EXPLICIT SUPPORT FOR LANGUAGES AND MOGRAMS IN THE SLEWORKS LANGUAGE WORKBENCH

Igor Dejanović, Gordana Milosavljević, Branko Perišić, Ivan Vasiljević, Milorad Filipović
University of Novi Sad, Faculty of Technical Sciences, Novi Sad, Serbia

Abstract *– In our previous work we have described MoRP (**Mo**del-**R**eference-**P**roperty) meta-meta model [1], a simple meta-meta model which forms a foundation for SLEWorks language workbench. In this paper we present an explicit support for concepts of a language, language components (abstract and concrete syntaxes) and language utterances (i.e. mograms) in the SLEWorks platform. This support is implemented as a MoRP based language called MetaLanguage. Making these two concepts explicit enables us to organize our language utterances in the logical parts, i.e. mograms. Furthermore, we are able to specify relationship of conformance between mograms and languages. We also give an overview of some basic MoRP languages and their relationship inside SLEWorks repository.*

1. INTRODUCTION

In our previous work [1] we have presented the meta-meta model MoRP (**Mo**del-**R**eference-**P**roperty), which forms the core of our research platform called SLEWorks. The aim of SLEWorks is to form a foundation for building and evolving different domain-specific languages and language utterances.

For effective creation of languages and language utterances and their evolution sophisticated integrated environments (IDEs) should be developed. This kind of IDEs Martin Fowler calls *Language Workbenches* [3].

In the last few years we are witnessing the rise of the movement gathered around the Software Language Engineering (SLE) conference. In the view of SLE followers, the term "software language" comprises all sorts of artificial languages used in software development including general-purpose programming languages, domain-specific languages, modeling and meta-modeling languages, data models, and ontologies [4]. In that regard, programs in the classical sense and models should be treated in the same way. In order to make this clear Aneke Kleppe has defined term "mogram" as an amalgam of terms "model" and "program" to mean a program, model, XML file, database scheme or any other utterance given in some artificial software language [5].

This paper gives an overview of the theoretical foundation that outlines the framework for the SLEWorks implementation. It also proposes an explicit support for languages and mograms as an MoRP based language. We also give an overview of basic MoRP languages and their relationship inside SLEWorks repository.

The paper is organized as follows: Section 2 lays out formal foundation through definition of mogram and language; Section 3. gives an overview of core MoRP based languages of the SLEWorks platform which enables creation of mograms, languages and their relationships; Section 4. gives some more details on the creation of concrete syntaxes; related work is presented in Section 5. the conclusion is given in Section 6. together with some further research directions.

2. LANGUAGES AND MOGRAMS

Generally speaking, a language is a means for communicating ideas and information. We are mainly interested in artificial software languages. In that regard, we shall define the software language as a means for description of existing (descriptive role) or yet to be built systems (prescriptive role).

Mogram *M* always conforms to some software language *L*. This relation shall be denoted as [2]:

$$M \looparrowright L$$

Using definition and ideas presented in [5,6] we formally define language as 3-tuple [2]:

$$L = (M_{\mathrm{A}}, S, \mathscr{C})$$

Where:

- M_{A} - is a language utterance or *mogram* which defines abstract syntax of the language, i.e. it conforms to the language for abstract syntax definition. An abstract syntax defines validity of language utterances from the structural point of view. Abstract syntaxes of languages for abstract syntax definition are known as meta-meta models in the fields of modeling. MoRP is one of them.
- $S = (M_{\mathrm{S}}, \varphi_{\mathrm{S}})$ - is a definition of language semantics, i.e. the meaning of the language concepts. In [7] semantics is defined as a semantic domain and mapping from elements of abstract syntax to the elements of semantics domain. Following this definition M_{S} shall be the semantics domain while φ_{S} shall be the mapping of abstract syntax concepts to the semantics domain, i.e.

$$\varphi_{\mathrm{S}} : M_{\mathrm{A}} \rightarrow M_{\mathrm{S}}$$

- $\mathscr{C} = \{C_1, C_2, ..., C_n\}$ - is a set of concrete syntaxes defined as a tuple $C_j = (M_{Cj}, \varphi_{Cj})$ where M_{Cj} is a mogram which defines the abstract syntax of one of

possible concrete syntaxes and φ_{Cj} is the mapping:

$$\varphi_{Cj} : M_A \rightarrow M_{Cj}$$

Each mogram M_{Cj} conforms to the specific language for concrete syntax definition, denoted as

$$M_{C1} \ \looparrowright \ L_{C1}, M_{C2} \ \looparrowright L_{C2}, \ldots$$

- Concrete syntax specifies the way mograms will be presented to the user and the way the user will interact with them. It specifies the interface between the user and the mogram. Another use of concrete syntax is the mogram serialization, i.e. transforming abstract form to the concrete form that can be stored to the external memory (for example, XMI standard [8] defines concrete XML based syntax for model serialization in the context of MDA [9]).

