Rendszerintegráció és -felügyelet laboratórium (VIMIM309)

Rendszerfelügyelet támogatása komplexesemény-feldolgozással

Mérési segédlet

Készítette: Bergmann Gábor, Dávid István

Utolsó módosítás: 2015. április 9.

Verzió: 1.3

Budapesti Műszaki és Gazdaságtudományi Egyetem
Méréstechnika és Információs Rendszerek Tanszék

1 Bevezető

A labor során a méréseket végző hallgató a gyakorlatban is megismerkedik a rendszerintegráció és rendszerfelügyelet során használatos módszerekkel és eszközökkel. Végigköveti egy elosztott alkalmazás megvalósításának és felügyeletének legfontosabb lépéseit, ipari környezetben használt integrációs köztes réteg (middleware) technológiák és felügyeleti eszközök használatával. A mérések a következő témakörökhöz kapcsolódnak:

1. Munkafolyamatok megvalósítása Java nyelven
2. Rendszerfelügyelet támogatása komplexesemény-feldolgozással
3. Megbízható üzenetküldés IBM WebSphere MQ alapon
4. Kommunikáció JMS és JMX technológia segítségével
5. OSGi szolgáltatások fejlesztése
6. Aktor modell konkurens alkalmazások készítésére: Akka
7. Felügyeleti adatok vizuális elemzése

A mérések egy részén a hallgatók különféle elosztott szolgáltatásintegráció megoldásokkal valósítják meg egy példarendszer üzleti logikáját. A jelen mérés során a megvalósítás helyett a felügyelet lesz fókuszban: szabályalapú formalizmussal kell a korábban elkészült üzleti folyamat futását felügyelni, a JBoss Drools Fusion terméke segítségével. A vizsgált rendszer felműszerezésének hatására futás közben eseményfolyam keletkezik. A ha–akkor jellegű deklaratív szabályokból álló monitoring rendszer ebben kereshet komplex, időhöz kötött összefüggéseket. A hallgatók elsajátíthatják a komplex események szabály alapú feldolgozásának alapelveit, és megismerkedhetnek az üzleti folyamatok felügyeletével.

2 Háttérismeretek

2.1 A komplexesemény-feldolgozás alapfogalmai

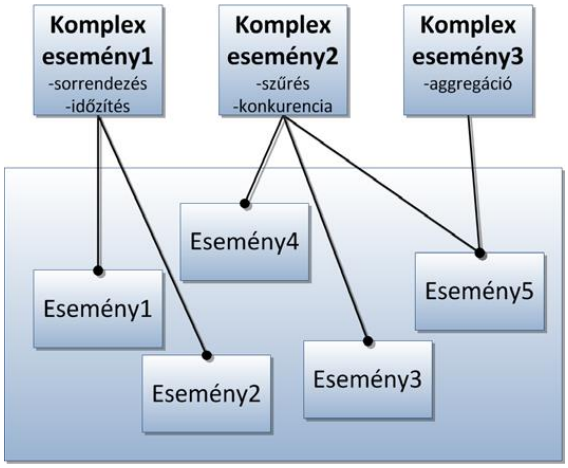
Az informatikai infrastruktúrákban, alkalmas monitorozó megoldásokat használva megfigyelhetővé válnak az erőforrás szintű diagnosztikai események. Ilyen erőforrás szintű esemény lehet például a következő: „a szerver processzorának terheltsége eléri a kritikus értéket”.
Az események feldolgozásával olyan információkat állíthatunk elő, amelyek vezérlési, döntési helyzetekben jól hasznosíthatóak. Az információk kinyerésének alapja az *eseményfolyamon (event stream)* értelmezett mintafelismerés.

Az *atomi esemény* egy adott forrásból származó primitív adategység – a forrás mérhető paramétereiről hordoz információkat. A gyakorlatban jellemző, hogy a releváns, minket érdeklő minták túl összetettek ahhoz, hogy egyetlen atomi esemény le tudja írni azokat, így leírásukhoz több atomi eseményt kell felhasználni, logikailag összekapcsolni – ezt az összekapcsolást nevezzük *komplex eseménynek*, feldolgozási technikáját pedig *komplexesemény-feldolgozásnak* (*CEP – complex event processing*).

Komplex esemény például a következő: „az SLA-t veszélyezteti, ha az adatbázis szerveren a processzor terheltsége a kritikus szint körül van huzamosabb ideig, miközben a tartalék szerveren a diszk megtelt.”

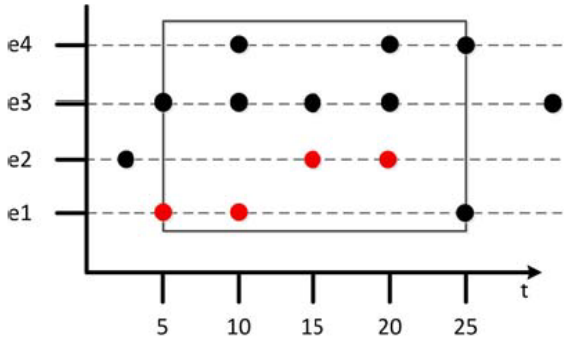
Az egyes mintákhoz végrehajtható szekvenciák, ún. akciók társíthatók, amelyek lefutását a minta felismerése váltja ki (*trigger*).

A komplex esemény tehát egy olyan struktúra, ami (i) atomi eseményekből, (ii) az atomi események közti logikai, algebrai, ill. időbeli kapcsolatokból, valamint (iii) a felismert eseményminták hatására végrehajtódó akciókból épül fel.



1. ábra – A komplex események atomi eseményekből épülnek fel, amelyek között algebrailag formalizálható kapcsolatokat definiálnak.

