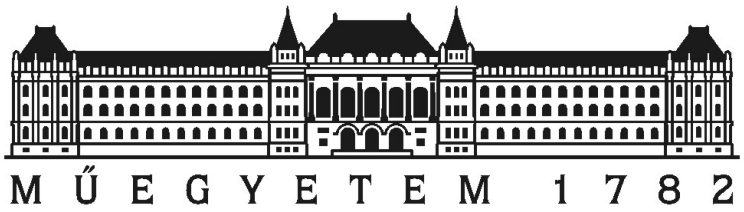


Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar

Hartwig János, Urbán Balázs

Szabályalapú diagnosztika üzleti folyamat vezérelt rendszerekben

Konzulens:

Gönczy László
Méréstechnika és Információs Rendszerek Tanszék

Tasi Katalin
Quanopt kft.

Budapest, 2013

Kivonat

Szoftver rendszerek megvalósításának egyik tipikus módja az üzleti folyamat alapú tervezés, melynek előnye, hogy a rendszer működését az adott terület szakértői számára átláthatóbbá teszi. Ilyen rendszerekben felmerül a kérdés, hogyan lehet i) igazolni azt, hogy a specifikáció (folyamatmodell) megfelel a felhasználók, üzemeltető, hatóságok, stb. elvárásainak, ii) igazolni azt, hogy a rendszer működése megfelel a specifikációban foglaltaknak, iii) kiértékelni magának a rendszernek a működését mind informatikai, mind üzleti teljesítmény, valamint szolgáltatásbiztonság szempontjából. Emellett a folyamatok komplex rendszerekre építhetnek, melyek belső hibái mellett adathibák, ill. felhasználói hibák akadályozhatják a rendszer elvárt működését. A gyakori futtatások, vagy a hosszú idejű használat esetén nagy mennyiségű historikus adat termelődik. Ilyenkor már nehézkessé válik az adatok elemzése, ezért szükség lehet a folyamatok hatékony diagnosztikai támogatására, ami segíti a szakértők munkáját.

A probléma megoldására készítettünk egy olyan környezetet, ahol a szakértő a folyamat saját fogalomkészletét használva írhat fel szabályokat, amiket a korábban futtatott folyamatok lefutásából generált eseményeken kiértékelhet. Az elkészített szabályrendszer ellenőrizhető teljességi és helyességi szempontból. Például ha a szakértő megjelöli a modellben a kritikus részeket, akkor ellenőrizhető, hogy az üzleti elemző az összes fontos esetre vonatkozó szabályt felírta. Az összes szabály felírásán túl, az alkalmazott szabályrendszer konzisztenciájának ellenőrzése is fontos feladat, hiszen ellentmondásos szabályok használata helytelen, megtévesztő eredményt adhat. A szabályok szemantikai vizsgálata tehát egy olyan lépés, amely nélkülözhetetlen a helyes és valós diagnosztika elkészítéséhez.

A szabályrendszer továbbfejlesztésének egy módja lehet a szabályok tényleges lefutásának vizsgálata. Például ha egy riasztási jellegű szabály kiugróan sokszor fut le, az átgondolásra figyelmeztethet, hiszen ennek oka lehet, hogy hibás a feltétel, vagy érdemes felbontani a szabályt több alesetre. De az is elképzelhető, hogy ez a rendszer kiugróan rossz működéséről tanúskodik. A dolgozatban bemutatjuk, hogyan lehet ilyen vizsgálatokat végezni, feltáró adatanalízis segítségével.

Mivel a dolgozat ipari projektből indult ki, ezért fontos szerepe volt a sebességnek, valós környezetben való alkalmazhatóságnak is, így több tesztet is végeztünk.
Vizsgáltuk a szabálykiértékelés sebességét, egy komplexebb példán ellenőrizzük a teljesség és a helyesség vizsgálat működését, valamint ugyanezen a példán bemutatjuk a feltáró adatanalízis hasznosságát.

Abstract

The implementation of software systems is often based on a design captured by business processes. This has the advantage that a domain expert can easily understand the operation of the system. In such systems, some important questions can be raised; i) how to prove that the specification meets the stakeholders' (users, operators, authorities, etc..) expectations, ii) how to prove that the operation of the system meets the specifications, iii) how to evaluate the system in terms of IT and business performance and dependability. In addition, activities in the processes can be based on functionalities provided by complex systems where internal faults and data errors can prohibit the desired functionality. Rule-based diagnosis can help to answer these questions. However, after many runs or long operational time, huge amount of data can be produced. In this case the analysis may become difficult therefore an effective tool is needed, which supports the experts' work.

For this reason, we created an environment where the business expert has the possibility to perform analysis by writing rules in a domain specific language, generated from the process model. These will be evaluated on events, generated by previously executed processes. The correctness and completeness of this rulebase can be analysed. For instance, if an expert notes some critical parts of a model, then it can be checked whether the business analytic has written rules for all important cases. In the design of diagnosis rulebases it is important to check the rules consistency since usage of controversial rules can lead to an incorrect, false statement. So the semantic analysis of the rules is essential to make correct diagnostics.

In the maintenance and development of such rulebases it's worth to investigate the behaviour of rules. For instance if a rule is executed extremely often, then it may indicate a design flaw, (e.g., the cause of this effect can be a bad condition formulation in the rule, or maybe it's worth to split the rule for multiple smaller cases) or it can be caused by a faulty component of the system. We present how to support this investigation by exploratory data analysis.

Our research was inspired by anindustrial project, so it was important to create a software which meets practical performance and usability requirements.

We tested the usability of the method on a real-life case study, and using a complex example we checked the work of the examination of correctness and completeness. In this same example we present the usefulness of exploratory data analysis.

Tartalomjegyzék

1. Bevezetés **1**
1.1. Motiváció . . . 1
1.2. Problémafelvetés . . . 2
1.3. Célkitűzés . . . 2
1.3.1. Diagnosztikai környezet kialakítása . . . 2
1.3.2. Szabályok statikus elemzése . . . 3
1.3.3. Szabálylefutások historikus elemzése . . . 3
1.3.4. Gyakorlati használhatóság . . . 4
1.4. Megközelítés . . . 4
1.4.1. Technológiai háttér . . . 5

2. Diagnosztika üzleti folyamatokban **6**
2.1. Üzleti folyamatok . . . 6
2.1.1. BPMN . . . 6
2.1.2. Esettanulmány . . . 7
2.2. Diagnosztika . . . 7
2.2.1. Eseményfeldolgozás . . . 8
2.2.2. Diagnosztika tervezése . . . 9

3. Kapcsolódó munkák **11**
3.1. Szabálygenerálás . . . 11
3.2. Grafikus szabályok, hibaok felderítés . . . 12
3.3. A szabályrendszer felírásának általánosítása . . . 13
3.4. Valós idejű diagnosztika . . . 13

4. Megvalósítás **14**
4.1. Diagnosztikai környezet . . . 14
4.1.1. Folyamatmotor . . . 15
4.1.2. Eseményfeldolgozó . . . 16
4.1.3. Szabályvégrehajtó motor . . . 17
4.1.4. Szabályszerkesztő . . . 18
4.1.5. Generátor . . . 20
4.1.6. Szimulátor . . . 24
4.1.7. Felhasználókezelés . . . 25
4.2. Szabályok és lefutásaik elemzése . . . 26

4.2.1. Statikus elemzés 26
4.2.2. Lefutások elemzése 29

5. Továbbfejlesztési lehetőségek **33**
5.1. További eseményforrások bevonása 33
5.2. Kezdeti szabálybázis generálása 33
5.3. Valós idejű szabálykiértékelés 33
5.4. Az eredmények visszavezetése a kezdeti folyamatmodellre 34
5.5. Az EDA során észrevett eredmények alapján szabályjavatok készítése . . 34
5.6. EPA használatával a hibaokok felderítése 34
5.7. A szabályok felírásának nyelvi támogatása 34
5.8. A komponensek rugalmas cserélhetősége 34
5.8.1. Folyamatmotor 35
5.8.2. Adatbáziskezelő 35
5.8.3. Szabályszerkesztő, folyamatmotor 35

6. Összefoglalás **36**
Irodalomjegyzék 38

1. fejezet

Bevezetés

Az üzleti folyamat alapú tervezés a szoftverrendszerek megvalósításának egy igen gyakran alkalmazott módja, nagy előnye, hogy átláthatóbbá teszi a rendszer működését az adott szakterület, a felhasználók számára.

A komplex rendszerek üzemeltetése közben problémát okozhat a rendszer működésének vizsgálata, annak megállapítása, hogy a rendszer megfelel-e a specifikációnak, a felhasználók és a megrendelők elvárásainak, vagy valamely szabványnak. További nehézséget jelenthet a rendszer üzleti, informatikai teljesítményének vizsgálata. A dolgozat során ezekre a problémákra próbálunk megoldást nyújtani.

1.1. Motiváció

Az üzleti folyamat alapú modellezés igen elterjedten alkalmazott fejlesztési technológia különböző rendszerek tervezésénél. Vállalatok belső ügymeneteinek, gyártási folyamatok modellezésére egyaránt használják.

Ugyanakkor megjelenik a jogos igény, hogy ellenőrizzük a rendszert tervezési és működési szempontból. Számos területen fontos lehet a folyamat tervezési, konfigurációs és működési hibáinak ellenőrzése annak érdekében, hogy a modell megfelelősége, illetve a modell szerinti működés garantálható legyen (pl. az adott implementációban nem keürlnek-e meg valamilyen követelményt/korlátozást). Ez utóbbihoz szükség lehet különböző információforrásokra a folyamatok alatt lévő erőforrásokból, ami bonyolítja az ellenőrzést.

A vállalatoknak különböző elvárásoknak, előírásoknak kell megfelelniük. Ilyenek a szabványok, ajánlások (például az informatikai biztonsággal foglalkozó COBIT[3]), törvények (például USA-ban a vállalatokra vonatkozó SOX[9]) és a vállalatok belső szabályzata is. Ezeket több folyamaton belül már nehéz átlátni, ellenőrizni. Dolgozatunkban igyekszünk megoldást találni az auditálás támogatására.

A megoldást elsősorban szabály alapú ellenőrző rendszer megvalósításában látjuk, ahol rögtön felmerül a kérdés: hogyan lehet a hibákat észrevevő szabályok fejlesztését támogatni? Válaszként a dolgozatban a felírt szabályhalmaz vizsgálatával (teljesség, helyesség), valamint az ellenőrzés után a szabályilleszkedéseken alapuló visszajelzéssel foglalkoztunk.

1.2. Problémafelvetés

Bár az üzleti folyamatok hatékony eszközt nyújtanak a munkafolyamatok automatizálására, azonban a folyamatok helyes végrehajtását nem tudják garantálni, sőt, az üzleti folyamatmodellező eszközök sokszor szándékosan lehetőséget is adnak a folyamat definíciójától való eltérésre, mert így rugalmasabban tud működni az adott szervezet. Ezeket az eltéréseket viszont utólag ki kell tudni mutatni, illetve számos olyan felhasználási terület létezik, ahol kritikus, hogy a folyamatok futása ellenőrizhető lehessen. Nagyszámú lefutás után ennek ellenőrzése azonban nem végezhető el manuális eszközökkel.

Az ellenőrzést, a diagnosztikát tehát segíteni kell egy hatékony eszközzel. Ennek egy lehetséges megvalósítása a szabály alapú diagnosztika, melynek segítségével üzleti elemzők végezhetnek ellenőrzéseket a folyamatokon. Így egyszerű szabályok definiálásával könnyedén észrevehetőek, szűrhetőek a hibás lefutások. Ezen felül egy hatékony diagnosztika alkalmas lehet arra, hogy segítségével könnyen vizsgáljuk szabványoknak való megfelelőséget (compliance) (Pl. [3] [9]) a folyamatokon. A szabályok felírása azonban egy újabb feladatot jelenthet. Elképzelhető, hogy az elemző hibás, vagy akár inkonzisztens szabályrendszert hoz létre.

Ezen hibák egy része ugyan nem védhető ki, de a szabályok felírása támogatható különféle vizsgálatokkal. Ellenőrizhető a szabályrendszer konzisztenciája, azaz hogy nem írt-e fel az elemző egymásnak ellentmondó, különböző kimenetelű szabályokat, ezzel kivédhetőek az esetleges téves riasztások, illetve az egymásnak ellentmondó eredmények. Valamint szakértői kategorizálás alapján ellenőrizhető az is, hogy a rendszer kritikusnak tartott elemei le vannak-e fedve diagnosztikai szabályokkal.

1.3. Célkitűzés

A dolgozat során egy olyan diagnosztikai környezetet fogunk bemutatni, amelyben az üzleti elemzők a saját szakterületük fogalmaival dolgozhatnak az üzleti folyamatokon. Ehhez megoldottuk egy folyamatmotor felműszerezését, az ebből származó adatok lementését. Megalkottunk egy folyamatmotor független eseménymodellt. A szabályok szerkesztéséhez készítettünk egy szakterület-specifikus eseménymodellt is, amely segítségével lehetővé tettük diagnosztikai szabályok definiálását egy grafikus szabályszerkesztőben. Ennek a modellnek előállítására biztosítunk egy eszközt, amely segítségével a folyamatok importálhatóak a szabályszerkesztőbe. E két modell között elvégezzük a futási idejű konverziót is.

Módszert adunk a diagnosztikai szabályok ellenőrzésére, teljességi és helyességi szempontból, valamint bemutatjuk, hogy lehet további ellenőrzéseket végezni a folyamatokon és a szabályokon feltáró adatanalízis segítségével.

1.3.1. Diagnosztikai környezet kialakítása

Célunk volt egy környezet kialakítása, mely támogatja a diagnosztikai elemzéseket és megoldást ad a fentebb felvetett problémákra. Lehetővé teszi üzleti folyamatok eseményeire szabályok felírását, azok eredményeinek megjelenítését, így segítve az elemzést.

Domain specifikus környezet készítése

Környezetünket domain specifikusnak szántuk, hogy az elemző a saját üzleti folyamataira, illetve azok lépéseire azok saját fogalomkészletével írhasson fel szabályokat, nem pedig kizárólag általános folyamatelemekre.

További eseményforrások integrálása

Az átfogóbb elemzések támogatása érdekében az üzleti folyamatmodellező eszközön kívül más eseményforrások bevonását is meg akartuk valósítani. Így észrevehetőek a használt erőforrások túlterheléséből származó hibák.

1.3.2. Szabályok statikus elemzése

A diagnosztika során az üzleti elemző által felírt szabályok minőségén múlik az eredmények használhatósága, ezért vizsgáljuk ezeket a szabályokat. Nagy mennyiségű szabály esetén ez azonban manuális módszerekkel problémát jelenthet, ezért a dolgozat során bemutatjuk egy olyan környezet megalkotását, ami képes a szabályok elemzésére, ezzel könnyítve az üzleti elemzők munkáját.

Szabályok helyessége

A szabályokat akkor nevezzük helyesnek, ha konzisztens a szabályrendszer. Konzisztencia alatt pedig az ellentmondásmentességet értjük. Az általunk elkészített eszköz képes ellenőrizni a szabályrendszer konzisztenciáját, így könnyen észrevehetőek olyan ellentmondások, mint két egybeeső szabály ellentétes kimenettel.

