



Modellek programozott feldolgozása

Rendszertervezés laboratórium 1

Jegyzőkönyv

2021. tavasz

Készítette	Tieger Balázs (FMZ298)
Dátum	2021. március 8.
GitHub hivatkozás	https://github.com/Tiegris/rete-lab-2
bónusz feladat	elkészült

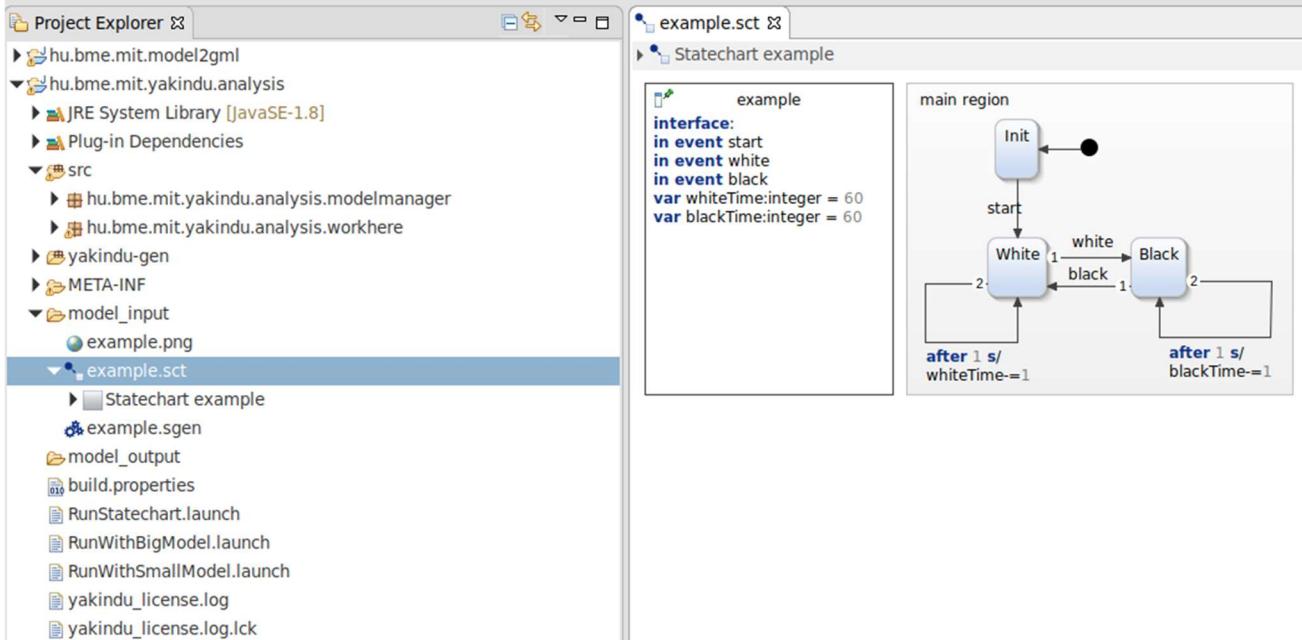
1 Előkészületek

Helyben futtatott virtuális gépen oldottam meg a feladatokat.

Létrehoztam egy gip repository-t (<https://github.com/Tiegris/rete-lab-2>), abban dolgoztam.

Hogy tudjam verziókezelni a munkámat, a REMO_WS mappát átmozgattam a virtuális gépen egy verziókövetett mappába.

A leírás alapján sikeresen importáltam és átnéztem a projektet.



Az importálás után commitoltam egyet.

