
Domain-Specific Languages

Martin Fowler

With Rebecca Parsons

♣ Addison-Wesley

Upper Saddle River, NJ • Boston • Indianapolis • San Francisco
New York • Toronto • Montreal • London • Munich • Paris • Madrid
Sydney • Tokyo • Singapore • Mexico City

Contents

Preface

Part I: Narratives

Chapter 1: An Introductory Example

Gothic Security

Miss Grant's Controller

The State Machine Model

Programming Miss Grant's Controller

Languages and Semantic Model

Using Code Generation

Using Language Workbenches

Visualization

Chapter 2: Using Domain-Specific Languages

Defining Domain-Specific Languages

Boundaries of DSLs

Fragmentary and Stand-alone DSLs

Why Use a DSL?

Improving Development Productivity

Communication with Domain Experts

Change in Execution Context

Alternative Computational Model

Problems with DSLs

Language Cacophony

Cost of Building

Ghetto Language

Blinkered Abstraction

[Wider Language Processing](#)

[DSL Lifecycle](#)

[What Makes a Good DSL Design?](#)

Chapter 3: Implementing DSLs

[Architecture of DSL Processing](#)

[The Workings of a Parser](#)

[Grammars, Syntax, and Semantics](#)

[Parsing Data](#)

[Macros](#)

[Testing DSLs](#)

[*Testing the Semantic Model*](#)

[*Testing the Parser*](#)

[*Testing the Scripts*](#)

[Handling Errors](#)

[Migrating DSLs](#)

Chapter 4: Implementing an Internal DSL

[Fluent and Command-Query APIs](#)

[The Need for a Parsing Layer](#)

[Using Functions](#)

[Literal Collections](#)

[Using Grammars to Choose Internal Elements](#)

[Closures](#)

[Parse Tree Manipulation](#)

[Annotation](#)

[Literal Extension](#)

[Reducing the Syntactic Noise](#)

[Dynamic Reception](#)

[Providing Some Type Checking](#)

Chapter 5: Implementing an External DSL

Syntactic Analysis Strategy

Output Production Strategy

Parsing Concepts

Separated Lexing

Grammars and Languages

Regular, Context-Free, and Context-Sensitive Grammars

Top-Down and Bottom-Up Parsing

Mixing-in Another Language

XML DSLs

Chapter 6: Choosing between Internal and External DSLs

Learning Curve

Cost of Building

Programmer Familiarity

Communication with Domain Experts

Mixing In the Host Language

Strong Expressiveness Boundary

Runtime Configuration

Sliding into Generality

Composing DSLs

Summing Up

Chapter 7: Alternative Computational Models

A Few Alternative Models

Decision Table

Production Rule System

State Machine

Dependency Network

Choosing a Model

Chapter 8: Code Generation

[Choosing What to Generate](#)
[How to Generate](#)
[Mixing Generated and Handwritten Code](#)
[Generating Readable Code](#)
[Preparse Code Generation](#)
[Further Reading](#)

Chapter 9: Language Workbenches

[Elements of Language Workbenches](#)
[Schema Definition Languages and Meta-Models](#)
[Source and Projectional Editing](#)
[*Multiple Representations*](#)
[Illustrative Programming](#)
[Tools Tour](#)
[Language Workbenches and CASE tools](#)
[Should You Use a Language Workbench?](#)

Part II: Common Topics

Chapter 10: A Zoo of DSLs

[Graphviz](#)
[JMock](#)
[CSS](#)
[Hibernate Query Language \(HQL\)](#)
[XAML](#)
[FIT](#)
[Make et al](#)

Chapter 11: Semantic Model

[How It Works](#)
[When to Use It](#)
[The Introductory Example \(Java\)](#)

Chapter 12: Symbol Table

[How It Works](#)

[Statically Typed Symbols](#)

[When to Use It](#)

[Further Reading](#)

[Dependency Network in an External DSL \(Java and ANTLR\)](#)

[Using Symbolic Keys in an Internal DSL \(Ruby\)](#)

[Using Enums for Statically Typed Symbols \(Java\)](#)

Chapter 13: Context Variable

[How It Works](#)

[When to Use It](#)

[Reading an INI File \(C#\)](#)

Chapter 14: Construction Builder

[How It Works](#)

[When to Use It](#)

[Building Simple Flight Data \(C#\)](#)

Chapter 15: Macro

[How It Works](#)

[Textual Macros](#)

[Syntactic Macros](#)

[When to Use It](#)

Chapter 16: Notification

[How It Works](#)

[When to Use It](#)

[A Very Simple Notification \(C#\)](#)

[Parsing Notification \(Java\)](#)

Part III: External DSL Topics

Chapter 17: Delimiter-Directed Translation

How It Works

When to Use It

Frequent Customer Points (C#)

Semantic Model

The Parser

Parsing Nonautonomous Statements with Miss Grant's Controller (Java)

Chapter 18: Syntax-Directed Translation

How It Works

The Lexer

Syntactic Analyzer

Output Production

Semantic Predicates

When to Use It

Further Reading

Chapter 19: BNF

How It Works

Multiplicity Symbols (Kleene Operators)

Some Other Useful Operators

Parsing Expression Grammars

Converting EBNF to Basic BNF

Code Actions

When to Use It

Chapter 20: Regex Table Lexer (by Rebecca Parsons)

How It Works

When to Use It

Lexing Miss Grant's Controller (Java)

Chapter 21: Recursive Descent Parser (by Rebecca Parsons)

How It Works

When to Use It

Further Reading

Recursive Descent and Miss Grant's Controller (Java)

Chapter 22: Parser Combinator (by Rebecca Parsons)

