Uporaba knjižnice Blockly za razvoj blokovnega jezika

Matej Moravec[1], **Tomaž Kosar**[1], **Marjan Mernik**[1]

[1] *Univerza v Mariboru,*
Fakulteta za elektrotehniko, računalništvo in informatiko
Koroška cesta 46, 2000 Maribor, Slovenia
E-pošta: matej.moravec@student.um.si, tomaz.kosar@um.si, marjan.mernik@um.si

Developing a visual language using Blockly library

Blockly is a client-side JavaScript library for creating visual block programming languages and editors. Visual block programming languages are becoming more and more popular for children and older people who want to learn the basic concepts of programming. They are also used to build more complex programs in different domains. So in this paper we show how to develop a visual block programming language using Blockly library. We show practical example of application in which we create a custom blocks and code generators for each block. From created custom blocks we assemble one example of program, generate code and show the results. In this paper we also mention some pros and cons about use of visual block programming languages.

1 Uvod

Programski jeziki so programerjeva najbolj osnovna orodja. Stroške izdelave nove aplikacije lahko drastično zmanjšamo z izbiro ustreznega programskega jezika. Prav tako so lahko stroški vzdrževanja z izbiro ustreznega programskega jezika občutno manjši. V zadnjih desetletjih je tehnologija programskih jezikov veliko napredovala v smislu tradicionalnih programskih paradigem, kot so funkcijsko, logično in objektno orientirano programiranje. Prav tako je napredovalo vizualno orientirano programiranje [1]. Glavna gonilna sila razvoja je bila in bo boljše izražanje programerjevih idej. Ravno zato so raziskave na področju programskih jezikov neskončna dejavnost in jedro računalništva. V prihodnosti lahko predvidimo nove lastnosti jezikov, nove programske paradigme in mehanizme z boljšim časom prevajanja in zagona.

Lažje izražanje v specifični domeni pa prinašajo tudi domensko specifični jeziki (ang. Domain-Specific Language (DSL)) [2]. DSL je jezik, ki je zasnovan z namenom, da zagotovi notacijo, ki je blizu aplikacijski domeni, in temelji samo na konceptih in lastnostih te domene. Kot tak je DSL sredstvo za opis in generiranje "članov" programske družine v določeni domeni. DSL z zagotavljanjem notacije, ki je blizu aplikacijski domeni, ponuja veliko prednosti pred t. i. splošno namenskimi jeziki (angl. General-Purpose Languages (GPL)) [3]. Ena izmed prednosti je, da pri uporabi DSL ni potrebno znanje splošnega programiranja in ga tako lahko uporabljajo tudi končni uporabniki [4].

Domensko specifični jeziki se v marsičem prekrivajo z domensko specifičnimi modelirnimi jeziki (angl. Domain-Specific Modeling Languages (DSML)) [5]. Ti specificirajo programe z modeli in uporabljajo vizualne domenske koncepte ter tako v nekaterih domenah še dodatno dvignejo stopnjo abstrakcije izven nivoja kodiranja. Končni produkt je lahko na podlagi teh specifikacij generiran na zelo visoki ravni.

Programi, ki jih pišemo z DSML-ji, so v veliki meri podobni programom, ki temeljijo na blokih (angl. block-based programming languages) [6]. Ti so se dobro uveljavili za uvajanje otrok v programiranje [7]. Blokovni jeziki, ki jih poznamo, pokrivajo različne domene, ki učence zabavajo in motivirajo za programiranje preprostih iger (Scratch), interaktivno multimedijo (Pixly), preprosto robotiko (RoboScratch) in mobilne telefone (MIT AppInventor). V tem delu nas je zanimalo, kako se blokovni jeziki razlikujejo od DSML-jev. Naš cilj je bil raziskati to področje programiranja, ki se v zadnjem času zelo razvija – programski jeziki, osnovani na blokih.

Članek je strukturiran v pet poglavij. V drugem poglavju prikažemo zapis vizualne notacije posameznega bloka. V tretjem poglavju predstavimo, kako se na podlagi vizualnega programa v ozadju generira koda, zapisana v poljubnem programskem jeziku. V četrtem poglavju v praksi predstavimo primer uporabe vizualnega jezika. V zadnjem poglavju povzamemo naše ugotovitve o uporabnosti knjižnice Blockly [8] za izdelavo vizualnega jezika in učinkovitost avtomatskega generiranja kode.