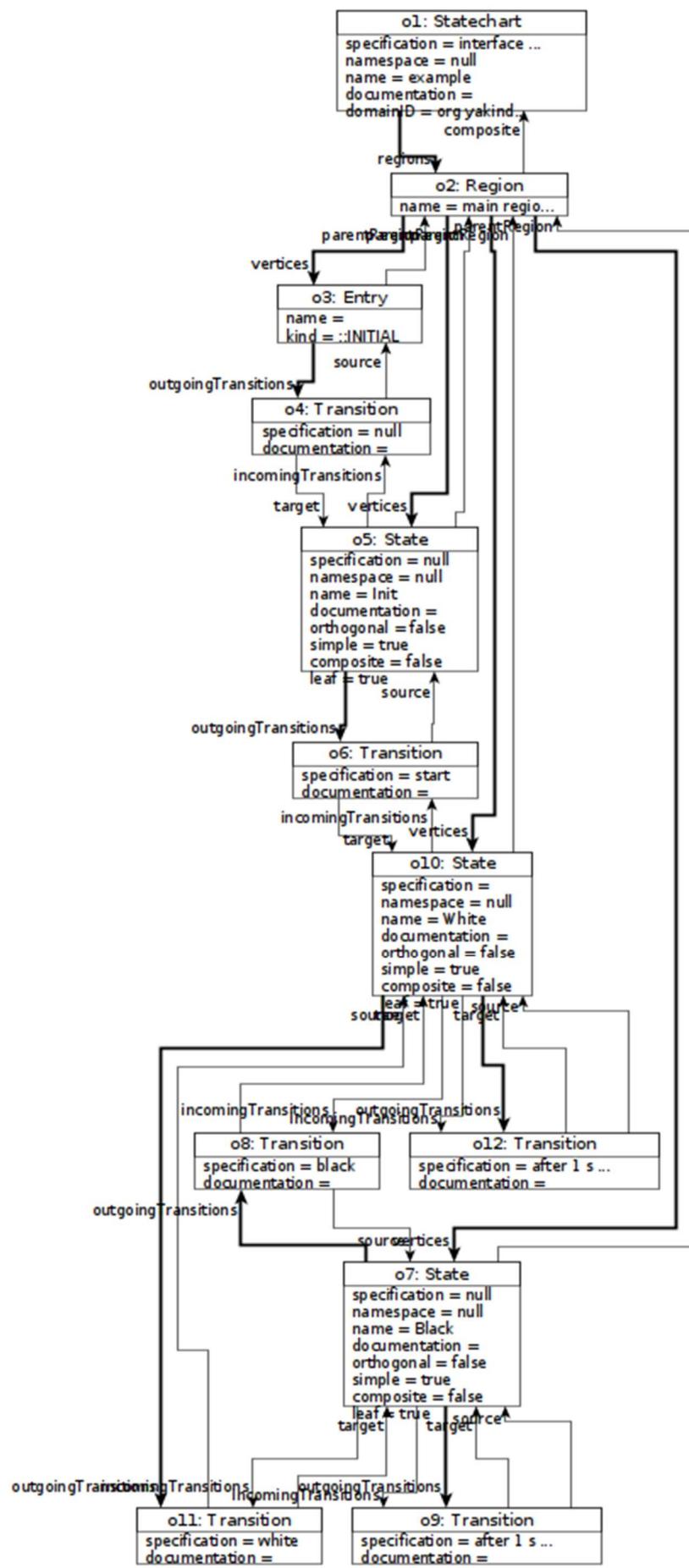
2 Modell Bejárása

Futtattam a RunWithSmallModel.launch -ot.

Konzol kimenete:

```
<terminated> RunWithSmallModel [Java Application] /usr/lib/jvm/java-11-openjdk-amd64/bin/java (Mar 8, 2021, 4:32:32 PM)  
Init  
Black  
White
```

A generált gráf:

