A Domain-Specific Language for Abstract Syntax Model to Concrete Syntax Model Mappings

Luis Quesada, Fernando Berzal, and Juan-Carlos Cubero
Department of Computer Science and Artificial Intelligence,
University of Granada, CITIC, 18071, Granada, Spain
{lquesada, fberzal, jc.cubero}@decsai.ugr.es

Keywords:
Model-driven software development, language specification, parser generators, abstract syntax model, concrete syntax model.

Abstract:
Model-based parser generators such as ModelCC effectively decouple language design from language processing. ModelCC allows the specification of the abstract syntax model of a language as a set of language elements and their relationships. ModelCC provides the necessary mechanisms to specify the mapping from the abstract syntax model (ASM) to a concrete syntax model (CSM). This mapping can be specified as a set of metadata annotations on top of the abstract syntax model itself or by means of a domain-specific language (DSL). Using a domain-specific language to specify the mapping from abstract to concrete syntax models allows the definition of different concrete syntax models for the same abstract syntax model. In this paper, we describe the ModelCC domain-specific language for ASM-CSM mappings and we showcase its capabilities by using the ModelCC ASM-CSM DSL to define itself.

1 INTRODUCTION

Model-based language specification techniques [Kleppe, 2007] decouple language design from language processing and automatically generate the corresponding language grammar, thus making the language design process less arduous.

ModelCC is a model-based parser generator [Quesada et al., 2011, Quesada, 2012] that allows the specification of the abstract syntax model of a language as a set of classes, which represent language elements, and relationships between those classes or language elements.

ModelCC allows mapping the abstract syntax model to concrete syntax models by imposing constraints over language elements and their relationships using either metadata annotations or a domain-specific language for the specification of language constraints.

In this paper, we propose the ModelCC domain-specific language for abstract syntax model to concrete syntax model mappings (from now on referred as the ModelCC DSL for ASM-CSM mappings) and present its specification in a model-based way using ModelCC. This domain-specific language ultimately allows model-based parser generators to decouple abstract syntax models from concrete syntax models.

Section 2 introduces model-based language specification and the ModelCC model-based parser generator. Section 3 describes ModelCC the ModelCC domain-specific language for ASM-CSM mappings. Finally, Section 4 presents our conclusions and future work.

2 MODEL-BASED LANGUAGE SPECIFICATION

Most existing language specification techniques [Aho et al., 2006] require the language designer to provide a textual specification of the language grammar. The proper specification of such a grammar is a nontrivial process that depends on the lexical and syntax analysis techniques to be used, since each kind of technique requires the grammar to comply with a specific set of constraints. Each analysis technique is characterized by its expression power and this expression power determines whether a given analysis technique is suitable for a particular language. The most significant constraints on formal language specification originate from the need to consider context-sensitivity, the need to perform an efficient analysis, and some techniques' inability to resolve conflicts caused by grammar ambiguities.

In practice, when we want to build a complex data structure from an input codified using a specific syntax, the implementation of the mandatory language processor requires the software engineer to build a grammar-based language specification for the input data and also to implement the conversion from the parse tree returned by the parser to the desired data structure, which is an instance of the data model.

Whenever the language specification has to be modified, the language designer has to manually propagate changes throughout the entire language processor tool chain, from the specification of the grammar defining the formal language (and its adaptation to specific parsing tools) to the corresponding data model. These updates are time-consuming, tedious, and error-prone. By making such changes labor-intensive, the traditional language processing approach hampers the maintainability and evolution of the language used to represent the data [Kats et al., 2010].

Moreover, it is not uncommon for different applications to use the same language. For example, the compiler, different code generators, and other tools within an IDE, such as the editor or the debugger, typically need to grapple with the full syntax of a programming language. Unfortunately, their maintenance typically requires keeping several copies of the same language specification synchronized.

The idea behind model-based language specification is that, starting from a single abstract syntax model (ASM) that represents the core concepts in a language, language designers can develop one or several concrete syntax models (CSMs). These CSMs can suit the specific needs of the desired textual or graphical representation. The ASM-CSM mappings can be performed, for instance, by annotating the abstract syntax model with the constraints needed to transform the elements in the abstract syntax into their concrete representation.