3. MoRP LANGUAGES

One of the design choices in MoRP development was simplicity. MoRP is based on the three main concepts shown in Figure 1 (Model, Reference, Property). While this is enough to specify abstract syntaxes for a wide range of languages it is difficult to manage those languages and to introduce explicit relationship among them.

In order to model repository capable of storing of languages and mograms defined in previous section it was necessary to introduce explicit concepts of *Language* and *Mogram*. Extending core MoRP with these concepts would make it unnecessary complex since the main purpose of MoRP is modeling abstract syntaxes where the

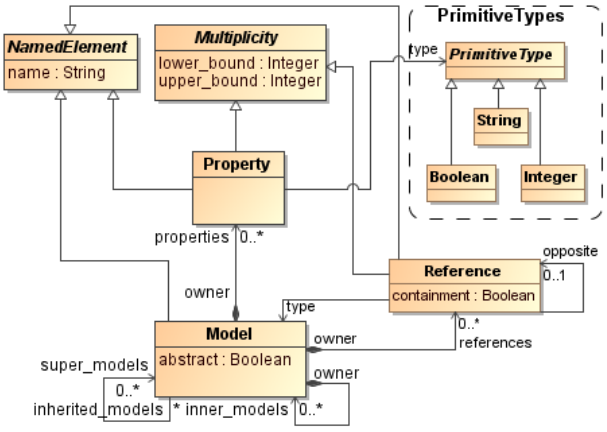

Figure 1: The MoRP meta-metamodel

concepts of *Language* and *Mogram* would be superfluous.

Our approach is the use of language *MetaLanguage* for specification of all MoRP based languages (Figure 2). The abstract syntax of this language is given in MoRP and is, at the same time, an extension of core MoRP meta-metamodel. Both *Language* and *Mogram* extends the core concept *NamedElement* therefore inheriting *name* property. Additionally, both new concepts are uniquely identifiable using URI (*uri* property). *Language* supports one abstract syntax and zero or more concrete syntaxes. Language semantics is currently defined using source code generator and therefore is not explicit part of the *MetaLanguage* language.

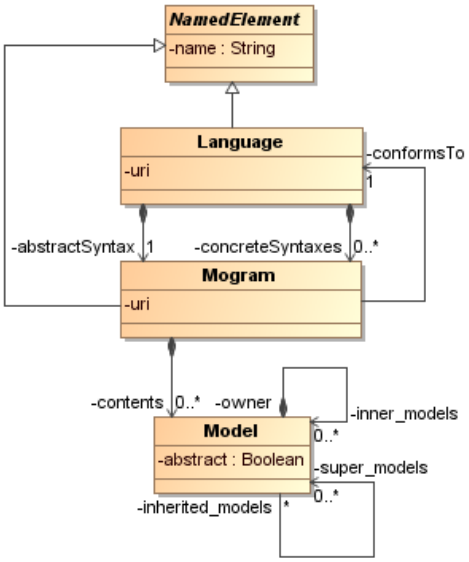

Figure 2: MoRP based abstract syntax of MetaLanguage

Each *Mogram* conforms to one and only one *Language* (reference *conformsTo*). This conformance means that the mogram is following the rules of the language it conforms to. The structure of the mogram must conform to the abstract syntax of the language. If the mogram is rendered to the user this representation must conform to one of its concrete syntaxes and the meaning of the mogram is interpreted using language semantics.

In addition to the *MetaLanguage* there are a few more base languages defined inside SLEWorks. The most important languages and their relationships are given in Figure 3. Concrete syntaxes are specified using *GraphSyntaxLanguage* and the *MoRPWeaver* languages. Usage of these two languages is described more thoroughly in the following section.

4. CONCRETE SYNTAXES

To be able to visualize and edit mograms we should be able to specify language concrete syntaxes. SLEWorks strives to support different concrete syntaxes so we are taking a projectional approach [12] where the user directly interacts with the underlying abstract syntax tree using the concrete syntax as an interface.

