código 3.10 – Fragmento da classe *Plot* contendo alguns de seus membros.

```
1  public class Plot
2  {
3      // Plot opening & endings
4      public HappeningEvent opening;
5      public List<HappeningEvent> endings;
6
7      // Happenings
8      public HappeningEvent inicio;
9      public HappeningEvent ponteBamba;
10     public HappeningEvent encerramento;
11
12     // Actions
13     public ActionEvent jogadorSubirNaPonte;
14     public ActionEvent jogadorCorrerPelaPonte;
15
16     // Collections of Events
17     public List<Event> events;
18     public List<HappeningEvent> happenings;
19     public List<ActionEvent> actions;
20
21     // Constructor and Methods...
22     // ...
23 }
```

Os campos das linhas 4–5 e 17–19 sempre existem e representam coleções de eventos e o evento de abertura da história. Todos estes campos, incluindo as coleções, são populados no construtor da classe *Plot*. Este, por sua vez, é chamado no método *Init* da classe *Story*.

3.2.2 Lógica e processos

O Diretor da Narrativa, além de conter os dados da história, possui também a lógica para guiar a narrativa. Tendo como prioridade a interatividade, o processo de condução da narrativa gira em torno das ações que o jogador realiza. Inicialmente, abre-se a história com uma sequência de eventos. Sequência essa que é sempre a mesma para uma determinada história, pois é determinado na STGEN que existe apenas um evento de abertura, o *opening*. Após isto, entra-se num laço onde o DN espera pela ação do jogador antes de prosseguir escolhendo mais eventos e então espera novamente por uma ação externa. Esta lógica fundamental está descrita diagramaticamente na figura 13.

Durante o laço narrativo, não acontecem apenas ações do jogador. Como visto anteriormente, podem ser declarados múltiplos atores em uma história, e cada um pode ter múltiplas ações disponíveis para executar. Além disso, existem também os eventos do tipo Acontecimento, que não dependem de um ator para ocorrer na história. Estes eventos – ações de outros atores e acontecimentos – podem ocorrer a cada laço do algoritmo do DN. Há restrição na ordem de ocorrência destes:

1. Aplica-se a ação do jogador e todo evento que essa ação desencadeia, caso exista alguma;

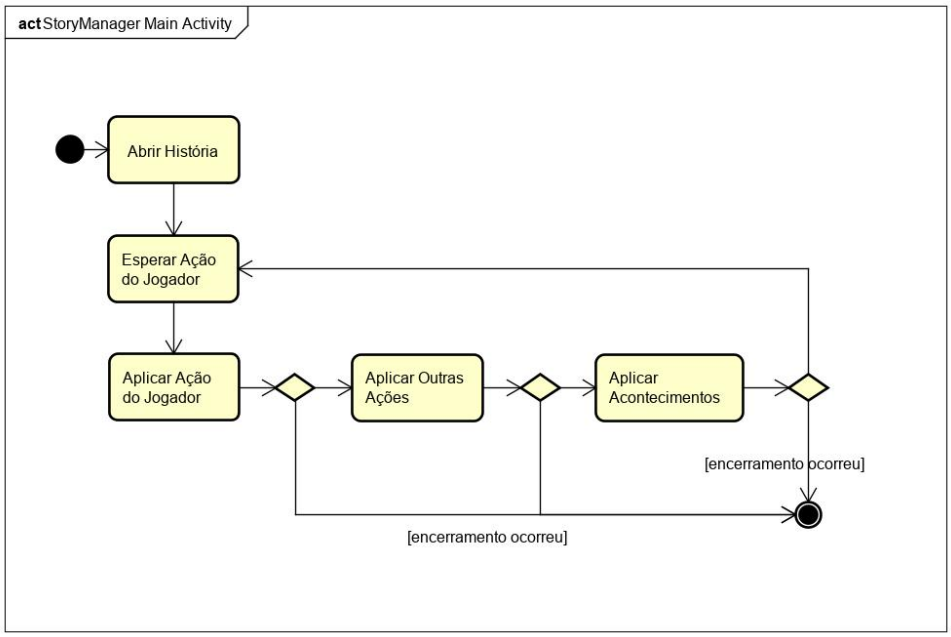

Figura 13 – Diagrama descrevendo a lógica fundamental do Diretor da Narrativa.

2. Para cada outro ator da história, verifica-se sua ações. Então são executadas as ações disponíveis, se houver alguma, e todos os eventos que são desencadeados por elas;

3. São analisados os acontecimentos da história. Se houver ao menos algum válido para o contexto atual, eles são executados.

Na etapa 1, a aplicação da ação do jogador, sempre é executado apenas uma ação por laço, pois o jogador é limitado a escolher uma ação por ponto de decisão da narrativa. Nas etapas 2 e 3 – execução de outras ações e acontecimentos – o DN verifica os eventos válidos no contexto e decide quais executar. Neste procedimento, normalmente apenas um evento válido é selecionado para ocorrer, e esta seleção é feita arbitrariamente. Este é o comportamento padrão do DN quando todos os eventos possuem prioridade *normal*. Como apresentado na seção 3.1.1, eventos podem ter prioridades elevadas como *high* ou *veryHigh*. A presença de eventos válidos que tenham prioridades elevadas alteram a lógica padrão de seleção de eventos como mostra a figura 14.

Portanto temos que, em todo momento de seleção de eventos:

- Se todos os eventos válidos são de prioridade *normal*, selecione um deles aleatoriamente;
- Se existe algum evento válido com prioridade *high* e nenhum *veryHigh*, selecione todos os eventos de prioridade *high*, e apenas estes;
- Se existir algum um evento válido com prioridade *veryHigh*, selecione todos os eventos deste nível, e apenas estes.

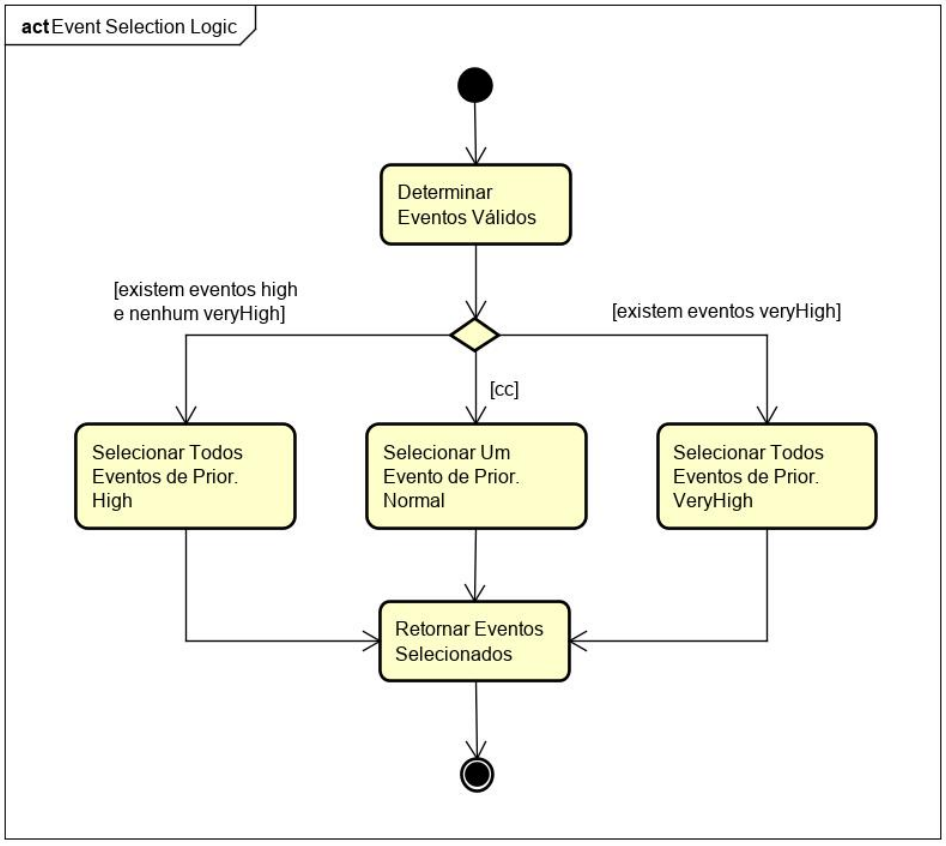

Figura 14 – Procedimento de seleção de eventos a ocorrerem do Diretor da Narrativa.

Assim sendo, o algoritmo de seleção de eventos pode selecionar múltiplos eventos a serem aplicados na história. Porém, pode ser que um destes eventos selecionados invalide outro. Ou seja, é possível que um evento seja selecionado nesta etapa de seleção e não executado na etapa de aplicação dos eventos, caso algum evento o invalide durante a execução dos múltiplos eventos. Nesta seleção, a ordem dos eventos não é previsível – a sequência de eventos selecionados de uma das prioridades elevadas, – e por isto um autor não deve depender desta ordenação.

A etapa de aplicação dos eventos selecionados é simples. Se houver apenas um evento selecionado, este é aplicado. Se houver mais de um, então para cada evento, verifica-se se ele ainda é válido e, caso positivo, é aplicado. Além disto, para cada evento também é verificado caso um deles "encerrou a história". E neste caso, o laço é interrompido.

O comportamento padrão do DN – seleção arbitrária de eventos *normais* – existe para facilmente possibilitar diferentes sequências de eventos no enredo da mesma narrativa. Porém esta incerteza nem sempre deve ser empregada, pois normalmente existem momentos que devem ter certa linearidade ou então porque certo evento não pode não ocorrer em uma dada situação narrativa. Por isto, existem as prioridades elevadas – *high* e *veryHigh*, – que garantem que o evento ocorrerá na situação que o torna válido. Além disto, na prática, em toda etapa de seleção de eventos, eventos de prioridade maior excluem a possibilidade

de ocorrência de um evento de prioridade menor. Ou seja, é possível que um evento de prioridade *high* não seja executado quando suas condições forem satisfeitas, no caso em que um evento de prioridade *veryHigh* se tornar válido também.

3.2.3 Integração com aplicações

O Diretor de Narrativas (DN) foi projetado para ser executado em conjunto com algum aplicativo terceiro, capaz de interagir com o DN e realizar o intermédio entre um usuário e a história em si. Desta forma, o DN possui um conjunto de métodos e atributos expostos como interface para serem utilizados por tais aplicações. Os principais métodos expostos pelo DN são:

- *Init()*: realiza o procedimento fundamental que prepara o DN para conduzir a história do início.
- *Begin()*: inicia a história, aplicando o evento de abertura e qualquer outro que possivelmente é desencadeado.
- *Tick(action)*: recebendo como parâmetro uma ação, progride a história em "um passo", aplicando a ação recebida, e qualquer outro evento que o DN decida aplicar.
- *GetPlayerActions()*: informa o conjunto de ações disponíveis no momento que podem ser executadas pelo usuário.

Portanto, o procedimento básico que qualquer aplicativo integrado deve realizar é preparar o DN com o *Init*, iniciar a história com *Begin* e então iterar na narrativa, recuperando as possíveis ações no momento com o *GetPlayerActions* e enviando a escolhida pelo usuário com o *Tick*. Além destes métodos, existem outros que servem para informar o estado da narrativa, incluindo também atributos públicos. Entre eles,

- *HasEventOccurred(event)* informa se dado evento já ocorreu na história, utilizado também no algoritmo de decisão do DN,
- *isOver* informa se a história já alcançou algum evento de encerramento,
- *GetAttributes()* informa todos os atributos da história e seus valores, também podendo ser chamado para cada ator e objeto.

Cada atributo da narrativa e dos existentes são acessíveis pelo aplicativo para eventuais consultas específicas. Caso seja uma consulta geral, pode-se usar o método *GetAttributes* do elemento de interesse. Dentre esses, o único obrigatório é o atributo *isOver*, que deve ser verificado pelo aplicativo a cada iteração para encerrar a história

assim que este indicar que a narrativa acabou. E quando for o caso de reiniciar a história, deve-se chamar novamente os métodos *Init* e *Begin*.

Além destas restrições apresentadas, o aplicativo não está sujeito a qualquer outra, podendo assim ter diversos formatos de apresentação e interação com usuário – desde interface textual até ambientes virtuais tridimensionais.

3.3 Método de descrição de narrativas

Enredos de narrativas podem ter diversos formatos e sequências, especialmente as interativas no meio digital. Em relação ao ambiente proposto, existem também múltiplos métodos para sintetizar diferentes sequências narrativas, devido a complexidade inerente a linguagem e ao sistema em si. Esta seção vai apresentar algumas noções comuns no desenvolvimento de uma narrativa interativa e então exemplificar como sintetizar estes padrões na linguagem STGEN.

3.3.1 Sequência Linear

Todo o ambiente proposto e a DSL STGEN são altamente especializados para enredos mais complexos e interativos. Mas isso não quer dizer que o básico deve ficar de fora. Qualquer enredo, até os mais complexos, tem alguma sequência linear de eventos, que não permite variações. Por conta do procedimento padrão do DN – selecionar eventos aleatoriamente – já existe uma certa complexidade para criar este caso simples de linearidade. E também existe mais de uma maneira para tal.

Pode-se usar um atributo para controle de "ato" da história, incrementando-o de forma fixa conforme os eventos ocorrem. Se existir apenas um evento válido para cada ato, e cada evento incrementa o valor do ato, então certamente ocorrerá uma sequência de eventos linear. O código 3.11 contém um exemplo deste uso.

O exemplo consta uma sequência que inicia-se com a ação do jogador de subir na ponte. Esta ação, a "SubirNaPonte", altera o valor do atributo "ato". Por isto, na etapa de acontecimentos a seguir, garantidamente ocorre o evento "PonteBamba". Este também incrementa o valor do atributo de controle. Após isso, o estado da narrativa – neste caso, apenas o atributo "ato", – juntamente com os requisitos dos eventos, garante que a única ação disponível é "CorrerPelaPonte". Em seguida, novamente na etapa de acontecimentos, apenas o evento "Encerramento" está válido. O controle do estado da história através de simples atributos como neste exemplo pode ser muito útil para exercer controle fino dos eventos. Alternativamente, pode-se usar condição *OrderCondition*, que requer um outro evento em questão tenha ocorrido ou não. O código 3.12 retrata esse caso, alterando o exemplo anterior.

