

Modellek programozott feldolgozása dokumentáció

Ulicska Gergely

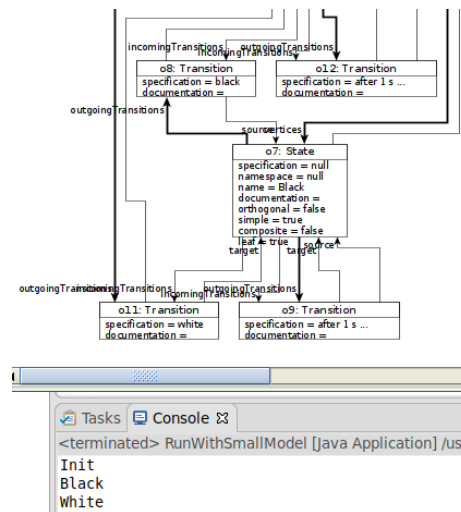
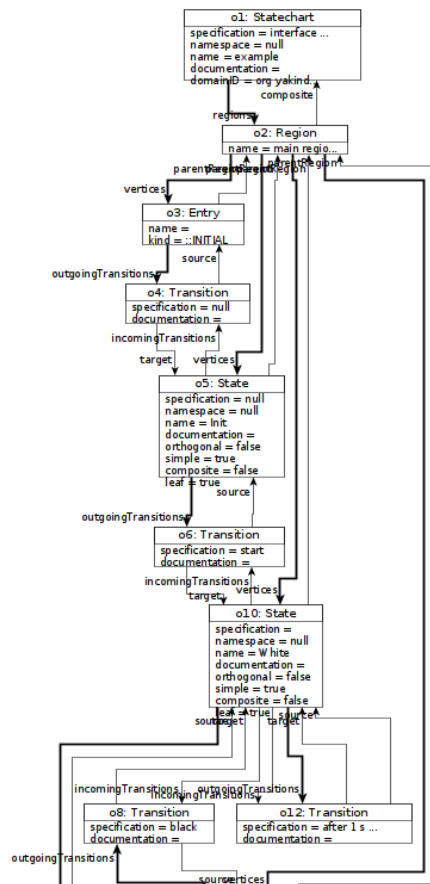
March 8, 2021

1 Előkészületek

A labor során az alábbi GitHub repository-t használtam: <https://github.com/ulicskagergo/retelab2>. A repository-ban megtalálhatóak a forráskód mellett jelen dokumentum és a benne szereplő képek (teljes méretükben).

2 Modell bejárása

A modell bejárása során, a fordítás után az alább látható állapotok álltak elő. A grafikus megjelenítésben a yEd szerkesztőprogramot használtam, de a második kép alján látható a futtatás parancssori kimenete is.



A csapda állapotok kipróbálása érdekében az Init állapotból kimenő start tranzíciót áthúztam, hogy a Black állapotból tartson a White állapotba, így a program észlelte ezt és kiírta: "Trap: Init".

Az üres nevű állapotok ellenőrzését ugyancsak az Init állapottal teszteltem, melynek eredménye az alábbiakban látható:

```
->
Trap:
State has no name. Maybe 2108798232?
Black
Black -> White
Black -> Black
Black -> White
White
White -> Black
White -> White
```

Az első sorban látható, hogy a program a belépési pontja után azonnal a név nélküli állapotba lép (majd külön ki is írja azt, ez a második, üres sor), majd a harmadik sorban megjelöli ezt az állapotot csapdaként, végül a negyedik sorban

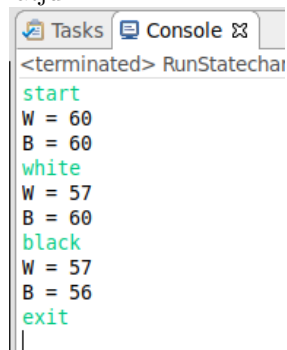
tájékoztató, hogy ennek az állapotnak nincs neve, melyre opcionális választásként egy egyszerű véletlenszerű számot generál.

3 Yakindu kódgenerátor használata

Az `example.sgen` fájlban a *GeneralFeatures* részhez az alábbiakat adtam hozzá:

1. **InterfaceObserverSupport** `false`-ra állításával kikapcsoltuk a listener interfészek létrehozását a state machine-hoz;
2. **RuntimeService** beállításával engedélyeztük a futásidejű service-t, ami elsüti a cycle state machine alapú futási ciklust;
3. **TimerService** engedélyezésével a Java util-ok Timer-ét használhatjuk.