As defined in Section 2, concrete syntaxes consist of two parts:

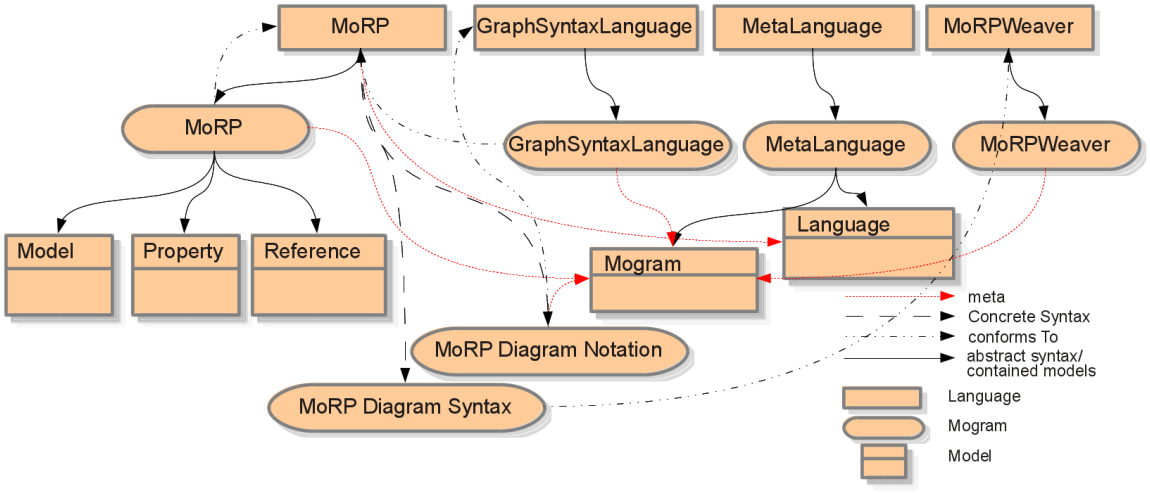

Figure 3: Base languages of SLEWorks and their relationships

- A mogram which conforms to the language for concrete syntax definition, and
- A mapping between mogram which describes the abstract syntax of the language and mogram conforming to the language for concrete syntax definition.

The first part defines the presentation elements, i.e. the form of the conforming mograms. This is given in an abstract way using mogram which conforms to the abstract syntax of language for concrete syntax specification. How each presentation element will be rendered to the user is what we consider to be a semantics of the language for concrete syntax definition. This semantics is implemented as the editor for each language for concrete syntax specification. The editor renders the mogram to the user and enables her to interact with the mogram.

The second part defines the meaning of each presentation element by using correlation with the language concepts (its abstract syntax elements).

These two parts are enough to support mogram rendering and presenting to the user. However, the concrete syntax should also have a behavioral component defining a way the user can interact with the mogram. In our view both these aspects of the concrete syntax makes its semantics. This element of the concrete syntax specification is built in the editor itself which is configured by the two-part concrete syntax definition, therefore it acts as an interpreter of the concrete syntax specification. Therefore, the editor itself is the specification of the semantics of the concrete syntax of the language.

For the purpose of presentation elements definition we are in the development of the MoRP language called *GraphSyntaxLanguage* which represents a language for graph-based concrete syntax definition. The main concepts of this language are *Symbol, Shape, Rectangle, Circle, Label, Layout, Compartment, Link, LinkDecoration*, etc. Using instances of these concepts we are able to specify the visual form of the language, its symbols, links, and their properties. Therefore, the presentation part of concrete syntax specification (its visual appearance) may be reused in different languages with different abstract syntaxes and different semantics.

For the mapping part of the concrete syntax definition we have developed simple MoRP language for general purpose weaving called *MoRPWeaver*. This language is used to define, as an external specification, relationships among elements from different mograms. Therefore, weaving that maps abstract syntax to the presentation elements of concrete syntax is itself mogram that conforms to the *MoRPWeaver* language.

One of the main parts of every language for concrete syntax definition is the editor which forms the dynamic or behavioral semantics of concrete syntax. The editor can be generated from concrete syntax specification or it can be configured by the syntax specification and in that case it runs as an interpreter of concrete syntax specification.

Figure 3 shows various SLEWorks languages, some real mograms that conform to them and their relationships. The first concrete syntax that is in the development is the concrete syntax of MoRP itself. Its visual properties are following the UML Class Diagram notation.

Mograms defining MoRP concrete syntax are presented in Figure 3. The first part of this syntax is specified by mogram *MoRP Diagram Notation* which conforms to the *GraphSyntaxLanguage*. This mogram specifies concrete syntax symbols, link and their styles. The second part is specified by the *MoRP Diagram Syntax* mogram, a weaver that conforms to the *MoRPWeaver* language and connects concepts from MoRP abstract syntax (*Model, Property, Reference...*) to the visual elements of the *MoRP Diagram Notation*.

