Compiladores para as Linguagens da Abordagem WED-flow

Eduardo Dias Filho

DISSERTAÇÃO APRESENTADA
AO
INSTITUTO DE MATEMÁTICA E ESTATÍSTICA
DA
UNIVERSIDADE DE SÃO PAULO
PARA
OBTENÇÃO DO TÍTULO
DE
MESTRE EM CIÊNCIAS

Programa: Ciência da Computação
Orientador: Prof. Dr. João Eduardo Ferreira

São Paulo, Julho de 2021

Compiladores de linguagens de modelagem e controle da abordagem WED-flow

Esta versão da dissertação contém as correções e alterações sugeridas pela Comissão Julgadora durante a defesa da versão original do trabalho, realizada em 02/06/2021. Uma cópia da versão original está disponível no Instituto de Matemática e Estatística da Universidade de São Paulo.

Comissão Julgadora:

- Prof. Dr. João Eduardo Ferreira (orientador) - IME-USP
- Prof. Dr. Luiz Camolesi Júnior - UNICAMP
- Prof. Dr. Rafael Liberato Roberto - UTFPR

Agradecimentos

Agradeço ao meu orientador, o Prof. Dr. João Eduardo Ferreira, cujo apoio, dedicação e ensinamentos foram indispensáveis para a conclusão deste trabalho e para meu aprendizado durante o mestrado.

Agradeço aos meus pais Paula e Eduardo não só pelo apoio incondicional, mas também pelos imensuráveis sacrifícios que fizeram para que eu pudesse ter a oportunidade de estudar.

Agradeço à minha irmã Lais e ao meu irmão Guilherme por me apoiarem durante a vida toda.

Agradeço aos meus amigos Edson, Felipe, Rafael, Rodrigo e Vinicius pelo apoio, sugestões e ensinamentos valiosos durante o mestrado.

Agradeço aos amigos do DataGroup, que contribuíram com meu trabalho com ensinamentos, sugestões e críticas construtivas.

Resumo

DIAS FILHO, E. **Compiladores para as Linguagens da Abordagem WED-flow**. Dissertação (Mestrado) - Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2020.

A abordagem WED-flow propõe modelar e implementar Sistemas de Informação Cientes de Processos (PAIS) a partir da especificação de estruturas que representam eventos, dados e tarefas de um processo de negócio, oferecendo a execução de instâncias do processo em um ambiente transacional definido sobre um banco de dados. A definição de estruturas que compõem o modelo WED-flow é expressa em uma linguagem declarativa de alto nível baseada em notação de conjuntos. A implementação atual do WED-flow permite que as definições sejam especificadas em uma linguagem intermediária chamada WED-SQL, que é traduzida para código SQL. No entanto, o código que realiza esta tradução não segue as etapas de compilação e não é criado a partir de uma descrição formal da gramática, o que torna o tradutor de difícil manutenção e propenso a erros. Este trabalho propõe a implementação de um compilador da linguagem WED-SQL para a linguagem destino SQL e de um outro compilador da linguagem declarativa de alto nível do WED-flow para a linguagem destino WED-SQL, ambos envolvendo as etapas de análise léxica, análise sintática, análise semântica e geração de código na linguagem destino. Com o uso de uma ferramenta de geração automática de analisadores sintáticos e léxicos a partir de regras definidas, os compiladores facilitam possíveis alterações nas regras de tradução da linguagem.

Palavras-chave: WED-flow, WED-SQL, Compilador, Análise sintática, Análise léxica, Análise semântica.

Abstract

DIAS FILHO, E. **Compilers for the WED-flow Approach's Languages**. Dissertation (Master's) - Institute of Mathematics and Statistics, University of São Paulo, São Paulo, 2020.

The WED-flow approach proposes a way to model and implement Process-Aware Information Systems (PAIS) from definitions of structures that represent events, dataflow and workflow of a business process, supporting the execution of the process instances in a transactional environment defined over a database. These definitions are expressed in a high-level declarative language based on set notation. The current WED-flow implementation allows the definitions to be expressed in an intermediate declarative language called WED-SQL, which translates to SQL code. However, the current WED-SQL to SQL translator code does not follow the stages of compiling and was not built based on a formal description of the grammar, which makes the translator difficult to maintain and prone to errors. In this work, we propose the development of a compiler that translates WED-SQL commands to SQL language and another compiler to translate commands written in high-level declarative language based on set notation to WED-SQL language, both compilers involving the lexical analysis, syntactic analysis, semantic analysis and code generation stages. With the help of a parser generator tool, the compilers make easy to implement future changes in the language translation rules.

Keywords: WED-flow, WED-SQL, Compiler, Syntactic Analysis, Lexical Analysis, Semantic Analysis.

Sumário

Lista de Abreviaturas	ix
Lista de Figuras	x
Lista de Tabelas	xi
Lista de Códigos	xiv
1 Introdução	1
1.1 Objetivos	5
1.2 Contribuições	5
1.3 Organização do Trabalho	6
2 Fundamentos	7
2.1 WED-flow	7
2.1.1 Exemplo de modelo WED-flow	9
2.1.2 Implementação do WED-flow	11
2.2 Linguagens	17
2.2.1 Formalismo de Backus-Naur Estendido	18
2.2.2 Árvore Sintática	18
2.3 Compilação	20
2.3.1 Algoritmos de Análise Sintática	21
2.3.2 Geração de Analisadores Léxicos e Sintáticos	23
2.3.3 ANTLR	23
2.3.4 Considerações finais	24
3 Compilador da linguagem WED-SQL	27
3.1 Compilador WED-SQL	27
3.2 Análise Léxica	29
3.3 Análise Sintática	32
3.4 Análise Semântica	35
3.5 Tradução para SQL	36
3.6 Tratamento de Erros	38
3.6.1 Considerações finais	39

4	Compilador da linguagem de modelagem WED-flow	41
4.1	A linguagem WED-flow	42
4.2	Análise Léxica	44
4.3	Análise Sintática	46
4.4	Análise Semântica	47
4.5	Tradução para WED-SQL	49
4.5.1	Considerações finais	50
5	Estudo de caso	53
5.1	O sistema SISAUT	53
5.1.1	Entrevista	54
5.1.2	Coleta de materiais	55
5.1.3	Processamento de materiais	55
5.2	Modelagem WED-flow do sistema SISAUT	56
5.2.1	Comandos WED-SQL	59
5.3	Simulação	61
5.3.1	Criação dos bancos de dados	61
5.3.2	Implementação das transições	65
5.3.3	Resultados	67
5.4	Considerações finais	71
6	Conclusões	73
6.1	Próximos passos	74
	Referências Bibliográficas	75

Lista de Abreviaturas

ALL	<i>Adaptative Left-to-right Leftmost derivation</i>
ANTLR	<i>ANother Tool for Language Recognition</i>
BPM	<i>Business Process Management</i>
BSD	<i>Berkeley Software Distribution</i>
CFG	Gramática livre de contexto (<i>Context Free Grammar</i>)
EBNF	Formalismo de Backus-Naur Estendido (<i>Extended Backus-Naur Form</i>)
GLR	<i>Generalized Left-to-right Rightmost derivation</i>
GPL	<i>General Public License</i>
LALR	<i>Look-Ahead Left-to-right Rightmost derivation</i>
LGPL	<i>Lesser General Public License</i>
LL	<i>Left-to-right Leftmost derivation</i>
LR	<i>Left-to-right Rightmost derivation</i>
PAIS	Sistema de informação ciente de processos (<i>Process Aware Information System</i>)
RL	<i>Right-to-left Leftmost derivation</i>
RR	<i>Right-to-left Rightmost derivation</i>
SGBDR	Sistema Gerenciador de Banco de Dados Relacional
SLR	<i>Simple Left-to-right Rightmost derivation</i>
SQL	<i>Structured Query Language</i>
SVOC	Serviço de Verificação de Óbitos da Capital
WED	<i>Work, event and data</i>
WSBPEL	<i>Web Services Business Execution Language</i>

Lista de Figuras

2.1	Arquitetura da abordagem WED-flow sem o uso de compiladores.	16
2.2	Arquitetura da abordagem WED-flow com o uso de compiladores.	17
2.3	Exemplo de árvore sintática.	19
2.4	Etapas da compilação.	20
3.1	Arquitetura do compilador da linguagem WED-SQL.	28

Lista de Tabelas

2.1	Símbolos da variante de EBNF usada.	18
5.1	Resultado da execução do comando “list wed-attributes;” no programa “wsq1” para obter a listagem de WED-attributes do modelo dos processos de coleta e entrevista. .	62
5.2	Resultado da execução do comando “list wed-conditions;” no programa “wsq1” para obter a listagem de WED-conditions do modelo dos processos de coleta e entrevista. .	62
5.3	Resultado da execução do comando “show final wed-state;” no programa “wsq1” para exibir a expressão da condição final do modelo dos processos de coleta e entrevista. .	63
5.4	Resultado da execução do comando “list wed-transitions;” no programa “wsq1” para obter a listagem de WED-transitions do modelo dos processos de coleta e entrevista. .	63
5.5	Resultado da execução do comando “list wed-triggers;” no programa “wsq1” para obter a listagem de WED-triggers do modelo dos processos de coleta e entrevista.	63
5.6	Resultado da execução do comando “list wed-attributes;” no programa “wsq1” para obter a listagem de WED-attributes do modelo do processo de processamento de materiais.	64
5.7	Resultado da execução do comando “list wed-conditions;” no programa “wsq1” para obter a listagem de WED-conditions do modelo do processo de processamento de materiais.	64
5.8	Resultado da execução do comando “show final wed-state;” no programa “wsq1” para exibir a expressão da condição final do modelo do processo de processamento de materiais.	64
5.9	Resultado da execução do comando “list wed-transitions;” no programa “wsq1” para obter a listagem de WED-transitions do modelo do processo de processamento de materiais.	64
5.10	Resultado da execução do comando “list wed-triggers;” no programa “wsq1” para obter a listagem de WED-triggers do modelo do processo de processamento de materiais. .	65
5.11	Projetos usados pelas WED-transitions na simulação.	67
5.12	Resultado do comando de listagem de instâncias no modelo de coleta e entrevista após a execução das instâncias.	68
5.13	Resultado do comando de exibição do histórico da instância do modelo de entrevista e coleta com body_id 1 após sua execução.	69
5.14	Resultado do comando de listagem de instâncias no modelo de processamento após a execução das instâncias.	70

5.15 Resultado do comando de exibição do histórico da instância do modelo de processamento com <code>body_id</code> 1 e material coração.	71
---	----

Lista de Códigos

1.1	Trecho de código Python da versão inicial do interpretador de comandos WED-SQL.	2
2.1	Definição dos WED-attributes do exemplo de locação de carros usando a abordagem WED-flow.	9
2.2	Definição das WED-conditions do exemplo de locação de carros usando a abordagem WED-flow.	10
2.3	Definição das WED-transitions do exemplo de locação de carros usando a abordagem WED-flow.	10
2.4	Definição das WED-triggers do exemplo de locação de carros usando a abordagem WED-flow.	10
2.5	Definição do modelo WED-flow do exemplo de locação de carros usando a abordagem WED-flow.	10
2.6	Comando WED-SQL de criação de modelo WED-flow.	11
2.7	Comando WED-SQL de remoção de modelo WED-flow.	11
2.8	Comando WED-SQL de listagem de modelos WED-flow.	11
2.9	Comando WED-SQL de criação de WED-attribute.	11
2.10	Comando WED-SQL de criação de WED-condition.	12
2.11	Comando WED-SQL de criação de WED-transition.	12
2.12	Comando WED-SQL de criação de WED-triggers.	12
2.13	Comandos WED-SQL de configuração e exibição de condição de WED-state final.	13
2.14	Comandos WED-SQL de ativação e desativação de WED-trigger respectivamente.	13
2.15	Comando WED-SQL de criação de instância.	13
2.16	Comandos WED-SQL de cancelamento e restabelecimento de instâncias.	13
2.17	Comandos WED-SQL de listagem de WED-attributes, WED-conditions, WED-transitions e WED-triggers respectivamente.	14
2.18	Comando WED-SQL de listagem de WED-transitions disparadas.	14
2.19	Comandos WED-SQL de listagem de instâncias.	14
2.20	Comando WED-SQL de exibição de histórico.	14
2.21	Comando WED-SQL de conexão a um modelo WED-flow.	14
2.22	Comandos WED-SQL de início, confirmação e cancelamento de uma transação.	15
2.23	Comandos WED-SQL para especificação do processo de exemplo de locação de carros.	15
2.24	Versão simplificada de regras da gramática que gera WED-SQL expressas em EBNF.	19
3.1	Exemplos de regras léxicas que definem tokens usados pela linguagem WED-SQL.	29
3.2	Exemplos de regras léxicas que definem tokens usados pela linguagem WED-SQL.	30
3.3	Regras sintáticas que representam os comandos da linguagem WED-SQL.	32
3.4	Regras sintáticas da linguagem WED-SQL.	33

3.5	Demais regras sintáticas da linguagem WED-SQL.	34
3.6	Validação semântica de criação de WED-trigger	36
3.7	Tradução do comando de criação de WED-trigger para SQL.	37
3.8	Uso da interface listener no programa wsql para chamar os métodos de análise se- mântica e de tradução para SQL.	37
4.1	Comando WED-flow de criação de WED-attributes.	42
4.2	Comando WED-flow de criação de WED-conditions.	43
4.3	Comando WED-flow de criação de WED-transitions.	43
4.4	Comando WED-flow de criação de WED-triggers.	43
4.5	Comando WED-flow de criação de modelo WED-flow.	44
4.6	Algumas das regras léxicas da linguagem WED-flow.	44
4.7	Regras léxicas da linguagem WED-flow que compõem o modo léxico WED_ CONDITION.	45
4.8	Regras sintáticas da linguagem WED-flow que derivam não-terminais para cada co- mando a partir do símbolo inicial da gramática.	46
4.9	Demais regras sintáticas da linguagem WED-flow.	46
4.10	Exemplo de verificação semântica do compilador de linguagem WED-flow.	48
4.11	Exemplo de modelo descrito na linguagem WED-flow.	49
4.12	Tradução para linguagem WED-SQL do modelo de exemplo.	50
5.1	Modelagem WED-flow dos processos de entrevista e coleta do SISAUT.	56
5.2	Modelagem WED-flow do processo de processamento de materiais do SISAUT.	58
5.3	Especificação WED-SQL dos processos de entrevista e coleta do SISAUT.	59
5.4	Especificação WED-SQL do processo de processamento de materiais do SISAUT.	60
5.5	Código Python da implementação da WED-transition “verificar_material” como um serviço WED-worker.	65
5.6	Comandos WED-SQL de instanciação do modelo de coleta e entrevista.	67
5.7	Comandos WED-SQL de instanciação do modelo de processamento de materiais.	69

Capítulo 1

Introdução

Com os avanços tecnológicos na área de tecnologia da informação, organizações estão sujeitas à intensificação da necessidade de reagir rapidamente a mudanças nas regras de negócio e incorporação de novas regras para manter seus processos de negócio atualizados. Essa demanda por sistemas de informação flexíveis, que oferecem facilidade na aplicação de mudanças no processo de negócio, popularizou os chamados Sistemas de Informação Cientes de Processos (PAIS, do inglês *Process-Aware Information Systems*), definidos como sistemas de software que gerenciam e executam processos operacionais envolvendo pessoas, aplicações e fontes de informação com base em modelos de processos [DvdAtH05].

As abordagens formais como redes de Petri [Mur89] e álgebra de processos [BPS01] realizam com sucesso o gerenciamento de processos de negócio (BPM - *Business Process Management*) oferecendo verificação formal do modelo. Essas abordagens são limitadas pois, embora funcionem bem para processos estáticos especificados *a priori*, não apoiam mudanças incrementais nos processos. Outras abordagens propõem o uso de linguagens como WSBPEL (*Web Services Business Process Execution Language*) [JEA⁺07], que oferece flexibilidade para apoiar alterações incrementais em processos, mas o poder de especificação da linguagem WSBPEL apresenta desafios quanto à verificação formal e checagem de modelo [FTMP10]. Para apoiar o gerenciamento de processos de negócio voláteis de forma consistente foi proposta a abordagem WED-flow.

A abordagem WED-flow (*Work, Event and Data - flow*) dá igual importância aos conceitos de fluxo de trabalho (*workflow*), eventos e fluxo de dados (*dataflow*) e permite modelar e executar processos de um workflow em um ambiente transacional de banco de dados, armazenando cada diferente estado de dados atingido durante a execução. Para a captura dos eventos, o modelo permite que sejam definidas condições na forma de predicados lógicos definidos sobre os estados de dados e quando um evento é capturado, o modelo deve definir uma transação no banco de dados a ser disparada, resultando em um novo estado de dados após sua execução. Deste modo, a abordagem WED-flow facilita o gerenciamento flexível de eventos na modelagem de um sistema volátil. Por manter o registro do fluxo de dados durante a execução de uma instância, o WED-flow também permite a recuperação de estados inconsistentes em uma execução, garantindo que toda instância seja executada de forma consistente. Como as estruturas de um WED-flow são definidas sobre dados, predicados lógicos e transações, uma modelagem WED-flow é facilmente mapeada para sua implementação no ambiente transacional.

Uma implementação da abordagem WED-flow foi desenvolvida e apresentada detalhadamente em [PRS⁺18] e já é usada para apoiar a modelagem e desenvolvimento de PAIS com dependências

entre processos [FTK⁺17]. Resumidamente, a implementação é composta por três componentes principais:

- O ambiente transacional WED-engine configurado sobre o SGBDR (Sistema Gerenciador de Banco de Dados Relacional) Postgres para a execução dos comandos de especificação de modelo e responsável pelo gerenciamento da execução das transações disparadas por instâncias de processos.
- A linguagem WED-SQL para especificação das estruturas do modelo e para controle e monitoramento da execução de instâncias do modelo.
- Um interpretador da linguagem WED-SQL que traduz os comandos WED-SQL para SQL e os executa no ambiente WED-engine.

Como o modelo WED-flow é baseado em conjuntos, o artigo [FBTP12] propõe uma linguagem de modelagem para a abordagem WED-flow (também referenciada pela forma reduzida “linguagem WED-flow” neste trabalho) baseada na notação de conjuntos. Em [PRS⁺18], foi criada uma linguagem intermediária baseada em SQL, chamada WED-SQL. Essa linguagem define as estruturas em um nível mais baixo de abstração e permite o controle e monitoramento da execução de instâncias do modelo.

Os comandos escritos na linguagem WED-SQL são traduzidos para comandos SQL, que são executados no ambiente transacional. Antes deste trabalho, o código que realiza a tradução dos comandos da linguagem WED-SQL se encontrava em uma versão temporária. O interpretador de linguagem WED-SQL inicialmente foi escrito para reconhecer os comandos fazendo comparações com expressões regulares, para então gerar código SQL equivalente.

Este trabalho introduz dois compiladores à implementação da abordagem WED-flow. Um dos compiladores recebe entrada escrita em linguagem de modelagem WED-flow e gera saída em linguagem WED-SQL. O outro compilador introduzido recebe entrada escrita em linguagem WED-SQL e gera saída em linguagem SQL.

A seguir é exibido um trecho da classe inicialmente usada para realizar a interpretação dos comandos WED-SQL:

```

1 class WedParser(object):
2
3     def __init__(self):
4         self.wed_sql = dict()
5         self.wed_sql['create_flow'] = re.compile(r"^\s*create\s+(wed-)?flow\s+([a-z0-9_]+)$",re.I)
6         self.wed_sql['remove_flow'] = re.compile(r"^\s*remove\s+(wed-)?flow\s+([a-z0-9_]+)$",re.I)
7         self.wed_sql['create_attr'] = re.compile(r"^\s*create\s+(wed-)?(attribute|attr)\s+([a-z0-9_]+)(\s+default\s+value\s+([a-z0-9_]{1,100})?)?$",re.I)
8         self.wed_sql['create_trans'] = re.compile(r"^\s*create\s+(wed-)?(trans|transition)\s+([a-z0-9_]+\s+stop\s+on\s+timeout\s+(.+?)(\s+condition\s+([a-z0-9_]+)?)?)?$",re.I)
9         ...
10
11     def parse_stmt(self,stmt):
12         self.actl = []
13         self.argl = []
14         for self.action in self.wed_sql.keys():
15             self.result = self.wed_sql[self.action].match(stmt)

```

```

16         if self.result:
17             self.actl.append(self.action)
18             if self.action == 'create_flow':
19                 self.argl = [self.result.group(2)]
20             elif self.action == 'remove_flow':
21                 self.argl = [self.result.group(2)]
22             elif self.action == 'create_attr':
23                 self.argl = [self.result.group(3), self.result.group(5)]
24             elif self.action == 'create_trans':
25                 self.argl = [self.result.group(3), self.result.group(5), self.result.group(7)]
26             ...
27         else:
28             raise SyntaxWarning("Expression without an action found !!!")
29     if not self.actl:
30         self.actl.append("SQL")
31         self.argl = [stmt]
32     if len(self.actl) > 1:
33         raise SyntaxWarning("Statement has matched more than one rule !!")
34     else:
35         return (self.actl[0], self.argl)

```

Código 1.1: Trecho de código Python da versão inicial do interpretador de comandos WED-SQL.

A classe “WedParser” era usada para reconhecimento de comandos WED-flow e captura de determinadas partes da entrada para processamento no programa wsql. A classe consistia de um construtor, que no trecho exibido escrito na linguagem python é o método nomeado “__init__”, e de um método “parse_stmt”. No construtor é definida uma expressão regular para cada comando WED-SQL. O método “parse_stmt” recebe a entrada fornecida pelo usuário na variável stmt e verifica se a entrada satisfaz alguma das expressões regulares armazenadas pelo construtor. Caso uma e apenas uma expressão regular tenha sido satisfeita pela entrada, o método retorna um identificador da expressão regular e uma lista de trechos importantes capturados da entrada, que variam de acordo com o tipo de comando. Caso contrário, o método retorna uma mensagem de erro.

Ao receber uma entrada, o tradutor precisa identificar a entrada como um comando WED-SQL específico. Em seguida, o tradutor deve segmentar a entrada e atribuir valor sintático aos segmentos para que possam ser usados na geração de comandos SQL. Essas tarefas são complexas e sua implementação está sujeita a erros de naturezas diversas.

Essa versão do interpretador não segue a arquitetura clássica de um compilador, que deve ser organizada conceitualmente em etapas de análise léxica, sintática, semântica e etapa de tradução para a linguagem alvo. Essa organização isola as etapas e facilita não só a identificação de problemas no código do compilador como também o tratamento de erros na entrada, já que o programa sabe em qual etapa do processo o erro ocorreu.

Além disso, nessa versão do interpretador, as regras gramaticais da linguagem WED-SQL estão implícitas nas expressões regulares, o que dificulta a leitura humana do código do interpretador. Conforme são adicionados ou alterados comandos da linguagem, se torna mais difícil gerenciar conflitos nas expressões regulares usadas no reconhecimento dos padrões da entrada.

Portanto, o código do interpretador era de difícil manutenção, tornando custosa a adição de novos comandos WED-SQL caso necessário, a implementação de possíveis alterações nas construções já

existentes da linguagem WED-SQL e de correções no interpretador.

O interpretador também não realizava verificações de consistência referentes às dependências entre estruturas do WED-flow. As verificações de consistência são tratadas no ambiente transacional WED-engine. No entanto, caso o código resultante precise ser usado fora da WED-engine, por exemplo, em outro SGBDR, então as verificações de consistência não seriam realizadas. Nesse caso, o interpretador permitiria que o usuário configure uma modelagem sintaticamente válida, porém semanticamente inconsistente.

Este trabalho propõe escrever um compilador para traduzir a linguagem WED-SQL para SQL. O compilador será construído baseado em uma descrição formal da linguagem WED-SQL e será organizado em etapas de compilação conceitualmente isoladas, o que resulta em um código melhor organizado e de fácil manutenção.

Com o sucesso da implementação da abordagem, trabalhos futuros estão sendo estudados para evoluir a implementação da abordagem WED-flow. Para que avanços na implementação sejam viáveis, o código que realiza a tradução da linguagem WED-SQL deve ser flexível para incorporar as extensões necessárias.

A linguagem WED-flow oferece, a um usuário com interesse apenas na modelagem de processos, comandos de modelagem usando a mesma notação formal de conjuntos apresentada no trabalho [FBTP12]. Enquanto a linguagem WED-SQL oferece comandos de especificação dos elementos do processo, de instanciação do processo e de monitoramento de instâncias e estruturas do processo. Por permitir descrever o modelo em nível mais baixo de abstração, os comandos de modelagem da linguagem WED-SQL oferecem a especificação de detalhes de implementação. Já os comandos de modelagem da linguagem WED-flow definem os elementos do modelo de forma enxuta, sem especificação de detalhes de implementação, o que facilita a representação do modelo.

No contexto da abordagem WED-flow, um projetista do modelo é um usuário da abordagem WED-flow com conhecimento de regras de negócio que atua definindo a modelagem de processos. Já um administrador do modelo é um usuário com conhecimento da parte técnica da abordagem, atuando na definição de detalhes de implementação do modelo e no monitoramento do modelo e das instâncias.

Sendo assim, é importante que um projetista possa descrever um modelo com uma sintaxe mais enxuta, se abstraindo de detalhes de implementação da abordagem. Então para um projetista, a linguagem WED-flow seria mais apropriada. Já para um administrador, é importante poder especificar detalhes de implementação do modelo e ter acesso a comandos de monitoramento. Portanto, para um administrador, a linguagem WED-SQL é mais apropriada.

Este trabalho apresenta um estudo de caso para mostrar o funcionamento dos dois compiladores na implementação do WED-flow. O estudo de caso é realizado sobre o sistema SISAUT [FTK⁺17], propondo uma modelagem de seus processos na linguagem de especificação WED-flow. Essa modelagem será traduzida para código WED-SQL, e posteriormente para código SQL para que seja executada no ambiente transacional WED-engine. Com o modelo criado, podem ser demonstrados os comandos de instanciação e monitoramento de execução de instâncias do modelo oferecidos pela linguagem WED-SQL.

1.1 Objetivos

Os principais objetivos deste trabalho são:

- Permitir ao projetista de um sistema WED-flow descrever o modelo usando a linguagem de modelagem WED-flow, oferecendo um nível maior de abstração do que o oferecido pela linguagem WED-SQL.
- Implementar compiladores robustos para as linguagens WED-SQL e WED-flow, fazendo uso de uma arquitetura clássica de compiladores e de uma descrição formal das regras léxicas e sintáticas de cada linguagem, resultando em uma melhor organização do código do compilador e na redução da ocorrência de erros de compilação.
- Prover facilidade de manutenção dos compiladores introduzindo a automação da geração de código em algumas das etapas do processo de compilação a partir de regras léxicas e sintáticas formalizadas, o que facilita que o compilador acompanhe possíveis mudanças na linguagem.
- Prover um arcabouço tecnológico para viabilizar a contínua evolução das linguagens da abordagem WED-flow.