Szabályok teljessége

Szabályok teljessége alatt azt értjük, hogy minden fontos esetre létezik szabály. A fontos esetek azonosítása viszont automatikus eszközökkel nem tehető meg. Ezért biztosítottunk egy olyan felületet az üzleti szakértők számára, ahol bevihetik ezt a szakértői tudást a rendszerbe, azaz megjelölhetik a fontos részeket a folyamatban.

1.3.3. Szabálylefutások historikus elemzése

A szabályok tényleges lefutásának vizsgálata révén átfogóbb összefüggéseket is észre lehet venni. Ilyen például két hibajelenség közötti korreláció. Ugyanakkor a lefutások historikus elemzése során a szabályok minőségét, hatékonyságát is ellenőrizhetjük. Egy szabály kiugróan gyakori lefutása a folyamat rendkívül rossz működésén túl mutathatja, hogy rosszul, vagy esetleg feleslegesen lett a kritérium megfogalmazva.

Feltáró adatanalízis

A szabályok lefutásának historikus elemzéséhez feltáró adatanalízist (exploratory data analysis - EDA) használunk. Ez egy vizualizációs módszer, mely hatékonyan támogat-

ja a kiugró értékek megtalálását és a korrelációkeresést. Használatáról részletes példát ismertetünk a a 4.2.2 alfejezetben.

1.3.4. Gyakorlati használhatóság

A TDK dolgozatban bemutatott rendszer egy része a Quanopt Kft. számára a KMOP-1.1.4-2011A-2011-0074 projektben elvégzett munkánkon alapul. Ezért fontos volt a diagnosztika használhatósága, hogy felhasználóbarát legyen a szabályok szerkesztése, ezért döntöttünk egy grafikus szabályszerkesztő mellett. Felhasználói szempontból fontos még a diagnosztika sebessége is, hogy a szabályok fejlesztése közben azok gyorsan, könnyedén lefuttathatóak legyenek.

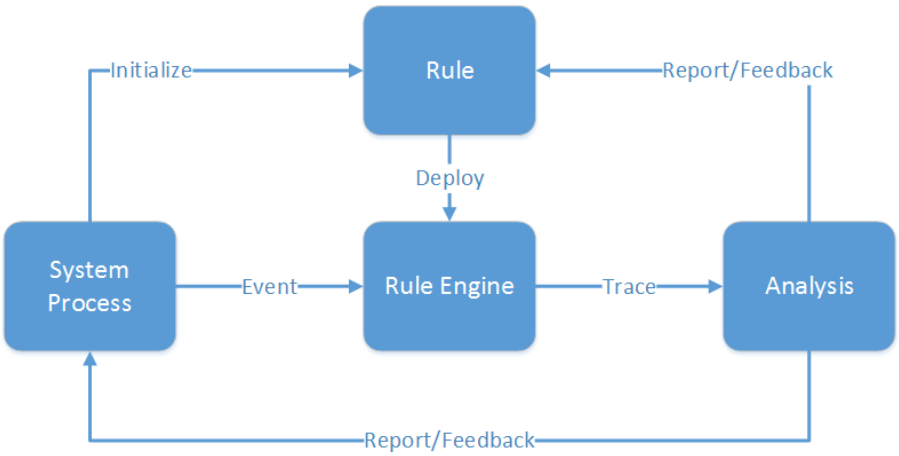
1.4. Megközelítés

A fentebb vázolt céljaink megvalósítására készült rendszerünk üzleti folyamatokra szabályok felírását teszi lehetővé, ellenőrzi azokat teljesség és helyesség szempontjából és kiértékeli korábban lefutott folyamatokra, majd ezután a szabályilleszkedésen vizuális adatelemzést tesz lehetővé.

Rendszerünk fontosabb jellemzői:

- Elválasztjuk a folyamatot és annak ellenőrzését.
- Adott folyamat domain-jére adunk szabályszerkesztő felületet.
- A felírt szabályokat ellenőrizzük (helyesség, teljesség).
- Szabálykiértékelést végzünk.

Az elkészült rendszer fő funkcióit szemlélteti, az 1.1. ábra.



1.1. ábra. A rendszer főbb funkcióinak összefoglalása

A domainspecifikus szabályfelírás lehetővé tételéhez ki kell nyernünk az üzleti folyamat modelljét és átalakítani a szabályszerkesztő felület számára feldolgozható formába. A felírt szabályokon helyesség- és teljességvizsgálatot végzünk (Analysis).

A szabályok kiértékelése (Rule Engine) offline módon működik, historikus adatokat felhasználva. Ehhez egy beépülő modult írtunk az üzleti folyamatmodellező eszközbe, mely futás közben naplóz minden folyamatokra vonatkozó eseményt.

A szabálykiértékelés után a szabályilleszkedésen feltáró adatelemzést lehet végezni (Analysis), mely alapján visszajelzéseket kapunk a folyamatokra és a szabályokra.

1.4.1. Technológiai háttér

Munkánk során nyílt forráskódú eszközöket használtunk. Üzleti folyamatmodellező eszközként a Bonitát[1], szabálykiértékeléshez a Drools-t[4], szabályszerkesztéshez pedig a Guvnor[7] nevű eszközt. Részletesebb leírásuk és a választások indoklásai a 4. fejezet egyes alfejezeteiben találhatóak.

2. fejezet

Diagnosztika üzleti folyamatokban

A következő alfejezetekben bemutatjuk a diagnosztika alapjait, az üzleti folyamatokat általánosságban, valamint kitérünk az üzleti folyamatok modell alapú megközelítésének előnyeire is.

2.1. Üzleti folyamatok

Üzleti folyamatnak nevezünk olyan munkafolyamatokat, amelyek több, egymástól jól elkülöníthető lépésből állnak. Az üzleti folyamat végrehajtása során megmunkál egy absztrakt munkadarabot, módosítja adatait. Ez a munkadarab lehet egy szerződéskötés, egy hitelkérelem, vagy bármilyen termék, amelynek a gyártási menetét írja le a munkafolyamat. Fontos, hogy ezek az összetett folyamatok felbonthatók egyszerűbb elemi tevékenységekre, amelyek így külön fejleszthetőek, tervezhetőek. Az üzleti folyamatok ezen elemi tevékenységeken kívül az ezek között futó átmenetekből, és a sorrendjüket meghatározó feltételekből állnak.

Az általunk felépített és vizsgált környezetben az üzleti folyamatok egy webszolgáltatásként jelennek meg. Ezen folyamatokra nyújtunk egy hatékony monitorozó környezetet.

Üzleti folyamatok használatával a folyamatok végrehajtása gyorsítható, esetleg egyes lépések automatizálhatók. Az üzleti folyamatok számítástechnikai háttér segítségével párhuzamosíthatók, hiszen egy absztrakt munkadarabon, például egy hitelkérelmen dolgozhatnak többen is egyszerre, míg ez papír alapú ügyintézés esetén nem valósítható meg. A folyamat informatikai megvalósítása hatékonyan tesztelhető, így akár már a futása előtt fellelhetőek a hibái. Ezen felül lényegesen könnyebben monitorozhatóak, felügyelhetőek, hiszen egyszerűbb az ehhez szükséges információk összegyűjtése. A dolgozat során bemutatjuk, hogyan lehet az üzleti folyamatok hatékony szabály alapú monitorozását megvalósítani.

2.1.1. BPMN

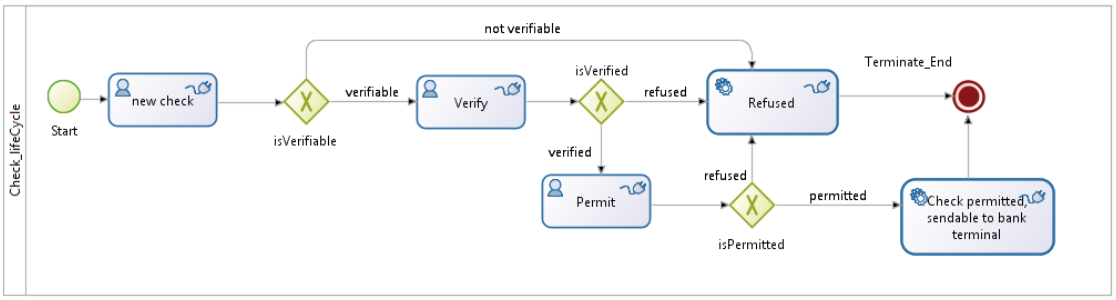
A BPMN (Business Process Model and Notation) egy OMG (Object Management Group)[2] szabvány az üzleti folyamatok grafikus reprezentációjára. Segítségével egyszerűen leírhatók a folyamatok az UML aktivitás diagramjához hasonló formában. A modellező eszkö-

zök általában ezt a nyelvet támogatják, de többnyire csak a nyelvi elemek egy részét, ahogy az általunk használt Bonita eszköz is.

Üzleti folyamatok kezelésére több modellalapú eszköz jött létre. Ezek az eszközök képesek a folyamat futtatható kódját előállítani a BPMN modellből, így jelentősen megkönnyítve a folyamatok fejlesztését, karbantartását, hiszen a modellalapú megközelítés segítségével elég csak a BPMN modellt szerkeszteni, magát a végrehajtható kódot nem szükséges. Ilyen eszköz az általunk használt Bonita[1] is, amely egy folyamat modelljéből képes egy webalkalmazás előállítására.

2.1.2. Esettanulmány

A környezet teszteléséhez elkészítettünk egy mintafolyamatot, amelyet futtatva teszteket végeztünk. A folyamat egy a vállalathoz beérkező csekk rögzítését, elfogadását, majd kifizetését írja le. A beérkező csekkeket a „new check” lépésben rögzíti a rendszerbe



2.1. ábra. Egy csekk elfogadása

az alkalmazott. Ezután ha az alkalmazott nem utasította el azonnal formai hiba miatt, akkor átkerül a „verify” fázisba, ahol egy másik alkalmazott elfogadhatja. A második alkalmazott döntésének megfelelően a csekk továbblép az engedélyezési fázisba, ahol egy alkalmazott eldönti, hogy kifizeti-e az adott csekket.

A folyamat végrehajtása során az összes folyamatlépés elmenti a csekk pillanatnyi állapotát egy adatbázisba, így később nyomon követhető marad a folyamat futása. A tervezés során fontos volt, hogy a folyamat több figyelhető paraméterrel rendelkezzen. Ezért lehetőséget adtunk a folyamat többfelhasználós végrehajtására, a csekk igen sok paraméterének beállítására is. Az adatbázis-elérés is potenciális hibaforrás lehet, így a folyamat során tudunk tesztelni olyan eseteket is, mikor nem a folyamat hibázik, hanem az alatta futó környezet. A folyamatban elrejtettünk egy hibát is, ami miatt egy konkrét esetben mindig hibával leáll a folyamat végrehajtása.

2.2. Diagnosztika

Bármilyen rendszerrel dolgozunk, főleg, ha az emberi erőforrásokra is támaszkodik, biztosak lehetünk abban, hogy előbb vagy utóbb valami nem az elvárásainknak megfelelően fog működni. Kisebb, egyszerűbb rendszereknél még elfogadható eredményt adhat a kézi

ellenőrzés, de komplexebb esetekben mindenképp érdemes diagnosztikai rendszert alkalmazni, mely segítséget és módszert ad a hibajelenségek felismerésében és azok lokalizálásában.

A diagnosztika egy tárgy, berendezés, vagy folyamat műszeres vizsgálatával szolgáltat olyan állapotinformációkat, melyek elemzésével következtetni lehet a vizsgált objektum aktuális állapotára, hibáira. Diagnosztika lehet hardvergyártás közben a memória, vagy a processzor áramkör-szintű tesztje. Szoftvergyártás esetében a programok tesztelése, monitorozása.

A diagnosztika támogathatja a fejlesztést is, hiszen rámutathatunk vele a rendszer szűk keresztmetszeteire, amik nem hibák ugyan, de érdemes tovább javítani, optimalizálni.

A dolgozat során diagnosztika alatt elsősorban a hiba detektálást és annak támogatását értjük egy működő rendszerben, ahol feltesszük, hogy az a modelljének megfelelően működik. A hibaokot nem tudjuk visszakeresni.

Ehhez historikus adatokat használunk, de a begyűjtés megtervezésével és implementálásával nem foglalkozunk. Nem vizsgáljuk az ellenőrzéshez használt adatok minőségét.

Diagnosztika esemény alapon

Az esemény alapú diagnosztika lényege, hogy ágensekkel monitorozzuk a rendszert működés közben, és a bekövetkező eseményeket azonnal, vagy offline módon feldolgozzuk. Esemény lehet minden mérhető változás, ami a rendszerben végbe megy, de akár bármilyen periodikusan küldött információ is.

A diagnosztika más megvalósítása lehet például egy állapot alapú diagnosztika, ahol a rendszer állapotait vizsgáljuk, nem a rendszerben végbemenő változásokat. Lehet csak a rendszer változásait ellenőrizni, mikor csak azok fontosak, a folytonos jellemzői nem.

2.2.1. Eseményfeldolgozás

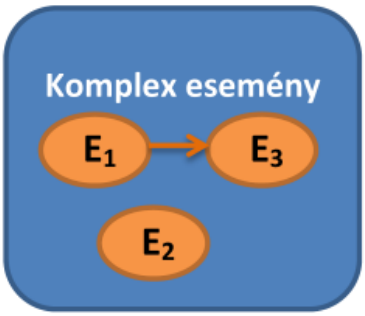
Eseményfeldolgozás során nagy mennyiségű adatból kiszűrjük a számunkra releváns információkat, mintákat. Így felismerhetünk különböző hibákat, illegális viselkedéseket rendszerünkben.

Az eseményfeldolgozásnak egyik lehetséges módja - melyet a dolgozat folyamán mi is követtünk -, hogy szabályokat írunk fel, vagyis eseménymintákat és azok bekövetkezése esetén végrehajtandó utasításokat definiálunk. A minta lehet egy hibás állapot a rendszerben, a végrehajtandó utasítás pedig egy riasztás, vagy bejegyzés-készítés. [15][10]

Komplex esemény: Több elemi eseményből összeálló struktúra, mely kiegészülhet az elemi események közötti kapcsolatokkal, például a közöttük fennálló sorrenddel.

A komplex eseményeknek fontos szerepük van az eseményfeldolgozásban, leírhatunk velük olyan mintákat, amik számunkra fontosak, érdekesek és ezeket kereshetjük az adathalmazunkban.

Példa komplex eseményre: Banki tranzakciónál megtörtént a jóváhagyás, de az utalás nem hajtódott végre egy adott időn belül.



2.2. ábra. Komplex esemény

Elemi esemény: Olyan esemény, amit már nem lehet, vagy nem érdemes tovább bontani. Például valamilyen érték megváltozott vagy üzleti folyamatoknál egy folyamat elindult.

2.2.2. Diagnosztika tervezése

Hatékony, jól működő diagnosztikához az adott üzleti folyamat és szakterület ismerete szükséges. Tudni kell, hogy mik a kritikus paraméterek a rendszerben, amiket érdemes figyelni, illetve hogy mi tekinthető az adott helyen megfelelő vagy hibás működésnek.