2 Zapis vizualne notacije

Blokovni jeziki omogočajo uporabniku specifikacijo programa z različnimi vizualnimi elementi, namesto s tekstom. Z uporabo knjižnice Blockly [9] lahko uporabniku pripravimo poljubno število različnih vizualnih elementov, s katerimi lahko sestavi kompleksen program.

V aplikaciji, izdelani s knjižnico Blockly, posamezen konstrukt v kodi predstavimo z blokom. Določene dele procesov pri konfiguraciji Blockly lahko avtomatiziramo z uporabo spletnega orodja Blockly Developer Tools [9]. Pri tem nam orodje z zlaganjem različnih gra-

dnikov omogoča samodejno generiranje notacije posameznega bloka. Notacija je lahko zapisana s programskim jezikom JavaScript ali z notacijo JSON.

Orodje Blockly Developer Tools vključuje ustvarjanje blokov po meri, gradnjo seznama, kjer se nahaja nabor razpoložljivih blokov, in konfiguriranje spletne delovne površine, ki je namenjena združevanju blokov v končni program. Razvojni proces Blockly z uporabo orodja sestoji iz treh delov:

- Ustvaritev oz. definiranje bloka in možnost izvoza bloka.
- Gradnja seznama, kjer se nahaja nabor blokov, in določitev izgleda privzete delovne površine.
- Konfiguracija oz. določitev funkcionalnosti delovne površine.

Z uporabo orodja Blockly Developer Tools nam je omogočeno poenostavljeno ustvarjanje blokovnih definicij in generatorjev kode za posamezen blok. Bloke lahko enostavno ustvarimo, spremenimo in jih shranimo. Ustvarjene bloke in generatorje kode lahko iz orodja izvozimo in jih uporabimo v aplikaciji. Z uporabo orodja nam je omogočena enostavna konfiguracija seznama, kjer se nahaja nabor razpoložljivih blokov za gradnjo programa. Določimo lahko tudi, kateri bloki se bodo ob zagonu aplikacije že nahajali na delovni površini.

2.1 Primer zapisa vizualne notacije za posamezen konstrukt v kodi

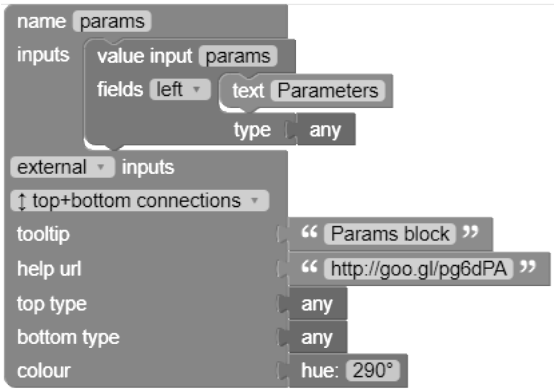
Na vsak blok se navezujemo z njegovim imenom, zato mora biti vsak posamezen blok predstavljen s svojim unikatnim imenom. Videz bloka in njegovo obnašanje opišemo z definicijo bloka, ki vključuje besedilo, barvo, obliko in opis, kako se drugi bloki lahko z njim povežejo [8].