1.2 Contribuições

Este trabalho contribui com duas ferramentas úteis para a atual implementação da abordagem WED-flow:

- Adição do compilador de comandos da linguagem de modelagem WED-flow, escritos pelo projetista do modelo para especificá-lo em termos de estruturas da abordagem WED-flow descritas em notação de conjuntos. O compilador traduz esses comandos de especificação de modelo da linguagem WED-flow para a linguagem WED-SQL de nível mais baixo.
- Adição do compilador de comandos da linguagem WED-SQL, escritos pelo administrador do sistema para controlar e monitorar a execução de instâncias do modelo ou pelo projetista para especificar o modelo em um nível mais baixo do que a linguagem de modelagem WED-flow. O compilador traduz os comandos da linguagem WED-SQL para a linguagem SQL, os comandos na linguagem SQL podem ser executados sobre um ambiente transacional construído sobre um SGBDR, como o ambiente WED-engine.

Outra contribuição importante é que os compiladores poderão ser facilmente adaptados para serem usados fora da implementação atual da abordagem WED-flow. Ou seja, os compiladores poderão ser úteis para outros programas escritos em Python que necessitem usá-los ou até mesmo em outras implementações do WED-flow sendo executadas sobre outros SGBDRs.

Caso seja necessário futuramente implementar um compilador das linguagens WED-SQL e WED-flow em outra linguagem de programação que não seja Python, as gramáticas desenvolvidas neste trabalho podem ser aproveitadas para facilitar a implementação dos compiladores, desde que a linguagem de implementação do compilador seja umas das oferecidas pela ferramenta ANTLR (*ANother Tool for Language Recognition*), usada neste trabalho para a geração automática de analisadores léxicos e sintáticos.

1.3 Organização do Trabalho

No Capítulo 2, são apresentados os fundamentos que dão apoio ao trabalho, mais detalhes sobre a abordagem WED-flow e os componentes de sua implementação como a linguagem WED-SQL, também são apresentados em detalhes a teoria de linguagens, a arquitetura clássica de compiladores, algoritmos de análise sintática e ferramentas geradoras de analisadores sintáticos e léxicos.

Em seguida, os capítulos 3 e 4 apresentam respectivamente os compiladores das linguagens WED-SQL e WED-flow, mostrando detalhadamente como os comandos das linguagens são formalmente explicitados e como os compiladores são criados a partir das descrições formais das linguagens, apresentando também detalhes de cada etapa da compilação.

Um estudo de caso é apresentado no Capítulo 5, no qual é introduzido o sistema SISAUT, cujos processos são modelados usando as linguagens de especificação WED-flow e WED-SQL. Também é demonstrado o funcionamento dos comandos WED-SQL de controle e monitoramento de execução de instâncias do modelo.

No Capítulo 6 são retomadas as contribuições do trabalho, discutidos os resultados obtidos e são apresentados os próximos passos.

Capítulo 2

Fundamentos

Este capítulo apresenta os fundamentos que apoiam o trabalho. São exibidos detalhes sobre a abordagem WED-flow e sua implementação, a linguagem WED-SQL, a WED-engine e o programa `wsql`. Além disso, são apresentados conceitos de linguagens formais e de compilação, são exibidos alguns métodos de análise sintática e são discutidas ferramentas de geração automática de analisadores léxicos e sintáticos.

2.1 WED-flow

WED-flow é uma abordagem para modelar e implementar PAIS dinâmicos que armazena o estado dos dados do sistema durante cada tarefa de um *workflow* para facilitar o gerenciamento de eventos de forma flexível [FWMP10] e garantir a consistência do sistema durante a execução transacional das tarefas do *workflow* [FBTP12].

A abordagem WED-flow combina as perspectivas de abordagens dirigidas por dados, orientadas a processos e baseadas em eventos. Sob a perspectiva de orientação a processos, as regras de negócio de um *workflow* podem ser representadas por transições que operam sobre dados na forma de atributos especificados na modelagem WED-flow. Do ponto de vista de eventos, condições definidas sobre um conjunto de atributos do estado de dados realizam a captura de eventos, que são representados por transações que realizam atividades que operam sobre o estado de dados. Quanto à perspectiva de fluxo de dados, o WED-flow permite a definição de estados de dados a partir dos atributos especificados na modelagem e uma instância de execução do sistema pode sofrer alteração em seu estado de dados por meio da execução de transações no banco de dados que alteram os valores dos atributos.

A modelagem de um sistema WED-flow é composta por estruturas que devem ser configuradas pelo projetista para definir os atributos que compõem um estado de dados, as condições de captura dos eventos, as condições de disparo de tarefas do *workflow* e seus efeitos nos dados. As seguintes estruturas compõem uma modelagem WED-flow:

- WED-attributes: Representam os atributos do ambiente relacional que têm importância na especificação do *workflow*. O conjunto dos WED-attributes é denotado formalmente por $\mathcal{A} = \{a_1, \dots, a_n\}$, onde todo elemento a_i , $1 \leq i \leq n$ é um atributo de interesse para a modelagem de processos.

- **WED-state:** É uma configuração específica de valores para todos os WED-attributes definidos. É expressa pela tupla $\mathcal{S} = \langle v_1, \dots, v_n \rangle$, onde todo elemento v_i , $1 \leq i \leq n$ é o valor do WED-attribute a_i . O conjunto de WED-states é denotado por $\mathcal{S} = \{ \langle v_1, \dots, v_n \rangle \mid \forall i \in [1; n], v_i \in \text{dom}(a_i) \}$, onde $\text{dom}(a_i)$ representa o domínio do atributo a_i . Um WED-state pode ser considerado: (1) consistente, caso respeite todas as restrições de integridade definidas sobre atributos; (2) transação-consistente, caso exista uma transação atualmente em execução ou uma transação que é disparada por alguma condição que o WED-state em questão satisfaz; ou (3) inconsistente, caso não seja nem consistente e nem transação-consistente.
- **WED-condition:** Uma WED-condition é uma expressão SQL de valor booleano escrita em função dos WED-attributes que representa uma condição de captura de eventos. O conjunto de condições é denotado por $\mathcal{C}$ e é dito que um WED-state satisfaz uma WED-condition caso sua configuração de valores de atributos faça com que a expressão da WED-condition seja avaliada como verdadeira. Uma WED-condition não precisa conter todos os WED-attributes em sua expressão. É possível que uma WED-condition seja satisfeita por mais de um WED-state e também que um WED-state satisfaça mais de uma WED-condition. Neste e em outros trabalhos relacionados ao WED-flow, é comum usarmos o termo “predicado” para nos referir à expressão SQL de uma WED-condition, já que esse tipo de expressão é equivalente a um predicado lógico de primeira ordem aplicado à teoria de cálculo relacional, apresentada no trabalho [Cod70], que é um dos fundamentos sobre os quais a linguagem SQL foi construída.
- **WED-transitions:** São funções que alteram os WED-attributes levando a instância de um WED-state a outro, ou seja, são da forma $t : \mathcal{S} \rightarrow \mathcal{S}$. O conjunto das WED-transitions é denotado por $\mathcal{T}$.
- **WED-triggers:** Estruturas que associam uma WED-condition a uma WED-transition de forma que quando a WED-condition for satisfeita, a WED-transition associada será executada. Uma WED-trigger é denotada pela tupla $g = \langle c, t \rangle$, em que $c \in \mathcal{C}, t \in \mathcal{T}$ e o conjunto de WED-triggers é representado por $\mathcal{G}$.
- **WED-flow:** O termo WED-flow também é usado para denominar um determinado modelo que segue a abordagem WED-flow. Assim, um WED-flow é formalmente denotado pela tupla $\langle \mathcal{G}', c_i, c_f \rangle$, em que $\mathcal{G}' \subseteq \mathcal{G}$ e $c_i, c_f \in \mathcal{C}$, sendo c_i chamada de condição inicial, c_f chamada de condição final. Um WED-state que satisfaz c_i é chamado de WED-state inicial e um WED-state que satisfaz c_f é chamado WED-state final. Cada instância desse WED-flow parte de um WED-state transação-consistente inicial a um WED-state consistente final pela execução das WED-transitions disparadas de acordo com as associações nas WED-triggers de $\mathcal{G}'$ quando as condições das WED-triggers são satisfeitas.

O fato de que as WED-transitions de um WED-flow são executadas na forma de transações no banco de dados permite a execução paralela de tarefas sem quebrar a consistência dos dados. A execução paralela de transições ocorre quando o sistema atinge um WED-state s_i para o qual existem mais de uma WED-trigger associadas às WED-conditions que s_i satisfaz. O trabalho [FTK⁺17] realiza um estudo de caso de uma dependência entre processos que executam paralelamente em um modelo WED-flow.

Um sistema WED-flow registra o histórico de WED-states alcançados durante a execução de uma instância. Caso ocorra um evento não esperado e uma instância esteja em um WED-state inconsistente, o armazenamento do histórico dos WED-states e as estruturas do modelo WED-flow facilitam o tratamento da exceção usando algoritmos de recuperação da instância. O trabalho [FTMP10] mostra como os elementos da abordagem WED-flow oferecem vantagens para facilitar o tratamento de exceções. Um gerenciador de recuperação de exceções de processos de negócio modelados em WED-flow é apresentado pelo trabalho [SRBF12]. O trabalho [SLPF18] apresenta um método de recuperação ciente de custo na abordagem WED-flow que é capaz de encontrar e seguir caminhos de recuperação que reduzem o custo do tratamento de exceções.

Uma primeira implementação da abordagem WED-flow é proposta pelo trabalho [GRBPF12]. Entretanto, a implementação da abordagem WED-flow usada neste trabalho é composta por um ambiente transacional apresentado em [PSL17] e uma linguagem de modelagem e gerenciamento apresentada em [PRS⁺18].

2.1.1 Exemplo de modelo WED-flow

Para exemplificar a abordagem WED-flow, suponhamos a modelagem WED-flow de um sistema de gerenciamento de locação de carros. Esse sistema se baseia em um processo composto pelas seguintes etapas:

- Requisição de veículo: O cliente especifica suas preferências de locação, informando sua cidade, o tipo de carro e os dias de retirada e devolução.
- Validação de login do cliente: Etapa necessária para validar a identificação do cliente que solicitou o veículo.
- Reserva do veículo: Caso seja possível, o sistema reserva um veículo disponível dentro das preferências do cliente.
- Validação de método de pagamento: O cliente informa o método de pagamento, podendo haver uma confirmação do método informado.
- Retirada do veículo: Registra quando o cliente faz a retirada do veículo reservado.
- Devolução do veículo: Registra quando o cliente devolve o veículo reservado.
- Emissão da conta: Emissão da conta para que o cliente possa pagar pela locação, finalizando o processo.

A modelagem WED-flow do processo apresentado se inicia com a definição dos WED-attributes. Cada instância desse processo irá se referir à locação de um veículo, solicitado por um pedido feito por um cliente. Portanto, devem ser criados atributos referentes ao cliente (identificação, status de validação do login), ao pedido (identificação, tipo de veículo, data de retirada, data de retorno, status do pedido) e ao veículo (identificação, tipo do veículo, status do veículo), como a seguir:

```
1  $\mathcal{A} = \{id\_cliente, status\_cliente, id\_pedido, tipo\_veiculo\_pedido, data\_retirada, data\_devolucao,$ 
2  $status\_pedido, id\_veiculo, tipo\_veiculo, status\_veiculo\}$ 
```

Código 2.1: Definição dos WED-attributes do exemplo de locação de carros usando a abordagem WED-flow.

Devem ser definidas em seguida as WED-conditions que farão as capturas dos eventos do processo (requisição de veículo, validação de identificação do cliente, reserva de veículo, validação de método de pagamento, retirada de veículo, devolução de veículo e pagamento de conta):

```

1  $\mathcal{C} = \{requisitou\_veiculo, validou\_cliente, reservou\_veiculo, validou\_metodo\_pagamento,$ 
2  $retirou\_veiculo, devolveu\_veiculo, pagou\_conta\}$ 
3
4  $requisitou\_veiculo : (status\_pedido = "Requerido" \text{ and } id\_pedido \neq null \text{ and}$ 
5  $status\_cliente = "Nao Validado" \text{ and } id\_cliente \neq null)$ 
6
7  $validou\_cliente : (status\_pedido = "Validado" \text{ and } status\_cliente = "Validado" \text{ and } id\_veiculo = null)$ 
8
9  $reservou\_veiculo : (status\_pedido = "Validado" \text{ and } id\_veiculo \neq null \text{ and } status\_veiculo = "Reservado")$ 
10
11  $validou\_metodo\_pagamento : (status\_pedido = "CC Validado" \text{ and } status\_veiculo = "Reservado")$ 
12
13  $retirou\_veiculo : (status\_veiculo = "Retirado")$ 
14
15  $devolveu\_veiculo : (status\_pedido = "Validado" \text{ and } status\_veiculo = "Devolvido")$ 
16
17  $pagou\_conta : (status\_pedido = "Finalizado")$ 

```

Código 2.2: Definição das WED-conditions do exemplo de locação de carros usando a abordagem WED-flow.

A seguir são declaradas as WED-transitions, que alteram o estado de dados da instância de acordo com o andamento do processo:

```

1  $\mathcal{T} = \{validar\_login\_cliente, reservar\_veiculo, validar\_metodo\_pagamento, retirar\_veiculo,$ 
2  $devolver\_veiculo, emitir\_conta\}$ 

```

Código 2.3: Definição das WED-transitions do exemplo de locação de carros usando a abordagem WED-flow.

As WED-triggers do modelo irão associar WED-conditions que capturam um evento às WED-transitions que devem ser disparadas quando o evento é reconhecido:

```

1  $\mathcal{G} = \{trigger1, trigger2, trigger3, trigger4, trigger5, trigger6\}$ 
2
3  $trigger1 = \langle requisitou\_veiculo, validar\_login\_cliente \rangle$ 
4  $trigger2 = \langle validou\_cliente, reservar\_veiculo \rangle$ 
5  $trigger3 = \langle reservou\_veiculo, validar\_metodo\_pagamento \rangle$ 
6  $trigger4 = \langle validou\_metodo\_pagamento, retirar\_veiculo \rangle$ 
7  $trigger5 = \langle retirou\_veiculo, devolver\_veiculo \rangle$ 
8  $trigger6 = \langle devolveu\_veiculo, emitir\_conta \rangle$ 

```

Código 2.4: Definição das WED-triggers do exemplo de locação de carros usando a abordagem WED-flow.

Por último, é necessário definir o modelo WED-flow, indicando o conjunto de WED-triggers e as condições inicial e final:

```

1  $wf\_locacao\_carro = \langle \mathcal{G}, requisitou\_veiculo, pagou\_conta \rangle$ 

```

Código 2.5: Definição do modelo WED-flow do exemplo de locação de carros usando a abordagem WED-flow.

2.1.2 Implementação do WED-flow

O trabalho [Pad18] apresenta uma implementação da abordagem WED-flow que se divide em três partes: a linguagem WED-SQL para especificação e monitoramento das estruturas do WED-flow; o ambiente transacional WED-Engine que gerencia as estruturas de um modelo WED-flow no banco de dados e executa transações de acordo com o modelo WED-flow configurado; e o programa `wsq` que interpreta os comandos WED-SQL e executa os respectivos comandos SQL no ambiente WED-Engine.

A linguagem WED-SQL é uma linguagem intermediária entre um modelo WED-flow e a linguagem SQL. Na implementação são criadas relações em um banco de dados relacional para representar as estruturas do modelo WED-flow. A linguagem WED-SQL permite a especificação de estruturas de modelagem de um sistema WED-flow abstraindo construções em SQL. A linguagem simplifica a especificação para o usuário em comparação com a linguagem SQL.

Os seguintes comandos de especificação de estruturas do modelo são oferecidos pela linguagem WED-SQL:

- Criação de WED-flow: O comando de criação de um modelo WED-flow recebe como parâmetro obrigatório o nome do modelo e cria um banco de dados de mesmo nome com as relações necessárias para o armazenamento das demais estruturas, instâncias e histórico de execução das instâncias dentro do escopo desse modelo WED-flow.

```
1 CREATE FLOW <id>;
```

Código 2.6: *Comando WED-SQL de criação de modelo WED-flow.*

- Remoção de WED-flow: O comando de remoção de um modelo WED-flow recebe como parâmetro obrigatório o nome de um modelo WED-flow existente e então remove o modelo WED-flow e apaga o banco de dados criado para o mesmo durante sua criação.

```
1 REMOVE FLOW <id>;
```

Código 2.7: *Comando WED-SQL de remoção de modelo WED-flow.*

- Listagem de WED-flows: Esse comando lista todos os modelos WED-flow que já foram criados no servidor de banco de dados configurado para a aplicação.

```
1 \lw;
```

Código 2.8: *Comando WED-SQL de listagem de modelos WED-flow.*

- Criação de WED-attribute: O comando de criação de WED-attribute recebe como parâmetro obrigatório o nome do atributo a ser criado e, como parâmetro opcional, um valor padrão que será usado caso uma instância seja criada sem especificar um valor para esse atributo. Esse comando adiciona uma linha da relação de atributos no banco de dados referente ao WED-flow em uso.

```
1 CREATE WED-ATTRIBUTE <id> [DEFAULT VALUE <valor>];
```

Código 2.9: *Comando WED-SQL de criação de WED-attribute.*

- Criação de WED-condition: O comando de criação de WED-condition recebe como parâmetros obrigatórios o nome da condição e o predicado expresso em linguagem SQL. Para manter a independência da implementação quanto aos Sistemas Gerenciadores de Banco de Dados Relacionais (SGBDR) e evitar trabalho redundante, a validação da expressão em SQL fica a cargo do interpretador do próprio SGBDR, sujeita às suas particularidades. Esse comando adiciona uma linha da relação de condições no banco de dados referente ao WED-flow em uso.

```
1 CREATE WED-CONDITION <id> AS <predicado>;
```

Código 2.10: *Comando WED-SQL de criação de WED-condition.*

- Criação de WED-transition: Esse comando recebe como parâmetro obrigatório o nome da transição a ser criada e opcionalmente pode receber uma expressão temporal válida no SGBDR que representa o tempo máximo de execução da transação e também pode receber o nome de uma WED-condition que será usada como condição de parada da transação executada. O que significa que quando a condição de parada for satisfeita a execução da transação será interrompida. Quando o estado de dados de uma instância é alterado, a WED-Engine verifica se alguma condição de parada foi satisfeita e, caso tenha sido, interrompe a execução das transações referentes às WED-transitions com tal condição de parada. Caso a verificação da condição de parada fosse feita pelas próprias transações, possivelmente alguma alteração no estado de dados da instância poderia não ser detectada pelas transações dependendo da ordem de operações de escrita e leitura sobre o estado de dados. Portanto, a configuração de condição de parada foi implementada para garantir que toda mudança de estado da instância seja verificada, permitindo a interrupção confiável da execução de WED-transitions. Embora esse comando declare quais são as WED-transitions do modelo, a implementação de suas respectivas transações não é especificada nesse comando. As transações devem ser implementadas como serviços WED-workers ou como procedimentos armazenados. Opcionalmente, é possível informar que a transação correspondente à transição criada deve ser implementada como um procedimento armazenado, adicionando o trecho "AS SP". Caso não seja informado, o WED-engine assume que a transação está implementada como um WED-worker.

```
1 CREATE WED-TRANSITION <id> [STOP ON TIMEOUT <time> [CONDITION <id_condition>] | STOP ON CONDITION <id_condition> [TIMEOUT <time>]] [AS SP];
```

Código 2.11: *Comando WED-SQL de criação de WED-transition.*

- Criação de WED-trigger: O comando de criação de WED-trigger recebe como parâmetros obrigatórios o nome da WED-trigger a ser criada, os nomes da condição e da transição que serão associadas. Esse comando adiciona uma linha da relação de triggers no banco de dados referente ao WED-flow em uso.

```
1 CREATE WED-TRIGGER <id> AS (<id_condition>, <id_transition>);
```

Código 2.12: *Comando WED-SQL de criação de WED-triggers.*

- Definição e exibição de estado final: O primeiro dos comandos abaixo recebe um predicado expresso em SQL que representa a condição final do WED-flow. O segundo dos comandos abaixo exibe a condição final do WED-flow. Um modelo WED-flow pode ter mais de um

WED-state final, porém não deve haver nenhuma WED-trigger associando sua condição final a uma transição.

```
1 SET FINAL WED-STATE AS <predicado>;
2 SHOW FINAL WED-STATE;
```

Código 2.13: *Comandos WED-SQL de configuração e exibição de condição de WED-state final.*

- Ativação e desativação de WED-trigger: Esses comandos oferecem ao projetista um maior controle sobre as WED-triggers, permitindo desativar ou ativar uma WED-trigger cujo nome é fornecido como parâmetro obrigatório. Ao contrário de uma WED-trigger ativa, uma WED-trigger desativada não irá disparar a execução de sua transição quando sua condição for satisfeita.

```
1 ENABLE WED-TRIGGER <id>;
2 DISABLE WED-TRIGGER <id>;
```

Código 2.14: *Comandos WED-SQL de ativação e desativação de WED-trigger respectivamente.*

A linguagem WED-SQL também oferece os seguintes comandos para a instanciação do modelo:

- Criação de instância: Diferente dos outros comandos, o comando de criação de instância não define uma estrutura no modelo, mas dá início à execução de uma instância. Esse comando recebe uma expressão para indicar o estado inicial da instância, que pode assumir o valor “DEFAULT VALUES”, indicando que o estado inicial é composto dos valores pré-definidos de cada WED-attributes. Também pode ser fornecida uma expressão composta pelos identificadores de cada WED-attribute entre parênteses seguidos dos respectivos valores desejados para cada WED-attribute no estado inicial entre parênteses.

```
1 CREATE INSTANCE AS DEFAULT VALUES;
2 CREATE INSTANCE AS (<id_atributo_1>, <id_atributo_2>, ... ) VALUES (<
    valor_atributo_1>, <valor_atributo_2>, ... );
```

Código 2.15: *Comando WED-SQL de criação de instância.*

- Cancelamento e restabelecimento de instâncias: O primeiro dos comandos abaixo recebe o id de uma instância como parâmetro obrigatório e interrompe manualmente a execução da instância independentemente do seu estado, as transações referentes à instância cancelada que estiverem em execução são abortadas. Para uma instância cancelada não é realizada a verificação do estado, ou seja, nenhuma WED-trigger pode ser acionada. O segundo dos comandos abaixo recebe como parâmetro obrigatório o id de uma instância previamente interrompida manualmente e restabelece sua execução, permitindo que a WED-engine realize a verificação de ser estado para acionar as WED-triggers.

```
1 TERMINATE INSTANCE <id>;
2 REINSTATE INSTANCE <id>;
```

Código 2.16: *Comandos WED-SQL de cancelamento e restabelecimento de instâncias.*

A seguir são exibidos comandos de monitoramento do modelo e de suas instâncias oferecidos por WED-SQL:

- Listagem de WED-attributes, WED-conditions, WED-transitions e WED-triggers: Os comandos abaixo listam os WED-attributes, as WED-conditions, as WED-transitions e as WED-triggers de um modelo WED-flow respectivamente.

```
1  LIST WED-ATTRIBUTES;
2  LIST WED-CONDITIONS;
3  LIST WED-TRANSITIONS;
4  LIST WED-TRIGGERS;
```

Código 2.17: *Comandos WED-SQL de listagem de WED-attributes, WED-conditions, WED-transitions e WED-triggers respectivamente.*

- Listagem de WED-transitions disparadas: Esse comando recebe como parâmetro obrigatório o id de uma instância do modelo WED-flow e lista todas as WED-transitions em execução que foram disparadas pela instância indicada.

```
1  LIST FIRED WED-TRANSITIONS FOR <id>;
```

Código 2.18: *Comando WED-SQL de listagem de WED-transitions disparadas.*

- Listagem de instâncias: Os comandos abaixo listam respectivamente todas as instâncias, as instâncias finalizadas e as instâncias que ainda estão em execução de um modelo WED-flow.

```
1  LIST INSTANCES;
2  LIST FINAL INSTANCES;
3  LIST TRANSACTIONING INSTANCES;
```

Código 2.19: *Comandos WED-SQL de listagem de instâncias.*

- Exibição de histórico: Esse comando recebe como parâmetro obrigatório o id de uma instância e retorna o histórico de execução dessa instância exibindo as transições disparadas durante a execução da instância e, para cada transição disparada, o momento do disparo e o WED-state que disparou a transição.

```
1  GET WED-TRACE FOR <id>;
```

Código 2.20: *Comando WED-SQL de exibição de histórico.*

A linguagem WED-SQL também oferece ao administrador os seguintes comandos de controle do modelo:

- Conexão a um WED-flow: Os demais comandos da linguagem WED-SQL só fazem sentido quando aplicados sobre um modelo WED-flow já criado. O comando de conexão a um WED-flow estabelece que os comandos seguintes devem ser executados no contexto do modelo cujo nome é indicado como parâmetro obrigatório.

```
1  \cw <id>;
```

Código 2.21: *Comando WED-SQL de conexão a um modelo WED-flow.*

- Início, confirmação e cancelamento de uma transação: É necessário definir o escopo de uma transação para que os comandos SQL resultantes da compilação dos comandos WED-SQL dentro do escopo definido sejam aplicados no banco de dados como uma transação. O comando “begin” indica o início do escopo da transação. O comando “commit” indica o final de uma transação confirmada, que deve ser aplicada no banco de dados. O comando “abort” indica o final de uma transação cancelada, que não deve ser aplicada no banco de dados.

```

1 BEGIN;
2 COMMIT;
3 ABORT;

```

Código 2.22: Comandos WED-SQL de início, confirmação e cancelamento de uma transação.

