Summary Sheet- 001

Jackson Raniel

March 28, 2012

1 OBJECTIVES AND PURPOSES OF THE CHAPTER

To make a introduction about what is a Domain Specific Language(DSL) and talk, with simple examples, about in what situations the use of this tool can be useful.

2 CENTRAL ARGUMENTATION

The author talks that DSls is a manneer to make software solutions more understandable for people that aren't programmers.

3 THE CONTRAPOSITIONS

4 CITATIONS

"An external DSL is a domain-specific language represented in a separate language to the main programming language it's working with. This language may use a custom syntax, or it may follow the syntax of another representation such as XML. An internal DSL is a DSL represented within the syntax of a general-purpose language. It's a stylized use of that language for a domain-specific purpose." [1, p.13]

"You may also hear the term embedded DSL as a synonym for internal DSL. Although it is fairly widely used, I avoid this term because "embedded language" may also apply to scripting languages embedded within applications, such as VBA in Excel or Scheme in the Gimp." [1, p.13]

"When people discuss a programming language, you often hear them talk about syntax and semantics. The syntax captures the legal expressions of

the program—everything that in the custom-syntax DSL is captured by the grammar. The semantics of a program is what it means—that is, what it does when it executes. In this case, it is the model that defines the semantics. If you're used to using Domain Models [poeaa], for the moment you can think of a Semantic Model as very close to the same thing." [1, p. 14]

"One opinion I've formed is that the Semantic Model is a vital part of a well-designed DSL. In the wild you'll find some DSLs use a Semantic Model and some do not, but I'm very much of the opinion that you should almost always use a Semantic Model." [1, p.15]

"Looking at it from this point of view, the DSL merely acts as a mechanism for expressing how the model is configured. Much of the benefits of using this approach comes from the model rather than the DSLs. The fact that I can easily configure a new state machine for a customer is a property of the model, not the DSL. The fact that I can make a change to a controller at runtime, without compiling, is a feature of the model, not the DSL. The fact I'm reusing code across multiple installations of controllers is a property of the model, not the DSL. Hence the DSL is merely a thin facade over the model." [1, p.15]

"The benefits of a DSL are particularly relevant for a state machine, which is particular kind of model whose population effectively acts as the program for the system. If we want to change the behavior of a state machine, we do it by altering the objects in its model and their interrelationships. This style of model is often referred to as an Adaptive Model. The result is a system that blurs the distinction between code and data, because in order to understand the behavior of the state machine you can't just look at the code; you also have to look at the way object instances are wired together. Of course this is always true to some extent, as any program gives different results with different data, but there is a greater difference here because the presence of the state objects alters the behavior of the system to a significantly greater degree." [1, p.15]

"Adaptive Models can be very powerful, but they are also often difficult to use because people can't see any code that defines the particular behavior. A DSL is valuable because it provides an explicit way to represent that code in a form that gives people the sensation of programming the state machine." [1, p.15]

"In my discussion so far, I process the DSL to populate the Semantic Model and then execute the Semantic Model to provide the behavior that I want from the controller. This approach is what's known in language circles as interpretation. When we interpret some text, we parse it and immediately produce the result that we want from the program." [1, p. 16]

"In the language world, the alternative to interpretation is compilation.

With compilation, we parse some program text and produce an intermediate output, which is then separately processed to provide the behavior we desire. In the context of DSLs, the compilation approach is usually referred to as code generation." [1, p. 16]

"With interpretation, the eligibility processor parses the rules and loads up the semantic model while it executes, perhaps at startup. When it tests a candidate, it runs the semantic model against the candidate to get a result." [1, p.16]

"In the case of compilation, the parser would load the semantic model as part of the build process for the eligibility processor. During the build, the DSL processor would produce some code that would be compiled, packaged up, and incorporated into the eligibility processor, perhaps as some kind of shared library. This intermediate code would then be run to evaluate a candidate." [1, p. 17]

"Code generation is often awkward in that it often pushes you to do an extra compilation step. To build your program, you have to first compile the state framework and the parser, then run the parser to generate the source code for Miss Grant's controller, then compile that generated code. This makes your build process muchmore complicated." [1, p. 17]

"Code generation is also useful when you want to use DSLs with a language platform that doesn't have the tools for DSL support." [1, p. 17]

"Using code generation is one case where many people don't use a Semantic Model, but parse the input text and directly produce the generated code. Although this is a common way of working with code-generating DSLs, it isn't one I recommend for any but the very simplest cases. Using a Semantic Model allows you to separate the parsing, the execution semantics, and the code generation. This separation makes the whole exercise much simpler. It also allows you to change your mind; for example, you can change your DSL from an internal to an external DSL without altering the code generation routines. Similarly, you can easily generate multiple outputs without complicating the parser. You can also use both an interpreted model and code generation off the same Semantic Model." [1, p.17]

"A language workbench is an environment designed to help people create new DSLs, together with high-quality tooling required to use those DSLs effectively." [1, p. 18]

"One of the big disadvantages of using an external DSL is that you're stuck with relatively limited tooling. " [1, p.18]

"Language workbenches make it easy to define not just a parser, but also a custom editing environment for that language." [1, p. 18]

"One of the common benefits of such tools is that they allow nonprogrammers to program." [1, p. 19]

"Language workbenches support developing new kinds of programming platforms like this. As a result, I think the DSLs they produce are likely to be closer to a spreadsheet than to the DSLs that we usually think of (and that I talk about in this book)." [1, p. 20]

"One of the great advantages of a language workbench is that it enables you to use a wider range of representations of the DSL, in particular graphical representations." [1, p. 20]

5 Resume

Domain Specific Languages(DSL) is languages that are designed to run in specific domains. This languages are a manner to make software solutions in a way where no technical people can colaborate. The sintax of that languages is fully builded with expressions understood in a domain.

There are two wais to make a DSL: building a language that works separately to the main programming language it's work with; or build a languange that use the sintax of a General Purpose Language (GPL). Languages builded in the first way is called external DSL, languages created in the second way are called embedded or internal DSLs.

The model of a DSL defines the semantic of them. "The Semantic Model is a vital part of a well-designed DSL" [1, p. 15]. A DSL is a mechanism for expressing how the model is configured, is a facade to the model. "The benefits of a DSL are particularly relevant for a state machine, which is particular kind of model whose population effectively acts as the program for the system. If we want to change the behavior of a state machine, we do it by altering the objects in its model and their interrelationships. This style of model is often referred to as an Adaptive Model." [1, p. 15]

When a language populate a Semantic Model and execute then to change the behavior of a state machine occurs a interpretation but, DSLs can be compiled too. In that case the DSL produce a output that will processed separately to change que behavior of the machine. This out put can be code and "code generation is also useful when you want to use DSLs with a language platform that doesn't have the tools for DSL support" [1, p. 17].

Some tools called "language workbech" are usefull to help people to create new DSLs. That tools can help not just to define a parser, but also provides a custom editing enviroment to the language. "One of the great advantages of a language workbench is that it enables you to use a wider range of representations of the DSL, in particular graphical representations." [1, p. 20]

References

[1] M Fowler. Domain-specific languages. *Addison-Wesley Professional*, 2010.