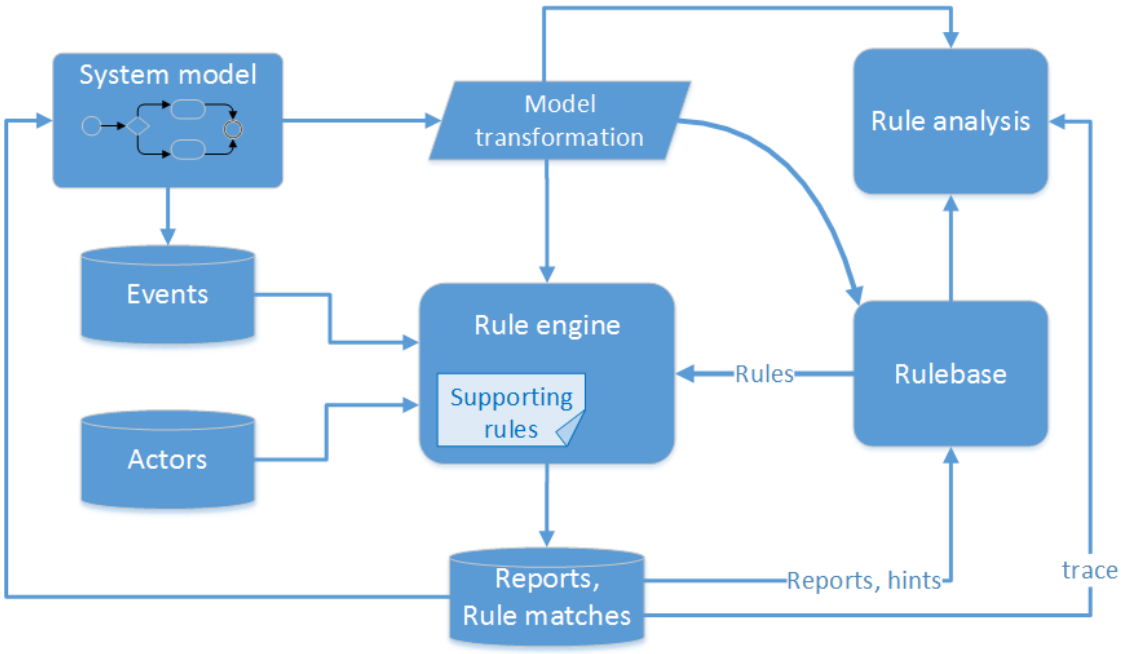
Rendszerünkben szakterület-specifikus szabályfelírást teszünk lehetővé, így az egyes folyamatlépésekre és azok változóira hivatkozhatunk. Egyébként ismerni kellene az üzleti elemzőnek a folyamatmodellező eszköz (esetünkben Bonita) belső működését, a futtatás során keletkező eseményeket, ami lényegesen bonyolultabb lenne. Már egy olyan kis példán (2.1. ábra), amit a 2.1.2 részben ismertettünk, körülbelül 10 esemény generálódik átlagosan egy lépéshez.

Diagnosztika támogatása

A diagnosztika minőségét alapvetően határozza meg a szakértői tudás, ezért a szakértő munkáját érdemes különböző módszerekkel támogatni.

Elsősorban a szakterületi elemző által felírt szabályok analízise lehet nagy segítség. Meg lehet mondani, hogy a szabályok között nincs ellentmondás, vagy azonos feltételekhez eltérő utasítás. Ha pedig rendelkezünk olyan információval, mely megadja, hogy mely események, mely paraméterek kritikusak, akkor automatikusan ellenőrizni lehet, hogy a felírt szabályhalmaz lefedi-e az összeset. Az ehhez szükséges információt szintén a szakértő adhatja meg, még a szabályok felírása előtt.

Az általunk megalkotott diagnosztikai környezet a 2.3. ábrán látható. A diagnosztikai környezet kialakításakor a szabályok könnyű megalkotását tartottuk szem előtt. A szabályok felírását úgy segítettük, hogy az elemző a folyamat fogalomkészletét használhassa. Ehhez a folyamatmodellből ki kell olvasni az abban megadott folyamatlépések, változók deklarációját. Az így kinyert szakterület-specifikus információkat át kell alakítani olyan formára, hogy azt a szabályvégrehajtó motor értelmezni tudja. Ennek leírása a 4.1.5. alfejezetben olvasható.



2.3. ábra. Diagnosztikai rendszer felépítése

A szabályok kiértékeléséhez szükséges információkat, eseményeket a folyamatokat végrehajtó motorból kell kinyerni (ld. 4.1.1. fejezet). A környezettel szemben elvárás, hogy a folyamatot végrehajtó motor más helyen lehessen, ezért ezeket az eseményeket hálózaton át küldjük el egy adatbázisba, ahol az eseményeket eltároljuk. Az tárolás módja a 4.1.2. fejezetben olvasható. Az események tárolására azért van szükség, mert a diagnosztikát off-line módon szeretnénk futtatni, bizonyos esetekben akár többször is egymás után. Ehhez azonban szükség van az események tárolására.

Az eltárolt események azonban még függetlenek a végrehajtott folyamattól, ezért ezeket át kell alakítani, hogy az üzleti elemző által könnyen értelmezhetőek legyenek. Ezt az átalakítást a szabályvégrehajtó motor végzi. A motor ezen kívül végez még statisztikai számításokat is, hogy a szabályok felírásakor rendelkezésre álljanak olyan adatok, mintaz egyes folyamatok átlagos lefutási ideje. A szabályvégrehajtó motor működésének bővebb leírása a 4.1.3. alfejezetben található.

Az üzleti elemző számára egy grafikus szabályszerkesztői felületet biztosítunk. Ennek segítségével egy webalkalmazásban állíthatja össze a szabályokat, ami segíti a szabályok helyes felírását. A szabályszerkesztő leírása a 4.1.4. alfejezetben olvasható.

A diagnosztika futtatása után az eredmények egy adatbázisban érhetőek el. A diagnosztika eredményeiből, a folyamatok lefutási eredményeiből további következtetéseket lehet levonni az eredeti folyamat hatékonyságáról, illetve a szabályok hasznosságáról. Ehhez a feltáró adatanalízis módszerét használtuk, erről a 4.2.2. alfejezetben írunk bővebben.

Ezen felül az üzleti elemző által felírt szabályok is ellenőrizhetőek. A szabályszerkesztőből való visszaolvasásuk után elemezhető azok helyessége, teljessége. Ennek vizsgálatáról a 4.2.1. alfejezetben írtunk bővebben.

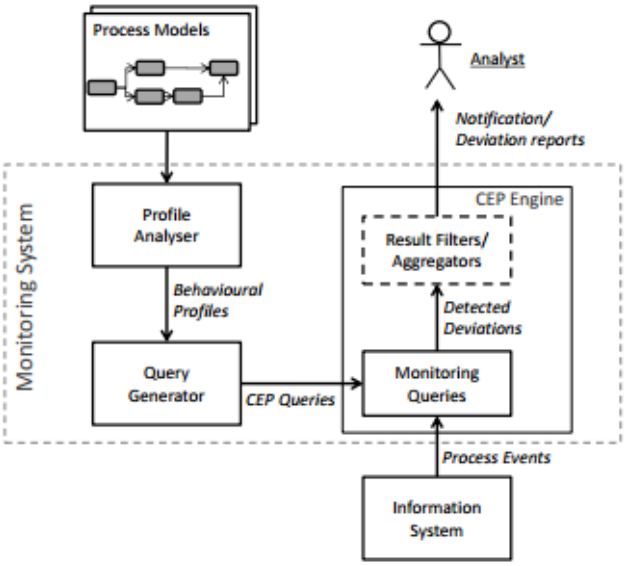
3. fejezet

Kapcsolódó munkák

A szabálykiértékelő rendszerek többnyire csak a szabályok számítását végzik el, az ehhez szükséges külső információk, ismeretek csak igen nehezen vihetőek be a rendszerbe. Az irodalomkutatás során olyan munkákat kerestünk, amelyek megpróbálják a szabályfelírást megkönnyíteni, vagy a szabályvégrehajtó környezetbe segítik a szükséges információk bevitelét. [12]

3.1. Szabálygenerálás

A [16] cikk olyan módszert mutat be, mely segítségével egy folyamatmodell leírásából olyan komplex eseményeket lehet előállítani, amikkel az adott folyamat szabályszerűsége, a modellre való illeszkedése ellenőrizhető. Csoportosítja, hogy egy folyamatmodellben milyen lehetséges szabályszegések fordulhatnak elő.



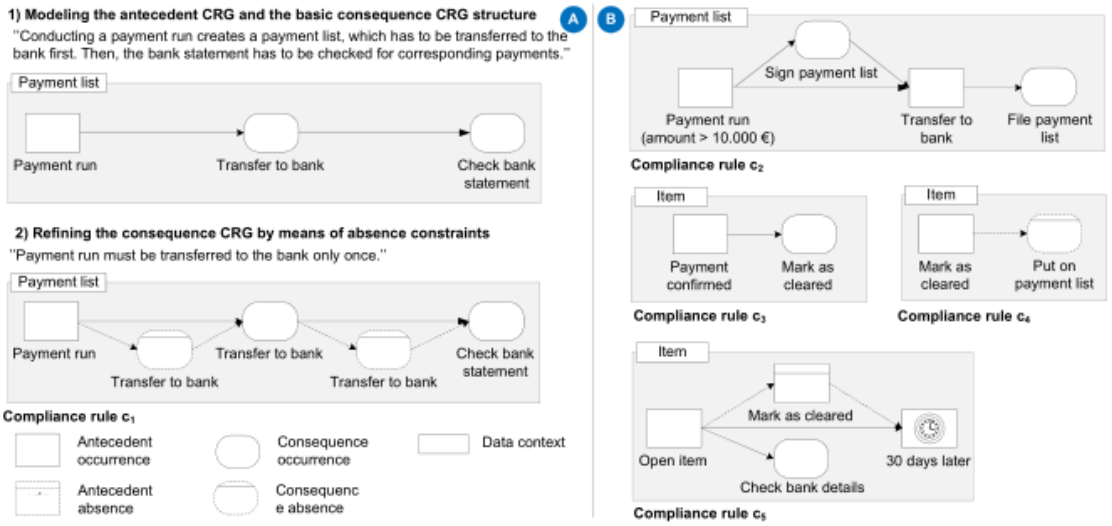
3.1. ábra. A [16] rendszer áttekintése

Az ismertetett rendszer áttekintése látható a 3.1. ábrán. Folyamatmodellből kinyert viselkedési modell alapján ellenőrző lekérdezések készülnek, amik lefutnak a folyamat eseményekre. A futások eredményéből szűrők által rendszerezett jelentések készülnek.

A megismert rendszer a folyamatmodellt sértő lefutások felismerésérét végzi el automatikusan. Mi elsősorban a rendszer magasabb szintű, az üzleti logikának az ellenőrzésével, annak támogatásával foglalkoztunk. Ehhez mindenképpen szükség van szakértői tudásra, de a modellező eszköz ellenőrzéséhez a cikkben bemutatott tudás hasznos lehet az eszközünk továbbfejlesztésében. Segítségével automatikusan előre definiálhatnánk szabályokat a konkrét folyamatmodellekre, amikkel dolgozunk.

3.2. Grafikus szabályok, hibaok felderítés

A [20] munka szerzői definiáltak egy Compliance Rule Graph (CRG) nyevlet az SeaFlows projektben, melynek segítségével folyamatok megfelelőségét vizsgálják. A nyelv segítségével különböző szabályokat (Compliance rules) lehet definiálni és a bejövő események alapján figyeli a rendszer, hogy a feltételek sérülnek, vagy sérülhetnek-e. Segíti a hiba gyökerének a megtalálását (Root cause identification) és a szabály megszegésének megelőzését (prevention of violation).
Ez a mi megoldásunkkal ellentétben valós idejű ellenőrzés.



3.2. ábra. Kényszer szerkesztése

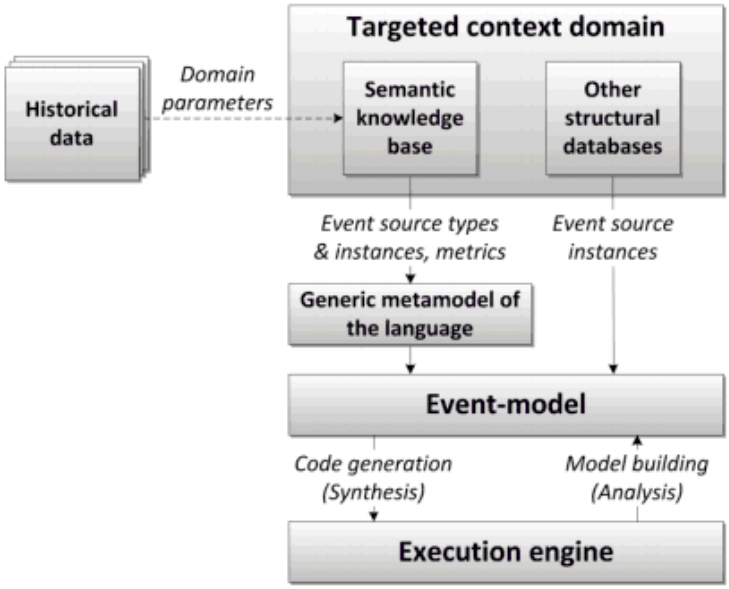
A SeaFlows Compliance Monitor integrálva van az AristaFlow BPM Suit-ba. A Compliance Rule Editor segítségével parametrizált megfelelőségi szabálymintákat lehet modellezni.

Ellentétben a mi rendszerünkkel nem támogatják a domainspecifikus elemeken való szabályszerkesztést és maguknak a szabályoknak az ellenőrzését. Ugyanakkor egy hasznos módszert mutat a hiba gyökerének megtalálására, ami számunkra egy jó továbbfejlesztési irány lehet.

Az ismertetett eszköz gráfok segítségével, tehát grafikusan fogalmazza meg kényszereit. Mi a szöveges szabálydefiníciót használtuk, amely komplex szabályok esetén tömörebb definíciós lehetőséget nyújt.

3.3. A szabályrendszer felírásának általánosítása

A [13] cikkben definiáltak egy komplex esemény leíró nyelvet (CEDL), amely egy egyszerűbb leírásmódot mutat komplex események definiálására. A nyelvhez implementáltak egy Eclipse alapú fejlesztőkörnyezetet is, amely segíti az eseményminták hatékony fejlesztését. Készítettek egy eszközt szemantikus tudásbázis importálására, amely képes egy ontológiát beolvasva létrehozni egy saját domainmodellt, amely tartalmazza az eseményforrásokat illetve azok mérhető értékeit, ezzel segítve a szakterület hozzákötését a szabálykiértékelő rendszerhez. A 3.3. ábrán látható a környezet logikai felépítése.



3.3. ábra. A CEDL környezet felépítése

A környezetet felhasználva a rendszerünket függetleníthetnénk a Drools szabálykiértékelőtől, hiszen a CEDL képes futtatható szabályok előállítására, sőt, egy domainmodellből elő tudja állítani az eseményobjektumokat is, így a szabálykiértékelő motor használatára egy teljesen generikus megoldást nyújt. Jelenleg a CEDL környezet az Esper szabálykiértékelő motor nyelvére képes szabályok előállítására, de ez tovább bővíthető.

3.4. Valós idejű diagnosztika

A [14] disszertáció bemutatja, mennyiben más egy valós idejű diagnosztika. A [18] cikk bemutatja, hogy hogyan lehet különféle eseménymodellű folyamatok összekapcsolását megoldani interfészek definiálásával. Ez más szabálymotor bekötésekor hasznos lehet.