Podemos escrever o modelo do exemplo de locação de carros exibido na Seção 2.1.1 usando os comandos da linguagem WED-SQL:

```

1 CREATE FLOW wf_locacao_carro;
2 \cw wf_locacao_carro;
3
4 CREATE WED-ATTRIBUTE id_cliente;
5 CREATE WED-ATTRIBUTE status_cliente DEFAULT VALUE "Nao Validado";
6 CREATE WED-ATTRIBUTE id_pedido;
7 CREATE WED-ATTRIBUTE tipo_veiculo_pedido;
8 CREATE WED-ATTRIBUTE data_retirada;
9 CREATE WED-ATTRIBUTE data_devolucao;
10 CREATE WED-ATTRIBUTE status_pedido DEFAULT VALUE "Requerido";
11 CREATE WED-ATTRIBUTE id_veiculo;
12 CREATE WED-ATTRIBUTE tipo_veiculo;
13 CREATE WED-ATTRIBUTE status_veiculo;
14
15 CREATE WED-CONDITION requisitou_veiculo AS (status_pedido = "Requerido" and id_pedido <>
    null and status_cliente = "Nao Validado" and id_cliente <> null);
16 CREATE WED-CONDITION validou_cliente AS (status_pedido = "Validado" and status_cliente = "
    Validado" and id_veiculo = null);
17 CREATE WED-CONDITION reservou_veiculo AS (status_pedido = "Validado" and id_veiculo <> null
    and status_veiculo = "Reservado");
18 CREATE WED-CONDITION validou_metodo_pagamento AS (status_pedido = "CC Validado" and
    status_veiculo = "Reservado");
19 CREATE WED-CONDITION retirou_veiculo AS (status_veiculo = "Retirado");
20 CREATE WED-CONDITION devolveu_veiculo AS (status_pedido = "Validado" and status_veiculo = "
    Devolvido");
21
22 CREATE WED-TRANSITION validar_login_cliente;
23 CREATE WED-TRANSITION reservar_veiculo;
24 CREATE WED-TRANSITION validar_metodo_pagamento;
25 CREATE WED-TRANSITION retirar_veiculo;
26 CREATE WED-TRANSITION devolver_veiculo;
27 CREATE WED-TRANSITION emitir_conta;
28
29 CREATE WED-TRIGGER trigger1 AS (requisitou_veiculo, validar_login_cliente);
30 CREATE WED-TRIGGER trigger2 AS (validou_cliente, reservar_veiculo);
31 CREATE WED-TRIGGER trigger3 AS (reservou_veiculo, validar_metodo_pagamento);
32 CREATE WED-TRIGGER trigger4 AS (validou_metodo_pagamento, retirar_veiculo);

```

```

33 CREATE WED-TRIGGER trigger5 AS (retirou_veiculo, devolver_veiculo);
34 CREATE WED-TRIGGER trigger6 AS (devolveu_veiculo, emitir_conta);
35
36 SET FINAL WED-STATE AS (status_pedido = "Finalizado");

```

Código 2.23: Comandos WED-SQL para especificação do processo de exemplo de locação de carros.

A implementação do ambiente WED-engine é uma extensão do SGBDR PostgreSQL que permite o armazenamento das estruturas de modelos WED-flow e que gerencia a execução de transações que podem ser expressas internamente como procedimentos armazenados no próprio SGBDR ou externamente como serviços denominados WED-workers que podem ser implementados em qualquer linguagem de programação que permita conexão com o PostgreSQL. O trabalho [Pad18] detalha como a WED-engine gerencia a execução das transações de um modelo WED-flow.

Figura 2.1: Arquitetura da abordagem WED-flow sem o uso de compiladores.

A figura 2.1 exibe a arquitetura da implementação da abordagem WED-flow anterior a este trabalho. A implementação contém códigos para a geração do ambiente transacional WED-engine, que consiste no banco de dados que contém estruturas de representação do modelo e as funções necessárias para gerenciar a execução de transações solicitadas pelas instâncias do modelo. A geração do ambiente WED-engine é acionada pelo programa wsql, quando o usuário fornece o comando WED-SQL de criação de modelo WED-flow. O programa wsql é responsável pela interação do usuário com a WED-engine, recebendo do usuário comandos nas linguagens WED-SQL e SQL para que sejam executados na WED-engine e exibindo o retorno desses comandos ao usuário quando esperado. Como a WED-engine está implementada no SGBDR Postgres, os comandos WED-SQL fornecidos ao programa wsql devem ser traduzidos para linguagem SQL. No entanto, como exibido no Capítulo 1, existe no programa wsql um trecho de código que realiza a tradução da linguagem WED-SQL para SQL, mas que não segue a arquitetura clássica de compiladores, o que gera dificuldades em sua manutenção e consequentemente na evolução da linguagem WED-SQL.

Uma das propostas deste trabalho é a criação de um compilador de linguagem WED-SQL que segue as etapas da arquitetura clássica de compiladores para que possa ser usado pelo programa wsql para traduzir comandos WED-SQL para a linguagem SQL, conforme mostra a figura 2.2. Este trabalho também contribui permitindo que o usuário informe o modelo usando a linguagem WED-flow de nível mais alto que a linguagem WED-SQL, oferecendo o programa wf2wsql que traduz comandos da linguagem WED-flow para a linguagem WED-SQL usando o compilador de linguagem WED-flow.

Figura 2.2: *Arquitetura da abordagem WED-flow com o uso de compiladores.*

2.2 Linguagens

Esta seção apresenta as definições formais de linguagens e de gramáticas livres de contexto, a representação EBNF para gramáticas e a representação de árvore sintática para derivações de cadeias de caracteres em uma gramática. A representação EBNF será usada ao longo do trabalho para descrever formalmente as linguagens WED-flow e WED-SQL. As árvores sintáticas serão necessárias para entender como são implementadas algumas etapas de compilação.

Um alfabeto é definido como um conjunto finito de símbolos. Uma cadeia sobre um alfabeto é definida como uma sequência finita de símbolos de um alfabeto. Uma linguagem é definida como qualquer conjunto de cadeias sobre um alfabeto.

Uma Gramática Livre de Contexto (GLC) é definida como uma tupla $G = (V, \Sigma, R, S)$, onde V é um alfabeto, Σ é um subconjunto de símbolos de V , $S \in V - \Sigma$ e R é um subconjunto finito de $(V - \Sigma) \times V^*$. Uma GLC (V, Σ, R, S) gera uma linguagem sobre o alfabeto Σ , cujos símbolos são chamados de terminais, já os símbolos de $V - \Sigma$ são chamados de não-terminais, ou seja, V contém todos os símbolos de Σ e mais os símbolos não-terminais. Cada elemento de R é chamado produção e representa uma função que leva de um símbolo não-terminal para uma sequência de símbolos de V . O símbolo não-terminal S é o símbolo inicial da GLC a partir do qual as produções de R são aplicadas sucessivamente sobre os símbolos não-terminais para gerar cadeias de símbolos terminais. O conjunto de todas as cadeias geradas em uma GLC é a linguagem gerada pela GLC.

Sejam $a \in V^*$ e $A \in V - \Sigma$, se existe uma produção $(A, a) \in R$, então podemos escrevê-la na forma $A \rightarrow a$. Sejam $u, v \in V^*$, então pode-se escrever $u \Rightarrow_G v$ se e somente se existem $x, y \in V^*$ e $A \in V - \Sigma$ tais que $u = xAy$, $v = xv'y$ e existe a produção $A \rightarrow v'$. Uma sequência de forma $w_0 \Rightarrow_G w_1 \Rightarrow_G \dots \Rightarrow_G w_n$ é chamada de derivação na gramática G de w_n a partir de w_0 .

Em GLCs, o termo livre de contexto é usado pelo fato das produções representando regras gramaticais poderem ser aplicadas sobre um não-terminal independentemente do contexto no qual se encontra. Uma linguagem gerada por uma GLC é chamada de linguagem livre de contexto [LP98].

2.2.1 Formalismo de Backus-Naur Estendido

Conhecido como EBNF (Extended Backus-Naur Form) [EBN], o formalismo de Backus-Naur Estendido é uma linguagem para expressar gramáticas que geram linguagens formais. Uma gramática é expressa em EBNF definindo os símbolos e as regras que a compõem. Há muitas variantes de EBNF em uso, a variante considerada neste trabalho [Par13] envolve os símbolos apresentados na tabela 2.1:

Símbolo	Significado
:	Definição
;	Terminação
	União
[]	Um dos caracteres entre os colchetes
()	Agrupamento
*	Zero ou mais vezes
+	Uma ou mais vezes
?	Zero ou uma vez
“ ”	Literal dos símbolos entre as aspas

Tabela 2.1: Símbolos da variante de EBNF usada.

2.2.2 Árvore Sintática

Sejam $G = (V, \Sigma, R, S)$ uma gramática livre de contexto, $L(G)$ a linguagem gerada por G e S o símbolo não-terminal inicial de G , então para cada cadeia $w \in L(G)$, podemos representar graficamente pelo menos uma derivação de w a partir de S usando uma árvore sintática.

Uma árvore sintática é uma estrutura composta por um ou mais nós, podendo conter arestas ligando um nó a outro. Em uma árvore sintática cada nó tem um rótulo equivalente a um símbolo de V , existe um nó no topo denominado raiz e um ou mais nós na parte de baixo denominados folhas, as folhas são rotuladas por um símbolo terminal ou pelo símbolo de cadeia vazia ε .

Definindo formalmente uma árvore sintática para a gramática G :

1. Para cada $a \in \Sigma$ existe uma árvore sintática de apenas um nó com rótulo a , que é ao mesmo tempo raiz e folha.
2. Seja $A \in V - \Sigma$, então se existir em R uma produção $A \rightarrow \varepsilon$ então existe uma árvore sintática com 2 nós ligados por uma aresta, a raiz rotulada A e a folha rotulada ε .
3. Para cada produção $A \rightarrow A_1 A_2 \dots A_n$ onde $A_1, A_2, \dots, A_n \in V$, se existir para cada símbolo A_i do lado direito da produção uma árvore sintática T_i na qual A_i é a raiz, então vai existir uma árvore sintática com raiz A que é ligada por uma aresta a cada árvore T_i .
4. Nada mais é uma árvore sintática.

Para exemplificar uma árvore sintática, consideremos uma versão simplificada das regras léxicas e sintáticas de um dos comandos (criação de WED-attributes) da linguagem WED-SQL expressas em EBNF.

```

1 start : statement_list;
2 statement_list : create_attr_stmt SEMICOLON statement_list | create_attr_stmt
  SEMICOLON;
3 create_attr_stmt: CREATE WED_ATTRIBUTE valid_name opt_def_value_expr;
4 opt_def_value_expr: /*nil*/ | DEFAULT VALUE value_expr;
5 value_expr: QUOTED_VALUE | NAT_NUMBER;
6 valid_name : (NON_SPECIAL | NAT_NUMBER | reserved_words)+;
7 reserved_words : (CREATE | WED_ATTRIBUTE | DEFAULT | VALUE);
8 SEMICOLON : ';' ;
9 CREATE : [Cc] [Rr] [Ee] [Aa] [Tt] [Ee];
10 WED_ATTRIBUTE : [Ww] [Ee] [Dd] '-' [Aa] [Tt] [Tt] [Rr] [Ii] [Bb] [Uu] [Tt] [Ee];
11 QUOTED_VALUE : '"' .*? '"' | '\'' .*? '\'';
12 NON_SPECIAL : [a-zA-Z] | [\-_];
13 NAT_NUMBER : [0-9]+;

```

Código 2.24: Versão simplificada de regras da gramática que gera WED-SQL expressas em EBNF.

As regras listadas acima se dividem entre regras léxicas expressas em caixa alta no lado esquerdo do símbolo “:” e regras sintáticas, expressas em caixa baixa. A partir dessas regras, o algoritmo de análise sintática, recebendo o comando “create wed-attribute a;” construirá a árvore sintática mostrada na Figura 2.3.

Figura 2.3: Exemplo de árvore sintática.

Uma mesma árvore sintática pode representar mais de uma derivação de uma mesma cadeia em $L(G)$, por exemplo, na árvore de geração do comando de criação do WED-attribute temos as seguintes derivações:

- $D_1 = \text{start} \Rightarrow \text{statement_list} \Rightarrow \text{create_attr_stmt SEMICOLON} \Rightarrow \text{CREATE WED_ATTRIBUTE valid_name opt_def_value_expr SEMICOLON} \Rightarrow \text{CREATE WED_ATTRIBUTE NON_SPECIAL opt_def_value_expr SEMICOLON} \Rightarrow \text{CREATE WED_ATTRIBUTE NON_SPECIAL SEMICOLON}$
- $D_2 = \text{start} \Rightarrow \text{statement_list} \Rightarrow \text{create_attr_stmt SEMICOLON} \Rightarrow \text{CREATE WED_ATTRIBUTE valid_name opt_def_value_expr SEMICOLON} \Rightarrow \text{CREATE WED_ATTRIBUTE valid_name SEMICOLON} \Rightarrow \text{CREATE WED_ATTRIBUTE NON_SPECIAL SEMICOLON}$

A diferença entre as duas derivações está na ordem de aplicação das produções $\text{valid_name} \rightarrow \text{NON_SPECIAL}$ e $\text{opt_def_value_expr} \rightarrow \epsilon$. Quando uma derivação é feita realizando primeiro

a aplicação de regras sobre os não-terminais mais à esquerda, chamamos essa derivação de mais à esquerda. De modo análogo, quando a derivação é feita priorizando aplicar regras sobre os não-terminais mais à direita, então chamamos a derivação de mais à direita. No caso do exemplo, D_1 é a derivação mais à esquerda e D_2 a derivação mais à direita.

2.3 Compilação

Compiladores são programas que fazem a tradução de sentenças de uma linguagem de alto nível para uma linguagem de nível mais baixo, podendo ser código de máquina ou uma linguagem intermediária [TST93]. O trabalho de um compilador é organizado em etapas, conforme ilustrado na Figura 2.4 e explicado nos próximos parágrafos.

Figura 2.4: *Etapas da compilação.*

A **análise léxica** tem como objetivo classificar sequências de caracteres em representações que serão úteis nas próximas etapas da tradução, tais representações são chamadas *tokens*. Essa classificação é feita pelo reconhecimento de padrões especificados por expressões regulares. Além disso, a análise léxica pode ser usada para realizar um tratamento inicial da sequência de caracteres fornecido como entrada, desconsiderando por exemplo espaços em branco, caracteres de quebra de linha e comentários.

A etapa seguinte é a **análise sintática**, que verifica se a sequência w de *tokens* resultante da análise léxica está disposta em uma ordem considerada como correta por regras sintáticas definidas na GLC que gera a linguagem, ou seja, de forma que sendo $G = (V, \Sigma, R, S)$ a GLC que gera a linguagem, existam regras em R que aplicadas sucessivamente levam do não-terminal inicial S à sequência w .

A etapa de **análise semântica** é responsável por verificações para decidir se uma entrada é válida dentro da configuração do sistema no momento de sua ocorrência.

A última etapa de um compilador é a geração da saída, na qual o compilador decide qual a melhor maneira de escrever a saída em uma linguagem de nível mais baixo, buscando a otimização da entrada a partir das informações resultantes das etapas anteriores.

Além das etapas propostas pela arquitetura, um compilador também deve possuir uma estratégia de tratamento de erros causados por entradas consideradas inválidas pelas regras definidas na gramática.

Para implementar a arquitetura proposta do compilador é necessário explicitar formalmente as regras léxicas e sintáticas da linguagem. Isto facilita o entendimento e alteração do compilador da linguagem. Além disso, o particionamento conceitual do processo de compilação nas etapas propostas é conveniente para isolar os problemas que podem ocorrer em cada etapa [AU72].

2.3.1 Algoritmos de Análise Sintática

Existem duas principais estratégias para a realização da análise sintática em uma GLC $G = (V, \Sigma, R, S)$, a estratégia *bottom-up* e a estratégia *top-down*.

Na estratégia *bottom-up*, o algoritmo parte de uma sequência de terminais resultantes da análise léxica e tenta aplicar reduções até resultar no símbolo não-terminal inicial S de G . Uma operação de redução em G é definida como o inverso de uma produção de R , ou seja, para toda regra de produção (A, a) existe uma regra de redução (a, A) . Já na estratégia *top-down*, o algoritmo parte do símbolo não-terminal inicial S da gramática e tenta encontrar uma derivação que leva à sequência de terminais equivalente aos *tokens* gerados pela análise léxica. Em ambas as estratégias os algoritmos precisam ler os terminais de entrada para decidir se a sequência é sintaticamente válida.

Analisadores sintáticos também podem ser classificados quanto ao sentido de leitura dos terminais de entrada. Geralmente os algoritmos realizam a leitura da esquerda para a direita. Neste sentido de leitura os algoritmos *top-down* tentam encontrar a derivação mais à esquerda da entrada já que aplicam as produções lendo da esquerda para a direita. De modo análogo, no mesmo sentido de leitura da esquerda para a direita os algoritmos *bottom-up* aplicam reduções lendo da esquerda para a direita, o que resulta em uma derivação composta pelas produções equivalentes em ordem reversa das reduções, ou seja, uma derivação mais à direita. Pela mesma lógica, quando falamos de algoritmos que lêem a entrada no sentido contrário, isto é, da direita para a esquerda, então algoritmos *top-down* tentam obter a derivação mais à direita e algoritmos *bottom-up* tentam gerar a derivação mais à esquerda [Knu71]. Essas classificações dão origem à seguinte nomenclatura:

- Algoritmos LR (*Left-to-right Rightmost derivation*), *bottom-up*.
- Algoritmos LL (*Left-to-right Leftmost derivation*), *top-down*.
- Algoritmos RR (*Right-to-left Rightmost derivation*), *top-down*.
- Algoritmos RL (*Right-to-left Leftmost derivation*), *bottom-up*.

Durante a análise sintática, algoritmos podem empregar mecanismos de recuperação de tentativas falhas de encontrar derivação, como empregar a estratégia de *backtrack* para voltar para um passo anterior em caso de erro até que todas as possíveis opções se esgotem. Quando é necessário um passo de recuperação em um analisador sintático, o analisador é dito não-determinístico. São chamados de determinísticos os analisadores nos quais cada passo da derivação é a única opção de passo [Knu71].

Os analisadores determinísticos têm a vantagem de executar a análise sintática em menos tempo comparado com os analisadores não-determinísticos. No entanto, para cada tipo de algoritmo de análise sintática, nem toda gramática pode ser analisada de forma determinística. O trabalho [Knu65]

apresenta as condições que devem ser satisfeitas por uma gramática para que seja deterministicamente analisada por algoritmos *bottom-up* LR(k) onde k representa o número de terminais da entrada considerados pelo analisador antes de decidir um passo. Analogamente, o trabalho [LS68] apresenta as condições para que uma gramática seja deterministicamente analisada por algoritmos *top-down* LL(k).

Quando surgiu a teoria de análise sintática os recursos computacionais eram escassos e portanto era importante que desenvolvedores condicionassem suas gramáticas às restrições exigidas pelos analisadores determinísticos. Os trabalhos [Der69] e [DeR71] apresentam respectivamente os analisadores determinísticos *bottom-up* LALR(k) (*Look-Ahead Left-to-right Rightmost derivation*) e SLR(k) (*Simple Left-to-right Rightmost derivation*), que são simplificações do algoritmo LR(k). Ambos os analisadores oferecem melhorias no uso de memória comparados ao analisador LR(k) com o custo de poderem ser aplicados a classes de gramáticas mais restritas.

Em contraste com computadores da época do surgimento da teoria de análise sintática, computadores modernos oferecem mais recursos computacionais para a execução da análise sintática, de modo que a eficiência do desenvolvedor torna-se mais importante no sentido de que mais esforços foram dirigidos a analisadores não-determinísticos [PF11]. O trabalho [Tom85] apresenta um analisador *bottom-up* não-determinístico conhecido como *Generalized Left-to-right Rightmost derivation*, ou GLR, uma extensão do algoritmo LR com a finalidade de realizar análise sintática de linguagem natural de forma eficiente. O trabalho [For02] apresenta um algoritmo não-determinístico *top-down* chamado de analisador *Packrat*, que realiza *backtrack*, porém memoriza as decisões tomadas em cada passo para melhorar a eficiência. O trabalho [PF11] apresenta o algoritmo não-determinístico *top-down* LL(*), que é uma otimização do analisador *Packrat*. O símbolo * no nome significa que pode haver variação no número de símbolos da entrada que são consultados a cada passo para tentar evitar o *backtrack* em decisões.

Uma gramática é dita diretamente recursiva à esquerda caso exista ao menos uma regra na qual o símbolo à esquerda de $\rightarrow$ é o mesmo que o primeiro símbolo à direita de $\rightarrow$, como ocorre na regra $A \rightarrow Aa$. A recursividade à esquerda nem sempre é explícita e direta, pode aparecer na forma indireta como, por exemplo, nas regras $A \rightarrow B$ e $B \rightarrow Aa$. Além disso, a recursividade à esquerda pode aparecer na forma oculta como nas regras $A \rightarrow BA$ e $B \rightarrow \varepsilon$. Geralmente algoritmos *top-down* como LL(k), *Packrat* e LL(*) não podem ser aplicados para gramáticas com recursividade à esquerda, pois a ordem de prioridade das derivações pode resultar em uma análise infinita.

O algoritmo *top-down* não determinístico ALL(*) (*Adaptative Left-to-right Leftmost derivation*) [PHF14] faz uso de um mecanismo de tomada de decisão que consiste em explorar todos os caminhos possíveis de acordo com o que já foi lido da entrada e com o restante da entrada fornecida. O algoritmo ALL(*) incrementalmente armazena para cada ponto de decisão um autômato finito determinístico contendo os possíveis caminhos, para que seja consultado futuramente, reduzindo o tempo de execução do algoritmo em análises subsequentes. Apesar de que no pior caso ALL(*) ofereça complexidade $O(n^4)$ onde n é a quantidade de símbolos analisados, o trabalho [PHF14] afirma que em prática o algoritmo ALL(*) aparenta ser linear na maioria das vezes, apresentando experimentos que mostram um melhor desempenho do que outros analisadores como GLR, *Packrat*, LALR(1), LR(k) e LL(*). Além disso, ALL(*) trata de regras que são diretamente recursivas à esquerda, removendo automaticamente a recursividade nesses casos, reduzindo sua limitação a gramáticas com regras indiretamente recursivas à esquerda ou com recursão oculta à esquerda.

2.3.2 Geração de Analisadores Léxicos e Sintáticos

Implementar manualmente algoritmos de análise léxica e sintática é uma tarefa que demanda tempo e que está sujeita a erros, portanto é conveniente usar uma das muitas ferramentas de geração automática de analisadores léxicos e sintáticos a partir da especificação formal das regras de uma linguagem. Entre as principais ferramentas estão:

- ANTLR (*ANother Tool for Language Recognition*) [Par]: Atualmente em sua versão 4, é capaz de, a partir de uma gramática expressa em EBNF, gerar analisadores léxicos e sintáticos nas linguagens Java, C#, Python (versões 2 e 3), JavaScript, Go, C++ e Swift. ANTLR é distribuída sob uma licença BSD. O algoritmo de análise sintática usado pela ANTLR é o algoritmo ALL(*).
- GNU Bison [Fou]: A partir de gramáticas expressas em BNF (Backus-Naur Form), essa ferramenta permite gerar analisadores sintáticos nas linguagens C, C++ e Java. Essa ferramenta é distribuída sob uma licença GNU GPL (General Public License). Um analisador sintático gerado por Bison pode implementar os algoritmos LR(k), LALR(1) ou GLR de acordo com a necessidade do usuário.
- Coco/R [MLWoL]: Permite gerar analisadores léxicos e sintáticos a partir de uma gramática expressa em EBNF. Na versão mais recente, os analisadores são gerados nas linguagens C#, Java e C++, embora nas versões anteriores exista suporte para outras linguagens, inclusive Python. Um analisador sintático gerado por Coco/R implementa o algoritmo LL(1).
- SableCC [ÉG]: Partindo de gramáticas expressas em EBNF, SableCC pode gerar analisadores léxicos e sintáticos nas linguagens C#, C++, O'Caml, Python, C e XML. Essa ferramenta é distribuída sob uma licença GNU LGPL (Lesser General Public License). O algoritmo LALR(1) é usado pelos analisadores sintáticos gerados por SableCC.

O algoritmo ALL(*) usado pela ferramenta ANTLR possui a limitação de não permitir a análise de gramáticas com regras indiretamente recursivas à esquerda ou com recursão oculta à esquerda, o que não ocorre atualmente nas gramáticas propostas no escopo deste trabalho.

A ferramenta ANTLR foi escolhida para gerar os analisadores neste trabalho devido aos seguintes motivos:

- A compatibilidade com Python, linguagem na qual a implementação do WED-flow foi escrita.
- As vantagens do algoritmo ALL(*).
- O fato da limitação do algoritmo de análise sintática não ser um problema para o uso neste projeto.
- As classes auxiliares de navegação que permitem integração das análises léxica e sintática com a análise semântica e o gerador de saída.

2.3.3 ANTLR

A ferramenta ANTLR é dividida em duas partes:

- O programa ANTLR que gera automaticamente analisadores léxicos e sintáticos a partir de regras de uma linguagem.
- A biblioteca ANTLR que contém classes para facilitar o uso dos analisadores, a navegação na entrada do compilador e o tratamento de erros.

A partir de regras léxicas e sintáticas expressas em linguagem EBNF, o programa ANTLR gera automaticamente os códigos que realizam análise léxica e sintática, uma classe que armazena os símbolos da gramática (*tokens* e não-terminais) e a interface *listener* que facilita a navegação na entrada do compilador [Par13].

Ao receber uma entrada, um analisador léxico gerado pelo programa ANTLR retorna uma sequência de *tokens*, que armazenam uma sequência de caracteres que satisfaz um padrão de reconhecimento e a classificação atribuída a ela. Recebendo os *tokens* como entrada, um analisador sintático gerado por ANTLR fornece uma derivação que leva a computação do símbolo inicial até os *tokens* recebidos.

A interface *listener*, que também é gerada automaticamente pelo programa ANTLR a partir da gramática, pode ser usada para definir um comportamento para ser executado na ocorrência de um não-terminal específico da árvore sintática da entrada, resultante da análise sintática, permitindo que seja adicionado código de análise semântica e geração de saída para ser executado para alguns determinados não-terminais da linguagem.

A biblioteca ANTLR oferece classes para integrar e executar os analisadores léxico e sintático gerados pelo programa ANTLR. Existe na biblioteca ANTLR uma classe que representa a árvore sintática da entrada do compilador, após a entrada ter passado pelas etapas de análise léxica e sintática. A biblioteca também contém uma classe que permite percorrer os nós da árvore sintática da entrada, facilitando a navegação pela entrada.

Quanto ao tratamento de erros, a biblioteca ANTLR oferece não só classes com um tratamento de erro padrão implementado, como também interfaces que podem ser usadas para configurar um tratamento de erros específico de acordo com a necessidade da aplicação. Nesse contexto, um erro significa que a sequência de caracteres fornecida como entrada não foi considerada válida na gramática especificada.

