A Language Description is More than a Metamodel

Anneke Kleppe[1],

University Twente, Netherlands
a.kleppe@utwente.nl

Abstract. Within the context of (software) language engineering, language descriptions are considered first class citizens. One of the ways to describe languages is by means of a metamodel, which represents the abstract syntax of the language. Unfortunately, in this process many language engineers forget the fact that a language also needs a concrete syntax and a semantics. In this paper I argue that neither of these can be discarded from a language description. In a good language description the abstract syntax is the central element, which functions as pivot between concrete syntax and semantics. Furthermore, both concrete syntax and semantics should be described in a well-defined formalism.

1 Introduction

In recent years attention for languages used in the development of software has grown. In [1] I have used the term *software languages* to indicate either a modeling or programming language. I will use the same term in this paper to address the overall group of languages used in the development of software. No matter whether a software language is a modeling, programming, mark-up, or any other kind of language, it is created artificially as opposed to natural languages, which evolution and growth can be called organic.

A consequence of the artificial nature of software languages is that its description is important and becomes an instrument for several tasks. First, the language description is used as input for the process in which the language designer creates a set of supporting tools for the language user. Second, the language user is guided by the language description in her understanding of a program/model written in that language. Language descriptions should therefore be created with utmost care.

One of the ways to describe languages is by means of a metamodel, which represents the abstract syntax of the language. Unfortunately, when using metamodeling many language engineers forget the fact that a language also needs a concrete syntax and a semantics. Some language descriptions contain not even the slightest hint on either topic, for instance, the OMG's Action Semantics [2] consists of an abstract syntax definition only. Others contain only rudimentary information on these two topics, often written in natural language, the most well-known example is of course the UML [3].

1. The author is employed in the GRASLAND project funded by the Dutch NWO (project number 612.063.408).

In this paper I describe what the components of a sound language description should be and some of the formalisms in which these elements can be expressed. I also show the mutual relationships between the elements of a language description and argue that the abstract syntax model is the central element, which acts as a pivot between the concrete syntax description and the semantics descripttion.

Section 2 of this paper sets the context for the discourse on language engineering. Section 3 explains the importance of concrete syntax, whereas Section 4 addresses the topic of semantics. Section 5 summarizes the elements of a solid language description and Section 6 concludes this paper.

2 The Context

Before focusing on what a sound language description should look like, let us first set up the context of our discourse. When dealing with language engineering, obviously we need to have a definition of language. Fortunately, in the formal language theory there is an excellent, simple definition of what a language is (see for instance [4]). I adhere to this definition, although stated a little differently.

Definition 1 *(Language) A language L is the set of all linguistic utterances of L.*

Obviously, we still need to define the concept of *linguistic utterance*. This term stems from natural language research where it represents any expression in a certain language, for instance a word, a sentence, or a conversation, see for instance [5, 6]. Some of the words that have been used in computer science with a similar meaning are: sentence, expression, statement, program, model. Unfortunately, like many other terms in our field, these terms have completely different meaning for different people. Therefore I use the term *linguistic utterance* when I want to be formal, and to avoid any bias I use the word *mogram*, which is short for model/program, in other cases.

Now we are ready to focus on the language description, which I define as follows.

Definition 2 *(Language Description) A language description of language L is the set of rules according to which the linguistic utterances of L are structured, optionally combined with a description of the intended meaning of the linguistic utterances.*

Although I have the opinion that a language description is not complete without a description of the language's semantics[1], I know that there are people that disagree. To accommodate some freedom of mind, the above definition states that the semantics description is optional.

1. What's the use of speaking a syntactically perfect Russian sentence, when you do not know what it means? You might get slapped in the face, because of the abusive meaning of the sentence. Most humans prefer to speak sentences that both speaker and listener can understand, even if the sentences are syntactically flawed.

3 The Divide: Abstract and Concrete Syntax

According to Definition 2 a sound language description minimally includes the syntactical rules according to which a mogram is structured. However, syntax can and should be divided into two: the concrete syntax and the abstract syntax, because the superficial structure of a mogram might be completely different from the underlying structure. In the words of the famous Noam Chomsky: "Surface structure does not necessarily provide an accurate indication of the structure and relations that determine the meaning of a sentence" ([7] on page 93).

With regard to the divide between concrete and abstract syntax we witness a peculiar bias, either towards the concrete or towards the abstract. For instance, when a language description takes the form of a (BNF) grammar, the concrete syntax takes the upperhand. On the other hand, when a language description takes the form of a metamodel, the abstract syntax is overvalued.

