Tartalomjegyzék

1. fejezet

Bevezetés

A BPEL (*Business Process Execution Language*) nyelv létrejötte elsődlegesen a Web szolgáltatások területéhez kapcsolódik, de a nyelv mint általános munkafolyamat (*workflow*) leíró nyelv, más alkalmazási témakörökhöz is köthető. A BPEL szerepét, fontosságát jól mutatja az a tény is, hogy igen gazdag irodalom található az egyes alkalmazási területekről és speciális szabvány kiegészítésekről.

A BPEL aktualitását jelzi, hogy a megvalósító motorok köre is folyamatosan bővül. Ugyan már lassan 15 év eltelt a szabvány bevezetése óta, a meglévő nagyobb rendszerek (Oracle BPEL Process Manager, IBM WebSphere Process Server, Microsoft BizTalk Server, SAP SAP Exchange Infrastructure) alternatívájaként most is jelennek meg új végrehajtó motor implementációk. A Wikipédia forrása szerint [1] a közelmúltban az alábbi szabad szotver implementációk születtek (1.1. táblázat).

1.1. táblázat. A BPEL nyelv szabad szoftveres implementációi.

Termék neve	**Fejlesztő**	**Megjelenés éve**	**Licensz**
JBPM	JBoss	2016	Apache
Apache ODE	ASF	2016	Apache
Activiti	Alfresco	2014	Apache

Annak ellenére, hogy napjainkra már több BPEL motor elérhető és használatos, a BPEL szerkesztők és különösen a BPEL validációs rendszerek köre igen szegényes. Ezen tapasztalatokból kiindulva a dolgozat célja egy olyan BPEL validációs rendszer elkészítése, amely a BPEL rendszerek egyik fontos tulajdonságát, a terhelés korlátosságát (*bounded model*) vizsgálja. A korlátosság azt jelzi, hogy minden csomópontban van egy felső korlát a végrehajtható feladatok számára, intenzitására vonatkozóan. Ha a rendszer nem teljesíti ezt a kritériumot, akkor túlcsordul valamely megmunkáló/tároló helyen. Az elemzés során a korlátosság ténye mellett, a korlát értékei is fontos vizsgálandó jellemzők.

A meglévő tervezői rendszerekben legtöbbször szimulációval történik a főbb paraméterek, a korlátosság vizsgálata. Ezen megközelítésnek rendszerint két problémája van: a vizsgálat teljessége (azaz valóban minden lehetséges esetet áttekintettük-e) illetve a végrehajtási idő (a szimulációk futtatása hosszabb időt is igénybe vehet).

A dolgozatban a BPEL folyamatok Petri-háló alapú vizsgálatát végzem el. A Petri-háló alapú reprezentáció egy elfogadott és többek által alkalmazott megközelítés. A kidolgozott rendszer inputként egy BPEL modell leírását várja és kimenetként az elemzés eredményét illetve a folyamatok nyomkövetését adja vissza.

2. fejezet

A BPEL és folyamatainak bemutatása

A BPEL szabvány alapjai a 2000-es évek közepén jöttek létre az üzleti folyamatok szabványos leírására, elsősorban a Web szolgáltatás (*Web Service*, a továbbiakban röviden WS) alapú környezetre fókuszálva [2]. A nyelv célja definiálni az egyes WS folyamatok vezérlését, koordinálását a kívánt üzleti logika megvalósítására. A BPEL modell magja a folyamatok leírására szolgál, melyekhez együttműködő partnereket szimbolizáló modulok kapcsolódhatnak. A folyamaton belüli lépéseket, végrehajtási algoritmust az aktivitási elemekkel írhatjuk le. A folyamatokhoz változók is rendelhetőek, melyek a kapcsolódó adatkezelést reprezentálják. Az egyes processz modulok üzenetváltással kommunikálhatnak egymással.

A BPEL nyelv a WS környezetből adódóan szorosan kötődik az XML alapú adattároláshoz. A BPEL modell XM állományként áll elő, melyben az egyes séma megkötéseket az XMLSchema szabvány biztosítja. Az XMLSchema nyelv lehetővé teszi az XML dokumentumok strukturális és tartalmi ellenőrzését. A sémanyelv gazdag integritási elem készlettel rendelkezik és támogatja a típusok származtatását is. Az elemi adatkezelő műveletek, szabályok és kifejezések megadásánál az XPath szabványt kell alkalmazni.

A BPEL nyelv tulajdonságaival és alkalmazási lehetőségeivel számos kutatás foglalkozott már az elmúlt évtizedben. A BPEL nyelv nagy előnye a deklaratív formalizmus, mellyel nem szükséges az egyes modulok kódjába beleégetni az üzleti szabályokat. Az elosztott környezet esetén fontos, hogy az egyes üzleti szabályok illeszthetőek, integrálhatóak és validálhatóak legyenek [14]. A BPEL modell fejlesztése ezen célkitűzések teljesítésére irányult. Több dolgozatban (például [12]) a BPEL nyelv, mint cél modellezési nyelv jelenik meg, s más szabvány folyamat modellezési nyelv, mint UML, BPEL-re történő konvertálását vizsgálják.

A létrehozott BPEL modellnyelv tulajdonságait és alkalmazhatóságát aktívan vizsgálták a 2000-es évek második felében. Ennek keretében a [3] dolgozat az elkészített BPEL modellek dinamikus nyomkövetésére, monitorizására mutat be hatékony algoritmusokat. A modell főbb elágazási és szinkronizációs elemeihez kapcsolódó elemzéseket a [13] dolgozat foglalja össze. A BPEL üzleti folyamatok magasabb szintű, a folyamat egységteszt alapú elemzésére ad javaslatot a [10].

A BPEL nyelv egyik gyakori alkalmazási területe a workflow rendszerek fejlesztése. Több irányban is történtek lépések, hogy a BPEL munkafolyamat modelljét különböző alkalmazási területeken használják. Ezen kísérletek között megemlíthetők a grid és tudományos workflow folyamatok [15] és a mobil alkalmazások fejlesztése [6]. A témakörhöz kapcsolódóan megemlíthető, hogy sokan a BPEL-t tekintették az univerzális workflow leíró nyelvnek, s javaslatok is születtek a BPEL központú workflow szemléletre [16]. A hazai vonatkozású fejlesztések körében kiemelhető a BPEL rendszerek formális validációjára irányuló vizsgálatok köre [8]. A validációs elemzések mellett a modell komplexitás meghatározására is találunk példákat a nemzetközi irodalomban [5].

BPEL nyelv kiterjesztésére is több dolgoztat született, mint például az elosztott rendszerekre történő kiterjesztés [4]. A kiterjesztések körét a [7] dolgozat foglalja össze. Ennek az egyes kritériumait és karakterisztikáját a 2.1. táblázat foglalja össze.

2.1. táblázat. A BPEL kiterjesztéseinek összefoglaló táblázata.

Terület	**Paraméterek**
Kritériumok	Lehetőség a kiszervezésre Rugalmasság Funkcionalitás Fenntarthatóság Teljesítmény Újrafelhasználhatóság Robusztusság Használhatóság
Felhasználás	Vezérlési folyam Adatintegráció Kifejezések és hozzárendelő utasítások Nagy mennyiségű adat kezelése Egyéb Szolgáltatás társítása Szolgáltatás meghívása Változó hozzáférés
Munkafolyam dimenziója	IT infrastruktúra Processz logika Szervezés
Helye a BPM életciklusban	Modellezés IT finomítás Statikus analízis, ellenőrzés Deployment Munkavégzés Megfigyelés

3. fejezet

Petri-hálók és alkalmazásaik

3.1. A Petri-hálók matematikai modellje

A Petri-háló egy matematikai leírómodell elosztott rendszerek bemutatására. A modellt Carl Adam Petri készítette. A modell nagyon hasonlít a programozók körében elterjedt folyamat ábrára. A háló irányított élekből, helyekből és átmenetekből (*mint elemek*) áll. Az élek csak két különböző típusú elem között állhatnak. A helyeken jelölő objektumok, úgynevezett tokenek állhatnak. A tokenek csak diszkrét számban fordulhatnak elő egy helyen, és a token átvitele atomi folyamat, azaz nem félbeszakítható. A tokenek elláthatóak attribútummal is, ilyen esetben a tokeneket "kiszínezzük" és színezett petri hálóról beszélünk.

A Petri háló alapvető matematikai modellje egy páros, irányított és súlyozott multigráf $PN(P, T, A, W, S)$, ahol

- $P = \{p_1, p_2, \ldots, p_N\}$: a helyek véges halmaza,
- $T = \{t_1, t_2, \ldots, t_M\}$: egy véges tranzició halmaz,
- $P \cap T = \emptyset$,
- $A \subseteq P \times T \cup T \times P$: az élek halmaza,
- $W : F \Rightarrow N^+$: az élsúlyok halmaza,
- $S : P \Rightarrow N^+$: a kezdőállapot.

3.2. Színezett Petri-hálók

Az elemi színezett háló felírható, mint egy

$$CPN(P, T, A, \Sigma, V, C, G, E, S)$$

struktúra, ahol

- $P = \{p_1, p_2, \ldots, p_N\}$: a helyek véges halmaza,
- $T = \{t_1, t_2, \ldots, t_M\}$: egy véges tranzició halmaz,
- $A \subseteq P \times T \cup T \times P$: az élek halmaza,
- Σ: a színek halmazainak halmaza,
- V: a változók halmaza, ahol $\forall v \in V$: változóhoz egy $Type[v] \in \Sigma$ típus rendelhető,

- $C : P \to \Sigma$: a helyekhez színeket rendelő függvény,
- $G : T \to EXPR_V$: az egyes tranzíciókhoz kapcsolódó validációs, ellenőrzési kifejezés (logikai értékű),
- $E : A \to EXPR_V$: az egyes élekhez kapcsolódó kifejezés, amely a kapcsolódó hely színhalmazához tartozó értéket vehet fel,
- $S : P \Rightarrow N^+$: a kezdőállapot.