2.3.4 Considerações finais

A primeira seção deste capítulo contextualiza o trabalho, introduzindo a abordagem WED-flow, apresentando os conceitos da abordagem, outros trabalhos que tratam de WED-flow e um exemplo de modelagem de um processo de locação de veículos usando a abordagem. Também foi apresentada a implementação da abordagem, descrevendo mais detalhadamente a linguagem WED-SQL, que está diretamente relacionada aos compiladores propostos neste trabalho, e mostrando como os elementos da implementação se relacionam, retomando a motivação do trabalho, a dificuldade na evolução da linguagem resultante da falta de um compilador clássico.

A segunda seção apresenta os conceitos de linguagens, gramáticas, derivações e as representações EBNF, para descrever regras gramaticais, e árvore sintática, para descrever derivações de uma cadeia de caracteres em uma gramática. Uma variante da representação EBNF será usada para escrever formalmente as regras das gramáticas que geram as linguagens WED-SQL e WED-flow. A estrutura de árvore sintática será usada pela biblioteca ANTLR para armazenar uma derivação da entrada

após a realização da etapa de análise sintática para facilitar a implementação das etapas seguintes de compilação.

A terceira seção se inicia exibindo a arquitetura clássica de compiladores e suas etapas. Foram apresentados alguns algoritmos de análise sintática, contextualizando o algoritmo ALL(*) usado dos compiladores implementados neste trabalho e exibindo a limitação desse algoritmo. As principais ferramentas de geração automática de analisadores léxicos e sintáticos foram comparadas e foram exibidos os principais motivos para a escolha do uso da ferramenta ANTLR neste trabalho. A ferramenta foi descrita em mais detalhes para facilitar o entendimento dos capítulos seguintes que tratam da implementação dos compiladores fazendo uso das funcionalidades oferecidas por ANTLR.

Capítulo 3

Compilador da linguagem WED-SQL

Este capítulo apresenta o compilador da linguagem WED-SQL, partindo de sua arquitetura e mostrando como a descrição formal da linguagem e a ferramenta ANTLR auxiliam a implementação de seus componentes e oferecem flexibilidade ao compilador. Em seguida, é exibida a implementação de cada componente do compilador de forma detalhada, mostrando a descrição formal da linguagem na forma da gramática livre de contexto e explicando os detalhes e escolhas de implementação de cada etapa. Também são apresentadas algumas particularidades do uso do compilador no programa `wsql`, que faz parte da implementação atual do WED-flow.

3.1 Compilador WED-SQL

O objetivo do compilador de linguagem WED-SQL é reconhecer comandos da linguagem WED-SQL (Seção 2.1.2) e traduzi-los em linguagem SQL. A arquitetura do compilador se baseia na Figura 2.4, que divide o processo de compilação nas etapas de análise léxica, análise sintática, análise semântica e tradução da linguagem origem para a linguagem destino.

A ferramenta ANTLR permite a geração automática dos algoritmos de análise léxica e sintática a partir da descrição formal da linguagem de origem, no caso WED-SQL, expressa como uma gramática livre de contexto. Além disso, a ferramenta ANTLR gera automaticamente interfaces que auxiliam a implementação do tratamento de erros e das etapas de análise semântica e tradução para linguagem destino, no caso, a linguagem SQL. É importante observar que a ferramenta ANTLR apenas gera automaticamente os analisadores léxico e sintático e as demais etapas devem ser manualmente alteradas caso ocorram alterações na definição da linguagem.

As regras léxicas e sintáticas da gramática fornecidas à ferramenta ANTLR são expressas em linguagem EBNF. Neste trabalho, as definições estão distribuídas em dois arquivos por linguagem, de forma que um deles contém as regras léxicas e o outro contém regras sintáticas. Essa separação não é mandatória, mas resulta em uma forma mais organizada de descrever as regras da gramática.

O procedimento de compilação proposto neste trabalho é composto pelos objetos das classes automaticamente geradas pela ferramenta ANTLR para a análise léxica e sintática e de outras duas classes, responsáveis por realizar verificações de consistência das definições especificadas e por gerar o código na linguagem SQL. A interface *listener* gerada automaticamente pela ANTLR captura a ocorrência de cada comando sintaticamente válido na linguagem WED-SQL em tempo de execução e permite especificar o código de tratamento para executar após a análise sintática. Assim, a interface *listener* é usada para chamar os métodos das classes de análise semântica e de

tradução para a linguagem SQL. A interface *listener* permite que os códigos sejam implementados de forma isolada, o que é uma vantagem em relação à implementação antiga do interpretador de linguagem WED-SQL, pois uma possível alteração em uma das etapas de compilação impactará apenas na parte afetada, dificultando a introdução de comportamentos inesperados na compilação.

Figura 3.1: *Arquitetura do compilador da linguagem WED-SQL.*

A Figura 3.1 ilustra a arquitetura do compilador, os comandos expressos na linguagem WED-SQL serão fornecidos como entrada do compilador, sendo processados pelos analisadores léxico, sintático e semântico, dos quais os dois primeiros são automaticamente gerados a partir das regras expressas na gramática da linguagem e o analisador semântico é escrito manualmente fazendo uso da interface *Listener* gerada também pelo ANTLR a partir da descrição da linguagem. Por fim, o gerador de código na linguagem alvo, também escrito manualmente fazendo uso da interface *listener*, gera a saída do compilador em linguagem SQL.

3.2 Análise Léxica

O objetivo da análise léxica é classificar a entrada do compilador em uma sequência de *tokens* considerados válidos de acordo com as regras léxicas. Nesta seção serão apresentadas as regras léxicas fornecidas à ferramenta ANTLR para a geração do algoritmo de análise léxica usado no compilador da linguagem WED-SQL, e serão descritos detalhes do processo de geração do analisador para explicar o impacto que decisões na escrita das regras têm no analisador gerado por ANTLR.

As regras léxicas são definições de sequências de caracteres, ou *tokens*, descritas em uma sintaxe variante da EBNF usada pela ferramenta ANTLR. Por convenção da ANTLR, os nomes das regras léxicas devem iniciar com letras maiúsculas para que sejam diferenciáveis de regras sintáticas, já que as regras de ambos os tipos são escritas usando a mesma sintaxe e podem estar dispostas no mesmo arquivo [Par13]. Seguem exemplos de definições das regras léxicas da gramática que gera a linguagem WED-SQL:

```

1 CREATE : [Cc] [Rr] [Ee] [Aa] [Tt] [Ee];
2 WED_FLOW : [Ww] [Ee] [Dd] '-' [Ff] [Ll] [Oo] [Ww];
3 REMOVE : [Rr] [Ee] [Mm] [Oo] [Vv] [Ee];
4 FLOW : [Ff] [Ll] [Oo] [Ww];
5 BEGIN : [Bb] [Ee] [Gg] [Ii] [Nn];
6 COMMIT : [Cc] [Oo] [Mm] [Mm] [Ii] [Tt];
7 ABORT : [Aa] [Bb] [Oo] [Rr] [Tt];
8 ROLLBACK : [Rr] [Oo] [Ll] [Ll] [Bb] [Aa] [Cc] [Kk];
9 CW : '\\' [Cc] [Ww];
10 LW : '\\' [Ll] [Ww];
11 WED_ATTRIBUTES : [Ww] [Ee] [Dd] '-' [Aa] [Tt] [Tt] [Rr] [Ii] [Bb] [Uu] [Tt] [Ee] [Ss];
12 WED_ATTRIBUTE : [Ww] [Ee] [Dd] '-' [Aa] [Tt] [Tt] [Rr] [Ii] [Bb] [Uu] [Tt] [Ee];
13 WED_CONDITIONS : [Ww] [Ee] [Dd] '-' [Cc] [Oo] [Nn] [Dd] [Ii] [Tt] [Ii] [Oo] [Nn] [Ss];
14 WED_CONDITION : [Ww] [Ee] [Dd] '-' [Cc] [Oo] [Nn] [Dd] [Ii] [Tt] [Ii] [Oo] [Nn];
15 WED_TRANSITIONS : [Ww] [Ee] [Dd] '-' [Tt] [Rr] [Aa] [Nn] [Ss] [Ii] [Tt] [Ii] [Oo] [Nn] [Ss];
16 WED_TRANSITION : [Ww] [Ee] [Dd] '-' [Tt] [Rr] [Aa] [Nn] [Ss] [Ii] [Tt] [Ii] [Oo] [Nn];
17 WED_TRIGGERS : [Ww] [Ee] [Dd] '-' [Tt] [Rr] [Ii] [Gg] [Gg] [Ee] [Rr] [Ss];
18 WED_TRIGGER : [Ww] [Ee] [Dd] '-' [Tt] [Rr] [Ii] [Gg] [Gg] [Ee] [Rr];
19 INSTANCES : [Ii] [Nn] [Ss] [Tt] [Aa] [Nn] [Cc] [Ee] [Ss];
20 INSTANCE : [Ii] [Nn] [Ss] [Tt] [Aa] [Nn] [Cc] [Ee];
21 INCONSISTENT : [Ii] [Nn] [Cc] [Oo] [Nn] [Ss] [Ii] [Ss] [Tt] [Ee] [Nn] [Tt];
22 TRANSACTIONING : [Tt] [Rr] [Aa] [Nn] [Ss] [Aa] [Cc] [Tt] [Ii] [Oo] [Nn] [Ii] [Nn] [Gg];
23 FINAL : [Ff] [Ii] [Nn] [Aa] [Ll];
24 SET : [Ss] [Ee] [Tt];
25 WED_STATE : [Ww] [Ee] [Dd] '-' [Ss] [Tt] [Aa] [Tt] [Ee];
26 ENABLE : [Ee] [Nn] [Aa] [Bb] [Ll] [Ee];
27 DISABLE : [Dd] [Ii] [Ss] [Aa] [Bb] [Ll] [Ee];
28 LIST : [Ll] [Ii] [Ss] [Tt];
29 FIRED : [Ff] [Ii] [Rr] [Ee] [Dd];
30 GET : [Gg] [Ee] [Tt];
31 WED_TRACE : [Ww] [Ee] [Dd] '-' [Tt] [Rr] [Aa] [Cc] [Ee];
32 SHOW : [Ss] [Hh] [Oo] [Ww];
33 DEFAULT : [Dd] [Ee] [Ff] [Aa] [Uu] [Ll] [Tt];
34 VALUES : [Vv] [Aa] [Ll] [Uu] [Ee] [Ss];
35 VALUE : [Vv] [Aa] [Ll] [Uu] [Ee];
36 CONDITION : [Cc] [Oo] [Nn] [Dd] [Ii] [Tt] [Ii] [Oo] [Nn];
37 TIMEOUT : [Tt] [Ii] [Mm] [Ee] [Oo] [Uu] [Tt];
38 STOP : [Ss] [Tt] [Oo] [Pp];

```

```

39 ON : [Oo] [Nn];
40 TERMINATE : [Tt] [Ee] [Rr] [Mm] [Ii] [Nn] [Aa] [Tt] [Ee];
41 REINSTATE : [Rr] [Ee] [Ii] [Nn] [Ss] [Tt] [Aa] [Tt] [Ee];
42 AS : [Aa] [Ss];
43 FOR : [Ff] [Oo] [Rr];
44 SP : [Ss] [Pp];

```

Código 3.1: Exemplos de regras léxicas que definem tokens usados pela linguagem WED-SQL.

A maior parte das definições para a linguagem WED-SQL especificam *tokens* para palavras reservadas da linguagem WED-SQL. É importante observar que o padrão `[Aa]` reconhece tanto o caractere ‘a’ quanto o caractere ‘A’. Assim, as três primeiras regras listadas acima reconhecem respectivamente as cadeias de caracteres ‘create’, ‘wed-flow’ e ‘remove’ sem diferenciar as letras minúsculas de maiúsculas, ou seja, de modo *case-insensitive*. Seguem outras regras léxicas que também reconhecem palavras reservadas da linguagem WED-SQL:

Quando uma entrada é processada pelo algoritmo de análise léxica, são prioritárias as regras sintáticas em relação às léxicas, ou seja, se uma cadeia de caracteres satisfaz ao mesmo tempo os padrões de uma regra léxica e de uma sintática, então o analisador léxico não vai atribuir o *token* para a cadeia, a regra sintática tem prioridade e a cadeia será usada na etapa de análise sintática. Para evitar esse tipo de ambiguidade entre regras léxicas e sintáticas, este trabalho não faz uso de cadeias de caracteres na especificação de regras sintáticas, as cadeias são usadas para definir as regras léxicas enquanto as regras sintáticas são definidas sobre *tokens*.

Também é possível a ocorrência de ambiguidade entre duas ou mais regras léxicas. Nesse caso, o algoritmo léxico gerado por ANTLR prioriza a regra léxica que é lida antes na gramática, ou seja, a ordem de escrita das regras léxicas na gramática é importante quando ocorre ambiguidade entre regras léxicas. Esse tipo de ambiguidade não ocorre na linguagem WED-SQL.

As demais regras léxicas da linguagem WED-SQL são:

```

1 SEMICOLON : ';';
2 COMMA : ',';
3 OPEN_PARENTHESIS : '(';
4 CLOSE_PARENTHESIS : ')';
5 LINE_COMMENT : ('/' '/' | '—') .*? SEMICOLON;
6 fragment DOUBLE_QUOTES : '"';
7 fragment SINGLE_QUOTES : '\'';
8 QUOTED_VALUE : DOUBLE_QUOTES .*? DOUBLE_QUOTES | SINGLE_QUOTES .*?
  SINGLE_QUOTES;
9 NON_SPECIAL : [a-zA-Z] | [\_—];
10 NAT_NUMBER : DIGIT+;
11 fragment DIGIT : [0–9];
12 fragment VALID_NAME : (NON_SPECIAL | NAT_NUMBER | CREATE | WED_FLOW | FLOW |
  REMOVE | BEGIN | COMMIT | ABORT | ROLLBACK | WED_ATTRIBUTE | WED_CONDITION
  | WED_TRANSITION | WED_TRIGGER | WED_ATTRIBUTES | WED_CONDITIONS |
  WED_TRANSITIONS | WED_TRIGGERS | INSTANCES | INSTANCE | INCONSISTENT |
  TRANSACTIONING | FINAL | SET | WED_STATE | ENABLE | DISABLE | LIST | FIRED | GET |
  WED_TRACE | SHOW | DEFAULT | VALUES | VALUE | CONDITION | TIMEOUT | STOP | ON |
  AS | FOR | SP | TERMINATE | REINSTATE )+;
13 SET_FINAL_STATE_EXPR : FINAL SPACE+ WED_STATE SPACE+ AS SPACE+ .+? SEMICOLON;
14 CONDITION_EXPR : WED_CONDITION SPACE+ VALID_NAME SPACE+ AS SPACE+ (. |
  QUOTED_VALUE)+? SEMICOLON;
15 SPACE : ' ';

```

```
16 LB : [\t\r\n]+->skip;
```

Código 3.2: Exemplos de regras léxicas que definem tokens usados pela linguagem WED-SQL.

Um comportamento comum em analisadores léxicos gerados automaticamente, inclusive nos gerados por ANTLR, é que os analisadores seguem o princípio da máxima correspondência (*Maximal Munch Principle*) [Par13], o que significa que o analisador sempre tenta validar a maior entrada possível em uma única regra léxica. Isso significa que uma entrada, por exemplo, “wed-transitions” seria associada inteira ao *token* “WED_TRANSITIONS” ao invés de ser quebrada em duas partes (“wed-transition” e “s”) e associada aos *tokens* “WED_TRANSITION” e “NON_SPECIAL”, independentemente da ordem de leitura dessas três regras na gramática.

Regras que são precedidas pela palavra “fragment” são somente usadas na definição de outras regras léxicas e portanto não geram *tokens*. Por exemplo, a regra “DIGIT” é precedida por “fragment”, então quando uma entrada satisfizer essa regra o analisador não gerará um *token* “DIGIT” para ser usado na próxima etapa da compilação (análise sintática). Nesse caso, a regra “DIGIT” serve apenas para facilitar a escrita e o entendimento humano de outras regras léxicas, como “NAT_NUMBER”.

Na EBNF usada para definir os padrões em ANTLR, o símbolo “.” é satisfeito por qualquer caractere, sendo assim, o trecho “.*” representa uma iteração que é satisfeita por uma cadeia de qualquer tamanho de quaisquer caracteres e, quando usado para definir uma regra léxica, por conta do princípio de máxima correspondência, o analisador considerará a maior cadeia que satisfaz essa regra, ou seja, toda a entrada. Regras que apresentam o comportamento de consumir toda a entrada usando um padrão iterativo são chamadas regras gulosas.

A sintaxe usada por ANTLR permite o uso de expressões não-gulosas fazendo o uso do sufixo “?”, que são expressões iterativas delimitadas que capturam a menor cadeia de caracteres da entrada que satisfaz as delimitações especificadas. Por exemplo, a regra que define o *token* “QUOTED_VALUE” é composta por uma iteração não-gulosa “.*?” significando que a regra deve consumir a menor cadeia de quaisquer caracteres entre aspas simples ou duplas. Caso a regra fosse escrita com a expressão gulosa “.*” o algoritmo consideraria o *token* como a primeira ocorrência de aspas simples ou dupla seguida de quaisquer caracteres (inclusive aspas) até a última ocorrência de uma aspas simples ou dupla, podendo considerar toda uma sequência de comandos WED-SQL como um valor entre aspas, o que não é desejado.

A regra “CONDITION_EXPR” é um caso especial de regra não-gulosa. É desejável que a parte do predicado de uma WED-condition seja permissiva, uma vez que o predicado está sujeito às particularidades sintáticas de diferentes SGBDRs, o que impossibilita a especificação de regras de interpretação do predicado. Assim, o compilador abstrai-se da validação do predicado, que é realizada pelo próprio SGBDR após a compilação dos comandos para SQL e a execução do código SQL no SGBDR.

Geralmente uma regra não-gulosa determina padrões ou *tokens* como delimitadores da parte iterativa da regra, porém não é possível definir um símbolo ou um *token* como delimitador da regra “CONDITION_EXPR”, já que há o risco do mesmo delimitador fazer parte da sintaxe do SGBDR e estar presente no predicado, o que causaria um erro de validação na etapa de análise léxica para uma entrada lexicamente correta. Para resolver esse problema, a parte iterativa não-gulosa dessa regra recebe um delimitador inicial muito específico, a sequência de *tokens* que compõem o comando de criação de WED-conditions “WED_CONDITION SPACE+ VALID_NAME SPACE+ AS SPACE+”, que muito provavelmente não fará parte do predicado. O outro delimitador é o

símbolo “;”, que é usado para terminar um comando WED-SQL e é geralmente usado para indicar o final de um comando SQL na maioria dos SGBDRs, e portanto não deve fazer parte de um predicado SQL.

Para auxiliar a definição de regras léxicas, ANTLR oferece alguns poucos e básicos comandos léxicos que podem ser usados para realizar uma ação simples sobre a entrada que satisfazer alguma regra léxica. O comando léxico “skip”, usado nessa gramática na definição da regra “LB”, realiza a ação de desconsiderar a parte da entrada que satisfaz a regra sem gerar um *token*, isso significa que quando o padrão dessa regra for satisfeito por algum caractere de quebra de linha, o analisador léxico não criará o *token* da quebra de linha para as próximas etapas, efetivamente ignorando a quebra de linha.

3.3 Análise Sintática

Esta seção explica como foram especificadas as regras sintáticas da linguagem WED-SQL levando em conta o impacto de tais definições no analisador resultante da geração automática pela ferramenta ANTLR a partir das regras sintáticas.

Embora a etapa de análise sintática ocorra após a etapa de análise léxica em um compilador, a definição de *tokens* deve ser baseada em como eles servirão para compor as regras sintáticas. A partir do símbolo inicial e da expansão das regras sintáticas, o analisador sintático decide se um comando é válido, ou seja, se existe uma derivação que resulta na mesma sequência de *tokens* que representa o comando.

O algoritmo de análise sintática gerado deve ser capaz de reconhecer não apenas todos os comandos considerados válidos na linguagem WED-SQL, mas também qualquer sequência de comandos WED-SQL. Assim, as regras sintáticas são inicialmente definidas como:

```

1 wsq: stmt _list;
2
3 stmt _list: SPACE* stmt SPACE* stmt _list
4           | SPACE* stmt SPACE*;
5
6 stmt: create_flow_stmt SPACE* SEMICOLON
7      | remove_flow_stmt SPACE* SEMICOLON
8      | create_attr_stmt SPACE* SEMICOLON
9      | create_cond_stmt
10     | create_trans_stmt SPACE* SEMICOLON
11     | create_trig_stmt SPACE* SEMICOLON
12     | create_inst_stmt SPACE* SEMICOLON
13     | set_fstate_stmt
14     | enable_trig_stmt SPACE* SEMICOLON
15     | disable_trig_stmt SPACE* SEMICOLON
16     | list_attr_stmt SPACE* SEMICOLON
17     | list_cond_stmt SPACE* SEMICOLON
18     | list_trans_stmt SPACE* SEMICOLON
19     | list_trig_stmt SPACE* SEMICOLON
20     | list_ftrans_stmt SPACE* SEMICOLON
21     | list_insts_stmt SPACE* SEMICOLON
22     | show_fstate_stmt SPACE* SEMICOLON
23     | get_trace_stmt SPACE* SEMICOLON
24     | terminate_stmt SPACE* SEMICOLON

```



```

25 | reinstate_stmt SPACE* SEMICOLON
26 | begin_stmt SPACE* SEMICOLON
27 | commit_stmt SPACE* SEMICOLON
28 | abort_stmt SPACE* SEMICOLON
29 | shell_connect_flow_stmt SPACE*
30 | shell_list_flow_stmt SPACE*
31 | LINE_COMMENT;

```

Código 3.3: Regras sintáticas que representam os comandos da linguagem WED-SQL.

Partindo do símbolo inicial “wsql”, a única regra possível nesse ponto é o não-terminal “stmt_list”, que representa a lista de comandos WED-SQL válidos, cada comando da lista é representado por um não-terminal “stmt”. Existe uma regra partindo de “stmt_list” que é considerada recursiva à direita, pois o mesmo não-terminal “stmt_list” aparece na posição mais à direita da regra “SPACE* stmt SPACE* stmt_list”. É importante garantir que não existam na gramática regras sintáticas com recursividade indireta à esquerda por conta da limitação da ferramenta ANTLR explicada na Seção 2.3.1. No entanto, os comandos da linguagem WED-SQL atualmente não demandam o uso de recursão direta ou indireta à esquerda.

O não-terminal “stmt” tem muitas opções de expansão para representar os diferentes tipos de comando na linguagem WED-SQL, a seguir são exibidos exemplos de regras de expansão de “stmt”.

```

1 create_flow_stmt: CREATE SPACE+ FLOW SPACE+ valid_name;
2 remove_flow_stmt: REMOVE SPACE+ FLOW SPACE+ valid_name;
3 create_attr_stmt: CREATE SPACE+ WED_ATTRIBUTE SPACE+ valid_name SPACE*
    opt_def_value_expr;
4 create_cond_stmt: CREATE SPACE+ CONDITION_EXPR;
5 create_trans_stmt: CREATE SPACE+ WED_TRANSITION SPACE+ valid_name SPACE*
    opt_stop_expr SPACE* opt_proc_expr;
6 create_trig_stmt: CREATE SPACE+ WED_TRIGGER SPACE+ valid_name SPACE+ AS SPACE+
    trigger_expr;
7 create_inst_stmt: CREATE SPACE+ INSTANCE SPACE+ AS SPACE+ inst_expr;
8 set_fstate_stmt: SET SPACE+ SET_FINAL_STATE_EXPR;
9 enable_trig_stmt: ENABLE SPACE+ WED_TRIGGER SPACE+ valid_name;
10 disable_trig_stmt: DISABLE SPACE+ WED_TRIGGER SPACE+ valid_name;
11 list_attr_stmt: LIST SPACE+ WED_ATTRIBUTES | LIST SPACE+ WED_ATTRIBUTES SPACE+
    valid_name;
12 list_cond_stmt: LIST SPACE+ WED_CONDITIONS | LIST SPACE+ WED_CONDITIONS SPACE+
    valid_name;
13 list_trans_stmt: LIST SPACE+ WED_TRANSITIONS | LIST SPACE+ WED_TRANSITIONS SPACE+
    valid_name;
14 list_trig_stmt: LIST SPACE+ WED_TRIGGERS | LIST SPACE+ WED_TRIGGERS SPACE+
    valid_name;
15 list_ftans_stmt: LIST SPACE+ FIRED SPACE+ WED_TRANSITIONS SPACE* opt_id;
16 list_insts_stmt: LIST SPACE+ INSTANCES | LIST SPACE+ TRANSACTIONING SPACE+ INSTANCES
    | LIST SPACE+ INCONSISTENT SPACE+ INSTANCES | LIST SPACE+ FINAL SPACE+
    INSTANCES | LIST SPACE+ INSTANCES SPACE* opt_id | LIST SPACE+ TRANSACTIONING
    SPACE+ INSTANCES SPACE* opt_id | LIST SPACE+ INCONSISTENT SPACE+ INSTANCES
    SPACE* opt_id | LIST SPACE+ FINAL SPACE+ INSTANCES SPACE* opt_id;
17 show_fstate_stmt: SHOW SPACE+ FINAL SPACE+ WED_STATE;
18 get_trace_stmt: GET SPACE+ WED_TRACE SPACE+ FOR SPACE+ NAT_NUMBER;
19 terminate_stmt: TERMINATE SPACE+ INSTANCE SPACE+ NAT_NUMBER;
20 reinstate_stmt: REINSTATE SPACE+ INSTANCE SPACE+ NAT_NUMBER;

```

```

21 begin_stmt: BEGIN;
22 commit_stmt: COMMIT;
23 abort_stmt: ABORT | ROLLBACK;
24 shell_connect_flow_stmt: CW SPACE+ valid_name;
25 shell_list_flow_stmt: LW;

```

Código 3.4: Regras sintáticas da linguagem WED-SQL.