BNF as Formalism for Language Descriptions. When a language description is written in BNF, the focus of the language designer is biassed towards the concrete. A well-known fact is that a grammar contains superfluous information, like keywords and ordering, that is not relevant to understand a particular mogram. Therefore, in the process of parsing a distinction is being made between a parse tree and an abstract syntax tree. Although the BNF rules specify the parse tree in a formal way, the abstract syntax tree is completely undefined. Most of the time it can be more or less directly derived from the parse tree. But certainly when an attributed grammar is being used, as is often the case in the field of visual language research (see for instance [8, 9]), the abstract syntax tree can differ greatly from the parse tree, as it is usually built in the form of a single attribute. More importantly, using the formalism of BNF, there is no adequate way to write an abstract syntax description.

Metamodeling as Formalism for Language Descriptions. When a language description is written in the form of a metamodel, the abstract syntax is the centre of focus. However, a metamodel does not contain information on how the concepts in the metamodel are to be represented to the language user. This is sometimes amended by introducing an attributed form of metamodel in which each abstract concept holds an attribute that specifies its notation, for instance a graphical symbol. Many language engineering tools based on graph grammars (e.g. [10, 11, 12]) take this approach. Remembering the quote from Chomsky, immediately the flaw of this approach is clear: the concrete syntax is forced to take on the same structure as the abstract syntax.

3.1 Concrete and Abstract Syntax Models: Two Equally Valued Elements

Although it is my opinion that the abstract syntax should play the central role in a language description, the concrete syntax must be treated with equal respect. It is crucial to language design and it deserves to be a separate element within the language description. Furthermore, the language description should at least contain a mapping from concrete to abstract syntax, and preferably also from abstract to concrete syntax.

Some argue that in modern (graphical) language environments there is no need to be specific about concrete syntax. There is no longer a need for a formally defined parser, and as one reviewer remarked: “At the low level information can be exchanged between different computers applications in a concrete-syntax free way using XML, and at the high level, humans can input linguistic utterances into a tool using forms (most tools provide pop up forms for directly manipulating the attributes associated with model elements). Thus, the only thing that remains is the need for renderings of models which are meaningful to humans.” However, because a language is a means of communication, all language users need to agree on the XML schema for interchange as well as on the symbols to be used in rendering the mogram. Furthermore, both the XML schema and the set of symbols for rendering are definitions of (one of) the concrete syntax(es) of the language. If no agreement on concrete syntax would exist, anything could represent anything, and language users would no longer understand each other.

So how to proceed when creating a new language? Before answering this question, let’s have a look at some existing approaches. In [13, 14] for graphical (visual) and textual languages respectively, the description of the concrete syntax and the description of the abstract syntax are separate entities belonging to one language description. Both syntax descriptions may be written in the same formalism, but it is also possible to use different formalisms. Fondement and Baar in [13] use the formalism of metamodeling for both. A separate metamodel representing concrete syntax elements is build and related to the abstract syntax metamodel via a model transformation. xText [14] uses both metamodeling and BNF. From a existing BNF grammar that represents the concrete syntax, a metamodel is generated that represents the abstract syntax.

My approach to textual languages (described in [15]) also uses both metamodeling and BNF. The concrete syntax is partially described by a metamodel, called parse model (PM), which is used to generate a BNF grammar, as depicted in Figure 1. Special here is that the PM is generated from the abstract syntax metamodel. Both generation steps take in user directives to produce the desired output.

In all of these approaches there is a close link between the description of the concrete syntax and the description of the abstract syntax. In the process of creating a language description either one can be chosen as starting point, the other is developed together with the mapping to the first. My preference is to start with the abstract syntax in order not to introduce limitations posed by the concrete syntax in the description of the abstract syntax. A second reason for this preference is that in the case of a textual syntax the expressive power of BNF rules is less than that of a metamodel. A third reason is the fact that languages can have, and often have, multiple concrete syntaxes.

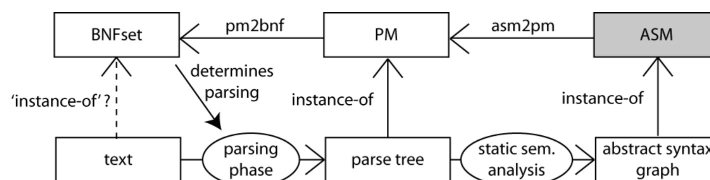

Fig. 1 Separate concrete and abstract syntax definitions.

3.2 Expressive Power of BNF and Metamodels

BNF rules are capable of describing trees, i.e. parse trees. At the same time a metamodel describes graphs, i.e. its instances are graphs. As trees are a subset of graphs, metamodels in general are capable of describing a larger group of instances than sets of BNF rules. Therefore it is better to use a metamodel as starting point and create a BNF grammar from the metamodel, instead of using the more popular, and more easy, direction of creating a metamodel from a grammar.

