Building visual domain-specific languages using the semiotic approach: A case study on EasyTime

Iztok Fister Jr.
Faculty of Electrical Engineering and Computer Science, University of Maribor
Smetanova 17, 2000 Maribor, Slovenia
iztok.fister1@um.si

Iztok Fister
Faculty of Electrical Engineering and Computer Science, University of Maribor
Smetanova 17, 2000 Maribor, Slovenia
iztok.fister@um.si

ABSTRACT

This paper presents the development and usage of the EasyTime III Visual Domain-Specific Language (VDSL) for measuring time during sports competitions that enables users to create the VDSL programs visually. Indeed, this new hybrid VDSL creating approach deals with semiotics' analysis in place of building a meta-model. Thus, each visual element, like an icon, is handled as a sign with its representation (form and shape), reference (object) and meaning (implementation of object). Each visual element is then mapped into DSL language construct as defined by object implementation. In general, this approach is suitable, when Domain-Specific Language (DSL) using a Textual User Interface (TUI) is already implemented within a certain application domain and, therefore, the developer can be focused mainly on designing a Graphical User Interface (GUI). In this study, the existing EasyTime DSL has been upgraded to EasyTime III VDSL using the hybrid VDSL creation approach. Experiments of measuring time in an aquathlon have shown that the programming in EasyTime III is simple and efficient, whilst all other features of EasyTime DSL are preserved within the new VDSL.

Keywords

domain-specific language, EasyTime, semiotics

1. INTRODUCTION

Until recently, measuring time during sports competitions was performed manually by timekeepers using time obtained from stopwatches assigned to specific competitors according to their starting numbers. The rapid expansion of Radio Frequency IDentification (RFID) [6] technology has led into the development of various measuring devices. However, measuring devices represent only one part of the story, because measuring times during sports competitions cannot be achieved without a measuring system. This system collates timed-events from different measuring devices into a central database that enables organisers to track the results of competitors faster, more efficiently, and more accurately. In order to simplify working with the measuring system for tracking the results, the EasyTime domain-specific language was proposed for the Double Triathlon in 2009 [7]. This version, based on a specific compiler/generator, was developed for compiling an EasyTime source code into an abstract machine code. Modifying and widening the EasyTime domain-specific language demanded interventions directly into the compiler/generator. Therefore, an improved version of EasyTime was proposed in [8, 11, 10, 9], where EasyTime was defined formally as (domain analysis, abstract syntax, concrete syntax, formal semantics, and code generation), whilst a LISA [18] was used for building as a suitable compiler/generator. In this paper, we move forward in order to improve user experience and, thus, propose EasyTime III Visual Domain-Specific modelling Language (VDSL). This explores the visual interface to simplify the programming tasks of the measuring system. This VDSL addresses the shortcomings of its predecessor, i.e., simplifying its development. The users' visual programming consists of constructing the user model. This model is then translated into EasyTime DSL constructs. Indeed, a semiotics theory [12] is applied. Semiotics is the study of signs [2], where each sign consists of representation, object, and meaning. The meanings of the signs are defined explicitly in the user model [5]. Thus, a meta-model step can be avoided by the traditional creation of DSMLs. This approach is, therefore, also named as a hybrid VDSL creation. The VDSL was tested on building the EasyTime III program for measuring time during the aquathlon competition. The obtained results showed the power of the created visual tool that brings a great user experience on the one hand and simplified creating the visual programs.

The structure of the paper is as follows. Section 2 deals with a description of the proposed EasyTime III VDSL. In Section 3, created VDSL was applied for building the EasyTime III VDSL program in measuring time during an aquathlon competition. The paper concludes with Section 4, which summarizes the performed work and outlines the possible directions for the future.

2. EASYTIME III VISUAL DSL

The design of visual languages is a very complex process that, besides a knowledge of computer science, demands also the knowledge of areas like Psychology, Sociology, Anthropology, Linguistics, Design, and Engineering. For the development of EasyTime III VDSL, the following phases need

to be carried out:

- Concept analysis,
- Visualization of features,
- Semiotic analysis,
- Code generation.