O analisador sintático gerado por ANTLR se baseia nas regras sintáticas para tentar construir uma árvore sintática que representa a entrada na qual as folhas são os *tokens* resultantes da análise léxica, a raiz equivale ao não-terminal “wsql” e os demais nós representam todos os não-terminais necessários para chegar da raiz até os *tokens* da entrada. Seguem as demais regras sintáticas que definem não-terminais:

```

1 opt_def_value_expr:
2     | DEFAULT SPACE+ VALUE SPACE+ value_expr;
3 value_expr: QUOTED_VALUE
4     | NAT_NUMBER;
5 opt_stop_expr:
6     | STOP SPACE+ ON SPACE+ (stop_condition_expr (SPACE+ timeout_expr)? SPACE* |
7         timeout_expr (SPACE+ stop_condition_expr)? SPACE*);
8 timeout_expr: TIMEOUT SPACE+ QUOTED_VALUE;
9 stop_condition_expr: CONDITION SPACE+ valid_name;
10 opt_proc_expr:
11     | AS SPACE+ SP;
12 trigger_expr : OPEN_PARENTHESIS SPACE* valid_name SPACE* COMMA SPACE* valid_name SPACE
13     * CLOSE_PARENTHESIS;
14 inst_expr : DEFAULT SPACE+ VALUES
15     | attribute_list_expr value_list_expr;
16 value_list_expr : SPACE+ VALUES SPACE+ OPEN_PARENTHESIS SPACE* value_expr SPACE* (
17     COMMA SPACE* value_expr SPACE*)* CLOSE_PARENTHESIS ;
18 attribute_list_expr : OPEN_PARENTHESIS SPACE* valid_name SPACE* (COMMA SPACE* valid_name
19     SPACE*)* CLOSE_PARENTHESIS;
20 opt_id :
21     | FOR SPACE* NAT_NUMBER;
22 valid_name : (NON_SPECIAL | NAT_NUMBER | reserved_words)+;
23 reserved_words : (CREATE | WED_FLOW | FLOW | REMOVE | BEGIN | COMMIT | ABORT |
24     ROLLBACK | WED_ATTRIBUTE | WED_CONDITION | WED_TRANSITION | WED_TRIGGER
25     | WED_ATTRIBUTES | WED_CONDITIONS | WED_TRANSITIONS | WED_TRIGGERS |
26     INSTANCES | INSTANCE | INCONSISTENT | TRANSACTIONING | FINAL | SET | WED_STATE |
27     ENABLE | DISABLE | LIST | FIRED | GET | WED_TRACE | SHOW | DEFAULT | VALUES |
28     VALUE | CONDITION | TIMEOUT | STOP | ON | AS | FOR | SP | TERMINATE | REINSTATE);

```

Código 3.5: Demais regras sintáticas da linguagem WED-SQL.

O símbolo “|” representa a união entre padrões, isso significa que na regra “opt_def_value_expr: | DEFAULT SPACE+ VALUE SPACE+ value_expr;” são aceitos tanto o padrão “DEFAULT SPACE+ VALUE SPACE+ value_expr;” quanto a cadeia vazia, ou seja, a parte do comando que o não-terminal “opt_def_value_expr” representa é opcional.

3.4 Análise Semântica

A etapa de análise semântica para a linguagem WED-SQL visa detectar erros de consistência na declaração das estruturas do WED-flow que dependem de outras estruturas do modelo. Como exemplo de verificação de consistência, se for definida uma WED-trigger *g1* associando o identificador *c1* de uma condição com o identificador *t1* de uma transação, então ambas as estruturas *c1* e *t1* devem ter sido declaradas previamente para que a especificação seja consistente.

Quanto à implementação da análise semântica, como o modelo se encontra em um banco de dados, em versões anteriores da implementação foram declaradas restrições de consistência nas tabelas que representam as estruturas de modelo. Porém, para que o compilador seja independente de SGBDR, foi implementada uma classe Python com métodos de verificação de consistência usados na etapa de análise semântica. Essa classe não é gerada automaticamente pela ferramenta ANTLR e caso seja necessário adicionar ou alterar alguma verificação semântica, essa classe deve ser alterada manualmente.

Embora a ferramenta ANTLR não automatize a etapa de análise semântica do compilador, ela ainda facilita sua implementação. Quando ANTLR gera os analisadores léxico e sintático também cria uma interface *listener* que permite definir ações para serem executadas quando cada uma das regras sintáticas é validada pelo analisador sintático. Assim, foi criada uma classe no programa *wsql* que implementa a interface *listener* da linguagem e chama os métodos da classe de análise semântica quando necessário. A mesma estratégia é usada para chamar os métodos da etapa de tradução.

Atualmente estão implementadas nessa etapa as seguintes validações de consistência de modelo:

- Existência da WED-condition e da WED-transition quando uma WED-trigger é declarada usando identificadores de condição ou transação.
- Existência de WED-condition quando é informada na criação de uma WED-transition uma condição de parada.
- Existência dos WED-attributes usados para definir o estado inicial de uma instância.

Embora essa etapa verifique a existência dos WED-attributes que definem o estado inicial, não é viável realizar essa verificação para os WED-attributes que compõem os predicados das WED-conditions. A regra sintática de definição de WED-condition é muito permissiva quanto ao predicado, permitindo o uso de construções próprias de diferentes SGBDRs, porém tornando a identificação dos WED-attributes usados inviável.

A classe de análise semântica mantém listas das estruturas sobre as quais a consistência deve ser verificada (WED-attributes, WED-conditions e WED-transitions), que são inicialmente populadas com as informações do banco de dados, caso já exista um banco de dados para o modelo com alguma WED-trigger definida. Porém não basta carregar as informações do banco de dados, pois os comandos WED-SQL podem estar dispostos dentro de uma transação e se houver criação de novas WED-attributes, WED-conditions ou WED-transitions, elas não estarão no banco de dados no momento da verificação. Para resolver esse problema, as listas são atualizadas na própria etapa de verificação semântica, conforme as estruturas são criadas. Assim, as listas são mantidas atualizadas para que os métodos possam verificar se as estruturas indicadas na entrada estão nas listas.

Para implementar a obtenção das estruturas já definidas no banco de dados na verificação semântica e continuar a garantir a independência do compilador quanto ao SGBDR, a classe de validação semântica recebe do programa `wsq` uma função como parâmetro denominado *“running_function”*, que recebe um comando SQL e é responsável por sua formatação e execução no SGBDR Postgres, retornando o resultado do comando SQL caso haja resultado. Portanto, caso um outro programa queira fazer uso do compilador com outros SGBDR, basta fornecer à classe de validação semântica uma outra *“running_function”* que se atende às particularidades do SGBDR usado.

A seguir é apresentado como exemplo o método de análise semântica do comando de criação de WED-trigger:

```

1      def validate_trigger_creation(self, ctx):
2          trigger_id = ctx.valid_name().getText()
3          trigger_expr = ctx.trigger_expr()
4          cond_id = trigger_expr.valid_name(0).getText()
5          trans_id = trigger_expr.valid_name(1).getText()
6          error = ""
7          if cond_id not in self.condition_set:
8              error += "Error: Error while attempting to create trigger '{}',\n"
                        "condition '{}' does not exist.\n".format(trigger_id, cond_id)
9          if trans_id not in self.transition_set:
10             error += "Error: Error while attempting to create trigger '{}',\n"
                       "transition '{}' does not exist.\n".format(trigger_id, trans_id)
11         return error

```

Código 3.6: Validação semântica de criação de WED-trigger

O método apresentado acima recebe um objeto “ctx” que representa a árvore sintática da derivação da entrada como o comando de criação de WED-trigger e retorna uma mensagem de erro no caso da verificação semântica do comando falhar. As linhas 2 a 5 do código apresentado usam os métodos do objeto de entrada para obter os identificadores da WED-condition e da WED-transition que compõem a WED-trigger declarada na entrada. As demais linhas verificam se a WED-condition e a WED-transition informadas na entrada foram encontradas no banco de dados. Caso não tenham sido, prepara uma mensagem de erro para o retorno do método.

3.5 Tradução para SQL

A etapa de tradução para SQL é responsável por gerar comandos SQL que serão executados sobre as estruturas do SGBDR que representam o modelo WED-flow.

De forma similar à etapa de análise semântica, a implementação da tradução deve ser implementada manualmente definindo uma classe para o tradutor com métodos que serão usados no programa `wsq`, chamados por outra classe que implementa a interface *listener*.

No entanto, é importante observar que nem todos os comandos da linguagem WED-SQL são traduzidos para a linguagem SQL, alguns comandos devem realizar ações mais complexas no banco de dados que dependem das particularidades de cada SGBDR, como é o caso do comando de criação do WED-flow, que cria e configura todas as tabelas usadas para representar o modelo, ou então os comandos de controle de transação de banco de dados, como “begin”, “commit” e “rollback”. Para comandos como esses não é realizada a etapa de tradução, as ações que devem ser realizadas são

responsabilidade do programa que faz uso do compilador. Na implementação atual, o programa *wsql* implementa a ação de cada um desses comandos usando a interface *listener*.

O código apresentado a seguir exemplifica como é implementada a etapa de tradução para SQL, exibindo o método responsável pela tradução do comando WED-SQL de criação de WED-trigger para a linguagem SQL. O método retorna o comando SQL resultante e os parâmetros usados no comando e recebe como entrada na variável “ctx” um objeto da classe que representa uma árvore sintática construída para validar o comando de criação de WED-trigger em WED-SQL já validado pelas etapas anteriores de compilação. As linhas de 2 a 5 percorrem a árvore sintática para buscar os identificadores da WED-trigger criada, da WED-condition e da WED-transition usadas na definição. As linhas 6 e 7 montam o comando SQL equivalente à criação de WED-trigger e os parâmetros desse comando, respectivamente.

```

1      def create_trigger(self, ctx):
2          trigger_id = ctx.valid_name().getText()
3          trigger_expr = ctx.trigger_expr()
4          cond_id = trigger_expr.valid_name(0).getText()
5          trans_id = trigger_expr.valid_name(1).getText()
6          stmt = "INSERT INTO wed_trig (tgname, cname, trname) VALUES (%s,%s,%s);"
7          params = (trigger_id, cond_id, trans_id,)
8          return (stmt, params)

```

Código 3.7: Tradução do comando de criação de WED-trigger para SQL.

A seguir é apresentado um exemplo de como a interface *listener* é usada para executar os métodos de análise semântica e de tradução para a linguagem SQL. A etapa de análise sintática cria uma árvore sintática para uma entrada válida, essa árvore sintática pode ser percorrida usando um objeto “walker” oferecido pela ferramenta ANTLR. A interface *listener* possui um método para cada não-terminal da linguagem, de modo que quando o objeto *walker* percorrendo a árvore sintática da entrada alcança um não-terminal, seu respectivo método da classe *listener* é executado. No exemplo apresentado é implementado o método “enterCreate_trig_stmt” da interface que é executado quando o não-terminal “create_trig_stmt” é atingido pelo *walker*. Esse método recebe como entrada o objeto “ctx” que representa a sub-árvore sintática com origem no nó do não-terminal “create_trig_stmt” que disparou o método. As linhas de 2 a 4 executam o método da análise semântica, parando a execução em caso de erro. A linha 5 chama o método de tradução para obter o código SQL equivalente ao comando de criação de WED-trigger, que é executado na linha que a sucede.

```

1      def enterCreate_trig_stmt(self, ctx):
2          self.errors = self.semantic.validate_trigger_creation(ctx)
3          if self.errors != "":
4              raise Exception(self.errors)
5          (stmt, params) = self.translator.create_trigger(ctx)
6          self.SQL(stmt, params)

```

Código 3.8: Uso da interface *listener* no programa *wsql* para chamar os métodos de análise semântica e de tradução para SQL.

3.6 Tratamento de Erros

No contexto de compiladores, é chamada de erro a situação na qual a entrada fornecida ao compilador não é considerada válida por alguma das etapas de análise léxica, sintática ou semântica. É necessário determinar um comportamento para o programa no caso de ocorrência de erros, isto é, definir um tratamento de erros.

O tratamento de erros não faz parte da implementação do compilador em si, mas sim da implementação do programa que faz uso do compilador, no caso o programa `wsql`. Embora não seja parte do compilador, o tratamento de erros depende de como o compilador é implementado, uma vez que precisa saber quais os tipos de exceções levantadas pelo compilador que devem ser tratadas.

Quando uma entrada é enviada ao programa `wsql`, o comportamento desejado é tentar compilar a entrada como um comando WED-SQL e se a entrada não for válida, então o programa tenta executar o comando como um comando do programa `psql` do SGBDR Postgres, caso o `psql` retorne um erro ao receber a entrada, então o programa `wsql` exibe o erro. Esse comportamento foi definido desse jeito para oferecer praticidade a usuários que estão familiarizados com a linguagem SQL e com o SGBDR Postgres usado nessa implementação da abordagem WED-flow. Sendo assim, esses usuários podem verificar e alterar o estado do banco de dados utilizando o mesmo programa em que executam seus comandos WED-SQL.

É importante observar que embora o programa `wsql` contenha especificidades do SGBDR Postgres, o código do compilador em si é independente de SGBDR, podendo ser usado por outros programas voltados para outros SGBDRs e que podem tratar erros de uma forma diferente.

Os analisadores léxico e sintático gerados pela ferramenta ANTLR possuem um mecanismo pré-definido de tratamento de erros, responsável por detectar e emitir exceções para tipos específicos de erros, exibir mensagens informativas para cada exceção e tentar se recuperar do erro.

As exceções lançadas pelos analisadores gerados por ANTLR podem ser referentes a casos específicos, por exemplo, os analisadores são capazes de estimar quando falta apenas um *token* para que uma regra sintática seja válida e, nesse caso, é lançada uma exceção específica de *token* faltante.

Nos analisadores léxicos e sintáticos há um objeto chamado “*errorListener*”, que define onde e como são exibidas as mensagens de erro das exceções lançadas, como padrão os analisadores são configurados para exibir mensagens de erro informativas na saída padrão, mostrando o tipo de erro, os números da linha e coluna da entrada nas quais foi detectado o erro e mais detalhes dependendo do tipo de exceção.

Os analisadores sintáticos gerados por ANTLR também possuem uma configuração de estratégia de tratamento de erros, que define qual deve ser o comportamento do programa após encontrar um erro na entrada, o comportamento pré-definido na classe “*DefaultErrorStrategy*” da biblioteca ANTLR tenta recuperar a execução do compilador para a entrada, consumindo os *tokens* da regra com erro até chegar em um *token* que o algoritmo julga como seguro para continuar a execução normal do compilador.

No entanto, o comportamento pré-definido da biblioteca ANTLR não é o desejado para o programa `wsql` e podemos redefini-lo configurando os analisadores léxico e sintático.

Existe na biblioteca ANTLR uma alternativa à estratégia padrão de recuperação, o uso da estratégia implementada na classe “*BailErrorStrategy*”, que quando recebe um erro interrompe a execução do compilador para a entrada fornecida, sem tentar recuperar a execução para o restante

da entrada. Portanto, a configuração de estratégia de recuperação de erros do analisador sintático WED-SQL foi alterada para usar a classe “*BailErrorStrategy*”, que oferece o comportamento desejado para o uso no programa *wsql*.

Também foi necessário alterar a configuração de exibição de mensagens de erro do compilador, já que não é desejado que sejam exibidas no *wsql* pois quando o usuário executar um comando SQL puro a exibição de mensagens de erro referentes às regras WED-SQL pode causar confusão. Consequentemente foram removidos os objetos “*errorListener*” dos analisadores léxico e sintático da linguagem WED-SQL. Além disso, as exceções lançadas pela etapa de análise semântica são tratadas silenciosamente.

Concluindo, o comportamento resultante do tratamento de erros implementado para o programa *wsql* é que quando uma entrada é considerada inválida na linguagem WED-SQL, o compilador não segue a execução para a etapa de tradução dos comandos e não são exibidas mensagens de erro, mas o programa *wsql* envia a mesma entrada para ser executada no banco de dados Postgres. Caso o comando também seja inválido como SQL, então a biblioteca de conexão com o banco de dados lança uma exceção que é exibida ao usuário.

3.6.1 Considerações finais

Neste capítulo foi apresentado o compilador que traduz comandos da linguagem WED-SQL para a linguagem SQL, sendo exibidos detalhes de sua implementação.

Na primeira seção foi exibida a arquitetura de sua implementação, mostrando como os componentes do compilador são gerados. Os analisadores léxico e sintático são gerados pela ferramenta ANTLR a partir das regras da linguagem WED-SQL. O analisador semântico e o gerador de código SQL são implementados manualmente com auxílio da interface *listener* gerada por ANTLR.

As seções seguintes tratam respectivamente da implementação das etapas de análise léxica, análise sintática, análise semântica e geração de comandos SQL do compilador. Nas seções referentes às análises léxica e sintática foram mostradas e comentadas as regras léxicas e sintáticas da gramática que gera a linguagem WED-SQL. Na seção referente à análise semântica, foram listadas as verificações de consistência e foi exibida a implementação manual de uma das verificações. Na seção referente à geração de código SQL foi explicado como são escritos os comandos SQL a partir da estrutura de árvore sintática de uma cadeia de caracteres, mostrando um exemplo de implementação manual.

Por fim, foi explicado como são recuperados ou tratados os erros detectados pelo compilador no programa *wsql* e como esse comportamento foi implementado usando a biblioteca ANTLR.

De maneira similar, o próximo capítulo (Capítulo 4) apresenta a implementação de outro compilador, que traduz comandos da linguagem de modelagem WED-flow para a linguagem WED-SQL.

Capítulo 4

Compilador da linguagem de modelagem WED-flow

Este trabalho também introduz à implementação da abordagem WED-flow o uso da linguagem de modelagem WED-flow. Essa linguagem, oferece apenas comandos para descrever o modelo, não contendo os comandos de controle e monitoramento que são oferecidos pela linguagem WED-SQL. A linguagem de modelagem WED-flow oferece um nível mais alto de abstração do que a linguagem WED-SQL, oferecendo uma representação mais simples dos elementos de um modelo. Os comandos da linguagem WED-flow servem para definir os elementos de um modelo de acordo com as definições da abordagem, sem especificar detalhes de implementação como os valores padrão dos WED-attributes, as condições de parada e os tempos máximos de execução das WED-transitions. Assim, a linguagem WED-flow reduz a carga semântica da modelagem em comparação à linguagem WED-SQL, facilitando a representação dos elementos do modelo.

Como a linguagem WED-SQL é uma linguagem intermediária entre a notação formal de conjuntos do WED-flow e a linguagem SQL, então semanticamente faz sentido que a linguagem de modelagem WED-flow seja traduzida para linguagem WED-SQL. Essa tradução é viável, pois a linguagem WED-SQL contém os comandos de especificação equivalentes a todos os comandos oferecidos pela linguagem de modelagem WED-flow. Sendo assim, para evitar a manutenção de consistência entre trechos de código duplicados dos compiladores das duas linguagens, o compilador da linguagem WED-flow foi criado de forma que traduz os comandos da linguagem WED-flow para a linguagem WED-SQL. Assim como o compilador da linguagem WED-SQL, o compilador da linguagem WED-flow segue a arquitetura apresentada na Seção 2.3, fazendo uso da ferramenta ANTLR para gerar os analisadores léxico e sintático a partir de descrições formais da linguagem WED-flow e auxiliar as demais etapas de compilação.

Na implementação WED-flow, um programa chamado wf2wsql foi criado para receber arquivos com comandos escritos na linguagem WED-flow e traduzi-los para a linguagem WED-SQL, escrevendo os comandos resultantes em um arquivo de saída. O arquivo de saída gerado pode ser interpretado pelo programa wsq, que permite apenas comandos WED-SQL e SQL como entrada. Para realizar a tradução dos comandos, o programa wf2wsql fará uso do compilador que será apresentado neste capítulo.

Diferente do programa wsq, o programa wf2sql espera que sejam fornecidas entradas em uma só linguagem, a linguagem WED-flow. Por causa disso, não é necessário nenhum tratamento específico

de erros e será empregado o tratamento de erros padrão da biblioteca ANTLR. O tratamento de erros padrão consiste em exibir uma mensagem detalhada sobre o erro detectado e tentar recuperar a execução do restante da entrada caso seja possível.

Este capítulo apresenta a linguagem de modelagem WED-flow, detalhando os comandos oferecidos pela linguagem. Também são apresentados os componentes do compilador que realiza a tradução da linguagem WED-flow para a linguagem WED-SQL e são detalhadas suas etapas de compilação, mostrando como a ferramenta ANTLR gerou os analisadores léxico e sintático a partir da descrição formal da linguagem. Por fim, discute-se como foram implementadas manualmente as etapas de análise semântica e de tradução para a linguagem alvo WED-SQL.

4.1 A linguagem WED-flow

A linguagem WED-flow define conjuntos para cada tipo de estrutura que compõe um modelo WED-flow, permitindo listar em um único comando todos os elementos de um determinado tipo de estrutura em um modelo. A linguagem também permite a especificação mais detalhada de um elemento após a declaração do conjunto do qual ele faz parte.

Antes da declaração de qualquer conjunto, uma anotação deve ser usada para identificar o tipo de estrutura que está sendo declarada. Todos os elementos definidos por comandos WED-flow se referem ao tipo de estrutura indicado pela última anotação lida.

Além das cadeias específicas para cada anotação e de outros símbolos específicos, a linguagem WED-flow é composta por identificadores de elementos e nomes de conjuntos. Ambos identificadores e nomes de conjuntos devem ser composto de letras, números ou os caracteres “-” e “_”. Os nomes de conjuntos devem ser iniciados com letra maiúscula.

Os seguintes comandos compõem a linguagem WED-flow:

- Criação de WED-attributes: Esse comando inicia-se pela anotação “@WED-attributes”, que indica qual o tipo de estrutura que o conjunto definido em seguida se refere. Após a anotação, é definido um conjunto cujos elementos são os identificadores dos WED-attributes que farão parte do modelo WED-flow. O comando é equivalente a uma sequência de comandos WED-SQL de criação de WED-attributes, cada elemento do conjunto equivale a um comando WED-SQL de criação de um único WED-attribute, sem a configuração de valor pré-definido para os WED-attributes criados. Assim, enquanto a sintaxe da linguagem de modelagem WED-flow é enxuta, a linguagem WED-SQL oferece a especificação de detalhes para algumas estruturas, como é o caso da configuração de valores pré-definidos para WED-attributes.

```
1  @WED-attributes
2  <nome_conjunto> = {<id_1>, <id_2>, <id_3>, ... };
```

Código 4.1: Comando WED-flow de criação de WED-attributes.

- Criação de WED-conditions: De forma similar ao comando anterior, esse comando inicia-se com uma anotação “@WED-conditions” seguida do conjunto, cujos elementos representam os identificadores das WED-conditions do modelo. Para cada elemento do conjunto de WED-conditions deve haver uma linha definindo o predicado referente à WED-condition indicada pelo identificador, não sendo permitido que exista um identificador no conjunto sem definição

de predicado e nem uma definição de predicado para um identificador que não esteja no conjunto. De forma similar à linguagem WED-SQL, cada predicado é validado pelo próprio SGBDR.

```

1  @WED-conditions
2  <nome_conjunto> = {<id_1>, <id_2>, <id_3>, ... };
3  <id_1>: <predicado_1>;
4  <id_2>: <predicado_2>;
5  ...

```

Código 4.2: Comando WED-flow de criação de WED-conditions.

- Criação de WED-transitions: Esse comando consiste da anotação “@WED-transitions” seguida de um conjunto no qual cada elemento identifica uma WED-transition do modelo. O ambiente transacional assume que existirá uma implementação para cada WED-transition definida como um serviço WED-worker ou como um procedimento armazenado com o mesmo nome da WED-transition a que se refere. Assim como no caso da criação de WED-attributes, a linguagem WED-flow permite uma configuração limitada de WED-transitions comparada à configuração permitida pela linguagem WED-SQL, que permite definir um prazo temporal de parada ou uma condição de parada para a transação, além de permitir informar se a transação será implementada como um procedimento armazenado no SGBDR ao invés de um serviço WED-worker.

```

1  @WED-transitions
2  <nome_conjunto> = {<id_1>, <id_2>, <id_3>, ... };

```

Código 4.3: Comando WED-flow de criação de WED-transitions.

- Criação de WED-triggers: O comando que cria o conjunto de WED-triggers é composto pela anotação “@WED-triggers” seguida da definição de um conjunto cujos elementos identificam cada WED-trigger do modelo. Para cada identificador do conjunto deve existir uma linha definindo respectivamente quais os identificadores da WED-condition e da WED-transition que são associadas pela WED-trigger em questão. De forma similar à definição dos predicados das WED-conditions, não deve haver nenhum identificador no conjunto de WED-triggers sem uma linha posterior definindo a associação referente à WED-trigger e também não deve existir nenhuma atribuição de associação a um identificador que não esteja no conjunto de WED-triggers.

```

1  @WED-triggers
2  <nome_conjunto> = {<id_1>, <id_2>, <id_3>, ... };
3  <id_1>: (<id_cond_1>, <id_trans_1>);
4  <id_2>: (<id_cond_2>, <id_trans_2>);
5  ...

```

Código 4.4: Comando WED-flow de criação de WED-triggers.

- Criação de WED-flow: Esse comando se baseia na definição formal de um modelo WED-flow e é composto pela anotação “@WED-flow” e a definição de um conjunto cujo identificador representa o nome do modelo e os elementos do conjunto representam respectivamente o nome

do conjunto de WED-triggers, o identificador da WED-condition inicial e o identificador da WED-condition final do modelo.

```
1  @WED-flow
2  <id_modelo> = {<nome_conj_trigger>, <id_cond_ini>, <id_cond_fim>;}
```

Código 4.5: Comando WED-flow de criação de modelo WED-flow.

4.2 Análise Léxica

Os comandos da linguagem WED-flow são compostos por *tokens* que representam anotações, nomes de conjuntos, identificadores e outros símbolos. A seguir são exibidas algumas das regras léxicas usadas para identificar os *tokens* em uma sequência de caracteres fornecida como entrada.

```
1 ANNOTATION_WED_ATTRIBUTES : '@' [Ww] [Ee] [Dd] '-' [Aa] [Tt] [Tt] [Rr] [Ii] [Bb] [Uu] [Tt] [Ee] [Ss];
2 ANNOTATION_WED_CONDITIONS : '@' [Ww] [Ee] [Dd] '-' [Cc] [Oo] [Nn] [Dd] [Ii] [Tt] [Ii] [Oo] [Nn] [Ss]
  -> mode(WED_CONDITION);
3 ANNOTATION_WED_TRANSITIONS : '@' [Ww] [Ee] [Dd] '-' [Tt] [Rr] [Aa] [Nn] [Ss] [Ii] [Tt] [Ii] [Oo] [Nn]
  [Ss];
4 ANNOTATION_WED_TRIGGERS : '@' [Ww] [Ee] [Dd] '-' [Tt] [Rr] [Ii] [Gg] [Gg] [Ee] [Rr] [Ss];
5 ANNOTATION_WED_FLOW : '@' [Ww] [Ee] [Dd] '-' [Ff] [Ll] [Oo] [Ww];
6 SET_NAME : [A-Z] NON_SPECIAL*;
7 VALID_NAME : NON_SPECIAL+;
8 COLON : ':';
9 EQUALS : '=';
10 SEMICOLON : ';';
11 COMMA : ',';
12 OPEN_PARENTHESSES : '(';
13 CLOSE_PARENTHESSES : ')';
14 OPEN_CURLY_BRACKETS : '{';
15 CLOSE_CURLY_BRACKETS : '}';
16 LINE_COMMENT : ('/'|'-' ) .*? '\r'? '\n' -> skip;
17 fragment NON_SPECIAL : [a-zA-Z] | DIGIT | [\_];
18 fragment DIGIT : [0-9];
19 SPACE : ' ';
20 LINEBREAK : '\r'? '\n';
21 TAB : [\t]+ -> skip;
```

Código 4.6: Algumas das regras léxicas da linguagem WED-flow.