Adott $CPN(P, T, A, \Sigma, V, C, G, E, S)$ színezett hálóhoz az alábbi kezelő funkciók köthetőek:

- $M(p)$: a jelölő (*marker*) függvény, melynek értéke a p helyhez kapcsolódó tokenek halmaza (színezett Petri háló esetén az $M(p)$ elemek színeinek illeszkedni kell a $C(p)$ színhalmazhoz),
- $M_0(p)$: helyek induló tokenkészlete,
- $Var(t)$: a tranzíciók viselkedését leíró változók halmaza,
- $b(v)$: az adott v változó értékét megadó kifejezés, ahol $b(v) \in Type[v]$.

Egy t tranzíció esetén a $Var(t)$ kifejezés a tranzícióhoz rendelt változók együttese, ahol a változók a $G(t)$ vagy E(a: t-hez kötődő él) kifejezésekben szerepelnek. Az egyes esetekhez tartozó halmazok tehát az alábbiak.

$$Var(t) = \begin{cases} \{n, d\}, & \text{ha } t = SendPacket, \\ \{n, d, success\}, & \text{ha } t = TransmitPacket, \\ \{n, d, k, data\}, & \text{ha } t = ReceivePacket, \\ \{n, success\}, & \text{ha } t = TrancmitAck, \\ \{n, k\}, & \text{ha } t = ReceiveAck. \end{cases}$$

A hálóban egy tranzíció akkor engedélyezett (*ready*), ha a bemenő helyeknél a kívánt tokenszám megtalálható. Jelölt hálók esetében:

$$M'(p) = M(p) - I(p, t) + O(p, t) : \forall p \in P,$$

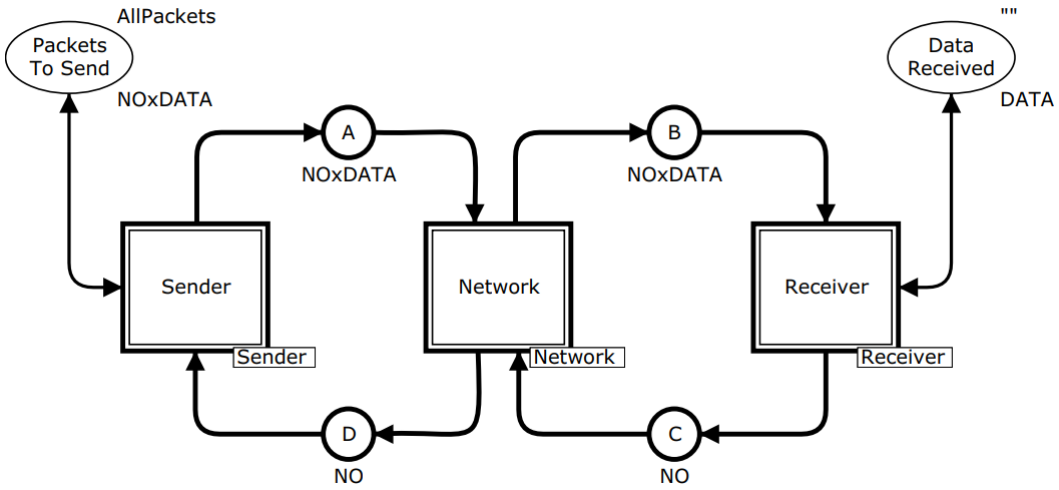
ahol

- $I : F \Rightarrow N^+$: bejövő áram, intenzitás
- $O : F \Rightarrow N^+$: kimenő áram. intenzitás

A hierarchikus CPN rendszerben az átláthatóság növelése érdekében összefogó modulokat is lehet alkalmazni. Egy modul más elemi egységek együttese, tárolója (3.1. ábra).

A moduloknál fontos szerepet kapnak az átadó helyek, melyeken keresztül a tokenek bejöhetnek a modulba illetve kiléphetnek a modulból. Az ilyen port jellegű helyek lehetnek bemeneti portok (IN) illetve kimeneti portok (OUT).

A CPN rendszerek egyik hasznos tulajdonsága, hogy lehetőséget adnak a felépített modell formális ellenőrzésére, validálására és értékelésére. A formális ellenőrzés egyik leggyakoribb eszköze az állapottér (*state space*) modell, ahol az állapottér egy olyan irányított gráf, melyben a csomópontok a háló egy lehetséges $M(CPN)$ jelölési állapota. Azaz a háló struktúrája rögzített, de az egyes elemeknél a tokenek és változók halmaza, azok állapota változhat. A véges állapottér modellt rendszerint szimulációkal állítják elő.



3.1. ábra. Rendszerséma 3 modullal

Az állapottér modellből kiindulva további elemzésekre ad lehetőséget a komponens gráf modell (*Strongly Connected Component Graph*, röviden SCC) formalizmus. Az SCC gráfból a rendszer általános viselkedési szabályaira lehet következtetni. Az SCC gráf olyan gráf, melynek csomópontjai az állapottér azon diszjunkt részhalmazai, ahol egy részhalmaz bármely két elemére igaz, hogy az egyik elem elérhető a másikból.

Az elemzések során az alábbi főbb tulajdonságok elemzésére szokás kitérni:

- Reachability Properties,
- Boundedness Properties,
- Home Properties,
- Liveness Properties,
- Fairness Properties.

4. fejezet

Az üzleti folyamatok elemeinek leképzése

4.1. A leképzés menete

A BPEL nyelv az egymással együttműködő modulok működését szimulálja. A BPEL modellben a folyamat lépéseit tevékenységnek *activity* nevezzük. A tevékenységek lehetnek elemiek és összetettek. Tipikus elemi tevékenységek:

- más funkció meghívása,
- üzenet küldés/fogadás,
- változók módosítása,
- várakozás, szinkronizálás.

Az elemi tevékenységekből összetett folyamatok építhetőek fel szekvencia, párhuzamosítás és ciklus elemek segítségével.

A BPEL szabvány tevékenység készletének bemutatásánál csak a legfontosabb elemekre térünk most ki. A szabvány részletei a http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html weboldalról érhetőek el.

A BPEL definíció egyértelműsíti a folyamat kezdetét és inicializálja is a megfelelő paraméterekkel. A Petri háló erre nem tér ki, ezért bevezetünk egy opcionális *START* tranazíciót. Ennek feladata, a megfelelő tokenek legenerálása a folyamat indításához. A folyamat zárása szintén egyértelmű a BPEL szabványban. Petri háló kapcsán beszélhetünk a háló leállásáról, ha már semmilyen hely és tranzíció nem produkál új tokent, nem várakozik és nem nyel el tokent. A köztes elemek leképzése általánosan nem bonyolult, de megvizsgálhatók alternatív leképzési módok.

Az alkalmazás jelölésrendszere: A rendszer a helyeket P-vel, míg a tranzíciókat T-vel indexeli. A helyek után listázza az adott helyen levő tokeneket. A jelenleg lépésben levő tranzíciók pedig színes kitöltést kapnak.

4.2. `<receive>`

Egy megfelelő üzenet után engedi a folyamatot továbbhaladni, így várakoztatáshoz használható. A `<receive>` típusú XML elemekhez tartozó sémaleírás:

```
<receive partnerLink="NCName"
   portType="QName"?
   operation="NCName"
```

```
   variable="BPELVariableName"?
   messageExchange="NCName"?>
   <fromParts>?
      <fromPart part="NCName" toVariable="BPELVariableNm"/>+
   </fromParts>
</receive>
```

A hálóban ezt egy tranzicióval könnyedén megoldhatjuk, hiszen csak egy specifikus üzenet token kell a továbblépéshez, és a többit addig az előző helyen parkoltatja.

4.3. <reply>

Üzenetküldő elem, ami <receive>; <onMessage>;<onEvent> események után léphet akcióba.

```
<reply partnerLink="NCName"
   portType="QName"?
   operation="NCName"
   variable="BPELVariableName"?
   messageExchange="NCName"?>
   <toParts>?
      <toPart part="NCName" fromVariable="BPELVariableNm"/>+
   </toParts>
</reply>
```

A lekezelése az előző példával analóg módon, annyi különbséggel, hogy a tokenek nem parkolnak, hanem tovább mennek és a tranzíció csak akkor generál új tokent ha üzenetet kap egy ágról.

4.4. <invoke>

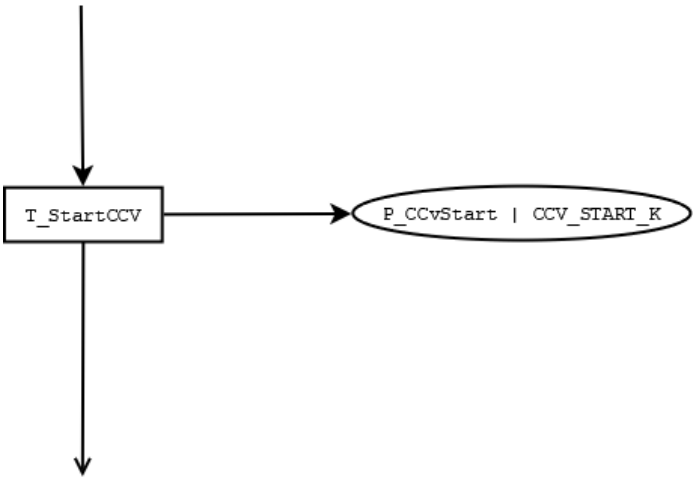
Egy BPEL vagy épp egy webszolgáltatás meghívására szolgál és definiálja a szolgáltatás feladatát is.

```
<invoke partnerLink="NCName"
   portType="QName"?
   operation="NCName"
   inputVariable="BPELVariableName"?
   outputVariable="BPELVariableName"?>
   <catch faultName="QName"? ... >*
   activity
   </catch>
   <toParts>?
      <toPart part="NCName" fromVariable="BPELVariableNm"/>+
   </toParts>
   <fromParts>?
      <fromPart part="NCName" toVariable="BPELVariableNm"/>+
   </fromParts>
</invoke>
```

A grafikus megjelenítése a 4.1. ábrán látható.

