A Model-Driven Parser Generator with Reference Resolution Support

Luis Quesada

Advisors: Fernando Berzal and Juan Carlos Cubero

Department of Computer Science and Artificial Intelligence
CITIC, University of Granada, Granada 18071 Spain
{lquesada,fberzal,jc.cubero}@decsai.ugr.es

ABSTRACT

ModelCC is a model-based parser generator. Model-based parser generators decouple language specification from language processing. This model-driven approach avoids the limitations imposed by parser generators whose language specifications must conform to specific grammar constraints. Moreover, ModelCC supports reference resolution within the language specification. Therefore, it does not parse just trees but it can also efficiently deal with abstract syntax graphs. These graphs can even include cycles (i.e. they are not constrained to directed acyclic graphs).

Categories and Subject Descriptors

D.3.4.h [**Programming Languages**]: Processors—*Parsing*

General Terms

Languages

Keywords

Model-based software development, language specification, parser generators, abstract syntax graphs.

1. INTRODUCTION

Traditional language processing tools constrain language designers to specific kinds of grammars, which are specified by BNF-like productions [1] and tie language specification to specific parsing techniques.

Model-based language specification [3] techniques decouple language design from language processing and automatically generate the language grammar corresponding to the language model, thus making the language design process less arduous.

Syntax analysis defers some analysis tasks to later stages in the language processing pipeline, such as reference resolution and other semantic checks. However, a model-driven parser generator can be employed to automate some of these tasks.

ModelCC [7] is a model-based parser generator that supports references between language elements, thus incorporating into the parsing process the reference resolution that is traditionally hand-crafted with the help of symbol tables.

In this paper, we show how ModelCC [7] is able to resolve references and obtain abstract syntax graphs as the result of the parsing process, rather than the abstract syntax trees returned by traditional parser generators.

2. BACKGROUND

The tight connection between language design and language processing in traditional syntax-driven language processing tools makes language-processor maintenance tedious, time-consuming, and error-prone [2]. Language designers must ensure that their language specification conforms to the specific constraints of the parsing technique used by their parser generator. When a language processor parses a linear text to produce a graph-like data structure, the language designer is forced to specify a standard context-free grammar suitable for its textual input and manually implement the conversion from the resulting parse tree to the desired data structure. Also, whenever the language specification has to be modified, the language designer has to manually propagate changes throughout the entire language processor tool chain. Moreover, when different applications use the same language, their maintenance typically requires keeping several copies of the language specification in sync.

In contrast, model-based language specification allows the specification of a single abstract syntax model (ASM) that represents the core concepts in a language, accompanied by one or several concrete syntax models (CSMs) that constrain the ASM and suit the specific needs of its desired textual or graphical representations. This way, the ASM representing the language can be modified as needed without having to worry about the peculiarities of the chosen parsing technique, since the corresponding language processor will be automatically updated whenever the ASM changes.

A diagram summarizing the traditional language design process is shown in Figure 1. The corresponding diagram for the model-based approach is shown in Figure 2.

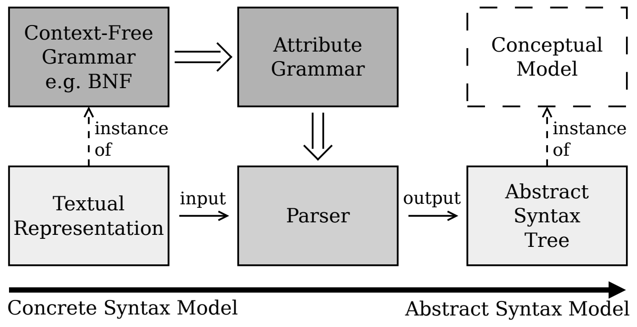

Figure 1: Traditional language processing approach.