Entre as regras léxicas listadas acima estão as definições dos *tokens* das anotações, que, de maneira similar aos *tokens* de palavras reservadas da linguagem WED-SQL, são definidas a partir de padrões compostos por determinados caracteres em ordem específica, independentemente de estarem em caixa alta (de modo *case-insensitive*).

Seguindo as anotações, os *tokens* que representam nomes de conjuntos e identificadores são definidos a partir dos fragmentos “NON_SPECIAL” e “DIGIT”. Como nomes de conjuntos devem ser iniciados por uma letra maiúscula, a regra “SET_NAME” é escrita antes da regra “VALID_NAME”. Assim, a convenção de prioridade do analisador léxico garante que quando uma sequência de caracteres que satisfaz a regra “SET_NAME” não será associada ao *token* “VALID_NAME”, mesmo observando que toda entrada que satisfaz “SET_NAME” também satisfaz “VALID_NAME”.

Assim como na linguagem WED-SQL, há na linguagem WED-flow uma regra para comentários que faz uso do comando skip, oferecido pela ferramenta ANTLR, para ignorar linhas de entrada que se iniciem com “—” ou “//”.

É importante observar que na regra “ANNOTATION_WED_CONDITIONS” é utilizado o comando “mode”, que serve para alterar o modo léxico sob o qual o analisador opera. O analisador léxico gerado pela ferramenta ANTLR tem um atributo chamado modo léxico, de modo que apenas as regras que compõem o modo léxico atual do analisador são levadas em conta na análise léxica. Por padrão o modo léxico tem o valor “DEFAULT_MODE”, o que significa que o analisador utilizará apenas as regras definidas antes da definição explícita de outros modos léxicos. O comando usado na regra “ANNOTATION_WED_CONDITIONS” altera o modo léxico para “WED_CONDITION”, que é definido no seguinte trecho da descrição formal da linguagem:

```

1 mode WED_CONDITION;
2 WC_ANNOTATION_WED_ATTRIBUTES : '@' [Ww] [Ee] [Dd] '-' [Aa] [Tt] [Tt] [Rr] [Ii] [Bb] [Uu] [Tt] [
    Ee] [Ss] -> mode(DEFAULT_MODE);
3 WC_ANNOTATION_WED_TRANSITIONS : '@' [Ww] [Ee] [Dd] '-' [Tt] [Rr] [Aa] [Nn] [Ss] [Ii] [Tt] [Ii] [Oo]
    ] [Nn] [Ss] -> mode(DEFAULT_MODE);
4 WC_ANNOTATION_WED_TRIGGERS : '@' [Ww] [Ee] [Dd] '-' [Tt] [Rr] [Ii] [Gg] [Gg] [Ee] [Rr] [Ss] ->
    mode(DEFAULT_MODE);
5 WC_ANNOTATION_WED_FLOW : '@' [Ww] [Ee] [Dd] '-' [Ff] [Ll] [Oo] [Ww] -> mode(
    DEFAULT_MODE);
6 WC_SET_NAME : [A-Z] NON_SPECIAL*;
7 WC_VALID_NAME : NON_SPECIAL+;
8 WC_EQUALS : '=';
9 WC_SEMICOLON : ';';
10 WC_COMMA : ',';
11 WC_OPEN_PARENTHESSES : '(';
12 WC_CLOSE_PARENTHESSES : ')';
13 WC_OPEN_CURLY_BRACKETS : '{';
14 WC_CLOSE_CURLY_BRACKETS : '}';
15 WC_LINE_COMMENT : ('//'|'—') .*? '\r'? '\n' -> skip;
16 fragment WC_NON_SPECIAL : [a-zA-Z] | DIGIT | [\ -_];
17 fragment WC_DIGIT : [0-9];
18 WC_SPACE : ' ';
19 WC_LINEBREAK : '\r'? '\n';
20 WC_TAB : [\t]+->skip;
21 PREDICATE : ':' .+? SEMICOLON;

```

Código 4.7: Regras léxicas da linguagem WED-flow que compõem o modo léxico WED_CONDITION.

O modo léxico “WED_CONDITION” é definido na linha 1 e todas as regras que sucedem a definição do modo valem apenas nesse modo léxico. As regras léxicas definidas antes da definição do modo “WED_CONDITION” compõem o modo léxico padrão “DEFAULT_MODE”.

Algumas das regras do modo “WED_CONDITION” são duplicadas com nomes de *tokens* diferentes para evitar erros na criação do analisador. As regras das anotações referentes a tipos de estrutura diferentes de WED-conditions possuem um comando mode para voltar o estado do analisador ao modo padrão, indicando o fim da configuração de WED-conditions.

A motivação para usar um modo léxico separado para a configuração de WED-conditions é que a regra “PREDICATE” deve ser permissiva e portanto deve fazer uso de uma expressão não-gulosa, mas não deve ser satisfeita quando a entrada recebida for referente à definição de outros tipos de

estruturas, mais especificamente WED-triggers, cuja definição possui uma sintaxe similar.

4.3 Análise Sintática

A gramática da linguagem WED-flow parte de um símbolo inicial “wflow”, que é derivado em uma lista de não-terminais definidos abaixo. Cada um dos não-terminais em questão representam os comandos mostrados na Seção 4.1.

```

1 wflow: stmt_list;
2 stmt_list: empty_space stmt empty_space stmt_list
3           | empty_space stmt empty_space;
4 stmt: create_flow_stmt empty_space SEMICOLON
5      | create_attr_stmt empty_space SEMICOLON
6      | create_cond_stmt empty_space
7      | create_trans_stmt empty_space SEMICOLON
8      | create_trig_stmt empty_space;
```

Código 4.8: Regras sintáticas da linguagem WED-flow que derivam não-terminais para cada comando a partir do símbolo inicial da gramática.

Os não-terminais de cada comando são derivados em função de outros não-terminais intermediários e dos *tokens* gerados pela análise léxica de acordo com as seguintes regras sintáticas:

```

1 create_flow_stmt: annotation_wed_flow SPACE* LINEBREAK empty_space valid_name empty_space
2                  EQUALS empty_space wed_flow_set_expr;
3 create_attr_stmt: annotation_wed_attributes SPACE* LINEBREAK empty_space SET_NAME
4                  empty_space EQUALS empty_space set_expr;
5 create_cond_stmt: ANNOTATION_WED_CONDITIONS WC_SPACE* WC_LINEBREAK
6                  wc_empty_space WC_SET_NAME wc_empty_space WC_EQUALS wc_empty_space wc_set_expr
7                  wc_empty_space WC_SEMICOLON wc_empty_space (condition_def wc_empty_space)*;
8 create_trans_stmt: annotation_wed_transitions SPACE* LINEBREAK empty_space SET_NAME
9                  empty_space EQUALS empty_space set_expr empty_space;
10 create_trig_stmt: annotation_wed_triggers SPACE* LINEBREAK empty_space SET_NAME
11                  empty_space EQUALS empty_space set_expr empty_space SEMICOLON empty_space (trigger_def
12                  empty_space)*;
13 annotation_wed_flow : (WC_ANNOTATION_WED_FLOW | ANNOTATION_WED_FLOW);
14 annotation_wed_attributes : (WC_ANNOTATION_WED_ATTRIBUTES |
15                             ANNOTATION_WED_ATTRIBUTES);
16 annotation_wed_transitions : (WC_ANNOTATION_WED_TRANSITIONS |
17                              ANNOTATION_WED_TRANSITIONS);
18 annotation_wed_triggers : (WC_ANNOTATION_WED_TRIGGERS |
19                            ANNOTATION_WED_TRIGGERS);
20 set_expr : OPEN_CURLY_BRACKETS empty_space ((valid_name empty_space COMMA empty_space)
21         * valid_name empty_space)? CLOSE_CURLY_BRACKETS;
22 wed_flow_set_expr : OPEN_CURLY_BRACKETS empty_space SET_NAME empty_space COMMA
23                   empty_space valid_name empty_space COMMA empty_space valid_name empty_space
24                   CLOSE_CURLY_BRACKETS;
25 trigger_def : valid_name empty_space COLON empty_space OPEN_PARENTHESSES empty_space
26              valid_name empty_space COMMA empty_space valid_name empty_space CLOSE_PARENTHESSES
27              empty_space SEMICOLON;
28 valid_name : (SET_NAME | VALID_NAME);
29 empty_space : (SPACE | LINEBREAK)*;
```

```

15 wc_set_expr : WC_OPEN_CURLY_BRACKETS wc_empty_space ((wc_valid_name wc_empty_space
    WC_COMMA wc_empty_space)* wc_valid_name wc_empty_space)?
    WC_CLOSE_CURLY_BRACKETS;
16 condition_def : wc_valid_name wc_empty_space PREDICATE;
17 wc_valid_name : (WC_SET_NAME | WC_VALID_NAME);
18 wc_empty_space : (WC_SPACE | WC_LINEBREAK)*;

```

Código 4.9: Demais regras sintáticas da linguagem WED-flow.

Para representar as anotações que compõem os comandos WED-flow, foram criadas regras sintáticas de forma que possam ser satisfeitas pelos *tokens* que representam as anotações em ambos os modos léxicos, com exceção da anotação de WED-condition, que só deve ser detectada no modo léxico padrão.

O não-terminal referente ao comando de criação de WED-conditions e os não-terminais usados em sua derivação são compostos por *tokens* do modo léxico “WED_CONDITION”, com exceção do não-terminal de anotação, já que o *token* de anotação de WED-condition altera o modo léxico para o modo “WED_CONDITION”.

Identificadores de estruturas são representados pelo não-terminal “valid_name”, satisfeito pelos *tokens* “SET_NAME” ou “VALID_NAME”. Assim, os identificadores podem se iniciar com letras maiúsculas ou minúsculas.

4.4 Análise Semântica

A etapa de análise semântica para a linguagem WED-flow visa detectar erros de consistência na definição do modelo. De modo similar à análise semântica do compilador da linguagem WED-SQL, essa etapa deve incluir a verificação da existência das WED-conditions e WED-transitions usadas na configuração de cada WED-trigger.

Na linguagem WED-flow a condição que define os estados finais do modelo é fornecida no comando de criação do WED-flow como um identificador de uma condição que deve já ter sido definida no comando de criação das WED-conditions. Sendo assim, nessa etapa também é realizada a verificação da existência de condição final do modelo.

Pelo fato de alguns dos comandos da linguagem WED-flow serem compostos pela declaração de conjuntos de elementos do modelo e por linhas de configurações específicas para cada elemento do conjunto, é necessário verificar a consistência entre os elementos do conjunto e os elementos configurados para impedir a configuração de uma estrutura não declarada, ou a existência de um elemento não configurado. Esse tipo de verificação é realizado sobre WED-conditions e WED-triggers, que são os tipos de estruturas que permitem essa configuração.

O analisador semântico do compilador da linguagem WED-flow foi implementado como uma classe Python com métodos que realizam as verificações semânticas descritas nesta seção. Os métodos do analisador sintático são solicitados pelo programa wf2wsq, que percorre a árvore sintática gerada para a entrada após a análise sintática e usa a interface *listener* para acionar os métodos de verificação. As verificações de consistência entre definição de conjunto e configurações na criação de WED-conditions e WED-triggers são acionadas quando o programa encontra na árvore sintática os não-terminais referentes aos comandos de criação de WED-conditions e WED-triggers, respectivamente. As demais verificações descritas nesta seção são acionadas no comando de criação de WED-flow, que deve ser o último comando de um modelo escrito na linguagem WED-flow.

Se algum erro é detectado pelas verificações no programa wf2wsql, então uma mensagem de erro é exibida ao usuário e o programa interrompe a execução. Caso contrário, o wf2wsql continua sua execução.

A seguir é exibido um dos métodos da etapa de verificação semântica do compilador da linguagem WED-flow, o método Python responsável pela verificação de consistência entre os elementos do conjunto de triggers e as linhas de configuração das triggers:

```

1 def check_trigger_set(self, ctx):
2     trigger_name_list = ctx.set_expr().valid_name()
3     self.trigger_set = [t.getText() for t in trigger_name_list]
4     defined_triggers = []
5     trigger_definition_list = ctx.trigger_def()
6     self.trigger_set_name = ctx.SET_NAME().getText()
7     error = ""
8     for trigger_definition in trigger_definition_list:
9         trigger_name = trigger_definition.valid_name()[0].getText().strip()
10        condition_name = trigger_definition.valid_name()[1].getText().strip()
11        transition_name = trigger_definition.valid_name()[2].getText().strip()
12        defined_triggers.append(trigger_name)
13        self.trigger_def[trigger_name] = {"condition" : condition_name, "transition" : transition_name}
14        if trigger_name not in self.trigger_set:
15            error += "Error: Trigger '{ }' not declared in the trigger set.\n".format(trigger_name)
16    for trigger in self.trigger_set:
17        if trigger not in defined_triggers:
18            error += "Error: Trigger '{ }' declared but not defined.\n".format(trigger)
19    return error

```

Código 4.10: *Exemplo de verificação semântica do compilador de linguagem WED-flow.*

O método exibido acima é acionado pelo programa wf2wsql quando, ao percorrer a árvore sintática gerada pelo analisador sintático para uma entrada, encontra o não-terminal “create_trig_stmt” referente ao comando de criação de WED-triggers. O método retorna uma mensagem de erro caso a verificação falhe e retorna uma cadeia vazia caso contrário. Além disso, o método recebe como entrada o objeto ctx, que representa a composição da entrada em termos de não-terminais e *tokens* após a derivação do não-terminal “create_trig_stmt”. O objeto ctx é usado para facilitar a identificação de cada componente da entrada.

A linha 2 busca no objeto ctx todos os identificadores que estão no conjunto da entrada e a linha 3 armazena na lista “trigger_set” os elementos encontrados no conjunto de triggers. A linha 5 armazena na lista “trigger_definition_list” todas as linhas de configurações de triggers.

O laço da linha 8 percorre todas as configurações de triggers. Dentro dele, as linhas 9 a 11 capturam os identificadores da WED-trigger, WED-condition e WED-transition que compõem a configuração, respectivamente. Nas linhas restantes do laço, o identificador da WED-trigger é inserido na lista “defined_triggers”, os demais dados da configuração são armazenados no atributo “trigger_def” do objeto de verificação semântica para que possam ser usados futuramente. Também é verificado se o identificador da WED-trigger a ser configurada existe no conjunto de WED-triggers, caso não exista, uma mensagem de erro é escrita para o retorno.

O laço da linha 16 itera sobre a lista de identificadores encontrados no conjunto de WED-trigger e verifica se existe configuração para cada um dos identificadores. Caso não exista para algum identificador, uma nova mensagem de erro é adicionada no retorno. Por fim, o método

retorna as possíveis mensagens de erro, ou uma cadeia vazia caso nenhuma inconsistência tenha sido encontrada.

4.5 Tradução para WED-SQL

Cada comando de criação da linguagem WED-flow define um conjunto de elementos do modelo. A etapa de tradução para a linguagem WED-SQL gera um comando para cada elemento de cada conjunto definido na entrada. Para os elementos de configuração obrigatória como WED-conditions e WED-triggers são adicionadas as configurações nos comandos WED-SQL resultantes. Os demais elementos, WED-attributes e WED-transitions, são declarados sem configurações opcionais. Caso seja necessária alguma configuração adicional em WED-attributes e WED-transitions, ela deve ser adicionada posteriormente no código resultante WED-SQL.

O comando de criação do modelo na linguagem WED-flow não só define o nome do modelo como indica sua condição final. Portanto, são gerados dois comandos na linguagem WED-SQL a partir da definição do modelo na linguagem WED-flow: O comando de configuração dos estados finais e o comando de criação de modelo.

A etapa de tradução para linguagem WED-SQL gera o comando de configuração de estados finais usando o predicado da condição final, que deve ser fornecido previamente no comando de criação de condições.

O comando WED-SQL de criação de WED-flow é gerado de forma que o modelo criado tem o mesmo nome que o identificador do conjunto fornecido no comando da linguagem WED-flow de criação de modelo WED-flow.

Ao final da etapa de tradução, os comandos WED-SQL são reordenados pelo programa wf2sql para que sejam escritos em um arquivo de saída. A ordem na qual os comandos WED-SQL são escritos na saída leva em conta a execução dos comandos em um banco de dados:

- Comando que cria o modelo WED-flow.
- Comando que se conecta ao modelo criado.
- Comando de início de transação (*begin*).
- Comandos de criação de WED-attributes, WED-conditions, WED-transitions e WED-triggers.
- Comando de configuração dos estados finais.
- Comando que confirma a transação (*commit*).

Para exemplificar a tradução dos comandos da linguagem WED-flow para a linguagem WED-SQL, consideremos o modelo de exemplo a seguir:

```
1 @WED-attributes
2 A = {a1, a2, a3};
3
4 @WED-conditions
5 Cond = {c1, c2};
6 c1 : a1 = true;
7 c2 : a2 = false AND a3 > 5;
8
```

```

9 @WED-transitions
10 Trans = {t1, t2};
11
12 @WED-triggers
13 TRG = {tg1, tg2};
14 tg1 : (c1, t1);
15 tg2 : (c2, t2);
16
17 @WED-flow
18 Modelo_teste = {TRG, c1, c2};

```

Código 4.11: *Exemplo de modelo descrito na linguagem WED-flow.*

Supondo o modelo do exemplo como a entrada do compilador, no final da etapa de tradução, o programa wf2sql gerará um arquivo de saída com o seguinte código WED-SQL:

```

1 create flow Modelo_teste;
2 \cw Modelo_teste;
3 begin;
4
5 create wed-attribute a1;
6 create wed-attribute a2;
7 create wed-attribute a3;
8
9 create wed-condition c1 as a1 = true;
10 create wed-condition c2 as a2 = false AND a3 > 5;
11
12 create wed-transition t1;
13 create wed-transition t2;
14
15 create wed-trigger tg1 as (c1, t1);
16 create wed-trigger tg2 as (c2, t2);
17
18 set final wed-state as a2 = false AND a3 > 5;
19
20 commit;

```

Código 4.12: *Tradução para linguagem WED-SQL do modelo de exemplo.*

O código WED-SQL gerado no arquivo de saída pode ser alterado manualmente pelo projetista do modelo para adicionar detalhes de implementação como valores padrão nos WED-attributes, intervalos temporais máximos de execução e condições de parada nas WED-transitions. No entanto, caso seja necessário executar o programa wf2sql para gerar o código WED-SQL novamente, o arquivo de saída será sobrescrito. Assim, é recomendável fazer uma cópia do arquivo de saída caso sejam necessárias adições de detalhes de implementação.

4.5.1 Considerações finais

Neste capítulo foram apresentadas a linguagem WED-flow e a implementação do compilador que traduz os comandos da linguagem WED-flow para a linguagem WED-SQL.

Inicialmente o capítulo introduz a linguagem WED-flow, diferenciando-a da linguagem WED-SQL e explicando como a implementação da abordagem irá oferecer o uso da linguagem WED-flow

para o usuário. Em seguida, a primeira seção exibiu a sintaxe dos comandos da linguagem WED-flow.

De forma similar ao capítulo anterior (Capítulo 3), as seções seguintes trataram respectivamente da implementação das etapas de análise léxica, análise sintática, análise semântica e geração de código em linguagem WED-SQL do compilador. Nas seções referentes às análises léxica e sintática foram mostradas e comentadas as regras léxicas e sintáticas da gramática que gera a linguagem WED-flow. Na seção referente à análise semântica, foram listadas as verificações de consistência e foi exibida a implementação manual de uma das verificações. Na seção referente à geração de código WED-SQL mostrou como os comandos WED-SQL resultantes são escritos e ordenados a partir da estrutura de árvore sintática de uma cadeia de caracteres, mostrando um exemplo de implementação manual.

O próximo capítulo (Capítulo 5) introduzirá um sistema e usará os compiladores implementados neste trabalho para modelar e instanciar os processos desse sistema, visualizando as estruturas criadas e as instâncias dos modelos.

Capítulo 5

Estudo de caso

Este capítulo apresenta um estudo de caso realizado sobre a modelagem WED-flow do sistema SISAUT, apresentado em [FTK⁺17]. Esse sistema possui três processos de negócio, os quais serão modelados seguindo a abordagem WED-flow. Os modelos serão implementados no ambiente transacional WED-flow e exemplos de instâncias serão gerados para a análise de sua execução nos modelos.

O modelo de cada processo será descrito na linguagem WED-flow e posteriormente será fornecido como entrada do compilador de linguagem WED-flow para gerar o código WED-SQL equivalente ao modelo. Alguns detalhes na definição de modelos estão além das limitações impostas pela linguagem WED-flow, portanto, devem ser adicionados manualmente no código WED-SQL referente ao modelo.

Quando o modelo escrito em WED-SQL estiver pronto, será fornecido como entrada do programa `wsql`, que, fazendo uso do compilador de linguagem WED-SQL, gera código SQL para executar no ambiente transacional da implementação WED-flow, criando um banco de dados que contém o modelo em questão.

O programa `wsql` permite que sejam fornecidos como entrada comandos WED-SQL de criação e monitoramento de instâncias do processo modelado, gerando e executando o código SQL resultante no ambiente transacional para que seja possível a análise do funcionamento do processo WED-flow em execução. Além disso, o programa `wsql` também permite que consultas SQL sejam executadas no banco de dados que contém o modelo, o que também pode ser útil para a análise das instâncias.

5.1 O sistema SISAUT

A Universidade de São Paulo oferece um serviço público de realização de autópsias na cidade de São Paulo, o SVOC (Serviço de Verificação de Óbitos da Capital), responsável por determinar a causa da morte que deve constar em atestados de óbito. Além disso, o SVOC também permite que os órgãos do falecido sejam doados para projetos de pesquisa científica caso haja consentimento da família do falecido. O sistema SISAUT foi criado para gerenciar os processos envolvidos na doação de órgãos realizada por intermédio do SVOC para fins de pesquisa.

O SISAUT é composto pelos seguintes processos:

- Entrevista com familiares do falecido: Obtém informações sobre o falecido, o consentimento e as preferências dos familiares quanto a seleção de quais projetos de pesquisa receberão os

materiais doados.

- Coleta de materiais do corpo: Realiza os procedimentos necessários para coletar materiais (tecidos ou órgãos) do corpo de acordo com o consentimento dos familiares obtido no processo de entrevista e com a elegibilidade dos materiais para pesquisa. Ao fim do processo de coleta, é iniciado um novo processo para cada material cuja coleta for realizada com sucesso, a etapa de processamento do material.
- Processamento de materiais: Realiza procedimentos sobre os materiais coletados, preservando os materiais próprios para pesquisa e descartando de forma correta os materiais impróprios para pesquisa.

Quando um corpo chega ao SVOC, acompanhado dos familiares do falecido, são obtidas informações básicas de identificação, como nome e números de documentos de identidade, e também são obtidos dados demográficos, como idade, gênero, etnia e cor de pele. Os dados demográficos são usados para decidir se há projetos de pesquisa que têm interesse na doação de materiais do corpo em questão. Caso haja interesse de algum projeto de pesquisa, o sistema SISAUT é ativado, iniciando os processos de entrevista dos familiares do falecido e de coleta dos materiais do corpo.

5.1.1 Entrevista

Os familiares do falecido são recebidos no SVOC por uma equipe de profissionais que os entrevistam para obter as informações pertinentes aos projetos de pesquisa e os devidos documentos legais de consentimento de doação necessários para cada projeto. O processo de entrevista é composto pelas seguintes etapas:

- Obtenção de informações para seleção de projetos: Nesta etapa a equipe de entrevistadores realiza perguntas aos familiares com a finalidade de obter informações para decidir quais são os projetos elegíveis para receber os materiais doados pelo falecido.
- Escolha de projetos: Os dados obtidos na etapa anterior, junto às informações demográficas obtidas previamente, são usados para determinar um conjunto de projetos elegíveis para receber a doação de materiais de acordo com as especificações de cada projeto.
- Aplicação de consentimento: O sistema SISAUT gera os formulários de consentimento referentes aos projetos elegíveis resultantes da etapa anterior. A equipe de entrevista auxilia os familiares do falecido no preenchimento dos formulários.
- Seleção de projetos: Nesta etapa são analisados os formulários de consentimento preenchidos para remover do conjunto de projetos elegíveis os projetos que não possuem consentimento dos familiares. Ao fim desta etapa, a coleta de materiais para os projetos elegíveis consentidos pode ser iniciada.
- Aplicação de questionários: Alguns projetos de pesquisa podem necessitar do preenchimento de questionários pelos familiares do falecido. Nesta etapa, são preenchidos os questionários dos projetos elegíveis consentidos caso alguns dos projetos requeiram questionários.

5.1.2 Coleta de materiais

Quando um corpo chega ao SVOC para a realização da autópsia, sua elegibilidade para doação de materiais para um dos projetos de pesquisa é verificada por um membro da equipe de coleta, que é composta de patologistas e técnicos. Quando um corpo é considerado elegível para doação de materiais, é iniciado o processo de coleta, que é composto pelas seguintes etapas:

- Registro de medidas: Um membro da equipe de coleta deve realizar medições sobre o corpo e registrar as medidas obtidas.
- Coleta dos materiais: Após o registro das medidas do corpo e a obtenção do consentimento para doação para projetos, são coletados os materiais. A equipe de coleta obtém os materiais necessários para cada projeto consentido e registra o estado de cada material, assim como informações pertinentes para cada pesquisa. Caso um material não possa ser coletado por algum motivo, um membro da equipe deve registrar uma justificativa no sistema.
- Checagem de coleta de materiais: Nesta etapa, o SISAUT verifica a lista de materiais necessários para cada um dos projetos elegíveis consentidos e checa se eles foram devidamente coletados. Caso algum material de um projeto não tenha sido coletado, a doação do material para o projeto é abortada, o que pode causar o cancelamento da etapa de aplicação de questionários do processo de entrevista para os questionários referentes ao projeto cuja doação foi abortada, caso o questionário já não tenha sido aplicado. Após realizar a checagem e abortar doações de materiais que não foram devidamente coletados, é iniciado o processo de processamento do material para cada material devidamente coletado.