4. fejezet

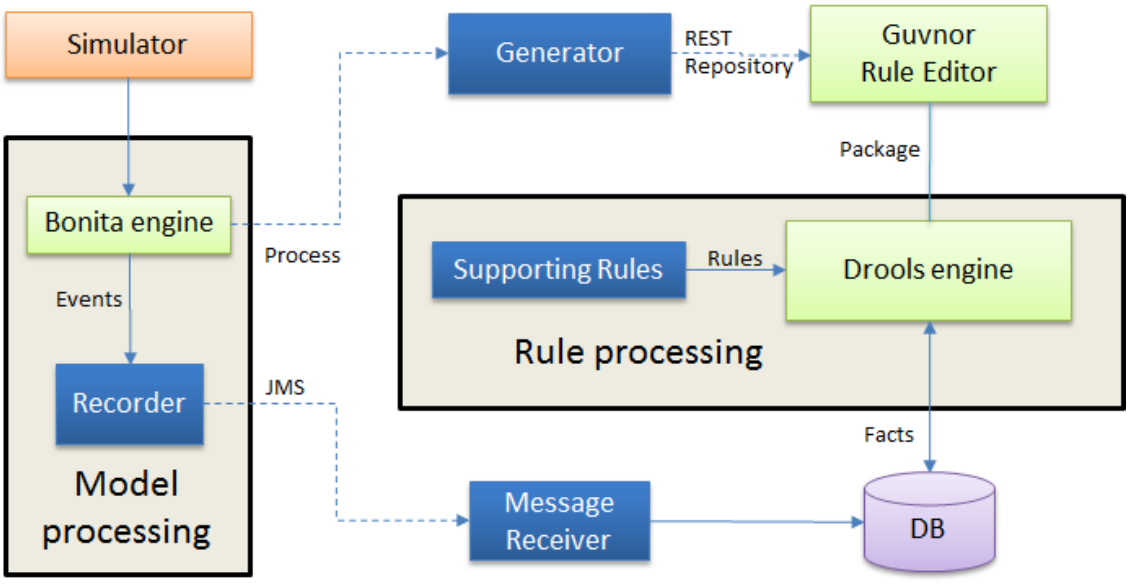
Megvalósítás

Ebben a fejezetben a dolgozat keretein belül elkészült diagnosztikai környezet, és az ahhoz tartozó kiértékelő rendszer megvalósítására, annak technikai részleteire térnénk ki.

4.1. Diagnosztikai környezet

A diagnosztikai környezetet egy ipari projekt keretein belül valósítottuk meg, így több megkötéssel dolgoztunk.

A diagnosztikai környezet számos kisebb komponensből áll. A környezet felépítése a 4.1. ábrán látható.



4.1. ábra. A diagnosztikai környezet felépítése

A diagnosztikai környezetben szereplő komponensek:

Folyamatmotor Az üzleti folyamatot végrehajtó eszköz.

Bonita A Bonita az általunk használt folyamatmotor. Egy modell alapú üzleti folyamat tervező eszköz. Egy BPMN-hez hasonló formalizmussal megadható benne a megvalósítandó üzleti folyamat, ebből képes a folyamatot előállítani.

Recorder Egy a Bonitába beépülő komponens, ami elkapja az eseményeket.

Simulator A szimulátor komponens tesztelési célokat szolgál. Automatizáltan képes a folyamatok végrehajtására.

Generator A generátor komponens a Bonita folyamatleíróiból állít elő egy az eseményeket leíró csomagot a szabályszerkesztő számára.

Szabálymotor A szabályokat végrehajtó, kiértékelő motor.

Drools engine A szabályokat kiértékelő motor.

Supporting Rules A statisztikákat számító szabályok.

Message Receiver A folyamat végrehajtásáról érkező eseményeket fogadó komponens.

DB Az eseményeket és a kiértékelés eredményeit tartalmazó adatbázis.

Guvnor Rule Editor Az üzleti elemző által használható szabályszerkesztő.

A komponensek részletes leírása a következő alfejezetekben található.

4.1.1. Folyamatmotor

A folyamatmotort az ipari partner határozta meg, így a Bonitát használtuk. A Bonita egy egyszerűen használható üzleti folyamattervező eszköz. A folyamatokat egy a BPMN formalizmusát követő szerkesztőben lehet létrehozni. Az itt létrehozott folyamatok lépéseihez könnyen hozzáadhatóak kapcsolatok, mint például egy adatbázis elérés, vagy e-mail küldés, de akár egy általunk megírt tetszőleges funkció is.

A Bonita végrehajtó motorja többféle API-val rendelkezik. A folyamatok végrehajthatóak egy testreszabható webfelületen, vagy programozottan REST vagy EJB felületen át. Ezen felül a végrehajtó motort használhatjuk egyszerű programkönytárként is, ilyenkor helyi hívásokkal lehet irányítani a benne futó folyamatokat. Ezt használtuk ki a szimulátor komponens megírásakor.

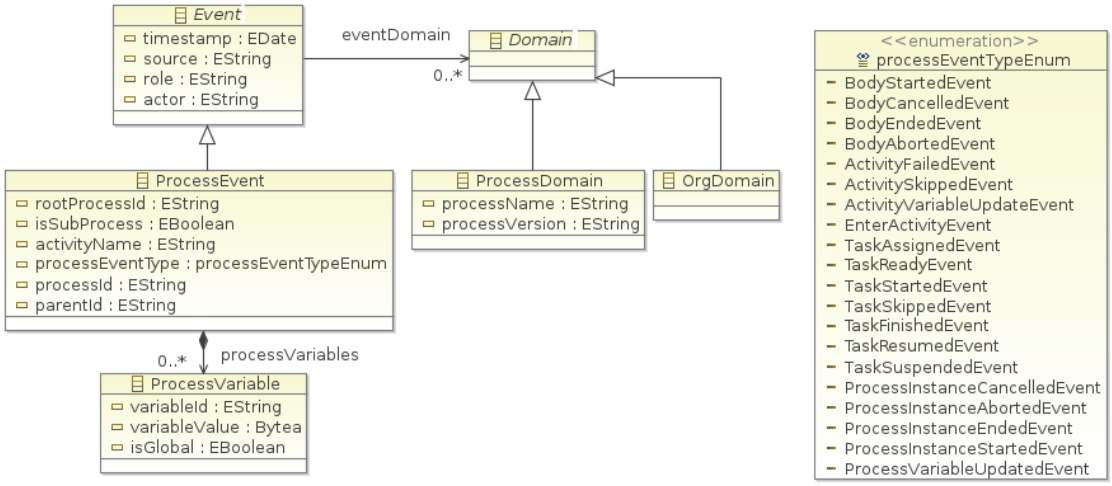
A folyamatmotor könnyen kiegészíthető, felműszerezhető. Ezt segíti egy recorder komponens, amit megvalósítva értesítéseket tud küldeni a motor a futó folyamatok állapotáról. Egy ilyen komponens megvalósításával készítettük fel a motort a diagnosztikára. A komponens minden bekövetkező eseményt feldolgoz, valamint kigyűjti a folyamatmotorból az eseményhez tartozó változók értékét, egyéb a folyamathoz tartozó metaadatokat, majd az így keletkezett eseményobjektumot elküldi a diagnosztika számára.

A rendszerrel szemben elvárás, hogy a folyamatmotor másik környezetben fut, mint a diagnosztika, ezért az eseményeket hálózaton át, JMS (Java Message Service) használatával küldjük át egy másik szerveren futó komponens számára, ami feldolgozza, tárolja azokat, hogy majd a lefuttatandó diagnosztika belolvashassa azokat.

Az események gyűjtése szükség esetén távolról kikapcsolható, ha esetleg bizonyos időre le szeretnénk állítani a diagnosztikát, például túlterhelődés miatt. Ezen felül szűrhető is az események feldolgozása, folyamatonként ki lehet kapcsolni, így adott esetben, ha van egy alfolyamat, ami üzleti szempontból érdektelen, akkor ki lehet zárni a diagnosztikából, ne terhelje felesleges eseményekkel azt.

4.1.2. Eseményfeldolgozó

Az eseményfeldolgozó komponens fogadja a recorder által küldött eseményeket, azokat feldolgozza. A recorder minden eseményt elküld a folyamatmotorból szűrés, feldolgozás nélkül, hiszen minden a recorderben végzett művelet lassítja a folyamatok futását. Így minden feldolgozást a fogadó komponens végez. Az érkezett eseményeket átalakítja egy általános, a Bonitától nagyrészt független eseményformára, így esetleges leváltása során az e felett futó komponensen már nem kell változtatni. Az itt keletkező események már csak a ténylegesen szükséges eseményeket tartalmazzák. A 4.2. ábrán látható esemény-



4.2. ábra. Az eseménymodell

modellt úgy alkottuk meg, hogy más forrásból származó események is könnyedén belefoglalhatók lehessenek. Így a diagnosztikába könnyen felvehetünk olyan eseményeket is, amik nem a folyamatmotorból jönnek, hanem például az alatta futó adatbáziskezelőből. Ilyen események segítségével olyan esetekre is írhatunk fel szabályokat, amelyek a folyamat alatt futó adatbázisban keletkeznek, például egy hiányzó idegen kulcs. Így megfigyelhetjük a hiba valódi okát is, nem csak a hibajelenséget, miszerint a folyamat vagy egyáltalán nem, vagy hibásan fut le.

Az eseménymodell lefedi a Bonita folyamatainak teljes életciklusát, ami viszont túl sok információt tartalmaz az üzleti elemzőnek. Ezért ezen a modellen még további egyszerűsítést végzünk. A szabályszerkesztő számára előállítandó eseménymodell azonban így is lefedi az elemző számára érdekes összes állapotot.

4.1.3. Szabályvégrehajtó motor

A szabályvégrehajtó motor által elvégzett feladatok:

- Események fact-ekké alakítása:
  Alapszintű eseményfeldolgozást végeztünk. Adott folyamathoz vagy folyamatlépéshez tartozó különböző eseményekből (például elindult, változó értéke módosult, befejeződött, vagy sikertelenül ért véget) egyetlen objektumot állítunk elő. Ez tartalmazza a folyamat vagy folyamatlépés azonosítóit (folyamat típus, folyamat példány, folyamatot előállító eszköz azonosítója és folyamatlépés esetén a neve), összes változóját a legkésőbb beállított értékükkel, a kezdés és befejezés időpontját, valamint egyéb információt a lefutásával kapcsolatban. Folyamatlépés esetén lehet sikeres, hibával végződő, kihagyott vagy későn végrehajtott. Ez utóbbi eldöntéséhez a szakértőnek kell megadnia az elvárt lefutási időt.
  Erre azért van szükség, hogy ténylegesen a folyamatra és annak elemeire lehessen hivatkozni a szabályok megfogalmazásakor és ne kelljen ismerni az adott folyamatmodellező eszköz belső eseményeit.

- Statisztikai adatok számítása:
  Folyamat- és folyamatlépés-típusonként feljegyezzük a következőket: a leghosszabb és az átlagos lefutási időt, a sikeres, a sikertelen, a késedelmes végrehajtások, a kihagyott lépések és az összes végrehajtás számát, valamint a sikeres és nem sikeres lefutások arányát. Ezek az adatok önmagukban is hasznosak a diagnosztika során, de ezen túl így lehetővé válik például az átlagos lefutásnál hosszabb lépésekre szabályt felírni.

- Üzleti elemzők által írt szabályok kiértékelése:
  A végrehajtó motor importálja a szabályszerkesztőből az elkészült szabályokat és a lentebb ismertetett Drools engine elvégzi a szabálykiértékelést.

- Szabályok illeszkedésének loggolása:
  Minden üzleti elemzők által felírt szabály elsülését rögzítjük. Egyrészt tároljuk minden egyes szabályilleszkedéshez annak időpontját, a szabály nevét, a folyamat példány azonosítóját, a mintában szereplő lépések nevét, végrehajtóik nevét és lefutási idejüket. Másrészt loggoljuk szabályonként a hozzájuk tartozó futások számát és hogy az egyes folyamat lépések hányszor illeszkedtek.
  Az elkészült CSV fájlokat a feltáró adatanalízishez használjuk fel, melynek működését a 4.2.2 alfejezetben ismertetünk részletesen.

A fenti felsorolás eltekintve egyben végrehajtási sorrend is. Először feldolgozzuk az eseményeket, aztán az előállt objektumokon statisztikai vizsgálatokat végzünk, majd kiértékeljük az üzleti elemzők által írt szabályokat, melyekről illeszkedés esetén naplóbejegyzést készítünk.

Inkrementalitás A szabályvégrehajtó részlegesen inkrementálisan működik. Amennyiben új események érkeznek az adatbázisba, akkor a korábban már feldolgozott eseményeket nem alakítja át újra fact-ekké, hanem az előzőleg eltároltakat használja a további

műveletekhez. A statisztikák számításánál amit lehet, nem végez el többször. (A folyamatlépésekre típusonként az átlagon felüli lefutások számának megadásánál nem tudtuk az újbóli kiértékelést elkerülni.)

Az inkrementális működés hatékonyságát kezdeti mérések igazolták, jelentős teljesítménynövekedést értünk el. Ennek részletes vizsgálata folyamatban van, de a kellő mennyiségű, precíz mérések elvégzésére a dolgozat kereteiben már nem volt lehetőség.

Drools A szabályok kiértékeléséhez a Drools üzleti szabálymenedzselő rendszert (Business Rule Management System - BRMS) használtuk, mely egy előrefelé következtetésen alapuló szabálymotor segítségével találja meg a szabályilleszkedéseket.

Azért választottuk a Drools-t, mert nyílt forráskódú, létezik hozzá grafikus szabályszerkesztő, illetve Drools-szal már voltak korábbi tapasztalataink.

Drools szabály A Drools szabály alapvetően két részből áll; egy feltételeket megfogalmazó (WHEN) és egy JAVA utasításokat leíró (THEN) ágból. Hogyha a feltételek teljesülnek, akkor lefutnak az utasítások. Ezenkívül meg lehet adni metainformációkat, mint például a szabály salience értéke, ahol a nagyobb értékkel rendelkezőt hamarabb értékel ki a Drools motor.

A következő egyszerű szabály ha olvasatlan üzenetet talál, akkor azt kiírja a konzolra:

```
rule "PrintMessage"
  when
    Message(status = "unread", $m:message)
  then
    System.out.println($m);
end
```

A when ágban egy Message objektum status attribútumát vizsgáljuk, illetve elnevezzük a message attribútumát ($m), hogy arra hivatkozhassunk. Ha a status értéke "unread", akkor le fog futni a then ágban lévő JAVA utasítás, mely kiírja a konzolra a $m változót, vagyis az üzenetet.

4.1.4. Szabályszerkesztő

A szabályok szerkesztéséhez a Guvnor webes felületű eszközt használtuk, mely lehetővé teszi Drools szablyok felírását kevesebb informatikai ismeretel rendelkező üzleti elemzők számára is. Ezt egy grafikus szabályszerkesztő felülettel valósítja meg, így nem kell közvetlenül Drools kódot írni, nincs szükség a nyelv mélyebb ismeretére.

A Guvnor használata mellett szólt az előbb említett egyszerűbb szabályszerkesztés, és hogy más ilyen jellegű Drools-hoz kapcsolódó eszköz nem létezik. Leszámítva az Uberfire nevűt, mely még kísérleti, fejlesztési stádiumban van, ezért ezzel nem foglalkoztunk.