A konzolról olvasó program megírása után az alábbi (interaktív) kimenetet láthatjuk:

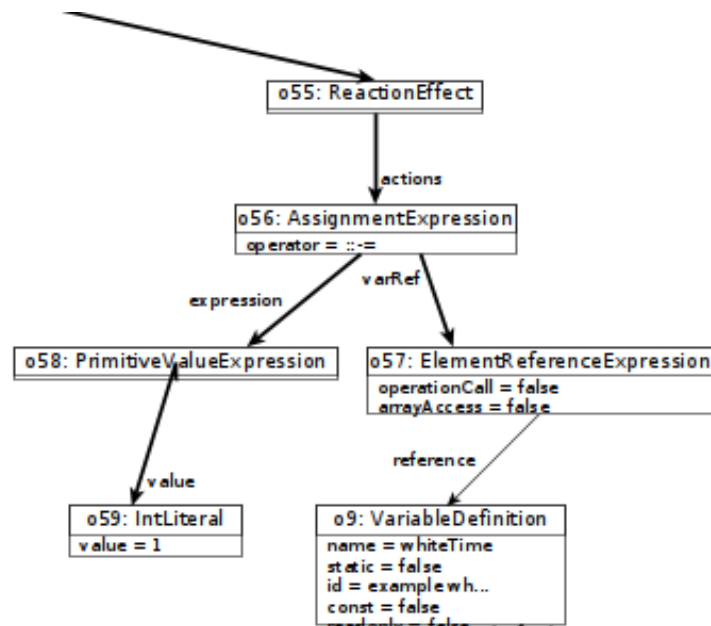


```
<terminated> RunStatechar
start
W = 60
B = 60
white
W = 57
B = 60
black
W = 57
B = 56
exit
```

4 Saját kódgenerátor készítése

Immáron az új (BigModel) konfigurációval futtatva az alkalmazást, az előzőekhez képesti különbség szembetűnő: első rápillantásra is látszik, hogy a gráf lényegesen megnőtt, belenyúlva pedig látszódnak az állapotok nevei, olyanok is, melyek az eddigi gráfon nem látszódtak, mivel már megjelennek az eddig nem látott trigger-ek és effect-ek is (a Transition-öknél) és a scope rész is, mely később a belső állapotok és események lekérdezésére használható.

A diagrammot jobban megnézve láthatóak lesznek a kapcsolatok, így például a `o50`-es tranzícióról megállapíthatjuk, hogy az jelöli a `White` állapot hurkát, mely állapotból az "effect" kapcsolaton továbbhaladva láthatjuk a `whiteTime -= 1` történést is. Ennek "kivitelezője" az `o55 ReactionEffect`, mely `AssignmentExpression`-höz csatlakozik, melynek egyik oldalán a `PrimitiveValueExpression` és `IntLiteral` egy `1`-es értékkel található, másik oldalán pedig egy `ElementReferenceExpression` változó referencia a `whiteTime` nevű változó definícióra.



A kódgenerátor működése azon alapul, hogy míg a statikus részeket egyszerűen kiírja (importok, függvénynevek, Scanner, state machine inicializálása stb) addig a kimeneti kódban két változó dolog van:

- a bemenő események és
- a belső változók értéke.

Ezek megvalósítását a Scope-on keresztüli lekérdezéssel tettem, az eseményeket a `getEvents`-el, míg a belső változókat a `getVariables`-el lehet elérni. Ezeket lekérdezve (és az első betűjüket a Java szintaktikája miatt nagybetűsítve) ki lehet írni őket a case struktúra feltételébe és függvényhívásába, valamint a `getSCInterface` függvényen hívott getter metódus nevébe. Így tesztelve a programot különböző modell bemenetekre (például a state chartot magyarosítva: black helyett fekete stb.) mindig a helyes forráskódot kapjuk.

```
Main.java  RunStatechart.java  example.sct
Scanner scanner = new Scanner(System.in);\r\n" +
String readed = \"\";\r\n" +
while(true) {\r\n" +
    readed = scanner.nextLine();
    switch(readed) {
    case "exit":
        System.exit(0);
    case "start":
        s.raiseStart();
    case "feher":
        s.raiseFeher();
    case "fekete":
        s.raiseFekete();
    }
    s.runCycle();
    print(s);
}

}

public class RunStatechart {

    public static void main(String[] args) throws IOException {

        ExampleStatemachine s = new ExampleStatemachine();
        s.setTimer(new TimerService());
        RuntimeService.getInstance().registerStatemachine(s, 200);
        s.init();
        s.enter();
        s.runCycle();

        Scanner scanner = new Scanner(System.in);
        String readed = "";
        while(true) {
            readed = scanner.nextLine();
            switch(readed) {
            case "exit":
                System.exit(0);
            case "start":
                s.raiseStart();
            case "feher":
                s.raiseFeher();
            case "fekete":
                s.raiseFekete();
            }
            s.runCycle();
            print(s);
        }
    }

    public static void print(IEExampleStatemachine s) {
        System.out.println("F = " + s.getSCInterface().getFeherId());
        System.out.println("F = " + s.getSCInterface().getFeketeId());
    }
}
```