Használatát a programrészek újrafelhasználhatósága indokolja, valamint az átláthatósági alapelvek. Például, az ábrán látható CCvalidation használható ATM-es pénz felvét, egyenleglekérdezés, vagy egyéb ATM nél végezhető művelet során.

4.1. ábra. Az `invoke` grafikus jelölése az Oracle BPEL designer-ben



4.2. ábra. Az `<invoke>` leképzése petri hálóra

Leképzés során ügyelni kell arra, hogy az `invoke` paramétereinek megfelelő tokenek keletkezzenek és legyenek átadva a részháló start elemének.

4.5. <assign>

Egy változó értékadására szolgáló esemény. Ellentétben egy imperatív értékadással egy `<assign>` blokkban bármennyi értékadás, másolás történhet, amíg azt a kliens kezelni tudja, így logikailag egy egységbe zárja a műveleteket.

```
<assign validate="yes|no"? standard-attributes>
   (
   <copy keepSrcElementName="yes|no"?
   from-spec
   to-spec
   </copy>
</assign>
```

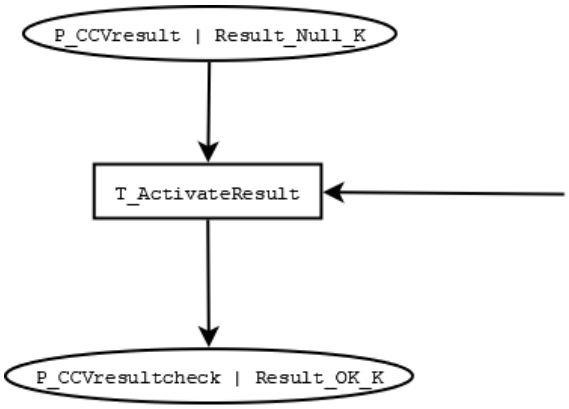
Az érték hozzárendelése nagyon egszerűen átírható egy tranzicióra ami a megfelelő tokenek színét módosítja. A grafikus megjelenítése az 5.1. ábrán látható. A színmódosítás egyszerűen a token nevének átírását jelenti.

Az ábrán egy változós értékadásra látható példa. Hasonló módon kezelendő a blokkosított értékadás, a különbség, hogy a több token jöhet és távozhat több forrásba is.

4.6. <validate>

Egy sémára validálja az XML (BPEL) állományt.

4.3. ábra. Az `assign` grafikus jelölése az Oracle BPEL designer-ben



4.4. ábra. Az `assign` leképzése petri hálóra

```
<validate variables="BPELVariableNames" standard-attributes>
   standard-elements
</validate>
```

Mivel a Petri háló nem tartalmaz validációs elemeket, ezért a `<validate>` nem képződik le.

4.7. <throw>

Egy rész processzen belül fault generálására szolgál.

```
<throw faultName="QName"
   faultVariable="BPELVariableName"?
   standard-attributes>
   standard-elements
</throw>
```

Nagyon egyszerűen egy *fault* tokent generáló tranzició komponens. Explicit hálórésze nincs, hanem a megfelelő inputtokenek megléte vagy hiánya generálja egy tranzició során.

4.8. <wait>

Időre vonatkoztatva várakoztat. Például 5000 tick vagy 14:00:23 (hh:mm:ss)

```
<wait standard-attributes>
   standard-elements
```

```
   (
   <for expressionLanguage="anyURI"?>duration-expr</for>
   |
   <until expressionLanguage="anyURI"?>deadline-expr</until>
   )
</wait>
```

Megadható egy részhálóval ami valójában egy oszcillátor és a megfelelő iteráció után folytat tokent küld. Továbbá létezik úgynevezett időérzékeny háló, ahol az elemek tokenátadási sebessége ismert, vagy állítható. Ezzel időzítőt lehet létrehozni. Esetenként hardware-es óra is használható

4.9. <empty>

No-op (*no operations*) esemény szinkronizációra szolgál.

```
<empty standard-attributes>
   standard-elements
</empty>
```

Beiktatható egy semleges tranzició és hely.

4.10. <sequence>

Sorozatot ad meg.

```
<sequence standard-attributes>
   standard-elements
   activity+
</sequence>
```

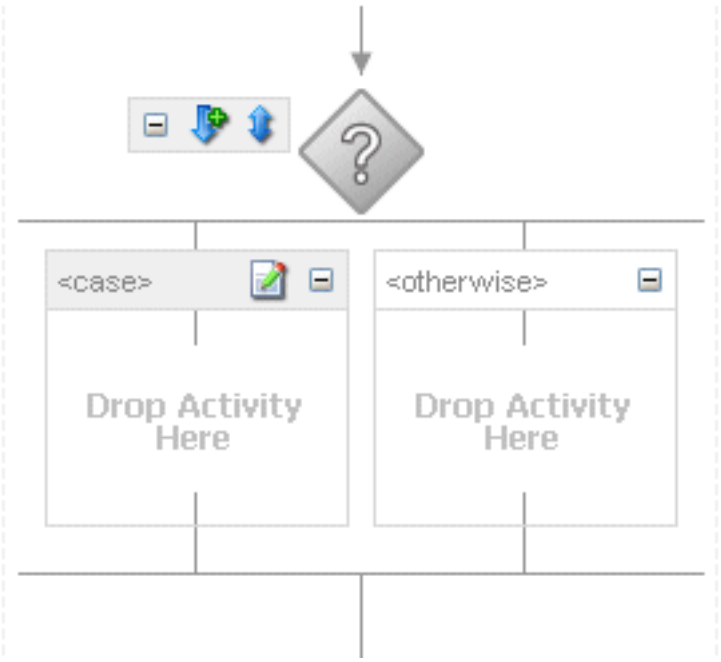
Egyszerűen csak tranziciók és helyek összefűzése.

4.11. <if>

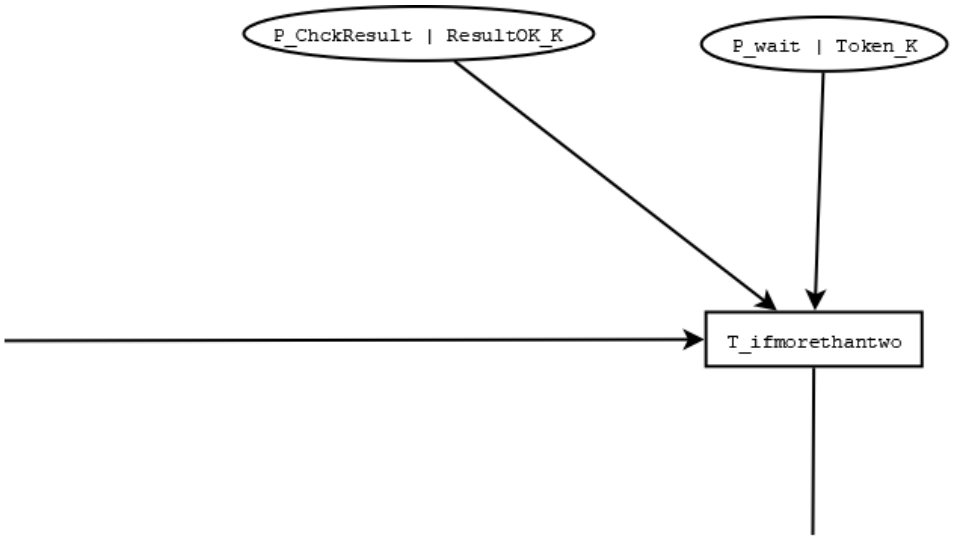
<if> Standard kétirányú elágazás. Logikai XPATH kifejezést vár.

```
<if standard-attributes>
   standard-elements
   <condition >bool-expr</condition>
   activity
   <elseif>*
      <condition>bool-expr</condition>
   activity
   </elseif>
   <else>?
   activity
   </else>
</if>
```

Egy tranzició, mely tokenek függvényében más felé küldi tovább, vagy generál tokeneket. Ez lehet egy helyen összegyúlt tokenek mennyisége, vagy egy adott helyen egy specifikus színű token megléte, vagy nemléte. Analóg módon egy *Switch-Case* elágazás is definiálható vele. A PBEL-es grafikus megjelenítése a 4.5. ábrán látható.



4.5. ábra. Az `if` grafikus jelölése az Oracle BPEL designer-ben



4.6. ábra. Az `if` Egy hálóképzése

4.12. <while>

Elöltesztelős ciklus. Addig iterál, amíg az ciklus feltétel igaznak értékelődik ki.

```
<while standard-attributes>
   <condition>bool-expr</condition>
   activity
</while>
```

Egy tranzició, mely token függvényében a folyamat egy korábbi pontjára csatol vissza, vagy éppen egy későbbire, a feltétel hamis logikai állapota esetén. A feltétel persze egy színes token jelenléte, vagy tokenek száma is lehet.

4.13. <repeatUntil>

Egy hátultesztelős ciklusnak feleltethető meg, amely akkor enged tovább, ha a feltétel igaz.

```
<repeatUntil standard-attributes>
   standard-elements
   activity
   <condition expressionLanguage="anyURI"?>bool-expr</condition>
</repeatUntil>
```

A <while>-al analóg módon megadható a Petri-hálós leképzése.

