Modellek programozott feldolgozása

Laboratórium ismertető

A munkája értékelésénél a dokumentációt, az elkészített forráskódot vesszük figyelembe.

- Készítsen egy rövid dokumentációt (pár oldalas pdf állomány), amiben dokumentálja a látottakat és a feladat megoldását.
- Git-en verziókezelje megoldását, és a dokumentációban hivatkozza a repositoryt amelyben dolgozott.

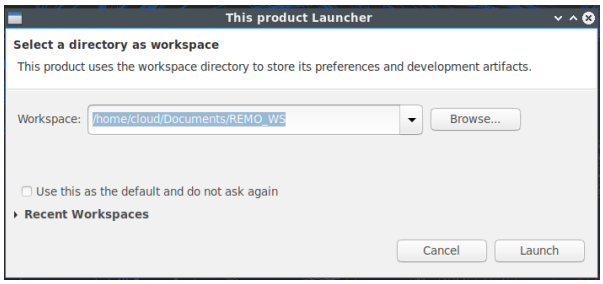
A Labor során a Yakindu eszközt egészítjük ki kódgenerálási funkciókkal.

1. Előkészületek

1.1. Töltse le és indítsa el a virtuális gépet (Felhasználónév/Jelszó: Cloud/LaborImage), vagy nyissa meg a felhőben a virtuális gépet.
(Linkek a feladatkiírásban találhatóak meg)

1.2. Töltse le a kiindulási állapotot (link ugyancsak a feladatkiírásban találhatóak meg).

1.3. Nyissa meg a Yakindu STC alkalmazást. Amikor kérdezi, használjuk az előre beállított workspace-t:



1.4. **Tipp:** a labor során különböző fájlokat fogunk automatikusan létrehozni, amiket a projekt mappájába helyez az alkalmazás. Ahhoz, hogy az újonnan létrehozott állományok automatikusan megjelenjenek a fejlesztőkörnyezetben állítsuk be az alábbi opciót:

Window > Preferences > General > Workspace > Refresh using native hooks or polling > Apply

1.5. Importálja a kiindulási projektet: File > Import > General > Existing Projects into Workspace > Next > Select archive file > Válasszuk ki az állományt > Finish.

1.6. Vizsgáljuk meg a kiindulási projekteket!

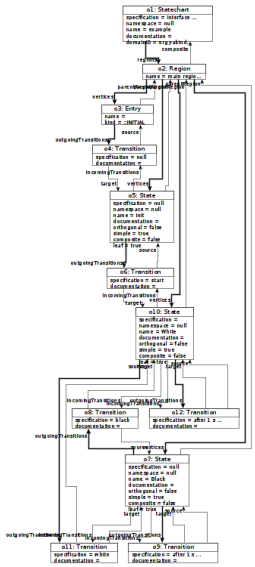
- `hu.bme.mit.model2gml`: Modellek absztrakt szintaxisának megjelenítéséért felelős projekt. Ezt csak használni fogjuk, szerkeszteni nem.
- `hu.bme.mit.yakindu.analysis`: Ebbe a projektbe dolgozzunk! A projekt az alábbi fontos állományokat tartalmazza:
  - `model_input/example.sct`: egy példa állapotgépet tartalmaz. Ezt fogjuk az alkalmazásunk bemeneteként használni.
  - `model_input/example.sgen`: konfigurációs állomány, ami meghatározza, hogy milyen kódot generáljon a Yakindu keretrendszer az állapotgépekből.
  - `yakindu-gen`: ebbe a mappába generál a Yakindu keretrendszer forráskódot. Ezt a kódot használni fogjuk szerkeszteni nem.