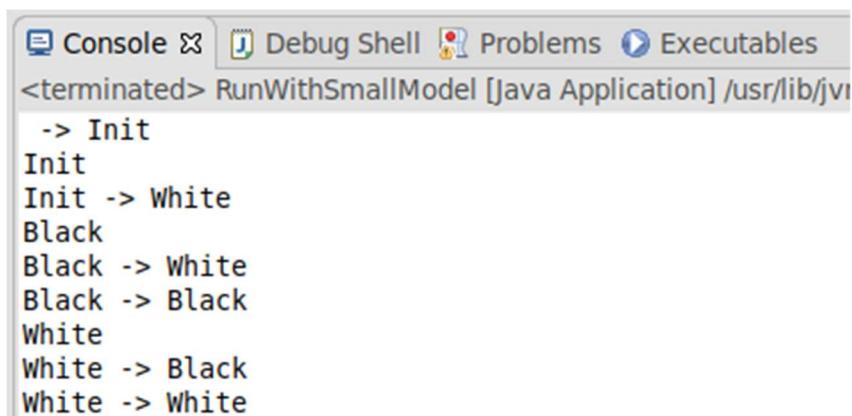


2.1 Kiegészítések

Tranzíciók kiírása kódrészlet:

```
//tranzakciok kiirasa
if(content instanceof Transition) {
    Transition t = (Transition) content;
    String source = t.getSource().getName();
    String target = t.getTarget().getName();
    System.out.println(source + " -> " + target);
}
```

Konzol kimenet:



```
Console & Debug Shell Problems Executables
<terminated> RunWithSmallModel [Java Application] /usr/lib/jvm/java-1.8.0-e...
    -> Init
Init
Init -> White
Black
Black -> White
Black -> Black
White
White -> Black
White -> White
```

Az első átmenetnél azért nincs kiinduló állapot, mert az Init a kezdőállapot.

Csapda állapotokat kereső függvény:

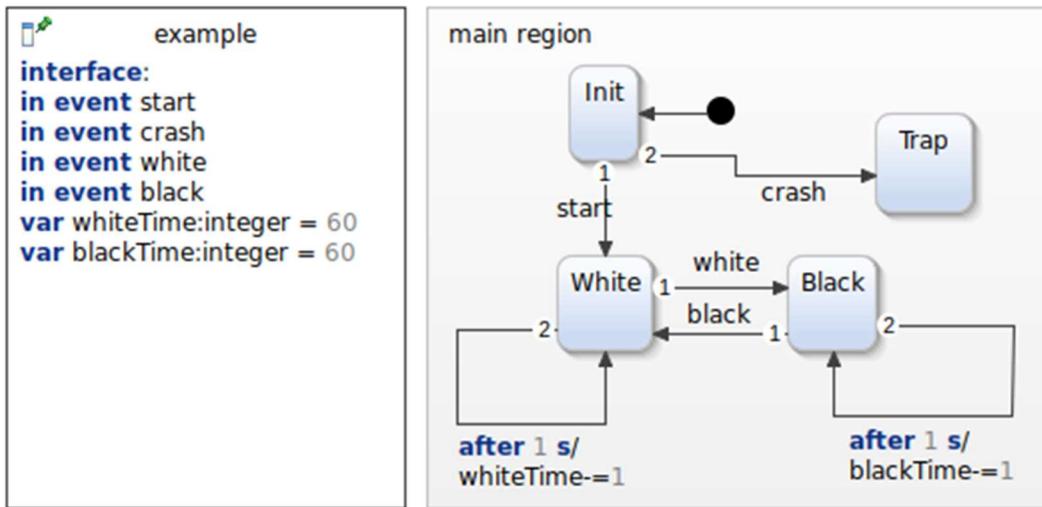
```
public static ArrayList<String> SearchTraps(Statechart s) {
    TreeIterator<EObject> iterator;
    ArrayList<String> allStates = new ArrayList<String>();

    iterator = s.eAllContents();
    while (iterator.hasNext()) {
        EObject content = iterator.next();
        if(content instanceof State) {
            State state = (State) content;
            allStates.add(state.getName());
        }
    }

    iterator = s.eAllContents();
    while (iterator.hasNext()) {
        EObject content = iterator.next();
        if(content instanceof Transition) {
            Transition t = (Transition) content;
            //Minden olyan állapot, ami source-ja valamifele transition-nak, az nem lehet csapda,
            //mert aból megy ki transition.
            String source = t.getSource().getName();
            allStates.remove(source);
        }
    }

    //csak az maradt benne ami csapda állapot.
    return allStates;
}
```

Hozzáadott csapda állapot:



Futtatás eredménye a módosított modellel:

The screenshot shows the Eclipse IDE's 'Console' tab displaying the execution trace of the statechart. The trace starts at the 'Init' state, moves to 'White', then to 'Black', and back to 'White', before finally reaching the 'Trap' state. The output is as follows:

```
<terminated> RunWithSmallModel [Java Application] /usr/lib/jvm/ja
-> Init
Init
Init -> White
Init -> Trap
Black
Black -> White
Black -> Black
White
White -> Black
White -> White
Trap
Csapda állapotok:
Trap
```

Tehát megtalálja a csapda állapotokat, és csak azokat.