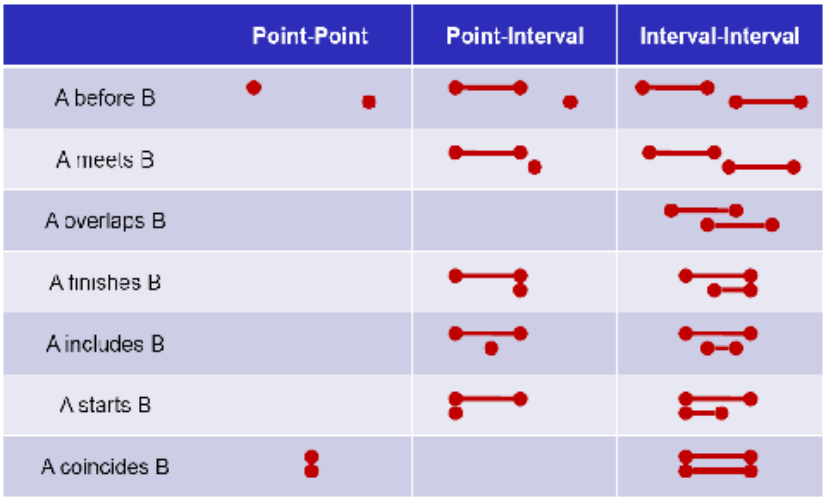
Az atomi események közti kapcsolatok alatt olyan fogalmakat értünk, mint sorrendiség definiálása, időbeliség megkötése (például időablakok segítségével), az események előfordulásának számossága, stb. (Ilyen alkalmas algebrai formalizmust definiál az *Event Detection Algebra*[1] és az *Allen-féle intervallum algebra*[2].)



2. ábra – Az eseményfolyamon e1, e2, e3 és e4 események érkeznek adott időpillanatokban. Fel kell ismerni, ha két e1 esemény után két e2 következik 20 másodpercen belül. A minta sorrendezés, számosság és időzítés dimenziók segítségével írható le.

Fontos megkülönbözetetni pont- és intervallum típusú eseményeket, mert az egyes típusoknak más-más tulajdonságai definiáltak, így a köztük értelmezett algebrai operátorok is eltérőek lehetnek. Például a JBoss Drools Fusion eseményfeldolgozó a következő ábrán részletezett temporális operátorokat definiálja az egyes típusok között:

[1] James F. Allen, "Maintaining knowledge about temporal intervals," Communications of the ACM, vol. 26 Issue, no.11, pp. 832-843, November 1983.
[2] Jan Carlson, "An Intuitive and Resource-Efficient Event Detection Algebra", 2004.



3. ábra – A JBoss Drools Fusion által definiált temporális operátorok.

2.2 Mikor célszerű a komplexesemény-feldolgozás alkalmazása?

Bár a mérés során rendszerfelügyeleti feladatokat oldunk meg a CEP segítségével, maga az elv és a kapcsolódó technikák más szakterületeken is jól bevált megoldásnak számítanak. Klasszikus alkalmazási terület például az algoritmikus kereskedés (*algorithmic trading*), az online csalások felderítése (*fraud detection*), vagy a rendszerbiztonsági monitorozás (*security monitoring*).

Az összes alkalmazási terület sajátossága, hogy **nagyszámú**, elosztottan és egymástól függetlenül működő **erőforrás**, vagy ágens eseményeit szükséges feldolgozni, mert feltételezzük, hogy az **események kombinált vizsgálata** olyan új információkat állít elő, ami üzletileg releváns; mindezt pedig **alacsony késleltetés**sel szükséges végrehajtani.
(Az algoritmikus kereskedés esetében eseményfolyam lehet például a részvények árfolyamának idősora. Mivel jellemzően nem egy típusú részvénnyel szokás kereskedni, hanem portfóliókba szervezve több különböző értékpapírral, ezért nem egyértelmű, hogy egy adott részvény értékének megváltozásakor szükséges-e adás-vételi akciót indítani, vagy sem. A döntéshez portfólió szinten kell optimalizálni, amihez a többi értékpapír változását is figyelembe kell venni, a számításokat pedig a lehető leggyorsabban szükséges végrehajtani – erre a CEP egy tökéletes megoldás lehet.[3])

2.3 Az eseményfeldolgozás technikai aspektusai, eseményfeldolgozó platformok

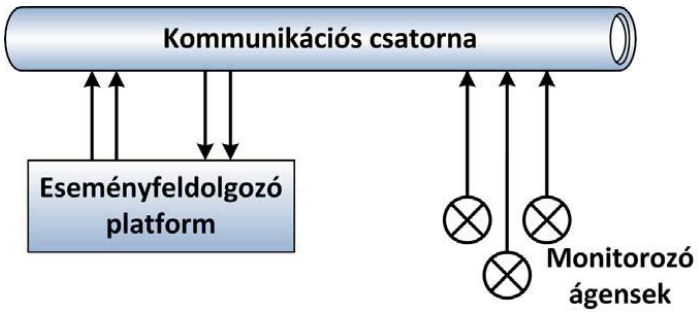
Az eseménymintákat egy *eseményfeldolgozó platform* megfelelő leírónyelvének segítségével definiáljuk, amelynek bemenete az eseményfolyam. A nyílt forráskódú platformok közül a két legeltejedtebb a JBoss Drools Fusion, valamint az Esper. Emellett – természetesen – minden nagy gyártó rendelkezik saját implementációval: IBM InfoSphere Streams (korábban: System-S), Oracle CEP, Microsoft StreamInsight, TIBCO BusinessEvents, StreamBase CEP.

A mérés során használt Drools Fusion intuitív, szabály alapú szintakszist nyújt a minták és akciók leírásához (LHS→RHS); de akadnak olyan platformok is, amelyek SQL-szerű nyelvet használnak a minták definiálásához (az eseményfolyamot „query-zik”), a végrehajtandó szekvenciákat pedig update listener osztályokban tárolják – ilyen elven működik például az Esper.

Az eseményeket az erőforrás maga is publikálhatja, vagy ha erre nincs felkészítve, a monitorozó keretrendszer szolgáltatásai biztosítanak számára ilyen jellegű képességet.