Código 3.11 – Exemplo em STGEN que descreve sequência linear de eventos.

```
1  Story data
2  {
3    quantity ato = 1
4  }
5
6  Plot
7  {
8    opening event Inicio {
9      description: "Era uma vez uma ponte que você tinha que atravessar..."
10   }
11
12   action SubirNaPonte {
13     type: movement
14     description: "Você começou a andar pela ponte bamba."
15     short-description: "Subir na ponte."
16     actor: Jogador
17     require: Story.ato == 1
18     change: Story.ato += 1
19   }
20
21   event PonteBamba {
22     description: "A ponte começa a balançar muito, parece que vai cair."
23     require: Story.ato == 2
24     change: Story.ato += 1
25   }
26
27   action CorrerPelaPonte {
28     type: movement
29     description: "Você correu e pulou desesperadamente."
30     short-description: "Correr pela ponte!"
31     actor: Jogador
32     require: Story.ato == 3
33     change: Story.ato += 1
34   }
35
36   ending event Encerramento {
37     description: "... e ponte foi atravessada."
38     require: Story.ato == 4
39   }
40 }
```

Neste caso não é necessário utilizar um atributo para controlar o estado numericamente. A história inicia-se com o evento "Inicio". A seguir é avaliado qual ação está disponível e apenas a "SubirNaPonte" qualifica pois ela tem a condição de ocorrer *após o evento Inicio* e *antes dela mesma*. Ambas condições são verdadeiras. Ter uma *OrderCondition* que requer que o evento ocorra antes dele mesmo é uma forma prática de garantir que este evento ocorre apenas uma vez na narrativa. Neste exemplo, todos os eventos e ações utilizam essa estratégia, descrito nas linhas 13, 19 e 28. Todos os outros eventos mantém este padrão – deve ocorrer depois do anterior e antes dele próprio – assim garantindo a linearidade.

Devido ao algoritmo padrão do DN descrito na figura 13, é quase imperativo que seja intercalado ações do jogador com acontecimentos. Porém, caso seja interessante uma sequência linear de acontecimentos, pode-se usar múltiplos desencadeadores *trigger*. O código 3.13 contém um exemplo que realiza uma sequência linear de acontecimentos

Código 3.12 – Exemplo que descreve sequência linear de eventos de forma alternativa.

```
Plot
{
  opening event Inicio {
    description: "Era uma vez uma ponte que você tinha que atravessar..."
  }

  action SubirNaPonte {
    type: movement
    description: "Você começou a andar pela ponte bamba."
    short-description: "Subir na ponte."
    actor: Jogador
    require: after Inicio
    require: before SubirNaPonte
  }

  event PonteBamba {
    description: "A ponte começa a balançar muito, parece que vai cair."
    require: after SubirNaPonte
    require: before PonteBamba
  }

  action CorrerPelaPonte {
    type: movement
    description: "Você correu e pulou desesperadamente."
    short-description: "Correr pela ponte!"
    actor: Jogador
    require: after PonteBamba
    require: before CorrerPelaPonte
  }

  ending event Encerramento {
    description: "... e ponte foi atravessada."
    require: after CorrerPelaPonte
  }
}
```

utilizando apenas o desencadeador em questão.

Neste trecho, uma vez que a ação "SubirNaPonte" é executada pelo jogador, o evento "PonteBamba" é desencadeado. Este, por sua vez, desencadeia o evento "PonteCaindo" logo em seguida. Isto resulta em um sequência imediata que ocorre juntamente com a ação inicial. Como estes eventos não podem ocorrer de outra maneira, e também não podem ser repetidos, pode-se dizer que fazem parte da ação que os desencadeia. Este método pode ser usado para segmentar eventos complexos em um sequência de eventos mais simples.

3.3.2 Bifurcação Simples

Em um narrativa interativa, o mínimo esperado é que haja ações que o jogador interfira na história e isto leve a desenvolvimentos diferentes. O caso mais simples de aplicar isto é criando uma bifurcação no enredo, aonde dependendo de algum parâmetro, o história segue um caminho ou o outro. Em STGEN, uma das formas mais intuitivas de criar um bifurcação é utilizando o desencadeador *may-trigger*. O código 3.14 contém um fragmento de descrição da história utilizando esta técnica.

Código 3.13 – Exemplo que descreve sequência linear de acontecimentos usando *triggers*.

```
action SubirNaPonte {
  type: movement
  description: "Você começou a andar pela ponte bamba."
  short-description: "Subir na ponte."
  actor: Jogador
  triggers: PonteBamba
  require: before SubirNaPonte
}

event PonteBamba {
  description: "A ponte começa a balançar muito, parece que vai cair."
  triggers: PonteCaindo
  require: before PonteBamba
}

event PonteCaindo {
    description: "A ponte começou a desmoronar"
    triggers: Ending
}
```

Código 3.14 – Código em STGEN que usa o *may-trigger* para bifurcar a história.

```
action SubirNaPonte {
  type: movement
  description: "Você começou a andar pela ponte."
  short-description: "Subir na ponte."
  actor: Jogador
  triggers: PonteBamba
  require: before SubirNaPonte
}

event PonteBamba {
  description: "A ponte começa a balançar muito..."
  may-trigger: PonteCaindo, PonteEstabilizou
  require: be-triggered
}
event PonteCaindo {
  description: "A ponte começou a desmoronar por causa do peso!"
  require: Jogador.peso >= 90
  require: be-triggered
}
event PonteEstabilizou {
  description: "A ponte aguenta o peso e se estabiliza!"
  require: Jogador.peso < 90
  require: be-triggered
}

action AtravessarPonte {
  type: movement
  description: "Foi um susto mas tudo deu certo. Você atravessou a ponte."
  short-description: "Continuar atravessando a ponte."
  actor: Jogador
  require: after PonteEstabilizou
  require: before AtravessarPonte
}
```

No exemplo, assim que o jogador realizar a ação “SubirNaPonte”, é desencadeado o evento “PonteBamba”. Este evento atua como ponto de decisão entre outros dois possíveis eventos por conter o *may-trigger* que faz referência aos eventos “PonteCaindo” e “PonteEstabilizou”. O desencadeador *may-trigger*, como apresentado na seção 3.1.1, indica que esses eventos podem disparar mas isto depende das condições dos eventos. Além disto, a verificação dos eventos a serem disparados neste momento segue a lógica apresentada na figura 14. Ou seja, é possível que nenhum seja desencadeado, ou apenas um, e qualquer outra combinação é possível.

Neste caso, como os eventos são normais e possuem condições sem interseção, apenas um deles ocorre simultaneamente e com certeza um ocorrerá. Desta forma, esta construção é efetivamente uma bifurcação simples. Existem mais formas de se garantir essa situação. Por exemplo, se fosse usada a prioridade *high* em um dos eventos decisivos, daria para simplificar as condições do outro evento, pois o de maior prioridade seria executado sozinho caso ambos fossem válidos.

A decisão entre qual dos eventos ocorre é o atributo “peso” do jogador. Não está exposto qualquer modificação neste fragmento de código. Porém este é o caso em que este atributo pode ser modificado em momentos anteriores da narrativa e este ponto faz uso de alterações em eventos passados. Por exemplo, pode ter ocorrido alguns pontos de decisão anteriores onde o jogador optou por carregar muita coisa, aumentando seu peso para mais que 90. Assim, ao tentar atravessar a ponte, seria desencadeado a situação na qual a ponte desmorona. Caso tenha optado por viajar leve, vai atravessar a ponte sem maiores problemas.

Alguns dos eventos neste exemplo possuem o *TriggerCondition* – presente nas linhas 13, 20 e 25 – para garantir que não serão selecionados pelo DN fora de contexto. Com isso, é garantido que este conjunto de eventos só pode ocorrer após e juntamente com o evento “SubirNaPonte”.

3.3.3 Modelo do Jogador

Como discutido na seção 2.1, geradores da narrativa interativas podem conter uma representação abstrata que busca mapear as escolhas do usuário que interage com a história, de forma a possibilitar uma adaptação da história ao seu perfil. Em STGEN, é possível criar um modelo de jogador ao simplesmente definir atributos que sejam utilizados para armazenar dados com esta semântica. Isto é, atributos criados podem servir tanto para controlar o estado do personagem da história quanto para mapear o próprio jogador. O código 3.15 contém um exemplo deste método.

Neste fragmento existem dois atributos que são usados para definir o tipo do jogador: “prefereCombate” e “prefereDialogo”. O DN pode, com estes dados, guiar a

Código 3.15 – Código em STGEN contendo atributos usados como modelo do jogador.

```
Space
{
  player actor Jogador {
      name: "Joe"

      quantity combatePontos = 0
      quantity dialogoPontos = 0

      fact prefereCombate = (Jogador.combatePontos > Jogador.dialogoPontos)
      fact prefereDialogo = (not Jogador.prefereCombate)
  }
}
```

história para caminhos que contenham mais combates ou caminhos com mais diálogos, dependendo dos valores dos atributos. Estes dois atributos factuais são definidos através de outros dois atributos quantitativos, que podem ser incrementados arbitrariamente ao longo da narrativa, de acordo com as ações do jogador. Por exemplo, pode-se apresentar pontos de decisão no começo da história que sirvam para aumentar o valor destes atributos de acordo, como deixar o jogador escolher entre uma espada ou um livro; ignorar possíveis interações com outros personagens pode indicar que o jogador prefere mais ação; entre outros.

3.4 STGENapp - Aplicativo Dedicado

STGENapp é o aplicativo desenvolvido respeitando as definições da seção 3.2.3, de forma a permitir testes utilizando as narrativas definidas em STGEN.

O aplicativo é baseado em clássicos jogos de *Interactive Fiction* (IF), ou ficção interativa, onde jogadores participam do mundo narrativo através de comandos limitados. o aplicativo também segue o padrão mais comum das IF onde a principal interação com o jogo é através de texto. Para os propósitos deste trabalho, o aplicativo é bastante adaptado ao DN e, por este motivo, genérico em termos estéticos e práticos. Isto é, o jogo apresenta uma interface e interação simples, com pouca temática própria, mas que se adéquam a qualquer narrativa gerada pelo STGEN. A figura 15 apresenta a estrutura visual do aplicativo.

O aplicativo é visualmente dividido nestas três seções demarcadas com os símbolos I, II, e III. A seção I é a região onde são expostas as ações disponíveis a serem realizadas em um dado momento da história. Esta contém um conjunto de ações, cada uma com um símbolo e um texto breve que indica o que será realizado ao selecionar tal ação. A cada iteração da narrativa, apenas uma ação pode ser efetuada pelo jogador. Cada ação normalmente afeta o estado da história, de forma não explícita. A forma com que cada ação afeta a história depende de como foram modeladas pelo autor. A seção II descreve em parte

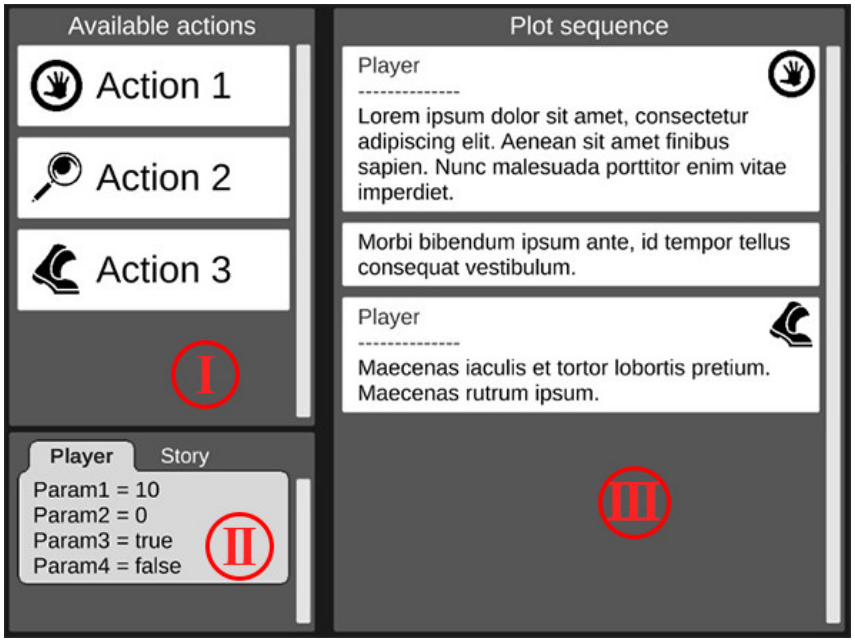

Figura 15 – Interface do aplicativo desenvolvido com suas seções destacadas.

i. Seção que permite a seleção de uma das possíveis ações em um determinado instante. ii. Região que apresenta o estado na história. iii. Sequência de eventos ocorridos da história.

o estado da história que é disponibilizado pelo DN. Neste aplicativo, optou-se por revelar apenas os dados do ator que representa o jogador e os dados pertencentes à "história em si". Isto porque estes são os elementos da história que são obrigatórios e, por conta disso, são provavelmente os mais relevantes para uma dada história qualquer desenvolvida no sistema em questão. Pode-se interagir nesta seção ao alternar entre visualizar os dados do jogador e os dados gerais da história, selecionando uma das duas abas. E a seção III apresenta a sequência de eventos (o enredo) da história. A cada iteração, o DN retorna um ou mais eventos e ações que ocorreram na história. Estes são recebidos no aplicativo e descritos nesta seção, em formato de lista ordenada e acontecimentos narrativos. Ao iniciar-se uma história, é apresentado nesta seção a ambientação inicial da narrativa, através de uma sequência de eventos iniciais.

Em relação à lógica de processos, o aplicativo possui procedimentos simples, uma vez que sua função é basicamente servir como interface entre o jogador e o DN. Sua lógica principal está descrita nos diagramas da figura 16.

