Is My DSL a Modeling or Programming Language?

Yu Sun[1], Zekai Demirezen[1], Marjan Mernik[2, 1], Jeff Gray[1], Barrett Bryant[1]

[1]University of Alabama at Birmingham, USA
Department of Computer and Information Sciences
{yusun, zekzek, gray, bryant}@cis.uab.edu

[2] University of Maribor, Slovenia
Faculty of Electrical Engineering and Computer Science
marjan.mernik@uni-mb.si

Abstract

It is often difficult to discern the differences between programming and modeling languages. As an example, the term "domain-specific language" has been used almost interchangeably in academia and industry to represent both programming and modeling languages, which has caused subtle misconceptions. The borders between a modeling and programming language are somewhat vague and not defined crisply. This paper discusses the similarities and differences between modeling and programming languages, and offers some suggestions on how to better differentiate such languages. A list of criteria is presented for language classification, but it is suggested that a set of the criteria be used, rather than a single criterion. Several example domain-specific languages are used as case studies to motivate the discussion.

1. Introduction

This paper focuses on the differences and similarities between programming and modeling languages and provides some criteria for classifying a language. The lack of a well-established definition of the terms "programming" and "modeling" has fostered some widely held misconceptions (e.g., the idea that textual languages are always programming languages and that graphical languages are always modeling languages). This is a difficult question that others have also identified. For example, Greenfield et al. [9] noticed this distinction, but did not provide any criteria for classifying a language. They concluded that the differences between programming and modeling languages are rather insignificant. However, we believe that the differences may have some consequences that are worth considering. The implementation of a language can be better informed by understanding the differences, available tools and implementation strategies for each type of language.

There are many different views and opinions on whether a particular language is a programming or modeling language. The distinction becomes more challenging when domain-specific languages (DSLs) [20] are considered. The aim of this paper is to discuss criteria for classification and apply the criteria to several DSLs to determine if the DSL represents a modeling language or programming language. Several existing criteria for classifying a DSL include: 1) whether the language is expressed in a visual or textual notation, 2) how the language syntax and semantics are defined, 3) issues of language executability, 4) level of abstraction, 5) underlying fundamental concepts of the language, 6) how the language is used in a specific development phase, and 7) multiple views. However, from our experience many of these criteria can lead to false classification. Hence, relying on just one criterion to classify a language is often not sufficient; applying several criteria together may give a more accurate classification. We understand that some of the criteria may be objectionable to some readers, but offer this collection to stimulate discussion on this topic.

The structure of the paper is organized as follows. Section 2 presents the initial context for discussing the differences between programming and modeling languages. A set of suggested criteria are enumerated in Section 3, which are then used in Section 4 to classify several example DSLs. Concluding comments are offered in Section 5.

2. On Programming and Modeling Languages

Programming languages play a central role in computer science. Not surprisingly, it has been observed that programming languages are a programmer's most basic tools [11]. Several informal definitions of a programming language are offered as follows:

- A programming language provides notations that are used to describe a computation in a human-readable form that can be translated into a machine-readable representation [18].
- A programming language is a formal notation that can be used to describe problem solutions in a precise manner [10].
- A programming language is a notation that can be used to write programs [25].
- A programming language is a notation for expressing computation [25].
- A programming language is a standardized communication technique for expressing instructions to a computer. It is a set of syntactic and semantic rules used to define computer programs [29].

However, some of these definitions are too vague to differentiate between programming and modeling languages because some modeling languages may also fit some of these definitions. Modeling is a well-accepted

engineering technique [24], where models are used to understand and comprehend the parts of a complex system under development. The confusion among modeling and programming languages comes from the perception of what constitutes the essential properties of a model. Despite the fact that models are the core of Model Driven Engineering (MDE) [24], there is still not a widely accepted consensus on the definition of a model. Researchers use either too narrow (e.g., a model is an artifact of a modeling language, such as UML [2], describing a system) or too broad definitions (e.g., everything is a model).

Recent attempts to unify various views are described in [17], where a model is an abstraction of a system allowing predictions or inferences to be made. The intention of a model is to represent or describe the system, where the model elements correspond to a concept in the system's domain. An important feature of a model is the reduction principle [17], which states that a model only reflects some of the system's properties. Hence, some models can capture only particular aspects of the system, while other models might be more detailed. In this view, a program is also a model, albeit a very detailed model. Hence, the distinction between modeling languages and programming languages can be blurry.

