Razvoj in uporaba hibridnega domensko-specifičnega modelirnega jezika hEasyTime za merjenje časa na športnih tekmovanjih

Iztok Fister Jr., Borko Bošković, Iztok Fister, Janez Brest

Univerza v Mariboru
Fakulteta za elektrotehniko, računalništvo in informatiko
Smetanova 17, 2000 Maribor, Slovenija
E-pošta: iztok.fister1@um.si

Abstract

Domain-specific modeling languages (DSML) are languages devoted to ordinary users to create programs in a specific domain visually. This paper presents a development and use of a novel domain-specific modeling language for measuring time in sport competitions. A new approach is called hybrid DSML and can be deployed when domain-specific leanguage already exist. Therefore, developer of this DSL can be mainly focused on building graphical user interface (GUI). The result of this, a hEasyTime DSML is developed and which enables ordinary users to create programs for measuring time in sporting competitions.

1 Uvod

Merjenje časa na športnih tekmovanjih je problem s katerim se vsako leto sooča več organizatorjev športnih prireditev/tekmovanj. Glavni razlog je v tem, da se vsako leto takih prireditev udeležuje več in več ljudi, kar pomeni da organizatorji več ne morejo meriti časa s štoparico, kot se je to dogajalo v preteklosti. Dandanes je zamenjava za štoparico, postala elektronska merilna naprava, ki sestoji iz strojne in programske opreme. Ponavadi je programska oprema že priložena k strojni tako, da organizatorji dobijo celotno merilno napravo v enem paketu. Običajno je ta oprema namenjena merjenju ene same discipline, zato jo je težko prilagajati zahtevam modernih športnih disciplin sestavljenih iz več športnih disciplin, kot so npr. triatlon, duatlon, akvatlon. Pri triatlonih je namreč potrebno meriti 5 različnih kontrolnih točk, kot so: čas plavanja, čas prve tranzicije, čas kolesarjenja, čas druge tranzicije in čas teka.

Da bi se organizatorji uspešno spopadali s takšnimi tekmovanji in bili sposobni brez znanja programiranja konfigurirati merilni sistem, smo razvili domensko-specifični jezik EasyTime [9, 7, 8, 10, 11, 6]. Ta jezik je majhen in je prikrojen točno določeni domeni, t.j. merjenju časa. S pomočjo tega jezika lahko uspešno in v kratkem času prilagodimo merilni sistem zahtevam katerega kakoli tekmovanja oz. športne prireditve. S tem DSL-jem smo se uspešno spopadli tudi z zahtevnim merjenjem Dvojnega ultra triatlona 2009, ki je štel za svetovno prvenstvo. Ta verzija je temeljila na principu prevajalnik/generator [17], kjer se je izvorna koda DSL-ja EasyTime prevedla v kodo za abstraktni stroj.

Ta DSL se je v praksi pokazal kot prezahteven za klasičnega uporabnika oziroma organizatorja tekmovanja, saj se je ta moral naučiti sintakse jezika. Zaradi tega smo začeli z razvojem domensko-specifičnega modelirnega jezika EasyTime II [11], katerega razvoj je zahtevnejši in sestoji iz več faz:

- analize domene,
- gradnje meta-modela z Eclipse Modeling Framework (EMF),
- ustvarjanjem grafičnega modela z Eclipse Graphical Modeling Framework (GMF) in
- ustvarjanjem semantičnega modela v programskem jeziku Java, ter uporabe generatorja modela-v-kodo (angl. model-to-code) za generiranje kode primerne za izvajanje na abstraktnem stroju (angl. abstract machine).

Zaradi prevelikih problemov z ogrodjem Eclipse se je razvoj EasyTime II po tretji fazi ustavil.

V tem prispevku predstavljamo novo vrsto razvoja domensko-specifičnih modelirnih jezikov. Podobno kot pri razvoju EasyTime II smo tudi pri razvoju novega DSML hEasyTime začeli z analizo domene, in nadaljevali z razvojem grafičnega modela s pomočjo ogrodja QtCreator. Semantični model smo vgradili v programski jezik Ruby, ki je bil zadolžen za generiranje kode, katera teče na abstraktnem stroju. Ker v tem primeru ni bilo potrebno razvijati meta-modela, smo ta pristop poimenovalo hibridni DSML.