Mivel a gyakorlatban rövid idő alatt nagyszámú esemény keletkezhet, praktikus okokból az események nem tárolhatóak korlátlan ideig. Az eseményfeldolgozó platform egyik feladata az *események életciklusá*nak kezelése, hogy az elavult, régi események ne terheljék tovább a monitorozó eszközt.

[3] http://www.wallstreetandtech.com/technology-risk-management/204203344



4. ábra – A monitorozó ágensek a kommunikációs csatornára továbbítják az erőforrások eseményeit, amit az eseményfeldolgozó platform értelmez és a megfelelő akciókat végrehajtja.

2.4 Egy CEP feladat megvalósításának tipikus lépései

Egy feladat CEP segítségével történő megoldása tipikusan a következő fázisokat érinti.

1. *A megfigyelt rendszer felműszerezése.*

   Ezzel biztosítjuk, hogy a rendszerben valóban lesznek megfigyelhető események.

2. *A releváns eseményminták meghatározása.*

   Az üzleti célokat (*business goals*) figyelembe véve meghatározzuk a releváns eseménymintákat. A minták a megfigyelhető események attribútumaira írnak elő matematikai, logikai megkötéseket.

3. *Az eseményminták leírása valamely platform által kínált leírónyelv segítségével és telepítés.*

   Az eseményminták attribútumait ezen a szinten már konkrét értékekkel, számszerűsítve szükséges megadni. Ezzel előáll az *eseménykonfiguráció*, amelyet a feldolgozó platform értelmezni tud.

4. *Feldolgozás.*

3 A JBoss Drools Fusion technológia

A JBoss Drools szabályalapú technológiai platform részét képezi a Fusion eseményfeldolgozó. Az alábbiakban a – korábbi félévben már felhasznált - Drools Expert szabályvégrehajtó motor ismétlő jellegű áttekintése után bemutatjuk a Drools Fusion főbb elemeit. A jelen segédletben leírtakon túl erősen ajánlott az on-line JBoss Drools dokumentáció használata.

3.1 Ismétlés: Drools Expert alapismeretek

A Drools platform központi szerepű modulja az Expert üzleti szabály végrehajtó rendszer (business rule engine). Tartalmazza a szabályleíró nyelv(ek)et, a végrehajtó szabálymotort a megfelelő programozói interfészekkel, továbbá az Eclipse alapú fejlesztőkörnyezetet.

A Drools terminológiában a tudásbázis (Knowledge Base) a szabályok definícióiból áll (és egyéb elemekből, pl. folyamatleírás). Tényleges szabályvégrehajtáshoz ezért kérni kell egy úgynevezett sessiont a tudásbázisból, amely már tartalmazni fog ténybázist is, így beszúrhatóak üzleti objektumok és tüzelhetnek szabályok. Egy tudásbázis fölé több független session is nyitható egyszerre.

A Drools által kínált kétféle session közül a mi céljainknak az állapotos (stateful) session fog megfelelni, ezért a továbbiakban csak ezzel foglalkozunk. Itt van valódi munkamemória (working memory, WM), amelybe beszúrhatóak, ill. kivehetőek tények (POJO objektumok). Ha a munkamemória tartalma alapján egy produkciós szabály minden feltétele ki van elégítve, akkor a szabály aktivált. Mivel a feltételrészben jellemzően szabad (lekötetlen) változók vannak, a feltételeket valójában ezen változók egy konkrét lekötése (behelyettesítése) elégíti ki; ezeket a kielégítő behelyettesítéseket nevezzük *aktivációnak* (activation), és a változók konkrét értékeiből formált tuple („n-nes”) azonosítja őket. A változók egy

konkrét behelyettesítése – vagyis az aktiváció – alapján a szabálypéldány tényleges végrehajtása a *tüzelés* (firing). Az összes pillanatnyilag érvényes aktiváció halmaza a napirend (*agenda* vagy conflict set), ezek közül kell kiválasztani a tüzelendőt.

A WM feltöltése után után meghívható a `fireAllRules()` metódus, amely iterálva (előre láncolással) hajtja végre a szabályokat, folyamatosan módosítva közben a WM tartalmát (a napirendet is karbantartva); akkor tér vissza a metódus, ha nincs már tüzelhető szabály napirenden. Ezek után a WM „kívülről" tovább módosítható, majd újra hívható a `fireAllRules()`, stb.

Mivel a szabályok aktivációinak meghatározására a Drools is inkrementális mintaillesztőt használ, az agenda frissítéséhez szükséges értesülnie a WM változásairól. A WM-be beszúrást és törlést természetesen a Drools API-jain keresztül bonyolítjuk, ezért ez nem okoz gondot. A Java objektumok módosítása azonban nem detektálható általános esetben, ezért ezt a Drools számára jelezni kell valamilyen módon. A megoldás az objektum beszúrásakor az `insert()` metódustól visszakapott `FactHandle` referencia, melyen keresztül értesíthető a Drools Engine a módosításról, hogy újraolvashassa az objektum attribútumait. Szerencsére a leggyakoribb esetben, a szabály akció részében történő módosítások esetén ennél elegánsabb megoldás is van (ld. később).

Egy nagyon egyszerű Drools-os programnak definiálnia kell az üzleti objektumok adatmodelljét; ez Java osztályokat jelent, amelyek a JavaBean konvenciókhoz igazodnak, pontosabban `getXXX()` és `setXXX()` metódusokon keresztül attribútumokat kínálnak fel (az objektumok mezői a Drools számára nem láthatóak). Ezen felül szükség van a szabályokat definiáló fájlra (alapesetben .drl formátumban). A keretprogram a szabálydefiníciós fájl alapján felépíttet egy Drools tudásbázist, majd kér egy állapotos sessiont (Drools 5.x esetén a `newStatefulKnowledgeSession()` metódussal). A session WM-jébe `insert()` metódussal elhelyezi a kezdeti üzleti objektumokat, majd meghívja a `fireAllRules()` metódust. Az IDE varázslója által generált Hello World projekt hasznos példát szolgáltat a Drools Expert API megismeréséhez.