In this paper, we assume that a model must exhibit the reduction principle and be free of full details. But, this also offers a challenge to the ability to distinguish programming and modeling languages. There are domain-specific models that are not detailed, but are still complete and executable (i.e., it might be that in a narrow domain these details are fixed and should not be represented in a model). Below are some standard definitions for modeling languages:

- "A modeling language is an organized collection of model unit kinds that focus on a particular modeling perspective. A model unit kind is a specific kind of model unit, characterized by the nature of the information that it represents and the intention of using such a representation" [7].
- A modeling language is a language used to specify, visualize, construct, and document a software system [2].
- A modeling language is a language used to present a high-level architectural view of a system [2].

3. Criteria for Language Classification

This section introduces criteria that could be used to classify a language as a programming or modeling language. The thesis of this paper is that no single criterion can classify a language reliably, but that a profile of several criteria may suggest whether a language is a programming or modeling language. The criteria are as follows:

- **Concrete Notation:** Most programming languages use textual notations, but many modeling languages use a graphical notation. Hence, it is often assumed incorrectly that a programming language must be textual, and that a modeling language must be graphical. In some books (e.g., Greenfield et al. [9]), the authors are aware of this misconception, but occasionally their definitions are not precise enough (e.g., "A modeling language is a visual type system for specifying model-based programs" [9]). In others (e.g., Gray et al. [8]), the authors acknowledged that graphical notation is not a key criterion, but do not provide a clear classification. However, the criterion of the concrete notation used by a language is not reliable because there are programming languages that are visual [3], and modeling languages that are textual [4].
- **Language Definition:** Computer languages are defined by their syntax, semantics and pragmatics. A standard and well-established formal method for programming language syntax definition is a context-free grammar (CFG). As an alternative to informal semantics, several formal methods for programming language semantic definition are well-known, such as: attribute grammars, axiomatic semantics, operational semantics, denotational semantics, abstract state machines, or algebraic specifications [22]. Comparatively, modeling languages are often specified using different (semi-)formal methods. The syntax of a modeling language is typically specified using a metamodel in some semi-formal notation (e.g., UML class diagrams adorned with OCL constraints). The semantics of a modeling language is more problematic and often defined through a model compiler or interpreter that translates a model into a program (translational semantics). Actually, both a CFG and a metamodel can be used to describe the syntax of programming and modeling languages. Hence, the criterion pertaining to the method used for language definition is not reliable because there are modeling languages that are not defined by metamodels [4]. Furthermore, the syntax of a graphical language could also be described with a CFG [3].
- **Language Executability:** Paige et al. [23] stated that the primary difference among programming and modeling languages is in their intended domain of use. Programming languages describe executable systems, but modeling languages may not be concerned with executability. However, the issue of executability should not be used as a sole criterion for distinguishing programming and modeling languages. A common misconception is that an executable language must necessarily be a programming language. With a model interpreter or model compiler, a modeling language also can be executed

[19]. In addition, it is not always possible to reach a consensus about the executability of a certain modeling language, so depending on this criterion alone is not reliable for language classification.