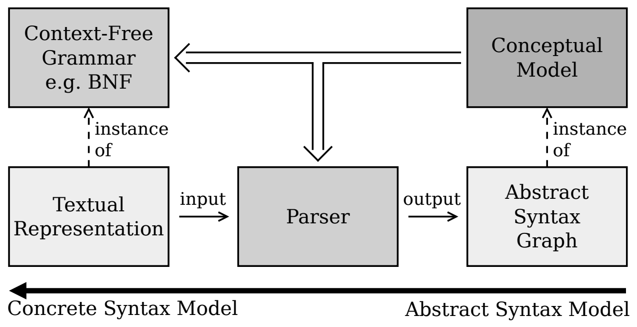

Figure 2: Model-based language processing approach.

ModelCC [7] is a parser generator that supports a model-based approach for the design of language processing systems. Its starting ASM is created by defining classes that represent language elements and establishing relationships among those elements. Once the ASM is specified, metadata annotations can be used to impose constraints over language elements and their relationships, thus producing an ASM-CSM mapping.

When the ASM represents a tree-like structure, a model-based parser generator is equivalent to a traditional grammar-based parser generator in terms of expression power. When the ASM represents non-tree structures (hence the use of the 'abstract syntax graph' term in Figure 2), reference resolution techniques can be employed to make model-based parser generators more powerful than grammar-based ones.

3. MODELCC REFERENCE RESOLUTION

Reference resolution consists of finding the object a reference refers to and, in the case of ModelCC, automatically linking the reference to the corresponding object instantiation. This resolution process is what leads to abstract syntax graphs instead of trees.

In ModelCC, an object reference is embodied by a subset of the elements in its full object definition. This subset of elements acts as an identifier (or key in database terms) that, when found in the input text, can be recognized as a reference to an object in the model and linked to its instantiation in the ASM.

References in ModelCC can be anaphoric, when they are preceded by the object definition, but also cataphoric, when the reference precedes the definition, and even recursive, when they appear within the definition they refer to.

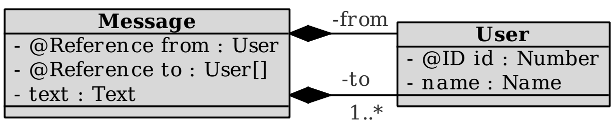

Figure 3: ModelCC specification of *Messages*, their senders, and their receivers.

3.1 The @ID Annotation

ModelCC uses an *@ID* metadata annotation to support the specification of references. Such annotation is applicable to a set of members of a language model element, which determines the syntax of references to particular instances of such elements. That is, any appearance of the same set of values will be considered a reference to the same instance of the referred language element.

The use of references is resolved in the implementation of ModelCC by the introduction of grammar productions that characterize such references and semantic actions that map them to the corresponding language elements.

In Figure 3, the *@ID* annotation is employed to identify users by a single number.

3.2 The @Reference annotation

ModelCC resorts to the *@Reference* metadata annotation to complete its support for reference resolution. Such annotation is applicable to individual members of a language element provided that such element contains at least one *@ID*-annotated member in the language model.

Whenever a language model element member is annotated with *@Reference*, the corresponding grammar productions are modified so that they refer to the symbol corresponding to the element reference specification rather than the symbol that corresponds to its full specification. These productions are associated to a semantic action that resolves the references at the end of the parsing process, in order to support cataphoric and recursive references, apart from the anaphoric references that could be resolved during the parsing process.

In Figure 3, the textual syntax of messages includes numbers that, as identifiers, refer to a particular users. ModelCC will parse such identifiers, recognize the references, resolve them, and return the correct object graph.

4. A WORKING CASE STUDY

In this section, we present an example language that allows the specification and rendering of complex 3D objects using the reference resolution capabilities of ModelCC. The UML class diagram in Figure 4 shows our annotated 3D object specification language model.

