Computer Languages, Systems & Structures 52 (2018) 1–20

Contents lists available at ScienceDirect

Computer Languages, Systems & Structures

journal homepage: www.elsevier.com/locate/cl

Towards a completely extensible dynamic geometry software with metadata

Davorka Radaković*, Đorđe Herceg

University of Novi Sad, Faculty of Sciences, Department of Mathematics and Informatics, Trg Dositeja Obradovića 4, 21000 Novi Sad, Serbia

ARTICLE INFO

Article history:
Received 27 April 2017
Revised 8 November 2017
Accepted 13 November 2017
Available online 22 November 2017

MSC:
68N19

PACS:
89.20.Ff

Keywords:
Source code annotations
Metadata
Dynamic geometry software
Component development
Functional languages
Lazy evaluation

ABSTRACT

Dynamic Geometry Software (DGS) are widely accepted as tools for creating and presenting visually rich interactive teaching and learning materials, called dynamic drawings. Dynamic drawings are specified by writing expressions in functional domain-specific languages. Due to wide acceptance of DGS, there has arisen a need for their extensibility, by adding new semantics and visuals. We have developed the SLGeometry dynamic geometry software with a genericized functional language and the corresponding expression evaluator that act as a framework into which specific semantics is embedded in the form of code annotated with metadata. SLGeometry is implemented in C# on the .NET Framework. Although attributes are a preferred mechanism to provide association of declarative information with C# code, they have certain restrictions which limit their application to representing complex structured metadata. By developing a metadata infrastructure which is independent of attributes, we were able to overcome these limitations. Our solution, presented in this work, provides extensibility to simple and complex data types, unary and binary operations, type conversions, functions and visuals, thus enabling developers to seamlessly add new features to SLGeometry by implementing them as C# classes annotated with metadata. It also provides insight into the way a domain specific functional language of dynamic geometry software can be genericized and customized for specific needs by extending or restricting the set of types, operations, type conversions, functions and visuals.

1. Introduction

Metadata and techniques such as metaprogramming, attribute-oriented programming and component-object programming are widely used in last decades [1–4]. Attribute-oriented programming is a program-level marking technique that allows developers to declaratively enhance programs through the use of metadata. Developers can mark program elements (e.g. classes, interfaces, methods, fields) with attributes (annotations) to indicate that they maintain application-specific or domain-specific semantics [5–7]. The existence of attributes can be checked at runtime and actions taken depending on their values. Besides providing data which have an effect on the runtime program behavior, they are used as the indication to developers about the aim and the behavior of the tagged code. Component software design has experienced enormous benefits from the re-usability point of view [8,9].

* Corresponding author.
E-mail addresses: davorkar@dmi.uns.ac.rs (D. Radaković), herceg@dmi.uns.ac.rs (Đ. Herceg).

https://doi.org/10.1016/j.cl.2017.11.001
1477-8424/

Metadata is data about data in computer science and can be interpreted in different ways according to needs. When observed in object-oriented programming, it is the information about the program structure itself [10]. Metadata can be used in various scopes: it started with metadata in public digital libraries [11], pedagogical metadata for learning objects (LO) [12,13], metadata for ontology description and publication [14], metadata in social-networks [15,16], metadata in DSL [17,18], up to database systems [19]. Metadata standards are introduced for simple and generic resource descriptions [20].

Dynamic Geometry Software (DGS) [21–28] is a software that enables creation and real-time manipulation of visually rich interactive teaching and learning materials, called *dynamic drawings*. Dynamic drawings are specified by writing expressions in a functional domain-specific language (abbreviated: FLG) and assigning them to named variables. In that regard, the set of variables can be considered equivalent to the drawing it represents. The set of types (T), type conversions (C), operations (O), functions (F) and visuals (V) in the FLG is denoted with $\tau = \{T, C, O, F, V\}$. A DGS consists of an expression evaluator (*Engine*) and a front end which displays dynamic drawings on the screen (*GeoCanvas*). Visual objects (*visuals*) are declaratively bound to certain types, and the DGS takes care of drawing them on the GeoCanvas. If a visual exists that corresponds to the constant type obtained by evaluating a variable, that visual appears on the screen. Some objects are dependent on other objects, i.e. their expressions contain variable references. For example, a segment is defined by its two endpoints. Whenever an endpoint is moved, the segment also moves. The Engine can also make use of a Computer Algebra System (CAS) [29–32] to manipulate and transform expressions.