O procedimento inicia-se quando o aplicativo é aberto. A primeira ação realizada é iniciar a história, que é feito através da solicitação ao DN pela abertura da narrativa. Neste momento já são retornados ao jogo uma sequência com um ou mais eventos que ocorrem no começo, e com isso, já faz-se necessário atualizar a exibição dos dados que representam o estado da história. Em seguida, e caso a história não tenha chegado a um fim, o algoritmo fica ocioso até a próxima ação do jogador. Assim que o jogador escolhe uma ação, o aplicativo a envia ao módulo DN, acarretando em mais um ou mais eventos

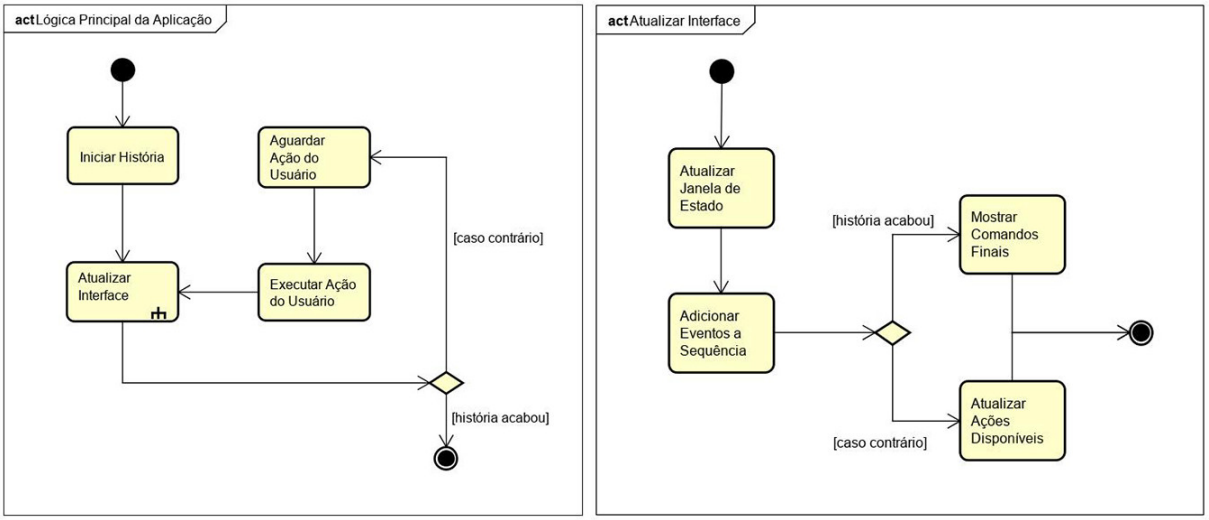

Figura 16 – Diagramas de atividade descrevendo a lógica principal do aplicativo.

que alteram o estado da história. Com isso é necessário novamente atualizar a interface do jogo, e esta sequência de procedimentos se repete até que a condição de parada seja atingida, isto é, algum evento final tenha ocorrido na história.

A atualização da interface é a alteração dos elementos visuais que contam a história e permitem a interação do usuário com a narrativa. Neste procedimento é feito, em sequência, – e de acordo com a figura 6 – a atualização da região ii, os atributos do jogador e da história; a adição dos últimos eventos ocorridos na lista da região iii; e por último atualiza-se as ações disponíveis ao jogador de acordo com o novo estado da história. Nesta última etapa, caso a história tenha alcançado algum evento final, são mostrados comandos para Sair ou Reiniciar a história no lugar de possíveis ações dentro da narrativa.

4 Implementação

Nesta seção será discorrido em detalhes o desenvolvimento do sistema apresentado, incluindo as ferramentas utilizadas.

4.1 DSL STGEN e ambiente

A linguagem específica de domínio (DSL) STGEN foi desenvolvida utilizando a *framework* Xtext[1] – criado pela *Eclipse Project* e feito para a IDE Eclipse[2] – que provê uma infraestrutura robusta para criação de linguagens de programação e DSLs – incluindo definição de gramática, regras de validação e métodos para geração de código.

Uma vez definida a linguagem, o Xtext disponibiliza ainda suporte para edição de texto personalizada a linguagem criada utilizando LSP, algum navegador ou a própria IDE Eclipse. Foi optado utilizar a própria Eclipse para como editor de texto da STGEN, por questões de praticidade e robustez, uma vez que este contém total integração com as funcionalidade do Xtext. Com isto, o ambiente de criação de narrativas é habilitado como uma consequência da definição completa da DSL STGEN em Xtext.

Desenvolver uma DSL em Xtext envolve a definição de três módulos, a saber, definição da sintaxe, validador e gerador de código. No resto dessa seção serão apresentada as definições destes módulos na *framework*.

4.1.1 Definição da Sintaxe

É notável na definição de linguagens em Xtext que não define-se apenas a sintaxe, mas também como os elementos da linguagem estão relacionados. Com este molde, a *framework* constrói um modelo semântico da sintaxe. Este modelo, também chamado de *abstract syntax tree* (AST), ou árvore de sintaxe abstrata, é utilizado nas outras etapas da construção da DSL – definição dos métodos de validação e gerador de código, – permitindo uma navegação estruturada entre elementos da linguagem.

No código 4.1 está o fragmento inicial da definição da DSL STGEN.

Neste fragmento estão os elementos superiores da linguagem, incluindo o *Story*, que pode ser chamado de raiz da AST que representa essa sintaxe. Este é simplesmente composto por três outros elementos, isto é, para descrever uma história válida em STGEN, deve-se descrever totalmente os elementos *StoryData*, *Space* e *Plot*. Estes, por sua vez,

[1] *framework* de desenvolvimento de linguagens de programação, disponível em <https://www.eclipse.org/Xtext/>.

[2] IDE popular para Java, disponível em <https://www.eclipse.org/ide/>.

Código 4.1 – Fragmento da sintaxe que define os recipientes superiores.

```
Story:
  data=StoryData
  space=Space
  plot=Plot
;

StoryData:
  {StoryData} 'Story' 'data' '{'
    attributes+=Attribute*
  '}'
;

Space:
    {Space} 'Space' '{'
        existents+=Existent*
    '}'
;

Plot:
  {Plot} 'Plot' '{'
    events+=Event*
  '}'
;
```

contém palavras literais a serem escritas – definidas entre aspas simples – e também um conjunto de atributos, existentes e eventos, respectivamente.

Pegando, por exemplo, a construção do *Space*, deve-se definir um conjunto de existentes. A definição de existentes em Xtext está contida no código 4.2.

Código 4.2 – Fragmento da sintaxe que define os tipos de existentes.

```
Existent: Actor | Object ;

Actor:
  (playable?='player')? 'actor' name=ID
  '{' 'name:' displayName=STRING attributes+=Attribute* '}'
;

Object:
  'object' name=ID
  '{' attributes+=Attribute* '}'
;
```

Como previamente definido, um existente da história pode ser um ator ou objeto. Isto é definido na DSL na primeira linha deste fragmento. Desta forma, enquanto popula-se o espaço da narrativa, pode ser intercalado atores e objetos. Objetos são mais simples, contendo a definição de um conjunto de atributos entre chaves e prefixados pela escrita do literal *object* e um identificador para este objeto. Os identificadores são usado para fazer referência entre os elementos durante a escrita da história. Um ator possui também a mesma estrutura, porém contém um nome "a mais", uma *string* que armazena um nome de exibição. Pode-se pensar neste parâmetro como um atalho para suprimir a falta de atributos do tipo *string* na linguagem. Além disso pode-se escrever o literal *player* antes

de definir o ator. Isto o marca como o ator que representa o usuário externo a história.

No apêndice B encontra-se a sintaxe completa da STGEN em Xtext. E o modelo semântico em ECore da linguagem, gerado pela definição da gramática, encontra-se no apêndice C.

4.1.2 Validador

A validação é definida utilizando Xtend[3], um dialeto de Java que estende a linguagem com funcionalidades extras, como expressões lambda, métodos de extensão e invocação polimórfica de métodos.

Para cada uma das validações descritas na seção 3.1.2, existe um método de verificação em Xtend. Por exemplo, para garantir que todos os existentes da história possuem nomes únicos, há o método descrito no código 4.3.

Código 4.3 – Verificador que garante que os nomes dos existentes são únicos.

```
1  @Check
2  def void checkExistentNameIsUnique(Existent existent) {
3    var space = existent.eContainer as Space;
4
5    for (other : space.existents) {
6      if (other != existent && existent.name.equals(other.name)) {
7        error(
8          "Existent names have to be unique.",
9          StoryGenPackage.Literals.EXISTENT__NAME
10       );
11       return;
12     }
13   }
14 }
```

Este tipo de validação é estática e feita em relação ao tipo explicitado na declaração do método. Neste exemplo, na linha 2, é declarado que o alvo do validador é o tipo *Existent*. Desta forma, este método de validação é aplicado a todo elemento da história do tipo *Existent*. Isto é, todo ator e objeto declarado será analisado de acordo com este procedimento.

Em outro caso de validação, o código 4.4 contém o método que garante a existência de um e apenas um evento de abertura na história.

Neste exemplo a validação é feita sobre o objeto *Plot*, representante do enredo, contendo todos os eventos definidos pelo autor. Nas linhas 3–5, realiza-se seleção dos eventos que são *HappeningEvent* – uma vez que apenas eventos deste tipo podem ser de abertura – e que estão marcados como *opening* e conta-se a quantidade destes. Este valor resultante deve ser 1. Se for maior, então existe mais de um evento de abertura, e se for 0, não existe um evento de abertura. Nestes dois casos inválidos, são disparados erros

[3] Dialeto que estende as funcionalidade de java, disponível em <https://www.eclipse.org/xtend/>.

Código 4.4 – Verificador que garante a existência de um evento de abertura.

```
1  @Check
2  def void checkPlotOpening(Plot plot) {
3    val openingsCount = plot.events
4      .filter(HappeningEvent)
5      .filter[e | e.isOpening].size;
6
7    if(openingsCount <= 0) {
8      error(
9        "An opening event is required.",
10       StoryGenPackage.Literals.PLOT__EVENTS
11     );
12   }
13   else if(openingsCount > 1) {
14     error(
15       "Only ONE opening event is supported.",
16       StoryGenPackage.Literals.PLOT__EVENTS
17     );
18   }
19 }
```

condizentes, definidos nas linhas 8–10 e 14–17.

4.1.3 Gerador de Código

O gerador de código, também na linguagem Xtend, é definido através de uma série de métodos com *template code*, uma forma expressiva para definição de modelos de código, intercalando texto a ser tratado como literal e códigos de forma prática. Por exemplo, o código 4.5 é o responsável por converter um ator em STGEN para seu correspondente em C#. Este gera, por exemplo, o código 3.5.

Entre os caracteres ’’’ das linhas 4 e 28, está o bloco de código no qual tudo é considerado literal, exceto o que estiver entre os chevrons « e », sendo considerados como código Xtend novamente. O método é composto pela definição da classe que será correspondente ao ator definido, cujo nome será o nome do ator, e que herda da classe *StoryActor*; um método construtor; um método que recupera os atributos do ator em um dicionário; e por fim a sequência de seus atributos. Neste último é chamado outro método de conversão para cada atributo do ator. Este método está descrito no código 4.6.

Neste caso é usado a invocação polimórfica de métodos, aonde para qualquer *Attribute* é invocado o método *convertAttribute*, e o método correspondente ao tipo especializado do atributo – seja *QuantityAttribute* ou *FactAttribute* – será executado.

O resto dos métodos de geração de código seguem estrutura similar aos dos apresentados até aqui, incluindo bastante código estático que compõe o DN.

Código 4.5 – Método que converte um ator em STGEN para classe em C#.

```
def dispatch convertExistent(Actor actor) {
  val actions = plot.events.filter(ActionEvent).filter(a | a.actor.equals(actor))

'''
public class «actor.name.toFirstUpper» : StoryActor
{
  public «actor.name.toFirstUpper»()
  {
    displayName = "«actor.displayName»";
    actions = new List<Plot.ActionEvent>() { «FOR a : actions»Story.plot.«a.actor
        .name.toFirstLower»«a.name.toFirstUpper», «ENDFOR» };
  }

  public override IDictionary<string, string> GetAttributes()
  {
    var attrs = new Dictionary<string, string>();

    «FOR attr : actor.attributes»
    attrs.Add("«attr.name.toFirstUpper»", «attr.name.toFirstUpper» + "");
    «ENDFOR»

    return attrs;
  }

  «FOR attr : actor.attributes»
  «attr.convertAttribute(false)»
  «ENDFOR»
}
'''
}
```

Código 4.6 – Métodos que geram os atributos dos existentes.

```
def dispatch convertAttribute(FactAttribute fa, boolean isStatic)
'''
«IF fa.initValue !== null»private «IF isStatic»static«ENDIF» bool «fa.name.
    toFirstLower» = «fa.initValue.value»;«ENDIF»
public «IF isStatic»static«ENDIF» bool «fa.name.toFirstUpper»
{
  «IF fa.macroExp !== null»
  get { return «fa.macroExp.convertExpression»; }
  «ELSE»
  get { return «fa.name.toFirstLower»; }
  set { «fa.name.toFirstLower» = value; }
  «ENDIF»
}
'''

def dispatch convertAttribute(QuantityAttribute qa, boolean isStatic)
'''
private «IF isStatic»static«ENDIF» int «qa.name.toFirstLower» = «qa.initValue.
    value»;
public «IF isStatic»static«ENDIF» int «qa.name.toFirstUpper»
{
    get { return «qa.name.toFirstLower»; }
    set { «qa.name.toFirstLower» = value; }
}
'''
```

5 Aplicação

Nesta seção serão apresentados dois exemplos de construção de narrativas interativas em STGEN no ambiente de criação de narrativas. A primeira é uma recriação da história de um episódio de série interativa, a nova modalidade de séries televisivas que a Netflix[1] está criando (seção 5.1). A segunda é uma história de autoria própria, criada com o objetivo de explorar o potencial de interatividade do ambiente desenvolvido (seção 5.2).

5.1 Episódio de série interativa

Este exemplo é baseado inteiramente no seriado interativo *You vs Wild* da Netflix. Foi feito o estudo do conteúdo da história do primeiro episódio e então recriado em STGEN para avaliação da complexidade e conformidade com a qualidade de expressão da narrativa original.

O episódio trata de uma aventura de resgate que Bear Grylls (BG), aventureiro e sobrevivente profissional, deve encontrar a Dra. Ramos, desaparecida enquanto em uma missão para entregar vacinas de malária a crianças de uma vila remota. O episódio se passa em uma selvagem mata da américa central, onde o espectador – no caso um "conselheiro" – deve fazer escolhas para ajudar o BG a resgatar a doutora.

A interatividade na série é simples. Em alguns momentos, o espectador pode escolher uma dentre duas opções. E a escolha pode ter consequência imediata, tardia, ou até nenhum efeito diferente da outra opção. A figura 17 contém o gráfico de enredo construído a partir da análise do episódio.