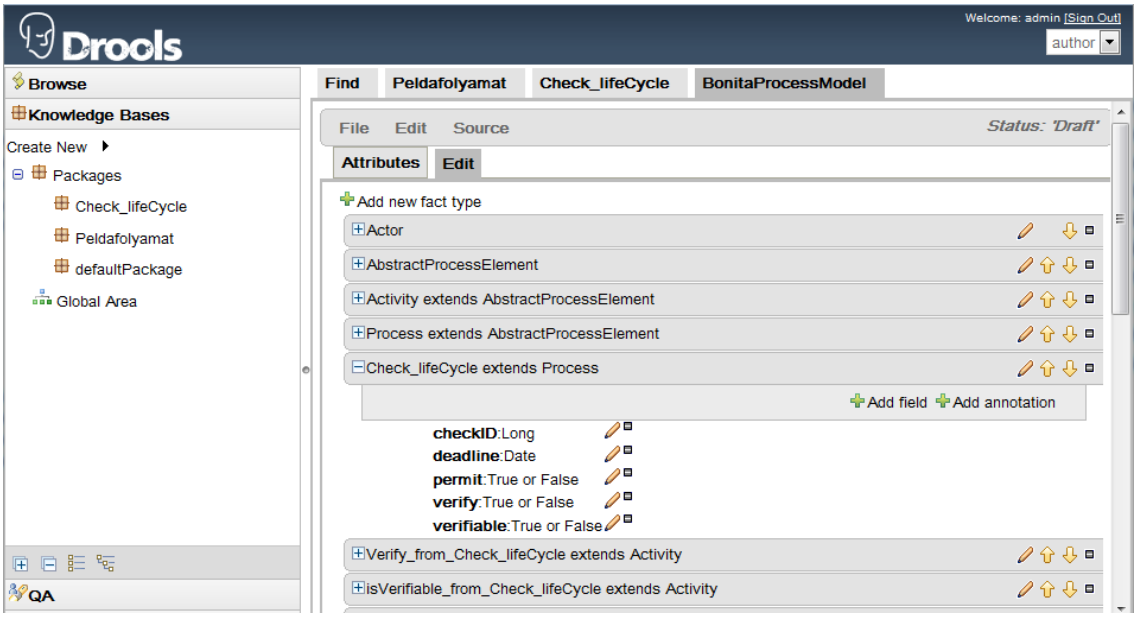
Modell A szabályok felírása előtt szükség van egy modellre, amire a feltételeket felírhatjuk. Esetünkben ez a Bonitából származó folyamatok modellje, amit Drools szintaktikára alakítunk. (Alapvetően a Guvnor is a Drools felírási módját alkalmazza.) A definíció

a declare kulcsszóval indul, ami után megadjuk az osztály (fact type) nevét. A következő sorokban pedig az attribútumokat és kettősponttal elválasztva azok típusait adhatjuk meg. Az end kulcsszóval zárul a felírás. Az alábbi példa egy folyamat definícióját szemlélteti:

```
declare Process
  processInstanceID: String
  processTypeID: String
  startTime: Date
  finishTime: Date
end
```

A Process nevű fact-ünk tehát négy attribútumot tartalmaz, kettő szöveges és kettő dátum típusút.

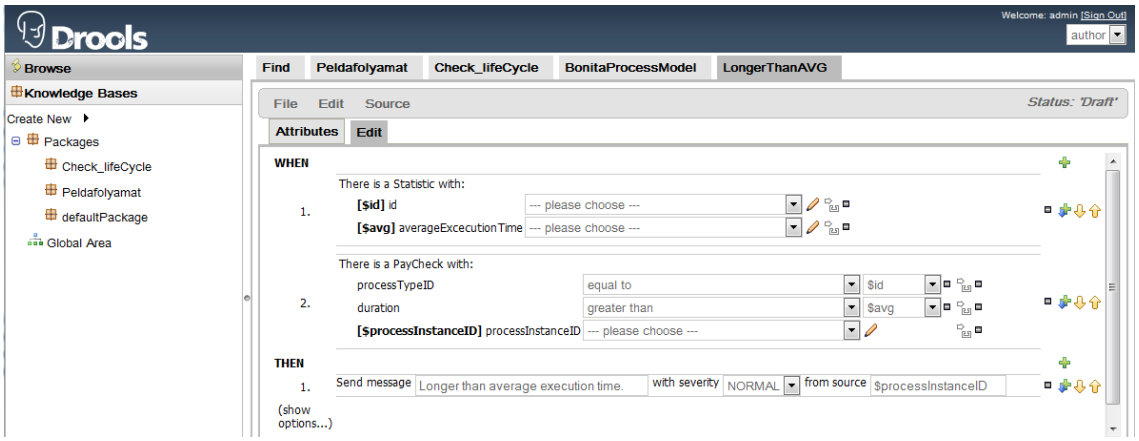
A Guvnor felületen ugyanakkor ennél picit egyszerűbb az új fact felvétele. A kezdő és lezáró kulcsszavakkal nem kell törődnünk, és az elérhető típusokat is felajánlja az eszköz. A 4.3 ábrán példaként látható az általunk használt üzleti folyamat Guvnor modellje.



4.3. ábra. Általunk előállított modell Guvnorban

Szabályfelírás Szabályok összeállításánál a fact típusokat egy listából kiválaszthatjuk a when feltételünkhöz. A kiválasztottakra kattintva pedig azok attribúmait is kiválaszthatjuk, valamint a lehetséges kényszereket sem kell gépelni, azokat is felajánlja az eszköz. A then ágban is választani lehet az egyes objektummódosító műveletek között, de Drools nyelven is lehetőség van utasítások felírására.

A 4.4. ábra egy példát mutat Guvnorban felírt szabályra, mely azokról a PayCheck nevű folyamatokról, amik hosszabb lefutásúak, mint az átlag egy bejegyzést készít. A then ágban egy általunk írt DSL template található, ami egy Message objektumot hoz létre a megadott üzenettel, fontossággal (severity) és forrással. Az ilyen üzenet objektumokat pedig adatbázisba rögzíti a rendszerünk.

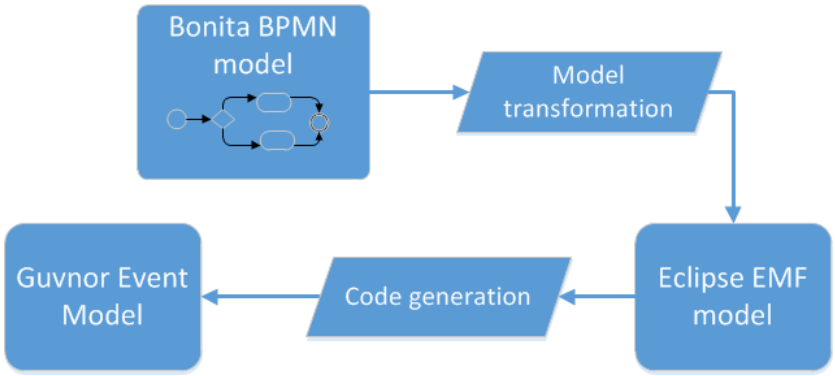


4.4. ábra. Szabály Guvnorban

4.1.5. Generátor

A leképezés

A környezetben a generátor végzi a leképzést az általános folyamatmodell, és a szakterület specifikus környezet között. A leképzés teljes menete a 4.5. ábrán látható.

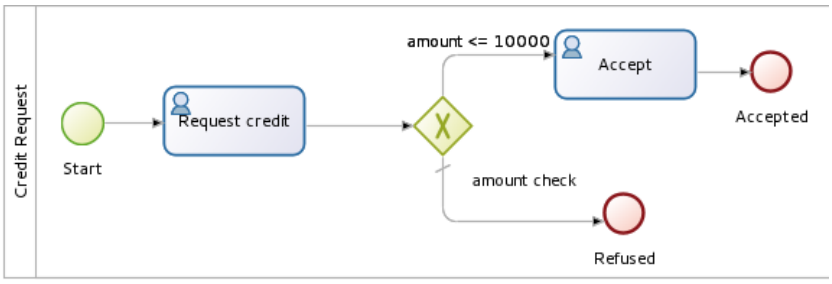


4.5. ábra. A folyamatmodell leképzése

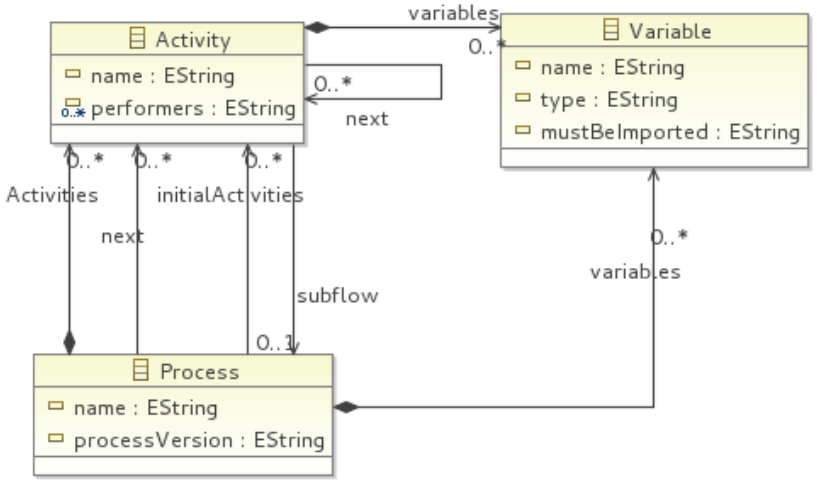
A szakterületspecifikus környezet létrehozásához szükség volt a Bonita eseményeinek leképzésére a szabályszerkesztő által ismert eseménymodellre. Ezt az eseménymodellt úgy terveztük meg, hogy az üzleti elemző a lehető legkönnyebben hozhasson létre új szabályokat.

A modell leképezéséhez kezdeti lépéseként fel kellett dolgozni a Bonita folyamatait, hogy abból kinyerjük a szükséges ismereteket. A Bonita a 4.6. ábrán egy egyszerű példán látható BPMN modellben tárolja, kezeli a folyamatokat.

A modellek életciklusa meglehetősen összetett, az üzleti elemző számára ez túl komplex lehet, illetve feleslegesen nagy mennyiségű információt hordoz. Ezért ezt a folyamatmodellt egy lényegesen egyszerűbb modellre képeztük le, ami már csak az elemző számára szükséges információkat tartalmazza.



4.6. ábra. Hitelkérelem



4.7. ábra. A folyamatmodell

A 4.7. ábrán látható az általunk elkészített folyamatmodell. Az üzleti folyamatok lehető legegyszerűbb ábrázolását választottuk, ugyanis egy üzleti elemző nem kezel többet a folyamatokból, csak az aktivitásokat, azok változóit. A folyamatok ezen kívül kapcsolóelemekből, végpontokból állnak. Ezek azonban felfoghatóak speciális aktivitásként, hiszen viselkedésük azokéval megegyezik, csak a végrehajtási útvonalat módosítják. Így együtt kezeljük az aktivitásokkal. Ennek a megközelítésnek további előnye az, hogy így az elemző képes lesz felírni olyan szabályokat is, amik a munkadarab a kapcsolón vagy végponton való áthaladásának pillanatában érvényes tulajdonságaira vonatkozik.

A modell leképzésére megvalósítottunk egy eszközt, amely képes a Bonita folyamatokat automatizáltan leképezni erre a modellre. Viszont szükségesnek találtuk az eszköz Bonita-függetlenségét megőrizni, elősegíteni. Ezért elkészítettünk egy XTEXT [11] alapú nyelvet, amellyel egyszerű szöveges formában is leírható a folyamat felépítése.

Az XTEXT egy Eclipse alapú környezet, amely használatával szakterületspecifikus nyelveket készíthetünk a nyelv nyelvtanának definiálásával. Az XTEXT környezet képes ebből a leírásból egy fejlesztőfelületet biztosítani a nyelvnek, amely futás közben a szerkesztett fájlt feldolgozva egy a nyelvnek megfelelő objektummodellt képes felépíteni. Ezt a felépült objektummodellt tetszőlegesen felhasználhatjuk. A mi esetünkben ez a modell az előbb ismertetett folyamatmodell.

Ezzel megoldottuk, hogy a Bonita esetleges leváltása esetén használható ez a nyelv a

folyamat megadására, nem feltétlenül szükséges egy másik hasonló leképezést megvalósító komponens megírása a diagnosztika használatához. Az előzőekben mutatott hitelkérelmi folyamatnak egy részlete az általunk definiált folyamatleíró nyelven:

```
Process Credit_Request {
  initialActivities ( Start )
  variables {
    @IMPORTANT  //Annotation for completeness query
    Variable amount {type "java.lang.Long"}
  }
  Activities {
    ... //Start activity
    @CRITICAL
    Activity Request_credit {
      next ( amount_check )
    },
    ... //amount check, Accept, Accepted, Refused activities
  }
}
```

Látható, hogy a leírt modell is igen egyszerű, könnyen kezelhető. Így egy esetleges nem Bonitából származó folyamatmodell felírása is egyszerű. A példán láthatunk egy annotációt is, amelyet a szabályok teljességének vizsgálatkor használunk fel.

Az XTEXT-ben definiált folyamatokból EMF (Eclipse Modelling Framework) [5] objektummodell készül. Ezt a modellt felhasználva egy kódgenerátorral állítjuk elő a szabályszerkesztő által használható eseménymodellt. Ez a modell már egy szakterületspecifikus megfelelője az előbb leírt folyamatmodellnek. Ezzel a leképzéssel is az üzleti elemző munkáját segítjük, hiszen így már közvetlenül hivatkozhat a folyamatokra, azok változóira.

Az EMF egy modellező környezet, amelyre igen sok eszköz épül, köztük az XTEXT is. Az XTEXT által előállított EMF objektummodellt bejárva egy XTEND generátor segítségével állítjuk elő a Drools eseménymodellt.

A mintában szereplő folyamat egy a Drools nyelvén felírt rövid darabja:

```
declare Credit_Request extends Process
  amount: java.lang.Long
end
... //More activities
declare Request_credit_from_Credit_Request extends Activity
  amount: java.lang.Long
end
... //More activities
```

Látható, hogy az eseménymodell már csak a szakterületspecifikus fogalmakkal dolgozik. Az aktivitásoknak (Activity) és a folyamatoknak (Process) ősosztályokat definiáltunk, amelyekben a szakterülettől független információkat tároljuk. Ilyen információk például az adott folyamatpéldány végrehajtásának kezdete, annak sikeressége, vagy az adott aktivitást végrehajtó felhasználó.

Így a leképzés során szétválasztottuk a szakterület független értékeket a szakterületspecifikustól, így lehetővé téve az általános szabályok felírását is. A szakterületspecifikus események segítségével pedig könnyen felírhatóak a specifikus szabályok. Látható továb-

bá, hogy a folyamatszintű változók („amount”) megjelennek az aktivitásoknál is, így szabályfelírás közben közvetlenül tudunk hivatkozni a folyamatok változóira is. A leképezés hasznosságát egy egyszerű példán be is mutatjuk.