The reference support extension we propose in this paper can be observed in the *Definition*, *ObjectName*, and *Defined-Object* classes. The *name* member of the *Definition* class is

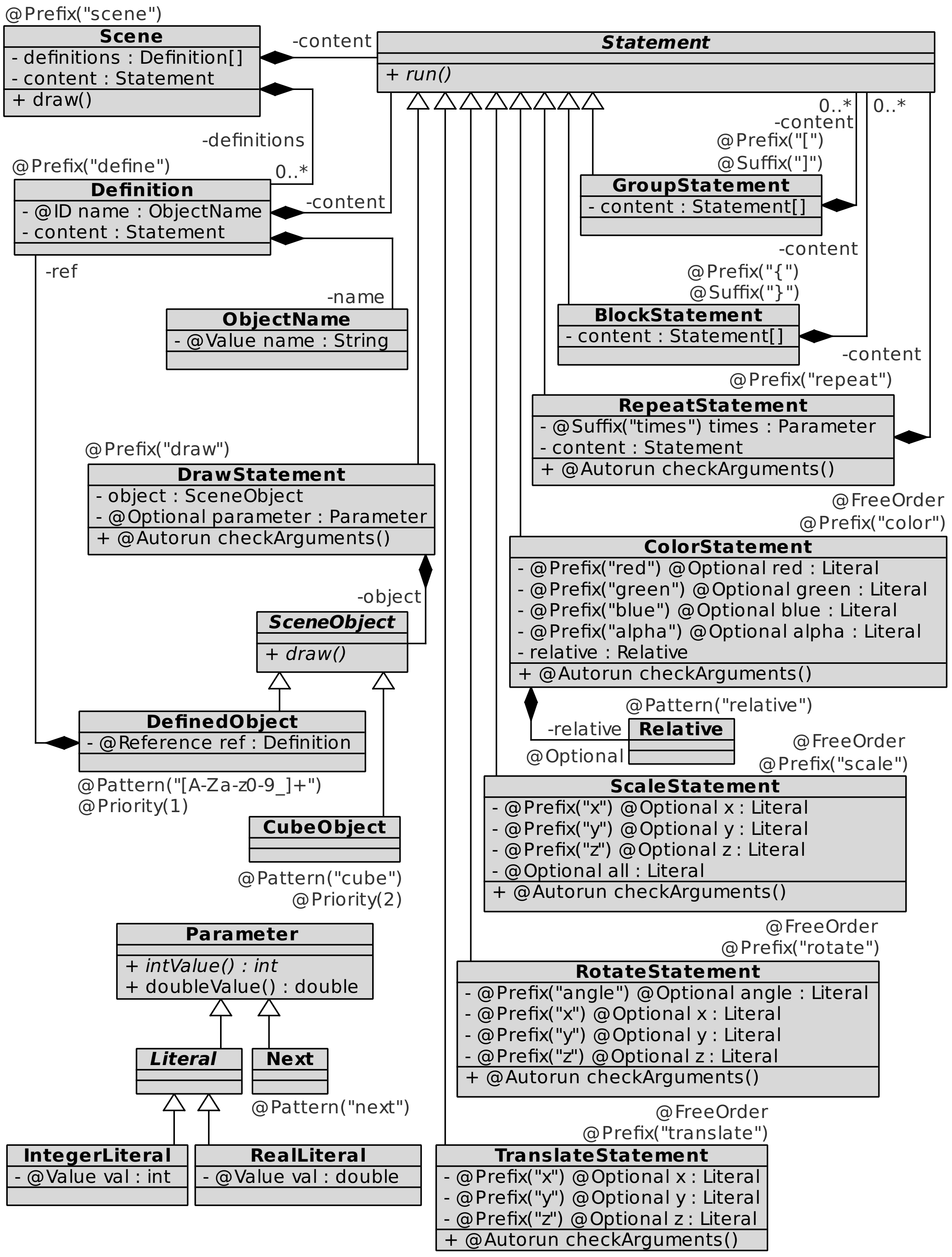

Figure 4: ModelCC definition of a 3D object specification language. ModelCC reference resolution support is used to allow the specification of complex 3D objects in the *Definition* class.

```
define snail [
  draw cube
  {
    scale .3
    color blue 1
    repeat 6 times [
      draw cube
      translate y 1
      rotate z 1 angle -5
      color relative alpha -.06
    ]
  }
  translate x .8
  rotate z 1 angle 10
  scale .98
  color relative red -.05 green +.05 alpha -.008
  draw snail next
]
scene [
  color red 1
  draw snail 400
]
```

Figure 5: Snail specification in our 3D object specification language.

annotated with *@ID*, which means that a *Definition* instance can be identified by an *ObjectName*. Then, the *ref* member of a *DefinedObject* is annotated with *@Reference*, which means that, in textual form, a *DefinedObject* can refer to a *Definition* by its *ObjectName*. ModelCC reference resolution allows references to be resolved during the parsing process and makes the implementation of a traditional symbol table unnecessary.