Struktura članka je naslednja: v poglavju 2 prikažemo šport akvatlon in problem merjenja časa na teh tekmovanjih. Poglavje 3 opisuje razvoj hibridnega domensko-specifičnega jezika hEasyTime, medtem ko je 4 sekcija posvečena praktični uporabi jezika v okviru merjenja časa na športnem tekmovanju akvatlon. V zaključku potegnemo črto pod opravljenim delom in predstavimo ideje za nadaljnje delo.

2 Merjenje časa na akvatlonih

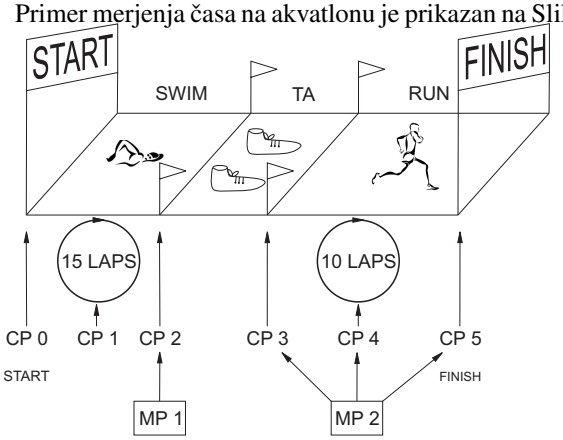
Akvatlon je sorezmerno mlad šport, saj njegove korenine segajo v leto 1965, ko so v Združenih državah Amerike organizirali prvo državno prvenstvo v tej disciplini. Ta

šport lahko uvrstimo v razred multi-športov, kamor spadajo tudi triatlon, duatlon in akvabike. Akvatlon sestoji iz plavanja in teka. Končni čas akvatlona zajema tudi čas, ki ga prebije tekmovalec v tranziciji, kjer se tekmovalec pripravi za tek.

Čeprav je akvatlon zelo podoben triatlonu pa ga je organizatorjem veliko lažje izvesti, predvsem zaradi naslednjih prednosti:

- zmanjša se logistika, ker ni tretje discipline,
- dirka se hitreje konča kot triatlon, in
- zmanjša se prostor tekmovanja.

Primer merjenja časa na akvatlonu je prikazan na Sliki 1.



Slika 1: Merjenje časa na akvatlonskem tekmovanju

Merjenje časa na tekmovanju razdelimo na kontrolne točke (CP), na katerih merimo čas z dvema merilnima napravama. Prva meri izhod tekmovalcev iz vode, medtem ko druga pokrije kar tri kontrolne točke: izhod iz tranzicije, štetje krogov teka in končni čas.

3 Hibridni domensko-specifični modelirni jezik hEasyTime

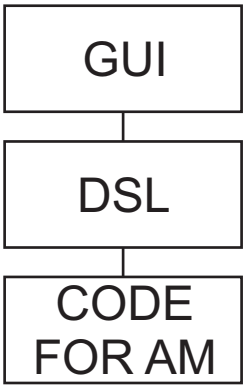
Programiranje je že od nekdaj zelo zahteven in kompleksen proces. Da nekdo napiše zelo dobro in učinkovito programsko kodo je potrebno veliko vaje, znanja, in izkušenj. Programiranja se ne da naučiti čez noč in dobri programerji porabijo tudi več let, da obvladajo vse koncepte programiranja. Zato se znanstveniki in raziskovalci v zadnjih letih trudijo razviti nove metode programiranje, ki programerjem omogočajo uporabo vizualnih orodijn. V tem primeru je programiranje prevedeno na sestavljanje grafičnih elementov in risanja povezav med njimi s pomočjo črt. Eno izmed takšnih vizualnih orodij je tudi domensko-specifični modelirni jezik (angl. domain-specific modeling language (DSML)), ki je namenjen vizualnemu programiranju in prikrojen točno določeni aplikacijski domeni.