A szabályt arra az esetre írtuk fel, ha egy 1000 feletti hitelkérelmet nyújtanak be. Ez a leképzés nélkül így nézne volna ki:

```
WHEN
  ProcessInstanceEnded(ProcessId = 'Credit request'
    AND Variables.get('amount') > 1000)
THEN
  --Send warning..
```

A leképzéssel lényegesen egyszerűsödik:

```
WHEN
  Credit_request(amount > 1000)
THEN
  --Send warning..
```

További előny, hogy az így leképzett eseményeknél már nem egy értékként kell hivatkozni az azonosítókra, hanem az események neve azonosítja a folyamatokat, lépéseket. Ez egyrészt csökkenti a hibák lehetőségét, mivel már a szabályok fordításakor kiderül, hogy hibás azonosítót adott meg az elemző, nem csak futási időben látszik annyiból, hogy egyszer sem sül el a szabály. Valamint ha az üzleti elemző a grafikus szabályszerkesztőt használja, ami javasolt, akkor a szerkesztőben ilyen, létező objektumokra tud hivatkozni.

Ezen felül az így keletkező eseményobjektumokhoz további számított értékeket is kapcsoltunk, mint a lefutás hossza, sikeressége. Ezzel azt értük el, hogy az üzleti elemzőnek nem kell két objektumot bonyolultan összekapcsolnia.
A számított értékek nélkül egy folyamat lefutási idejére vonatkozó szabály így néz ki:

```
WHEN
  ProcessInstanceStarted(ProcessId = 'Credit request'
      AND start:TimeStamp AND instance:InstanceId)
  AND
  ProcessInstanceEnded(ProcessId = 'Credit request'
      AND end:TimeStamp AND InstanceId = instance)
  AND
  (end - start) > 2 hours
THEN
  --Send warning..
```

A számított értékek segítségével pedig lényegesen egyszerűbb:

```
WHEN
  Credit_request(duration > 2 hours)
THEN
  --Send warning..
```

Ebben az esetben viszont a számított értékek nélküli kód még hiányos. Ugyanis ha egy folyamat nem sikerrel ért véget, akkor nem lesz „ProcessInstanceEnded” esemény, így nem tudjuk meg, ha egy esetlegesen hibával zárult folyamatnak volt túl lassú a végrehajtása. Így az összes ilyen esethez fel kellene venni egy ágat a szabályban. A számított értékek azonban ezt is megoldják, hiszen ezt az attribútumot a folyamat kezdete és annak bármilyen terminálása közt számítjuk.

Egyéb segédobjektumok készítése

A generátor más objektumokat is előállít a szabályszerkesztő számára, nem csak a folyamatspecifikus eseménymodellt. Készít a szabályszerkesztés megkönnyítésére objektumokat:

Enumerációk: segítségükkel a szövegkonstansokat nem kell minden alkalommal beírni, hanem egy legördülő menüből választhatóak,

Riasztás minta: Egy sablonszerű minta a riasztások létrehozására. Az üzleti elemző ezt kitöltve küldhet riasztásokat a szabályok elsülésekor.

Kiegészítő információkat tartalmazó objektumokat:

Statisztika: Egy olyan objektum, amely segítségével az adott folyamathoz vagy aktivitáshoz érhetőek el statisztikák, mint például egy folyamat átlagos lefutási ideje. A statisztikákat bővebben a 4.1.3, fejezetben írjuk le.

Actor: A felhasználók egyszerű modellje. Segítségével a szabály felírásakor nem csak az aktivitás végrehajtójára, hanem annak főnökeire is hivatkozhatunk.

Készítünk kezdeti szabályokat is:

Fail: Egy egyszerű mintaszabály, amely minden hibával zárult folyamat esetén riaszt. Kiindulási alapul szolgálhat komplex szabályok felírásához.

maxDurations: Egy kezdeti inicializáló szabály. Értékeinek beállításával jelezhetjük a szabályvégrehajtó motornak, mikor számítson egy szabályt elkésettnek.

Ezen segédobjektumok, a számított értékek nélkül egyes szabályoknál igen bonyolultan kellene megírni az értékek átalakítását, aggregációját. Egyes segédobjektumok nélkül pedig nem lehetne felírni olyan szabályokat, mint az összes végrehajtás átlagánál hosszabb lefutások elemzése, hiszen ehhez nincs meg minden információ a szabály kiértékelésekor.

Feltöltés

A generátor alapvetően Eclipse kiterjesztésként készült el. A környezetet felkészítettük az előállított erőforrások automatikus telepítésére. Így az előállított objektumok a szerkesztő REST felületére feltölthetőek.

4.1.6. Szimulátor

A szimulátor a környezet tesztelését segíti, feladata a folyamatmotor automatizált meghajtása. Nem célszerű egy éles webszerveren futó folyamatmotoron tesztelni a környezetet, ezért a szimulátort úgy oldottuk meg, hogy ne kelljen egy Bonitához csatlakoznia, hanem egy saját Bonitát indít el stand-alone módban. Így a Bonita a szimulátoron belül fut, nincs hatása az esetlegesen már élesen futó rendszerre. Azonban a stand-alone módban futó Bonitába ugyanúgy betölthető a recorder komponens, mint egy éles motorba. Ezen felül

mivel egy valódi Bonitát futtat, nem csak az eseményeket állítjuk elő, ezért maga az üzleti folyamat, annak stabilitása is tesztelhető segítségével.

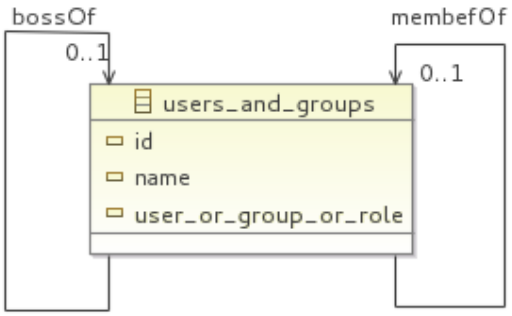
Célunk volt, hogy a szimulátor a lehető legtöbb esetet lefedje. Ezért a szimulátor képes bármilyen Bonitában felírt folyamat lefuttatására. Minden futáshoz, aktivitás végrehajtáshoz tetszőleges változóértékeket állíthatunk. Az aktivitások egymás után, akár időzítve is lefuttathatóak, sőt akár összefésülve is futtathatók a folyamatok. A végrehajtás során minden lépésben megadható az azt végrehajtó felhasználó is. Így a lehetséges futások nagy része szimulálható segítségével.

A lefutási mintákat egy előre összeállított CSV-állományból olvassuk be, és ennek megfelelően hajtja végre a szimulátor a folyamatot. A szimuláció során a stand-alone Bonita ugyanúgy, mint egy éles rendszer-beli, meghajtja a recordert a teszteseteknek megfelelően. Így hatékonyan tesztelhető az egész diagnosztikai környezet, valamint a Bonitában megírt folyamatok.

4.1.7. Felhasználókezelés

Az üzleti folyamatok kezelése során fontos szerepet játszik a felhasználókezelés, jogosultságellenőrzés. Ezért a folyamatmotorból gyűjtünk információkat a folyamatokat végrehajtó felhasználókról is, így lehetőség adódik a felhasználók alapján is szabályok felírására. A gyakorlatban azonban szükség lehet ennél többre is. Szükséges a csoportok, szerepek és a köztük fennálló relációk kezelése is. Ezért létrehoztunk egy modellt a felhasználók szerepeinek tárolására, melynek segítségével be tudjuk vinni ezeket az információkat is a szabályvégrehajtó motorba. Ezáltal az üzleti elemző fel tud írni olyan szabályokat, amellyel ellenőrizni tudja, hogy az adott lépést arra jogosult személy hajtotta-e végre. Ezt többnyire a folyamatot végrehajtó motor is tudja ellenőrizni, azonban az olyan komplex szabályokat mint, hogy ne lehessen egy folyamat két lépésének ugyan az a végrehajtója, már nem képes ellenőrizni. Ez olyan esetekben érdekes, ha például olyan szabályt akarunk felírni, hogy saját magának nem hagyhat jóvá egy szerződést az egyik csoport.

A felhasználók modelljét egy relációs adatbázisban tároltuk. Ennek sémája a 4.8. ábrán látható. A séma képes tárolni a felhasználók alapvető adatait, mint nevét, felhaszná-



4.8. ábra. A felhasználókat tartalmazó séma

lónevet, valamint csoporttagságait, főnökeit, szerepeit.

4.2. Szabályok és lefutásaik elemzése

4.2.1. Statikus elemzés

A szabályok statikus elemzése során a szabályok teljességének és helyességének vizsgálatát értjük. Mivel a diagnosztika eredményessége a szabályok minőségén múlik, ezért kell vizsgálnunk őket.

A szabályok vizsgálata modell alapon történik. Elkészítettünk egy eszközt, amely képes a szabályszerkesztőben felírt szabályok visszaolvasására, azokból egy EMF modellt épít fel. Az EMF (Eclipse Modelling Framework)[5] egy széles körben elterjedt Eclipse alapú modellező környezet, amelyre igen sok környezet épít így az elkészített modellek sokfelé felhasználhatóak.

Az így elkészített modelleken végzett lekérdezések segítségével állapítjuk meg a szabályrendszer teljességét és helyességét a tanszéken fejlesztett EMF-IncQuery,[6] modelllekérdező eszköz segítségével. Ehhez szükség volt a szabálymodell összekötésére a folyamat eredeti modelljével.

A folyamat automatizálására Eclipse kiterjesztést készítettünk, így az elemző könnyen tudja ellenőrizni a szabályrendszert.

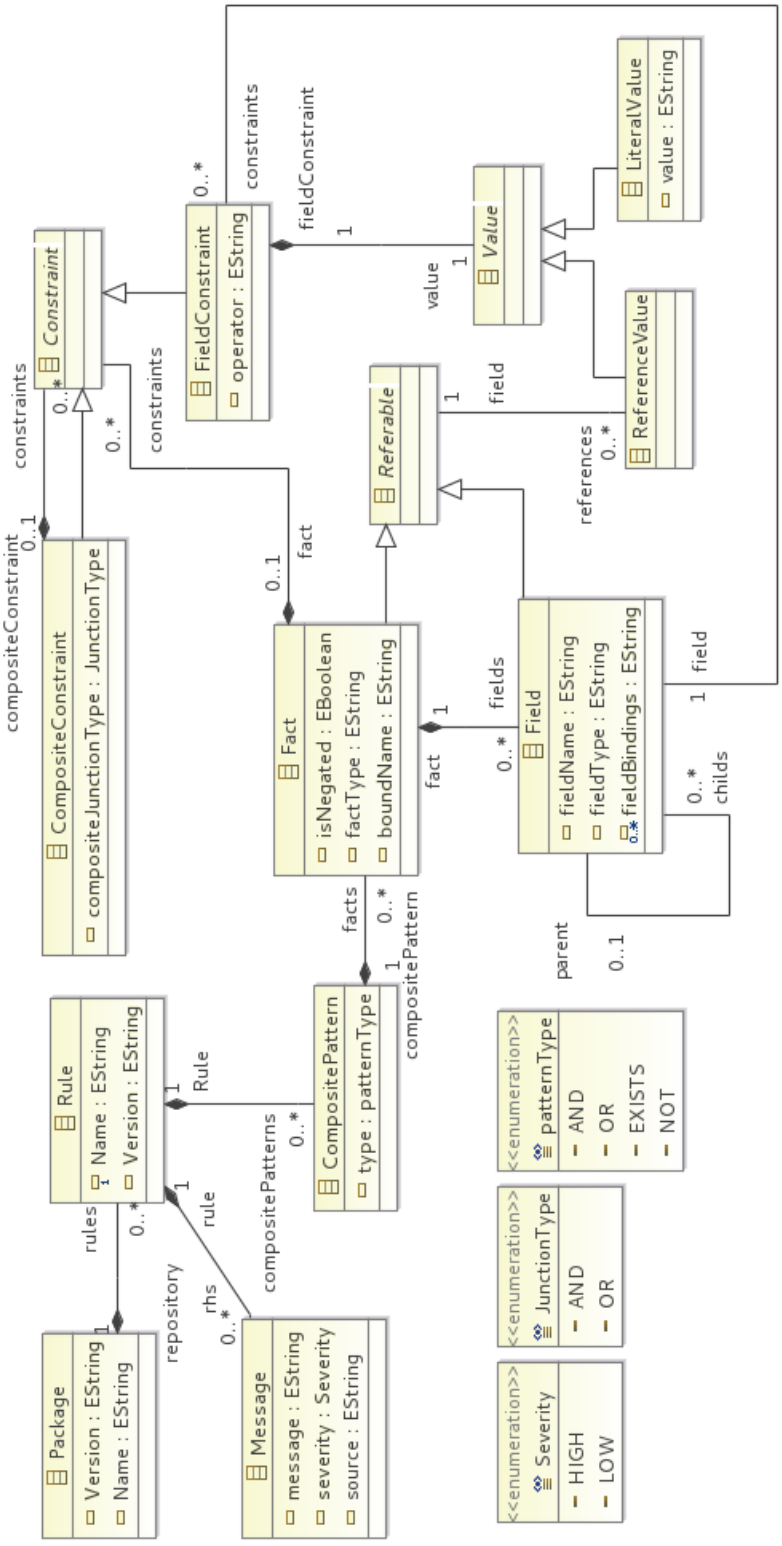
A szabályok EMF modellje a 4.9. ábrán látható. A modellt úgy terveztük meg, hogy a lehető legtöbb, a szabályszerkesztőben felírható szabály modellezhető legyen. Azonban pár megszorítást tennünk kellett, mert a szabályszerkesztő nyújt lehetőséget a szabályok szöveges megadására is, illetve Java ellenőrző metódusok használatára. Ezeknek támogatását nem tudtuk megoldani rendkívül nagy komplexitásuk miatt.

Teljesség vizsgálata

A szabályrendszer teljességének vizsgálatához szükség van egy szakértő segítségére, aki megjelöli az üzleti szempontból érdekes részeket a folyamat modelljén. Az ezután visszaolvasott szabálymodellel ezt a folyamatmodellt összekötve egy olyan modell áll rendelkezésre, amely tartalmazza mind a szakértő jelöléseit, mind az erre a folyamatra felírt szabályokat. Ezen a modellen már futtathatunk olyan kereséseket, amelyek megtalálhatják a megjelölt, azonban figyelmen kívül hagyott részeket a folyamatban. Ezt a keresést IncQuery segítségével oldottuk meg, amely hatékonyan, inkrementálisan tud mintaillesztéseket végezni EMF modelleken.

Egy folyamat érdekes részei lehetnek egyes aktivitások, változók, de akár a teljes folyamat is. Egy aktivitás fontos lehet, ha abban sok külső erőforrást használunk, ilyenkor azért érdemes rá szabályt felírni, mert így az üzleti folyamatot kiszolgáló erőforrást is tudjuk figyelni. Ennél azonban fontosabb az az eset, mikor az adott aktivitásban a felhasználóknak az adott szakterület szempontjából fontos döntéseket kell meghozni, például nagyobb pénzösszegekről kell dönteniük. Ilyenkor a szabályok felírásával csalásokat is felderíthet az üzleti elemző. Ezért igen fontos, hogy az elemző ezekre a kritikus részekre megfogalmazzon olyan szabályokat, amelyek azok kritikus tulajdonságait figyelik. Ez az általunk adott módszerrel biztosítható.

