Chapter 69
Abstraction of Computer Language Patterns: The Inference of Textual Notation for a DSL

Jaroslav Porubän
Technical University of Košice, Slovakia

Ján Kollár
Technical University of Košice, Slovakia

Miroslav Sabo
Technical University of Košice, Slovakia

ABSTRACT

In general, designing a domain-specific language (DSL) is a complicated process, requiring the cooperation of experts from both application domain and computer language development areas. One of the problems that may occur is a communication gap between a domain expert and a language engineer. Since domain experts are usually non-technical people, it might be difficult for them to express requirements on a DSL notation in a technical manner. Another compelling problem is that even though the majority of DSLs share the same notation style for representing the common language constructs, a language engineer has to formulate the specification for these constructs repeatedly for each new DSL being designed. The authors propose an innovative concept of computer language patterns to capture the well-known recurring notation style often seen in many computer languages. To address the communication problem, they aim for the way of proposing a DSL notation by providing program examples as they would have been written in a desired DSL. As a combination of these two ideas, the chapter presents a method for example-driven DSL notation specification (EDNS), which utilizes computer language patterns for semi-automated inference of a DSL notation specification from the provided program examples.

DOI: 10.4018/978-1-4666-6042-7.ch069

INTRODUCTION

Designing computer languages is hard, and designing domain-specific computer languages is even harder. The notation of a language must be defined to suit a specific domain but at the same time it has to be processable by a computer. What we often see in the notation of various computer languages is the recurrence of some particular patterns. These notation patterns are used to represent general language constructs found in many computer languages. They help users understand programs written in such languages by basing the notation on their prior experience. People who are familiar with such patterns are able to immediately comprehend a rough idea of a program without having to learn the language first. Since the human usability is one of the main features of DSLs, the notation of these languages is naturally full of such recurring language patterns. From the perspective of the language design, these patterns must be identified and translated into the grammar rules repeatedly for each new DSL. Although for experienced language engineers, this is a simple yet menial and repetitive task, for new ones it may present a serious assignment. To address the problems of both cases we propose to capture the knowledge of a language engineer as computer language patterns. Each language pattern systematically names, explains and captures the recurring notation and provides the means to reflect this notation in a language design. Our ultimate goal is to capture the knowledge of the language design in a form that people can use effectively.

MOTIVATION

Scenario 1: Communication with Domain Experts

Anna works as a developer for a global market company. Until now, the goods between stores and warehouses have been ordered and delivered according to handwritten enquiries. To improve this process, the company decided to go for automation. However, as the style used for writing the enquiries has been retained over the years and many employees are used to it, the company wants to keep it in the electronic version as well.

The task that Anna has been assigned is to develop a language with the well-known structure and notation defined by existing enquiries. The approach to specifying a language notation by providing example documents is also very convenient even if documents do not exist and have to be created at first. This is very common in development of vertical DSLs[1] since for the non-technical domain experts, writing down the examples is often the best way of communicating their thoughts on the look and feel of a language being designed.

Scenario 2: Analyzers of Generated Textual Output

An SMS or email notification is a service commonly provided by many companies. Bob has subscribed to an online auctioning site and now he receives emails every time some important events happen (e.g. somebody bids on an item he is interested in). To increase his chances in the auction, he decided to track the biddings of each participant and analyze them for predictions of their future behavior. Since the only source of such information is emails with well formatted messages, Bob has to extract the desired data from them. Messages can be of a variable length and different content depending on the type and number of events they report therefore using regular expressions would not suffice for this purpose.

This scenario describes a situation where a parser for existing formatted documents has to be created. Software systems generate a lot of textual output, either as a main product of their execution or for other purposes such as logging or reporting. If the output is intended for information transfer and further processing by another system,