However, using the difference between non-composite and composite associations, a metamodel can be created in such a way that its instances are graphs in which a tree structure (the composite structure) is recognisable. In the work described in [15] this tree structure is used to generate the PM and ultimately the BNF rule set.

3.3 Multiple Syntaxes

Languages can have, and often have, multiple concrete syntaxes. The mere fact that many languages have a separate interchange format (often XML-based), actually means that they have multiple concrete syntaxes, a 'normal' syntax and an interchange syntax. At the same time there is a growing need for languages that have both a graphical and a textual syntax, as shown by tools like TogetherJ which uses UML as a graphical notation for Java. The UML itself is a good example of multi-syntax language. There is the well-known UML diagram notation [3], which is combined with the Human-readable UML Textual Notation (HUTN) [16] and the interchange format called XMI [17].

A natural consequence of multiple concrete syntaxes is that concrete syntax can no longer be the focus of language design. There has to be a common representation of a linguistic utterance, which is independent of the concrete syntax in which it is entered by or shown to the language user. This common representation is the abstract syntax form of a linguistic utterance. The same form can be used to determine the meaning of the utterance, thus giving the abstract syntax description a pivotal role in the language description.

4 The Semantics

The first thing we need to establish about a semantics description is that it is nothing more than a means to convey the understanding of a mogram from one person to another. Usually this is done not on a per mogram base, but on the base of the constructions in the language in which the mogram is expressed. This is more formally expressed in the following definition.

Definition 3 *(Semantics Description) A description of the semantics of language L is means to communicate a subjective understanding of the linguistic utterances of L to another person or persons.*

A semantics desciption is included in a language description because the language designer wants to, or needs to, communicate your understanding of the language to other persons. An important consequence of this view is that computers are never part of the

audience of a semantics decription. Semantics descriptions of software languages are intended for human consumption, even when they describe the actions of a computer when executing a program. It is very important to explain the intended meaning of a language to other persons as clearly as possible. Similar to writing a paper or giving a presentation, the format of the semantics description should be adapted to its audience, the persons to whom the semantics description is addressed.

4.1 Different Types of Semantics

There are at least four ways in which we can describe the semantics of a software language. (See [18] for a survey of semantic description frameworks.) These are:

- Denotational, that is by constructing mathematical objects (called *denotations* or *meanings*) which represent the meaning of the program/model.
- Operational, that is by describing how a valid mogram is interpreted as sequences of computational steps. The sequence of computational steps is often given in the form of a *state transition system*, which shows how the runtime system progresses from state to state.
- Translational, that is by translating the mogram into another language that is well understood.
- Pragmatic, that is by providing a tool that executes the program/model. This tool is often called a *reference implementation*.

In my opinion, the best way to describe semantics to an audience of computer scientists is either translational or operational. If you can find a good target language, then the translational approach is probably the best. For instance, explaining the semantics of C# to a Java programmer is best done by translating the C# concepts to Java concepts, because the two languages are so alike. When you lack a good target language, the operational approach is second best.

An example of the difference between the translational and operational approaches is a description of the meaning of the English phrase "to cut down a tree". You can either translate this phrase into some other language, like French: "réduire un arbre", or you can make a comic strip showing all the subsequent phases of cutting down a tree (a video would also do nicely). Each individual frame in the video, or picture in the comic strip, would be a state in the state transition system, hence the word *snapshot* that is often used.

4.2 Operational Semantics using Graph Transformations

The operational semantics of a software language describe what happens in a computer when a mogram of that language is executed. Execution means the processing of data. Therefore a semantics description should explain (1) the data being processed, (2) the processes handling the data, and (3) the relationship of these two with the possible linguistic utterances of the language. Together we call 1 and 2 the *runtime system*, (3) is called the *semantic mapping*.

The runtime system can be described using the formalism of metamodeling. For instance, in [19] we used a metamodel to describe what in compiler terminology is known as the heap, the data part of the runtime system, and the stack, the process part of the runtime system. The semantic mapping was in this case given by a set of graph transformation rules. Each rule can be applied only when a certain syntactic construction is present in the mogram and when at the same time a certain runtime state has been reached. For example, the rule describing assignment is executed only when the program counter has reached an assignment statement in the (abstract syntax graph of the) mogram and the value of the expression in the right hand side of the assignment is available on the heap simultaneously.

5 The Components of a Language Description

A sound language description contains the following elements:

1. an abstract syntax description;
2. one or more concrete syntax descriptions;
3. for each concrete syntax description: a mapping from concrete to abstract and preferably from abstract to concrete as well;
4. a description of the semantics in the form of a set of rules that govern an abstract machine that is able to execute any syntactically correct linguistic utterance of the language.