In the remainder of the paper all the mentioned phases are described in detail.

A concept analysis identifies the concepts and relations between concepts. The analysis of a measuring time domain summarizes the results as proposed in [8] because VDSL EasyTime III addresses the same application-domain (i.e., measuring time) as the original EasyTime DSL. The concept analysis divides the concepts into features, and then these features into sub-features. However, the features and sub-features may be mandatory or optional. In order to denote the attitudes between concepts, features and sub-features, the concept analysis defines the following relations:

- *all*: All features or sub-features are mandatory.
- *more-off*: The feature may be either one of the sub-features from a set of sub-features or any combination of sub-features from the same set.
- *one-off*: The feature may be one of the sub-features from a set of sub-features but not any combination of sub-features from the same set.

The concept diagram of the measuring time domain is depicted in Figure 1, from which it can be seen that the concept *Race* consists of seven features (i.e., *Events*, *Transition area*, *Control points*, *Measuring time*, *Begin*, *End* and *Agents*).

2.1 Visualization of features

During the visualization process, the features are mapped into appropriate graphical interfaces, as presented in Table 2.1. The Table is interpreted as follows. Icons I_{begin} and I_{end} denote the features *Begin* and *End*, respectively. Events can be represented using icons I_{swim}, I_{bike}, and I_{run} and described the sub-features, as follows: *Swimming*, *Cycling*, and *Running*. The feature *Transition areas* is identified by icon I_{ta}, while *Measuring devices* are marked using icons I_{md0}, I_{md1}, and I_{md2}.

2.2 Semiotics‘ analysis

Semiotics is the study of 'everything that can be taken as a sign' [17]. This is an interdisciplinary theory that comprises domains of meanings in a variety of other disciplines like Psychology, Anthropology, Biology, Logic, Linguistics, Philosophy, and even Software Engineering [4]. Modern semiotics consists of two independent theories, as follows: Semiology and semiotics. The former was developed in Switzerland by Ferdinand de Saussure [16], while the latter in North America by Charles Sanders Peirce [13]. De Saussure's theory of signs originated from the language theory as a system of arbitrary signifying units. The Peircean theory of signs is based on logic and epistemology [2]. He defined the sign as anything that stands for something else, to somebody, in some respect or capacity. Nothing is a sign until it is interpreted by somebody. Peirce described signs as threefold structures consisting of: Representation, reference (object), and meaning (interpretation) (Figure 2). Two characteristics of these structures are valid: Firstly, a direct connection between a representation and reference need not necessarily exist. Secondly, a meaning is always a mediator between a representation and reference. That is, the sign does not exist until some interpretation of representation is taken that has some meaning for somebody. In other words, a sign requires the concurrent presence of all three characteristics. On the other hand, there can be many meanings to a sign. In our study, the Peircean structure of signs was taken into

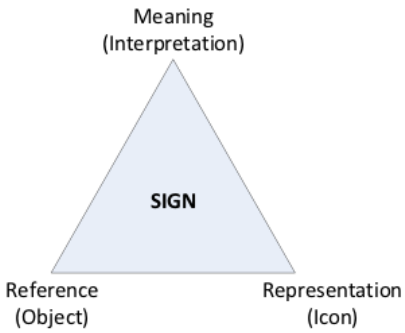

Figure 2: Structure of signs.

account in order to prescribe the unique meanings of them. Indeed, unique translation of signs can be achieved to EasyTime domain-specific language constructs. In line with this, semantic analysis is not needed for this translation. Therefore, this process of generating the EasyTime III VDSL was named a hybrid approach, and it can be applied usefully when the textual DSL is already developed in this application domain and an upgrade to the visual interface needs to be developed. The semiotics of EasyTime III can be described with the structures as illustrated in Table 2, from which it can be seen that each icon is represented by a corresponding object. In our study, objects are represented as C++ classes. The objects' meanings are defined by the implementation code. The source code in EasyTime DSL is generated as a result of the implementation.