V programski kodi 1 vidimo, da moramo bloku v prvi vrstici najprej definirati njegovo ime. Nato v vrsticah 2-12 definiramo celotno vizualno predstavitev bloka. Vrstice 3-5 definirajo vhod v blok oz. vrednost, ki jo blok lahko prejme. Vrstice po vrsti definirajo najprej oznako vhodne vrednosti, s katero se lahko kasneje pri generiranju kode nanjo navežemo oz. do nje dostopamo. Zatem je definiran tip, ki ga lahko blok na vhodu prejme (v našem primeru je nastavljen na *any* oz. *null*). Z določitvijo tipa vplivamo na pravilno sestavljanje programa. Pri tem Blockly podpira preverjanje več vrst tipov, kot so *Boolean*, *Number*, *String*, itd. V primeru, da želimo k bloku pripeti blok z blokom, ki ne vrača ustreznega tipa, kot je zapisano v definiciji bloka, Blockly teh dveh blokov ne bo povezal med sabo. Za določitvijo tipa je bloku dodano polje, s katerim nanj dodamo poljubno besedilo. Smiselno je, da na blok dodamo besedilo, ki uporabniku pomaga pri razumevanju pomena bloka. Vrstice 6-11 definirajo lastnosti bloka, ki povejo, kako blok spada v kontekst oz. kako se blok lahko v končnem programu poveže oz. združi z drugimi bloki programa. S temi lastnostmi definiramo, kako naj se vhodni bloki k bloku pripnejo, določimo, če se blok lahko pripne k predhodnemu bloku in če se neki drug blok lahko pripne nanj. Z metodo *setColour* definiramo odtenek barve, ki pri definiciji bloka pove, kako bo blok izgledal. S tem uporabniku intuitivno pomagamo razumeti, kateri bloki spadajo skupaj. S funkcijama *setTooltip* in *setHelpUri* uporabniku nudimo dodaten opis bloka oz. koristne informacije o uporabi bloka.

```
1  Blockly.Blocks['params'] = {
2    init: function () {
3      this.appendValueInput("params")
4          .setCheck(null)
5          .appendField("Parameters");
6      this.setInputsInline(false);
7      this.setPreviousStatement(true, null);
8      this.setNextStatement(true, null);
9      this.setColour(290);
10     this.setTooltip("Params block");
11     this.setHelpUrl("http://goo.gl/pg6dPA");
12   }
13 };
```

Programska koda 1: Definicija bloka “params”

Ekvivalent programski kodi JavaScript 1 predstavlja slika 1. Na sliki je prikazana sestava bloka v orodju Blockly Developer Tools. Na podlagi te sestave se nam programska koda 1 samodejno generira.



Slika 1: Vizualna definicija bloka “params”

Na sliki 2 je prikazana vizualna predstavitev bloka “params”, ki je zgrajena na podlagi programske kode 1. Vidimo, da se blok lahko pripne v kontekst drugega bloka in k njemu se lahko pripne naslednji blok v programu. Na desni strani lahko blok na vhodu prejme drug blok, katerega koda se pri generiranju kode združi s kodo bloka “params”. Na bloku podano besedilo “Parameters” pripomore k lažjemu razumevanju, čemu pri gradnji programa je blok namenjen.

Slika 2: Vizualna predstavitev bloka “params”

3 Generiranje kode

Večina aplikacij Blockly mora v programu uporabljene bloke, pretvoriti v izvajalno kodo. V ta namen za vsak

blok zapišemo generator kode. Izvajalna koda je lahko zapisana v različnih jezikih (JavaScript, Python, PHP, Lua, Dart, C#, itd.).

Prva naloga kateregakoli bloka je, da si pri generiranju kode pridobi vse argumente in podatke drugih blokov, ki so z njim združeni na vhodnem polju. Knjižnica Blockly vsebuje več funkcij, ki se lahko uporabijo za to nalogo:

- *getFieldValue*.
- *valueToCode*.
- *statementToCode*.

V programski kodi 2 je prikazan primer generatorja kode za blok "params". V kodi se najprej navežemo na ime bloka, za katerega želimo zapisati generator kode. Ko smo se navezali na ustrezen blok, zapišemo funkcijo, ki predstavlja generator kode. V vrsticah 3-5 s funkcijo *valueToCode* dostopamo do bloka "params", preko katerega pridobimo generirano kodo bloka, ki se nahaja na vhodu bloka "params". Kodo zapišemo v spremenljivko *value_params* in za tem v spremenljivko *code* ter slednjo, ki predstavlja generirano kodo, vrnemo. Spremenljivka *code* v tem primeru prejme samo vrednost spremenljivke *value_params*. V primeru, ko je blok bolj kompleksen in ima več možnih argumentov in parametrov, se v spremenljivki *code* vsi ti argumenti in vrednosti iz vhodov združijo v kodo, ki se vrne kot rezultat funkcije.

