ELEKTROTEHNIŠKI VESTNIK 78(1-2): 36–41, 2011
ENGLISH EDITION

Measuring Time in Sporting Competitions with the Domain-Specific Language EasyTime

Iztok Fister ml., Iztok Fister

University of Maribor, Faculty of Electrical Engineering and Computer Science, Smetanova 17, 2000 Maribor, Slovenija
email: iztok.fister@uni-mb.si

Abstract. Measuring time in mass sporting competitions is unthinkable manually today because of their long duration and unreliability. Besides, automatic timing devices based on the RFID technology have become cheaper. However, these devices cannot operate stand-alone. To work efficiently, they need a computer timing system for monitoring results. Such system should be capable of processing the incoming events, encoding and assigning results to a individual competitor, sorting results according to the achieved time and printing them. In this paper, a domain-specific language named EasyTime will be defined. It enables controlling an agent by writing events to a database. Using the agent, the number of measuring devices can be reduced. Also, EasyTime is of a universal type that can be applied to many different sporting competitions.

Key words: domain specific language, parser, BNF notation, code generation, time measuring, RFID technology

1 INTRODUCTION

Not long ago, time in sporting competitions was measured manually by timekeepers. The measured time was assigned to the starting number of competitors who were arranged according to the final result and the competitor's category. With the arrival of the Radio Frequency Identification technology (RFID) [8], the cost of the measurement technology (like ChampionChip [7] and RFID Race Timing System [8]) was reduced. It thus became accessible to a wider class of users, e.g. sport clubs, organizers of sporting competitions etc. They then began to compete with the existing monopolies (Timing Ljubljana [9]) by measuring results in smaller sporting competitions.

Besides the measuring technology in a sporting competition, a flexible computer timer system is also needed allowing measuring of various sporting competitions with an arbitrary number of measuring places, online time tracking, printing result lists and ensuring reliability and security. Flexibility of such a system can be increased by using the domain-specific language (DSL) EasyTime.

The domain-specific languages[1] are suitable for the application domain and have definite advantages over general-purpose languages in a specific domain. First of all, these advantages are expressed in a higher expressive power and, therefore, higher productivity, ease of use (even for domain experts that are not programmers), easier verification and optimization. EasyTime is used to configure agents to write an event that arises in a measuring device into a database. Thus, agents are crucial elements of the timing system. When carefully configured, the number of the necessary devices can be decreased.

The structure of the rest of the paper is as follows. In Section 2, problems of time measuring in sporting competitions are described. The focus will primarily be on the triathlon competitions which are the hardest to be efficiently measured because of the three different disciplines involved. In Section 3, EasyTime is presented in detail. Section 4 describes performance of a program written in EasyTime. The paper ends with a short analysis of the performed work and plans for the future work.

2 MEASURING TIME IN SPORTING COMPETITIONS

In practice, the measuring time in sporting competitions can be measured $manually$ (classically or with a computer timer) or $automatically$ (by using a measuring device). The computer timer is an application that runs usually on a workstation (portable computer) and measures the real time. Through this, a processor tact is exploited. The processor tact is the velocity with which the processor executes computer instructions. The computer timer enables tracking events that are triggered by a competitor crossing over the measuring place (MP) similarly to a measuring device. However, in that case, the event is triggered by an operator on the computer pressing the appropriate key on the keyboard.

Received 4 October 2010
Accepted 10 February 2011

The operator generates events in a form of triples $\langle \#, MP, TIME \rangle$, where $\#$ denotes the starting number of the competitor, MP the measuring place at which the event takes places, and $TIME$ the timestamp that is generated by the device at the moment of triggering and represents the number of seconds since 1.1.1970 at 0:0:0.

Today, the measuring devices are usually based on the RFID technology [8]. Thus, identification is performed by electro-magnetic wave motions within range of radio frequencies that are radiated by antenna fields. The measuring devices consist of:

- RFID tag readers,
- primary memory,
- LCD display, and
- numeric keyboard.