Név javasló függvény:

```
public static void SuggestNamesAndPrint(Statechart s, ArrayList<State> unnamed) {
    if (unnamed.size() == 0) {
        System.out.println("No unnamed states found");
        return;
    }

    System.out.println("Unnamed states found, count: " + unnamed.size());
    System.out.println("Suggested names:");

    TreeIterator<EObject> iterator;
    ArrayList<String> allNames = new ArrayList<String>();

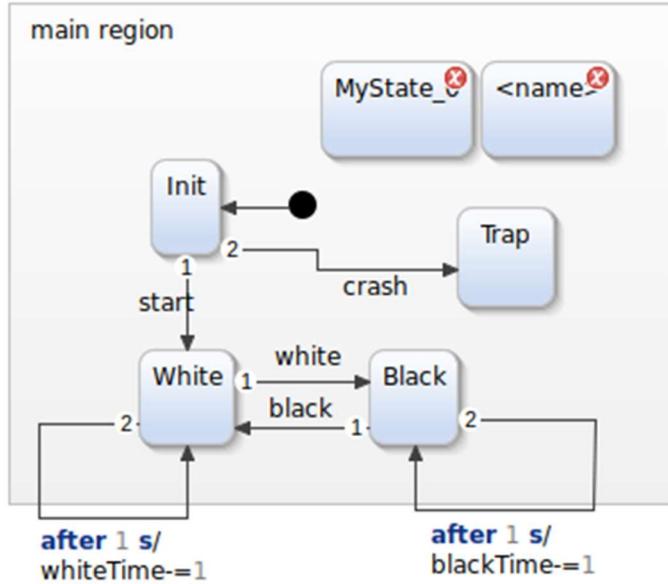
    iterator = s.eAllContents();
    while (iterator.hasNext()) {
        EObject content = iterator.next();
        if(content instanceof State) {
            State state = (State) content;
            allNames.add(state.getName());
        }
    }

    int k=0;
    for (int i = 0; i < unnamed.size(); i++) {
        String suggestion;
        do {
            suggestion = "MyState_" + k;
            k = k+1;
        } while (allNames.contains(suggestion));
        System.out.println(suggestion);
    }
}

public static ArrayList<State> FindUnnamed(Statechart s) {
    TreeIterator<EObject> iterator;
    ArrayList<State> unnamedStates = new ArrayList<State>();

    iterator = s.eAllContents();
    while (iterator.hasNext()) {
        EObject content = iterator.next();
        if(content instanceof State) {
            State state = (State) content;
            if (state.getName() == null || state.getName().isEmpty()) {
                unnamedStates.add(state);
            }
        }
    }
    return unnamedStates;
}
```

Módosított modell, a névjavaslat kipróbálására:



Kimenet:

```
Console Debug Shell Problems Executables
<terminated> RunWithSmallModel [Java Application] /usr/lib/j
-> Init
Init
Init -> White
Init -> Trap
Black
Black -> White
Black -> Black
White
White -> Black
White -> White
Trap
MyState_0
null
Csapda állapotok:
Trap
MyState_0
null
Unnamed states found, count: 1
Suggested names:
MyState_1
```

A függvény figyel arra is, hogy ne javasoljon olyan nevet, ami már használatban van.

Git Commit, Git Push

3 Yakindu kódgenerátor használata

Hozzáadtam az új sorokat.