Drools 6.x esetén az API némileg megváltozott. A kmodule.xml leírófájlban megadott konfiguráció révén bizonyos beállítások egyszerűbbek; a `kContainer.newKieSession("ksession-rules")` hívással példányosítható az XML-ben `ksession-rules` néven deklarált session.

3.2 Ismétlés: Drools Expert szabálynyelv

A szabályok definiálására a Drools alapértelmezetten a DRL szöveges nyelvet használja; a továbbiakban csak ezzel a formátummal foglalkozunk. A szabálydefiníciót a **`rule`** kulcsszó vezeti be, amelyet a szabály idézőjelek közé tett neve követi. Ezek után igény szerint a szabály működését befolyásoló *opciók* specifikálhatóak. A szabály két legfontosabb része a **`when`** kulcsszó után következő feltételrész és a **`then`** kulcsszó után megadott akciólista. A szabálydefiníciót `end` zárja. Példaként nézzük meg két Drools szabályt!

```
rule "GoodBye"
  when
    MessageBox( status == MessageBox.GOODBYE, myMessage : message )
  then
    System.out.println( myMessage );
end

rule "Cukroz"
  when
    $szilva: Szilva ()
    $pult: KonyhaPult (tartalom contains $szilva)
    not Object (this memberOf $szilva.tartalom) // ures a szilva
  then
    KockaCukor $cukor = new KockaCukor();
    insert($cukor);
    modify($szilva) {
      berak($cukor);
```

```
    }
end
```

A feltételrészbe leggyakrabban objektumminták kerülnek, amelyek a megnevezett Java osztály WM-beli példányaira illeszkednek (ha az osztály más Java package-ben van, a Drools fájl fejlécében **import** kulcsszóval behivatkozható). Az objektumminta az osztály teljes példányhalmazát megszoríthatja opcionális attribútum-korlátozásokkal; a hagyományos összehasonlításokon (pl. ==, <) kívül Drools-specifikus operátorok is elérhetőek, pl. halmaz-elem viszony vizsgálatára (**contains**/**memberOf** a fenti példakódban). Az objektumok ill. attribútumaik a kettőspont operátorral felfoghatóak változóba (amelynek gyakran – de nem kötelezően – dollárjellel kezdődő nevet adnak), és a változó hivatkozható lesz a feltétel ág többi részében ill. a következményrészben. További kényszerfeltételek **eval** kifejezésben adhatóak meg.

A feltételrészben szereplő feltételek (objektumminta, eval) alapértelmezésben és-kapcsolatban állnak, de **and** és **or** operátorokkal ill. zárójelekkel ez befolyásolható. Nagy a jelentősége a **not** kvantornak, amely az utána következő feltétel (vagy feltételek zárójelezett összessége) *kielégíthetetlenségét* mondja ki, vagyis: nincsenek olyan elemek a WM-ben, amelyek megadott típusúak és megadott viszonyban állnak egymással és a többi objektummal. Hasonló az **exists** kvantor, amely egzisztenciális jelleggel kielégíthetőséget mond ki; ha egy feltétel elé helyezzük, akkor nem fog számítani, hányféleképpen elégíthető ki a feltétel, csak hogy kielégíthető-e. Végül a **forall** kvantor azt mondja ki, hogy az első feltételt kielégítő bármely elemekre a további feltételek is kielégíthetőek. A három kvantor valamelyikének hatáskörén belül megkötött Drools változó a hagyományos kvantor szemantikának megfelelően nem „látszik ki" a kvantoron kívülre. A fenti példában megfigyelhető a **not** kvantor alkalmazása.

A kvantorokhoz hasonló működésű az akkumuláció, amelyre a komplexesemény-feldolgozásban különösen gyakran van szükség. Az akkumuláció a megadott feltételeket kielégítő összes változóbehelyettesítésből aggregál egyetlen értéket, amely valamilyen szám típus (pl. `java.lang.Integer`, `java.lang.Number`) lesz, és további feltételeket lehet kimondani rá. Az aggregátor függvények listája kiterjeszthető, de a legfontosabb esetek (**average**, **min**, **max**, **sum**, **count**, …) támogatása beépített. A Drools dokumentációban szereplő alábbi példa azt mutatja be, hogyan lehet azon rendeléseket kiválasztani, amelyeknek a tételei 100 fölötti összértékűek.

```
$order : Order()
$total : Number( doubleValue > 100 )
  from accumulate( OrderItem( order == $order, $value : value ),
    sum( $value ) )
```

Amint a példakódok is mutatják, az akció blokkba egyszerű Java utasítások írhatóak; ezen felül a Drools biztosít néhány különleges műveletet. Az **insert**(obj) és **retract**(obj) parancsok a paraméterül kapott objektumot beszúrják ill. kiveszik a WM-ből. Az **insertLogical**(obj) logikai beszúrást hajt végre, amely visszavonódik, ha a szabály feltételrésze később érvényét veszti. A korábban tárgyalt okok miatt az attribútumok manipulációja (vagy ilyen hatással járó metódushívások) után **update**(obj) hívandó, hogy a Drools is értesülhessen a változásról. Ennek egy elegánsabb alternatívája a **modify**(obj){metódus1(), ..., metódusK();} blokk, amelynek a végén ez automatikusan megtörténik; a blokkon belül pedig az objektum metódusai kényelmesebben hívhatóak, 'obj.' prefix nélkül. Az akció blokk további különlegessége, hogy elérhető egy `kcontext` nevű változó, amelyen keresztül a Drools runtime különféle szolgáltatásai (pl. globális adatok) hozzáférhetőek.

Nem beszéltünk még a szabály opcióiról, amelyeket a **when** kulcsszó előtt lehet megadni. Az egyik legfontosabb opció a **salience**, amelyet egy egész szám követ. A megadott szám lesz a szabály prioritása (alapértelmezésben 0), és több aktivált szabály esetén a legmagasabb prioritású fog tüzelni.