- **Level of Abstraction:** The style of specification, which may range from *imperative* to *declarative*, is also used informally to distinguish programming and modeling languages. Some argue that both programming and modeling languages are used to describe software systems, albeit at different levels of abstraction [16]. However, abstraction level is difficult to measure and hence hard to use as a criterion (e.g., some languages are both declarative and imperative). Furthermore, domain-specific languages also raise the abstraction level, but may be programming languages. A more intuitive criterion is whether the language addresses concerns of the problem space (more closely aligned to modeling) or the technical solution space (typically associated with programming).
- **Fundamental Concepts:** Most programming languages are based on a few concepts [28], such as: values, storage, bindings, abstractions, encapsulations, type systems, and sequencers. Among these, the most basic concepts are: values, storage, and bindings, which are less likely to appear in modeling languages. The common constructs in modeling languages are entities (e.g., atoms that are primitive constructs and models that can contain atoms and other submodels) and connections between those entities through ports [24].
- **Development Phase:** A simple criterion for classifying a language is to consider the particular development phase of the software lifecycle where the language is used. For example, modeling languages are used often during the early phases of development to specify the system behavior, structure and requirements so that designers, programmers, and end users can understand the system being modeled, while programming languages are used more frequently in the implementation stages to control the behavior of a computer, express algorithms, and implement systems. However, by raising the abstraction level of software development, domain-specific programming languages have the tendency to move implementation closer to design; on the other hand, with rich semantics and powerful tool support, some modeling languages can be interpreted or compiled to executable entities or codes, which actually play the role of implementation as well. It is possible though rare that languages can be used for different purposes. For instance, Prolog, is a logical programming language that can serve as a formal definition of a metamodel [14], which is actually applied for modeling purposes. The potential dual nature of a language makes classification even more different. Hence, the criterion of development phase needs deep consideration when serving as a classification criterion.
- **Multiple Views:** In contrast with programming languages, MDE makes the assumption that a single model can have different views and that the target system is described by many different models, possibly using different metamodels. As an example, the design of the UML was based on the principle of multiple views, "No single model is sufficient. Every nontrivial system is best approached through a small set of nearly independent models" [2]. Again, this criterion might be useless in isolation because some domains may be narrow enough that just one view is sufficient.

The next section considers these criteria to classify several different types of DSLs.

4. Classifying Different DSLs using the Criteria

We have selected a few DSLs and applied the criteria from Section 3 to determine if a language is a programming or a modeling language. The goal is to check if these criteria are indeed adequate for making such a determination. In Table 1, we use the notation "$\Rightarrow$ P" to denote if the particular criterion suggests a programming language, otherwise we write "$\Rightarrow$ M." The following DSLs were included in our study:

- **VHSIC Hardware Description Language (VHDL)** [1] is a standard DSL for describing digital circuit designs. By offering an appropriate hardware-oriented vocabulary and constructs for using standard libraries and predefined packages, the language yields a substantial reduction in circuit design effort. Widely used in industry and academia, it is among the most successful DSLs.
- **Extended Backus Naur Form (EBNF)** [12] is a DSL for specifying the syntax of computer languages. An EBNF syntax for a language is a set of production rules that generate the sentences of the language and no others.
- **Atlas Transformation Language (ATL)** [13] is a DSL for specifying model transformations. It is a hybrid transformation language, containing both declarative and imperative constructs. Generally, an ATL transformation definition consists of a header section (some basic description about the transformation), an import section (declare imported ATL libraries), a number of helpers (behave like functions to provide navigations over source models) and transformation rules (the basic construct to express the transformation details).
- **Kernel Meta Meta Model (KM3)** [14] is a DSL to define metamodels (i.e., the definition of KM3 is a meta-metamodel). KM3 is intended to be a

lightweight metamodel definition language allowing easy creation and modification of metamodels.

- **Embedded Systems Language (ESML)** [15] is a DSL developed for modeling real-time mission computing embedded avionics applications. It describes a system from such aspects as interfaces, events, components, interactions, and configurations. The ESML is defined within the Generic Modeling Environment (GME) [6], with several interpreters available to generate different artifacts.
- **Structured Query Language (SQL)** [5] is a DSL that provides retrieval and management of data facilities in relational database management systems. SQL enables a programmer to operate on data without needing to know about various aspects of the database internals. Standard SQL is a declarative language; however, imperative constructs have been included in various extensions.
- **XML Transformation Language (XSLT)** [26] is a declarative DSL that is designed to transform XML documents into other XML or human-readable documents. However, it has been extended to include string and date manipulation, as well as data typing capabilities.

Although only a limited number of sample DSLs are listed in the table as test cases, it is obvious that no single criterion can precisely determine the type of the language. For instance, the textual language KM3 has a syntax defined by BNF and an operational semantics, but is really a modeling language, which shows that textual languages based on CFG are not necessarily a programming language. In addition, SQL and ATL are declarative and at a high level of abstraction, but they are programming languages, indicating that both modeling and programming languages can be raised to a high level of abstraction. Furthermore, having multiple views is not enough to confirm a modeling language, because some of the modeling languages like KM3 and EBNF do not support this feature. With the development of MDE, modeling languages will not only serve in requirements and design analysis, but also be able to play an increasingly important role in the implementation phase or even become executable directly. Thus, the particular development stages when a DSL is used, as well as the issue of executability, are not a sole criterion for determining if a language is a modeling or programming language.