Here can be several antenna fields connected to the device. These fields represent a particular measuring place. Competitors trigger events by crossing over the antenna field with passive RFID tags that bear their identification numbers. These numbers are unique and different from the starting numbers. The event on the measuring device is represented in a quadruple form $\langle \#, RFID, MP, TIME \rangle$, where the RFID identification number is also added to the triplet. The accuracy of those measuring devices is usually limited to $1/10$ second that is enough for the propositions of the referee associations.

The measuring devices and workstations with an installed computer timer that represent the measuring places in the timing system can be connected to a local-area network (LAN). With these devices we communicate via a control program, i.e. an agent that runs on a database server. The agent gets connected with the measuring device via a suitable TCP/IP socket that supports an appropriate TCP/IP protocol. The measuring devices usually support the protocol $Telnet$ that is easy to implement and enables a text stream-oriented communication. The agent communicates with the manual timer via a file transfer protocol.

2.1 An Example: Measuring Time in a Triathlon

Special requirements appear in triathlon competitions where there are three disciplines to be dealt with in one competition. Therefore, we will focus on that problem in this paper.

The first triathlon competition was performed in the USA in 1975. The competition is regarded today as an olympic discipline in which the competitor starts with swimming, then rides a bicycle and finishes with running. All the three activities are performed sequentially and continuously. For the summary time, delays in both transitions are added. In the first transition, the competitor goes from swimming to bicycling, while in the second transition, he/she moves from bicycling to running. Today, there are many kinds of triathlon competitions. They are distinguished according to the length of particular courses. Normally, organizers use circular courses of shorter lengths (laps) over which competitors need to pass multiple times. However, this makes measuring considerably more difficult, since the number of laps also needs to be counted.

As seen from Fig. 1, the measuring time in triathlon competitions is divided into nine control points (CP). Control points are locations on the course where organizers need to track the measured time that can be $intermediate$ or $finish$. In Fig. 1, we are dealing with a double ultra triathlon (7.6 kilometers of swimming, 360 kilometers of bicycling and 84 kilometers of running), where the length of the swimming course is 380 meters (or 20 laps), the bicycle course is 3.4 kilometers (or 105 laps) and the running course is 1.5 kilometers (or 55 laps).

The summary time of a triathlon competition consists of five final times (the swimming time SWIM (CP2), the first transition time TA1 (CP3), the bicycling time BIKE (CP5), the second transition time TA2 (CP6) and running time RUN (CP8)) as well as three intermediate times (the intermediate swimming time (CP1), the intermediate bicycling time (CP4) and the intermediate running time (CP7)). With the intermediate times, the number of laps ROUND_x and achieved time INTER_x are measured. Here, $x = 1, 2, 3$ denotes a particular course.

Suppose that a measuring device with two measuring places (MP3 and MP4) is available for the measurement in Fig. 1 and the competition is performed at one location. In such case, the last crossing over the MP3 can indicate the CP5 time, the first crossing over the MP4 the CP7 time and the last crossing over the MP4 the final result (CP8). The measuring places MP1 and MP2 are measured manually by the computer timer. Finally, the number of measuring points can be reduced by three if the timing system and the control points are appropriately set up. Thus, 162 events for each competitor can be measured with one measuring device (or 87.5%). Moreover, the measurement technology for measuring swimming in sees and lakes still being expensive and thus still being measured by referees manually, almost 98% of all events at such competition can be measured.