4.14. <forEach>

Végig iterál a gyerekelemeken. Megadható párhuzamos feldolgozás is. Egy *Complete condition* segítségével megadható egy break utasítás ami kilép a ciklusból.

```
<forEach counterName="BPELVariableName" parallel="yes|no">
   <startCounterValue expressionLanguage="anyURI"?>
   unsigned-integer-expression
   </startCounterValue>
   <finalCounterValue expressionLanguage="anyURI"?>
   unsigned-integer-expression
   </finalCounterValue>
</forEach>
```

Egyszerű loop utasítás, azonban párhuzamosítás esetén a részhálóból megfelelő példányszámot generáltatunk.

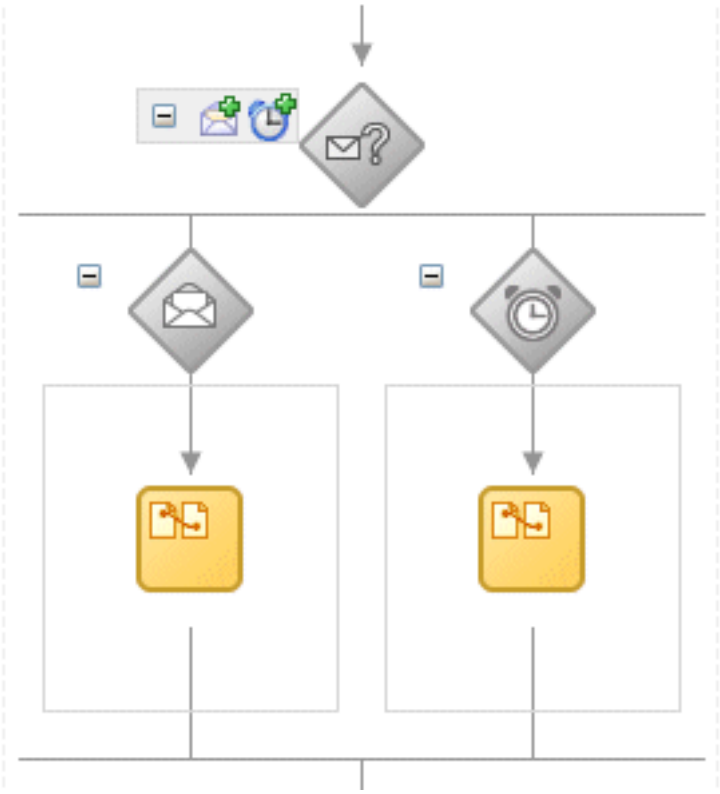
4.15. <pick>

Üzenetek várására vagy időtúllépés eseményre figyel. Ezek bármelyike a szubprocessz végrehajtásához vezet.

```
<pick createInstance="yes|no"? standard-attributes>
   standard-elements
   <onMessage partnerLink="NCName"
      portType="QName"?
      operation="NCName"
      variable="BPELVariableName"?
      messageExchange="NCName"?>+
      <correlations>?
         <correlation set="NCName" initiate="yes|join|no"? />+
      </correlations>
      <fromParts>?
         <fromPart part="NCName" toVariable="BPELVariableName" />+
      </fromParts>
      activity
   </onMessage>
   <onAlarm>*
      (
```

```
      <for expressionLanguage="anyURI"?>duration-expr</for>
      |
      <until expressionLanguage="anyURI"?>deadline-expr</until>
      )
      activity
   </onAlarm>
</pick>
```

A grafikus megjelenítése a 4.7. ábrán látható.



4.7. ábra. A `pick` grafikus jelölése az Oracle BPEL designer-ben

Összetartó hálóval és egy tranzícióval képezhető le.

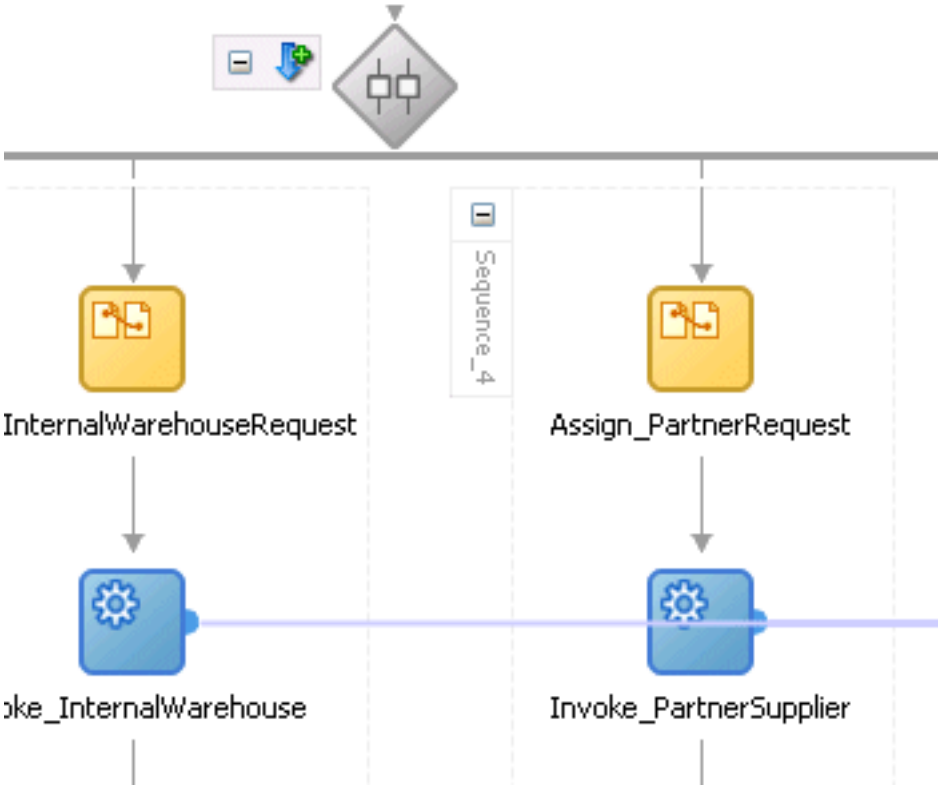
4.16. `<flow>`

Konkurens elemek deklarálására szolgál. Linkek segítségével megadható függőségi viszony a gyerekek között.

```
<flow standard-attributes>
   standard-elements
   <links>?
      <link name="NCName" />+
   </links>
   activity+
</flow>
```

A grafikus megjelenítése a 6.1. ábrán látható.

Leképzése egyszerű, hisz a megfelelő gyerek elemek nem szekvenciálisan helyezkednek el, hanem párhuzamosan.



4.8. ábra. A `flow` grafikus jelölése az Oracle BPEL designer-ben

4.17. <scope>

A gyerek elemek hatókörét lehet vele szabályozni.

```
<scope isolated="yes|no"? exitOnStandardFault="yes|no"?
   standard-attributes>
   standard-elements
   <partnerLinks>?
      ... see above under <process> for syntax ...
   </partnerLinks>
   <messageExchanges>?
      ... see above under <process> for syntax ...
   </messageExchanges>
   <variables>?
      ... see above under <process> for syntax ...
   </variables>
   <correlationSets>?
      ... see above under <process> for syntax ...
   </correlationSets>
   <faultHandlers>?
      ... see above under <process> for syntax ...
   </faultHandlers>
   <compensationHandler>?
      ...
   </compensationHandler>
   <terminationHandler>?
      ...
```

```
   </terminationHandler>
   <eventHandlers>?
      ... see above under <process> for syntax ...
   </eventHandlers>
   activity
</scope>
```

Nem generál új elemet, csak a láthatósági, azaz visszacsatolási elemeket adja meg.

5. fejezet

Szimulációs és validációs keretrendszer

5.1. Elvárások az alkalmazással szemben

Az alkalmazás legfőbb feladata egy konverzió elvégzése BPEL és Petri-háló modellek között. Ebből adódóan minden BPEL elemet le kell tudnia kezelni, illetve az azok közti összefüggéseket feltérképezi, és az összefüggés halmazból egy Petri-hálót előállítani. A rendszer a hálót felépíti, valamint a hálón belüli mozgásokat rajzolja, és szükségesség esetén input file-t vagy felhasználói inputot kezel. Ha a háló nem hozható létre, akkor azt tudatnia kell, és lehetőség szerint rövid indoklással alátámasztania.

5.2. Az alkalmazás felépítése

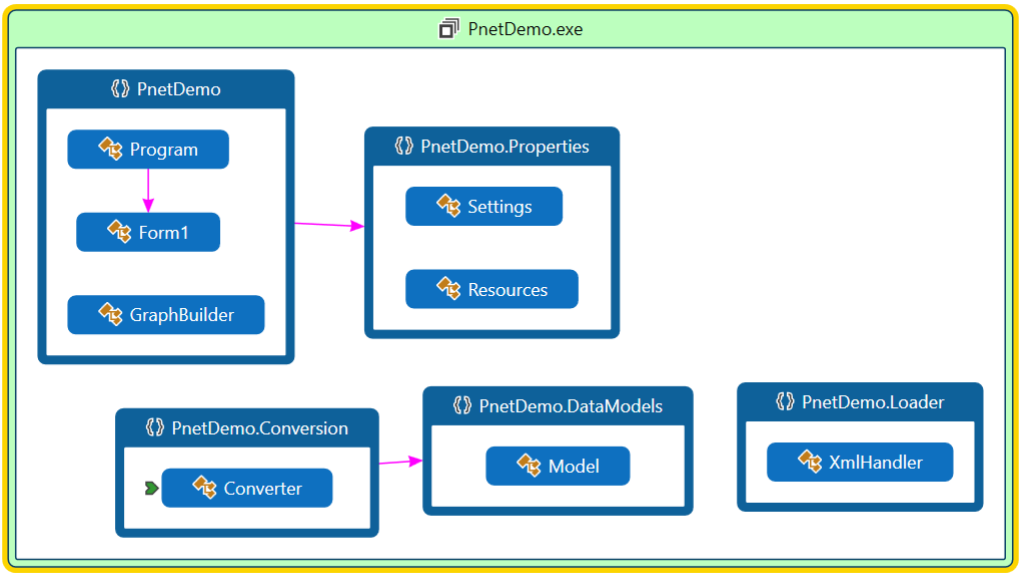
Az alkalmazás logikailag a következő fő részekből áll:

- I/O module: Beolvassa az XML dokumentumot és értelmezi. Szükség esetén menti a kész hálót.
- Conversion module: Átkonvertálja a beolvasott dokumentumot.
- Data Structure module: A saját típusú Petri elemeket kezeli, és adatszerkezeti implementációt tartalmaz.
- UI talker: A UI ra illeszti a megfelelő input mezőt, az abba felvitt értéket átadja a feldolgozó egységnek.
- Graphics module: Az MSGL libraryre épül. Feladata a gráf rajzolása és megjelenítése animációval együtt.
- Computing module: A háló animációjához végzi a szükséges számításokat és időzítéseket.