As decisões do espectador começam tão cedo quanto a preparação antes do começo do resgate, onde pode-se escolher entre um gancho de escalada ou um estilingue como último item que o BG levará. Este é o ponto de decisão D1, logo após o cenário C1. Esta decisão afeta a história de forma tardia, em cenários mais avançados. O gancho tem apenas um uso, que é impedindo o fracasso da missão no cenário C11, no qual é utilizado para sair de um precipício. O estilingue facilita o encontro com um crocodilo no cenário C15, evitando um ponto de decisão, e evita o fracasso no cenário C4 ao impedir o desencadeamento da situação C5, onde BG persegue um macaco e acaba ferido e incapacitado de continuar a aventura.

O D1 é o único ponto de decisão que afeta a narrativa de forma não imediata, e o D7 é o único cujas opções resultam no mesmo caminho – onde o espectador escolhe entre comer cupins ou uma grande larva. O caminho mais rápido – o que requer menos decisões

[1] Netflix é uma plataforma de *streaming* de filmes e séries. <https://www.netflix.com/>

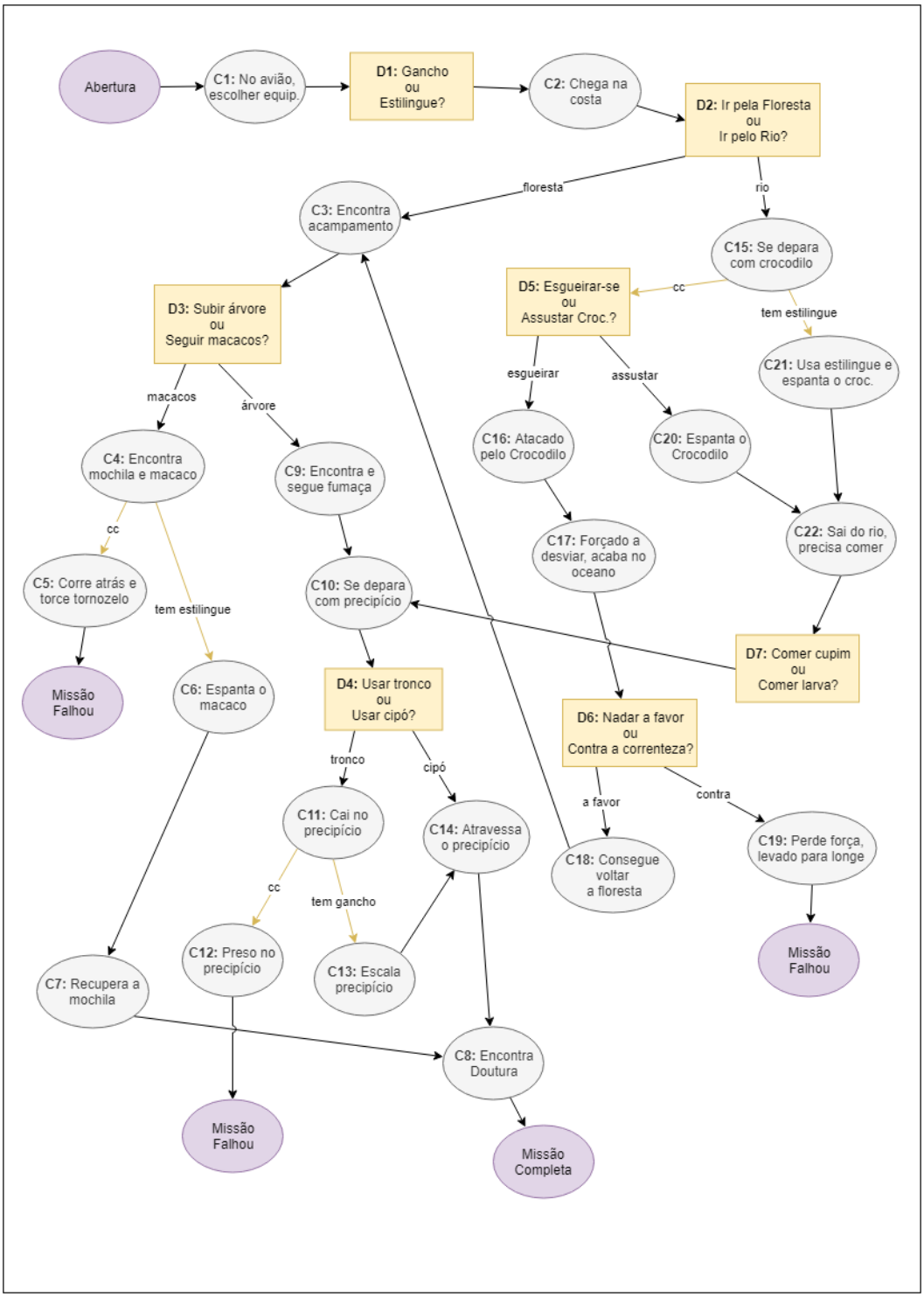

Figura 17 – Gráfico do enredo do episódio 1 de *You vs Wild*, da Netflix.

Os nós elipsais, marcados de C1 até C22 representam os cenários da narrativa; os nós retangulares, notados com D1 até D7, são os pontos de decisão onde o espectador conselheiro escolhe uma das duas possíveis opções; e os demais nós elipsais destacados representam o início da história e os possíveis finais.

– é o C1, D1 (estilingue), C2, D2 (floresta), C3, D3 (seguir macacos), C4, C6, C7 e C8, passando por apenas três pontos de decisão. E o mais longo é o C1, D1 (gancho), C2, D2 (rio), C15, D5 (esgueirar-se), C16, C17, D6 (nadar a favor), C18, C3, D3 (subir árvore), C9, C10, D4 (atravessar no tronco), C11, C13, C14 e C8, passando por 6 pontos de decisão.

Esta narrativa interativa, recriada em STGEN, possui pouca necessidade de atributos, uma vez que a maioria dos cenários – mapeados como eventos no contexto da STGEN – influenciam os próximos simplesmente por sua ocorrência no enredo. Desta forma, foi possível criar a narrativa sem uso de atributos em *Story data* e existe apenas um ator que representa o encarregado do resgate. Este contém apenas dois atributos, para representar se foi escolhido o gancho ou o estilingue. Poderia ser apenas um atributo factual, porém criar um pra cada torna mais legível ao longo da descrição dos eventos. O código 5.1 contém a descrição do espaço da narrativa.

Código 5.1 – Descrição do espaço completo do episódio mapeado para STGEN.

```
1 Space
2 {
3   player actor BearGrylls {
4     name: "Bear Grylls"
5     fact possuiGancho = false
6     fact possuiEstilingue = false
7   }
8 }
```

De acordo com o episódio, estes dois atributos só podem ser alterados no primeiro ponto de decisão, que ocorre logo no início da narrativa. O código 5.2 contém o início da narrativa e este ponto de decisão.

O evento de abertura “Abertura” apresenta apenas um texto e então desencadeia o próximo evento, “DP_GanchoOuEstilingue”, o primeiro ponto de decisão. Todos os pontos de decisão estão prefixados com “DP”. Neles, o BG se dirige ao conselheiro, informando as opções, contidas no campo *description* da ação. Com esta ação desencadeada, as outras duas, das linhas 13 e 24, têm todas as condições satisfeitas e então estarão disponíveis ao jogador. As condições das linhas 20, 21, 31 e 32 existem para evitar que estas ações do jogador fiquem disponíveis em cenários futuros, garantindo que elas só ocorrem uma vez pois só devem “ocorrer antes dela mesma e do seu par”. Todos os pontos de decisão seguem este padrão, uma ação do BG para informar as opções, e duas outras ações que ficam válidas após a primeira, e que são realizadas pelo jogador.

As alterações de atributo nestas duas ações são levadas em considerações em possíveis eventos seguintes no enredo. Por exemplo, o cenário onde o Bear Grylls cai no precipício é está recriado em STGEN no código 5.3.

Na linha 1 está o anúncio do ponto de decisão do precipício, onde é descrito que podemos escolher tentar atravessar usando um tronco estendido através da queda ou

Código 5.2 – Início da narrativa do episódio em STGEN e o primeiro ponto de decisão.

```
1 opening event Abertura {
2   description: "A missão é resgatar a doutora que está perdida nesta floresta
        amazônica."
3   triggers: DP_GanchoOuEstiligue
4 }
5
6 action DP_GanchoOuEstiligue {
7   type: communication
8   description: "Tenho espaço para mais um item. O gancho pode me ajudar caso eu
        fique preso em algum lugar, e o estilingue é ótimo para lidar com animais
        indesejados. O que acha? Devo levar o gancho ou o estilingue?"
9   short-description: ""
10  actor: BearGrylls
11  require: be-triggered
12 }
13 action EscolherGancho {
14  type: communication
15  description: "Ok, vou levar o gancho."
16  short-description: "Gancho."
17  actor: BearGrylls
18  triggers: DesceuDoAviao
19  require: after DP_GanchoOuEstiligue
20  require: before EscolherGancho
21  require: before EscolherEstilingue
22  change: BearGrylls.possuiGancho = true
23 }
24 action EscolherEstilingue {
25  type: communication
26  description: "Ok, vou levar o estilingue."
27  short-description: "Estilingue."
28  actor: BearGrylls
29  triggers: DesceuDoAviao
30  require: after DP_GanchoOuEstiligue
31  require: before EscolherGancho
32  require: before EscolherEstilingue
33  change: BearGrylls.possuiEstilingue = true
34 }
```

escolher usar um cipó "estilo Tarzan". Escolhendo o tronco – ação da linha 3, – sempre segue a queda no precipício, com a quebra da ponte natural. O evento "CairNoPrecipicio" representa esse cenário, e nele há um *may-trigger* que realiza a decisão automática para saber se será possível continuar na aventura ou não. Neste caso, os dois eventos a possivelmente serem desencadeados têm condições disjuntas. Escalar o precipício requer que o atributo "possuiGancho" seja verdadeiro, e o "TentarEscalarPrecipicio", que representa a falha em escalar, requer que o mesmo atributo seja falso. Desta forma, certamente um dos dois ocorre. Caso não tenha sido escolhido o gancho, nesta situação a aventura é encerrada, e o evento de encerramento "PresoNoPrecipicio" é desencadeado, finalizando a história.

Existem quatro finais alternativos nesta história, sendo apenas um deles o final "bom". Os outros três são situações onde a missão falhou, onde BG não consegue chegar a doutora e é forçado a desistir. Os finais, em STGEN, são eventos como dispostos no código 5.4.

Cada um destes é um evento marcado com *ending* e tem a condição *be-triggered*. Para cada um destes, existe um outro evento, representando o cenário condizente, que

Código 5.3 – Eventos que recriam o cenário do precipício do episódio.

```
action DP_TroncoOuCipo {/* ... */}

action EscolherTronco {
  type: communication
  description: "Você acha melhor eu tentar o tronco? Não dá pra saber se vai
      aguentar, e só vou descobrir tentando. Vou me segurar a esse cipó aqui por
      segurança."
  short-description: "Passar pelo tronco."
  actor: BearGrylls
  triggers: CairNoPrecipicio
  require: after DP_TroncoOuCipo
  require: before EscolherTronco
  require: before EscolherCipo
}
action EscolherCipo {/* ... */}

event CairNoPrecipicio {
  description: "Bear Grylss tenta atravessar pelo tronco mas ele cede. O cipó
      salva Bear Grylls de uma queda livre, mas ainda cai no fundo do precipício."
  may-trigger: EscalarPrecipicio, TentarEscalarPrecipicio
  require: be-triggered
}

action EscalarPrecipicio {
  type: interaction
  description: "Ainda bem que você escolheu o gancho de escalada. Com ele consigo
      escalar o precipício."
  short-description: ""
  actor: BearGrylls
  triggers: ProximoADoutora
  require: BearGrylls.possuiGancho
  require: be-triggered
}
action TentarEscalarPrecipicio {
  type: idle
  description: "As paredes são muito escorregadias. Sem equipamento apropriado, n
      ão vou conseguir sair daqui..."
  short-description: ""
  actor: BearGrylls
  triggers: PresoNoPrecipicio
  require: not BearGrylls.possuiGancho
  require: be-triggered
}
```

desencadeia o encerramento. Desta forma é garantido que o encerramento só ocorre na situação correta, sem possibilidade dele ser escolhido pelo DN em algum momento incoerente.

5.2 História interativa sobre naufrágio

Este exemplo é uma história sobre uma pessoa que sofre um naufrágio e milagrosamente sobrevive, se encontrando em uma ilha deserta. Nesta história, o jogador interage decidindo as ações que o personagem sobrevivente realiza na ilha. O conjunto de ações é limitado em diversas situações que o personagem se encontra. Se assemelhando a um jogo de sobrevivência em ambientes selvagens, o jogador deve manter seu persona vivo, coletando alimento, se mantendo aquecido a noite, e no final deve encontrar uma forma de

Código 5.4 – Eventos que representam os finais alternativos do episódio.

```
ending event PresoNoPrecipicio {
  description: "Preso neste precipício e sem ter como subir, a missão falhou."
  require: be-triggered
}

ending event IncapacitadoDeAndar {
  description: "Com o tornozelo ferido desta forma, a missão falhou."
  require: be-triggered
}

ending event IlhadoAposCorrenteza {
  description: "Ilhado e muito longe do objetivo, a missão falhou."
  require: be-triggered
}

ending event EncontrouDoutura {
  description: "A doutora foi resgatada, missão cumprida!"
  require: be-triggered
}
```

ser resgatado da ilha remota.

A história é dividida em dois "atos" – o primeiro ato é a introdução da narrativa, com sequência simples e pouca opção de ações, onde o jogador é apresentado ao mundo da história; e o segundo ato é semelhante a uma rotina de sobrevivência, com o objetivo de manter-se vivo e construir um grande sinalizador para conseguir um resgate, que permite sequências variadas e complexas de ações.

