Chapter 6
Iterative and Pattern-Based Development of Internal Domain-Specific Languages

Sebastian Günther
Vrije Universiteit Brussel, Belgium

ABSTRACT
Internal DSLs are a special kind of DSLs that use an existing programming language as their host. In this chapter, the author explains an iterative development process for internal DSLs. The goals of this process are: (1) to give developers a familiar environment in which they can use known and proven development steps, techniques, tools, and host languages, (2) to provide a set of repeatable, iterative steps that support the continuous adaptation and evolution of the domain knowledge and the DSL implementation, and (3) to apply design principles that help to develop DSLs with essential properties and to use host language independent design patterns to plan and communicate the design and implementation of the DSL. The process consists of three development steps (analysis, language design, and language implementation) and applies four principles: open form, agile and test-driven development, design pattern knowledge, and design principle knowledge.

INTRODUCTION
Domain-Specific Languages (DSLs) are languages specifically tailored to express the concepts and notations of a domain by embodying suitable abstractions and notations (van Deursen et al., 2000). Traditionally there are two kinds of DSLs. External DSLs allow the developer to freely choose the syntax and semantics of a DSL by writing a custom grammar, lexer, parser, and source code generators, often supported with dedicated development environments. Internal DSLs are built on top of an existing programming language by...
leverages the host constructs and exploiting syntactic variations. Internal DSLs also benefit from the fact that the existing language infrastructure — compiler/interpreter, IDE, debugger — can be reused. We focus our research efforts on internal DSL because they help developers to reuse their existing knowledge of the particular host language and they simplify the integration with other DSLs, frameworks, and libraries. From here on, if we speak of DSL, we mean internal DSLs.

Related work about DSL development can be broadly distinguished into general processes and processes specific for internal DSLs. General processes describe the DSL development as consisting of similar phases like they are used in software development: analysis, design, implementation, testing, deployment (Consel & Marlet, 1998; van Deursen et al., 2000; Mernik et. al, 2005). These phases need to be extended with precise advice how to provide the domain-specific notations and abstractions. Although several case studies for the development of internal DSLs exist, only a few of them explain the development process: In Dinkelaker & Mezini (2008), classes are created for all domain objects, in and Cunningham (2008) commonality/variability analysis is used for analyzing the domain, from which natural language expressions and finally statements and constructs of the DSL are retrieved. These approaches, however, explain host language specific mechanisms adapted for DSL implementation.

These and further related work show the diverse options available to developers of internal DSLs. In this chapter, we carefully review existing approaches, identify their challenges, and present a development process with three phases and four process principles.

In our process, the first phase, domain analysis, collects various domain material like handbooks, manuals, and expert knowledge, to define a domain model that details the concepts, attributes, and relationships of the domain. From this domain model, a set of valid host language expressions is developed during the language design phase. Finally, with the language implementation phase, the semantics behind these expressions are provided. These steps are tightly integrated and iterative in their nature. They allow designers to begin with a small excerpt of the domain: a few domain concepts and operations, for which sample expressions are defined, and implemented. Each step produces an executable DSL. Successive steps add to these expressions or refine existing ones, continuously growing the DSL expression by expression.

In addition to these steps, four principles are applied. The open form principle allows developers to use their known development steps (the three steps of the development process), their techniques (for requirements analysis, domain analysis, and domain modeling), and their host language tools (like the compiler/interpreter or the IDE) for the DSL development. This gives developers a familiar environment in which they can focus on understanding the domain and implementing the DSL. Through the agile and test-driven development principle, the tight integration between the development steps is enabled. The DSL is developed expression by expression, with the tests first, followed by the implementation. During iterations, a complete test suite for the DSL is created. This helps to continuously evolve the DSL and to use the DSL early on. Finally, design pattern knowledge and the design principle knowledge address the identified challenge to plan and communicate DSL designs. Host language independent patterns that document recurring design challenges and their solutions form a rich vocabulary to share DSL design knowledge. Each pattern provides a problem and generic solution, as well as explaining a host language specific application, therefore being also precise enough to be applied to a DSL. The DSL design principles identify essential properties of DSLs and suggest a specific design goal that can be achieved through applying the patterns.

The section “Background and related work” first discusses the background and related work to give a complete overview to other DSL devel-
Related Content

Transformation Mechanisms in the Business Model/Business Process Interface
www.igi-global.com/chapter/transformation-mechanisms-in-the-business-model-business-process-interface/96627?camid=4v1a

Responding to Ongoing Change: Challenges for Information Systems Modeling
www.igi-global.com/article/responding-to-ongoing-change/120173?camid=4v1a

A Software Tool for Reading DICOM Directory Files
www.igi-global.com/chapter/software-tool-reading-dicom-directory/29441?camid=4v1a

Accountability for Service Compliance: A Survey
Jinhui Yao, Shiping Chen and David Levy (2012). International Journal of Systems and Service-Oriented Engineering (pp. 16-43).
www.igi-global.com/article/accountability-service-compliance/64197?camid=4v1a