A programmodulok pedig a következők:

- Form: Az alkalmazás layoutját tartalmazza, illetve a rajta levő elemekhez rendelt kódot. (pl, gombok függvényhívását, töltődési inicializációt, stb.)
- GraphBuilder: A petri hálót (mint gráfot) generálja. Saját leíró nyelvet használ, a Modelben levő hálóz veszi alapul.
- Model: Tartalmazza a Petri háló és elemeinek leírását, valamint az MASGL saját node alapú architektúra definíciót, a könnyebb generálás érdekében.

- Resources: A projekt által használt "erőforrásokat" ide tartoznak a tesztelési adatstruktúrák, bemenetek, illetve a különböző placeholder képek.
- XmlHandler: Az BPEL (xml) fileok beolvasását és parse-olását ahjtja végre. Kimenete egy feltérképezett XML struktúra, amiben majdnem azonnal lehet különböző node-okra ugrani.
- Converter: A különböző adatstruktúrák közötti konverziós logikát tartalmazza, a segéd adatstruktúrákkal.



5.1. ábra. Az alkalmazás osztálydiagrammja

6. fejezet

Az alkalmazás implementációja

6.1. C# implementáció

Az alkalmazás implementációja során fontos volt a C# alapelveinek betartása (OOP elvek és nyelv specifikus elvek együttvéve). Szerencsére nem kellett újra feltalálni semmit, hisz a .NET rendelkezik gráf rajzolóval, megjelenítővel, (hozzá különféle szolgáltatásokkal) és különféle standard formátumú fileokra készített feldolgozó modulokkal. Ebből adódóan a lényeges munka az átalakító modulok és az adatstruktúrák megírása volt.

```
internal class Model
    {
        public class Net
        {
            public List<Place> Places { get; set; }
            public List<Transition> Transitions { get; set; }
            public List<Arc> Arcs { get; set; }

            public Net()
            {
                Places = new List<Place>();
                Transitions = new List<Transition>();
                Arcs = new List<Arc>();
            }
        }

        public class Node
        {
            public string Label { get; set; }
        }

        public class Place : Node
        {
            public List<Token> Tokens { get; set; }
        }

        public class Transition : Node
        {

        }
```

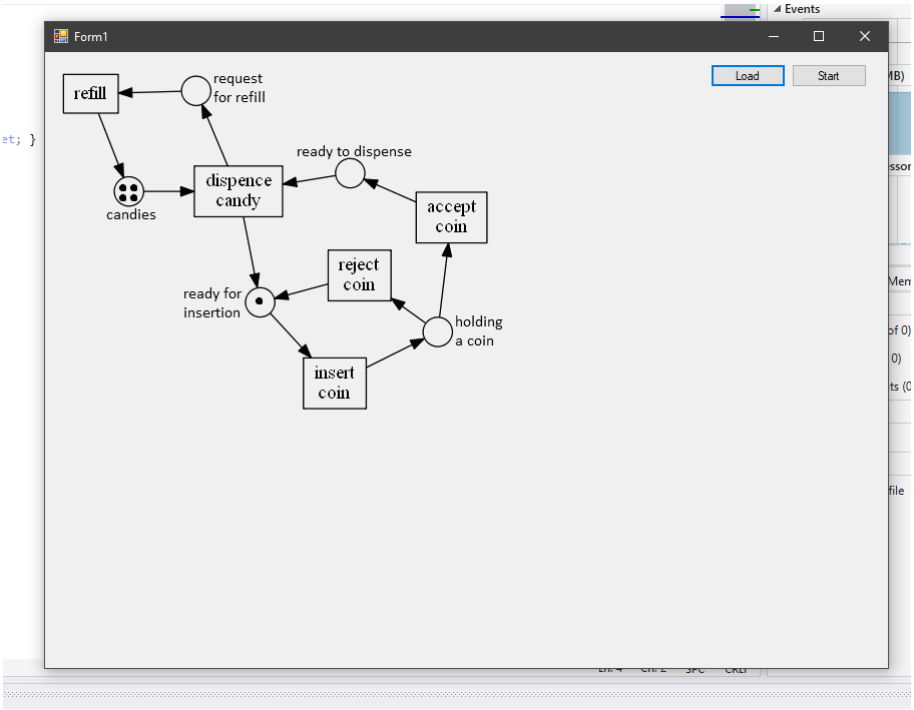
```
        public class Arc
        {
            public Node Input { get; private set; }
            public Node Output { get; private set; }
            public int Multiplicity { get; set; }
        }

        public class Token
        {
            public string Color { get; set; }
        }
    }
```

A Színezett Petri-háló adatstruktúrája

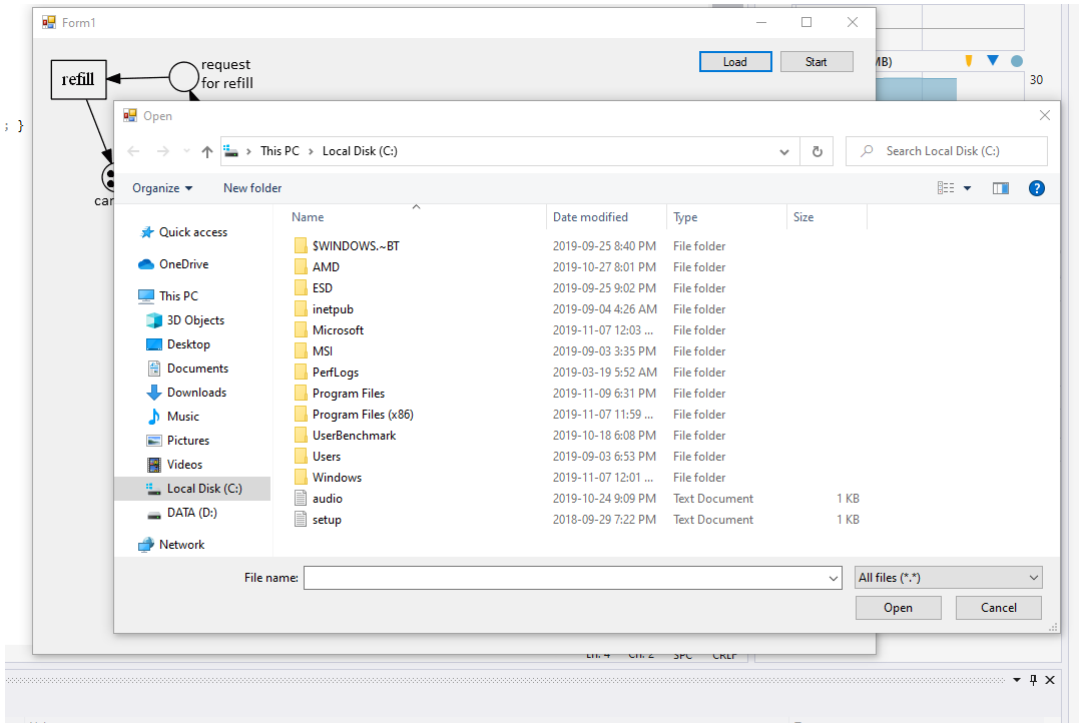
A step-by-step animáció úgy készül, hogy a megfelelő lépések legenerálódnak. A képek bekerülnek egy pipe-ba, ami a program beállításaiban megadott időzítő ütemére átadásra kerülnek megjelenítésre. A lépések addig számítódnak amíg a folyamat teljes egészében lefut, vagy egy olyan nodehoz nem ér ami felhasználói bevitelt vár. Ekkor a számítás szünetel és a UI megjelenítő előkészít egy input mezőt. Az input mező a felhasználói adatbevitel után (ha nem szükséges a következő lépéshez) eltűnik, az átláthatóság kedvéért. A mező tartalma átkerül feldolgozásra és általában token formájában kerül megjelenítésre. (Előfordulhat, hogy a program egy várakozási időt kér. Ez esetben is megjelenítődik token formájában, de a lényeg, a háló tranzíciói csak egy bizonyos idő (feldolgozási-ciklusidő) függvényében tüzelnek. Az alkalmazásban nem volt szempont a konvertált hálók mentése, de könnyűszerrel megoldható.