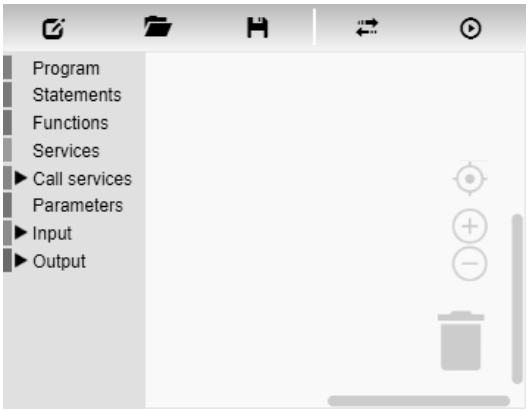
```
1  Blockly.JavaScript['params']=function(block)
2  {
3     var value_params = Blockly.JavaScript
4        .valueToCode(block, 'params',
5           Blockly.JavaScript.ORDER_ATOMIC);
6
7     var code = value_params;
8
9     return code;
10 };
```

Programska koda 2: Generiranje kode za blok "params"

4 Primer uporabe

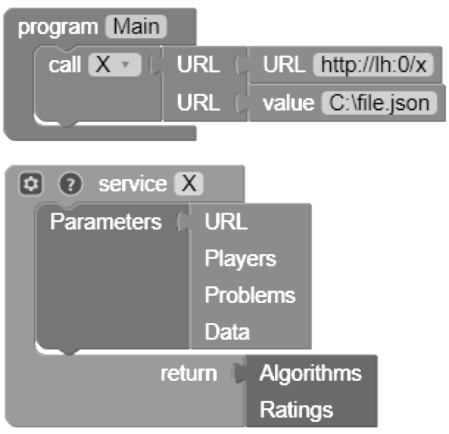
V tem poglavju prikažemo praktični primer izdelane spletne aplikacije. Praktični primer obsega že izdelane bloke, zapisano generiranje kode za vsak blok in funkcionalnosti, kot so priprava nove delovne površine, odpiranje že sestavljenih programov, shranjenih v XML-zapisu, itd.

Na sliki 3 je prikazana spletna aplikacija, ki jo lahko uporabljamo v brskalniku. Strukturirana je iz treh delov. Na levi strani vsebuje seznam vseh blokov, ki jih lahko uporabimo pri gradnji programa. Na vrhu aplikacije se nahaja menijska vrstica, v kateri se nahajajo gumbi, ki predstavljajo funkcionalnosti, namenjene za lažjo pouporabo že sestavljenih programov, in gumbi za generiranje izvajalne kode v programski jezik C#. Na desni strani aplikacije največji del predstavlja površina, na kateri lahko uporabnik sestavi želeni program. Na skrajni desni strani delovne površine se nahajajo gumbi, katerih namen je lažji pregled nad sestavljenim programom. Te funkcionalnosti omogoča knjižnica Blockly in z njimi lahko bloke v sestavljenem programu povečujemo oz. zmanjšujemo, jih na delovni površini centriramo in s potegom posameznega bloka ali skupka blokov na ikono v obliki koša blok oz. skupek blokov izbrišemo. Povečevanje oz. zmanjševanje in centriranje blokov nam koristi predvsem v primeru, ko imamo sestavljen bolj kompleksen program in se želimo po tem programu pomikati.



Slika 3: Aplikacija Blockly

Z uporabo spletne aplikacije lahko sestavimo različne programe. Primer sestavljenega programa prikazuje slika 4. Program je sestavljen iz dveh blokov, ki lahko vsebujeta sklop drugih blokov in v generirani kodi predstavljata funkciji. Vsak izmed blokov vsebuje druge bloke, ki predstavljajo telo tega bloka oz. funkcije. Prvi blok predstavlja "glavni" program, v katerem se kliče funkcija "X", ki pa je sestavljena v drugem bloku. Ob klicu funkcije se uporabijo še bloki, ki predstavljajo vrednosti parametrov, ki jih ob klicu pošljemo v funkcijo. V drugem bloku so vsebovani bloki, ki predstavljajo strukturo funkcije "X" in na podlagi katerih se znotraj funkcije kliče storitev. Izhod iz funkcije določimo s pripetim blokom ob napisu "return". Pri gradnji programa je pomembno, da pripnemo vsebino (bloke) vsem tistim blokom, ki so strukturirani tako, da imajo definirane vhode. V primeru, da bloka na vhodu ne pripnemo, bo pri generiranju manjkal del kode in programa ne bo mogoče izvesti.