This way, the ASM representing the language can be modified as needed without having to worry about the language processor and the peculiarities of the chosen parsing technique, since

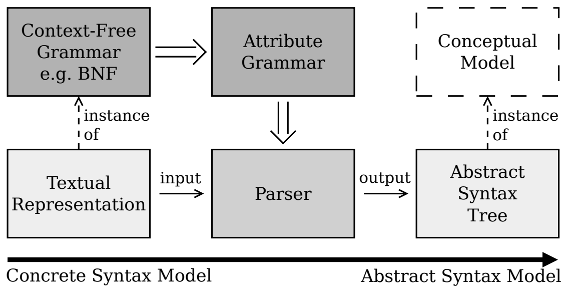

Figure 1: Traditional language processing.

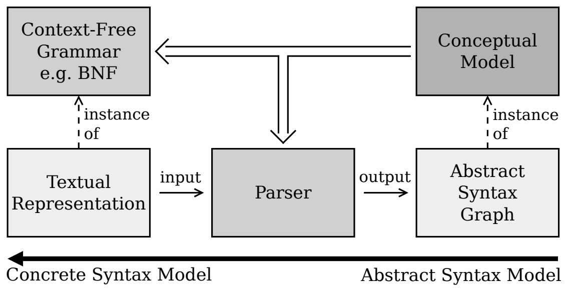

Figure 2: Model-based language processing.

the corresponding language processor will be automatically updated. In this case, the language designer does not have to manually propagate changes throughout the language processor tool chain. Also, when different applications use the same language, there is no need to keep or maintain duplicate language models.

Finally, as the ASM is not bound to a particular parsing technique, evaluating alternative and/or complementary parsing techniques is possible without having to propagate their constraints into the language model. Therefore, by using an ASM, model-based language specification completely decouples language specification from language processing, which can be performed using whichever parsing techniques are suitable for the formal language implicitly defined by the abstract model and its concrete mapping.

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

It should be noted that ASMs may represent non-tree structures. Hence the use of the 'abstract syntax graph' term in Figure 2.

ModelCC is a parser generator that supports a model-based approach to the design of language processing systems [Quesada et al., 2011, Quesada, 2012].

Its starting ASM is created by defining classes that represent language elements and establish-

Constraints on...	Annotation	Function
...patterns	@Pattern	Pattern matching definition of basic language elements.
	@Value	Field where the recognized input element will be stored.
...delimiters	@Prefix	Element prefix(es).
	@Suffix	Element suffix(es).
	@Separator	Element separator(s) in lists of elements.
...cardinality	@Optional	Optional elements.
	@Minimum	Minimum element multiplicity.
	@Maximum	Maximum element multiplicity.
...evaluation order	@Associativity	Element associativity (e.g. left-to-right).
	@Composition	Eager or lazy composition for nested composites.
	@Priority	Element precedence level/relationships.
...composition order	@Position	Define an element member position relative to other.
	@FreeOrder	All the element members positions may vary.
...references	@ID	Identifier of a language element.
	@Reference	Reference to a language element.
Custom constraints	@Constraint	Custom user-defined constraint.

Table 1: The metadata annotations supported by the ModelCC model-based parser generator.

ing relationships among those elements. Once the ASM is established, constraints can be imposed over language elements and their relationships as annotations in order to produce the desired ASM-CSM mappings.

The ASM is built on top of basic language elements, which can be viewed as the tokens in the model-driven specification of a language. ModelCC provides the necessary mechanisms to combine those basic elements into more complex language constructs, which correspond to the use of concatenation, selection, and repetition in the syntax-driven specification of languages.

3 MODELCC DSL FOR ASM-CSM MAPPINGS

In ModelCC, the constraints imposed over ASMs to map them to particular CSMs can be declared as metadata annotations on the model itself. Now supported by all the major programming platforms, metadata annotations are often used in reflective programming and code generation [Fowler, 2002]. Table 1 summarizes the set of constraints supported by ModelCC.

However, in order to allow the developer to specify several mappings, ModelCC also allows the specification of separate input files corresponding to separate sets of constraints by using the ModelCC DSL for ASM-CSM mappings.

In this section, we describe the ModelCC DSL for ASM-CSM mappings. We provide the ModelCC implementation of a parser for the DSL as an ASM complemented with annotations.