Az alkalmazás megnyitásakor egy helyfoglaló kép, és az alkalmazás kezdeti állapotú vezérlőfelülete fogad. A vezérlőfelület két gombot tartalmaz alapállásban. Egy load és egy start



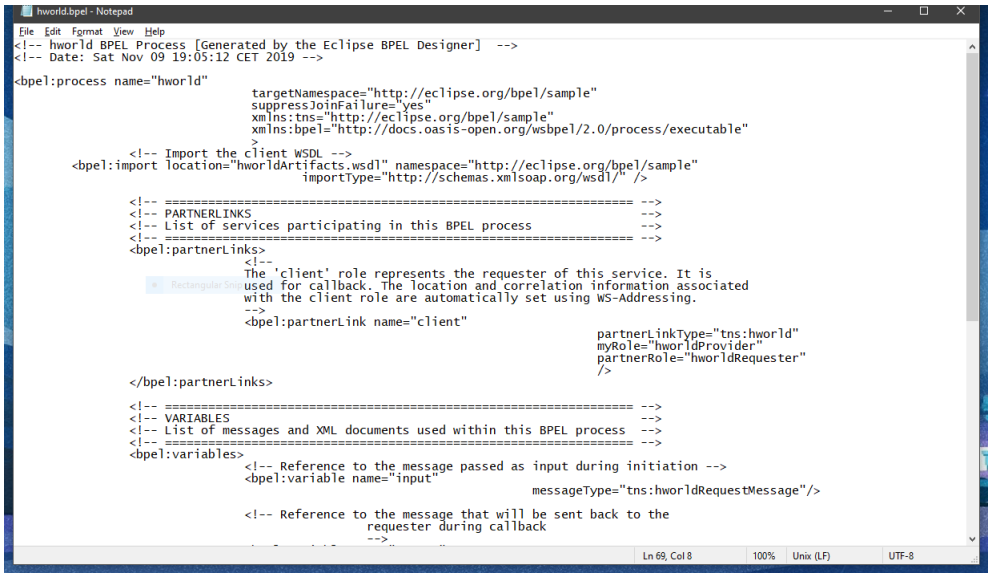
6.1. ábra. Az alkalmazás indulóképernyője

gombot. A load gomb segítségével egy BPEL file tallózható, míg a start a file kiválasztása után indítja a konverziós és szimulációs szubrutinokat. Példaként egy kisebb BPEL folyamatot mu-



6.2. ábra. Az alkalmazás tallózó ablaka

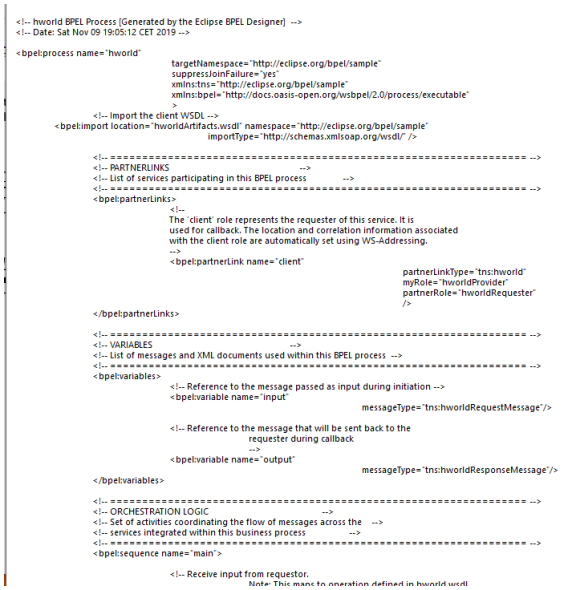
tatunk be, mely egy szekvenciát és egy kiíratást tartalmaz. A forráskód egy részletét mutatja az alábbi ábra. A teljes forráskód a mellékletben található meg.



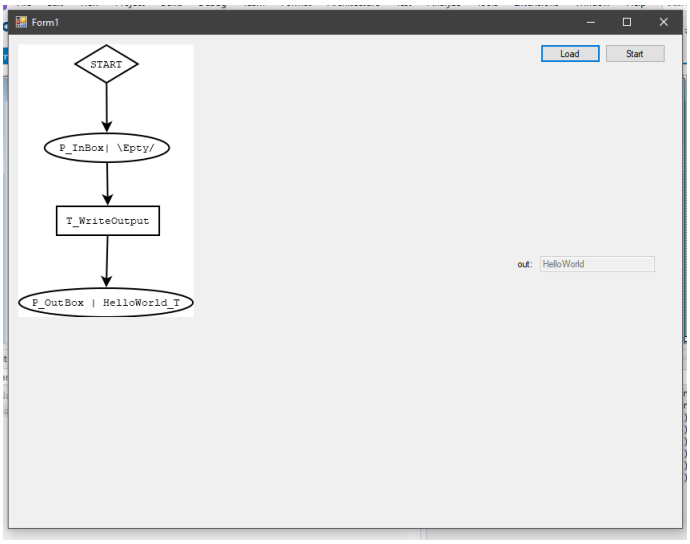
6.3. ábra. A minta forrás BPEL leírás részlete

A tesztelés során szükségessé vált a beolvasott file kiíratása, ezért DEBUG üzemmód esetén a beolvasott file a képernyőre kerül egy MessageBox tartalmaként.

Ha a konverzió sikeres akkor a megjelenítőbe kerül a fordított folyamat, ha nem akkor az alkalmazás üzenttel jelzi a konverziós hibát. Sikeres futás után az alkalmazás vissza áll STOP állapotba.



6.4. ábra. Az alkalmazás debug üzenete



6.5. ábra. Az alkalmazás szimuláció végi állapota

6.2. Python implementáció

A dolgozatban két programozási nyelvet használtam, ami az alábbiakkal indokolható. Az első hogy, a dolgozat kezdeti szakaszán még csak a modellezésre gondoltam, és ezért egy olyan nyelvet választottam ami konverziós és automatizálási lehetőségekben gazdag, mindemellett hatékony és gyors is, valamint már tapasztaltabban alkalmazom. Így esett a választás a C# -ra. A dolgozat későbbi fejezetében tárgyalom a validáció számítást, ami optimalizáló algoritmust használ. Ez a funkció később merült fel a projektben. Kezdetben az is C# nyelven lesz implementálva. Később komplikáció adódott, ami a második indok; A C# , ugyanis főleg nagyvállalati termelésre szánt programnyelv, nem kutatási célokra készült. Ez a gyakorlatban azt jelenti, hogy bár LP feladatot lehet vele megoldani, nem rendelkezik megfelelő (ingyenes, vagy kényelmesen alkalmazható) LP feladatmegoldó könyvtárral. Ekkor került a fókusz a mára általánossá vált adatmodellezési, és kutatási nyelvre, a Pythonra. A Python PuLP csomagja egy LP feladat megoldó csomag, ami rendelkezik előregyártott parametrizálásra kész szubrutinokkal és struktúrákkal, amik megkönnyítik az LP megoldásokat. A megoldás egyenes úton előáll

és nem kell extra műveletek sokaságát végezni. A nyelv egyszerűsége miatt a Python nem szült különösebb problémákat. A két modul kommunikációja megoldható egy kapcsolattartó modullal vagy rendszerhívásokkal.

6.3. Tesztelés, tapasztalatok

A megjelenítéskor probléma lehet a gép számítási kapacitása, vagy annak kihasználhatatlansága. Tesztelés során nagy méretű hálók generálása során, a rajzfolyamat elhúzódhat, ez azt eredményezi, hogy a megjelenítéskor az animáció lassabb lesz, mert a megjelenítő szubrutin a képek elkészülésére vár. A lassulási probléma valamilyen szinten kiküszöbölhető a program aszinkron, több szálú futtatásával, de ekkor ügyelni kell arra, hogy az így generált képek megfelelő sorrendben legyenek pipeolva a megjelenítő számára. (Mivel a két tesztgépen nem volt értelme a többszálú futtatásnak, ezért kivételre került. Az első gépen a processzor egy magja csak több ezer node esetén kezdett lassabban dolgozni, a második gépen a 2 mag pedig nem hozott számottevő javulást.)

A teszteket nehezítette, hogy nem mindig lehetséges kölcsönös, egyértelmű leképzés, ezért a képzési szabályok se adódtak triviálisan. Ezekről alapos kivizsgálással lehetett csak meggyőződni, hogy valóban helyes eredményt produkálnak. További nehezítő körülmény, (bár nem képzi a dolgozat törzs szoftveres részét) a különböző szükséges BPEL teszt szerverek megléte. A Microsoft biztosít BizTalk 2010es kiadású teszt szervert fejlesztésre, de ez szigorúan Hyper-V virtuális gépen futtatható. (Magáért a tényleges szerver szoftverért csak mint cég/vállalkozás lehet teszt licencet igényelni. Továbbá a Microsoft nem folytatja BizTalk szerverhez tartozó jelentős frissítések fejlesztését, és jelenleg nem tervezi új szerver kiadását sem.) A másik alternatíva, a szabadon elérhető Apcahe ODE szerver. Ennek beüzemelése egy JAVA virtuális géppel kezdődik. Erre épül egy Apache TomCat szerver, ami ténylegesen az ODE környezetet tudja biztosítani. Ez az együttállás azt eredményezi, hogy bármelyik modul enyhe hibája egy láncreakciót indíthat, ami ugyan engedi a szervert futni, de tényleges használata nem lehetséges ilyenkor. További érdekesség, hogy a portokon rendszerint fennakad, azaz ha osztozni kell egy porton (pl 8080) még ha az nincs jelenleg használva, de (Windows rendszerben lehet előre foglalni, ilyenkor) valamelyik program igényelheti, hiába akadályozza a Windows a portok egymásra definícióját az ODE nem működik vagy ép JAVA runtime errort ad ami nincs rendesen dokumentálva vagy timeout hosszú ideig várakozik, aztán hibakóddal leáll.

7. fejezet

A hálón végezhető elemzések

7.1. Háló korlátosság és puffer kapacitási ellenőrzés

A háló egy adott helye akkor tekinthető korlátos (*bounded*) helynek, ha bármely jelölésnél a tokenek száma az adott helynél nem megy egy adott korlát fölé. A Petri háló korlátos, ha minden helye korlátos hely. A háló korlátossága az egyik leggyakoribb és legfontosabb minőségi jellemzője a Petri hálóknak.