A 2.1.2. alfejezetben leírt folyamaton egy egyszerű példa lehet a határidő. A folyamatban nem követeltük meg a határidő éles betartását, ez valós rendszerekben is sokszor



4.9. ábra. A szabályok EMF modellje

előfordulhat, technológiai vagy üzleti szempontok alapján nem lehet vagy célszerű megakadályozni egy dokumentum adott határidő utáni létrehozását, azonban fontos azonosítani a határidő utáni lépéseket. Ezért érdemes a határidőre szabályt felírni, ami riaszt, ha egy határidőt lekéstek. Ezen a szabály felhasználásával a következő fejezetekben bemutatott EDA használatával kimutathatunk statisztikákat például az egyes üzletfelek és a késések időtartama között. Ennek megkövetelésére egy egyszerű annotációt kell megtennünk a folyamatmodellen:

```
...
    @IMPORTANT
    Variable deadline {
      type "java.util.Date"
    },
...
```

Ezután a helyességellenőrzés ki fogja mutatni, ha nem írtunk volna fel szabályt a határidő értékére, így egyszerűen kijavíthatjuk ezt a problémát.

Helyesség vizsgálata

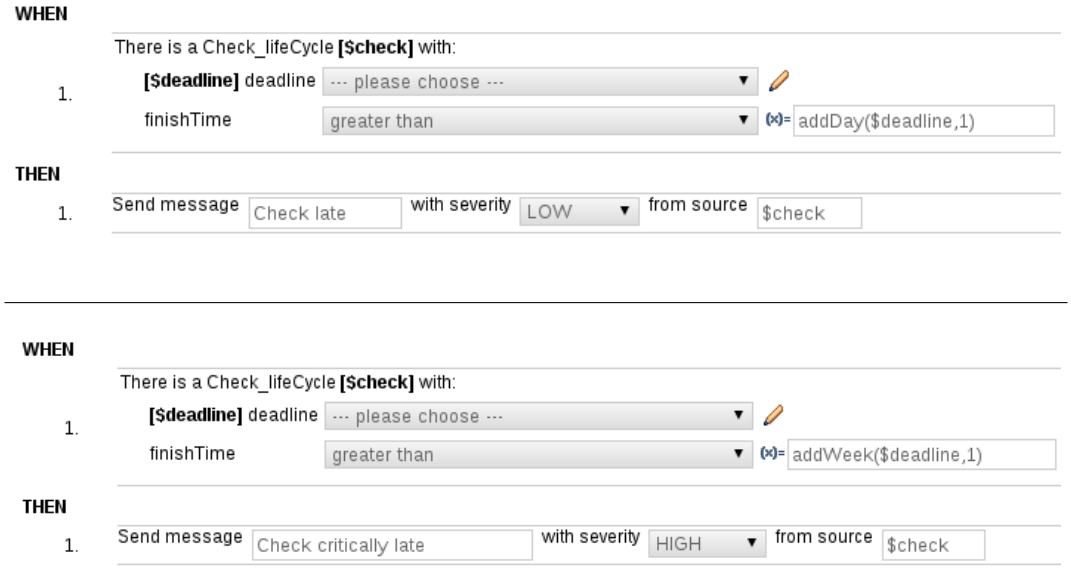
A szabályrendszer helyességének vizsgálata során a szabályokat tartalmi szempontból vizsgáljuk meg. Megvizsgáljuk, vannak-e felesleges, vagy egymásnak ellentmondó szabályok. Két egybevágó szabály tévesen dupla riasztást jelenthet a rendszer futása során. Az egymásnak ellentmondó szabályokból pedig mindenképp hibára következtethetünk, hiszen egy esemény nem lehet egyszerre hibás és helyes is. Ellenőrizzük még a csak részben egybevágó szabályokat is, mikor az egyik szabály a másiknak a részhalmazára vonatkozik. Ebben az esetben ugyanis érdemes lehet a bővebb eseményhalmazra riasztó szabály riasztásait elnyomni, ha a másik szabály riaszt.

A szabályok helyességének a vizsgálatát is az előzőekben leírt modellen végeztük el IncQuery szabályok definiálásával. Ellenőrizzük a szabály struktúrájának azonosságát, azaz, hogy az egyes kényszerek milyen kapcsolatban állnak egymással, illetve az egyes kényszerek azonosságát. Azonban a szabályokban felírt konkrét értékeket már nem, hiszen valós esetben több ugyanolyan szabály, amelynek csak hasonlóak a feltételei, is hibát jelenthet, Ezért figyelmeztetjük az elemzőt akkor is, ha az adott szabályok csak hasonlítanak egymáshoz. Ezek a figyelmeztetések után az elemző már könnyedén át tudja gondolni, mely szabályok okozhatnak problémát.

A hibás szabályok problémát jelenthetnek a folyamatok utólagos elemzésénél is, hiszen a sok hibás kimenet megnehezíti a szabályok lefutási eredményének, és végső soron magának a rendszernek a kiértékelését. Így a következő fejezetben tárgyalt EDA esetén különösen oda kell figyelni a szabályok helyességére, hiszen ott a szabályok végeredményeivel is foglalkozunk.

Az előző pontban bemutatott példán láthattuk, hogy a határidő egy igen fontos érték lehet, azonban nehéz mondani a késésre egy egzakt limitet, mikor számítjuk a késést akkorának, hogy emiatt hibát jelezzünk. Ezért érdemes lehet több szabályt is felvenni. Egyet, ami egy kisebb késés esetén figyelmeztet, és egy másikat, ami nagyobb késés esetén, már hibát, erősebb riasztást eredményez. Ezt a két szabályt létre is hoztuk Guvnorban, a szabályszerkesztőben, a 4.10. ábrán látható módon. Az első szabály akkor küld egy ala-

csony súlyosságú üzenetet, ha minimum egy napot késtek, a második pedig akkor küld egy kritikus üzenetet, ha már legalább egy hete nem dolgozták fel az adott csekket.



4.10. ábra. Két szabály a határidő vizsgálatára

Ha ezt a két szabályt megvizsgáljuk a helyesség ellenőrzővel, hibát fog jelezni, ugyanis ez a két szabály hibás. Mivel a második szabály esetében is csak egy feltétel van a határidőre vonatkozóan, ezért ugyanolyan a felépítésük. Ezen példa esetében az egy hétnél nagyobb késések esetében jelent problémát, ugyanis ekkor mind a két szabály riaszt, feleslegesen, ami későbbi elemzéseknél problémát okozhat. Ilyenkor az eseményelnyomás-technikáját kell alkalmazni, azaz a kisebb súlyosságú szabálynak nem szabadna elsülni, mikor a magas prioritású is. Ezt ebben az esetben legegyszerűbben úgy tehetjük meg, ha az alacsony súlyosságú szabályhoz hozzáadunk egy újabb feltételt, miszerint csak az egy hétnél kisebb késésekre riasszon.

4.2.2. Lefutások elemzése

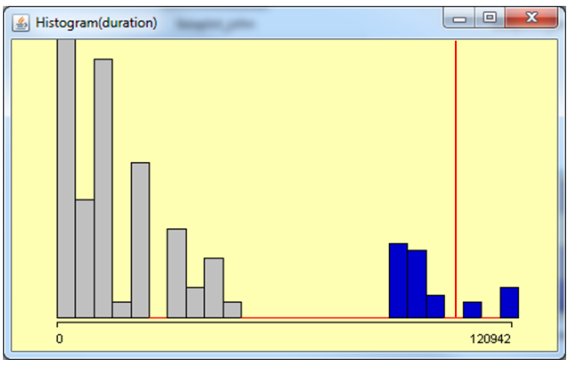
Ebben a részben megmutatjuk, hogy a feltáró adatelemzéssel (EDA[17]) hogyan lehet segíteni a szabályok fejlesztését.

A példákhoz a korábban ismertetett számlaéletciklus folyamatot(2.1) használtuk. Az adatokat mi generáltuk a Szimulátorunk(4.1.6) segítségével. Tehát nem valós, de életszerű, és a szemléltetésre alkalmas adatokkal dolgoztunk.

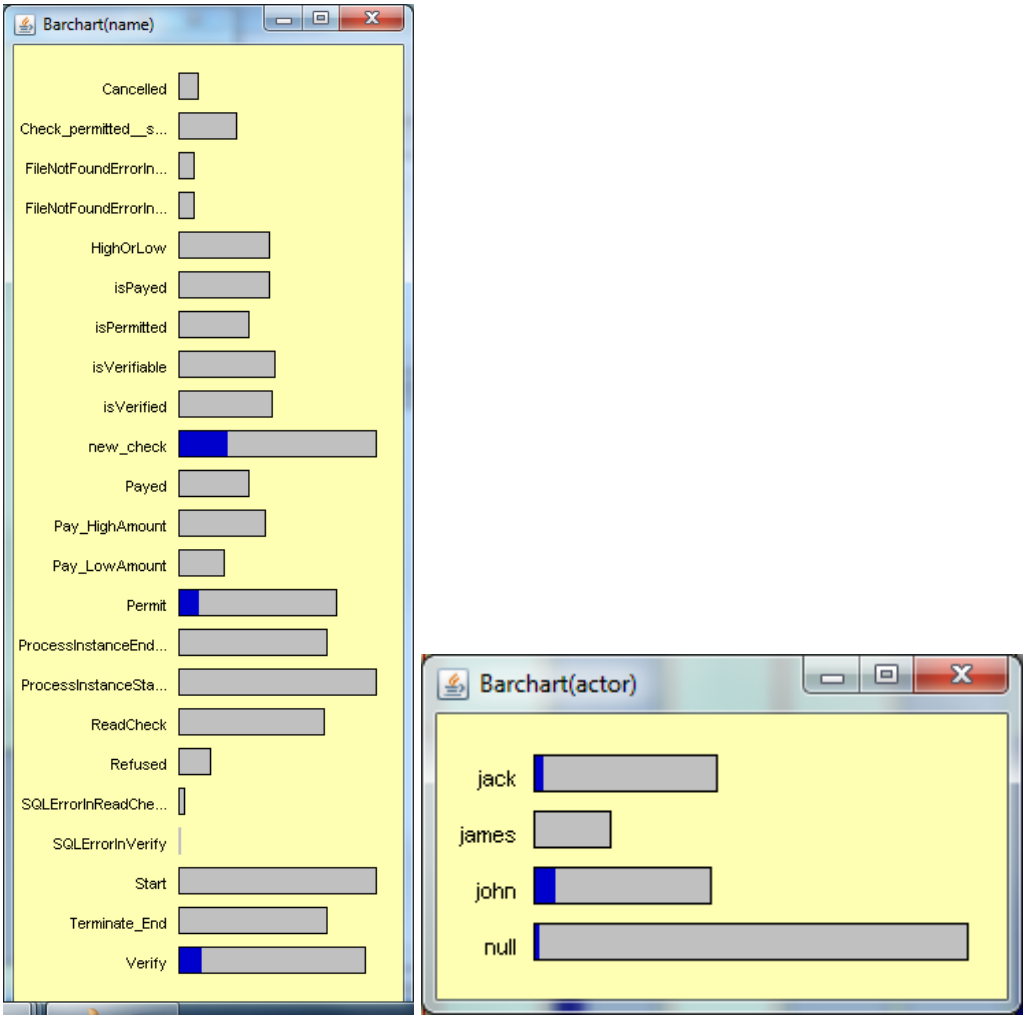
Mondrian[8][19] A vizuális adatanalízis elvégzéséhez a nyílt forráskódú Mondrian nevű eszközt használtuk, mely interaktív kiemelő vizualizációs technológiákat támogat és képes nagy adathalmazok kezelésére.

Késedelmes lefutások A 4.11 ábrán a lefutások számát és idejét ábrázoltuk, valamint utólag a képen jelöltük azt az időhosszt, ami felett későnek ítéltük a folyamatot. Látható,

hogy a küszöbérték alatt igen sok lefutás található és ez a csoport jól elkülönül a többitől. Ez alapján érdemes lehet elgondolkozni azon, hogy ez a küszöbérték jól lett-e megválasztva.



4.11. ábra. Hisztogram a lefutások hosszáról

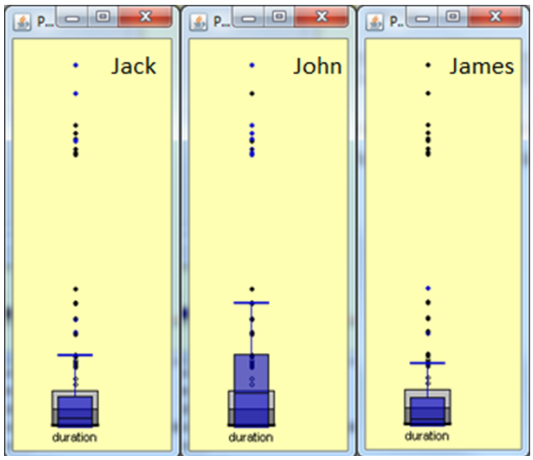


4.12. ábra. Barchart a folyamatlépésekről és a végrehajtókról

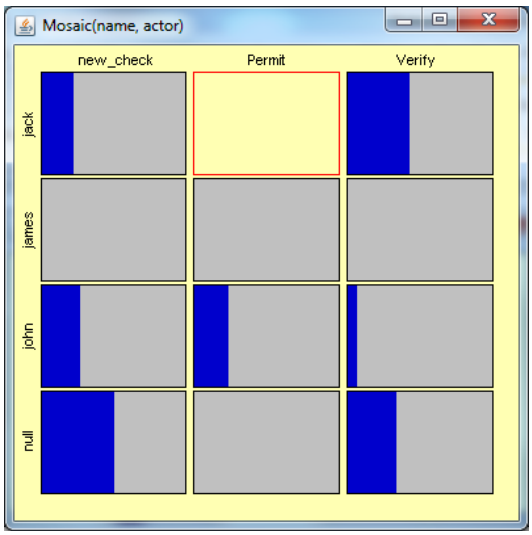
A 4.12 barchart diagramon a folyamat lépések nevei láthatók előfordulási gyakoriság szerint, miközben az előző ábrán bejelölt, hosszan lefutó folyamatlépések kék színnel jelennek meg. Ez a három folyamatlépés az, ami emberi erőforrásokra támaszkodik. Sok olyan adatunk van tehát, ami számunkra nem érdekes, ezért érdemes megtisztítani azokat.

Hasonló módon megnéztük, hogy mely dolgozók (actor) hajtottak végre lassan lépéseket. (A 4.12 ábrán látható null actor a nem emberi végrehajtókat jelenti.) Megfigyelhető, hogy Jack és John végzett lassan bizonyos lépésekkel.

A következő, a 4.13 ábrán három boxplot van egymás mellett. Látható, hogy az egyes dolgozókhoz tartozó folyamatlépések lefutási ideje és szórása mennyivel tér el az összes lefutási időtől és szórásától. Kitűnik, hogy John nagy szórással dolgozik és gyakran tér el az átlagtól.