How It Works

Dealing with the Actions

Functional Style of Combinators

When to Use It

Parser Combinators and Miss Grant's Controller (Java)

Chapter 23: Parser Generator

How It Works

Embedding Actions

When to Use It

Hello World (Java and ANTLR)

Writing the Basic Grammar

Building the Syntactic Analyzer

Adding Code Actions to the Grammar

Using Generation Gap

Chapter 24: Tree Construction

How It Works

When to Use It

Using ANTLR's Tree Construction Syntax (Java and ANTLR)

Tokenizing

Parsing

Populating the Semantic Model

Tree Construction Using Code Actions (Java and ANTLR)

Chapter 25: Embedded Translation

How It Works

When to Use It

Miss Grant's Controller (Java and ANTLR)

Chapter 26: Embedded Interpretation

How It Works

When to Use It

A Calculator (ANTLR and Java)

Chapter 27: Foreign Code

How It Works

When to Use It

Embedding Dynamic Code (ANTLR, Java, and Javascript)

Semantic Model

Parser

Chapter 28: Alternative Tokenization

How It Works

Quoting

Lexical State

Token Type Mutation

Ignoring Token Types

When to Use It

Chapter 29: Nested Operator Expression

How It Works

Using Bottom-Up Parsers

Top-Down Parsers

When to Use It

Chapter 30: Newline Separators

How It Works

When to Use It

Chapter 31: External DSL Miscellany

Syntactic Indentation

Modular Grammars

Part IV: Internal DSL Topics

Chapter 32: Expression Builder

How It Works

When to Use It

A Fluent Calendar with and without a Builder (Java)

Using Multiple Builders for the Calendar (Java)

Chapter 33: Function Sequence

How It Works

When to Use It

Simple Computer Configuration (Java)

Chapter 34: Nested Function

How It Works

When to Use It

The Simple Computer Configuration Example (Java)

Handling Multiple Different Arguments with Tokens (C#)

Using Subtype Tokens for IDE Support (Java)

Using Object Initializers (C#)

Recurring Events (C#)

Semantic Model

The DSL

Chapter 35: Method Chaining

How It Works

Builders or Values

Finishing Problem

Hierarchic Structure

Progressive Interfaces

When to Use It

The Simple Computer Configuration Example (Java)

Chaining with Properties (C#)

Progressive Interfaces (C#)

Chapter 36: Object Scoping

How It Works

When to Use It

Security Codes (C#)

Semantic Model

DSL

Using Instance Evaluation (Ruby)

Using an Instance Initializer (Java)

Chapter 37: Closure

How It Works

When to Use It

Chapter 38: Nested Closure

How It Works

When to Use It

Wrapping a Function Sequence in a Nested Closure (Ruby)

Simple C# Example (C#)

Using Method Chaining (Ruby)

Function Sequence with Explicit Closure Arguments (Ruby)

Using Instance Evaluation (Ruby)

Chapter 39: Literal List

How It Works

When to Use It

Chapter 40: Literal Map

How It Works

When to Use It

The Computer Configuration Using Lists and Maps (Ruby)

Evolving to Greenspun Form (Ruby)

Chapter 41: Dynamic Reception

How It Works

When to Use It

Promotion Points Using Parsed Method Names (Ruby)

Model

Builder

Promotion Points Using Chaining (Ruby)

Model

Builder

Removing Quoting in the Secret Panel Controller (JRuby)

Chapter 42: Annotation

How It Works

Defining an Annotation

Processing Annotations

When to Use It

Custom Syntax with Runtime Processing (Java)

Using a Class Method (Ruby)

Dynamic Code Generation (Ruby)

Chapter 43: Parse Tree Manipulation

How It Works

When to Use It

Generating IMAP Queries from C# Conditions (C#)

Semantic Model
Building from C#
Stepping Back

Chapter 44: Class Symbol Table

How It Works
When to Use It
Statically Typed Class Symbol Table (Java)

Chapter 45: Textual Polishing

How It Works
When to Use It
Polished Discount Rules (Ruby)

Chapter 46: Literal Extension

How It Works
When to Use It
Recipe Ingredients (C#)

Part V: Alternative Computational Models

Chapter 47: Adaptive Model

How It Works
Incorporating Imperative Code into an Adaptive Model
Tools
When to Use It

Chapter 48: Decision Table

How It Works
When to Use It
Calculating the Fee for an Order (C#)
Model
The Parser

Chapter 49: Dependency Network

How It Works

When to Use It

Analyzing Potions (C#)

Semantic Model

The Parser

Chapter 50: Production Rule System

How It Works

Chaining

Contradictory Inferences

Patterns in Rule Structure

When to Use It

Validations for club membership (C#)

Model

Parser

Evolving the DSL

Eligibility Rules: extending the club membership (C#)

The Model

The Parser

Chapter 51: State Machine

How It Works

When to Use It

Secret Panel Controller (Java)

Part VI: Code Generation

Chapter 52: Transformer Generation

How It Works

When to Use It

Secret Panel Controller (Java generating C)

Chapter 53: Templated Generation

How It Works

When to Use It

Generating the Secret Panel State Machine with Nested Conditionals
(Velocity and Java generating C)

Chapter 54: Embedment Helper

How It Works

When to Use It

Secret Panel States (Java and ANTLR)

Should a Helper Generate HTML? (Java and Velocity)

Chapter 55: Model-Aware Generation

How It Works

When to Use It

Secret Panel State Machine (C)

Loading the State Machine Dynamically (C)

Chapter 56: Model Ignorant Generation

How It Works

When to Use It

Secret Panel State Machine as Nested Conditionals (C)

Chapter 57: Generation Gap

How It Works

When to Use It

Generating Classes from a Data Schema (Java and a Little Ruby)

Bibliography

Index