5.1.3 Processamento de materiais

Após o processo de coleta, cada material resultante da coleta é enviado a um laboratório especializado e para cada material se inicia o processamento de materiais. Esse processo é composto pelas etapas:

- Verificação da condição do material: O processamento de um material inicia-se avaliando a condição do material coletado por um profissional especializado. Caso a condição do material esteja aceitável, são realizados sobre o material os procedimentos de inclusão. Caso contrário, são realizados os procedimentos de exclusão. Quando uma doação para um projeto for abortada pelo processo de coleta, então os materiais referentes à tal doação são enviados aos procedimentos de exclusão, independente de suas condições.
- Procedimentos de inclusão: Caso o material apresente condição aceitável, são realizados sobre o material os procedimentos de inclusão, procedimentos físicos que visam preservar o material para pesquisa científica.
- Procedimentos de exclusão: Caso a condição do material não seja aceitável, o material sofre os procedimentos de exclusão para que seja descartado de forma segura.

5.2 Modelagem WED-flow do sistema SISAUT

O sistema SISAUT é composto por processos cujas etapas são realizadas por diferentes atores e algumas das etapas dependem da realização de outras etapas do mesmo processo e, em alguns casos, de etapas de outros processos. Para garantir a consistência e sincronismo dos processos envolvidos, o sistema SISAUT é implementado usando a abordagem WED-flow.

Ambos os processos de entrevista e coleta de materiais são instanciados para cada autópsia realizada pela SVOC. Já o processo de processamento de materiais é realizado para cada um dos materiais coletados de cada corpo. Por causa da diferença de granularidade entre os processos, dois modelos foram criados, um deles representa os processos de entrevista e coleta de materiais, o outro representa o processo de processamento de materiais.

O modelo contendo os processos de entrevista e coleta expresso na linguagem WED-flow 4.1 é apresentado a seguir:

```

1 @wed-attributes
2 A = {body_id, dados_necessarios, projetos_escolhidos, termos_consentimento, projetos_consentidos,
      coleta_aprovada,
3 questionarios_aplicados, medidas_obtidas, materiais_coletados, materiais_descartados, projetos_abortados
      };
4
5 @wed-conditions
6 C = {condicao_inicial, iniciou_entrevista, coletou_dados, escolheu_projetos, aplicou_consentimentos,
      selecionou_projetos, iniciou_coleta,
7 pode_coletar_materiais, coletou_materiais, condicao_final};
8
9 condicao_inicial : body_id is not null and dados_necessarios is null and medidas_obtidas is null;
10 iniciou_entrevista : body_id is not null and dados_necessarios is null;
11 coletou_dados : dados_necessarios::bool and projetos_escolhidos is null;
12 escolheu_projetos : projetos_escolhidos is not null and termos_consentimento is null;
13 aplicou_consentimentos : termos_consentimento::bool and projetos_consentidos is null;
14 selecionou_projetos : projetos_consentidos is not null and questionarios_aplicados is null;
15 iniciou_coleta : body_id is not null and medidas_obtidas is null;
16 pode_coletar_materiais : medidas_obtidas::bool and coleta_aprovada::bool and materiais_coletados is
      null and materiais_descartados is null;
17 coletou_materiais : (materiais_coletados is not null or materiais_descartados is not null) and
      projetos_abortados is null;
18 condicao_final : coleta_aprovada is not null and questionarios_aplicados is not null and ((coleta_aprovada
      ::bool and projetos_abortados is not null) or coleta_aprovada::bool = false);
19
20 @wed-transitions
21 Trn = {coletar_dados, escolher_projetos, aplicar_consentimentos, selecionar_projetos, aplicar_questionarios
      , registrar_medidas, coletar_materiais, abortar_projetos};
22
23 @wed-triggers
24 Trg = {tg0, tg1, tg2, tg3, tg4, tg5, tg6, tg7};
25
26 tg0 : (iniciou_entrevista, coletar_dados);
27 tg1 : (coletou_dados, escolher_projetos);
28 tg2 : (escolheu_projetos, aplicar_consentimentos);
29 tg3 : (aplicou_consentimentos, selecionar_projetos);
30 tg4 : (selecionou_projetos, aplicar_questionarios);
31 tg5 : (iniciou_coleta, registrar_medidas);

```



```

32 tg6 : (pode_coletar_materiais, coletar_materiais);
33 tg7 : (coletou_materiais, abortar_projetos);
34
35 @WED-flow
36 coleta_entrevista = {Trg, condicao_inicial, condicao_final};

```

Código 5.1: *Modelagem WED-flow dos processos de entrevista e coleta do SISAUT.*

No modelo acima, inicialmente são definidos os WED-attributes, consistindo em:

- `body_id`: Identificador para o corpo ao qual uma instância do modelo se refere.
- `dados_necessarios`: Atributo booleano que indica se já foram obtidas as informações para seleção de projetos no processo de entrevista.
- `projetos_escolhidos`: Lista de projetos selecionados na etapa de escolha de projetos no processo de entrevista.
- `termos_consentimento`: Atributo booleano que indica se já foi realizada a etapa de aplicação de consentimento do processo de entrevista.
- `projetos_consentidos`: Lista de projetos selecionados de acordo com os formulários de consentimentos preenchidos. Este atributo é resultante da etapa de seleção de projetos do processo de entrevista.
- `coleta_aprovada`: Atributo booleano que indica se a coleta de materiais foi aprovada pelos familiares do falecido na etapa de seleção de projetos do processo de entrevista.
- `questionarios_aplicados`: Lista de identificadores dos questionários que precisaram ser preenchidos pelos familiares na etapa de aplicação de questionários do processo de entrevista.
- `medidas_obtidas`: Atributo booleano que indica se já foi realizada a etapa de registro de medidas do corpo no processo de coleta.
- `materiais_coletados`: Lista de identificadores dos materiais que foram coletados com êxito na etapa de coleta de materiais do processo de coleta.
- `materiais_descartados`: Lista de identificadores dos materiais que não puderam ser devidamente coletados na etapa de coleta de materiais do processo de coleta.
- `projetos_abortados`: Lista de identificadores dos projetos cuja doação do material foi abortada na etapa de checagem de coleta de materiais do processo de coleta.

As WED-conditions são declaradas e expressas como predicados sobre os WED-attributes definidos acima. A maioria das condições verificam se uma etapa foi realizada para que possa se dar início à etapa subsequente, no caso das etapas sequenciais.

É importante observar que um estado de dados que satisfaça a WED-condition “`condicao_inicial`” do modelo também irá satisfazer as WED-conditions “`iniciou_entrevista`” e “`iniciou_coleta`”, que iniciam respectivamente os processos de entrevista e coleta. Assim, os processos de entrevista e coleta são executados paralelamente.

Quando o processo de coleta termina a etapa de registro de medidas do corpo, a etapa de coleta de materiais só pode ser iniciada quando for recebido o consentimento provindo do processo de entrevista. A WED-condition “pode_coletar_materiais” além de verificar se a etapa anterior do processo de coleta terminou, também verifica se a coleta foi aprovada pelo processo de entrevista, implementando a dependência entre os dois processos. Portanto, a WED-transition responsável pela implementação da etapa de coleta de materiais só será iniciada quando o estado de dados satisfazer a WED-condition “pode_coletar_materiais”.

Em seguida são declaradas as WED-transitions, que serão definidas em Python como serviços WED-worker. Neste experimento foram escritas WED-transitions que simulam o funcionamento do sistema SISAUT. Então são definidas as WED-triggers, associando as WED-conditions às WED-transitions que devem ser executadas quando o estado de dados do modelo satisfazer a WED-condition associada.

Ao final do processo de coleta de materiais, ainda na etapa “abortar_projetos”, para cada um dos materiais cuja doação não foi abortada, é criada uma instância do processo de processamento de materiais. Na implementação da WED-transition é estabelecida uma conexão com o banco de dados referente ao modelo de processamento e são inseridas as linhas referentes às novas instâncias. O modelo WED-flow do processo de processamento de materiais é exibido a seguir:

```

1 @wed-attributes
2 A = {body_id, material_coletado, material_bom_estado, procedimento_inclusao, procedimento_exclusao};
3
4 @wed-conditions
5 C = {inicio_processamento, aprovou_material, reprovou_material, condicao_final};
6
7 inicio_processamento : body_id is not null and material_coletado <> " " and material_bom_estado is null;
8 aprovou_material : material_bom_estado::bool and procedimento_inclusao is null;
9 reprovou_material : material_bom_estado::bool = false and procedimento_exclusao is null;
10 condicao_final : material_bom_estado is not null and (procedimento_inclusao is not null or
    procedimento_exclusao is not null);
11
12 @wed-transitions
13 Trn = {verificar_material, realizar_proc_inclusao, realizar_proc_exclusao};
14
15 @wed-triggers
16 Trg = {tg0, tg1, tg2};
17
18 tg0 : (inicio_processamento, verificar_material);
19 tg1 : (aprovou_material, realizar_proc_inclusao);
20 tg2 : (reprovou_material, realizar_proc_exclusao);
21
22 @WED-flow
23 processamento = {Trg, inicio_processamento, condicao_final};

```

Código 5.2: Modelagem WED-flow do processo de processamento de materiais do SISAUT.

A modelagem acima se inicia com a definição dos WED-attributes, a identificação do corpo (atributo “body_id”), a identificação do material a ser processado (atributo “material_coletado”), o estado do material (atributo “material_bom_estado”) e os atributos “procedimento_inclusao” e “procedimento_exclusao”, que indicam respectivamente se o material foi submetido aos procedimentos de inclusão ou de exclusão.

As WED-conditions representam os possíveis estados de cada etapa do processo. O processo se inicia quando existe uma instância com os atributos de identificação do corpo e o material coletado preenchidos, mas sem um valor para o atributo de estado do material. Por causa da WED-trigger “tg0”, quando o estado de uma instância satisfazer a WED-condition “inicio_processamento”, o sistema dispara a execução da WED-transition “verificar_material”, que deve preencher o atributo de estado do material coletado. Quando esse atributo é preenchido, o estado da instância irá satisfazer uma das WED-conditions “aprovou_material” ou “reprovou_material”, já que ainda não foram preenchidos os atributos “procedimento_inclusao” e “procedimento_exclusao”.

De forma similar à etapa anterior, as WED-triggers “tg1” e “tg2” implicam respectivamente na execução das WED-transitions “realizar_proc_inclusao” e “realizar_proc_exclusao” quando o estado da instância satisfazer as WED-conditions indicadas em cada WED-trigger. Após a execução de uma das WED-transitions “realizar_proc_inclusao” ou “realizar_proc_exclusao”, será preenchido o WED-attribute referente ao término da execução da etapa de realização de procedimentos de inclusão ou de exclusão, dependendo da transição executada. Qualquer um dos dois atributos irá fazer com que o estado da instância satisfaça a condição final do modelo, terminando a execução da instância.

5.2.1 Comandos WED-SQL

Após definidos os modelos WED-flow dos processos do SISAUT, será usado o programa “wf2wsq1” para traduzir os comandos escritos na linguagem WED-flow para a linguagem WED-SQL. Esse passo é necessário para que os comandos de criação dos modelos sejam posteriormente fornecidos como entrada ao programa “wsq1” e também caso seja necessário adicionar algum detalhe de implementação na especificação do modelo que não tenha sido possível adicionar na modelagem WED-flow devido às limitações da linguagem, como por exemplo, valores pré-definidos para atributos. No caso dos modelos do SISAUT não será necessário adicionar nenhum detalhe no código WED-SQL.

A seguir são exibidos os comandos WED-SQL que representam a especificação dos processos do SISAUT de coleta de materiais e de entrevista com familiares do falecido resultante do programa “wf2wsq1”, que utiliza o compilador de linguagem de modelagem WED-flow para linguagem WED-SQL introduzido neste trabalho.

```

1 create flow coleta_entrevista;
2 \cw coleta_entrevista;
3 begin;
4
5 create wed-attribute body_id;
6 create wed-attribute dados_necessarios;
7 create wed-attribute projetos_escolhidos;
8 create wed-attribute termos_consentimento;
9 create wed-attribute projetos_consentidos;
10 create wed-attribute coleta_aprovada;
11 create wed-attribute questionarios_aplicados;
12 create wed-attribute medidas_obtidas;
13 create wed-attribute materiais_coletados;
14 create wed-attribute materiais_descartados;
15 create wed-attribute projetos_abortados;
16

```

```

17 create wed-condition condicao_inicial as body_id is not null and dados_necessarios is null and
    medidas_obtidas is null;
18 create wed-condition iniciou_entrevista as body_id is not null and dados_necessarios is null;
19 create wed-condition coletou_dados as dados_necessarios::bool and projetos_escolhidos is null;
20 create wed-condition escolheu_projetos as projetos_escolhidos is not null and termos_consentimento is
    null;
21 create wed-condition aplicou_consentimentos as termos_consentimento::bool and projetos_consentidos is
    null;
22 create wed-condition selecionou_projetos as projetos_consentidos is not null and questionarios_aplicados
    is null;
23 create wed-condition iniciou_coleta as body_id is not null and medidas_obtidas is null;
24 create wed-condition pode_coletar_materiais as medidas_obtidas::bool and coleta_aprovada::bool and
    materiais_coletados is null and materiais_descartados is null;
25 create wed-condition coletou_materiais as (materiais_coletados is not null or materiais_descartados is not
    null) and projetos_abortados is null;
26
27 create wed-transition coletar_dados;
28 create wed-transition escolher_projetos;
29 create wed-transition aplicar_consentimentos;
30 create wed-transition selecionar_projetos;
31 create wed-transition aplicar_questionarios;
32 create wed-transition registrar_medidas;
33 create wed-transition coletar_materiais;
34 create wed-transition abortar_projetos;
35
36 create wed-trigger tg0 as (iniciou_entrevista, coletar_dados);
37 create wed-trigger tg1 as (coletou_dados, escolher_projetos);
38 create wed-trigger tg2 as (escolheu_projetos, aplicar_consentimentos);
39 create wed-trigger tg3 as (aplicou_consentimentos, selecionar_projetos);
40 create wed-trigger tg4 as (selecionou_projetos, aplicar_questionarios);
41 create wed-trigger tg5 as (iniciou_coleta, registrar_medidas);
42 create wed-trigger tg6 as (pode_coletar_materiais, coletar_materiais);
43 create wed-trigger tg7 as (coletou_materiais, abortar_projetos);
44
45 set final wed-state as coleta_aprovada is not null and questionarios_aplicados is not null and ((
    coleta_aprovada::bool and projetos_abortados is not null) or coleta_aprovada::bool = false);
46
47 commit;

```

Código 5.3: Especificação WED-SQL dos processos de entrevista e coleta do SISAUT.

De forma similar, segue abaixo a especificação do processo de processamento de materiais do SISAUT, também resultante do uso do compilador de linguagem WED-flow para linguagem WED-SQL pelo programa “wf2wsq1”.

```

1 create flow processamento;
2 \cw processamento;
3 begin;
4
5 create wed-attribute body_id;
6 create wed-attribute material_coletado;
7 create wed-attribute material_bom_estado;
8 create wed-attribute procedimento_inclusao;
9 create wed-attribute procedimento_exclusao;

```

```

10
11 create wed-condition inicio_processamento as body_id is not null and material_coletado <> '' and
    material_bom_estado is null;
12 create wed-condition aprovou_material as material_bom_estado::bool and procedimento_inclusao is null;
13 create wed-condition reprovou_material as material_bom_estado::bool = false and procedimento_exclusao
    is null;
14
15 create wed-transition verificar_material;
16 create wed-transition realizar_proc_inclusao;
17 create wed-transition realizar_proc_exclusao;
18
19 create wed-trigger tg0 as (inicio_processamento, verificar_material);
20 create wed-trigger tg1 as (aprovou_material, realizar_proc_inclusao);
21 create wed-trigger tg2 as (reprovou_material, realizar_proc_exclusao);
22
23 set final wed-state as material_bom_estado is not null and (procedimento_inclusao is not null or
    procedimento_exclusao is not null);
24
25 commit;

```

Código 5.4: Especificação WED-SQL do processo de processamento de materiais do SISAUT.

5.3 Simulação

Esta seção apresenta como as instâncias de um processo são executadas na implementação da abordagem WED-flow. Iniciando pela criação dos bancos de dados que representam os modelos do SISAUT a partir dos comandos expressos em linguagem WED-SQL e a possibilidade de consultar as estruturas criadas usando comandos de consulta da linguagem WED-SQL. Será explicado com mais detalhes como são implementadas as WED-transitions usadas pelos modelos dos processos do SISAUT. Além disso, serão exibidos exemplos de execução de instâncias do SISAUT com dados simulados.

5.3.1 Criação dos bancos de dados

Quando os comandos de criação de modelo da linguagem WED-SQL são fornecidos ao programa “wsq”, é acionado o compilador de linguagem WED-SQL para traduzir os comandos para a linguagem SQL. O comando inicial de criação de um modelo WED-flow realiza os procedimentos necessários para a criação de um banco de dados com tabelas representando os tipos de estrutura que compõem um modelo WED-flow. Os demais comandos SQL referentes a esse modelo são executados sobre o banco de dados criado para o modelo.

Além dos comandos de especificação, a linguagem WED-SQL oferece comandos que permitem verificar o estado do modelo, listando por tipo as estruturas que compõem o modelo. Para o caso do banco de dados criado a partir do modelo dos processos de entrevista e coleta do SISAUT, temos as seguintes estruturas:

WED-attribute	Default value
body_id	NULL
dados_necessarios	NULL
projetos_escolhidos	NULL
termos_consentimento	NULL
projetos_consentidos	NULL
coleta_aprovada	NULL
questionarios_aplicados	NULL
medidas_obtidas	NULL
materiais_coletados	NULL
materiais_descartados	NULL
projetos_abortados	NULL

Tabela 5.1: Resultado da execução do comando “list wed-attributes;” no programa “wsql” para obter a listagem de WED-attributes do modelo dos processos de coleta e entrevista.

WED-condition	Predicate
condicao_inicial	body_id is not null and dados_necessarios is null and medidas_obtidas is null
iniciou_entrevista	body_id is not null and dados_necessarios is null
coletou_dados	dados_necessarios::bool and projetos_escolhidos is null
escolheu_projetos	projetos_escolhidos is not null and termos_consentimento is null
aplicou_consentimentos	termos_consentimento::bool and projetos_consentidos is null
selecionou_projetos	projetos_consentidos is not null and questionarios_aplicados is null
iniciou_coleta	body_id is not null and medidas_obtidas is null
pode_coletar_materiais	medidas_obtidas::bool and coleta_aprovada::bool and materiais_coletados is null and materiais_descartados is null
coletou_materiais	(materiais_coletados is not null or materiais_descartados is not null) and projetos_abortados is null

Tabela 5.2: Resultado da execução do comando “list wed-conditions;” no programa “wsql” para obter a listagem de WED-conditions do modelo dos processos de coleta e entrevista.

É importante observar que a WED-condition final não aparece na lista de WED-conditions pois não é criada no banco de dados, já que não faz sentido existir uma WED-trigger associada à WED-condition final. Apenas a expressão da WED-condition final é usada para verificar se o estado de uma instância é final. É possível verificar a expressão da WED-condition final usando o comando “show final wed-state;”, conforme exemplificado abaixo:

Final WED-state
coleta_aprovada is not null and questionarios_aplicados is not null and ((coleta_aprovada::bool and projetos_abortados is not null) or coleta_aprovada::bool = false)

Tabela 5.3: Resultado da execução do comando “show final wed-state;” no programa “wsql” para exibir a expressão da condição final do modelo dos processos de coleta e entrevista.

WED-transition	Timeout	Stop condition
coletar_dados	365 days, 0:00:00	NULL
escolher_projetos	365 days, 0:00:00	NULL
aplicar_consentimentos	365 days, 0:00:00	NULL
selecionar_projetos	365 days, 0:00:00	NULL
aplicar_questionarios	365 days, 0:00:00	NULL
registrar_medidas	365 days, 0:00:00	NULL
coletar_materiais	365 days, 0:00:00	NULL
abortar_projetos	365 days, 0:00:00	NULL

Tabela 5.4: Resultado da execução do comando “list wed-transitions;” no programa “wsql” para obter a listagem de WED-transitions do modelo dos processos de coleta e entrevista.

Para os modelos abordados neste capítulo, as WED-transitions não precisaram receber configurações de tempo máximo de execução (*timeout*) ou de condição de interrupção (*stop condition*) nos comandos WED-SQL de criação de WED-transitions. Sendo assim, os campos “*Timeout*” e “*Stop condition*” do resultado foram preenchidos com valores padrão da abordagem, respectivamente, tempo máximo de um ano para a execução da WED-transition e valor nulo indicando a ausência de condição de interrupção de execução da WED-transition.

WED-trigger	WED-condition	WED-transition	Enabled
tg0	iniciou_entrevista	coletar_dados	True
tg1	coletou_dados	escolher_projetos	True
tg2	escolheu_projetos	aplicar_consentimentos	True
tg3	aplicou_consentimentos	selecionar_projetos	True
tg4	selecionou_projetos	aplicar_questionarios	True
tg5	iniciou_coleta	registrar_medidas	True
tg6	pode_coletar_materiais	coletar_materiais	True
tg7	coletou_materiais	abortar_projetos	True

Tabela 5.5: Resultado da execução do comando “list wed-triggers;” no programa “wsql” para obter a listagem de WED-triggers do modelo dos processos de coleta e entrevista.

Quando uma WED-trigger é criada, o comportamento padrão da abordagem é deixá-la habilitada e, por conta desse comportamento, o campo “*Enabled*” do resultado acima tem valor “*true*” para todas as WED-triggers. Caso seja interessante ao usuário desabilitar uma das WED-triggers, a linguagem WED-SQL oferece o comando de desabilitação de WED-trigger.

De forma similar, para o modelo de procedimentos temos os seguintes resultados para os comandos de listagem:

WED-attribute	Default value
body_id	NULL
material_coletado	NULL
material_bom_estado	NULL
procedimento_inclusao	NULL
procedimento_exclusao	NULL

Tabela 5.6: Resultado da execução do comando “list wed-attributes;” no programa “wsql” para obter a listagem de WED-attributes do modelo do processo de processamento de materiais.

WED-condition	Predicate
inicio_processamento	body_id is not null and material_coletado <> ” and material_bom_estado is null
aprovou_material	material_bom_estado::bool and procedimento_inclusao is null
reprovou_material	material_bom_estado::bool = false and procedimento_exclusao is null

Tabela 5.7: Resultado da execução do comando “list wed-conditions;” no programa “wsql” para obter a listagem de WED-conditions do modelo do processo de processamento de materiais.

Final WED-state
material_bom_estado is not null and (procedimento_inclusao is not null or procedimento_exclusao is not null)

Tabela 5.8: Resultado da execução do comando “show final wed-state;” no programa “wsql” para exibir a expressão da condição final do modelo do processo de processamento de materiais.

WED-transition	Timeout	Stop condition
verificar_material	365 days, 0:00:00	NULL
realizar_proc_inclusao	365 days, 0:00:00	NULL
realizar_proc_exclusao	365 days, 0:00:00	NULL

Tabela 5.9: Resultado da execução do comando “list wed-transitions;” no programa “wsql” para obter a listagem de WED-transitions do modelo do processo de processamento de materiais.

WED-trigger	WED-condition	WED-transition	Enabled
tg0	inicio_processamento	verificar_material	True
tg1	aprovou_material	realizar_proc_inclusao	True
tg2	reprovou_material	realizar_proc_exclusao	True

Tabela 5.10: Resultado da execução do comando “list wed-triggers;” no programa “wsq” para obter a listagem de WED-triggers do modelo do processo de processamento de materiais.

5.3.2 Implementação das transições

Para executar os testes neste estudo de caso, foram implementadas WED-transitions na forma de serviços Python seguindo o modelo de WED-worker, apresentado em [Pad18]. As WED-transitions simulam a entrada de dados responsáveis por alterarem os estados de dados das instâncias.

As WED-transitions que compõem o sistema SISAUT, além de poderem causar a mudança do estado de dados de uma instância, podem iniciar diversos procedimentos complexos, manuais ou automatizados, realizados por múltiplos atores. Este trabalho se abstrai das particularidades de cada WED-transition, implementando apenas a parte de cada transição que altera o estado de dados da instância.

Para simular a variação no tempo de realização das tarefas das transições, foi implementada uma espera de um número proceduralmente escolhido de segundos antes da alteração no estado de dados em cada transição. Quando uma transição altera um atributo do estado de dados, o valor escolhido para o atributo também é baseado em um número proceduralmente selecionado para simular a variação nos possíveis valores dos atributos. Neste trabalho, a simulação dos valores dos atributos não leva em conta distribuições probabilísticas reais dos valores.

A seguir é apresentada a implementação da WED-transition “verificar_material” do modelo de processamento de materiais:

```

1 from WEDWorker import WEDWorkerTemplate
2 import sys,random
3 import time
4
5 class verificar_material(WEDWorkerTemplate):
6
7     trname = 'verificar_material'
8     dbs = 'user=processamento dbname=processamento application_name=verificar_material'
9     wakeup_interval = 5
10
11     def __init__(self):
12         super().__init__(self.trname, self.dbs, self.wakeup_interval)
13
14
15     def wed_trans(self, wed_state, wid):
16         print("\033[33m> Verificando o estado do material coletado (Id = %d)\033[0m" % wid)
17         print("\033[33m> WED-state: %s\033[0m" % wed_state)
18
19         trtime = random.uniform(1.0, 2.0)
20         if trtime < 1.2:
21             material_bom_estado = 'false'

```

```

22     else:
23         material_bom_estado = 'true'
24         time.sleep(trtime)
25         return "material_bom_estado = '%s'" % (material_bom_estado)
26
27 w = verificar_material()
28
29 try:
30     w.run()
31 except KeyboardInterrupt:
32     print()
33     sys.exit(0)

```

Código 5.5: Código Python da implementação da WED-transition “verificar_material” como um serviço WED-worker.

Para que a classe “verifica_material” siga o modelo de serviço WED-worker é necessário que herde da classe “WEDWorkerTemplate”, que faz parte da implementação WED-flow atual. A classe “WEDWorkerTemplate” é composta de métodos que gerenciam a execução de WED-transitions como serviços WED-workers, realizando a comunicação com o banco de dados referente ao modelo ao qual a WED-transition faz parte. Dentre os métodos da classe “WEDWorkerTemplate”, existe o método abstrato “wed_trans”, que deve ser implementado por classes filhas e deve ser responsável pela execução das tarefas particulares da WED-transition, retornando as alterações que devem ser realizadas nos WED-attributes escritas na sintaxe usada no comando “update” da linguagem SQL.