ModelCC is able to automatically generate a grammar from the ASM defined by a class model and the ASM-CSM mapping defined as a set of metadata annotations on the class model. References in that grammar are automatically resolved by ModelCC so that further work is not needed.

Figures 5 and 6 illustrate the specification and rendering of a 3D snail in our 3D object specification language. The *snail* object is defined as a single section of the snail consisting of the shell and a blue strip, and a slightly smaller, more transparent, and more greenish snail. The scene consists of a 400-section *snail* object.

5. CONCLUSIONS

We have described how the ModelCC model-based parser generator, which employs metadata annotations to implement ASM-CSM mappings, supports reference resolution and allows parsing abstract syntax graphs rather than abstract syntax trees.

6. FUTURE WORK

In the future, ModelCC will incorporate a wider variety of parsing techniques and it will be able to automatically determine the most efficient parsing algorithm that is able to parse a particular language (the current version employs the Fence parsing algorithm [5] on top of the Lamb scanning algorithm [4, 6]).

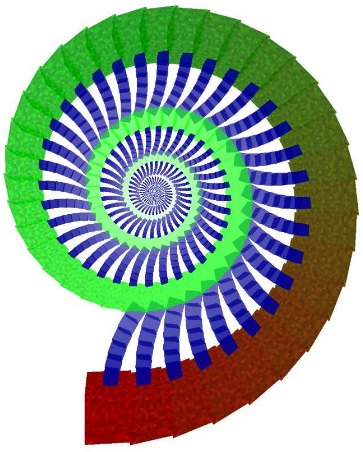

Figure 6: Rendering of the snail specified by the input text shown in Figure 5.

ModelCC will also be extended to support multiple concrete syntax models for the same abstract syntax model.

We plan to study the possibilities tools such as ModelCC open up in different application domains, including traditional language processing systems, model-driven software development (MDSD) tools, natural language processing in restricted domains, domain-specific languages (DSLs) and language workbenches, model induction, text mining applications, data integration, and information extraction.

Acknowledgements

Work supported by research project TIN2009-08296.

7. REFERENCES

[1] R. S. Alfred V. Aho, Monica S. Lam and J. D. Ullman. *Compilers: Principles, Techniques and Tools.* Pearson Education, Inc., 2nd edition, 2006.

[2] L. C. L. Kats, E. Visser, and G. Wachsmuth. Pure and declarative syntax definition: Paradise lost and regained. In *Proc. OOPSLA'10*, pages 918–932, 2010.

[3] A. Kleppe. Towards the generation of a text-based IDE from a language metamodel. In *Proc. MDAFA'07. LNCS*, volume 4530, pages 114–129, 2007.

[4] L. Quesada, F. Berzal, and F. J. Cortijo. Lamb — A lexical analyzer with ambiguity support. In *Proc. ICSOFT'11*, volume 1, pages 297–300, 2011.

[5] L. Quesada, F. Berzal, and F. J. Cortijo. Fence — A context-free grammar parser with constraints for model-driven language specification. In *Proc. ICSOFT'12*, 2012. (in press).

[6] L. Quesada, F. Berzal, and F. J. Cortijo. Parallel finite state machines for very fast distributable regular expression matching. In *Proc. ICSOFT'12*, 2012. (in press).

[7] L. Quesada, F. Berzal, and J.-C. Cubero. A language specification tool for model-based parsing. In *Proc. IDEAL'11. LNCS*, volume 6936, pages 50–57, 2011.