The screenshot shows a portion of the Yakindu Statechart Designer documentation. It includes code snippets for Java and C++, and detailed explanations of features like GeneralFeatures, InterfaceObserverSupport, RuntimeService, and TimerService. A 'GeneralFeatures' section is highlighted, describing its purpose and parameters. Below the main text, there are sections for 'Example' and 'Help'.

```
feature GeneralFeatures {
    Inte
    Runt
    Time
} The GeneralFeatures feature allows to configure additional services to be generated along with the state machine. Per default, all parameters are false, meaning to disable the corresponding features, respectively. GeneralFeatures is an optional feature and has the following parameters:
```

- *InterfaceObserverSupport* (Boolean, optional): Enables/disables the generation of listener interfaces for the state machine.
- *RuntimeService* (Boolean, optional): Enables/disables the generation of a runtime service that triggers the run cycle of a cycle-based state machine.
- *TimerService* (Boolean, optional): Enables/disables the generation of a timer service implementation using `java.util.Timer`.

Example: [?](#) [Help](#)

Mentés, majd futtatás után meg is jelentek új fájlok a yakindu-gen mappában, RuntimeService.java és TimerService.java (ezek lettek true-ra állítva)

A dokumentáció szerint ezekkel új kódok generálását kapcsoltuk be. A TimerService generálását és a RuntimeService generálását.

Vezérelhető sakkóra:

```
public static void Parse(ExampleStateemachine sm, String s) {
    //Minden beolvasott sor után írjuk ki az összes változó (WhiteTime és BlackTime) értékét!
    //Itt a sor beolvasása után egyből kiírom, mert így értelmeztem,
    // hogy még a parancs értelmezése előtt írijam ki.
    print(sm);
    switch (s) {
        case "start":
            sm.raiseStart();
            sm.runCycle();
            break;
        case "white":
            sm.raiseWhite();
            sm.runCycle();
            break;
        case "black":
            sm.raiseBlack();
            sm.runCycle();
            break;
        case "exit":
            System.exit(0);
    }
}
```

Main fv.:

```
ExampleStateemachine s = new ExampleStateemachine();
s.setTimer(new TimerService());
RuntimeService.getInstance().registerStateemachine(s, 200);
s.init();
s.enter();
s.runCycle();
print(s);

Scanner sc = new Scanner(System.in);
while (true)
    Parse(s, sc.nextLine());
```

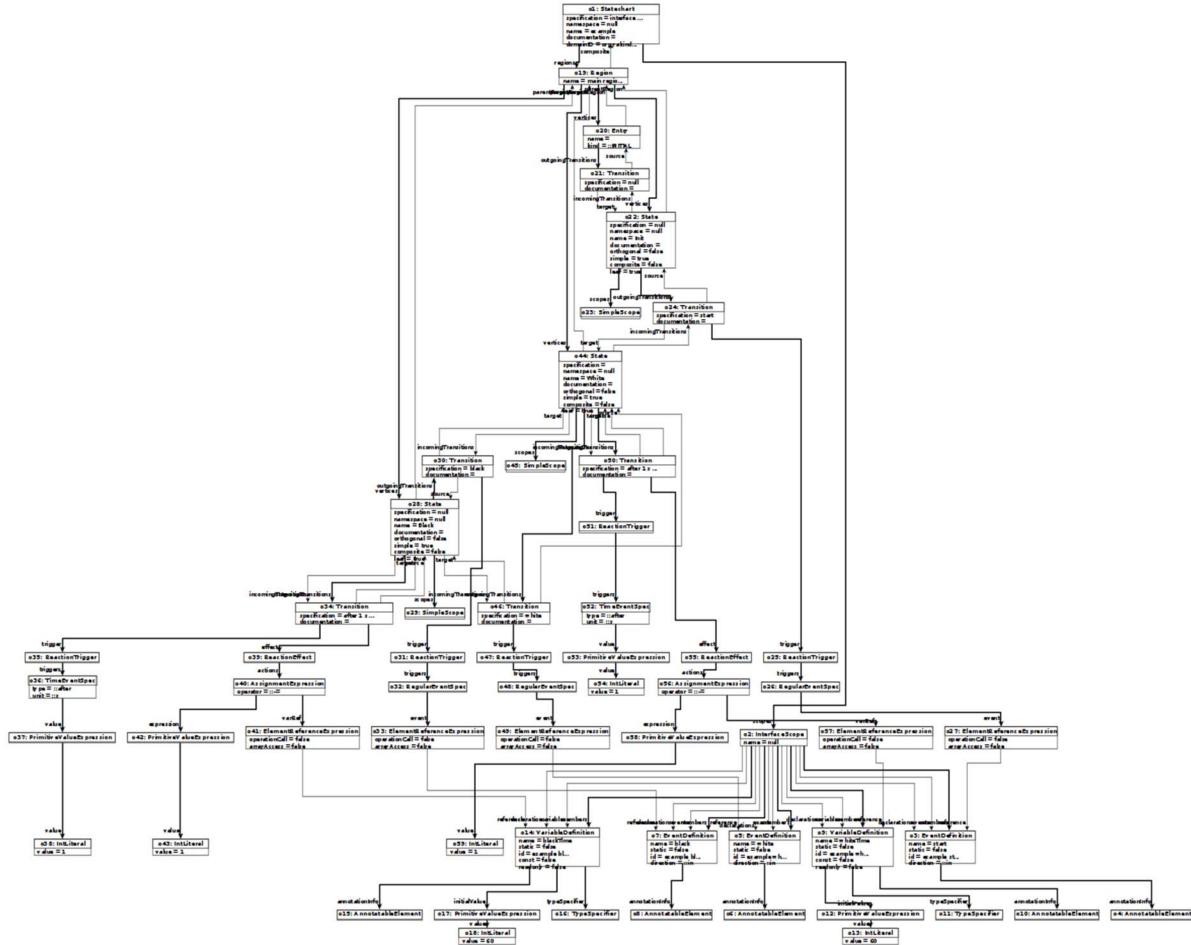
Kipróbálás:

```
W = 60
B = 60
white
W = 60
B = 60
black
W = 60
B = 60
start
W = 60
B = 60
white
W = 48
B = 60
black
W = 48
B = 55
white
W = 44
B = 55
white
W = 44
B = 53
white
W = 44
B = 51
exit
W = 44
B = 50
```