Kljub enostavni uporabi DSML-ja, pa konstrukcija tega jezika ni enostavna, saj zahteva od programerjev veliko znanja in izkušenj s področja razvoja programskih jezikov [1, 3]. Za hitrejši razvoj DSML-jev obstaja tudi veliko orodij, ki omogočajo razvoj DSML-jev in podpirajo različne razvojne faze. Nekaj izmed teh orodij je: Eclipse GMF [14], Monticore [15], MetaEdit [5, 13], MetaEdit+ [16].

3.1 Razvoj hibridnega domensko-specifičnega modelirnega jezika hEasyTime

Predlagani pristop razvoja hibridnega domensko-specifičnega jezika lahko uporabimo, ko imamo domensko-specifični jezik že ustvarjen. Ta je predstavljen na Sliki. 2 in sestoji iz naslednjih razvojnih faz:

- definicije grafičnega vmesnika (angl. graphical user interface (GUI)),
- preslikave grafičnih elementov v kontrukte DSL,
- generacija kode za abstraktni stroj (angl. abstract machine (AM)).



Slika 2: Razvoj hibridnega DSML-ja hEasyTime

Če primerjamo ta postopek z razvojem klasičnega DSML-ja, lahko opazimo dve glavni razliki, tj. tukaj nam ni potrebno razvijati meta-modela, in semantični model je že vključen v DSL. Omenjene faze na kratko opisujemo v nadaljevanju.

3.2 Definiranje grafičnega vmesnika

Pri prvi fazi moramo zasnovati izgled našega grafičnega vmesnika. Moramo določiti kateri koncepti aplikacijske domene bodo sploh predstavljeni. Predpogoj za to fazo je dobro opravljena domenska analiza. Ker je ta analiza zelo dobro predstavljena v [9] je tukaj ne obravnavamo podrobno.

3.3 Razvoj grafičnega vmesnika

Grafični vmesnik omogoča človeku interakcijo z računalnikom, kjer so informacije kodirane preko ikon [12]. V našem primeru s pomočjo ikon na grafičnem vmesniku preslikamo grafične elemente v konstrukte DSL jezika (Slika 1).

Ikone predstavljajo naslednje koncepte:

- *plavanje in tek*,
- *tranzicija*,
- zastavice predstavljajo *kontrolne točke*,
- merilne naprave predstavljajo *avtomatskega* in štoparice predstavljajo *manualnega* agenta.

Tabela 1: Preslikava konceptov aplikacijske domene v konstrukte grafičnega vmesnika

Application domain concepts	GUI
Tekmovanje	Vsi grafični elementi na zaslonu.
Dogodki (plavanje, kolo, tek)	Ikone za plavanje, kolo in tek.
Tranzicija	Ikona za tranzicijo (start, cilj).
Kontrolne točke (start, število krogov, cilj)	Ikone za zastavice.
Merilna mesta (update time, decrement lap)	Črte med merilnimi napravami in kontrolnimi točkami.
Agenti (automatični, manualni)	Ikone za merilne naprave in manualne štoparice.

Povezave povezujejo *kotrolne točke* (ikona zastavica) z merilnimi napravami (*avtomatski* agent) in/ali štoparica (*manualni* agent).

Grafični vmesnik lahko ustvarimo na več načinov. Pri našem razvoju smo uporabili knjižnice Qt s pomočjo Qt Creatorja [2, 4]. Qt Creator je ogrodje omogoča razvoj grafičnih uporabniških vmesnikov s pomočjo programskega jezika C++. Sestoji iz naslednjih komponent:

- urejevalnika kode C++ in JavaScript,
- integriranega oblikovalca grafičnega vmesnika,
- orodij za prevajanje kode,
- vizualnega razhroščevalnika.

Grafični vmesnik hEasyTime smo razvili s pomočjo integriranega oblikovalca grafičnega vmesnika. Sestoji iz grafičnega urejevalnika za oblikovanje tekmovanje. Ikone pa predstavljajo koncepte aplikacijske domene (plavanje, tek, merilna naprava...) Vmesnik vsebuje tudi nekatere semantične informacije, kot npr. število krogov.

3.4 Generacija kode

Generacija kode je zadnja faza in se začne takoj, ko uporabnik konča z oblikovanjem v uporabniškem grafičnem vmesniku. Tukaj se elementi, ki predstavljajo koncepte preslikajo v predefinirane konstrukte jezika DSL. Na začetku dobimo kodo za klasični domensko-specifični jezik EasyTime. Ta koda se pa potem prevede v kodo, ki bo tekla na abstraktnem stroju.

