

The Addison-Wesley Signature Series



A MARTIN FOWLER SIGNATURE
BOOK
Martin

DOMAIN- SPECIFIC LANGUAGES

MARTIN FOWLER
WITH REBECCA PARSONS



List of Patterns

Adaptive Model (487): Arrange blocks of code in a data structure to implement an alternative computational model.

Alternative Tokenization (319): Alter the lexing behavior from within the parser.

Annotation (445): Data about program elements, such as classes and methods, which can be processed during compilation or execution.

BNF (229): Formally define the syntax of a programming language.

Class Symbol Table (467): Use a class and its fields to implement a symbol table in order to support type-aware autocompletion in a statically typed language.

Closure (397): A block of code that can be represented as an object (or first-class data structure) and placed seamlessly into the flow of code by allowing it to reference its lexical scope.

Construction Builder (179): Incrementally create an immutable object with a builder that stores constructor arguments in fields.

Context Variable (175): Use a variable to hold context required during a parse.

Decision Table (495): Represent a combination of conditional statements in a tabular form.

Delimiter-Directed Translation (201): Translate source text by breaking it up into chunks (usually lines) and then parsing each chunk.

Dependency Network (505): A list of tasks linked by dependency relationships. To run a task, you invoke its dependencies, running those tasks as prerequisites.

Dynamic Reception (427): Handle messages without defining them in the receiving class.

Embedded Interpretation (305): Embed interpreter actions into the grammar, so that executing the parser causes the text to be directly interpreted to produce the response.

Embedded Translation (299): Embed output production code into the parser, so that the output is produced gradually as the parse runs.

Embedment Helper (547): An object that minimizes code in a templating system by providing all needed functions to that templating mechanism.

Expression Builder (343): An object, or family of objects, that provides a fluent interface over a normal command-query API.

Foreign Code (309): Embed some foreign code into an external DSL to provide more elaborate behavior than can be specified in the DSL.

Function Sequence (351): A combination of function calls as a sequence of statements.

Generation Gap (571): Separate generated code from non-generated code by inheritance.

Literal Extension (481): Add methods to program literals.

Literal List (417): Represent language expression with a literal list.

Literal Map (419): Represent an expression as a literal map.

Macro (183): Transform input text into a different text before language processing using Templated Generation.

Domain-Specific Languages, Portable Documents

Table of Contents

Contents

Preface

Part I: Narratives

Chapter 1: An Introductory Example

Gothic Security

The State Machine Model

Programming Miss Grants Controller

Languages and Semantic Model

Using Code Generation

Using Language Workbenches

Visualization

Chapter 2: Using Domain-Specific Languages

Defining Domain-Specific Languages

Why Use a DSL?

Problems with DSLs

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

Table of Contents

Parsing Data

Macros

Testing DSLs

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

Mixing-in Another Language

XML DSLs

Chapter 6: Choosing between Internal and External DSLs

Learning Curve

Cost of Building

Programmer Familiarity

Table of Contents

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

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

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

Table of Contents

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

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

When to Use It

Chapter 16: Notification

Table of Contents

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#)

Parsing Nonautonomous Statements with Miss Grants Controller (Java)

Chapter 18: Syntax-Directed Translation

How It Works

When to Use It

Further Reading

Chapter 19: BNF

How It Works

When to Use It

Chapter 20: Regex Table Lexer

How It Works

When to Use It

Lexing Miss Grants Controller (Java)

Chapter 21: Recursive Descent Parser

How It Works

When to Use It

Further Reading

Recursive Descent and Miss Grants Controller (Java)

Chapter 22: Parser Combinator

How It Works

Table of Contents

When to Use It

Parser Combinators and Miss Grants Controller (Java)

Chapter 23: Parser Generator

How It Works

When to Use It

Hello World (Java and ANTLR)

Chapter 24: Tree Construction

How It Works

When to Use It

Using ANTLRs Tree Construction Syntax (Java and ANTLR)

Tree Construction Using Code Actions (Java and ANTLR)

Chapter 25: Embedded Translation

How It Works

When to Use It

Miss Grants 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)

Chapter 28: Alternative Tokenization

How It Works

When to Use It

Chapter 29: Nested Operator Expression

How It Works

Table of Contents

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#)

Chapter 35: Method Chaining

How It Works

When to Use It

Table of Contents

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#)

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)

Table of Contents

Chapter 41: Dynamic Reception

How It Works

When to Use It

Promotion Points Using Parsed Method Names (Ruby)

Promotion Points Using Chaining (Ruby)

Removing Quoting in the Secret Panel Controller (JRuby)

Chapter 42: Annotation

How It Works

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#)

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#)

Table of Contents

Part V: Alternative Computational Models

Chapter 47: Adaptive Model

How It Works

When to Use It

Chapter 48: Decision Table

How It Works

When to Use It

Calculating the Fee for an Order (C#)

Chapter 49: Dependency Network

How It Works

When to Use It

Analyzing Potions (C#)

Chapter 50: Production Rule System

How It Works

When to Use It

Validations for club membership (C#)

Eligibility Rules: extending the club membership (C#)

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

Table of Contents

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