Git Commit, Git Push

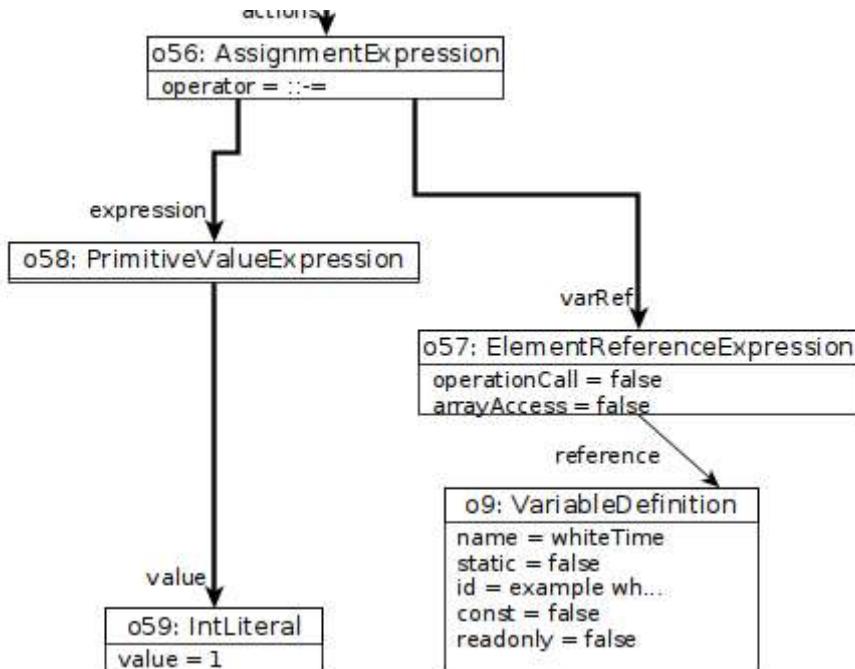
4 Saját kódgenerátor készítése

Futtatás után generált gráf:



Sokkal nagyobb gráfot generált.

whiteTime=-1 absztrakt szintaxisa:



Ehhez kézzel át kellett rendezzem a gráfot, hogy ilyen szépen látszódjon.