Esta história contém menos desenvolvimento que o exemplo anterior, sendo mais focada na interatividade. Existem múltiplas ações disponíveis quase em todos os momentos da narrativa, levando a alterações de estado diferentes. O ato 2, no qual o jogador passa o maior tempo da narrativa, é uma rotina do dia-a-dia do jogador na ilha, contendo uma mecânica de dia e noite. Durante o dia pode-se realizar várias ações, enquanto a noite é perigosa ficar vagando, então o jogador deve-se limitar a dormir em seu acampamento. Só existe um final positivo, no qual o jogador consegue ser resgatado por um barco, após criar um grande sinal de fogo. Este é o objetivo na história, juntar uma grande quantidade de madeira para construir uma grande fogueira.

Em STGEN, essa história possui um estado mais complexo, onde muitos atributos são usados para decidir o que pode ocorrer. O código 5.5 apresenta os dados da história e seu espaço.

O *Story data* contém atributos para estabelecer a passagem de tempo e mudança de dia. O atributo *clock* aumenta em um a cada ação do jogador, e quando chega em valores específicos, eventos são disparados de acordo, mudando de dia para entardecer e de entardecer para noite. Esses três horários são definidos com os atributos *isNightTime*, *isGettingDark* e *isDayTime*. Cada um torna-se verdadeiro exclusivamente dos outros dois, e isso altera os possíveis eventos e ações durante a narrativa. O atributo *act* é usado para

Código 5.5 – Fragmento da história de naufrágio contendo todos os atributos.

```
Story data
{
  quantity clock = 0 [increments by 1 every player action]
  quantity act = 1

  fact isNightTime = false
  fact isGettingDark = false
  fact isDayTime = true
}

Space
{
  player actor Joe {
    name: "Player"
    quantity food = 0
    quantity wood = 0
    quantity health = 5
    quantity hunger = 10 [decrements by 1 every player action]

    fact hasStarvationRisk = false
    fact isNearCamp = true
  }

  object Campfire {
    fact isLit = false
  }

  object Pyre {
    quantity buildStage = 0
    fact isCompleted = (Pyre.buildStage >= 5)
    fact isLit = false
  }
}
```

separar os eventos dos dois atos previamente mencionados. Quando vale 1, apenas eventos do ato 1 vão ser válidos, e quando valer 2, apenas eventos do ato 2 serão válidos.

Há três existentes nesta história. Um sendo o próprio jogador, chamado de "Joe" em STGEN; outro é um objeto que representa uma fogueira que tem estado simples – acesa ou apagada; e o "Pyre", que representa a "grande fogueira" que o jogador pode contruir. Ela contém o atributo *buildStage* que conta o quanto o jogador já investiu de tempo e madeira nela; o *isCompleted* é um macro que indica de forma explícita quando a pira está completa; e *isLit* indica se ela está em chamas ou não. Assim, o objetivo do jogador é construir a pira 5 vezes e então acendê-la. Dessa forma é possível ser resgatado.

O ator que representa o jogador contém vários atributos que são usados para criar uma mecânica de fome, devendo comer para não passar fome. A saber, *food* representa a quantia discretizada de comida que o jogador possui; *hunger* conta o quanto de fome o jogador está sentindo, decrescendo a cada ação, sendo 0 a pior fome possível; e *hasStarvationRisk* indica se o jogador está prestes a sucumbir de fome. Este último se torna verdadeiro apenas quando o *hunger* é zero e é disparado um evento pelo DN que avisa o jogador de sua situação perigosa. E quando este é verdadeiro, pode ocorrer o evento que representa morrer de fome, encerrando a história. Além destes, existem os atributos *wood*,

contando a quantidade de madeira acumulada; *health* que indica a saúde de jogador; e *isNearCamp*, indicando a localização do jogador, podendo estar no acampamento ou mais distante.

No ato 1 da história, o jogador é limitado a realizar duas ações "de exploração" em sequência, que servem para introduzir o cenário e mais conteúdo de forma interativa. Uma vez realizadas estas ações, o jogador encontra um acampamento abandonado. Nesta situação, pode-se realizar três ações de investigação em qualquer ordem. Estas estão definidas no código 5.6.

Código 5.6 – Ações de investigação do acampamento abandonado na história.

```
action CheckFirepit {
  type: visual
  description: "The ashes are completely cold. Likely this pit wasn't lit in a
      very long time."
  short-description: "Check the fire pit."
  actor: Joe
  require: after ExploreFollowUp
  require: before CheckFirepit
  require: Story.act == 1
}

action SearchOutsideCamp {
  type: visual
  description: "There isn't much around, except for an axe stuck into a log. This
       will be useful."
  short-description: "Check around the camp."
  actor: Joe
  triggers: AxeCollect
  require: after ExploreFollowUp
  require: before SearchOutsideCamp
  require: Story.act == 1
}

action SearchInsideTent {
  type: visual
  description: "Inside is pretty much empty, but there is some jerky inside a
      small box. Why would anyone leave this behind?"
  short-description: "Check inside the tent."
  actor: Joe
  triggers: FoodCollect
  require: after ExploreFollowUp
  require: before SearchInsideTent
  require: Story.act == 1
  change: Joe.food += 1
}
```

Todas as três só podem ocorrer uma vez cada e só são válidas após o evento "ExploreFollowUp" que é o último dos eventos de exploração inicial previamente mencionados. Uma vez explorado o suficiente, o jogador poderá sair do acampamento e isso inicia o ato 2. O código 5.7 contém a ação de sair do acampamento e o fechamento do ato 1.

A ação "LeaveCamp" só torna-se válida após ocorridos os eventos "FoodCollect" e "AxeCollect". Estes são eventos desencadeados por duas das ações de investigação do exemplo anterior. Ou seja, só duas das três ações são obrigatórias para liberar a ação que encerra o ato 1.

Código 5.7 – Ações para sair do acampamento e transitar do ato 1 para o ato 2.

```
1  action LeaveCamp {
2    type: movement
3    description: "The camp is behind you. You need to find a way to survive and
         leave this island..."
4    short-description: "Leave the camp."
5    actor: Joe
6    triggers: Act1Closure
7    require: after FoodCollect
8    require: after AxeCollect
9    require: Story.act == 1
10 }
11
12 event Act1Closure {
13   description: "You can only think of making a big fire and hope it draws anyones
         attention."
14   require: be-triggered
15   change: Story.act = 2
16   change: Story.clock = 0
17   change: Joe.hunger = 3
18 }
```

Quando o ato 2 começa, o jogador pode realizar um conjunto de ações diferentes. Dentre elas, procurar comida, pegar madeira e comer. Estas são as ações básicas e recorrentes que o jogador vai realizar durante a história. O código 5.8 contém estas ações e suas condições e modificações.

Coletar madeira é simples e, desde que não seja noite, o jogador pode realizá-la e coletar 5 unidades de madeira. Para comer, é necessário existir ao menos uma unidade de comida com o jogador, e gastando uma unidade, a fome é recuperada totalmente. Coletar comida possui a mesma condição que coletar madeira, porém ela possui três possíveis desdobramentos: encontrar frutinhas, encontrar cenouras ou encontrar nada. Cada um aumenta a quantidade de alimento do jogador de forma diferentes, sendo que o último caso aumenta nada.

Caso o jogador demore muito a comer, os eventos do código 5.9 podem disparar, avisando o jogador de sua situação.

Estes eventos possuem prioridade *high* para garantir que vão ser ocorrer assim que suas condições sejam satisfeitas. E assim deve ser para manter a sensação de obrigatoriedade de se alimentar. Se continuar sem se alimentar, o evento "StarvationWarning" ocorre e a partir daí a história pode ser encerrada.

Afinal, o jogador deve construir a grande fogueira através de ações repetidas de coleta de madeira e ações para aprimorar a fogueira de sinalização, intercaladas com outras ações para manter-se vivo. Construir o sinalizador e aprimorá-lo é feito através das ações definidas no código 5.10.

Cada ação de construção custa 5 unidades de madeira, e após 5 vezes, o sinalizador estará completo. Assim validando a ação "LightThePyre" que completa a condição para

Código 5.8 – Ações básicas de sobrevivência na história de naufrágio.

```
action GatherWood {
  type: interaction
  description: "You chopped down a tree and got some logs."
  short-description: "Gather wood."
  actor: Joe
  require: not Story.isNightTime
  require: Story.act == 2
  change: Joe.wood += 5
  change: Joe.isNearCamp = false
}

action GatherFood {
  type: visual
  description: "You went looking for food."
  short-description: "Search for food."
  actor: Joe
  may-trigger: FoundBerries, FoundCarrots, FoundNothing
  require: not Story.isNightTime
  require: Story.act == 2
  change: Joe.isNearCamp = false
}

action EatFood {
  type: interaction
  description: "You ate and now feel satisfied."
  short-description: "Eat something."
  actor: Joe
  require: Joe.food >= 1
  require: Story.act == 2
  change: Joe.food -= 1
  change: Joe.hunger = 10
  change: Joe.hasStarvationRisk = false
}
```

Código 5.9 – Eventos que avisam o jogador do perigo da fome.

```
event HungerWarning {
    priority: high
    description: "You are hungry."
    require: (Joe.hunger <= 3) and (Joe.hunger > 1)
    require: Story.isDayTime
    require: Story.act == 2
}

event StarvationWarning {
    priority: high
    description: "You are starving!"
    require: Joe.hunger <= 1
    require: Story.isDayTime
    require: Story.act == 2
    change: Joe.hasStarvationRisk = true
}
```

Código 5.10 – Ações de construção do sinalizador de fogo.

```
action StartBuildingPyre {
  type: interaction
  description: "Near the beach, you lay down logs and start tying them together
      to construct a pyre."
  short-description: "Start building a pyre."
  actor: Joe
  triggers: PyreStartFollowUp
  require: (Joe.wood >= 5) and Story.isDayTime
  require: before StartBuildingPyre
  require: Story.act == 2
  change: Joe.wood -= 5
  change: Pyre.buildStage += 1
  change: Joe.isNearCamp = false
}

action UpgradePyre {
  type: interaction
  description: "You spend an hour working on the pyre."
  short-description: "Keep working on the pyre."
  actor: Joe
  may-trigger: PyreFinished, PyreNeedsMoreUpgrades
  require: (Joe.wood >= 5) and Story.isDayTime
  require: after StartBuildingPyre
  require: before PyreFinished
  require: Story.act == 2
  change: Joe.wood -= 5
  change: Pyre.buildStage += 1
  change: Joe.isNearCamp = false
}

action LightThePyre {
  type: interaction
  description: "The fire is huge. Maybe someone will see it."
  short-description: "Light up the pyre."
  actor: Joe
  require: Pyre.isCompleted and not Pyre.isLit
  require: Story.act == 2
  change: Pyre.isLit = true
  change: Joe.isNearCamp = false
}
```

que um barco chegue para resgatar o jogador. O código 5.11 contém o final da narrativa quando o navio chega na ilha.

Código 5.11 – Eventos finais quando o navio chega para o resgate.

```
1 event ShipArrives {
2   description: "A ship is arriving! They must have seen the smoke."
3   require: Pyre.isLit and Story.isDayTime
4   require: before ShipArrives
5   require: Story.act == 2
6 }
7 action GoToShip {
8   type: movement
9   description: "You run to the ship's location."
10  short-description: "Go to the ship."
11  actor: Joe
12  triggers: RescuedByShip
13  require: after ShipArrives
14  require: Story.act == 2
15 }
16
17 ending event RescuedByShip {
18  description: "You board the ship. You are saved!"
19  require: be-triggered
20 }
```

6 Conclusão

Neste trabalho foi elaborada uma abordagem para geração de narrativas interativas utilizando linguagem específica de domínio, incluindo uma proposta de um sistema capaz de guiar estas histórias interativas e também um aplicativo dedicado a este sistema.

A linguagem específica de domínio (DSL) STGEN foi desenvolvida de forma a priorizar a interatividade e histórias ramificadas. Apesar de limitações, mostrou um poder de expressão razoável para narrativas interativas. A DSL apresentada possibilita a criação de enredos com desenvolvimentos complexos, incluindo bifurcações e laços narrativos, porém também é bastante complexa na sua escrita.

A DSL criada inclui a geração de código que produz um sistema feito para guiar a história descrita, chamado Diretor de Narrativa (DN). O DN contém todos os dados da história, incluindo eventos e os atores da história. A lógica de execução do DN é fixa, descrita por um algoritmo regular de seleção de eventos. Com isto, o DN é capaz de guiar de forma estável qualquer narrativa produzida. Porém a falta de inteligência no algoritmo implica que o sistema como um todo, incluindo a linguagem, não é capaz de produzir novos conteúdos narrativos.

Com o aplicativo dedicado desenvolvido em C# e Unity, foi possível testar e verificar a validade da proposta. A estratégia é interessante por permitir uma escrita especializada em narrativa interativas, que no final produz um artefato especializado para a condução desse tipo de narrativa.

A seguir, na seção 6.1, estão destacados os trabalhos relacionados encontrados na literatura que compartilham noções em comum. Depois, na seção 6.2, é feita uma análise crítica do resultado deste trabalho, avaliando a abordagem apresentada como um todo e a viabilidade do sistema. Por fim, na seção 6.3, estão delineadas propostas e ideias de trabalhos futuros que podem ser conduzidas a partir deste projeto.

6.1 Trabalhos Relacionados

A área de narrativa computacional está ativa na comunidade por aproximadamente quarenta anos com um grande número de pesquisas e propostas sendo divulgadas no meio científico (KYBARTAS; BIDARRA, 2017). Entre estas, vários sistemas foram publicados que tentam alcançar formas mais automatizadas de produção de conteúdo narrativo, atacando o problema de geração de narrativa em vários aspectos distintos. A seguir serão mencionados alguns trabalhos publicados nesta área, apresentando-os e relacionando-os com o presente trabalho.

Façade (MATEAS; STERN, 2002) é um trabalho renomado neste âmbito, considerado como um dos mais bem sucedidos jogos de narrativa interativa que surgiram no meio acadêmico. Em Façade, o jogador atua como um amigo de um casal que está passando por problemas sérios no relacionamento, podendo interagir com eles no mundo 3D virtual através de interações físicas e diálogo por meio de processamento de linguagem natural. É considerado como um *Drama manager*, pois a narrativa gira em torno de relacionamentos e emoções de personagens fictícios. A pesquisa no desenvolvimento de Façade resultou em um *framework* que permite o autor a organizar a história em fragmentos narrativos chamados *beats* – que são criados e anotados pelo autor da história com pré-condições, efeitos que alteram o estado global da história, prioridades do *beat*, entre outros parâmetros. Está incluso nesta arquitetura a linguagem chamada *Beat Sequencing Language*, que é usada pelo autor para definição dos *beats*, compondo-os com o conhecimento para decisão, ações a serem performadas e parâmetros chamados de *beat variables*.