A háló alap tulajdonságainak, beleértve a korlátosságának az elemzésére több módszer is létezik, melyek közül kiemelhető a

- komponens/elérhetőségi gráf elemzése (SCC),
- dekompozíciós módszerek.

A mátrix reprezentáció esetén transzformációs mátrixok segítségével írják fel a Petri háló dinamikáját. Az alapstruktúra az ú.n. incidencia M mátrixban kerül megadásra, melynek elemei az alábbi jelentéssel bírnak: $A_{ij} = a_{ij}^{+} - a_{ij}^{-}$, ahol

- a_{ij}^{+} : az élerősség az i. tranzícióból a j. kimeneti hely felé
- a_{ij}^{-} az élerősség az i. tranzícióhoz a j. bemeneti hely felől.

A mátrix alapvetően a tokenek számának a változását mutatja az egyes tranzició átmenetek esetére. A Petri háló működési alapegyenlete a következő alakban adható meg:

$$M_k = M_{k-1} + Au_k,$$

ahol M_k jelöli a háló markereinek (tokenek) státuszát a k. lépésben. Az u vektor a helyek tüzelési státuszt írja le.

A fenti modellen alapuló elérhetőség vizsgálatok felhasználhatóak a korlátosság elemzésére [11]. A kapcsolódó egyenletek hatékony, lineáris programozási megoldása szintén ismert [9].

7.2. Saját modell alaphálóra

Modellünkben korlátosságnak egy folyam-gráf megközelítését dolgoztuk ki. A hálóban a következő típusú helyeket definiáljuk:

- forrás hely,
- nyelő hely,
- köztes hely.

Feltesszük, hogy csak a köztes helyeken lehet tokeneket tárolni, csak ott vannak pufferek. A hálóban az élekhez egy I_x token áramlás erősséget definiálunk, ahol x jelöli az él indexét. A forrás helyekhez egy Q_x forrás erősség indexet adunk meg. A hálóban az alábbi kapacitás korlátokat vezetjük be:

- C_x : az x. tranzíció maximális erőssége,
- C_y : az y. nyelő maximális folyam erőssége.

A tranzícióknál a bemenő élek vonatkozásában kétféle működési módot értelmezünk:

- AND-mód: akkor van tüzelés, ha minden bejövő élnél megvan az al elvárt tokenszám, van szinkron,
- OR-mód: akkor van tüzelés, ha megjelenik valamely bemeneten egy token, nincs szinkron.

A hálóban az alábbi megkötések élnek a folyamerősségekre:

- forrás helyek esetén: $\sum_y I_y = Q_x$ (y: kimenő élek),
- nyelő helyek esetén: $\sum_y I_y \leq C_x$ (y: bejövő élek),
- belső helyek esetén: $\sum_{y(ki)} I_y \leq \sum_{y(be)} I_y$,
- tranziciók esetén:
  - $\sum_y I_y \leq C_x$ (y : bejövő élek),
  - $\sum_{y(ki)} I_y = \sum_{y(be)} I_y$,
  - $\forall$ kimenő x, y élre $I_x = I_y$.

Az AND típusú tranzakciók esetén még ezen felül teljesül, hogy $\forall$ bejövő x, y élre:
$I_x = I_y$.

Az egyes belső helyeken a pufferbe áramló tokenek eredő intentitása:

$$F = \sum_{x(\text{belső hely})} \left(\sum_{y(\text{x bejövő él})} I_y - \sum_{y(\text{x kimenő él})} I_y \right)$$

Az F függvény 0 értéke esetén nincs szükség belső pufferre.

A fenti feladatot egy LP programozási feladatnak is tekinthető, ahol a változók az élek I_x nem negatív intenzitásai és a célfüggvény:

$$F \Rightarrow \min$$

alakú.

7.3. Az alkalmazott, kibővített modell színezett hálóra

A színezett Petri-hálók esetén több különböző típusú tokenek élnek a rendszerben. A kapacitás vizsgálatnál ekkor az egyes tranzícióknál eltérő lehet a kapacitás korlát (a maximális folyam erősség) a különböző típusú tokenek esetén. Emiatt külön kell vizsgálni az egyes típusok folyam erősségét, nem lehet összevonni őket.

A színezett hálóban az élekhez I_x^c token áramlás erősségeket definiálunk, ahol x jelöli az él indexét és c a színkód. A forrás helyekhez Q_x^c forrás erősség indexeket adunk meg a különböző c színekre vonatkozólag. A hálóban az alábbi kapacitás korlátokat vezetjük be:

- C_x^C: az x. tranzíció maximális erőssége a c szín esetén
- C_y^C: az y. nyelő maximális folyam erőssége a c szín esetén

A hálóban az alábbi megkötések élnek a folyamerősségekre:

- forrás helyek (x) esetén: $\forall c$ színre: $\sum_{y \text{ kimenő élek}} I_Y^C = Q_x^C$
- nyelő helyek (x) esetén: $\forall c$ színre: $\sum_{y \text{ bejövő élek}} I_y^C \leq C_x^C$
- belső helyek esetén: $\forall c$ színre: $\sum_{y \text{ kimenő élek}} I_y^C \leq \sum_{y \text{ bejövő élek}}$
- tranzíciók (x) esetén:

$$\forall c \text{ színre} \sum_{y \text{ kimenő élek}} I_y^C == \sum_{y \text{ bejövő élek}} I_y^C$$

$$\sum_{c \text{ színek}} \left(\frac{1}{C_x^C} \left(\sum_{y \text{ bejövő élek}} I_y^C \right) \right) \leq 1$$

$$\forall c \text{ színre:} \forall \text{ kimenő} (y,z) \text{élre: } I_y^C = I_z^C$$

- az AND típusú tranzakciók esetén még ezen felül teljesül, hogy $\forall c$ színre: $\forall$ bejövő (y,z) élre $I_y^C = I_z^C$

Az egyes belső helyeken a bufferbe áramló tokenek eredő intenzitása:

$$F = \sum_{c \text{ színek}} \left(\sum_{x \text{ belső hely}} \left(\sum_{y \text{ bejövő élek } x\text{-nél}} I_y^C - \sum_{y \text{ kimenő helyek } x\text{-nél}} I_y^C \right) \right).$$

Az F függvény 0 értéke esetén nincs szükség belső bufferre. A fenti feladat egy lineáris programozási feladatnak (röviden LP) is tekinthető, ahol a változók az élek I_x nem negatív intenzitásai és a célfüggvény $F \to \min$ alakú.

7.4. A validációs számítás algoritmusa

A hálót leíró struktúra három alappilléren nyugszik: helyek, tranzíciók, élek.

A helyek esetén az alábbi attribútumokat tárolja a rendszer:

- `id`: az egyedi azonosító kód,
- `inputs`: bejövő élek,
- `outputs`: kimenő élek,
- `tokens`: tárolt tokenek,
- `Q`: forrás intenzitás,
- `border`: pozíció jelző, belső vagy határ pozíció.

A tranzíciók jellemzői:

- `id`: egyedi azonosító kód,
- `inputs`: bejövő élek,

- `outputs`: kimenő élek,
- `C`: feldolgozási intenzitás,
- `mode`: működési mód (AND, OR).

Az élek attribútumai:

- `id`: azonosító kód,
- `input`: induló elem,
- `output`: cél elem,
- `alfa`: az él kapacitás jelzője,
- `inner`: él típusa, belső vagy határ.

A kapacitás vizsgálatot végző rutin az alábbi tevékenységeket hajtja végre. Új LP feladat létrehozása a `prop` változóba:

```
prop = Init_LpProblem(Minimize)
```

Az optimalizálási probléma változóinak inicializálása:

```
tr_vars = LpVariable("Iv", tr_items, lowBound=0, cat='Continuous')
```

Együtthatók meghatározása és beállítása:

```
for pp in places:
        if pp.border == 0:
                costs[ll] = costs[ll] +/- 1
```

A célfüggvény meghatározása:

```
Init_lpSum([costs[i] * tr_vars[i] for i in tr_items])
```

Belső pontok súlyának meghatározása:

```
for pp in places:
    if pl.border == 0:
        for e in pl.inputs:
            wgts[e] = wgts[e] + 1
        for e in pl.outputs:
            wgts[e] = wgts[e] - 1
        cnts = 0
```

Az egyenlőtlenség rendszer együtthatóinak meghatározása:

```
Init_lpSum([wgts[i] * tr_vars[i] for i in tr_items]) >= cnts
if pl.border == 1:
    for e in pl.outputs:
        wgts[e] = wgts[e] + 1
    cnts = pl.Q

Init_lpSum([wgts[i] * tr_vars[i] for i in tr_items]) == cnts
if pl.border == 2:
```

```
    for e in pl.inputs:
        wgts[e] = wgts[e] + 1
    cnts = -pl.Q

Init_.lpSum([wgts[i] * tr_vars[i] for i in tr_items]) <= cnts
for tr in self.transitions:
    for e in tr.inputs:
        wgts[e] = wgts[e] + 1
    for e in tr.outputs:
        wgts[e] = wgts[e] - 1

Init_lpSum([wgts[i]*tr_vars[i] for i in tr_items]) == 0
    for e in tr.inputs:
        wgts[e] = wgts[e] + 1
    cnts = tr.C

Init_lpSum([wgts[i] * tr_vars[i] for i in tr_items]) <= cnts

if tr.mode == 'AND':
    for e in range(1, len(tr.inputs)):
        Init_lpSum([wgts[i] * tr_vars[i] for i in tr_items]) == 0
    for e in range(1,len(tr.outputs)):
        Init_lpSum([wgts[i]*tr_vars[i] for i in tr_items]) == 0

if tr.mode == 'OR':
    for e in range(1,len(tr.outputs)):
        e1 = tr.outputs[0]
        e2 = tr.outputs[e]
        wgts[e1] =  1
        wgts[e2] =  -1
        Init_p.lpSum([wgts[i]*tr_vars[i] for i in tr_items]) == 0
```

Az optimalizálási probléma megoldása, majd az eredményeinek a kiírása:

```
prob.solve()
prob.print()
```

7.5. Mintafeladat

Vegyünk egy 4 helyből és 2 tranzícióból álló rendszert. A helyekből egy nyelő, egy forrás és kettő belső hely. A rendszerben 6 él van az ábrán megadott módon. A gráfban a sárga elem a helyeket, zöld a tranzíciókat jelöli. A csomópont elemben az első jel a hely kódja, a második a kapcsolódó kapacitás érték. A modellben a kisebb indexű tranzíció AND tulajdonságú, a másik OR tulajdonságú.