For instance, an icon I_{begin} is referenced with an object *Begin* that is created by a parameter *Laps* which determines the number of laps. This object is responsible for generating the EasyTime DSL language construct "upd *STARTx*". Note that this character x denotes the integer value (also location counter, *lc*) necessary for distinguishing the different instances of the same variables because, in contrast, the same names of the variables will be generated for different control points. The variable *lc* is initialized during race configuration. The icons $I_{swim}, I_{bike}, I_{run}$ representing the events are represented by objects *Event* (Algorithm 1) that are responsible for generating two DSL EasyTime language constructs "dec *ROUNDx*" and "upd *INTERx*" (Algorithm 2). A class *Event* consists of three variables (*type*, *laps*, *lc*) and three methods (constructor *Event*, *initialize*,

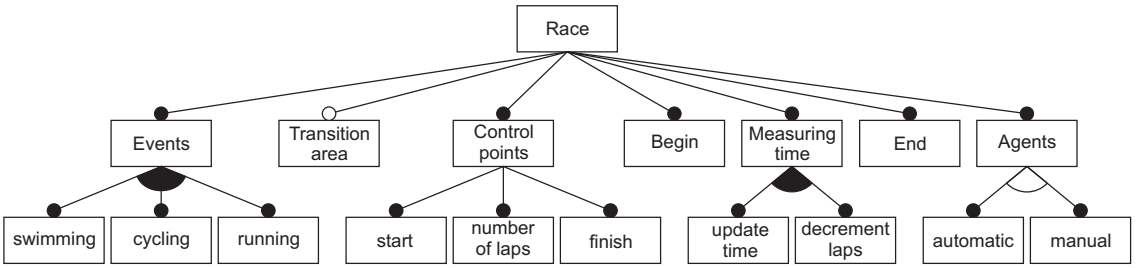

Figure 1: Concept diagram of measuring time domain.

Table 1: Translation of the application domain features to graphical elements.

Application domain features	Graphical elements
Begin race	I_{begin}
Events *(Swimming, Cycling, Running)*	$I_{swim}, I_{bike}, I_{run}$
Transition area	I_{ta}
End race	I_{end}
Measuring time	$I_{md0}, I_{md1}, I_{md2}$
Control points *(start, number of laps, finish)*	square dots (control points)
Agents *(automatic, manual)*	square dots (measuring points)

Algorithm 1 Definition of object 'Event'

```
1: class Event {
2:     int type;    // type of events Swim, Run, Bike
3:     int laps;    // number of laps
4:     int lc;      // index of generated variables
5:
6:     Event(int Type, int Laps, int Lc);
7:     void initialize();
8:     void generate();
9: }
```

generate). The variable *type* determines the type of event, i.e., *Swim*, *Run*, *Bike*, variable *laps* the number of laps that the competitor needs to accomplish a specific discipline, and variable *lc* determines the instance of a specific variable. The variable *laps* is a context information which is obtained by the user. The method *Event* constructs the object with appropriate parameters, method *initialize* generates the specific EasyTime DSL code necessary to initialize the variables within the scope, and the method *generate* generates the specific EasyTime DSL code appropriate for this event. A detailed implementation of the mentioned methods can be seen in Algorithm 2.

Algorithm 2 Implementation of object 'Event'

```
1: Event::Event(int Type, int Laps, int Lc)
2:     type = Type; laps = Laps;
3:     lc = Lc;
4: }
5: void Event::initialize() {
6:     gen(op_init, var_round, lc, laps);
7:     gen(op_init, var_inter, lc, 0);
8: }
9: void Event::generate() {
10:    gen(op_dec, var_round, lc);
11:    gen(op_upd, var_inter, lc);
12: }
```

Note that the other objects are represented and implemented in a similar manner.

2.3 Code generation

Code generation is divided into:

- the source code generation,
- the object code generation.

In the first phase, a visual representation of a real race using the graphical elements (also user model) is mapped into the EasyTime DSL source code while, in the second phase, this source code is compiled into object code using the LISA [18] compiler/generator. Semantic domains need to be defined for translating the graphical elements into the EasyTime DSL source code. In EasyTime III, two semantic domains are defined (Table 3). Both domains are used for a proper calling of the objects represented in the Table 4 in the translation phase. The former (i.e., D_{Event}) defines sub-features of the feature *Event*, while the latter (i.e., D_{Event}) sub-features of the feature *Md*. All the other parameters of the corresponding objects are of integer type. In the second phase, the EasyTime DSL source code translated from the corresponding user visual model is generated into virtual machine object code. The readers can obtain more information about this process in [8].