Finally, as an example of the usage of the language, we also provide the ModelCC implementation of a parser for the DSL as an ASM complemented with constraint specifications written in the DSL itself.

Subsection 3.1 outlines the language features. Subsection 3.2 provides the definition of the language as an ASM complemented with metadata annotations. Subsection 3.3 provides several equivalent definitions of the language as an ASM complemented with constraint specification files written in the language itself.

3.1 Language Features

The ModelCC DSL for ASM-CSM mappings supports the following features:

- The definition of constraints on patterns, delimiters, evaluation order, and references to language elements.
- The property-like specification of constraints for language elements and their members.
- The grammar-like specification of the concrete syntax of language elements by means of a regular-expression-like language.

While the semantics of property-like constraint definitions is equivalent to that of metadata annotation constraint definitions, grammar-like constraint specification allows for a more intuitive specification of ASM-CSM mappings.

Grammar-like constraint definitions may be more intuitive to traditional language designers who are familiar with syntax-driven language specification tools. Such constraint definitions can be redundant with the ASM as, for example, they can also include multiplicity constraints. ModelCC checks and reports if any syntax implicit in grammar-like constraint definitions conflicts with the language ASM.

Finally, ModelCC checks, reports, and ignores any constraints on language elements on language element members that do not exist.

3.2 ModelCC Definition of the DSL for ASM-CSM mappings

The ASM of the language is designed first. Then, it is mapped to a CSM by imposing constraints using metadata annotations on the model classes.

The resulting model, depicted as an UML class diagram in Figure 3, can be processed by ModelCC to generate the corresponding parser.

This Figure demonstrates the need of an alternative way of specifying constraints:

- When metadata annotations are used to define CSMs on top of the ASM, the concrete syntax is interleaved in the abstract syntax model in a way that burdens it, similar to language processing being coupled with language specification in traditional syntax-driven language specification techniques
- Also, separate CSMs cannot be defined on top of the ASM using metadata annotations.

3.3 Separating ASM and CSM

Once an initial implementation of the ModelCC DSL for ASM-CSM mappings provides a bootstrap, we provide implementations of the language that consist of an ASM and separate constraint definitions using the language itself.

The bare model is depicted as an UML class diagram in Figure 4.

Starting from this ASM, we provide three different ASM-CSM mappings for the language.

- **Grammar-like specification** Figure 5 presents a grammar-like constraint set specified using the ModelCC DSL for ASM-CSM mappings.

  Some of the advantages of grammar-like mappings can be observed in the specification of the *ConstraintDefinition* language element constraints. A single constraint specification can include prefix constraints, suffix constraints, and language element member order constraints. Also, the specification of the *ConstraintDefinition* language element constraints includes two multiplicity constraints (optionality, represented by the regex-like "?" operator) that are redundant with the ASM. ModelCC checks these multiplicity constraints for consistency with the ASM and reports any conflict in parser generation time.

  Another illustrative case of grammar-like mappings can be observed in the specification of the *Element* language element constraints. Although its member *name* is defined as a list in the ASM, the grammar-like constraint specification uses a classical explicit-list specification to specify the separator for list members.

- **Property-like specification** Figure 6 presents a property-like constraint set specified using the ModelCC DSL for ASM-CSM mappings.

  The property-like specification of ASM-CSM mappings mimics the specification of constraints on ASMs using metadata annotations. It can be observed that the constraints are specified as properties of language elements.

- **Mixed specification** Figure 7 presents another equivalent constraint set specified using the ModelCC DSL for ASM-CSM mappings.

  In this case, some constraints are specified grammar-like and some constraints are specified property-like. For example, separators in lists are specified using property-like constraint definitions, which may seem more intuitive to some language designers.

  It should be noted that constraint definitions differ from grammar rules in that several of them can be specified for separate members of the same language element, as can be observed in the *ConstraintDefinition* language element.

Finally, it should be noted that ASMs that are complemented with metadata annotations can be complemented with files written in the ModelCC DSL for ASM-CSM mappings.

Metadata annotation constraints represent default values that apply, unless otherwise specified, to all the ASM-CSM mappings of a language.