Algorithm 1 DSL EasyTime koda

```
1: 1 auto 192.168.225.100;
2: var SWIM := 0;
3: var ROUND1 := 1;
4: var INTER1 := 0;
5: var TRANS1 := 0;
6: var RUN := 0;
7: var ROUND2 := 10;
8: var INTER2 := 0;
9: mp[1] → agnt[1] {
10: (ROUND1 == 0) → upd SWIM;
11: }
12: mp[2] → agnt[1] {
13: (ROUND2 == 10) → upd TRANS1;
14: }
15: mp[1] → agnt[1] {
16: (ROUND2 == 0) → upd RUN;
17: }
```

Algorithm 2 Generirana koda akvatlona

```
1: (WAIT i TRUE BRANCH(FETCH connect(192.168.225.2) STORE INTER1, NOOP)
2: TRUE BRANCH( FETCH ROUND1 DEC STORE ROUND1, NOOP)
3: PUSH 0 FETCH ROUND1 EQ BRANCH( FETCH connect(192.168.225.2) STORE SWIM, NOOP),1)
4: (WAIT i TRUE BRANCH( FETCH connect(192.168.225.2) STORE INTER2, NOOP)
5: PUSH 10 FETCH ROUND2 EQ BRANCH( STORE TRANS1, NOOP)
6: TRUE BRANCH( FETCH ROUND2 DEC STORE ROUND2, NOOP)
7: PUSH 0 FETCH ROUND2 EQ BRANCH( FETCH connect(192.168.225.2) STORE RUN, NOOP), 2)
```

4 Praktična uporaba

Slika 3 predstavlja primer akvatlonskega tekmovanja v grafičnem uporabniškem vmesniku. V vmesniku so predstavljeni start in cilj tekmovanja, ter konfiguracija proge. Tekmovalci morajo na tem tekmovanju preplavati 20 krogov, ter pretečti 10 krogov. Za merjenje pa uporabimo le 2 merilni napravi (Slika 1). Ko uporabnik konča s konfiguriranjem tekmovanja, se začne generacija kode. Najprej generiramo EasyTime kodo(Algoritem 1) in potem poženemo še prevajalnik za to kodo. Rezultat prevajanja je koda, ki se bo izvedla na abstraktnem stroju (Algoritem 2).