4.13. ábra. 3 boxplot a dolgozókról

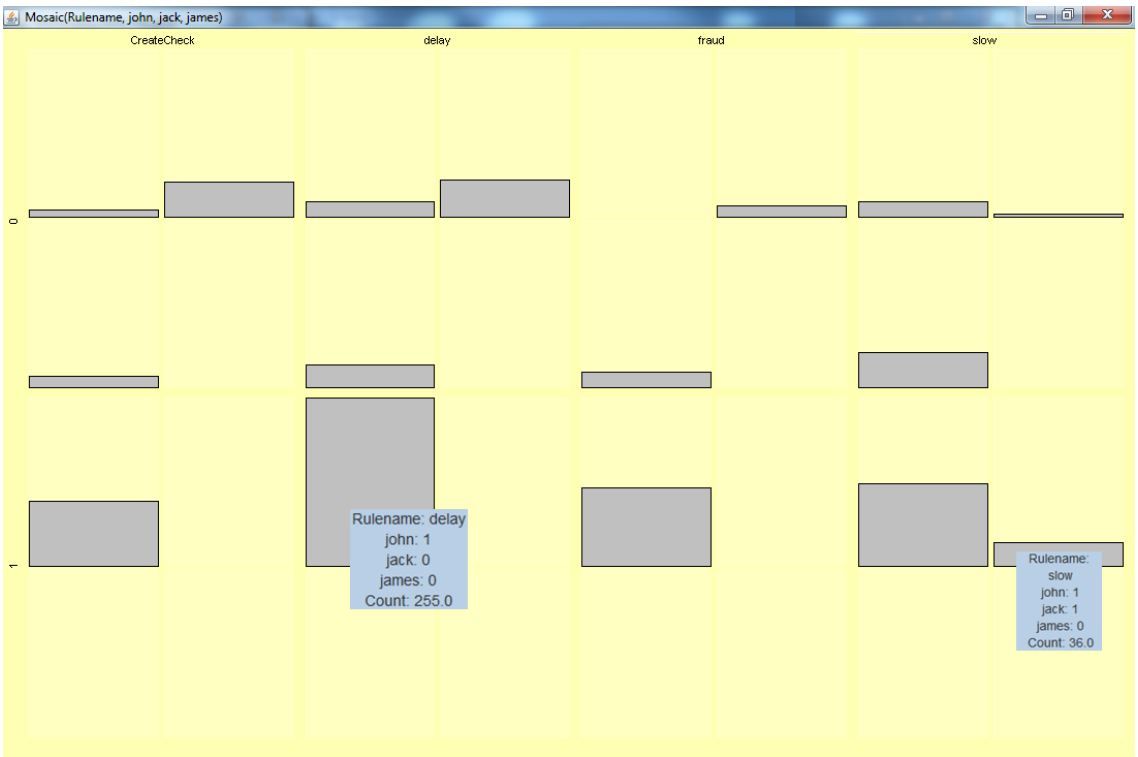


4.14. ábra. Mozaik a folyamatlépésekről és végrehajtókról

Egy mozaik ábrán(4.14) láthatjuk, hogy a végrehajtók az egyes folyamatlépéseket milyen arányban hajtottak végre késedelemmel. (Négyzeteket látunk, mert az adatok norma-

lizálva vannak. Nem mutatja az ábra, hogy az egyes lépéseket hányszor hajtották végre, csak a megfelelő és lassú lefutások arányát.) Jack igen magas, csaknem 50%-ban hosszan hajtja végre a Verify lépést.

A 4.15 mozaik diagram egy kicsit összetettebb. Felül az egyes alkalmazott szabályok láthatók, bal oldalt pedig az dolgozók a következőképpen. Vízszintesen kettéosztva az ábrát a felső részben olyan illeszkedések vannak, amiknél nem volt olyan folyamatlépés, amiben John részt vett volna, az alsóknál pedig mindegyikben részt vett. A két nagy sávot további két-két kisebb sávra osztva ugyanígy látható melyik folyamatokra vonatkozó szabályilleszkedéseknél játszott szerepet James. Itt is a felső jelenti, hogy abban nem vett részt így az ábrán James-hez egyedül a második sorban lévő értékek köthetők. Jack illeszkedései pedig függőlegesen minden második oszlopban láthatjuk.
Az ábráról ismételten leolvasható, hogy John igen nagy számban késve végezte el feladatát. 255 alkalommal illeszkedett az általa elvégzett folyamatra a delay szabály.



4.15. ábra. Mozaik a szabályilleszkedésekről

A feltáró adatelemzés segítségével tehát mélyebb rálátásunk lehet a folyamatok lefutására és a szabályok illeszkedésére, továbbá a módszer támpontot adhat a probléma helyének megtalálásához, ahogy a példa során is kiderült, hogy John késett túl sokszor.

5. fejezet

Továbbfejlesztési lehetőségek

A környezet fejlesztése során számos továbbfejlesztési lehetőség felmerült, a környezetre több helyen is lehetne építkezni.

5.1. További eseményforrások bevonása

Már a dolgozat írása során is felmerült az igény, további, nem a folyamatmotorból származó események bevonására. Ezen felül érdemes lenne bevonni a folyamatmotort kiszolgáló adatbázisból, vagy egyéb rendszerekből teljesítményjellemzőket is, hiszen az EDA során ezeket is érdemes lenne megvizsgálni.

5.2. Kezdeti szabálybázis generálása

Az elkészült környezet csak egy teljesen általános mintaszabályt nyújt telepítés után, azonban ezt ki lehetne terjeszteni. Vannak módszerek, amelyekkel szabályokat lehet generálni a folyamatmotor helyes működésének ellenőrzésére, ezt megvalósítva segíthetnénk az elemzők munkáját. [20] Továbbá a teljesség vizsgálatánál felhasznált jelölések segítségével, lehet egy szabályvázat generálni a kritikusnak ítélt változók vizsgálatára, amit az elemzőknek már csak ki kell tölteniük.

5.3. Valós idejű szabálykiértékelés

A környezet megalkotása során olyan modellt alkottunk meg, amely alapvetően kisebb kiegészítésekkel alkalmas valós idejű elemzésre. Egyes esetekben, például ha nagyszámú folyamat fut le, vagy ha azonnal kellenek az eredmények, érdemesebb valós idejű diagnosztikát használni. A mi esetünkben a diagnosztika többszöri lefuttathatósága volt a cél, azonban érdemes volna felkészíteni erre is a diagnosztikát. [13]

5.4. Az eredmények visszavezetése a kezdeti folyamatmodellre

Az elkészült eszköz egyelőre szöveges kimenetben jelzi az eredményeket. Ezt érdemes lenne visszavezetni a kezdet folyamatmodellre. Így a folyamat definiálásának helyén tudnánk megjeleníteni az eredményeket. Meg tudnánk mutatni a modellen, melyik megjelölt objektumra nem készült el szabály, valamint a diagnosztika kimenetét is vissza lehetne csatolni a folyamatmodellre.

5.5. Az EDA során észrevett eredmények alapján szabályjavatok készítése

Az adatanalízis során feltárt összefüggések felhasználásával készíthetőek javaslatok egyes szabályok módosítására, esetleg új szabályok definiálására. Ha egy folyamat például igen nagy szórású végrehajtási idővel rendelkezik, akkor érdemes lehet azt felülvizsgálni, szabályokat felírni rá. Esetleg, ha a folyamat egy változójára írunk fel olyan feltételt, amely annak egy

5.6. EPA használatával a hibaokok felderítése

Az EPA (Error Production analysis) használatával megoldható lenne a hibák okainak felderítése, nem csak a hibák detektálása. Az általunk felépített modell erre alapvetően alkalmas, azonban ehhez máshogy kell kezelni az eseményeket.

5.7. A szabályok felírásának nyelvi támogatása

A Guvnor alapvetően egy grafikus szabályszerkesztő, de mégis a szöveges felírás módját követi. Érdemes lehet azonban egy valóban grafikus szerkesztő elkészítése, amely segítségével például egy gráffal adhatunk meg szabályokat.

A másik alternatíva a szabályok felírásának könnyítésére valamilyen természetes nyelv támogatása. Ehhez már a Drools-hoz is létezik kiegészítés, amely támogatja az ilyen nyelvek definícióját.

5.8. A komponensek rugalmas cserélhetősége

A diagnosztikai környezet tervezésekor az egyik fő szempont, hogy a lehető legtöbb komponense könnyen cserélhető legyen. Ezért a használt összes modellt a lehető legáltalánosabbra alkottuk meg.

5.8.1. Folyamatmotor

A folyamatmotor lecserélhető, hiszen a 4.1.2. alfejezetben ismertetett eseménymodellt folyamatmotorfüggetlenül alkottuk meg. Ennek köszönhetően a folyamatmotor esetleges leváltásakor csak az események folyamatmotorból való kiolvasását kell megírni. Ennek eszközfüggőségét nem lehetett megkerülni, hiszen minden folyamatmotor más lehetőséget nyújt ezeknek a kinyerésére.

5.8.2. Adatbáziskezelő

Az adatbázis lecserélhető, semmilyen adatbázis-kezelő specifikus funkcióját nem használtuk ki a PostgreSQL adatbázis-kezelőnek.

5.8.3. Szabályszerkesztő, folyamatmotor

A szabályszerkesztő lecserélhető a generátor kisebb módosításával, amennyiben az támogatja a Drools nyelvén definiált szabályokat, objektumokat. A szabálymotor lecseréléséhez viszont több módosítás szükséges, mivel a statisztikákat számító szabályokat újra kell fogalmazni. Ehhez alternatíva lehet egy olyan általános szabálykezelő rendszer használata, amit a [13] könyvfejezetben mutatnak be.

6. fejezet

Összefoglalás

A dolgozat során elkészítettünk egy olyan környezetet, amely segítségével az üzleti elemzők könnyedén monitorozhatják az üzleti folyamataikat.

Felműszereztünk egy folyamatmotort A Bonita folyamatmotorból minden szükséges információt, esemény ki tudunk olvasni futásidőben.

A folyamatmotorból származó eseményeket feldolgoztuk A folyamatmotor által szolgáltatott eseményeket feldolgozzuk, leképezzük egy általános eseménymodellre, majd eltároljuk egy adatbázisba. Az általános eseménymodell használatával felkészítettük a rendszert a folyamatmotor leváltására, külső események bevonására.

Az eseményeket szakterület specifikus objektumokká képeztük A folyamatmotorból származó események teljesen generikusak, így ezeket le kellett képeznünk futási időben egy általunk definiált szakterület specifikus modellre.

Szakterület specifikus környezetet biztosítunk A Bonitából visszaolvassuk a folyamat definícióját, ebből előállítunk egy szakterület specifikus eseménymodellt, amelynek megfelelő eseményekre alakítottuk a folyamatmotorból származó eseményeket. Ezt az eseménymodellt egyéb segédobjektumokkal együtt feltöltjük egy grafikus szabályszerkesztőbe. A szakterület specifikus eseménymodellt úgy alkottuk meg, hogy azt a Guvnor grafikus szabályszerkesztő támogassa. Automatizáltuk a szabályszerkesztő beállítását, így a folyamatmodell egyszerű importálása után a Guvnorban rendelkezésre áll az eseménymodell, valamint a szabályfejlesztést megkönnyítő egyéb segédobjektumok.

Bevezettünk a folyamatmodellen kívüli további modelleket Az eseménymodellbe bevezettünk egy felhasználómodellt, így a szabályokban nem csak a folyamatmotor által feldolgozott felhasználókra tudunk szűrni, hanem azok főnökeire, csoportjaira, szerepeire is.

Segítjük a hatékony tesztelést A környezet teszteléséhez elkészítettünk egy szimulátor komponenst, amely segítségével automatizáltan lehet lefuttatni Bonita folyamatokat, ezzel a rendszert tesztelni.

Megvizsgáljuk a szabályokat és a rendszermodellt Az üzleti elemző által felírt szabályok vizsgálatára elkészítettünk egy eszközt, amely képes a szabályok ellenőrzésére teljességi és helyességi szempontból. A dolgozat során bemutattuk, hogy a folyamatok lefutásából, valamint a szabályok illeszkedéseiből, hogyan lehet feltárni a szabályrendszer, vagy a folyamat esetleges hibáit.

A teljes környezetet úgy készítettük el, hogy a lehető legtöbb komponense könnyedén cserélhető legyen, ezt a modell alapú megközelítés segíti elő. A rendszer igen sok továbbfejlesztési lehetőséggel rendelkezik.

Környezetünket elsősorban az ismertetett folyamattal(2.1) teszteltük. Összesen 80 ezer eseménnyel futtatuk a szabálykiértékelést, ami 1700 folyamatot jelent. Ilyen mennyiségű adat mellett is megfelelően működött a rendszer.

Irodalomjegyzék

[1] Bonita 5.10.1. http://www.omg.org/spec/BPMN/2.0/PDF, October 2013.

[2] BPMN 2.0. http://www.omg.org/spec/BPMN/2.0/PDF, October 2013.

[3] COBIT. http://www.isaca.org/Knowledge-Center/cobit/Documents/COBIT4.pdf, October 2013.

[4] DROOLS Rule Engine 5.5.0 Final. http://www.jboss.org/drools, October 2013.

[5] Eclipse modeling framework project (EMF) 2.9.1. http://www.eclipse.org/modeling/emf, October 2013.

[6] EMF-IncQuery 0.7.1. http://www.eclipse.org/incquery, October 2013.

[7] GUVNOR Rule Editor 5.5.0 Final. http://www.jboss.org/drools/drools-guvnor.html, October 2013.

[8] Mondrian 1.2. http://stats.math.uni-augsburg.de/Mondrian, October 2013.

[9] Sarbanes-Oxley. http://www.sec.gov/about/laws/soa2002.pdf, October 2013.

[10] www.complexevents.com. http://www.complexevents.com/, October 2013.

[11] XTEXT 2.4.3. http://www.eclipse.org/Xtext, October 2013.

[12] György Csertán, András Pataricza, P. Harang, Orsolya Dobán, Gabor Biros, András Dancsecz, and Ferenc Friedler. Bpm based robust e-business application development. In Andrea Bondavalli and Pascale Thevenod-Fosse, editors, *Dependable Computing EDCC-4*, volume 2485 of *Lecture Notes in Computer Science*, pages 32–43. Springer Berlin Heidelberg, 2002.

[13] László Gönczy and István Dávid. Ontology-supported design of domain-specific languages: A complex event processing case study. In *Advances and Applications in Model-Driven Engineering*, pages 106–133. IGI Global, 2013.

[14] Gabriel Hermosillo. *Towards Creating Context-Aware Dynamically-Adaptable Business Processes Using Complex Event Processing*. PhD thesis, Université des Sciences et Technologie de Lille-Lille I, 2012.

[15] David C Luckham. *The power of events*, volume 204. Addison-Wesley Reading, 2002.

[16] Linh Thao Ly, Stefanie Rinderle-Ma, David Knuplesch, and Peter Dadam. Monitoring business process compliance using compliance rule graphs. In *On the Move to Meaningful Internet Systems: OTM 2011*, pages 82–99. Springer, 2011.

[17] András Pataricza, Imre Kocsis, Ágnes Salánki, and László Gönczy. Empirical assessment of resilience. In Anatoliy Gorbenko, Alexander Romanovsky, and Vyacheslav Kharchenko, editors, *Software Engineering for Resilient Systems*, volume 8166 of *Lecture Notes in Computer Science*, pages 1–16. Springer Berlin Heidelberg, 2013.

[18] Kai Richter and Rolf Ernst. Event model interfaces for heterogeneous system analysis. In *Design, Automation and Test in Europe Conference and Exhibition, 2002. Proceedings*, pages 506–513. IEEE, 2002.

[19] Martin Theus and Simon Urbanek. *Interactive graphics for data analysis: principles and examples*. CRC Press, 2011.

[20] Matthias Weidlich, Holger Ziekow, Jan Mendling, Oliver Günther, Mathias Weske, and Nirmit Desai. Event-based monitoring of process execution violations. In *Business Process Management*, pages 182–198. Springer, 2011.