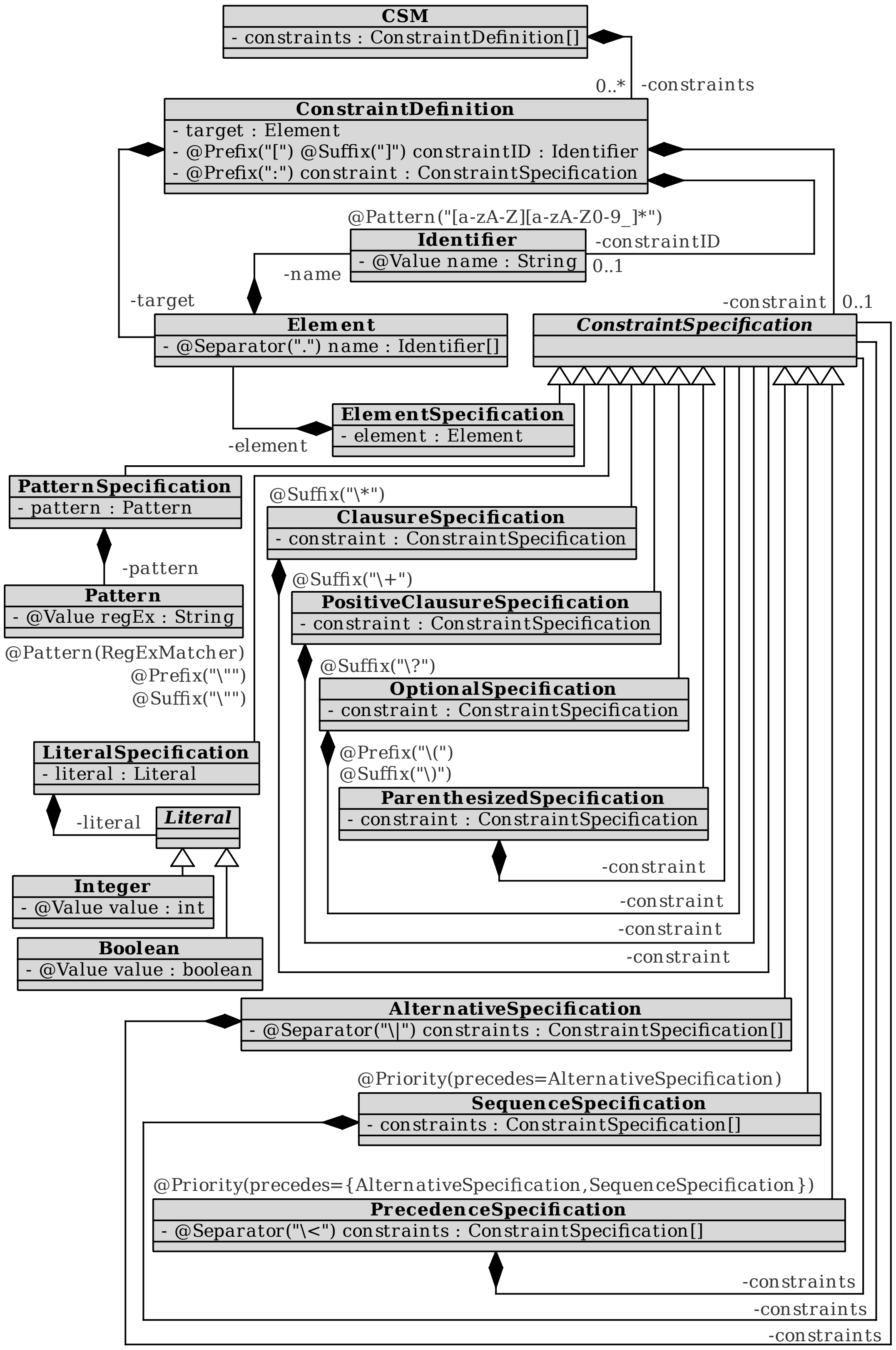

Figure 3: Definition of the ModelCC DSL for ASM-CSM mappings in ModelCC.