O construtor da classe “WEDWorkerTemplate” recebe três parâmetros: o nome da WED-transition à qual o serviço representa, uma cadeia de caracteres com a identificação do banco de dados que contém o modelo do processo, e o intervalo em segundos que o serviço deve esperar entre consultas ao banco de dados para verificar se existem requisições de execução do serviço. Esses parâmetros são definidos nas linhas de 7 a 9 da implementação e fornecidos ao construtor da classe mãe no construtor da classe “verifica_material” nas linhas 11 e 12.

A classe “verifica_material” implementa o método abstrato “wed_trans” da classe “WEDWorkerTemplate”, exibindo uma mensagem para documentar que o serviço foi requisitado por uma das instâncias do modelo e outra mensagem documentando o WED-state da instância no início da execução da WED-transition. Na linha 19 é selecionado de forma procedural um número de 1.0 a 2.0 de uma distribuição uniforme, que é usado na linha 24 para simular o tempo de execução das tarefas dessa WED-transition e também nas linhas de 20 a 23 para simular o valor do WED-attribute de estado do material coletado. O método então retorna uma cadeia de caracteres que representa a atribuição do valor simulado para o WED-attribute de estado do material coletado.

A seguir é instanciado um objeto da classe “verifica_material” e seu método “run” é acionado. Esse método verifica continuamente se há requisições de execução da WED-transition “verificar_material” resultantes de instâncias do processo de acordo com as configurações indicadas na implementação do WED-worker, atendendo requisições acionando o método “wed_trans” para cada requisição.

O propósito das implementações das transações neste estudo de caso não é a verossimilhança das distribuições dos dados simulados e nem a corretude de tarefas simuladas em cada etapa do processo, mas exibir as mudanças no estado de dados das instâncias de processos de um sistema WED-flow.

5.3.3 Resultados

Quando o banco de dados referente ao modelo está devidamente criado e os serviços WED-worker referentes às WED-transitions do modelo estão implementadas e executando, o sistema está pronto para receber instâncias dos processos modelados.

Esta seção propõe a criação de instâncias dos processos do SISAUT, simulando o comportamento do sistema. No modelo dos processos de entrevista e coleta do SISAUT, uma instância equivale à chegada de um corpo ao SVOC para o qual existe interesse na doação de materiais para projetos de pesquisa. Quanto ao processo de processamento de materiais do SISAUT, cada instância representa um material extraído no processo de coleta de materiais.

A linguagem WED-SQL também oferece comandos para monitorar as instâncias dos processos, possibilitando listar as instâncias e também acompanhar o histórico de execução de cada uma das instâncias. Os comandos de monitoramento de instâncias serão usados para acompanhar a simulação realizada sobre os processos do SISAUT.

Nessa simulação foi assumido um cenário com 4 projetos de pesquisa com diferentes materiais requeridos entre si. As demandas de cada projeto são:

Projeto	Materiais requeridos
Projeto 1	coração, pulmão
Projeto 2	coração, fígado
Projeto 3	rim
Projeto 4	cérebro

Tabela 5.11: *Projetos usados pelas WED-transitions na simulação.*

As instâncias do modelo de coleta e entrevista são criadas recebendo a identificação do corpo ao qual a coleta e a entrevista se referem. A seguir são exibidos os comandos para a criação de instâncias dos modelos cuja execução será monitorada:

```

1 create instance as (body_id) values (1);
2 create instance as (body_id) values (2);
3 create instance as (body_id) values (3);
4 create instance as (body_id) values (4);
5 create instance as (body_id) values (5);
6 create instance as (body_id) values (6);
7 create instance as (body_id) values (7);
8 create instance as (body_id) values (8);
9 create instance as (body_id) values (9);
10 create instance as (body_id) values (10);

```

Código 5.6: *Comandos WED-SQL de instanciação do modelo de coleta e entrevista.*

Após a execução das transações referentes às WED-transition do modelo de entrevista e coleta, a lista das instâncias criadas pode ser obtida usando o comando “list instances;” da linguagem WED-SQL:

wid	body_id	dados_necessarios	projetos_escolhidos	termos_consentimento	projetos_consentidos	coleta_aprovada	questionarios_aplicados	medidas_obtidas	materiais_coletados	materiais_descartados	projetos_abortados	Status
1	1	true	[2]	true	[2]	true	[2]	true	[coração, fígado]	[]	[]	Final
2	2	true	[1, 3]	true	[1, 3]	true	[1, 3]	true	[coração, pulmão]	[rim]	[3]	Final
3	3	true	[2, 3]	true	[3]	true	[3]	true	[rim]	[]	[]	Final
4	4	true	[1, 3, 4]	true	[3, 4]	true	[3, 4]	true	[rim, cérebro]	[]	[]	Final
5	5	true	[1, 4]	true	[1, 4]	true	[1, 4]	true	[coração, pulmão, cérebro]	[]	[]	Final
6	6	true	[2]	true	[2]	true	[2]	true	[coração]	[fígado]	[2]	Final
7	7	true	[2, 3]	true	[]	false	[]	true				Final
8	8	true	[1, 3]	true	[1]	true	[1]	true	[coração, pulmão]	[]	[]	Final
9	9	true	[1]	true	[1]	true	[1]	true	[coração, pulmão]	[]	[]	Final
10	10	true	[1, 4]	true	[1, 4]	true	[1, 4]	true	[coração, pulmão, cérebro]	[]	[]	Final

Tabela 5.12: Resultado do comando de listagem de instâncias no modelo de coleta e entrevista após a execução das instâncias.

Para cada instância criada, a listagem de instâncias retorna o identificador da instância (wid), o estado de dados da instância no momento da execução do comando de listagem e o status da instância (final, em transação ou estado de exceção). Durante a execução das transações são gerados proceduralmente valores para os atributos que dependem de fatores externos como quais dos projetos escolhidos receberam consentimento dos familiares (projetos_consentidos).

O histórico de alterações no estado de dados de uma instância pode ser obtido usando o comando “get trace for <wid>;” da linguagem WED-SQL. Para cada alteração no estado de dados da instância, esse comando retorna o identificador da instância (wid), a origem da alteração do estado de dados (Written by), as WED-transitions disparadas pelo novo estado de dados (Fired), o status da instância (Status), o momento da mudança do estado de dados (Timestamp) e o estado de dados após a mudança (WED-state). A origem da alteração do estado de dados geralmente é uma WED-transition, exceto pelo início da instância, que é gerado pelo comando de instanciação e marcado como “_INIT_”.

A seguir é exibido o histórico da instância do modelo de coleta e entrevista com identificador 1:

wid	Written by	Fired	Status	Timestamp	WED-state
1	_INIT_	['coletar_dados', 'registrar_medidas']	Transactioning	2020-12-18 16:14:13.809745	{'body_id': '1', 'dados_necessarios': None, 'projetos_escolhidos': None, 'termos_consentimento': None, 'projetos_consentidos': None, 'coleta_aprovada': None, 'questionarios_aplicados': None, 'medidas_obtidas': None, 'materiais_coletados': None, 'materiais_descartados': None, 'projetos_abortados': None}
1	coletar_dados	['escolher_projetos']	Transactioning	2020-12-18 16:16:52.488817	{'body_id': '1', 'dados_necessarios': 'true', 'projetos_escolhidos': None, 'termos_consentimento': None, 'projetos_consentidos': None, 'coleta_aprovada': None, 'questionarios_aplicados': None, 'medidas_obtidas': None, 'materiais_coletados': None, 'materiais_descartados': None, 'projetos_abortados': None}
1	registrar_medidas	[]	Transactioning	2020-12-18 16:16:52.571569	{'body_id': '1', 'dados_necessarios': 'true', 'projetos_escolhidos': None, 'termos_consentimento': None, 'projetos_consentidos': None, 'coleta_aprovada': None, 'questionarios_aplicados': None, 'medidas_obtidas': 'true', 'materiais_coletados': None, 'materiais_descartados': None, 'projetos_abortados': None}
1	escolher_projetos	['aplicar_consentimentos']	Transactioning	2020-12-18 16:16:54.379397	{'body_id': '1', 'dados_necessarios': 'true', 'projetos_escolhidos': '2', 'termos_consentimento': None, 'projetos_consentidos': None, 'coleta_aprovada': None, 'questionarios_aplicados': None, 'medidas_obtidas': 'true', 'materiais_coletados': None, 'materiais_descartados': None, 'projetos_abortados': None}
1	aplicar_consentimentos	['selecionar_projetos']	Transactioning	2020-12-18 16:16:56.200929	{'body_id': '1', 'dados_necessarios': 'true', 'projetos_escolhidos': '2', 'termos_consentimento': 'true', 'projetos_consentidos': None, 'coleta_aprovada': None, 'questionarios_aplicados': None, 'medidas_obtidas': 'true', 'materiais_coletados': None, 'materiais_descartados': None, 'projetos_abortados': None}
1	selecionar_projetos	['aplicar_questionarios', 'coletar_materiais']	Transactioning	2020-12-18 16:16:57.777038	{'body_id': '1', 'dados_necessarios': 'true', 'projetos_escolhidos': '2', 'termos_consentimento': 'true', 'projetos_consentidos': '2', 'coleta_aprovada': 'true', 'questionarios_aplicados': None, 'medidas_obtidas': 'true', 'materiais_coletados': None, 'materiais_descartados': None, 'projetos_abortados': None}
1	coletar_materiais	['abortar_projetos']	Transactioning	2020-12-18 16:16:59.165486	{'body_id': '1', 'dados_necessarios': 'true', 'projetos_escolhidos': '2', 'termos_consentimento': 'true', 'projetos_consentidos': '2', 'coleta_aprovada': 'true', 'questionarios_aplicados': None, 'medidas_obtidas': 'true', 'materiais_coletados': ['coração, fígado'], 'materiais_descartados': [], 'projetos_abortados': None}
1	abortar_projetos	[]	Transactioning	2020-12-18 16:16:59.196643	{'body_id': '1', 'dados_necessarios': 'true', 'projetos_escolhidos': '2', 'termos_consentimento': 'true', 'projetos_consentidos': '2', 'coleta_aprovada': 'true', 'questionarios_aplicados': None, 'medidas_obtidas': 'true', 'materiais_coletados': ['coração, fígado'], 'materiais_descartados': [], 'projetos_abortados': []}
1	aplicar_questionarios	[]	Final	2020-12-18 16:17:13.940332	{'body_id': '1', 'dados_necessarios': 'true', 'projetos_escolhidos': '2', 'termos_consentimento': 'true', 'projetos_consentidos': '2', 'coleta_aprovada': 'true', 'questionarios_aplicados': '2', 'medidas_obtidas': 'true', 'materiais_coletados': ['coração, fígado'], 'materiais_descartados': [], 'projetos_abortados': []}

Tabela 5.13: Resultado do comando de exibição do histórico da instância do modelo de entrevista e coleta com *body_id* 1 após sua execução.

As instâncias do modelo de processamento são criadas recebendo o tipo de material coletado e a identificação do corpo do qual foi coletado. A seguir são exibidos os comandos de instanciação do modelo de processamento referentes aos materiais coletados nas instâncias de coleta e entrevista:

```

1 create instance as (body_id, material_coletado) values (1, "coração");
2 create instance as (body_id, material_coletado) values (1, "fígado");
3
4 create instance as (body_id, material_coletado) values (2, "coração");
5 create instance as (body_id, material_coletado) values (2, "pulmão");
6

```

```

7 create instance as (body_id, material_coletado) values (3, "rim");
8
9 create instance as (body_id, material_coletado) values (4, "rim");
10 create instance as (body_id, material_coletado) values (4, "cérebro");
11
12 create instance as (body_id, material_coletado) values (5, "coração");
13 create instance as (body_id, material_coletado) values (5, "pulmão");
14 create instance as (body_id, material_coletado) values (5, "cérebro");
15
16 create instance as (body_id, material_coletado) values (6, "coração");
17
18 create instance as (body_id, material_coletado) values (8, "coração");
19 create instance as (body_id, material_coletado) values (8, "pulmão");
20
21 create instance as (body_id, material_coletado) values (9, "coração");
22 create instance as (body_id, material_coletado) values (9, "pulmão");
23
24 create instance as (body_id, material_coletado) values (10, "coração");
25 create instance as (body_id, material_coletado) values (10, "pulmão");
26 create instance as (body_id, material_coletado) values (10, "cérebro");

```

Código 5.7: Comandos WED-SQL de instanciação do modelo de processamento de materiais.

Após a execução das transações referentes às WED-transitions do modelo de processamento, a lista das instâncias do modelo criadas é a seguinte:

wid	body_id	material_coletado	material_bom_estado	procedimento_inclusao	procedimento_exclusao	Status
1	1	"coração"	true	true		Final
2	1	"fígado"	true	true		Final
3	2	"coração"	true	true		Final
4	2	"pulmão"	true	true		Final
5	3	"rim"	false		true	Final
6	4	"rim"	true	true		Final
7	4	"cérebro"	true	true		Final
8	5	"coração"	true	true		Final
9	5	"pulmão"	false		true	Final
10	5	"cérebro"	false		true	Final
11	6	"coração"	true	true		Final
12	8	"coração"	true	true		Final
13	8	"pulmão"	true	true		Final
14	9	"coração"	true	true		Final
15	9	"pulmão"	true	true		Final
16	10	"coração"	true	true		Final
17	10	"pulmão"	false		true	Final
18	10	"cérebro"	true	true		Final

Tabela 5.14: Resultado do comando de listagem de instâncias no modelo de processamento após a execução das instâncias.

A seguir é exibido o histórico de alterações no estado de dados da instância com wid = 1 do modelo de processamento:

wid	Written by	Fired	Status	Timest amp	WED-state
1	_INIT_	['verificar_material']	Transactioning	2020-12-19 02:06:50.590523	{'body_id': '1', 'material_coletado': "coração", 'material_bom_estado': None, 'procedimento_inclusao': None, 'procedimento_exclusao': None}
1	verificar_material	['realizar_proc_inclusao']	Transactioning	2020-12-19 02:07:59.191190	{'body_id': '1', 'material_coletado': "coração", 'material_bom_estado': 'true', 'procedimento_inclusao': None, 'procedimento_exclusao': None}
1	realizar_proc_inclusao		Final	2020-12-19 02:08:00.464656	{'body_id': '1', 'material_coletado': "coração", 'material_bom_estado': 'true', 'procedimento_inclusao': 'true', 'procedimento_exclusao': None}

Tabela 5.15: Resultado do comando de exibição do histórico da instância do modelo de processamento com *body_id* 1 e material coração.

5.4 Considerações finais

Este capítulo exemplifica o uso da implementação WED-flow para modelar os processos de um sistema, criar instâncias dos processos e monitorar estruturas do modelo e instâncias. Todos os passos apresentados no capítulo foram realizados usando a nova versão da implementação WED-flow com os compiladores de linguagem WED-SQL e WED-flow apresentados respectivamente nos capítulos 3 e 4.

Foi exibida a modelagem dos processos do SISAUT usando os comandos da linguagem de modelagem WED-flow e sua tradução para comandos de especificação WED-SQL realizada pelo programa wf2wsq, que faz uso do compilador de linguagem de modelagem WED-flow.

A linguagem WED-SQL é usada como intermediária entre a linguagem WED-flow e a linguagem SQL para criar as estruturas de um modelo WED-flow no banco de dados. Além disso, a linguagem WED-SQL é usada para visualizar as estruturas do modelo WED-flow criadas no banco de dados e para criar e monitorar instâncias dos processos. Todas essas funcionalidades são oferecidas pelo programa wsq, que foi modificado neste trabalho para fazer uso do compilador de linguagem WED-SQL.

Capítulo 6

Conclusões

Este trabalho contribui com a implementação da abordagem WED-flow, integrando a ela um compilador da linguagem de modelagem WED-flow e um compilador da linguagem WED-SQL de especificação, instanciação e monitoramento.

Com a implementação do compilador da linguagem de modelagem WED-flow e do programa wf2wsq, a implementação WED-flow passa a apoiar a linguagem de modelagem WED-flow. Essa linguagem oferece uma representação dos elementos do modelo em nível mais alto do que a linguagem WED-SQL, sem se preocupar com detalhes de implementação, o que facilita a representação e leitura do modelo, já que não requer do projetista conhecimento dos comandos de especificação da linguagem WED-SQL, de maior carga semântica.

Uma analogia para a relação entre as linguagens abordadas neste trabalho é a relação entre os modelos Entidade Relacionamento. De forma análoga à linguagem de modelagem WED-flow, o modelo conceitual representa as principais entidades e relacionamentos sem se preocupar com detalhes de implementação. Já o modelo lógico, assim como os comandos de especificação da linguagem WED-SQL, representa as entidades em um nível mais baixo, com detalhes de implementação que são independentes do SGBDR. O modelo físico é composto de comandos SQL que representam o esquema em um nível mais baixo, com todos os detalhes de implementação, inclusive os que são específicos do SGBDR, de forma similar aos comandos SQL resultantes da compilação da linguagem WED-SQL.

Ambos os compiladores seguem a arquitetura clássica de compiladores, que isola as etapas de análise léxica, análise sintática, análise semântica e geração de código na linguagem alvo. Os compiladores foram construídos a partir de uma descrição formal de suas linguagens de origem (WED-flow e WED-SQL). O uso da ferramenta ANTLR permite automatizar a escrita dos algoritmos de análise léxica e sintática e oferece uma biblioteca para auxiliar na implementação das demais etapas de compilação.

Os compiladores facilitam a adição e a alteração de comandos nas linguagens. Os padrões de reconhecimento de cada comando são expressos formalmente nas gramáticas, o que facilita o entendimento humano dos padrões. O isolamento das etapas de compilação auxilia na localização de possíveis erros inseridos no compilador, facilitando suas correções. Além disso, a automatização da geração de analisadores léxicos e sintáticos não só reduz o tempo gasto nas alterações nos compiladores como também dificulta a inserção de erros nessas etapas comparado à implementação manual.

A flexibilidade provida pelos compiladores possui duas limitações:

- As etapas de verificação semântica e de geração de código na linguagem alvo continuam a ser escritas manualmente, já que não podem ser automatizadas pela ferramenta ANTLR, embora tenham a implementação facilitada pela biblioteca da ANTLR.
- O algoritmo de análise sintática ALL(*) gerado pela ferramenta ANTLR restringe as gramáticas das linguagens a evitarem regras indiretamente recursivas à esquerda.

6.1 Próximos passos

Está em desenvolvimento uma plataforma web para execução e visualização da implementação da abordagem WED-flow. A plataforma irá permitir que um usuário crie e visualize modelos e instâncias em um ambiente remoto de tamanho limitado.

O usuário poderá inserir comandos WED-flow para a modelagem WED-flow de seus processos e comandos WED-SQL para instanciação e monitoramento. A interface terá botões e campos de entrada para auxiliar o usuário na escrita dos comandos, que serão exibidos ao usuário antes de serem enviados ao *backend*. A ideia é familiarizar o usuário com as linguagens WED-flow e WED-SQL.

Quando os comandos são enviados ao *backend*, as etapas de análise léxica, sintática e semântica dos compiladores são usadas para verificar se os comandos são válidos por questões de segurança e então são enviados a um ambiente isolado que contém a implementação do WED-flow já configurada para execução. Para comandos de monitoramento, a saída dos comandos será apresentada ao usuário usando uma forma de visualização gráfica intuitiva.

Os ambientes serão implementados usando *containers* Docker, que são abstrações de software que empacotam código e dependências [Inc]. Esses ambientes conterão o código da implementação WED-flow, suas dependências e as configurações necessárias. Após o uso do ambiente via plataforma, o usuário poderá realizar o *download* do ambiente contendo seus modelos, armazenado em um servidor por um tempo limitado.

Uma das limitações da implementação atual do WED-flow é a falta de controle de versionamento de modelos. Um trabalho futuro é implementar a representação estrutural de versões de modelos, oferecendo a visualização de regras de negócio de versões antigas de um modelo e do histórico de mudanças no processo.

Outra ideia para desenvolvimentos futuros é aplicar a abordagem WED-flow para implementar um sistema de gerenciamento de algoritmos de aprendizado de máquina usados para detectar e classificar eventos.

Referências Bibliográficas

- [AU72] Alfred V. Aho e Jeffrey D. Ullman. *The Theory of Parsing, Translation, and Compiling*. Prentice-Hall, Inc., USA, 1972. 21
- [BPS01] J. Bergstra, A. Ponse e S. Smolka. *Handbook of Process Algebra*. Elsevier Science, 1st edição, 2001. 1
- [Cod70] E. F. Codd. A relational model of data for large shared data banks. *Commun. ACM*, 13(6):377–387, Junho 1970. 8
- [Der69] F. L. Deremer. Practical translators for lr(k) languages. Relatório técnico, USA, 1969. 22
- [DeR71] Franklin L. DeRemer. Simple lr(k) grammars. *Commun. ACM*, 14(7):453–460, Julho 1971. 22
- [DvdAtH05] Marlon Dumas, Wil M. van der Aalst e Arthur H. ter Hofstede. *Process-Aware Information Systems: Bridging People and Software through Process Technology*. Wiley, 1st edição, 2005. 1
- [EBN] Iso/iec 14977:1996, information technology - syntactic metalanguage - extended bnf. 18
- [FBTP12] João E. Ferreira, Kelly R. Braghetto, Osvaldo K. Takai e Calton Pu. Transactional recovery support for robust exception handling in business process services. *2012 IEEE 19th International Conference on Web Services*, páginas 303–310, 2012. 2, 4, 7
- [For02] Bryan Ford. Packrat parsing: Simple, powerful, lazy, linear time, functional pearl. *SIGPLAN Not.*, 37(9):36–47, Setembro 2002. 22
- [Fou] Free Software Foundation. Bison 3.2. <https://www.gnu.org/software/bison/manual/bison.html>. Último acesso em 03/11/2018. 23
- [FTK⁺17] Joao Eduardo Ferreira, Pedro Losco Takecian, Leonardo Tadashi Kamaura, Bruno Padilha e Calton Pu. Dependency management with wed-flow techniques and tools: a case study. Em *2017 IEEE 3rd International Conference on Collaboration and Internet Computing (CIC)*, páginas 379–388. IEEE, 2017. 2, 4, 8, 53
- [FTMP10] João E. Ferreira, Osvaldo K. Takai, Simon Malkowski e Calton Pu. Reducing exception handling complexity in business process modeling and implementation: The wed-flow approach. Em *Proceedings of the 2010 International Conference on On the Move to Meaningful Internet Systems - Volume Part I, OTM'10*, páginas 150–167, Berlin, Heidelberg, 2010. Springer-Verlag. 1, 9
- [FWMP10] João Ferreira, Qinyi Wu, Simon Malkowski e Calton Pu. Towards flexible event-handling in workflows through data states. páginas 344–351, 07 2010. 7
- [GRBPF12] Marcela Garcia, Kelly Rosa Braghetto, Calton Pu e João Ferreira. An implementation of a transaction model for business process systems. *JIDM*, 3:271–286, 01 2012. 9

- [Inc] Docker Inc. Docker. <https://www.docker.com/>. Último acesso em 10/01/2021. 74
- [JEA⁺07] Diane Jordan, John Evdemon, Alexandre Alves, Assaf Arkin, Sid Askary, Charlton Barreto, Ben Bloch, Francisco Curbera, Mark Ford, Yaron Goland et al. Web services business process execution language version 2.0. *OASIS standard*, 11(120):5, 2007. 1
- [Knu65] Donald E. Knuth. On the translation of languages from left to right. *Information and Control, Volume 8, Issue 6, December 1965*, páginas 607–639, 1965. 21
- [Knu71] Donald E. Knuth. Top-down syntax analysis. *Acta Informatica, Volume 1, Issue 2, June 1971*, páginas 79–110, 1971. 21
- [LP98] Harry R. Lewis e Christos H. Papadimitriou. *Elements of the Theory of Computation*. Prentice-Hall, 2nd edição, 1998. 17
- [LS68] P. M. Lewis e R. E. Stearns. Syntax-directed transduction. *J. ACM*, 15(3):465–488, Julho 1968. 22
- [MLWoL] Hanspeter Mössenböck, Markus Löberbauer, Albrecht Wöß e University of Linz. The compiler generator coco/r. <http://www.ssw.uni-linz.ac.at/Coco/>. Último acesso em 03/11/2018. 23
- [Mur89] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, 1989. 1
- [Pad18] Bruno Padilha. Wed-sql: uma linguagem declarativa intermediária com apoio transacional para a modelagem e implementação de sistemas de informação cientes de processos. Dissertação de Mestrado, Instituto de Matemática e Estatística, Universidade de São Paulo, Brasil, Outubro 2018. 11, 16, 65
- [Par] ANTLR / Terence Parr. Antlr. <http://www.antlr.org>. Último acesso em 03/11/2018. 23
- [Par13] Terence Parr. *The Definitive Antlr 4 Reference*. Pragmatic Bookshelf, 2nd edição, 2013. 18, 24, 29, 31
- [PF11] Terence Parr e Kathleen Fisher. Ll(*): The foundation of the antlr parser generator. *SIGPLAN Not.*, 46(6):425–436, Junho 2011. 22
- [PHF14] Terence Parr, Sam Harwell e Kathleen Fisher. Adaptive ll(*) parsing: The power of dynamic analysis. *2014 ACM International Conference on Object Oriented Programming Systems Languages & Applications*, páginas 579–598, 2014. 22
- [PRS⁺18] Bruno Padilha, Rafael Liberato Roberto, André Luis Schwerz, Calton Pu e João Eduardo Ferreira. Wed-sql: An intermediate declarative language for pais execution. *2018 International Conference on Web Services (ICWS)*, páginas 407–421, 2018. 1, 2, 9
- [PSL17] Bruno Padilha, André Schwerz e Rafael Liberato. Wed-sql: A relational framework for design and implementation of process-aware information systems. páginas 364–369, 06 2017. 9
- [SLPF18] André Schwerz, Rafael Liberato, Calton Pu e João Ferreira. Robust and reliable process-aware information systems. *IEEE Transactions on Services Computing*, PP:1–1, 04 2018. 9
- [SRBF12] Pedro Silva, Kelly Rosa Braghetto e João Ferreira. Uma abordagem ad hoc para o tratamento de exceções em processos transacionais. 10 2012. 9

- [Tom85] Masaru Tomita. *Efficient Parsing for Natural Language: A Fast Algorithm for Practical Systems*. Kluwer Academic Publishers, USA, 1985. 22
- [TST93] Bernd Teufel, Stephanie Schmidt e Thomas Teufel. *C2 Compiler Concepts*. Springer-Verlag/Wien, 1993. 20
- [ÉG] Étienne Gagnon. Sablecc, an object-oriented compiler framework. <http://sablecc.sourceforge.net/thesis/thesis.html>. Último acesso em 03/11/2018. 23