5. RELATED WORK

Domain-specific languages and Language Workbenches have gained significant attention in the industry and academy which can be seen by the increasing number of related publications in the key conferences and journals as well as availability of some very good and mature language engineering platforms. Xtext [10] is a tool for creating domain-specific textual languages. It is based on EMF and ECore. It started as a part of a language engineering platform called openArchitectureWare and currently is part of Eclipse Modeling Project developed under Eclipse Foundation. The language grammar is described using a dialect of EBNF. Starting from the grammar definition Xtext can derive an ECore meta-model as well as a parser, linker, and a fully functional Eclipse based editor. Further customization of the editor is possible using Xtend DSL. Xtext based DSLs are well integrated with the Java type system making it possible for integration with existing Java libraries. Although Xtext is a great tool it focuses only on textual languages while our goal is to support languages with different types of concrete syntaxes. Furthermore, Xtext currently uses code generation for the editor and supporting tools where generated code must be deployed in another Eclipse instance. This has a drawback that the language cannot be tested and used inside the same Eclipse instance where it is developed which brings some overhead in the implementation-test cycle.

Another popular platform is Spoofax [11]. Spoofax supports language definition with declarative domain-specific languages. The modular, declarative syntax definition formalism called SDF enables language extensions and embeddings through composition. Analysis, transformation, and code generation is supported with Stratego language. One of the features of Spoofax is the integration of development and run-time environments. By using language-parametric editor services, that dynamically load and update language-specific service specifications, generated editors for a language can be used in the same Eclipse instance in which language definition itself is edited. This enables faster implementation-test cycle.

MPS [13] is a projectional language workbench developed by the JetBrains. To define language in MPS one have to define the language concepts (abstract syntax), the editor for the concepts (concrete syntax) and a code generator (semantics). The underlying abstract syntax tree is directly manipulated using editors. MPS supports composable language definitions. Languages can be extended and embedded. Although powerful, MPS is currently mainly focused on textual and tabular concrete syntaxes. Furthermore, MPS doesn't make underlying meta-model explicit.

6. CONCLUSION

In this paper we have presented an explicit support for concepts of *Language* and *Mogram* in the SLEWorks language workbench which is introduced in our previous work [1]. This support is implemented as a MoRP based language called *MetaLanguage*. Making these two concepts explicit enables us to organize our language utterances in the logical parts, i.e. mograms. Furthermore, we are able to specify relationship of conformance between mograms and languages.

Besides *MetaLanguage*, we have described languages used for the specification of MoRP graphical concrete syntax. A concrete syntax is defined by using languages for the specification of visual properties of the syntax as well as mapping from the elements of abstract syntax to the elements of concrete syntax. This mapping is done by the weaving language. An Editor for the specific type of concrete syntax (e.g. graphical) is configured using these descriptions for the specific concrete syntax.

In the further work we plan to experiment more with the various MoRP based languages and research the way of their integration. We also intend to analyze different concrete syntaxes and their influence to the performance of the users.

7. BIBLIOGRAPHY

[1] Dejanović, I.; Perišić, B., Milosavljević, G., *MoRP Meta-metamodel: Towards a Foundation of SLEWorks Language Workbench*, 2nd International Conference on Information Society Technology (ICIST 2012), Kopaonik, Serbia, pp. 36-40, 2012.

[2] Dejanović, I., *Prilog metodama brzog razvoja softvera na bazi proširivih jezičkih specifikacija*, phD disertation, Faculty of Technical Sciences, University of Novi Sad, 2012.

[3] Fowler, M., *Domain-Specific Languages*, Addison-Wesley Professional, 2010.

[4] Software Language Engineering Home Page, http://planet-sl.org, accessed 28. January 2013.

[5] Kleppe, A.: *Software language engineering: creating domain-specific languages using metamodels*, Addison-Wesley, 2009.

[6] Kleppe, A., *A Language Description is More than a Metamodel*, Fourth International Workshop on Software Language Engineering, 2007.

[7] Harel, D., Rumpe, B., *Meaningful modeling: what's the semantics of" semantics"?*, Computer, Vol. 37, pp. 64-72, 2004.

[8] *MOF 2.0/XMI Mapping*, Version 2.1.1, Object Management Group, 2007.

[9] Miller, J., Mukerji, J. et al., *MDA Guide*, Version 1.0.1, Object Management Group, 2003.

[10] Efftinge, S., Völter, M., *oAW Xtext: A framework for textual DSLs*, Eclipse Summit 2006, Workshop: Modeling Symposium, 2006.

[11] Kats, L. & Visser, E. The spoofax language workbench: rules for declarative specification of languages and IDEs, *ACM Sigplan Notices, Vol. 45, pp. 444-463*, 2010.

[12] Fowler, M. Language workbenches: The killer-app for domain specific languages, *Online* http://www.martinfowler.com/articles/languageWorkbench.html, *2005, Accessed 30. January 2013.*

[13] Voelter, M., *Language and IDE modularization, extension and composition with MPS*, Pre-proceedings of Summer School on Generative and Transformational Techniques in Software Engineering (GTTSE), pp. 395-431, 2011.

ACKNOWLEDGMENTS

Results presented in this paper are partialy funded as the research conducted within the Grant No. III-44010, Ministry of Science and Technological Development of the Republic of Serbia