Slika 4: Program Blockly

Ko imamo želeni program sestavljen, ga lahko z uporabo aplikacije, pretvorimo v generirano kodo. Pri generiranju se koda, ki jo predstavlja posamezen blok v sestavljenem programu, združi. Koda se v primeru naše aplikacije generira v programski jezik C#. Programska koda 3 predstavlja generirano kodo sestavljenega programa na sliki 4. Odsek kode med vrsticami 1-5 predstavlja generirano kodo iz prvega "večjega" bloka, medtem ko vrstice 7-13 predstavljajo generirano kodo drugega "večjega" bloka.

```
1  static void Main(string[] args)
2  {
3     var X = CallX("http://lh:0/x",
4        "C:\file.json");
5  }
6
7  static string CallX(string url, string json)
8  {
9     WSX.MyXWebService ws =
10       new WSX.MyXWebService();
11
12    return ws.X(json);
13 }
```

Programska koda 3: Generirana koda programa Blockly

5 Zaključek

V članku smo pokazali, kako izdelamo blokovni jezik s pomočjo uporabe knjižnice Blockly. Pri izdelavi posameznih blokov smo prišli do spoznanja, da moramo najprej dobro poznati domeno, za katero razvijamo blokovni jezik, tako lahko uporabniku najbolj poenostavimo "programiranje" oz. sestavljanje programa. Poleg izdelave je tudi pomembno, kako za vsak posamezen blok zapišemo generator kode, saj se ob sestavljenem programu generatorji kode posameznih blokov združijo in predstavljajo končni program.

Ustvarili smo spletno aplikacijo, ki vsebuje seznam ustvarjenih blokov različnih oblik. Vsak blok v seznamu predstavlja določen programski konstrukt in ima zapisan generator kode za programski jezik C#. Slabost pri programiranju s pomočjo blokovnega jezika je, da je uporabnik omejen le na ponujene programske konstrukte. Blokovni jeziki pa zagotavljajo to prednost, da ne moremo narediti sintaktičnih napak in uporabljati programskih konstruktov, ki ne spadajo skupaj.

V prihodnje načrtujemo obogatitev aplikacije z dodanimi še več programskimi konstrukti in generatorji kode za sestavljanje programa. Na podoben način, kot si sedaj uporabnik lahko ustvari nov blok s svojim imenom (ki predstavlja spremenljivko), bomo dodali še opcijo, da si bo lahko uporabnik ustvaril nov blok s svojim imenom, ki bo predstavljal funkcijo. Uporabniku bomo tudi omogočili zagon sestavljenega programa znotraj spletne aplikacije.

Literatura

[1] M. Boshernitsan and M. S. Downes. *Visual programming languages: A survey*. Computer Science Division, University of California, 2004.

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

[3] T. Kosar, S. Gaberc, Jeffrey C. Carver, and M. Mernik. Program comprehension of domain-specific and general-purpose languages: replication of a family of experiments using integrated development environments. *Empirical Software Engineering*, Feb 2018.

[4] T. Kos, T. Kosar, J. Knez, and M. Mernik. From DCOM interfaces to domain-specific modeling language: A case study on the Sequencer. *Computer Science and Information Systems*, 8(2):361–378, 2011.

[5] J. P. Tolvanen and S. Kelly. Defining domain-specific modeling languages to automate product derivation: Collected experiences. In *International Conference on Software Product Lines*, pages 198–209. Springer, 2005.

[6] D. Bau, J. Gray, C. Kelleher, J. Sheldon, and F. Turbak. Learnable programming: blocks and beyond. *Communications of the ACM*, 60(6):72–80, 2017.

[7] D. Weintrop and U. Wilensky. To block or not to block, that is the question: Students' perceptions of blocks-based programming. In *Proceedings of the 14th International Conference on Interaction Design and Children*, IDC '15, pages 199–208, New York, NY, USA, 2015. ACM.

[8] Google. Blockly. `https://developers.google.com/blockly/`. Dostopano 5. 7. 2018.

[9] Google. Blockly developer tools. `https://developers.google.com/blockly/guides/create-custom-blocks/blockly-developer-tools`. Dostopano 5. 7. 2018.