Egy másik fontos opció az alapértelmezésben hamis értékű **no-loop**. Volt már arról szó, hogy a szabálypéldány végrehajtásakor a saját aktivációja automatikusan eltűnik az agendáról, hogy azok a szabályok se kerüljenek „végtelen ciklusba", amelyek nem érvénytelenítik saját előfeltételeiket. Ha azonban az érintett szabály módosítja a feltételrészében szereplő valamelyik objektumot, akkor a Drools újra észleli az aktiváció érvényességét és visszateszi az agendára. Ez kerülhető el **no-loop true** megadásával. Tartsuk észben, hogy a több szabályon keresztül lezajló kölcsönös rekurzió ellen ez sem hatásos. Szabálycsoportok definiálása (**ruleflow-group**) után a **lock-on-active** opció azt garantálja,

hogy amíg a csoport aktív, a szabály adott változóbehelyettesítéssel nem aktiválódhat többször. A legáltalánosabb megoldás az, ha minden szabály valamilyen módon érvényteleníti saját aktivációját (pl. kockacukrot tesz a szilvába, amely így már nem lesz üres).

A nyelv részletesebb, pontosabb, példákban gazdagabb leírása az online dokumentációban olvasható.

3.3 Ismétlés: Drools fejlesztőkörnyezet és hibakeresés

A Drools Eclipse-be épülő fejlesztői környezetet biztosít. A szabálydefiníciós fájlokhoz fejlett editor támogatás tartozik (pl. környezetérzékeny kódkiegészítés), ezen kívül projekt varázslóval létrehozhatóak Hello World jellegű projektek (a tudásbázist felépítő kódot innen célszerű eltanulni).

A fejlesztőkörnyezet legfontosabb feladata a hibakeresés támogatása. Ehhez a tevékenységhez nagyon fontos segítséget nyújt az Audit nézet (Window > Show View > Other… > Drools > Audit), ahol visszamenőleg áttekinthetőek a végrehajtott szabálypéldányok, az elvégzett WM módosítások, és a hatásukra megjelent vagy megszűnt aktivációk. Ehhez azonban be kell kapcsolni a loggolást, majd az Audit nézetet az elkészült log fájlra irányítani. A loggolás a session létrejötte után bekapcsolható (Drools 5.x esetén) a `KnowledgeRuntimeLoggerFactory.newFileLogger(session, "logFileNeve.log")` metódussal, illetve (Drools 6.x KIE API felett) a Drools session létrehozására is használt KieServices objektummal `getLoggers().newFileLogger(kSession, "logFileNeve.log")` módon. A visszakapott logger objektum a program befejeződésekor lezárandó (`close()`).

Ha töréspontnál (pl. a `fireAllRules()` hívás előtt, vagy egy Drools szabály RHS részében) megállítjuk a programunkat, akkor további információkat kaphatunk a pillanatnyi állapotról. Az Agenda nézet a conflict set tartalmát mutatja (csak Drools 5.x alatt működik jól), a Globals nézet a globális objektumokat, a Working Memory pedig értelemszerűen a WM-beli tényeket. Ha Java töréspontnál állunk meg, akkor a Drools debug környezetnek nincs tudomása a futó Drools BRE példányról, ezért először a Java debuggolásnál megszokott Variables nézetben ki kell jelölni a Drools sessiont, és csak ezután töltődnek fel a Drools debug nézetei tartalommal. Ha azonban egy szabály akció részében elhelyezett töréspontnál állunk meg, akkor azonnal használhatóak ezek a nézetek, és a Variables nézet a Drools változókat mutatja. A szabály RHS-ben elhelyezett töréspontok csak akkor működnek, ha a programot Java Application helyett Drools Applicationként indítjuk, debug módban.

3.4 Drools Fusion események

A Drools Expert alapelemeihez képest a Fusion legfontosabb kiterjesztése az esemény fogalma. Az esemény egy speciális, időpecséttel ellátott tényobjektum a munkamemóriában, amely a megfigyelt rendszer valamely elemi jelenségéről tájékoztat, tehát egy alapvető állapotváltásról vagy tevékenységről. (A korábbi terminológia szerint tehát az atomi eseménynek felel meg; nem keverendő a komplex esemény fogalmával, amely egy vagy több elemi esemény és esetleg egyéb tények kapcsolatát vizsgáló Drools minta.)

Események esetén különleges elvárás / konvenció, hogy ha egy attribútumot egyszer beállítottunk null-tól különböző értékre, akkor onnantól nem szokás már megváltoztatni – hiszen az esemény már „megtörtént”. Ellenben kifejezetten támogatott viselkedés, hogy bizonyos mezőket az esemény beszúrásakor üresen hagyjunk, és később Drools szabályok által töltsünk ki.

A Drools Fusion pontszerű és időintervallum jellegű eseményeket is támogat. Így az eseménykezdetet jelző időbélyeg mellett opcionálisan időtartam is megadható (ennek hiányában pontszerű az esemény, vagyis időtartam=0). Mindkét időparamétert az eseményobjektum egy-egy mezőjéhez kell rendelni; ennek megfelelően eseménytípusonként egy alábbihoz hasonló deklarációt kell tenni az eseményt reprezentáló Java osztály jellemzésére:

```
declare TaszkLefutasa
  @role( event ) // ez jelzi, hogy eseményt reprezentál az osztály
  @timestamp( inditasDatuma ) // időbélyeg attribútum neve (opcionális)
  @duration( elteltIdo ) // hossz attribútum neve (opcionális)
  @expires( 4m30s ) // minimálisan elvárt élettartam, ld. később (opcionális)
end
```

Események bevitelekor az esemény kezdetét az időbélyeg attribútumban kell megadni. Ha ilyen attribútumot nem deklaráltunk, a Drools automatikusan társítja az eseményhez a beszúráskori időt. ennek ellenére célszerű az időbélyeget attribútum formájában kézben tartani, mert így részletesebb vizsgálatoknak vethető alá.