A arquitetura Façade realiza, através do *Beat sequencer*, a seleção e ordenação dos próximos *beats* a ocorrerem na narrativa, com base nos que já ocorreram, e em seus parâmetros. Verifica-se primeiro quais são válidos e então pesa-se suas prioridades e outros parâmetros e no final é selecionado um aleatoriamente, dos que possuírem as maiores pontuações. Fazendo uma ponte, *beat* está para os *eventos* deste trabalho. Ambos representam momentos da narrativa, possuindo pré-condições e alterações no estado da narrativa. O algoritmo de seleção também possui a mesma lógica, porém o de Façade leva em consideração um crescimento e queda de tensão, seguindo o padrão *Aristotelian* de arco narrativo (MATEAS; STERN, 2002).

Como Façade, existem vários outros sistemas com foco em interatividade, como IDTension, MARLINSPIKE, Prom Week, IDA, VERSU e PASSAGE (KYBARTAS; BIDARRA, 2017). Por exemplo, VERSU (EVANS; SHORT, 2014) enfatiza a simulação social, construindo personagens complexos, e realiza uma interface de interação em hipertexto, permitindo o jogador escolher qualquer personagem e suas ações na história. Enquanto em Façade os personagens servem ao *Drama Manager*, o gerente da narrativa, como se fossem marionetes seguindo os passos exigidos para que a história progrida, em VERSU o foco em simulação faz com que a história seja construída a partir das interações sociais. A arquitetura utilizada neste caso envolve um conjunto prescrito de características que mapeiam os complexos personagens na história, detalhando suas informações, relacionamento, emoções e objetivos sociais. Em um comparativo, o sistema apresentado neste trabalho não possui esta abrangência social, nem tenta mapear a complexidade do que um personagem pode representar, focando totalmente na noção da história em si. Os personagens servem a história como em Façade, realizando suas ações quando o diretor da narrativa decidir que devem.

IDA (*Interactive Drama Architecture*), utilizado na criação de Haunt II (MA-

GERKO, 2005), é uma outra abordagem que inclui uma linguagem de domínio própria, incluindo no sistema representações para o jogador, outros atores e um mundo virtual onde a história ocorre. Os atores neste arquitetura são semi-autônomos, possuindo objetivos próprios, mas em última análise realizam o que o diretor da história requer. Similar a este trabalho, na linguagem em IDA o autor define *plot points* em um plano parcialmente ordenado de eventos, porém sem causalidade explícita dos mesmos. Isto é, provê-se formas de manter a ordenação temporal dos eventos, mas não há relação coesa entre os conteúdos de cada evento.

Em outras abordagens, Riedl et al. (2008) desenvolveram o Automated Story Director, um sistema chamado de "Experience Manager", um avanço a partir do *Drama Manager*, que realiza estratégias de adaptação da narrativa com base em uma narrativa "ideal". O sistema, contendo um exemplar de narrativa ideal, disponibiliza ações do jogador esperadas e ações divergentes. Caso o jogador saia do caminho pretendido, o sistema refaz os planos, buscando sequências novas baseadas na narrativa ideal. A arquitetura inclui agentes semi-autônomos em um mundo virtual e um modelo do jogador, utilizado para controlar o estado da história. Gervás et al. (2005) definem um método de geração de narrativa baseado em casos e em um ontologia de domínio, utilizada para mapear narrativas. Neste caso, novas narrativas são criadas a partir de outras existentes do domínio e, apesar de não incluir interatividade, geram-se novas narrativas que respeitam as estruturas narrativas em questão.

6.2 Discussão

Criar narrativas interativas pode ser uma tarefa complexa e demorada. Considerar todas as opções, caminhos e consequências demanda um cuidado muito grande para manter a coerência e qualidade da história. Sob este aspecto, a linguagem STGEN auxilia um autor ao estabelecer um padrão especializado na interatividade. A STGEN possui uma abordagem de descrição de enredo baseado em eventos e ações que, a princípio, podem ocorrer a qualquer momento, e então são impostas condições que limitam suas ocorrências. Isto é, ao invés de começar a criação com algo linear e simples, o padrão inicial em STGEN é um enredo totalmente não-linear e embaraçado, sendo tarefa do autor restringir os eventos a permitir que apenas sequências coerentes com a história sejam contada.

Escrevendo em STGEN alguém deve ser perguntar "o que pode acontecer quando e o que isso muda?" Este paradigma é incomum e pode ser considerado uma barreira. Ao mesmo tempo, a mudança para um paradigma especializado em interatividade e ramificação de conteúdo pode promover resultados proveitosos.

A DSL STGEN apresentada dispõe de recursos limitados, como os tipos de atributo – apenas atributos que representam números inteiro ou valores boleanos. Ainda assim, a

linguagem provou-se bastante expressiva, permitindo construção de narrativas com enredos complexos. Isto devido a liberdade do uso dos atributos, possibilitando assim um autor elaborar formas criativas de utilização dos mesmos.

A geração automática do DN definida na STGEN tornou facilitado o processo de criação e testes das narrativas interativas. Fazendo parte da proposta da linguagem um paradigma de seleção e filtragem de possíveis acontecimentos, era essencial que fosse prático realizar testes durante o processo de criação. Apesar de possuir lógica simples, o DN atende às necessidades básicas para a execução das narrativas produzidas.

Levando em consideração o esquema de categorização proposto por (KYBARTAS; BIDARRA, 2017), apresentado na seção 2.1, pode-se classificar o sistema deste projeto. O sistema – DSL STGEN e o DN – podem ser considerados como um sistema de *Plot Structure* e *Manual Space*. Isto porque na descrição de narrativas, o autor deve criar todo conteúdo do espaço da narrativa – personagens, objetos, etc – manualmente e também todo o conteúdo do enredo. E, em relação ao enredo, o autor deve seguir os padrões da linguagem, que incluem algumas poucas restrições de formação do enredo.

Também na seção 2.1 foi discutido os aspectos de *Intenção autoral*, *Autonomia de personagens virtuais* e *Modelo de jogador*, propostos por (RIEDL; YOUNG, 2006), que qualificam sistemas geradores de narrativas. Aplicando estes conceitos para avaliar o sistema deste trabalho, pode-se dizer que este mantém completamente a intenção autoral, uma vez que o sistema não gera conteúdos novos para uma história. O sistema não permite autonomia dos personagens virtuais, pois os atores da história são apenas abstrações com atributos simples e não possuem lógica própria. E o conceito de modelo de jogador é possível ser aplicado em STGEN, dependendo do autor utilizar os atributos disponíveis com este intuito.

O sistema desenvolvido neste trabalho possui muitas limitações, que serão discutidas a seguir. Considerando seu propósito – uma abordagem para facilitar a criação de narrativas interativas, – uma das maiores limitações é a falta de inteligência do sistema, que se reflete na necessidade de criação manual de todo o conteúdo da narrativa. O ambiente de criação facilita este processo, ao dispor uma linguagem especializada para tal e a geração do DN, porém não é capaz de criar conteúdo independentemente. Isto implica em um crescimento exponencial de conteúdo no formato STGEN, o que dificulta o processo criativo.

A linguagem STGEN é bastante expressiva, mas também bastante complexa. De certa forma, a complexidade é inevitável para garantir um controle mais granular das possíveis sequências narrativas. Porém, ainda falta na linguagem algumas funcionalidades importantes para a criação de histórias, como descrever os personagens de forma mais profunda, incluindo mais características intrínsecas que definem o personagem e seus relacionamentos entre si.

Em certas situações, é possível querer definir certos padrões, ou "regras de interação", de atores ou características recorrentes em objetos. Isto requer algum tipo de generalização ou atribuição de funções dentro no mundo da narrativa. Esse tipo de definição não é possível em STGEN e requer um aprimoramento minucioso na linguagem.

O Diretor de Narrativas (DN) possui métodos simples para integração com outros sistemas, o que permite uma gama diversa de aplicativos. Neste caso, porém, essa simplicidade implica em certas dificuldades. O problema surge para aplicativos que tenham uma apresentação que envolve mais elementos, como vídeo e áudio, pois o DN notifica apenas um *evento* do seu contexto. Isso força às aplicações complexas a replicarem o conteúdo de cada evento no seu formato desejado. O interessante, neste caso, seria especializar o DN para um tipo de aplicativo, dispondo assim de uma integração mais complexa e completa, evitando a redundância de conteúdo.

Afinal, o sistema dispõe um ambiente especializado para descrição de narrativas interativas, incluindo a criação automática de um sistema capaz de guiar as narrativas criadas por autores. Porém, este possui muitas limitações que devem ser resolvidas para torná-lo uma opção viável a ser utilizada no processo criativo das narrativas.

6.3 Trabalhos Futuros

Existem muitas possibilidades de melhoria das propostas apresentadas neste trabalho. Dentre elas, o mais fundamental é aprimorar a expressão da linguagem e algoritmo do DN de forma a possibilitar algum dinamismo no desenvolvimento do enredo das histórias. O primeiro passo pode ser desenvolver a noção de *Contrained Plot*, como descrito em (KYBARTAS; BIDARRA, 2017), incluindo no algoritmo do DN inteligência para escolher e alocar atores a realizarem certas ações na história. Isto também inclui a alteração na linguagem para que exista, para uma ação, uma definição mais abstrata que permita múltiplos atores. Por exemplo, definir que a ação "Vilania" é realizada pelo ator que tem o papel de "Vilão", em vez de atribuir um ator específico à ação.

Como também apresentado em (KYBARTAS; BIDARRA, 2017), o DN e a STGEN podem ser aprimorados de forma a possuírem funcionalidade de *Space modification*. Isto quer dizer que o algoritmo seria capaz de alterar algum existente já definido durante a execução da narrativa, com o intuito de adaptá-lo à situação da história. Isto, em conjunto com o *Contrained Plot*, representaria um grande avanço na geração dinâmica de histórias.

Sob uma perspectiva mais branda, a linguagem pode ser estendida e melhorada em vários aspectos. Pode-se melhorar a expressão da linguagem, incluindo mais atributos com semântica explícita, como para definição de sentimentos, relações e objetivos de personagens. Padrões para definição genérica de atores e objetos possibilitaria aos autores a criação de regras e "rotinas" nos mundos fictícios de forma prática. Também é relevante

a inclusão de mais tipos de atributos, como *strings* e números de ponto flutuantes, para aumentar as possibilidades de uso e permitir lógicas mais complexas.

O DN contém algoritmos de decisão simples e fixos, como descrito previamente. Estes algoritmos não dependem do que o autor descreve e este não consegue influenciar na lógica de decisão. Nesta ótica, pode-se explorar uma expansão da linguagem a permitir que autores influenciem também nos procedimentos de decisão do DN. Alternativamente, pode-se facilmente alterar todos estes procedimentos, trocando-os por lógicas mais adaptadas a um dado contexto. A praticidade da geração de código permite até facilmente gerar procedimentos totalmente diferentes dependendo de parâmetros simples.

Em outro aspecto, pode ser significativo a existência de um editor visual para a linguagem STGEN. Dada a complexidade da linguagem em questão, o editor visual poderia simplificar esta questão ao gerar código STGEN. Esta opção pode tornar mais amigável a criação deste tipo de história, dado que a interface visual permite uma visualização causal dos eventos mais prática e imediata.

Referências

CONCEPCIóN, E.; GERVáS, P.; MéNDEZ, G. A common model for representing stories in automatic storytelling. In: . [s.n.], 2017. Disponível em: <https://www.researchgate.net/publication/321917883_A_Common_Model_for_Representing_Stories_in_Automatic_Storytelling>. Citado 4 vezes nas páginas 6, 21, 22 e 27.

EVANS, R.; SHORT, E. Versu—a simulationist storytelling system. *Computational Intelligence and AI in Games, IEEE Transactions on*, v. 6, p. 113–130, 06 2014. Citado na página 71.

GERVáS, P. Computational approaches to storytelling and creativity. *AI Magazine*, v. 30, n. 3, p. 49–62, 2009. Disponível em: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-78650733279&partnerID=40&md5=d189264d5aa02c06218979d06727a733>. Citado 3 vezes nas páginas 11, 12 e 21.

GERVáS, P. et al. Story plot generation based on cbr. *Knowledge-Based Systems*, v. 18, n. 4-5, p. 235–242, 2005. Disponível em: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-22544470647&doi=10.1016%2fj.knosys.2004.10.011&partnerID=40&md5=9ae8969227cb455b16ae0d349b543736>. Citado 5 vezes nas páginas 6, 12, 17, 18 e 72.

KYBARTAS, B.; BIDARRA, R. A survey on story generation techniques for authoring computational narratives. *IEEE Transactions on Computational Intelligence and AI in Games*, v. 9, n. 3, p. 239–253, 2017. Cited By 8. Disponível em: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85030110341&doi=10.1109%2fTCIAIG.2016.2546063&partnerID=40&md5=ccc9eda7d66d3e427f2690b52b1b33b6>. Citado 7 vezes nas páginas 6, 19, 21, 70, 71, 73 e 74.

LEWIS, M.; JACOBSON, J. Games engines in scientific research. *Communications of the ACM*, v. 45, n. 1, p. 27–31, 2002. Cited By 173. Disponível em: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-0345806559&partnerID=40&md5=0251ff2a506f00fcfebfeebb8d87e114>. Citado 2 vezes nas páginas 6 e 24.

MAGERKO, B. Story representation and interactive drama. In: . [s.n.], 2005. p. 87–92. Disponível em: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-84864865666&partnerID=40&md5=dc1881e862ac4a9d516ca86526144f9d>. Citado 2 vezes nas páginas 12 e 72.

MATEAS, M.; STERN, A. Integrating plot , character and natural language processing in the interactive drama façade. In: . [S.l.: s.n.], 2002. Citado 2 vezes nas páginas 12 e 71.

MERNIK, M.; HEERING, J.; SLOANE, A. When and how to develop domain-specific languages. *ACM Computing Surveys*, v. 37, n. 4, p. 316–344, 2005. Disponível em: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-33745167684&doi=10.1145%2f1118890.1118892&partnerID=40&md5=f5ddc8e076341d14a49146ca180c822e>. Citado 2 vezes nas páginas 12 e 23.