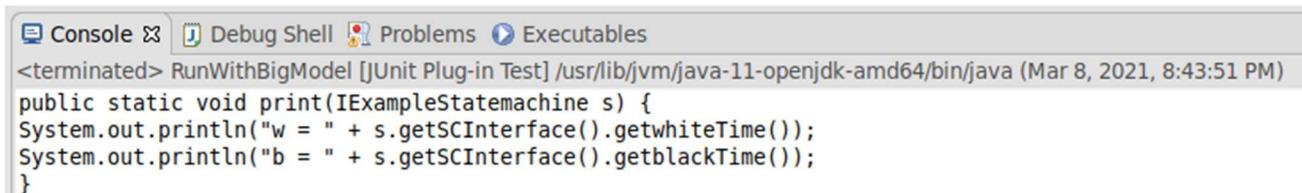
Összes bemenő esemény és belső változó kiírása:

```
//osszes event es osszes valtozo kiirasa
public static void Task4_3(Statechart s) {
    TreeIterator<EObject> iterator = s.eAllContents();
    while (iterator.hasNext()) {
        EObject content = iterator.next();
        if(content instanceof EventDefinition) {
            EventDefinition a = (EventDefinition) content;
            if (a.getDirection() == Direction.IN) //csak a bemenő események kiírása
                System.out.println(a.getName());
        } else
        if(content instanceof VariableDefinition) {
            VariableDefinition b = (VariableDefinition) content;
            System.out.println(b.getName());
        }
    }
}
```

4.4 generáló függvény:

```
public static void Task4_4(Statechart s) {
    TreeIterator<EObject> iterator = s.eAllContents();
    System.out.println("public static void print(IExampleStatemachine s) {");
    while (iterator.hasNext()) {
        EObject content = iterator.next();
        if(content instanceof VariableDefinition) {
            VariableDefinition b = (VariableDefinition) content;
            System.out.println("System.out.println(\""+b.getName().charAt(0)+" = "+ s.getSCInterface().get"+b.getName()+"();");
        }
    }
    System.out.println("}");
}
```

kimenete:



```
<terminated> RunWithBigModel [JUnit Plug-in Test] /usr/lib/jvm/java-11-openjdk-amd64/bin/java (Mar 8, 2021, 8:43:51 PM)
public static void print(IExampleStatemachine s) {
System.out.println("w = " + s.getSCInterface().getwhiteTime());
System.out.println("b = " + s.getSCInterface().getblackTime());
}
```

A kész generálóra külön osztályt írtam, ez a Generator.java fájlban található.

A program kimenetével lecseréltem a 3.5-ös feladat megoldását, majd megnéztem a GitHub Desktop alkalmazásban a fájl módosulásait.