The formalism used to express each of these elements can differ from the one to the other, even within one language description. The use of combinations of formalisms to express the various elements of a language description is a valid approach to language engineering.

Choices for the formalism used to specify the abstract syntax are, for instance, metamodeling and graph grammars. Some of the possibilities for the formalism for concrete syntax description are BNF rules, and again metamodeling. The mappings between concrete and abstract syntax can be described as model transformations, using one of the various formalisms like QVT or ATL. For the semantics component the language designer has an even larger choice. My preference is to use graph transformation rules based on the concepts in the abstract syntax metamodel.

6 Conclusion

A language description that only specifies abstract syntax is not enough. To rephrase the words of the old saying “one swallow does not make a summer”: one metamodel does not make a language. Language designers should take equally care to design concrete syntax as well as semantics.

The act of language design is one in which a carefull balance must be uphold between the three main elements of a language description: abstract syntax, concrete syntax, and semantics. A software language should be build iteratively starting with parts of the abstract syntax, then adding concrete syntax and semantics to these parts. Thus

the result of each iteration will be a sound language description, in which every element takes its due place.

References

[1] Anneke Kleppe. Towards general purpose high level software languages. In Alan Hartman and David Kreische, editors, *Model Driven Architecture – Foundations and Applications*, volume 3748 of *LNCS*, Berlin Heidelberg, November 2005. Springer-Verlag.

[2] OMG. OMG Unified Modeling Language Specification (Action Semantics). Technical Report ptc/02-01-09, OMG, 2002.

[3] OMG. Unified modeling language: Superstructure. Technical Report formal/05-07-04, OMG, 2005.

[4] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979.

[5] Georgios Tserdanelis and Wai Yi Peggy Wong, editors. *The Language Files: Materials for an Introduction to Language and Linguistics*. Ohio State University Press, 9th edition edition, April 2004.

[6] Victoria Fromkin, Robert Rodman, and Nina Hyams, editors. *An Introduction to Language*. Heinle, 7th edition edition, August 2002.

[7] Noam Chomsky. *Language and Mind*. Cambridge University Press, Cambridge, USA, 2006.

[8] Da-Qian Zhang, Kang Zhang, and Jiannong Cao. A context-sensitive graph grammar formalism for the specification of visual languages. *The Computer Journal*, 44(3):186–200, 2001.

[9] Gennaro Costagliola and Giuseppe Polese. Extended positional grammars. *2000 IEEE International Symposium on Visual Languages (VL'00)*, 00:103, 2000.

[10] Roswitha Bardohl. *GenGEd: Visual Definition of Visual Languages based on Algebraic graph Transformation*. PhD thesis, TU Berlin, Berlin, Germany, 1999.

[11] Mark Minas and G. Viehstaedt. Diagen: A generator for diagram editors providing direct manipulation and execution of diagrams. In V. Haarslev, editor, *11th IEEE International Symposium on Visual Languages*, pages 203–210, Los Alamitos/CA/USA, 1995. IEEE Computer Society Press.

[12] Juan de Lara and Hans Vangheluwe. Atom3: A tool for multi-formalism and meta-modelling. In *FASE '02: Proceedings of the 5th International Conference on Fundamental Approaches to Software Engineering*, pages 174–188, London, UK, 2002. Springer-Verlag.

[13] Frédéric Fondement and Thomas Baar. Making metamodels aware of concrete syntax. In Alan Hartman and David Kreische, editors, *ECMDA-FA*, volume 3748 of *Lecture Notes in Computer Science*, pages 190–204. Springer, 2005.

[14] xText. http://www.eclipse.org/gmt/oaw/doc/4.1/r80_xtextReference.pdf, 2007.

[15] Anneke Kleppe. Towards the generation of a text-based IDE from a language metamodel. In David H. Akehurst, Régis Vogel, and Richard F. Paige, editors, *Proceedings of the third European Conference on MDA, 2007*, volume 4530 of *LNCS*, pages 114–129, Berlin Heidelberg, June 2007. Springer-Verlag.

[16] OMG. Human-usable textual notation (HUTN) specification. Technical Report formal/04-08-01, OMG, 2004.
[17] OMG. MOF 2.0/XMI mapping specification, v2.1. Technical Report formal/05-09-01, OMG, 2005.
[18] Yingzhou Zhang and Baowen Xu. A survey of semantic description frameworks for programming languages. *SIGPLAN Notices*, 39(3):14–30, 2004.
[19] Harmen Kastenberg, Anneke Kleppe, and Arend Rensink. Defining object-oriented execution semantics using graph transformations. In R. Gorrieri and H. Wehrheim, editors, *FMOODS 2006*, volume 4037 of *LNCS*, pages 186–201, Berlin Heidelberg, June 2006. Springer-Verlag.