A more effective and accurate approach to determine the type of a language is to use multiple criteria in making the classification. From the three sample modeling languages of Table 1 (i.e., EBNF, KM3, ESML), it can be observed that they are all in a medium or higher level of abstraction and can be applied in the design phase. When meeting these criteria simultaneously, the language is more likely to be a modeling language. If additional properties like multiple views, metamodel syntax definition are also qualified, the conclusion is even more precise. Programming languages share the same characteristics such as being executable, applied during the implementation phase, having the same fundamental concepts. Using these criteria together can help toward classifying a language as a programming language. If the criterion of lower level abstraction is applicable, the result further leans toward a programming language.

Some obvious questions arise from this discussion, such as: How complete is this criteria list? What other criteria could be added? During our experimentation, the proposed criteria were very beneficial for classification of each language. Inevitably, there may be some criteria that can be added. As an example, the concept of Turing completeness could be used to differentiate among general-purpose languages (GPLs) and DSLs. GPLs are Turing complete, but there is no such requirement for DSLs. However, this criterion alone is not sufficient since some DSLs may also be Turing complete (e.g., XSLT).

5. Conclusion

The recent popularity of DSLs has created some confusion and misconceptions regarding the specific classification of each language. This paper has suggested several criteria that can be used to assist in classifying a DSL as a programming or modeling language. However, each criterion alone is not sufficient to classify a language – we suggest that a set of criteria together can better inform the determination of the language type.

There are some tangible benefits to providing such a classification. There are several thousand computer languages that have been developed over the history of computing. It is quite natural to classify these languages into different classes or groups instead of remembering the features and characteristics of each language. If the group to which a particular language belongs is known, then some general knowledge about that particular language is immediately available. Such a taxonomy would also help to organize existing knowledge about computer languages into hierarchical rankings in order to have improved understanding and better communication among researchers. Hence, an effective and correct classification about the type of languages enables developers to have a more precise understanding about the properties of a language so that a wise choice of using a language can be made for a certain problem domain. In addition, when extending a language to build a software engineering tool, knowing its type and essential characteristics is very important.

Acknowledgements

This work supported by NSF CAREER award CCF-0643725 and NSF award CCF-0811630.

Table 1: Summary of classification of various DSLs according to suggested criteria

	VHDL	EBNF	ATL	KM3	ESML	SQL	XSLT
Graphical Notation	textual ⇒ P	textual ⇒ P	textual ⇒ P	textual ⇒ P	graphical ⇒ M	textual => P	textual => P
Language Definition	BNF, informal semantics ⇒ P	BNF, informal semantics ⇒ P	BNF, informal semantics ⇒ P	BNF, operational semantics ⇒ P	metamodel, model interpreter ⇒ M	BNF, formal semantics [21] ⇒ P	XML schema, formal [27] semantics ⇒ P
Language Executability	executable ⇒ P	not directly executable ⇒ M	executable ⇒ P	not executable ⇒ M	not directly executable ⇒ M	executable ⇒ P	executable ⇒ P
Level of Abstractions	low ⇒ P	high ⇒ M	high/medium (both declarative and imperative) ⇒ M	high ⇒ M	high ⇒ M	medium (both declarative and imperative) ⇒ M	medium ⇒ M
Fundamental Concepts	values, storage, bindings, abstract ⇒ P	entities, connections ⇒ M	values, storage, bindings, abstract ⇒ P	entities, connections ⇒ M	entities, connections, ports ⇒ M	values, storage, bindings ⇒ P	values, storage, bindings ⇒ P
Development Phase	implementation phase ⇒ P	requirement/ design phase ⇒ M	implementation phase ⇒ P	requirement/ design phase ⇒ M	design / implementation ⇒ M	implementation phase ⇒ P	implementation phase ⇒ P
Multiple Views	no ⇒ P	no ⇒ P	no ⇒ P	no ⇒ P	yes ⇒ M	no ⇒ P	no ⇒ P
Conclusion	programming language	modeling language	programming language	modeling language	modeling language	programming language	programming language

References

[1] P. J. Ashenden. *The Designer's Guide to VHDL*. 2nd Edition, Morgan Kaufmann, 2002.