3 DSL EASYTIME

With EasyTime, various measurements need to be described. This is made with the timing system. Moreover, reduction in the number of measuring devices is expected when using more complex measuring time. EasyTime enables describing the rules for controlling the agent before the event registering at a measuring place is recorded into the database. However, each program written in EasyTime needs to be compiled before an execution. Compiling consists of:

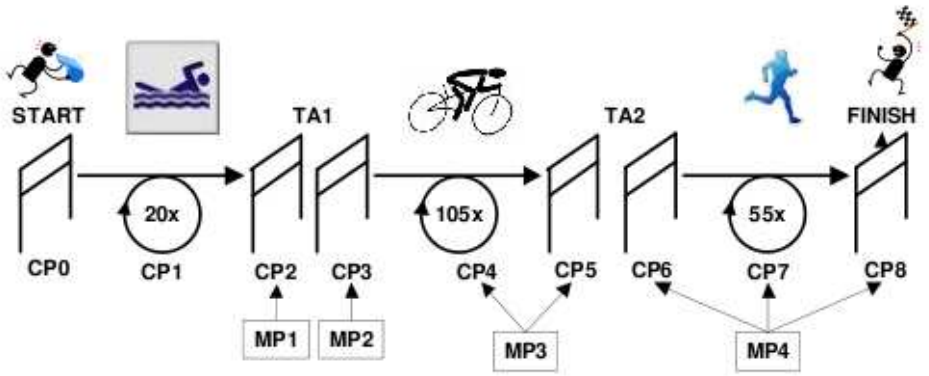

Figure 1. Definition of control points in the triathlon competition

- syntax analysis and
- code generation.

Syntax analysis is performed by a syntax analyzer (also parser) that the program written in EasyTime compiles into an intermediate code. From this code, a code-generator generates an executable code for a virtual machine and stores it in a database. The database table in which the code is stored is named the rule table because this code contains rules for controlling the agents. In the rest of the paper, the characteristics of the parser and code generator are presented in detail.

3.1 The Parser

Before developing a parser, a syntax for EasyTime needs to be defined. The syntax is a set of rules in which the structure of correct statements (grammar) is defined [6]. The syntax of EasyTime is presented in syntax diagrams in Fig. 2. The syntax diagrams are the most suitable form for writing parsers. It has the same expressive power as the BNF (Backus Naur Form) notation [4], i.e. a notation technique suitable for context-free grammars. The parser was developed in the programming language C/C++ [3].

Table 1. Names of variables in the table $RESULTS$

Variable	Description
ROUND_1	Number of laps of swimming
INTER_1	Intermediate time 1 (CP1)
SWIM	Finish time of swimming (CP2)
TRANS_1	Transition time 1 (CP3)
ROUND_2	Number of laps of bicycling
INTER_2	Intermediate time 2 (CP4)
BIKE	Finish time of bicycling (CP5)
TRANS_2	Transition time 2 (CP6)
ROUND_3	Number of laps of running
INTER_3	Intermediate time 3 (CP7)
RUN	Final time of triathlon (CP8)

The EasyTime program consists of definitions of:

- agents,
- variables, and
- measuring places.

The definition of agents described with the statement $Agents$ is represented in the syntactic diagram with the same name in Fig. 2. Variables denoting the names of columns in the database (RESULTS table) and representing the control points can be seen in Table 1. However, these needs to be defined before using them in the $Measuring - Places$ statement that defines the valid rules for a particular measuring place. The rules are in the form of ⟨Predicate⟩::=⟨Operation⟩. Note that in EasyTime character $ is set before the name of a variable.

For example, the definition of agents in EasyTime describes the measuring time in Fig. 1 as presented in Program 1, where the first agent saves the results of the manually measured time in the directory $/home/DC2$ and the second agent automatically obtains data from the measuring device with the IP address 192.168.225.100 via the UDP protocol on port 9999.