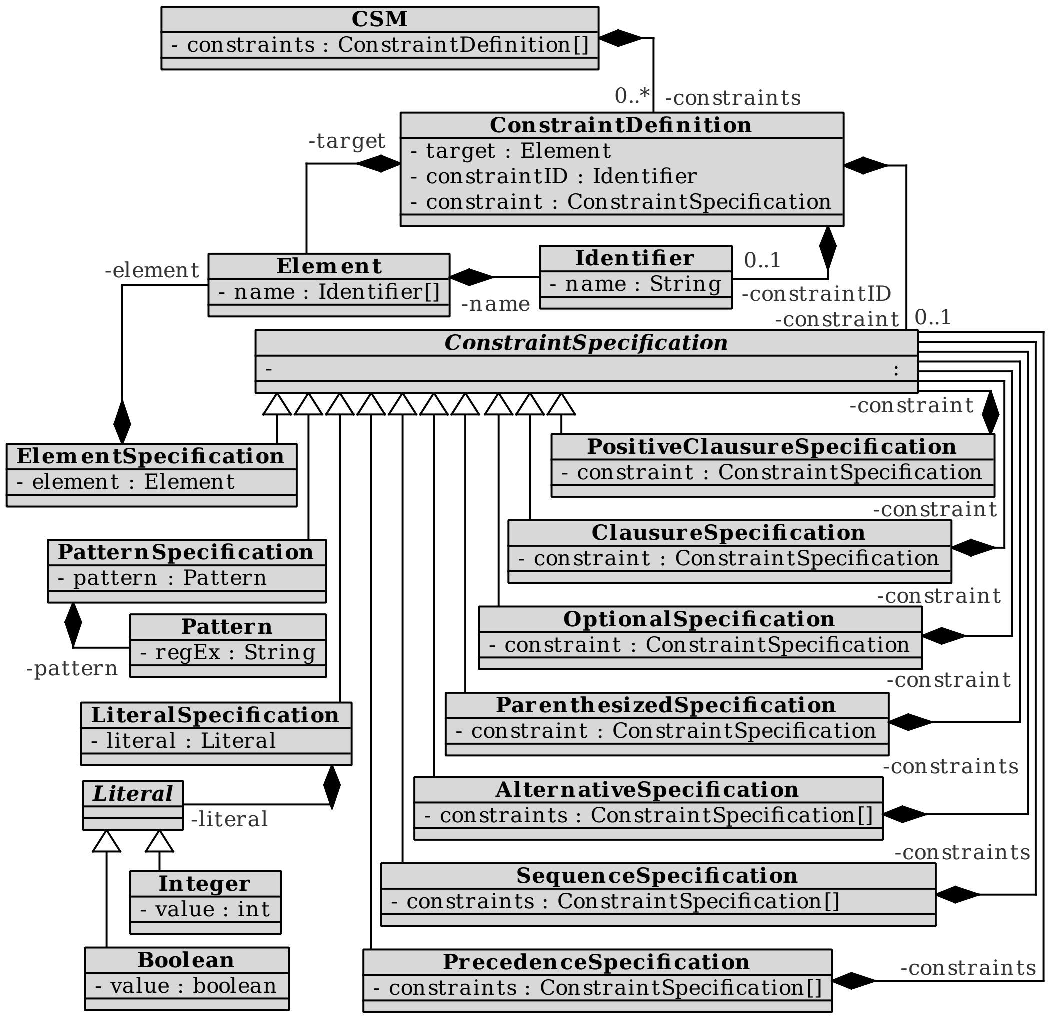

Figure 4: Definition of the abstract syntax model of the ModelCC DSL for ASM-CSM mappings in ModelCC.

```
ConstraintDefinition: target ("[" constraintID "]")? (":" constraint)?
Element: name ("." name)*
Identifier.name: "[a-zA-Z][a-zA-Z0-9_]*"
ClausureSpecification: constraint "\*"
OptionalSpecification: constraint "\?"
PositiveClauseSpecification: constraint "\+"
ParenthesizedSpecification: "\(" constraint "\)"
ConstraintSpecification: SequenceSpecification < PrecedenceSpecification
                         < AlternationSpecification
AlternationSpecification: constraints ("\|" constraints)*
PrecedenceSpecification: constraints ("\<" constraints)*
Boolean.value: "true|false"
Integer.value: "[0-9]+"
```