5 Zaključek

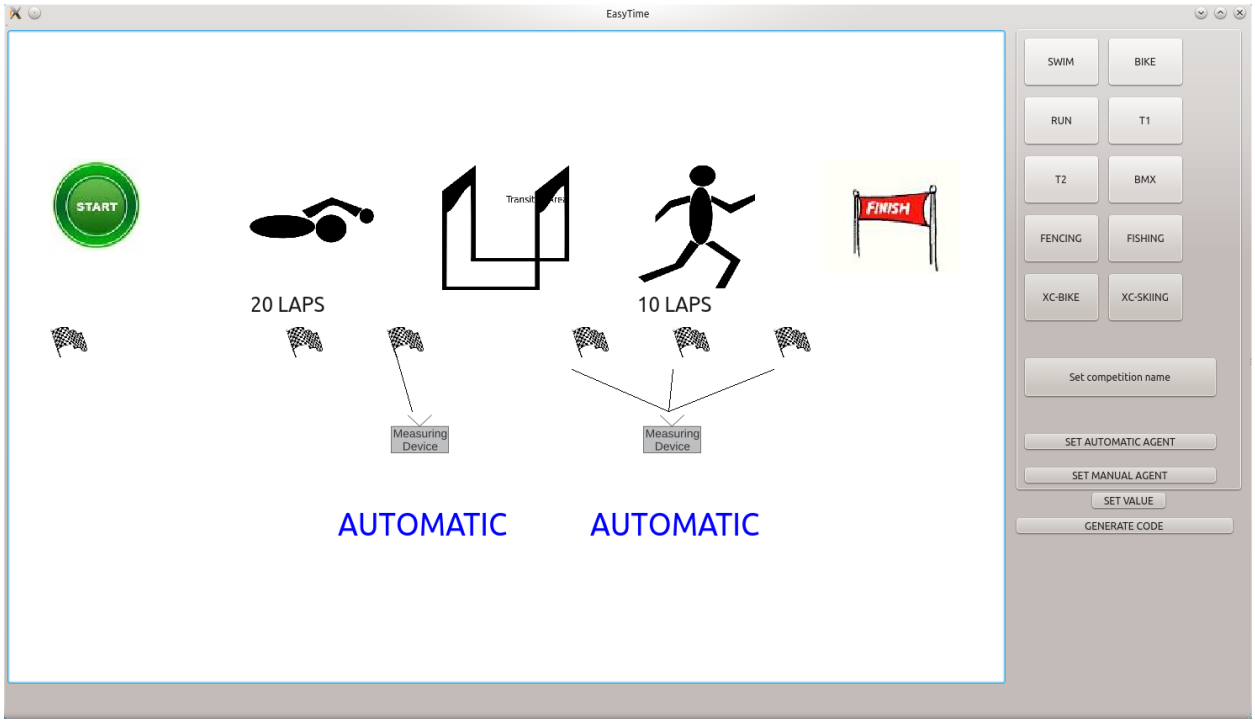
V članku smo predstavili novi pristop razvoja DSML. Uporabili smo ga pri razvoju hibridnega DSML hEasyTime za potrebe merjenja časa na športnih tekmovanjih. Jezik je zelo enostaven za uporabo, saj lahko z njim rokujejo ljudje, kateri nimajo osnovnega programerskega znanja. Praktična uporaba je to dejstvo tudi potrdila. V prihodnosti bi se ta pristop lahko poskusil tudi pri razvoju drugih DSML-jev.

Literatura

[1] A. Aho, M. Lam, R. Sethi, and J. Ullman. *Compilers: principles, techniques, and tools*, volume 1009. Pearson/Addison Wesley, 2007.

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

[3] K. Czarnecki. Generative programming: Methods, techniques, and applications tutorial abstract. *Software Reuse: Methods, Techniques, and Tools*, pages 477–503, 2002.



Slika 3: Uporabniški vmesnik hEasyTime

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

[5] E. EMF. Eclipse modelling framework. `http://www.eclipse.org/modeling/emf/`. Accessed 20 December 2012.

[6] I. Fister and I. Fister Jr. Concept of drafting detection system in ironmans. *Elektrotehniški vestnik*, 78(4):217–222, 2011.

[7] I. J. Fister, M. Mernik, I. Fister, and D. Hrnčić. Implementation of easytime formal semantics using a lisa compiler generator. *Computer Science and Information Systems/ComSIS*, 9(3):1019–1044, 2012.

[8] I. Fister Jr and I. Fister. Measuring time in sporting competitions with the domain-specific language easytime. *Elektrotehniški vestnik*, 78(1-2):36–41, 2011.

[9] 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.

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

[11] I. Fister Jr, T. Kosar, M. Mernik, and I. Fister. Upgrading easytime: from a textual to a visual language. *arXiv preprint arXiv:1208.4126*, 2012.

[12] A. Genco and S. Sorce. *Pervasive systems and ubiquitous computing*. WIT Press, Southampton, Boston, 2010.

[13] E. GMP. Graphical modelling project. `http://www.eclipse.org/modeling/gmp/`. Accessed 20 December 2012.

[14] R. C. Gronback. *Eclipse modeling project: a domain-specific language (DSL) toolkit*. Pearson Education, 2009.

[15] H. Grönniger, H. Krahn, B. Rumpe, M. Schindler, and S. Völkel. Monticore: a framework for the development of textual domain specific languages. In *Companion of the 30th international conference on Software engineering*, pages 925–926. ACM, 2008.

[16] H. Krahn, B. Rumpe, and S. Völkel. Monticore: Modular development of textual domain specific languages. *Objects, Components, Models and Patterns*, pages 297–315, 2008.

[17] M. Mernik, J. Heering, and A. Sloane. When and how to develop domain-specific languages. *ACM computing surveys (CSUR)*, 37(4):316–344, 2005.