A módosítások:

```
+ public static void print(IExampleStatemachine s) {
+     System.out.println("W = " + s.getSCInterface().getWhiteTime());
+     System.out.println("B = " + s.getSCInterface().getBlackTime());
+ }
+
-
- public static void print(IExampleStatemachine s) {
-     System.out.println("W = " + s.getSCInterface().getWhiteTime());
-     System.out.println("B = " + s.getSCInterface().getBlackTime());
- }
```

```

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();
print(s);
s.raiseStart();
s.runCycle();
System.in.read();
s.raiseWhite();
s.runCycle();
print(s);
System.exit(0);
*/
ExampleStateMachine s = new ExampleStateMachine();
s.setTimer(new TimerService());
RuntimeService.getInstance().registerStateMachine(s, 200);
s.init();
s.enter();
s.runCycle();
print(s);

Scanner sc = new Scanner(System.in);
while (true)
    Parse(s, sc.nextLine());
}

public static void Parse(ExampleStateMachine sm, String s) {
//Minden beolvasott sor után írjuk ki az összes változó
//Itt a sor beolvasása után egyből kiírom, mert így értel
//hogy még a parancs értelmezése előtt írjam ki.
print(sm);
switch (s) {
case "start":
    sm.raiseStart();
    sm.runCycle();
    break;
case "white":
    sm.raiseWhite();
    sm.runCycle();
    break;
case "black":
    sm.raiseBlack();
    sm.runCycle();
    break;
case "exit":
    System.exit(0);
}
}

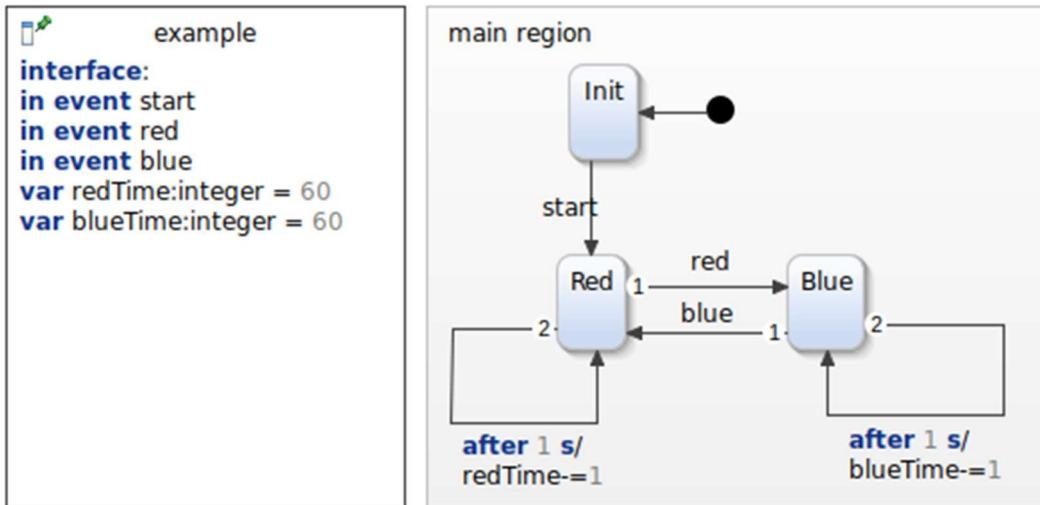
public static void main(String[] args) throws IOException {
ExampleStateMachine s = new ExampleStateMachine();
s.setTimer(new TimerService());
RuntimeService.getInstance().registerStateMachine(s, 200);
s.init();
s.enter();
s.runCycle();
print(s);
Scanner sc = new Scanner(System.in);
while (true) Parse(s, sc.nextLine());
}

public static void Parse(ExampleStateMachine sm, String s) {
print(sm);
switch (s) {
case "start":
    sm.raiseStart();
    sm.runCycle();
    break;
case "white":
    sm.raiseWhite();
    sm.runCycle();
    break;
case "black":
    sm.raiseBlack();
    sm.runCycle();
    break;
case "exit":
    System.exit(0);
}
}

```

A kommentektől és minimális formázástól eltekintve, a generáló program előállította tökéletesen a 3.5-ös feladatban írt kódot.

Ezek után módosítottam az example.sct -t, hogy lássam, hogy általánosságban is működik a programom.



Generált kód:

```
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();
        print(s);
        Scanner sc = new Scanner(System.in);
        while (true) Parse(s, sc.nextLine());
    }

    public static void Parse(ExampleStatemachine sm, String s) {
        print(sm);
        switch (s) {
            case "start":
                sm.raiseStart();
                sm.runCycle();
                break;
            case "red":
                sm.raiseRed();
                sm.runCycle();
                break;
            case "blue":
                sm.raiseBlue();
                sm.runCycle();
                break;
            case "exit":
                System.exit(0);
        }
    }

    public static void print(IExampleStatemachine s) {
        System.out.println("R = " + s.getSCIface().getRedTime());
        System.out.println("B = " + s.getSCIface().getBlueTime());
    }
}
```

A kód az importoktól és a package névtől eltekintve fordítható és futtatható:

```
R = 60
B = 60
blue
R = 60
B = 60
red
R = 60
B = 60
start
R = 60
B = 60
blue
R = 59
B = 60
red
R = 57
B = 60
red
R = 57
B = 56
exit
R = 57
B = 55
```

Felülírtam a 3.5-ös kódját, majd kipróbáltam a programot. Ugyan úgy működött, mint a kézzel írott.

A kipróbálás után visszaállítottam a 3.5-ös kódját a kézzel írottra, így a beadásban is az szerepel, viszont az example.sct -t nem állítottam vissza.