Table 3: Semantic domains in EasyTime III.

$D_{Event} = \{Swim, Bike, Run\}$	$Event\_Type \in D_{Event}$
$D_{Md} = \{Md0, Md1, Md2\}$	$Md\_Type \in D_{Md}$

Table 2: Translation of the application domain concepts to semiotic signs.

Icons	Objects	Meanings
I_{begin}	Begin($Laps, Lc$)	"($ROUNDx == \%Laps$) → upd $STARTx$;"
$I_{swim}, I_{bike}, I_{run}$	Event($Event\_Type, Laps, Lc$)	"dec $ROUNDx$;" "upd $INTERx$;"
I_{ta}	End(Lc) Begin($Laps, Lc$)	"($ROUNDx == 0$) → upd $FINISHx$;" "($ROUNDx == \%Laps$) → upd $STARTx$;"
I_{end}	End(Lc)	"($ROUNDx == 0$) → upd $FINISHx$;"
$I_{md0}, I_{md1}, I_{md2}$	Md($Md\_Type, Mp, Ag, [Ip\|Fn]$)	"mp[%Mp] ← agnt[%Ag] {" | "}"

3. MEASURING TIME IN AN AQUATHLON COMPETITION USING EASYTIME III VDSL

Aquathlon is a relatively young sport, the first National Competition being held in the USA in 1965. It belongs to a class of multi-sports, where the race consists of continuous two-stage disciplines involving swimming followed by running [14, 15]. Between both disciplines, however, a so-called transition area exists, where the competitors who finish the swimming prepare themselves for running. The time spent in the transition area is added to the final result of each specific competitor. Aquathlons are similar to triathlons. Triathlons, however, have cycling in addition to swimming and running. As a result, an aquathlon covers triathlon distances as well. For instance, a 1 km swim is followed by a 5 km run, etc. Distances vary depending upon the race venue and race organisers. For building the EasyTime III visual programs, an EasyTime III visual editor was developed using the Qt [1, 3], where a lot of bindings with other languages also exist, e.g., Java, Python, Ruby, etc. The visual program in an EasyTime III visual editor for measuring time during an aquathlon competition is illustrated in Figure 3. The visual editor is the graphical editor for describing the race to be measured. It consists of a graphical editor area, where the race should be described and a control area consisting of the buttons necessary for editing. In fact, the buttons represent either the application domain features or miscellaneous controls devoted to managing the visual programs. These buttons are divided into three subsets representing:

- the race configuration, i.e., buttons for dragging the icons representing the control points (e.g., I_{begin}, I_{end}, I_{swim}, $I_{bike}, I_{run}, I_{ta}$) and dropping them into specific positions within the visual editor area,
- situated measuring devices, i.e., buttons for dragging the icons representing the measuring points (e.g., I_{md1}, I_{md2}, I_{md0}) and dropping them into specific positions within the visual editor area,
- controls, namely the buttons necessary for opening, saving and erasing the visual programs, and generating the object code from the visual program.

Note that the graphical editor area consists of two fields in which icons are placed sequentially. The upper is sensitive to the buttons designated control points, whilst the lower to the buttons described measuring points. In fact, the upper fields of icons determine the configuration of a race, whilst the lower fields are where measuring devices have to be situated. Each icon also includes a square dot that enables the control point to be connected with the measuring point. Moreover, the transition area icon, similar to the measuring device with two mats, includes two square dots. Connection has to be made by dragging the source control point and dropping it into the destination measuring point or vice versa. Furthermore, the EasyTime III visual editor also includes certain context dependent information, like the number of laps, IP address of the measuring device, or an input file name. Zero laps on the swimming icon means that the swimming will not be measured on this measuring device. This information can be obtained by pressing the right-hand mouse button. The measuring time during an aquathlon competition is configured as follows. The upper graphical editor field determines the race configuration (i.e., start of race, swimming, transition area, running and finish of race), whilst the lower graphical editor field denotes situated measuring devices (i.e., realised by a measuring device with two measuring points). The connections between the control points and measuring points determine where the particular event has to be measured. For instance, the finish time of swimming has to be measured at measuring point 1 (antenna mat 1), whilst the other three running events (start time, intermediate times, and finish time) have to be measured at measuring point 2 (antenna mat 2).