Program 1 Definition of agents

```
1: AGENTS {
2:     {1,MANUAL,"/home/DC2/res.ets"}
3:     {2,AUTO,"192.168.225.100/UDP/9999"} }
```

Program 2 Definition of measuring places

```
1: MM[1] ::= AGNT[1] {
2:     { (TRUE) ::= UPD $SWIM }
3:     { (TRUE) ::= DEC $ROUND_1 } }
4: MM[2] ::= AGNT[1] {
5:     { (TRUE) ::= UPD $TRANS_1 } }
6: MM[3] ::= AGNT[2] {
7:     { (TRUE) ::= UPD $INTER_2 }
8:     { (TRUE) ::= DEC $ROUND_2 }
9:     { (ROUND_2 == 0) ::= UPD $BIKE } }
10: MM[4] ::= AGNT[2] {
11:     { (TRUE) ::= UPD $INTER_3 }
12:     { (ROUND_3 == 55) ::= UPD $TRANS_2 }}
13:     { (TRUE) ::= DEC $ROUND_3 }
14:     { (ROUND_3 == 0) ::= UPD $RUN } }
```

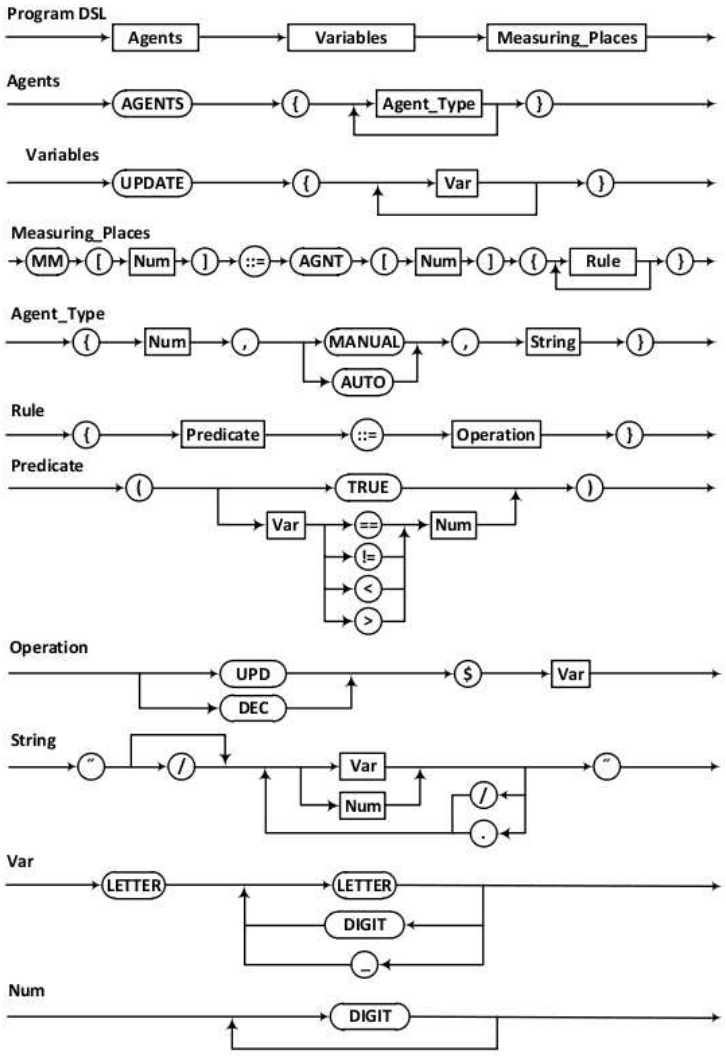

Figure 2. Sintax of DSL EasyTime

The rules for measuring places given in Fig. 1 are determined in EasyTime with the source code presented in Program 2. Each measuring place is denoted with its identification number and connected with an appropriate agent. The rules for the measuring place 4, for example, determine that the event generated by a measuring device at first updates the intermediate time of running INTER_3. In such case, the competitor crosses over the measuring place for the first time (predicate ROUND_3==55) and the time of the second transition TRANS_2 is updated. Then, a decrementing number of laps ROUND_3 follows. Finally, the agent announces the final result when the competitor is in the last lap (predicate ROUND_3==0) and, obviously, the variable RUN is updated.