Az ilyen módon definiált eseménykezdet és –befejezés alapján vizsgálhatóak majd a két esemény között fennálló intervallumlogikai feltételek (ld. 2.1 szakasz) a szabályok feltételrészében. Példaként nézzünk meg egy feltételrészt:

```
$boru: Uzemmod(uzemAllapot == „Ború”)
exists Uzemmod(uzemAllapot == „Derű”, this after[6s, 2m30s] $boru)
```

Ez a feltétel a Ború üzemmód fennállásáról értesítő eseményekre akkor illeszkedik, ha jön még rá (legalább hat másodperccel, legfeljebb két és fél perccel a Ború befejezte után kezdődő) Derű.

Az időbeli paraméterek lehetnek negatívak is, ill. esetenként elhagyhatóak. Például `$a before[0s,15s] $b` jelentése, hogy `$a` befejeződése megelőzi `$b` kezdetét, de legfeljebb 15 másodperccel; `$a before[15s] $b` esetén legalább 15 másodperccel előzi meg (nincs felső határ); végül `$a before[-5s] $b` jelentése, hogy `$a` vége legfeljebb 5 másodperccel késheti le `$b` kezdetét.

A temporális operátorok (intervallumlogikai viszonyok) teljes listája és szemantikája a dokumentációban található; néhányat már láthattunk illusztrálva (ld. 2.1 szakasz 3. ábra).

3.5 Eseményfolyam mód (STREAM mode)

Az események intervallumlogika alapján történő összekapcsolhatósága révén lehetővé válik az eseménynaplók offline feldolgozása. Az online eseményfeldolgozást (vagyis amikor valós időben történő eseményfolyamról van szó) támogató további szolgáltatások azonban csak egy speciális végrehajtási mód bekapcsolása után lesznek elérhetőek. A Drools végrehajtó motor STREAM módban történő futtatásához a tudásbázis konfigurációját ennek megfelelően kell beállítani. A beállítás módja Drools 5.x esetén:

```
kConfig.setOption(EventProcessingOption.STREAM);
```

Ha Drools 6.x-et használunk, akkor a korábban bemutatott `kmodule.xml` állományban a `kbase` bejegyzést kell kiegészíteni egy `eventProcessingMode="stream"` attribútummal.

Az eseményfolyam mód esetén be kell tartani az időrendi követelményt, vagyis az eseményeket csak az időbélyegeik sorrendjében szabad beszúrni (ha nem deklarálunk időbélyeget, ez automatikusan garantált). Ezért cserébe számos szolgáltatást nyújt a rendszer az ún. *eseményfolyam-óra* segítségével. Ezen óra értékét tekinti a Drools a jelenlegi pillanatnak (ezért tesztelési célból a valós óra helyettesíthető manuálisan léptetett órával); például ezen óra alapján dátumozza a kitöltetlen időbélyeggel beszúrt esemény objektumokat.

Eseményfolyam módban a Drools az óra alapján automatikusan *késlelteti* bizonyos feltételek kiértékelését, amíg azokhoz jövőbeli eseményekre lenne szükség, hiszen nem feltételezi, hogy nem fognak a jövőben ilyen események beérkezni. Például az alábbi feltétel azon határozatokra illeszkedik, amelyekre 15 napon belül egy fellebbezés se érkezett; a Drools STREAM módban kivárja a 15 napot, mielőtt a mintát egy határozatra illeszkedőnek tekinti.

```
$h : Hatarozat()
not( Fellebbezes(hatarozat == $h, this after[0d,15d] $h ) )
```

Eseményfolyam módban lehetőség nyílik ún. *csúszóablak*okat használni. Az objektumminta végéhez hozzáfűzött `over window:time(időtartam)` ill. `over window:length(darabszám)` megszorítással az objektummintára illeszkedő események közül a (idő, darabszám szerinti) legutóbbiak választhatók ki. (Vegyük észre, hogy az „utóbbi öt percben” fogalom (időablak) az eseményfolyam-óra révén válik a Drools számára értelmezhetővé.) Az alábbi minta például akkor illeszkedik, ha az utóbbi 30 bejelentkezés közül egy se sikeres:

```
not(
  $la: LoginAttempt($s: success) over window:length(30)
  eval ($s == true))
```

A darabszám szerinti időablaknál különösen figyeljünk oda, hogy csak a tényobjektum osztálya és az esetleges konstans értékre lekötött mezői tartoznak hozzá az ablak definíciójához, a többi szűrőfeltétel csak a darabszám szerinti ablakolás után lesz figyelembe véve. Ha a fenti példában az objektummintába írtuk volna be a `success` mező igazságértékének vizsgálatát, akkor az utolsó 30 sikeres bejelentkezés sorára vonatkozott volna, amely csak akkor lenne üres, ha sose történt sikeres bejelentkezés. Másfelől sajnos ilyen módon a Drools csúszóablakaival nem lehetséges egy `$u: User()` felhasználóhoz tartozó utolsó 30 lekérdezést vizsgálni, hiszen a felhasználóval történő összekapcsolás csak egy utólagos szűrés lenne.

Újabb Drools verziókban nevesített, újrafelhasználható ablakok is deklarálhatóak **`declare`** `window ablaknév …` **`end`** szintaxissal, ilyenkor az objektummintában **`from`** `window ablaknév` használandó.