3.1 Source code generation for measuring time in an aquathlon competition

Source code generation starts with translating the user visual model (consisting of icons and links) into semiotics objects. Indeed, an area of central points *CP* and area of measuring points *MP* together with an adjacent matrix representing connections between control and measuring points are generated. The results of this translation are illustrated in Table 4. It can be seen from the Table that the control points' area *CP* consists of six semiotics objects representing two disciplines (e.g., swimming and running) embraced between *Begin* and *End* semiotics objects. The generated names of the variables in these semiotics objects are distinguished according to their location counter. For instance, all variables referring to the first discipline are generated with $lc = 1$, whilst the variables referring to the second discipline with $lc = 2$. There are two measuring points within an area *MP*, and four links where the time should be measured. Note that the adjacent matrix *LN* designates the connections between the control and measuring points. The program presented in the Algorithm 3 is generated according to the race configuration. This generation is straightforward when someone follows the code generation as defined by the

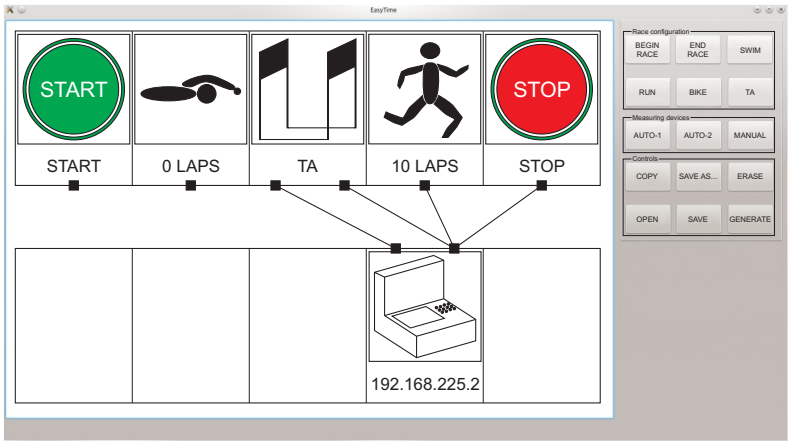

Figure 3: Visual program for measuring time in an aquathlon competition.

Table 4: Result of algorithm 'config_race'.

CP = { $Begin$(0,1),$Event$(Swim, 0, 1),End(1),$Begin$(0,2),$Event$(Run, 10, 2),End(2) }
MP = { Md(Auto-2,1,1,"192.168.225.2"), Md(Auto-2,2,1,"192.168.225.2") }
LN = { (3,1),(4,2),(5,2),(6,2) }

Algorithm 3 Source code for measuring time during on aquathlon.

```
 1: 1 auto 192.168.225.2;
 2:
 3: START1 = 0;
 4: ROUND1 = 0;
 5: INTER1 = 0;
 6: FINISH1 = 0;
 7: START2 = 0;
 8: ROUND2 = 10;
 9: INTER2 = 0;
10: FINISH2 = 0;
11:
12: mp1[1] → agnt[1] {
13:   (ROUND1 == 0) → upd FINISH1;
14: }
15: mp1[2] → agnt[1] {
16:   (ROUND2 == 10) → upd START2;
17:   (true) → upd INTER2;
18:   (true) → dec ROUND2;
19:   (ROUND2 == 0) → upd FINISH2;
20: }
```

meanings of the semiotics objects. Note that this code is functionally equivalent to the code written by the domain expert manually. Later, EasyTime LISA compiler/generator is used for object code generation [8].