- src: ebben a mappában vannak a kézzel írt forráskódok.
  - `ModelManager.java`: segédosztály modellek beolvasására. Csak használni fogjuk, szerkeszteni nem.
  - `Main.java`: olyan java alkalmazás forráskódja, ami betölti egy Yakindu modellt (jelen esetben az `model_input/example.sct` útvonalról), és csinál vele valamit. **Ezt fogjuk szerkeszteni a 2. és 4. feladat során!**
  - `RunStatechart.java`: olyan java alkalmazás forráskódja, ami a Yakindu által generált forráskódot használva csinál valamit. **Ezt fogjuk szerkeszteni a 3. feladat során!**
- Launch fájlok összekészített futtatási konfigurációkat tartalmaznak. A jobb klikk > Run as… / Debug as… opcióval indíthatóak.
  - RunWithSmallModel: Elindítja az alkalmazást a `Main.java` osztállyal, és lehetővé teszi, hogy megtaláljuk a `model_input/example.sct` modellt. **Ezt használjuk a 2. feladat során!**
  - RunStatechart: Elindítja az alkalmazást a `RunStatechart.java` osztállyal, ami a modellből generált kódot használja. **Ezt használjuk a 3. feladat során!**
  - RunWithBigModel: Elindítja az alkalmazást a `Main.java` osztállyal, és lehetővé teszi, hogy megtaláljuk és teljességében lefordítsuk a `model_input/example.sct` modellt. **Ezt használjuk a 4. feladat során!**

2. Modell Bejárása

2.1. Indítsa el a hu.bme.mit.yakindu.analysis/RunWithSmallModel.launch konfigurációval az alkalmazást (jobb klikk > Run as… > Run), tekintse meg a konzol kimenetet!

2.2. Nyissa meg a generált gráfvizualizálást a model_output mappában! A gráf a yEd nevű szerkesztőprogramban nyílik meg, amelyben Layout > Hierarchical > OK paranccsal elrendezheti a gráfot. Hasonló megjelenítést kéne hogy kapjon:



*A kimenetről készítsen **screenshotot** a dokumentációjába!*

2.3. Egészítse ki az alkalmazást, hogy az állapotok mellett a tranzíciókat is kiírja az alábbi formában:

`<Kiinduló állapot neve> -> <Cél állapot neve>`

(Ahol `<Kiinduló állapot neve>` szöveg helyett értelemszerűen a kiindulóállapot nevét írja ki!)

2.4. Egészítse ki az alkalmazást, hogy az keresse meg azokat a csapda állapotokat, amiből nem vezet ki él, és írassa ki a nevüket! A program működését demonstrálja egy példával (rajzolja át a modellt).

2.5. Egészítse ki az alkalmazást, hogy keresse meg azokat az állapotokat, amelyeknek nincs neve, és javasoljon nekik egy alkalmas nevet. A javasolt nevet írjaki a konzolra.

2.6. **Bónusz:** Az alkalmas nevek kiválasztásánál győződjön meg, hogy az adott név nem szerepelt korábban!

*A kimenetről készítsen **screenshotot** a dokumentációjába, és **commitálja** a megoldását!*

3. Yakindu kódgenerátor használata

3.1. Amennyiben módosította az example.stc állományt (például hibákat injekt, állítsa vissza

3.2. Egy összetett kódgenerátor jellemzően több módosítható paraméterrel rendelkezik. Nyissa meg a `hu.bme.mit.yakindu.analysis\model_input\example.sgen` állományt, és adja hozzá az alábbi paramétereket. Amennyiben az egeret az új funkció neve fölé viszi, megjelenik egy dokumentáció. Írja le hogy ezeknek a paramétereknek a bevezetésével milyen változást idézett elő!

```
GeneratorModel for yakindu::java {

	statechart example {

		feature Outlet {
			targetProject = "hu.bme.mit.yakindu.analysis"
			targetFolder = "yakindu-gen"
			libraryTargetFolder = "yakindu-gen"
		}
		feature GeneralFeatures {
			InterfaceObserverSupport = false
			RuntimeService = true
			TimerService = true
		}
	}
}
```

3.3. Tekintse át az alábbi kódrészletet! A kódrészlet megtalálható az alábbi állományban is:

`hu.bme.mit.yakindu.analysis/src/hu/bme/mit/yakindu/analysis/workhere/RunStatechart.java`

A kód az alábbi műveleteket végzi:

- Létrehoz és felparaméterez egy állapotgépet (első három sor).
- Inicializálja az állapotgépet az init és enter metódusokkal.