A rendszerben 6 (nem negatív) változó jelenik meg: $\{0 : Iv_0, 1 : Iv_1, 2 : Iv_2, 3 : Iv_3, 4 : Iv_4, 5 : Iv_5\}$

Az élek indexelése:

$$0:1 \to 5$$
$$1:5 \to 2$$
$$2:2 \to 6$$
$$3:6 \to 3$$
$$4:3 \to 5$$
$$5:6 \to 4$$

A rendszerhez az alábbi egyenlőtlenségek kapcsolódnak:

$$\begin{array}{rll} C_1 & : Iv_0 & = 20 \\ C_2 & : Iv_1 - Iv_2 & \geq 0 \\ C_3 & : Iv_3 - Iv_4 & \geq 0 \\ C_4 & : Iv_5 & \leq 25 \\ C_5 & : Iv_0 - Iv_1 + Iv_4 & = 0 \\ C_6 & : Iv_0 + Iv_4 & \leq 50 \\ C_7 & : Iv_0 - Iv_4 & = 0 \\ C_8 & : Iv_2 - Iv_3 - Iv_5 & = 0 \\ C_9 & : Iv_2 & \leq 50 \\ C_{10} & : Iv_3 - Iv_5 & = 0 \end{array}$$

A kapcsolódó célfüggvény:

$$1 \cdot Iv_1 + -1 \cdot Iv_2 + 1 \cdot Iv_3 + -1 \cdot Iv_4 \Rightarrow \min$$

Az LP feladat megoldható és a kapott megoldás:

$$\begin{array}{ll} Iv_0 & : 20.0 \\ Iv_1 & : 40.0 \\ Iv_2 & : 40.0 \\ Iv_3 & : 20.0 \\ Iv_4 & : 20.0 \\ Iv_5 & : 20.0 \\ Cost & = 0.0 \end{array}$$

Tehát a mintarendszerben nincs szükség belső pufferre. Ha lecsökkentjük a második tranzíció folyamerősségét, az akkor nem kapunk érvényes megoldást.
Infeasible

$$\begin{array}{ll} Iv_0 & : 10.0 \\ Iv_1 & : 20.0 \\ Iv_2 & : 20.0 \\ Iv_3 & : 10.0 \\ Iv_4 & : 10.0 \\ Iv_5 & : 10.0 \end{array}$$

8. fejezet

Összegzés

Összegzés

A BPEL szabvány az üzleti folyamatok szabványos leírására szolgál. A dolgozat témaköre a BPEL nyelven létrehozott üzleti folyamatok modellezése és elemzése a Petri-háló alapú formalizmus segítségével. A kidolgozott mintarendszer inputként egy BPEL modell leírását várja és kimenetként az elemzés eredményét illetve a folyamatok nyomkövetését adja vissza.

A TDK munka keretében az alábbi eredményeket értem el:

- BPEL folyamatok Petri háló formalizmusra történő konverziója,
- LP alapú végesség vizsgálat a Petri hálón,
- folyamatok grakus nyomon követése, szimuláció.

A projekt következő lépéseként a modell logisztikai folyamatokra történő adaptálását végezzük el. A továbbfejlesztett szimulációs motorba több új elosztás alapú generációs modult is el fogok készíteni. Az eredményeket egy cikk publikációjában kívánjuk megjeleníteni.

Irodalomjegyzék

[1] List of bpel engines. `https://en.wikipedia.org/wiki/List_of_BPEL_engines.htm`. Accessed: 2019-10-30.

[2] Andrews, T., Curbera, F., Dholakia, H., Goland, Y., Klein, J., Leymann, F., Liu, K., Roller, D., Smith, D., Thatte, S., et al. Business process execution language for web services, 2003.

[3] Baresi, L., and Guinea, S. Towards dynamic monitoring of ws-bpel processes. In *International Conference on Service-Oriented Computing* (2005), Springer, pp. 269–282.

[4] Baresi, L., Maurino, A., and Modafferi, S. Towards distributed bpel orchestrations. *Electronic Communications of the EASST 3* (2007).

[5] Cardoso, J. Complexity analysis of bpel web processes. *Software Process: Improvement and Practice 12*, 1 (2007), 35–49.

[6] Hackmann, G., Haitjema, M., Gill, C., and Roman, G.-C. Sliver: A bpel workflow process execution engine for mobile devices. In *International Conference on Service-Oriented Computing* (2006), Springer, pp. 503–508.

[7] Kopp, O., Görlach, K., Karastoyanova, D., Leymann, F., Reiter, M., Schumm, D., Sonntag, M., Strauch, S., Unger, T., Wieland, M., et al. A classification of bpel extensions. *Journal of Systems Integration 2*, 4 (2011), 3–28.

[8] Kovács, M., Varró, D., and Gönczy, L. Formal analysis of bpel workflows with compensation by model checking. *Computer Systems Science and Engineering 23*, 5 (2008), 349–363.

[9] Lasserre, J. B., and Mahey, P. Using linear programming in petri net analysis. *RAIRO-Operations Research 23*, 1 (1989), 43–50.

[10] Mayer, P., and Lübke, D. Towards a bpel unit testing framework. In *Proceedings of the 2006 workshop on Testing, analysis, and verification of web services and applications* (2006), ACM, pp. 33–42.

[11] Murata, T. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE 77*, 4 (1989), 541–580.

[12] Ouyang, C., Dumas, M., Breutel, S., and ter Hofstede, A. Translating standard process models to bpel. In *International Conference on Advanced Information Systems Engineering* (2006), Springer, pp. 417–432.

[13] Ouyang, C., Verbeek, E., van der Aalst, W. M., Breutel, S., Dumas, M., and ter Hofstede, A. H. Wofbpel: A tool for automated analysis of bpel processes. In *International Conference on Service-Oriented Computing* (2005), Springer, pp. 484–489.

[14] Rosenberg, F., and Dustdar, S. Business rules integration in bpel-a service-oriented approach. In *Seventh IEEE International Conference on E-Commerce Technology (CEC'05)* (2005), IEEE, pp. 476–479.

[15] Slominski, A. Adapting bpel to scientific workflows. In *Workflows for e-Science*. Springer, 2007, pp. 208–226.

[16] Van Der Aalst, W. M., and Lassen, K. B. Translating unstructured workflow processes to readable bpel: Theory and implementation. *Information and Software Technology 50*, 3 (2008), 131–159.

9. fejezet

Mellékletek

9.1. A minta BPEL folyamat forráskódja

```
<!-- hworld BPEL Process [Generated by the Eclipse BPEL Designer]  -->
<!-- Date: Sat Nov 09 19:05:12 CET 2019 -->

<bpel:process name="hworld"
 targetNamespace="http://eclipse.org/bpel/sample"
 suppressJoinFailure="yes"
 xmlns:tns="http://eclipse.org/bpel/sample"
 xmlns:bpel="http://docs.oasis-open.org/wsbpel/2.0/process/executable"
 >
<!-- Import the client WSDL -->
<bpel:import location="hworldArtifacts.wsdl" namespace="http://eclipse.org/bpel/sample"
importType="http://schemas.xmlsoap.org/wsdl/" />

<!-- ================================================================= -->
<!-- PARTNERLINKS                                                      -->
<!-- List of services participating in this BPEL process               -->
<!-- ================================================================= -->
<bpel:partnerLinks>
<!--
The 'client' role represents the requester of this service. It is
used for callback. The location and correlation information associated
with the client role are automatically set using WS-Addressing.
-->
<bpel:partnerLink name="client"
 partnerLinkType="tns:hworld"
 myRole="hworldProvider"
 partnerRole="hworldRequester"
 />
</bpel:partnerLinks>

<!-- ================================================================= -->
<!-- VARIABLES                                                         -->
<!-- List of messages and XML documents used within this BPEL process  -->
<!-- ================================================================= -->
<bpel:variables>
```

```
<!-- Reference to the message passed as input during initiation -->
<bpel:variable name="input"
messageType="tns:hworldRequestMessage"/>

<!-- Reference to the message that will be sent back to the
 requester during callback
 -->
<bpel:variable name="output"
messageType="tns:hworldResponseMessage"/>
</bpel:variables>

<!-- ================================================================= -->
<!-- ORCHESTRATION LOGIC                                               -->
<!-- Set of activities coordinating the flow of messages across the    -->
<!-- services integrated within this business process                  -->
<!-- ================================================================= -->
<bpel:sequence name="main">

<!-- Receive input from requestor.
 Note: This maps to operation defined in hworld.wsdl
 -->
<bpel:receive name="receiveInput" partnerLink="client"
 portType="tns:hworld"
 operation="initiate" variable="input"
 createInstance="yes"/>
<bpel:empty name="FIX_ME-Add_Business_Logic_Here"></bpel:empty>
<!-- Asynchronous callback to the requester.
 Note: the callback location and correlation id is transparently handled
 using WS-addressing.
 -->
<bpel:invoke name="callbackClient"
partnerLink="client"
portType="tns:hworldCallback"
operation="onResult"
inputVariable="output"
/>
</bpel:sequence>
</bpel:process>
```