4. CONCLUSION

This paper proposes a new hybrid VDSL creation approach that is suitable when the DSL already exists in certain application-domain. In this manner, a programmer exploits existing DSL and focuses on designing a graphical user interface. Thus, he/she avoids constructing the meta-model and, in line with this, the usage of the complex development frameworks, like Eclipse. The modelling step is, here, substituted with semiotic analysis, where each visual element, like icon or link, is handled as a sign with its representation (object) and meaning (implementation of the object). From these so-called semiotics objects, the source code is then generated, which can be translated into object code using the existing compiler/generator. The proposed approach was tested by the development of EasyTime III VDSL for measuring time during sports competitions and starting with the concepts of an application domain obtained through domain analysis. These concepts serve as the basis for building EasyTime III graphical user interfaces on the Qt based visual editor. The visual program (also user model) built using this editor is then mapped into EasyTime DSL constructs. The result of this translation is the EasyTime DSL source program. Translating this source program using the LISA compiler/generator generates an object AM code for the EasyTime measuring system. As a result, the developed EasyTime III VDSL enables ordinary users (e.g., organisers of sports competitions) to create programs for measuring time application-domain visually. That is, instead of writing the program in text editor, only 'point-and-click' is needed with the icons on the screen. With the proposed EasyTime III, the programming of a measuring system within visual environments is easier, faster, and more effective.

5. REFERENCES

[1] J. Blanchette and M. Summerfield. *C++ GUI programming with Qt 4*. Prentice Hall PTR, 2006.

[2] D. Chandler. *Semiotics: The Basics*. Routledge, New York, US, 2007.

[3] M. Dalheimer. *Programming with QT: Writing portable GUI applications on Unix and Win32*. O'Reilly Media, Incorporated, 2002.

[4] C. de Souza. *The semiotic engineering of human-computer interaction*. MIT Press, Cambridge, England, 2005.

[5] C. de Souza. Semiotic perspectives on interactive languages for life on the screen. *Journal of Visual Languages & Computing*, 24(3):218 – 221, 2013.

[6] K. Finkenzeller. *RFID handbook: fundamentals and*

applications in contactless smart cards, radio frequency identification and near-field communication. Wiley, 2010.

[7] I. Fister Jr. and I. Fister. Measuring time in sporting competitions with the domain-specific language Easytime. *Electrotechnical review*, 78(1–2):36–41, 2011.

[8] I. Fister Jr., I. Fister, M. Mernik, and J. Brest. Design and implementation of domain-specific language Easytime. *Computer Languages, Systems & Structures*, 37(4):151–167, 2011.

[9] I. Fister Jr, T. Kosar, I. Fister, and M. Mernik. Easytime++: A case study of incremental domain-specific language development. *Information Technology And Control*, 42(1):77–85, 2013.

[10] I. Fister Jr., M. Mernik, I. Fister, and D. Hrnčič. Implementation of Easytime formal semantics using a LISA compiler generator. *Computer Science and Information Systems*, 9(3):1019–1044, 2012.

[11] I. Fister Jr., M. Mernik, I. Fister, and D. Hrnčič. Implementation of the domain-specific language Easytime using a LISA compiler generator. In *Computer Science and Information Systems (FedCSIS), 2011 Federated Conference on*, pages 801–808. IEEE, 2011.

[12] C. Peirce. Collected papers of charles sanders peirce. volume 1–8, Cambridge, MA, 1931–1958. Harward University Press.

[13] C. Peirce and V. Welby. *Semiotic and significs: the correspondence between Charles S. Peirce and Lady Victoria Welby*. UMI-books on demand. Indiana University Press, 1977.

[14] S. Petschnig. *10 Jahre IRONMAN Triathlon Austria*. Meyer & Meyer Verlag, 2007.

[15] S. Rauter and M. Doupona Topič. Perspectives of the sport-oriented public in slovenia on extreme sports. *Kinesiology*, 43(1):82–90, 2011.

[16] F. Saussure. *Course in General Linguistics*. Duckworth, 1976.

[17] E. Umberto. *A theory of semiotics*. Advances in semiotics. Indiana University Press, 1976.

[18] University of Maribor. Lisa 2.2. `http://labraj.uni-mb.si/lisa/`, 2013. Accessed 17 August 2016.