A csúszóablakok egyik legfőbb felhasználása az akkumuláció kifejezések leszűkítése. Az alábbi minta azon szerverekre illeszkedik, amelyeknek az átlagos CPU kihasználtsága az elmúlt két percben a hozzájuk konfigurált riasztási küszöb felett volt:

```
$s: Server($ct: cpuTreshold)
$ua: java.lang.Number (doubleValue > $ct) from accumulate (
  ResourceRecord(server == $s, $util: cpuUtilization) over window:time(2m),
  average($util))
```

Az eseményfolyam-feldolgozó motorok gyakorlati használatának egyik fontos feltétele, hogy nem kell a teljes múltat eltárolniuk, az információtömeg már érdektelenné vált részei eltávolíthatóak. Eseményfolyam módban a Drools fontos szolgáltatása az *életciklus-kezelés*: az intervallum-összehasonlítások és idő alapú csúszóablakok alapján automatikusan meghatározza, hogy melyik típusú eseményt mennyi ideig kell megőriznie, és a lejáratuk után eldobja az eseményeket. A fenti példákban – ha máshol nincs ezekre az eseménytípusokra szükség – a `ResourceRecord` eseményeket két percig, az `Uzemmod` eseményeket két és fél percig őrzi meg, utána eltávolítja a munkamemóriából. Magasabb élettartam is garantálható egy `@expires(idő)` sorral az eseményobjektum deklarációjában.

Érdekes megfigyelés, hogy a Drools bizonyos verzió események beszúrásakor automatikusan eltüzelik az aktivált CEP szabályokat. Ennek ellenére nem célszerű kizárólag erre a hatásra hagyatkozni – inkább hívjuk meg minden esetben a a `fireAllRules()`metódust is.

3.6 Telepítés, dokumentáció és problémamegoldás Drools verziók szerint

Jelen segédlet eredeti verziója JBoss Drools 5.2.1.FINAL verzió alapján készült. Az időközben megjelent újabb verziók (pl. 5.6., 6.2) hoztak nyelvi kiterjesztéseket, azonban bizonyos esetekben instabilitást és egyéb nehézségeket is.

- **Drools 6.2:** Összességében jól használható a mérés feladataihoz, ám debug közben az Agenda nézet nem működik jól (mutatja az egyes aktivációk szabályát, de a paraméterértékek nem látszanak).
    - Eclipse Update Site: http://download.jboss.org/drools/release/6.2.0.Final/org.drools.updatesite/
    - Dokumentáció: http://docs.jboss.org/drools/release/6.2.0.Final/drools-docs/html_single/
- **Drools 5.3**: Java 8-cal már nem kompatibilis, Java 7-re kell szorítkozni. Cserébe jól működik a debugger.
    - Eclipse Update Site: http://download.jboss.org/drools/release/5.3.0.Final/
    - Dokumentáció: https://docs.jboss.org/drools/release/5.3.0.Final/drools-expert-docs/html_single/
- **Drools 5.2.1**: Java 7-tel sem kompatibilis, Java 6-ra kell szorítkozni. Viszont talán a legstabilabb változat.
    - Eclipse Update Site: http://download.jboss.org/jbosstools/updates/stable/helios/

- Dokumentáció: https://docs.jboss.org/drools/release/5.2.0.Final/drools-expert-docs/html_single/

A használt Drools verziótól függően előfordulhatnak az alábbi visszatérő problémák:

- JDT-t hiányol: explicit függőségként fel kell venni. Ez egyszerű Java alkalmazás esetén a *Java Build Path* beállításánál *External Jar* felvételével történhet, az Eclipse telepítésből kitallózva a `plugins` mappában található `org.eclipse.jdt.core` kezdetű .jar fájlt. Futtatókörnyezet (pl. OSGi) használatakor az arra jellemző módon kell a függőséget jelölni.
- Futtatáskor a hibakimenetre SLF4J hibaüzenet kerül: ahogy a hibaüzenet is írja, fel kell venni függőségként az egyik SLF4J megvalósítás .jar fájlját.
- Nem sikerül töréspontot létrehozni a szabályok akció részében: a DRL fájl fejlécébe beszúrandó a `dialect "mvel"` direktíva.

3.7 Tanácsok

Mivel a Drools nem szálbiztos, célszerű sorosítani és egy szálról bevezetni az eseményeket. Erre használhatóak például a folyamat implementációban egyébként is alkalmazott üzenetsorok. Ügyeljünk arra, hogy a Drools inicializálása, a szabályok létrehozása és az események bevezetése történjen egyazon a szálon.

Bár a Drools Fusion képes automatikusan időbélyegezni az eseményeket, érdemes azokat mégis saját időbélyeg attribútummal ellátni, amelyet például `new java.util.Date()` vagy `java.lang.System.currentTimeMillis()` visszatérési értékével lehet feltölteni a ténybázisba szúrás előtt. Ennek a megoldásnak az az előnye, hogy később a szabályokban az időbélyeg explicit hivatkozható; két esemény közötti időbeli eltérés könnyen számolható, vizsgálható. Természetesen amit ki lehet fejezni intervallumlogikai operátorokkal, azt (legalább eseményfolyam módban) mindenképp úgy érdemes.

Az események automatikus elavulása veszélyeket is rejt. Egy `not(X(this before[0s,10s] $y ) )` jellegű minta esetén X-hez kellően magas expires értéket kell rendelni, y-t pedig egy szűk (legalább 10 másodperccel rövidebb) időablakból kell venni; különben X elavulásakor és a munkamemóriából való automatikus eltávolításakor a mintának hamis illeszkedése keletkezik.

Amíg az eseménydömpingből adódó skálázódási problémák nem teszik szükségessé, nem érdemes sok energiát fordítani az események életciklusának röviden tartására. Ha ugyanis arra törekszünk, hogy minden atomi esemény csak éppen addig maradjon a WM-ben, ameddig még releváns, ezzel néha szükségtelenül megnehezíthetjük a komplexesemény-minták megfogalmazását. Bizonyos esetekben erre a Drools Fusion nem is alkalmas, érdemes lehet pl. Esper-rel próbálkozni helyette. Ezért a gyakorlatban sok esetben nyugodtan meg lehet adni egy kellően magas `@expires` értéket – ez a laborfeladat teljesítéséhez bőven elég lesz, és az előző bekezdésben részletezett problémán segíthet.