[2] G. Booch, J. Rumbaugh, and I. Jacobson. *The Unified Modeling Language User Guide*. Addison-Wesley, 1999.

[3] M. Burnett and M. Baker. A classification system for visual programming languages. *Journal of Visual Languages and Computing*, 5(3):287-300, 1994.

[4] J. Cuadrado and J. Molina. Building domain-specific languages for model driven development. *IEEE Software*, 24(5):48-55, 2007.

[5] C. J. Date and H. Darwen. *A Guide to SQL Standard*, 4th Edition, Addison-Wesley, 1997.

[6] GME. http://www.isis.vanderbilt.edu/projects/gme/

[7] C. Gonzalez-Perez and B. Henderson-Sellers. Modelling software development methodologies: A conceptual foundation. *Journal of Systems and Software*, 80(11):1778-1796, 2007.

[8] J. Gray, J.-P. Tolvanen, S. Kelly, A. Gokhale, S. Neema, and J. Sprinkle. Domain-specific modeling. In P. A. Fishwick, editor, *CRC Handbook of Dynamic System Modeling*. CRC Press, 2007. pp. 7-1 through 7-20.

[9] J. Greenfield, K. Short, S. Cook, and S. Kent. *Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools*. Wiley, 2004.

[10] G. Gupta. CS6371 Lecture Notes: Advanced Programming Languages. http://www.utdallas.edu/~gupta/courses/apl/lec1.html

[11] C. A. R. Hoare. Hints on programming language design. Stanford Artificial Intelligence Laboratory Memo AIM-224 / STAN-CS-73-403, 1973.

[12] ISO/IEC 14977:1996(e) International Standard EBNF Syntax Notation.

[13] F. Jouault, F., F. Allilaire, J. Bézivin, and I. Kurtev. ATL: A model transformation tool. *Science of Computer Programming*, 72(1-2):31-39, 2008.

[14] F. Jouault, and J. Bézivin. KM3: A DSL for Metamodel Specification. *International Conference on Formal Methods for Open Object-Based Distributed Systems*, Bologna, Italy, pp. 171-185, 2006.

[15] G. Karsai, S. Neema, and D. Sharp. Model-driven architecture for embedded software: A synopsis and an example. *Science of Computer Programming*, 73(1): 26-38, 2008.

[16] J. Kramer. Is abstraction the key to computing? *Communications of the ACM*, 50(4): 36-42, 2007.

[17] T. Kühne. Matters of (meta-) modeling. *Software and Systems Modeling*, 5(4):369-385, 2006.

[18] K. C. Louden. *Programming Languages: Principles and Practice*. 2nd Edition, Thomson - Course Technology, 2003.

[19] S. Mellor and M. Balcer. *Executable UML*. Addison-Wesley, 2002.

[20] M. Mernik, J. Heering, and A. M. Sloane. When and how to develop domain-specific languages. *ACM Computing Surveys*, 37(4):316-344, 2005.

[21] M. Negri, G. Pelagatti, and L. Sbattela. Formal semantics of SQL queries. *ACM Transactions on Database Systems*, 16 (3):513-534, 1991.

[22] F. G. Pagan. *Formal Specification of Programming Languages: A Panoramic Primer*. Prentice Hall, 1981.

[23] R. Paige, J. Ostroff, and P. Brooke. Principles for modeling language design. *Information and Software Technology*, 42(10):665-675, 2000.

[24] D. Schmidt. Guest editor's introduction. Model-driven engineering. *IEEE Computer*, 39(2):25-31, 2006.

[25] R. Sethi. *Programming Languages: Concepts and Constructs*. 2nd Edition, Addison-Wesley, 1996.

[26] XSLT. http://www,w3.org/TR/xslt.html

[27] P. Wadler. A formal semantics of patterns in XSLT, *Markup Technologies*, Philadelphia, USA, 1999.

[28] D. Watt. *Programming Language Concepts and Paradigms*. Prentice-Hall, 1990.

[29] Wikipedia. Programming languages, http://en.wikipedia.org/wiki/Category:Programming_languages