MONTGOMERY, R. *The Abominable Snowman*. Chooseco, 2006. Disponível em: <https://www.goodreads.com/book/show/190930.The_Abominable_Snowman>. Citado na página 16.

PORTEOUS, J.; CAVAZZA, M.; CHARLES, F. Applying planning to interactive storytelling: Narrative control using state constraints. *ACM Transactions on Intelligent Systems and Technology*, v. 1, n. 2, 2010. Disponível em: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-79955059589&doi=10.1145%2f1869397.1869399&partnerID=40&md5=92339061a01e96fdb08257f91b8242ce>. Citado 2 vezes nas páginas 12 e 21.

RIEDL, M.; BULITKO, V. Interactive narrative: An intelligent systems approach. *AI Magazine*, v. 34, n. 1, p. 67–77, 2013. Disponível em: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-84876178157&partnerID=40&md5=d8dfb9e97f2e15120bf558d9a722d93b>. Citado 9 vezes nas páginas 6, 11, 12, 13, 15, 16, 17, 18 e 21.

RIEDL, M.; YOUNG, R. From linear story generation to branching story graphs. *IEEE Computer Graphics and Applications*, v. 26, n. 3, p. 23–31, 2006. Disponível em: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-33744999663&doi=10.1109%2fMCG.2006.56&partnerID=40&md5=f32a7691ed3b69f1bc0691349b7f4913>. Citado 6 vezes nas páginas 6, 11, 15, 18, 21 e 73.

RIEDL, M. O. et al. Dynamic experience management in virtual worlds for entertainment, education, and training. *International Transactions on Systems Science and Applications - ITSSA*, v. 4, 01 2008. Citado 3 vezes nas páginas 12, 16 e 72.

SHARMA, M. et al. Drama management and player modeling for interactive fiction games. *Computational Intelligence*, v. 26, n. 2, p. 183–211, 2010. Disponível em: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-77951699329&doi=10.1111%2fj.1467-8640.2010.00355.x&partnerID=40&md5=582cd03d5cf2dc458f88aebd364e7371>. Citado na página 12.

YOUNG, R. M. An architecture for integrating plan-based behavior generation with interactive game environments. In: . [S.l.: s.n.], 2003. Citado na página 25.

Apêndices

APÊNDICE A – Sintaxe STGEN Completa

Neste capítulo estão descritos todos os elementos da linguagem STGEN, incluindo sua forma sintática e exemplos. A seguir, estão dispostos em ordem *top-down*, onde primeiro são apresentados os elementos que representam a história como um todo, e aos poucos cada parte da sintaxe é detalhada.

Story

Elemento que representa a história como um todo. É o recipiente total, contendo dados da história, o espaço da narrativa e o enredo.

Sintaxe

```
1 data:StoryData space:Space plot:Plot
```

Exemplo

```
1 Story data {...}
2 Space {...}
3 Plot {...}
```

StoryData

Representa dados gerais da história, sendo composto por uma coleção de atributos. Utilizado para armazenar dados na narrativa que não pertencem a um existente qualquer, e sim a narrativa em si.

Sintaxe

```
1 Story data { attributes:Attribute* }
```

Exemplo

```
1 Story data {
2     quantity ato=1
3     fact acabando=false
4 }
```

Space

É o espaço de conteúdo que a história pode fazer uso, isto é, tudo aquilo que existe no mundo desta narrativa. Contém um conjunto de existentes da história, como personagens e objetos.

Sintaxe

```
1 Space { existents:Existent* }
```

Exemplo

```
1 Space {
2     actor Fulano {...}
3     object Vela {...}
4     object Balde {...}
5 }
```

Plot

Caracteriza o enredo da narrativa, contendo um conjunto dos eventos que podem ocorrer.

Sintaxe

```
1 Plot { events:Event* }
```

Exemplo

```
1 Plot {
2     event Incendio {...}
3 }
```

Existent

Representa algo que existe na história. É um elemento abstrato que pode ser do tipo *Actor* ou *Object*.

Sintaxe

```
Actor | Object
```

Actor

Representa um ator na história, que pode disparar eventos, alterando o fluxo da narrativa. Contém um nome de exibição e um conjunto de atributos. Pode conter o marcador "player"prefixando o ator para sinalizá-lo como o ator que representa o usuário que interage com a história.

Sintaxe

```
player? actor id:ID { actorName:String
    attributes:Attribute* }
```

Exemplo

```
actor Fulano {
    name: "Fulano da Silva"
    quantity dinheiro=100
    fact saudavel=true
}

player actor Jogador {...}
```

Object

Representa um objeto na história, utilizado para armazenar dados relativos a um elemento relevante da narrativa que não atua por si só. Contém um conjunto de atributos que o caracteriza.

Sintaxe

```
object id:ID { attributes:Attribute* }
```

Exemplo

```
object Vela {
    fact acesa=false
}
```

Attribute

Representa um dado de algum existente da história. É um elemento abstrato que pode ser do tipo *FactAttribute* ou *QuantityAttribute*.

Sintaxe

```
FactAttribute | QuantityAttribute
```

FactAttribute

Atributo do tipo boleano presente em existentes ou no *StoryData*. Possui um identificador e um valor inicial ou uma expressão boleana.

Sintaxe

```
fact id:ID = (initValue:BooleanTerm |
    macroExp:Expression)
```

Exemplo

```
fact saudavel=true
fact rico = Fulano.dinheiro > 500
```

QuantityAttribute

Atributo do tipo inteiro presente em existentes ou no *StoryData*. Possui identificador, um valor inicial e opcionalmente uma configuração de atributo.

Sintaxe

```
quantity id:ID = initValue:IntegerTerm
    setting:AttributeSetting?
```

Exemplo

```
quantity dinheiro = 100
quantity tempo = 0 [increments every
    event]
```

AttributeSetting

Configuração de atributos inteiros que adiciona um comportamento de contador condicional.

Sintaxe

```
(increments|decrements) by amount:INT
    frequency:
    AttributeSettingFrequency
```

Exemplo

```
[increments by 1 every event]
[increments by 4 every action]
[decrements by 2 every player action]
[decrements by 1 every happening]
```

Event

Representa um evento que ocorre no enredo da narrativa. É um elemento abstrato que pode ser do tipo *HappeningEvent* ou *ActionEvent*.

Sintaxe

```
HappeningEvent | ActionEvent
```

HappeningEvent

Evento que representa um acontecimento na história. Ocorre independente de atores. Contém um de valor boleano que indica se ele é evento de abertura ou evento de fechamento; um identificador; um *EventPriority* que indica sua prioridade; duas strings para armazenar a descrição completa e a descrição pequena; opcionalmente um evento que ele desencadeia; um conjunto de eventos que podem ser desencadeados; um conjunto de condições para validar o evento; e um conjunto de modificações que esse evento aplica.

Sintaxe

```
(opening|ending)? event id:ID {
    priority:EventPriority desc:STRING
     shortDesc:STRING trigger:Event?
    mayTriggers:Event* conditions:
    Condition* changes:Change* }
```

Exemplo

```
event Incendio {
    priority: high
    description: "A vela caiu e
        provocou um incêncio na casa."
    require: Vela.acesa == true
    change: Casa.emChamas = true
}
```

ActionEvent

Evento que caracteriza uma ação realizada por um ator da história. Contém um identificador; um *EventPriority* que indica sua prioridade; um *ActionType* para mapear um dos tipos genéricos de ação; duas strings para armazenar a descrição completa e a descrição pequena; opcionalmente um evento que ele desencadeia; um conjunto de eventos que podem ser desencadeados; um conjunto de condições para validar o evento; e um conjunto de modificações que esse evento aplica.

Sintaxe

```
action id:ID { priority:EventPriority
    eventType:ActionType desc:STRING
    shortDesc:STRING eventActor:Actor
    trigger:Event? mayTriggers:Event*
```

```
    conditions:Condition* changes:
    Change* }
```

Exemplos

```
action TentarApagarFogo {
    type: interaction
    description: "Com um balde de água
        , você tenta apagar o fogo,
        jogando todo a água de uma vez
        ."
    short-description: "Tentar apagar
        fogo com a água."
    actor: Jogador
    may-trigger: FogoApagado
    require: Balde.comAgua == true
    change: Incendio.intensidade -= 5
}
```

Condition

Representa uma condição de um evento que deve ser satisfeita. É um elemento abstrato que pode ser do tipo *ExpressionCondition*, *OrderCondition* ou *TriggerCondition*.

Sintaxe

```
ExpressionCondition | OrderCondition |
    TriggerCondition
```

ExpressionCondition

Condição de evento que contém uma expressão boleana que indica se o evento é valido ou não em um determinado momento.

Sintaxe

```
require: exp:Expression
```

Exemplos

```
require: Vela.acesa == true
require: (Balde.comAgua) and (Incendio
    .intensidade < 5)
```

OrderCondition

Condição de evento que verifica ordem temporal relativa a outro evento para validação. Isto é, verifica se um outro evento já ocorreu ou não, e isto qualifica o primeiro evento caso seja correspondente a ordem descrita em *Order*.

Sintaxe

```
require: order:Order compEvent:Event
```

Exemplos

```
require: after Incendio
require: before TentarApagarFogo
```

TriggerCondition

Condição de evento que determina que o evento deve ser desencadeado diretamente por outro evento, através de algum *Trigger*.

Sintaxe

```
require: be-triggered
```

Exemplo

```
require: be-triggered
```

Change

Representa uma modificação a ser feita em algum atributo. É um elemento abstrato que pode ser do tipo *ExistentChange* ou *StoryDataChange*.

Sintaxe

```
ExistentChange | StoryDataChange
```

ExistentChange

Modificação a ser feita em um atributo de um existente. Contém referência ao existente, ao atributo do existente, um operador de atribuição e um termo do tipo inteiro ou boleano, que deve corresponder ao tipo do atributo do existente.

Sintaxe

```
change: existent:Existent . attribute:
    Attribute op:AttributionOperator
    term:(IntegerTerm|BooleanTerm)
```

Exemplos

```
change: Incendio.intesidade -= 5
```

StoryDataChange

Uma modificação igual a *ExistentChange*, porém específica para os atributos contidos em *StoryData*. Desta forma não precisa de uma referência a algum existente.

Sintaxe

```
change: Story . attribute:Attribute op
    :AttributionOperator term:(
    IntegerTerm|BooleanTerm)
```

Exemplos

```
change: Story.ato = 2
```

Expression

Representa uma expressão boleana da linguagem. É um elemento abstrato que só pode ser do tipo *BinaryExpression*.

Sintaxe

```
BinaryExpression
```

BinaryExpression

Expressão binária da linguagem. É um elemento abstrato que pode ser do tipo *UnitExpression*, *OrExpression*, *AndExpression* ou *ComparisonExpression*.

Sintaxe

```
UnitExpression | OrExpression |
    AndExpression |
    ComparisonExpression
```

OrExpression

Expressão de OU lógico da linguagem. Formada por duas ou mais *UnitExpression* conectadas pelo termo "or".

Sintaxe

```
operands:UnitExpression (or operands:
    UnitExpression)+
```

Exemplos

```
Vela.acesa or Fio.emCurto or Fogao.
    vazandoGas
```

AndExpression

Expressão de E lógico da linguagem. Formada por duas ou mais *UnitExpression* conectadas pelo termo "and".

Sintaxe

```
operands:UnitExpression (and operands:
    UnitExpression)+
```

Exemplos

```
(Balde.comAgua) and (Incendio.
    intensidade < 5)
```

ComparisonExpression

Expressão de comparação da linguagem. Formada por duas *UnitExpression* conectadas por um operador de comparação.

Sintaxe

```
1 left:UnitExpression op:
      ComparisonOperator right:
      UnitExpression
```

Exemplos

```
1 Jogador.dinheiro >= 50
2 Story.ato == 3
```

UnitExpression

Elemento abstrato que pode ser dos tipos *NegationExpression* ou *UnitaryExpression*.

Sintaxe

```
1 NegationExpression | UnitaryExpression
```

NegationExpression

Expressão de negação, que contém uma *UnitExpression*.

Sintaxe

```
1 not operand:UnitExpression
```

Exemplos

```
1 not Jogador.estaDormindo
2 not (Incendio.intesidade < 10)
```

UnitaryExpression

Elemento abstrato que pode ser uma *Expression* entre parêntesis ou um *Term*.

Sintaxe

```
1 ( Expression ) | Term
```

Exemplos

```
1 (Vela.acesa)
```

Term

Representa um terminal ou um atributo de um existente. Pode ser do tipo *ExistentTerm*, *StoryAttributeTerm*, *IntegerTerm* ou *BooleanTerm*.

Sintaxe

```
1 ExistentTerm | StoryAttributeTerm |
      IntegerTerm | BooleanTerm
```

ExistentTerm

Terminal que contém referência a um atributo de um existente.

Sintaxe

```
1 existent:Existent . attribute:
      Attribute
```

Exemplos

```
1 Balde.comAgua
```

StoryAttributeTerm

Terminal que contém referência a um atributo da história.

Sintaxe

```
1 Story . attribute:Attribute
```

Exemplos

```
1 Story.ato
```

IntegerTerm

Terminal do tipo inteiro.

Sintaxe

```
value:INT
```

Exemplos

```
5
14
```

BooleanTerm

Terminal do tipo boleano.

Sintaxe

```
value:(true|false)
```

Exemplos

```
true
false
```

APÊNDICE B – Sintaxe Completa da STGEN em Xtext

Neste capítulo está a definição completa da linguagem STGEN em Xtext. O código a seguir contém a implementação em Xtext das definições da linguagem presentes no apêndice A e gera o metamodelo da linguagem presente no apêndice C.