- Az állapotgépen tetszőleges sorrendben meghívhatjuk eseményeket a raise<Esemény neve>() metódusokkal. Fontos, hogy az események kiváltása után hívjuk meg a s.runCycle() függvényt!
- A s.getSCInterface() interface-en keresztül lekérdezhetjük az állapotgép változóinak értékét. Például s.getSCInterface().getWhiteTime() mutatja a világos játékoshátralévő idejét.

```
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);
}
```

Használja az alábbi importokat:

```
import java.io.IOException;

import hu.bme.mit.yakindu.analysis.RuntimeService;
import hu.bme.mit.yakindu.analysis.TimerService;
import hu.bme.mit.yakindu.analysis.example.ExampleStatemachine;
import hu.bme.mit.yakindu.analysis.example.IExampleStatemachine;
```

3.4. Futtassa az alkalmazást az alábbi konfigurációval: `hu.bme.mit.yakindu.analysis/RunStatechart.launch`

3.5. A kódrészlet alapján készítsen olyan alkalmazást, amely:

- A konzolról beolvas sorokat
  - Amennyiben a beolvasott szöveg megegyezik egy esemény nevével (`start`, `white` vagy `black`), meghívja az adott eseményt az állapotgépen,
  - Amennyiben a beolvasott szöveg az „`exit`”, az alkalmazás leáll.
- Minden beolvasott sor után írjuk ki az összes változó (WhiteTime és BlackTime) értékét!

*A kimenetről készítsen **screenshotot** a dokumentációjába, és **commitálja** a megoldását!*

4. Saját kódgenerátor készítése

4.1. Futtassa a `hu.bme.mit.yakindu.analysis/RunWithBigModel.launch` konfigurációval az alkalmazást, és tekintse át a kirajzolt gráfot! Milyen változást tapasztal?
(Innentől kezdve csak a `RunWithBigModel.launch` konfigurációt használjuk)

4.2. Mutassa meg a kirajzolt gráfban, hogy hogy néz ki a `whiteTime-=1` kifejezés absztrakt szintaxis fája!

*A kimenetről készítsen **screenshotot** a dokumentációjába!*

4.3. A 2. ponthoz hasonlóan járja be a modellt, és írja ki az összes belső változó és bemenő esemény nevét! Az eseményekhez definíciójának megtalálásához sokat segíthet a generált gráfnézetet!

4.4. Írjon egy olyan alkalmazást, ami az események és a változók nevét az alábbi formában írja ki:

```
public static void print(IExampleStatemachine s) {
System.out.println("W = " + s.getSCInterface().get<Első változó neve>());
…
System.out.println("B = " + s.getSCInterface().get<Utolsó változó neve>());
}
```

Azaz írjuk ki a konzolra azt a szöveget, hogy „`public static void`…”, majd amikor a `<Első változó neve>`-hez érünk, ott a változó neve jelenjen meg.

4.5. Vegyük észre, hogy ha egy `.java` kiterjesztésű állományba írnánk, akkor akár forráskódként is értelmezhető a kimenet! A továbbiakban egy olyan kódgenerátor megvalósítása a cél, amely a 3.4-es programozási feladatot automatizálja. Készítsen tehát egy alkalmazást, amely:

- Beolvas egy modellt, és kiolvassa az összes belső változó és bemenő esemény nevét!
- A változók nevei és bemenő események nevei alapján a konzolra kiír egy szöveget, amely java kódként értelmezve az a 3.5-ös feladat általánosítását végzi el:
    - A konzolról beolvas sorokat
        - Amennyiben a beolvasott szöveg megegyezik egy esemény nevével (itt használjuk az események neveit), meghívja az adott eseményt az állapotgépen,
        - Amennyiben a beolvasott szöveg az „`exit`”, az alkalmazás leáll.
    - Minden beolvasott sor után írjuk ki az összes változó értékét (itt használjuk a változók neveit)!
- **Tipp:** Először írjuk ki konzolra a 3.5-es feladatra adott megoldásunk forráskódját, majd fokozatosan általánosítsunk!

4.6. Mutassa meg, hogy a konzolra kiírt kód megegyezik a 3.5-es feladatra készített megoldással! (Elég csak a kód lényegi részére koncentrálni, az `import` szekciótól és egyéb lényegtelen részletektől most eltekinthetünk.)

4.7. Módosítsa az `example.sct` állományban lévő állapotgépet (események és változók megváltoztatásával), és vizsgálja meg, hogy változik-e a kimenet (a generált kódban más nevű függvények szerepelnek)!

4.8. **Bónusz:** Mutassa meg, hogy az ön által generált kód lefordítható!

*A kimenetről készítsen **screenshotot** a dokumentációjába, és **commitálja** a megoldását!*