3.2 The Code Generator

The code generator [5] is performed if the syntax analysis has been successfully completed. When the program fails, the parser provides error messages and stops. The code generator generates the code for each measuring place separately. The generated code is saved in the database. The code generator is developed in the programming language C/C++ as well.

The architecture of the process for which the code is generated needs to be defined before generation takes place. The compiled EasyTime program is executed on a virtual machine. In line with the process parallelization (multi-threading), the virtual machine is for each measuring place defined separately. The architecture of the virtual machine (Fig. 3) is simple. It only consists of a program segment, stack, data segment, instruction counter, and program and status registers. The generated code is loaded into the program segment. On the stack, arithmetical-logical operations are executed. The data segment consists of variables from the database. The instruction counter points to the instruction in the program segment that is currently interpreted. The data register (REG-A) holds the timestamp of the event that

is treated by the agent. The status register (REG-S) holds the status of predicates in a binary form ($TRUE$ if $z == 1$ or $FALSE$ if $z == 0$).

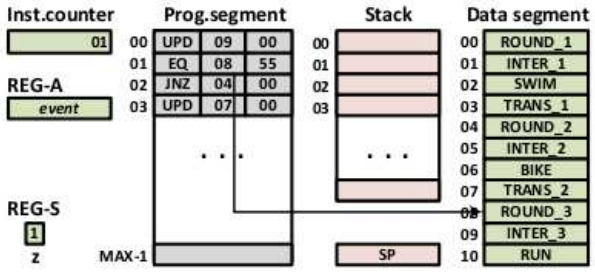

Figure 3. Architecture of the virtual machine

The program instructions loaded into the program segment of a particular virtual machine from the database are defined as a triple $\langle op, p1, p2\rangle$, where op denotes an operation code, and $p1$ and $p2$ are parameters (variable or constant). An instruction set consists of logical instructions EQ and NEQ, operations UPD, DEC and STOP, and branch instruction JNZ. The logical instructions affect the setup of the status register that controls interpretation of the branch instruction.

The code generator generates the code from the data structures that are built by the parser. In practice, the code is only generated from a data structure built from the $Measuring\_Places$ statement. The names of variables assembled in the table are translated into their addresses in the data segment. The addresses appear in instructions as parameters. The data structure built from the $Agents$ statement is designed for the configuration of a particular agent before its execution. An example of a code generated (in a symbolic form) from the source code (Program 1) determining the rules of the agent controlling MP3 is presented in Table 2.

Table 2. The program code for the measuring place 3

IC	OP.C.	P1	P2
00	UPD	5	0
01	DEC	4	0
02	EQ	4	0
03	JNZ	5	0
04	UPD	6	0
05	STOP	0	0

From the example given in Table 2, it can be seen that the generated code is optimized because from three lines of a source code six lines of the executable code are obtained. As each instruction recorded in the database is four bytes long (the operation code, two parameters and delimiter ';'), the size of the compiled program is not critical for the database.

4 Operation of the Agent

The agent that is controlled with the EasyTime program can process the following events:

- batch: manual mode of operation (*MANUAL*),
- online: automatic mode of operation (*AUTO*).

In batch processing, events assembled in a text file are read and written in an appropriate database by the agent. Typically, events captured with computer timers are batch-processed. In this processing mode, the agent checks every second if the file configured in the agent table exists or not. In case that it exists, batch processing begins. At the end of processing, the file is archived and then deleted. Online processing is event-oriented, i.e. each event registered by the measuring device is processed in time. The executing environment introduced by the compiled program written in EasyTime for the measuring time in the triathlon competition as illustrated in Fig. 1, is presented in Fig. 4.