Código B.1 – Sintaxe Completa da DSL STGEN em Xtext.

```
1  grammar br.ufes.inf.pg.StoryGen with org.eclipse.xtext.common.Terminals
2
3  generate storyGen "http://www.ufes.br/inf/pg/StoryGen"
4
5  Story:
6    data=StoryData
7    space=Space
8    plot=Plot
9  ;
10
11 StoryData:
12   {StoryData} 'Story' 'data' '{'
13     attributes+=Attribute*
14   '}'
15 ;
16
17 Space:
18     {Space} 'Space' '{'
19         existents+=Existent*
20     '}'
21 ;
22
23 Plot:
24   {Plot} 'Plot' '{'
25     events+=Event*
26   '}'
27 ;
28
29 Existent: Actor | Object ;
30
31 Actor:
32   (playable?='player')? 'actor' name=ID
33   '{' 'name:' displayName=STRING attributes+=Attribute* '}'
34 ;
35
36 Object:
37   'object' name=ID
38   '{' attributes+=Attribute* '}'
39 ;
40
41 Attribute: FactAttribute | QuantityAttribute ;
```

```
42
43 FactAttribute:
44   'fact' name=ID '=' (initValue=BooleanTerm  | '(' macroExp=Expression ')')
45 ;
46
47 QuantityAttribute:
48   'quantity' name=ID '=' initValue=IntegerTerm ('['setting=AttributeSetting']')?
49 ;
50
51 AttributeSetting:
52   (positive?='increments' | negative?='decrements') 'by' amount=INT frequency=
       AttributeSettingFrequency
53 ;
54
55 Event: HappeningEvent | ActionEvent;
56
57 HappeningEvent:
58   (opening?='opening' | ending?='ending')?
59   'event' name=ID '{'
60     ('priority:' priority=EventPriority)?
61     'description:' description=STRING
62     ('short-description:' shortDescription=STRING)?
63     ('triggers:' trigger=[Event])?
64     ('may-trigger:' (mayTriggers+=[Event] ',')* mayTriggers+=[Event])?
65     conditions+=Condition*
66     changes+=Change*
67   '}'
68 ;
69
70 ActionEvent:
71   'action' name=ID '{'
72     ('priority:' priority=EventPriority)?
73     'type:' type=ActionType
74     'description:' description=STRING
75     ('short-description:' shortDescription=STRING)?
76     'actor:' actor=[Actor]
77     ('triggers:' trigger=[Event])?
78     ('may-trigger:' (mayTriggers+=[Event] ',')* mayTriggers+=[Event])?
79     conditions+=Condition*
80     changes+=Change*
81   '}'
82 ;
83
84 Condition:
85   ExpressionCondition |
86   OrderCondition      |
87   TriggerCondition    ;
88
89 ExpressionCondition:
90   'require:' exp=Expression
91 ;
92
93 OrderCondition:
94   'require:' order=Order event=[Event]
95 ;
96
97 TriggerCondition:
```

```
98    'require:' trigger?='be-triggered'
99  ;
100
101 Change:
102   ExistentChange |
103   StoryDataChange  ;
104
105 ExistentChange:
106   'change:' existent=[Existent] '.' attribute=[Attribute]
107   op=AttributionOperator term=(IntegerTerm|BooleanTerm)
108 ;
109
110
111 StoryDataChange:
112   'change:' 'Story' '.' attribute=[Attribute]
113   op=AttributionOperator term=(IntegerTerm|BooleanTerm)
114 ;
115
116 Expression:
117   BinaryExpression
118 ;
119
120 BinaryExpression returns Expression:
121   UnitExpression
122     (  ({OrExpression.operands+=current}   'or'   operands+=UnitExpression ('or'
            operands+=UnitExpression)*)
123      | ({AndExpression.operands+=current}  'and'  operands+=UnitExpression ('and'
            operands+=UnitExpression)*)
124      | ({ComparisonExpression.left=current} operator=ComparisonOperator right=
            UnitExpression)
125     )?
126 ;
127
128
129 UnitExpression returns Expression:
130   NegationExpression |
131   UnitaryExpression  ;
132
133 NegationExpression returns Expression:
134   {NegationExpression} 'not' operand=UnitExpression
135 ;
136
137 UnitaryExpression returns Expression:
138     "(" Expression ")"
139     | Term
140 ;
141
142 Term:
143   ExistentAttributeTerm |
144   StoryAttributeTerm |
145   IntegerTerm    |
146   BooleanTerm    ;
147
148 ExistentAttributeTerm:
149   existent=[Existent] '.' attribute=[Attribute]
150 ;
151
```

```
152 StoryAttributeTerm:
153   'Story' '.' attribute=[Attribute]
154 ;
155
156
157 IntegerTerm:
158   value=INT
159 ;
160
161 BooleanTerm:
162   value=('true'|'false')
163 ;
164
165 enum ActionType:
166   VISUAL='visual' |
167   INTERACTION='interaction' |
168   MOVEMENT='movement' |
169   COMMUNICATION='communication' |
170   COMBAT='combat' |
171   IDLE='idle'
172 ;
173
174 enum ComparisonOperator :
175   LT="<"   |
176   LE="<="  |
177   EQ="=="  |
178   NEQ="!=" |
179   GE=">="  |
180   GT=">"   ;
181
182 enum AttributionOperator:
183   ASSIGN='='      |
184   INCREMENT='+=' |
185   DECREMENT='-=' ;
186
187 enum Order:
188   AFTER='after'   |
189   BEFORE='before' ;
190
191 enum AttributeSettingFrequency:
192   EVERY_EVENT='every event' |
193   EVERY_HAPPENING='every happening' |
194   EVERY_ACTION='every action' |
195   EVERY_PLAYER_ACTION='every player action'
196 ;
197
198 enum EventPriority:
199   NORMAL='normal' |
200   HIGH='high' |
201   VERY_HIGH='veryHigh'
202
203 ;
```

APÊNDICE C – Modelo ECore da DSL STGEN

Neste capítulo estão dispostas figuras que contém todo o metamodelo da linguagem STGEN, gerado a partir da sua definição em Xtext apresentada no apêndice B. Por conta do tamanho extenso, o diagrama foi quebrado em três figuras.

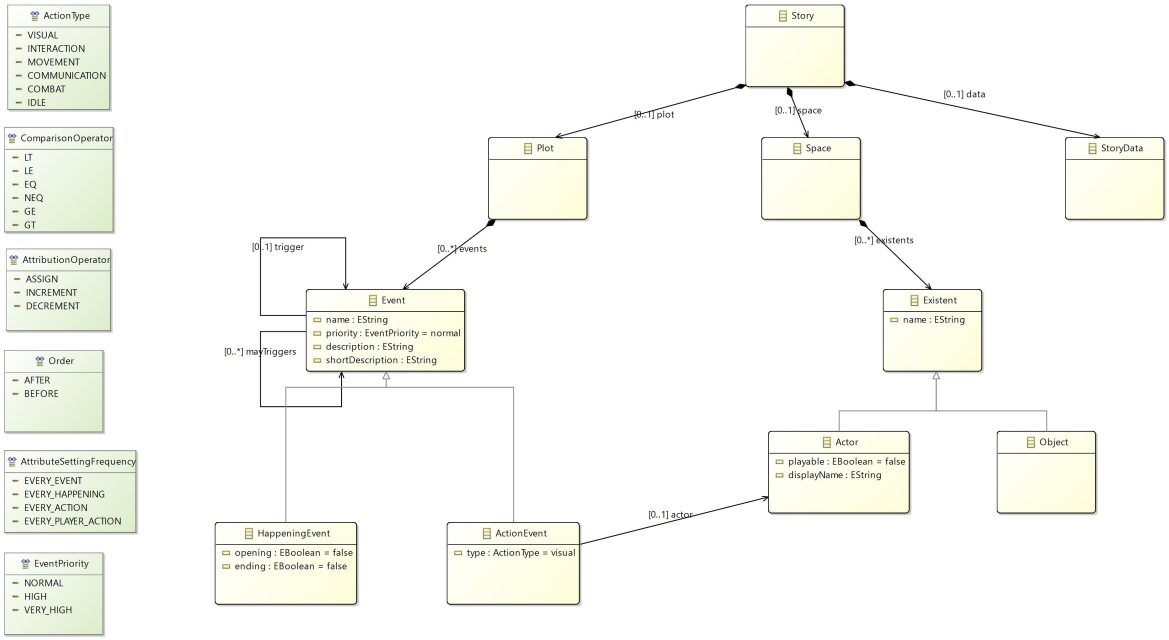

Figura 18 – Parte 1 de 3 do modelo ECore da linguagem STGEN.

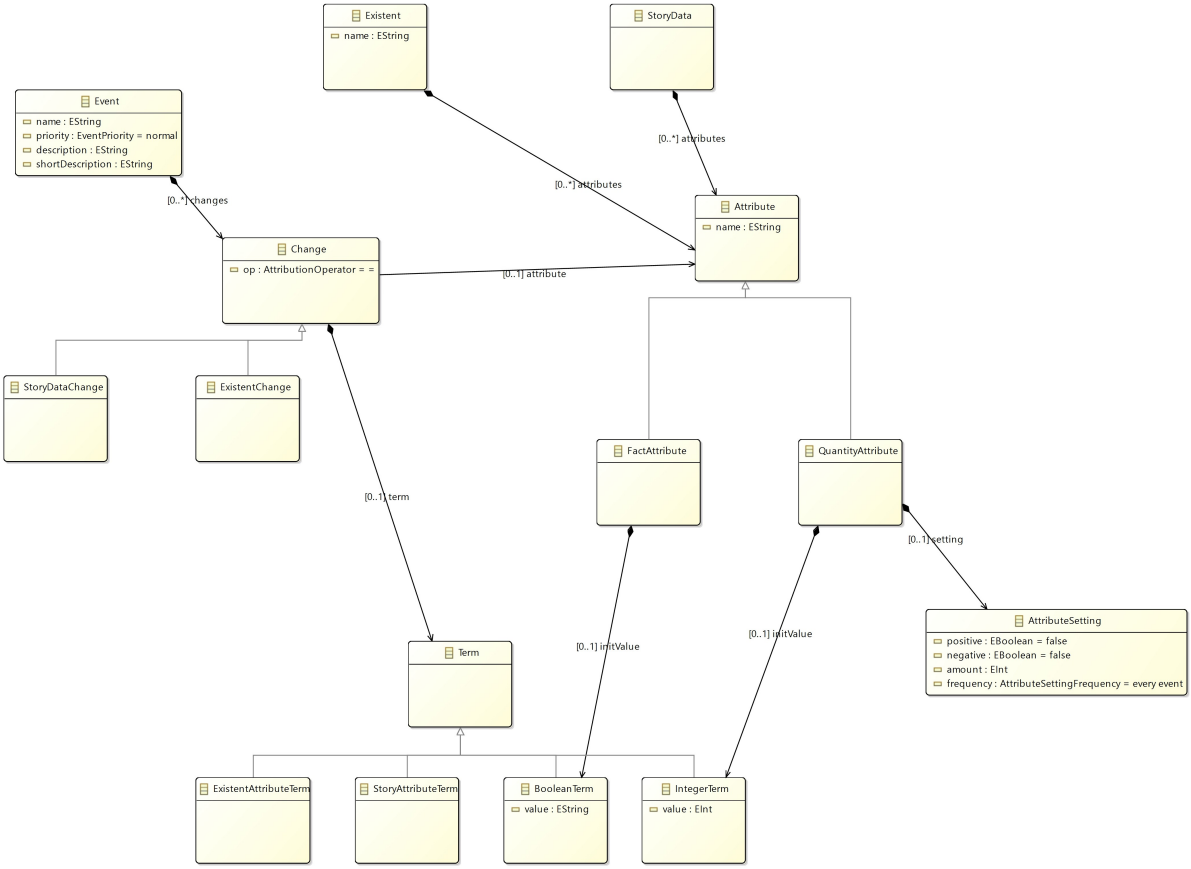

Figura 19 – Parte 2 de 3 do modelo ECore da linguagem STGEN.

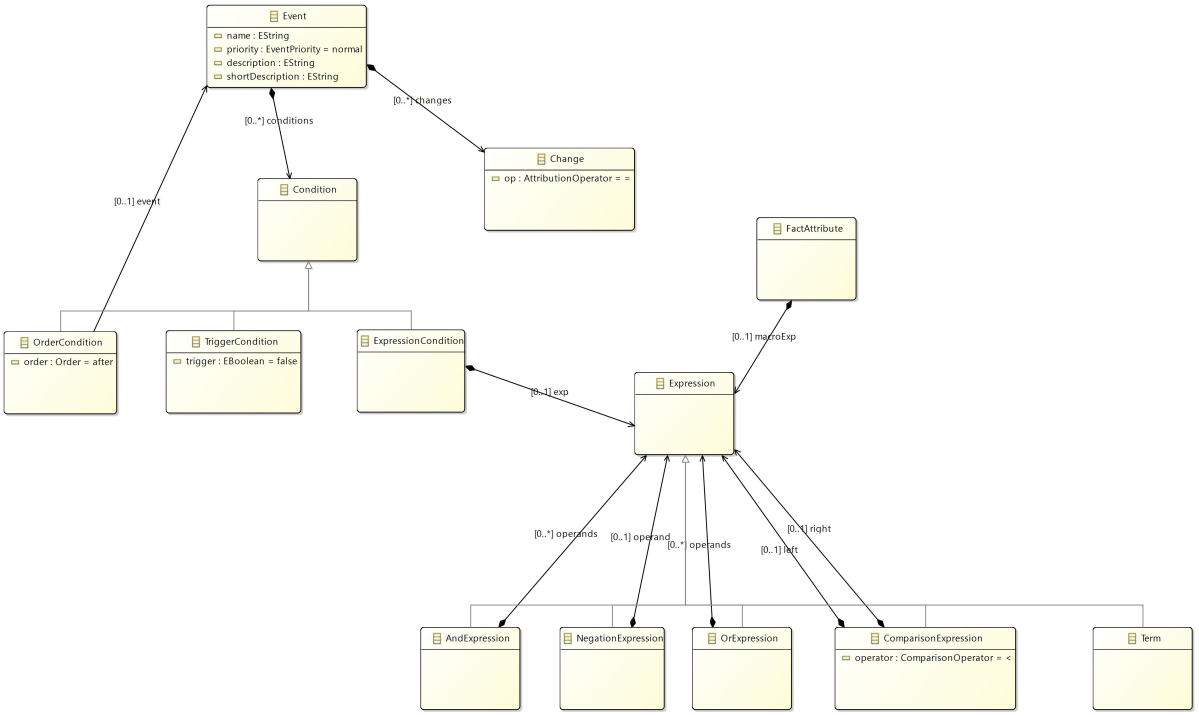

Figura 20 – Parte 3 de 3 do modelo ECore da linguagem STGEN.