Figure 5: Grammar-like specification of the mapping from the abstract syntax model to the concrete syntax model of ModelCC DSL for ASM-CSM mappings, written in the ModelCC DSL for ASM-CSM mappings itself.

```
ConstraintDefinition.constraintID[prefix] "\["
ConstraintDefinition.constraintID[suffix] "\]"
ConstraintDefinition.constraint[prefix]: ":"
Element.name[separator]: "."
Identifier.name: "[a-zA-Z][a-zA-Z0-9_]*"
ClausureSpecification[suffix]: "\*"
OptionalSpecification[suffix]: "\?"
PositiveClauseSpecification[prefix]: "\+"
ParenthesizedSpecification[prefix]: "\("
ParenthesizedSpecification[suffix]: "\)"
SequenceSpecification[precedes]: AlternationSpecification
                                  PrecedenceSpecification
ConstraintSpecification: SequenceSpecification < PrecedenceSpecification
AlternationSpecification.constraints[separator]: "\|"
PrecedenceSpecification[precedes]: AlternationSpecification
PrecedenceSpecification.constraints[separator]: "\<"
Boolean.value: "true|false"
Integer.value: "[0-9]+"
```

Figure 6: Property-like specification of the mapping from the abstract syntax model to the concrete syntax model of ModelCC DSL for ASM-CSM mappings, written in the ModelCC DSL for ASM-CSM mappings itself.

```
ConstraintDefinition: "[" constraintID "]"
ConstraintDefinition: ":" constraint
Element.name[separator]: "."
Identifier.name: "[a-zA-Z][a-zA-Z0-9_]*"
ClausureSpecification: constraint "\*"
OptionalSpecification: constraint "\?"
PositiveClauseSpecification: constraint "\+"
ParenthesizedSpecification: "\(" constraint "\)"
ConstraintSpecification: SequenceSpecification < PrecedenceSpecification
                         < AlternationSpecification
AlternationSpecification.constraints[separator]: "\|"
PrecedenceSpecification.constraints[separator]: "\<"
Boolean.value: "true|false"
Integer.value: "[0-9]+"
```

Figure 7: Mixed specification of the mapping from the abstract syntax model to the concrete syntax model of ModelCC DSL for ASM-CSM mappings, written in the ModelCC DSL for ASM-CSM mappings itself.

4 CONCLUSIONS AND FUTURE WORK

ModelCC is a model-based parser generator that allows using metadata annotations or a domain-specific language to specify abstract syntax model to concrete syntax model mappings.

In this paper, we have proposed and described the ModelCC domain-specific language for abstract syntax model to concrete syntax model mappings (ModelCC DSL for ASM-CSM mappings). This DSL allows the specification of separate abstract syntax model to concrete syntax model mappings.

As an example, we have specified the ModelCC DSL for ASM-CSM mappings as an ASM and several equivalent ASM-CSM mappings written in the DSL itself.

In the future, we plan to apply model-based language specification techniques to problems such as data integration and natural language processing. We also plan to incorporate different reference resolution techniques to ModelCC.

ACKNOWLEDGMENTS

Work partially supported by research project TIN2012-36951, "NOESIS: Network-Oriented Exploration, Simulation, and Induction System", cofunded by the Spanish Ministry of Economy and the European Regional Development Fund (FEDER).

References

[Aho et al., 2006] Aho, A. V., Lam, M. S., Sethi, R., and Ullman, J. D. (2006). *Compilers: Principles, Techniques, and Tools.* Addison Wesley, 2nd edition.

[Fowler, 2002] Fowler, M. (2002). Using metadata. *IEEE Software*, 19(6):13–17.

[Kats et al., 2010] Kats, L. C. L., Visser, E., and Wachsmuth, G. (2010). Pure and declarative syntax definition: Paradise lost and regained. In *Proceedings of the ACM International Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA'10)*, pages 918–932.

[Kleppe, 2007] Kleppe, A. (2007). Towards the generation of a text-based IDE from a language metamodel. volume 4530 of *Lecture Notes in Computer Science*, pages 114–129.

[Quesada, 2012] Quesada, L. (2012). A model-driven parser generator with reference resolution support. In *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*, pages 394–397.

[Quesada et al., 2011] Quesada, L., Berzal, F., and Cubero, J.-C. (2011). A language specification tool for model-based parsing. In *Proceedings of the 12th International Conference on Intelligent Data Engineering and Automated Learning. Lecture Notes in Computer Science*, volume 6936, pages 50–57.