A komplex eseményminták kisebb egységekre bontására használjuk bátran az `insertLogical` kulcsszót. Ezzel egy részminta eredményét szabadon felhasználható módon beilleszthetjük a ténybázisba, ahonnan automatikusan el lesz távolítva a feltételek megszűntekor (pl. ha az érintett események élettartama lejár). A nyelv (Drools 5.2) korlátozásai miatt ilyen jellegű dekompozícióra mindenképpen szükség lehet, ha egynél több objektummintából álló összetett feltételt szeretnénk aggregálni.

A munkamemóriába nem csak az eseményeket lehet bevezetni; nyugodtan tarthatunk benne egyéb üzleti objektumokat, amelyekkel az események a szabályok feltételrészében összekapcsolhatóak. Ha azonban egy objektumot nem használunk a szabályok feltételrészeiben, nem szükséges csak amiatt a WM-be tenni, hogy utána az akciórészben hivatkozhassunk rá. A szoftver futási idejű környezetét (pl. GUI elemek) elérhetővé tehetjük globális objektumként; egy .drl fájlban deklarált globális objektumokhoz a session futtatásakor érték rendelhető, amelyet a szabályok akciói a `kcontext`-en keresztül elérhető Drools runtimetól kérdezhetnek le.

Végül a legfontosabb tanács, hogy minden kérdéses pontról érdemes először a Drools dokumentációban tájékozódni: http://www.jboss.org/drools/documentation

4 A mérés elvégzése

A mérés során a korábban implementált munkafolyamat szolgáltatja majd az eseményeket, amelyeket a Drools Fusion segítségével megfigyelhetővé kell tenni. Eseményminták definiálása és az eseményfolyam feldolgozása útján fogjuk felügyelni a folyamat végrehajtását. Az elvégzendő feladatok tehát:

- Kiindulás a korábbi Java munkafolyamat-implementációból. Bizonyos előfeltételeknek teljesülnie kell, így az implementáció szükség esetén módosítandó:
  - futhasson egyszerre akárhány folyamatpéldány;
  - előnyös (nem kötelező), ha a folyamatpéldányok automatikusan, kézi léptetés nélkül (is) le tudnak futni, legalább tesztelési céllal;
  - a tesztelhetőség érdekében mindenképpen gondoskodni kell arról, hogy a folyamatpéldányok végrehajtása bizonyos pontokon mesterségesen késleltethető legyen;
  - végül legyen az egyes folyamatpéldányokhoz egyedi azonosító rendelve, amely a folyamaton végighaladás közben nem változik (beleértve elágazásokat és párhuzamosságot); ez lehet maga a munkadarab objektum is.
- A munkafolyamat felműszerezése, hogy szolgáltasson „x azonosítójú folyamatpéldány elkezdte / befejezte az y taszkot” jellegű, pillanatszerű eseményeket.
- Az események bevezetése egy eseményfolyam módban futtatott Drools példányba. Ügyeljünk arra, hogy a Drools nem szálbiztos!
- Drools szabály készítése néhány érdekes komplex esemény kezelésére. Az alábbi listáról az első két feladat kötelező az elégségeshez; a maradék háromból kettő megoldása elegendő a jeleshez:
  - egy adott join csomópontnál az egyik ágra sokáig várakozott a másikkal már végzett processz;
  - egy adott elágazás egyik kimenetét az utóbbi időben sokkal több folyamatpéldány választotta, mint a másikat;
  - egy adott taszk a legutóbbi néhány végrehajtásához képest kiugróan lassan futott most le;
  - egy adott taszk nagyon sok példányban van most egyszerre folyamatban (ha lehetséges egy taszkon belül párhuzamos végrehajtás) vagy nagyon sok példánya várakozik egyazon taszkra (ha egy taszk egyszerre egy folyamatpéldányon hajtható végre);
  - a fenti szabályok közül legalább az egyik általánosítása, hogy „egy adott” helyett „akármelyik” folyamatelemre (join, elágazás, taszk) működjenek, tehát maga a szabály ne legyen konkrét folyamatelemhez kötve.

  (Természetesen a megoldás során tetszés szerint használhatóak segédtények, segédszabályok, `insertLogical`-lal származtatott tények, globális objektumok, stb.)
- Demonstrálni, hogy a definiált CEP szabályok a munkafolyamat megfelelő lejátszásával elsüthetőek.

5 Ellenőrző kérdések

A beugró kérdések (melyek nem feltétlenül az alábbi listáról lesznek kiválasztva) kizárólag az ebben a dokumentumban leírt ismereteket fogják visszakérdezni, azonban a mérés sikeres elvégzéséhez feltétlenül célszerű a Drools dokumentáció használata is.

1. Mit nevezünk egy szabály aktivációjának? Mely elemekből épül fel az aktiváció Drools esetén?
2. Mi a conflict set (agenda, napirend)?

3. Milyen következményekkel jár, ha egy szabályalkalmazás nem érvényteleníti a saját aktivációját? Mi a teendő ilyen helyzetben?
4. Mire való az Eclipse-es Drools környezet Audit, Agenda ill. Working Memory nézete?
5. Miben különböznek a Drools események az egyéb tényektől, eseményfolyam módban és azon kívül?
6. Mi alapján dönti el a Drools, hogy egy WM-be beszúrt tényobjektum eseménynek minősül-e?
7. Az időadatok szempontjából milyen kétféle eseményt támogat a Drools? Tekinthető-e az egyik a másik speciális eseteként?
8. Mire valók a temporális (intervallumlogikai) operátorok?
9. Milyen alkalmazási helyzetekre való a Drools eseményfolyam (STREAM) módja, milyen feltételezéseket tesz, és milyen többletszolgáltatásokat nyújt?
10. Eseményfolyam módban hogyan kezeli a Drools az események életciklusát? Miért van erre szükség?
11. Milyen csúszóablakokat támogat a Drools, mi a szerepük, mikor alkalmazhatóak?