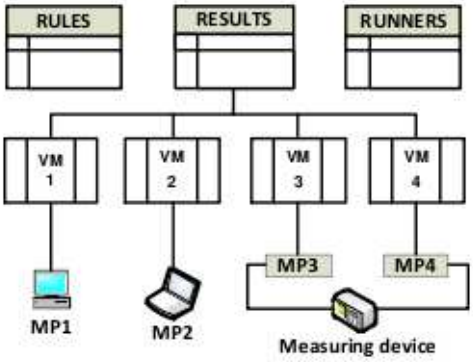

Figure 4. Executing environment of the EasyTime program

In both processing modes, the agent operates with the following database tables: rules ($RULES$), competitors ($RUNNERS$) and results ($RESULTS$). When the agent starts, initialization of the virtual machine for each measuring place is executed. Initialization consists of loading the program code from the rule database table. The code is loaded only once. At the same time, variables in the data segment are initialized. Recording the event processed by the agent can be divided into the following phases:

- reconstruction of the event: the competitor is identified either over the starting number (#) of $RFID$ tag (Fig. 4), while the appropriate virtual machine is determined according to the measuring place MP and the data register is loaded by the timestamp $TIME$,
- reading of the results: from the database table $RESULTS$, the current results of the competitor triggering the event are read,
- mapping of the results: the current results of the competitor are mapped to the data segment of a particular virtual machine,

- code interpretation: the instruction counter is set to zero and the program that was loaded into the program segment starts to execute, and
- recording of the results: the results from the data segment of a particular virtual machine are written to the database table $RESULTS$.

The program in a particular virtual machine is interpreted sequentially, i.e. instruction by instruction, until the instruction STOP is detected.

5 CONCLUSION

When developing universal software for the measuring time in sporting competitions the problem of flexibility in the timing system is often encountered. To cope with the issue, the domain specific language EasyTime was developed enabling rapid adaptation of a timing system to the demands of various sporting competitions. To monitor a new competition, modifications to the source program written in EasyTime need to be compiled and the agent needs to be restarted. As a result, the agent is ready to run in a completely new environment. Using EasyTime in practice shows that organizers can no longer employ specialized and expensive companies to measure time in sporting competitions. However, for large sporting competitions, configuration of the timing system can be simplified. Our future work will be towards upgrading EasyTime with a domain-specific modeling language further simplify the configuration process of the timing system.

REFERENCES

[1] M. Mernik, J. Heering, A. Sloane: When and how to develop domain-specific languages, 2005, ACM computing surveys, vol. 37, no. 4, pp. 316-344
[2] K. Finkenzeller: RFID Handbook, 2010, John Willey, Chichester, UK
[3] N. Wirth: Algorithms + Data Structures = Programs, 1978, Prentice Hall PTR, Upper Saddle River, NJ, USA
[4] A.V. Aho, J.D. Ullman: The theory of parsing, translation, and compiling (Volume I: Parsing), 1972, Prentice Hall PTR, Upper Saddle River, NJ, USA
[5] A.V. Aho, J.D. Ullman: The theory of parsing, translation, and compiling (Volume II: Compiling), 1972, Prentice Hall PTR, Upper Saddle River, NJ, USA
[6] A.V. Aho, M.S. Lam, R. Sethi, J.D. Ullman: Compilers: Principles, Techniques, and Tools with Gradiance, 2007, Prentice Hall PTR, Upper Saddle River, NJ, USA
[7] ChampionChip2010, http://www.championchip.com
[8] RFIDTechnology2010, http://www.rfidtiming.com
[9] Timing2010, http://www.timingljubljana.si

Iztok Fister ml. is a student of the third-year of Computer science and information technologies at the Faculty of Electrical Engineering and Computer Science, University of Maribor.

Iztok Fister graduated in computer science from the University of Ljubljana in 1983. He received his Ph.D. degree from the Faculty of Electrical Engineering and Computer Science, University of Maribor, in 2007. He works as an assistant in the Computer Architecture and Languages Laboratory at the same faculty. His research interests include program languages, operational researches and evolutionary algorithms.