A DGS is implemented in a host language (HL) [33] such as Java or C#. FLG types, functions and operations, as well as visuals, are written as HL classes which conform to some pre-established contracts. For example, all function classes must implement the Eval() method. It is possible to use the DGS directly from the HL, by programmatically creating expressions and calling the Engine API. In that sense, the FLG can be considered a language extension of the HL [18,34], since its classes are compiled with the unmodified HL compiler. Usually, however, the FLG is used as a standalone language inside a dedicated DGS, which provides at least two ways of entering expressions: by textual input through a parser, or by using drawing tools in the graphical user interface (GUI). From this point on, we shall consider C# as the HL of choice.

Dynamic geometry software has made a significant impact on the way geometry is taught in schools [35]. Creation of geometric drawings, that was once a tedious process, has become easy and available to anyone with even a modest personal computer. In time, teachers started using DGS to create teaching and learning materials for subjects other than geometry. Many examples in the subjects of geography [36], numerical analysis [37,38], combinatorics, architecture, mechanical engineering etc. can be found, see for example GeoGebraTube [39]. In time, two shortcomings of the current DGS have come to our attention.

First, DGS are not well suited for universal application, as they mostly contain geometric objects and functions that operate on them. [40] supports this viewpoint, arguing that majority of available computer-aided teaching materials in geometry are oriented towards fundamental problems in elementary and secondary school mathematics. The authors employed the GeoGebra DGS [26], supplemented with a professional-grade 3D modeling software Rhinoceros, to present and solve an array of practical engineering problems. In our paper [36] we successfully applied GeoGebra to solving geographical problems, but this effort resulted in very complex dynamic drawings. This is common occurrence, because many simple geometric shapes need to be combined to represent complex visual objects. The number of those objects and their respective variables can quickly become overwhelming for the user.

Second, geometric objects in DGS carry within them only the minimal necessary amount of data. Additional properties of objects are calculated by applying separate functions to those objects. This way, expressions are light and efficiently evaluated. For example, a constant which represents a linear segment between two points carries only the data for the two points. The midpoint of the segment is calculated only if needed, by using the Midpoint function. One significant disadvantage of this approach is function namespace congestion, because Midpoint shares the same namespace with numerous other functions, although it is limited to a very specific argument type and purpose. This issue becomes pronounced as more geometric objects are added to DGS, because each new object type can have many properties, which all require separate functions to be imported into τ. The other problem affects HL developers of DGS, who have to know the current state and anticipate the future functions in τ and invest additional work to avoid name clashes or provide overloads.

There are two choices posed before the developers who wish to extend a DGS with new features: either try to include as many different visual objects and corresponding functions into a DGS as possible, or provide an extensibility mechanism, such as plug-ins, and defer the development of new features to plug-in authors. It is evident in GeoGebra and other DGS that many new functions, visual shapes and other functionalities are added in each new version, for example see [40] or [41].

Having this in mind, our aim was to create such a DGS, which would mitigate the aforementioned problems. We created a generalized extensible DGS SLGeometry in C# on the .NET Framework, in order to observe the design requirements of such a software and propose a viable implementation of the extensibility framework based on metadata. We have set the following goals:

1. Objects must be unified with their properties, so that they can be implemented together in HL, and property access must be realized without the need for separate functions;
2. An extensibility mechanism, such as plug-ins, must be devised for the DGS, which would enable adding new types, operations, functions and visuals to τ;
3. There must be no differences in treatment of built-in and imported members in τ;