Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar

Dávid István

Modellalapú fejlesztési módszer komplex események feldolgozásához

Konzulens:
Gönczy László
Méréstechnika és Információs Rendszerek Tanszék

Budapest, 2011

Kivonat

Az eseményvezérelt architektúrák eseményeinek alkalmas feldolgozásával olyan (üzleti, diagnosztikai) információkat állíthatunk elő, amelyeket vezérlési és döntési helyzetekben jól hasznosíthatunk. Az információk kinyerésének alapja az eseményfolyamon értelmezett *mintafelismerés*. Az események, mint különböző forrásból származó adategységek önmagukban is hordozhatnak értéket, sok esetben azonban az érdekes minták nem köthetők csak egy forráshoz, hanem több, különböző típusú, eseményből állnak össze. Ezek a *komplex események*, amelyek az atomi eseményeknek egy logikai összekapcsolását jelentik, kiegészítve ezt olyan információkkal, mint például a sorrendiség, időbeliség, gyakoriság, stb. [1]
A komplex események feldolgozása – szemben a statisztikai és adatbányászati technikákkal – a releváns jelenségek valósidejű, legfeljebb másodpercnyi nagyságrendű késleltetésű előrejelzését célozza meg az aktuális információk alapján. A módszer eredményeit olyan fontos területek hasznosítják eredményesen, mint az *informatikai rendszermonitorozás és -diagnosztika*, a *pénzügyi előrejelzés*, vagy az *online csalások felderítése (fraud detection)*. [2]

A komplex események komponenseken átívelő leírása nem triviális feladat. Az eseményfeldolgozó rendszerek leírónyelveivel platformközeli szinten fogalmazhatunk meg eseménymintákat, aminek hátránya, hogy a minták összetettségét korlátozza, hiszen a fejlesztés, tervezés egy adott szinten már áttekinthetetlenné és kezelhetetlenné válik. [3]
Egy hatékony fejlesztési módszerrel szemben erős követelmény, hogy szakterülettől függetlenül tegye lehetővé az eseményminták definiálását úgy, hogy a fejlesztés időigénye ésszerű keretek között maradjon. További fontos követelmény lehet – tekintettel a megcélzott szakterületek potenciálisan nagy méretére – hogy automatizált eljárásokkal támogatható legyen egyrészt az érdekes minták definiálása, másrészt azok formális verifikációja.

A probléma megoldására egy olyan *modellvezérelt* módszert dolgoztam ki, ami teljesíti ezeket az általános kritériumokat. A módszert egy *modellezőnyelv*vel és egy alkalmas *fejlesztőeszköz*zel támogattam. A modellvezérelt megközelítés lehetővé teszi az eseményminták áttekinthetőbb definiálását és a *formális verifikáció*t, ami által olyan magas szinten definiált követelményeknek való *megfeleltetés biztosítható*, amelyeket a szakterület szabványai, jogszabályai definiálnak. (Mint például a COBIT az IT-biztonság szakterületén.)
A módszer *szakterület-független* megoldást ad a problémákra, a szakterület információi (eseményforrások, kapcsolatok, metrikák) egy, a területet leíró ontológiában, a *szemantikus tudásbázis*ban találhatók meg. A gyakorlatban jellemző, hogy a kézi modellezés időigényes lehet és nem biztos, hogy teljes eredményre vezet. (Például nagyméretű vállalati topológiák esetén.) Az ontológia alapú megközelítés lehetővé teszi egy olyan *automatizmus* kialakítását, amely ilyen esetekben a szakterület-specifikus információk alapján *javaslatokat tesz* a modellezéssel kapcsolatban, így segítve a tervező munkáját.

A kialakított fejlesztőeszköz intuitív és gyors modellezést biztosít, köszönhetően a szöveg alapú jellegnek és a kiegészítő funkcióknak (intelligens kódkiegészítés, szintaktikai validáció). Az eredmények tesztelését az *információs rendszerek monitorozásának szakterületén* végeztem, egy jellegzetes nagyvállalati környezet emuláló infrastruktúrán. A modellezőkörnyezet (és nyelv) korszerű Eclipse alapú technológiákra épül. Az ellenőrzött modellekből az Esper nyílt forráskódú eseményfeldolgozó platformhoz állít elő forráskódot.

Abstract

By adequate processing of events observed in event-driven architecture, valuable information can be extracted and used in controlling and decision situations. The technique of extracting information is the *pattern matching* defined over the event stream. Events, as atomic data-objects arising from different event sources, may be valuable by themselves as well, but in most cases the interesting patterns cannot be associated with one single source, but they consists of multiple events possibly of different type, format, characteristic. This structure, called *complex event*, is a logical linking of atomic events, extended by information such as ordering, timing, frequency, etc. [1]
Complex event processing, rather than statistical or data mining techniques, aims at real-time prediction of relevant phenomenon, based on current information, with a slight delay, at most in the order of seconds. Its accomplishments are employed successfully by the domains of *IT-infrastructure monitoring and diagnosis, financial prediction*, or *fraud detection*. [2]

Depicting complex events over multiple sources (usually of different type) is not a trivial task. Languages of event processing tools allow defining patterns on a platform-specific level. This approach limits the obtainable complexity of event pattern definitions, since with the growing codebase it gets harder to handle by the developer. [3]
A strong requirement against an effective development technique is to be able to depict event patterns in a domain-independent way, over multiple components or systems, while the time-cost of this task must remain within reasonable limits. A further requirement can be – considering the potential large scale of the targeted domain – the possible automation of determining valuable (interesting, relevant) event patterns and the formal verification of them.

As a solution to the problem I developed a *model-driven* technique that satisfies these requirements. I also created a *modeling language* and an *integrated development environment* (modeling tool) to support the technique. The model-driven approach provides a comprehensive way for defining patterns and enables application of *formal verification* methods, which can be used for *assuring high-level conformance* to domain-specific standards or law. (E.g.: COBIT in IT-security domain.)
The method offers a *domain-independent* solution for the problem, domain-specific information (event sources, the relationship among them, the defined metrics and measurements) are located in an ontology called *Semantic knowledge base*. In practice, manual model building turns out often too long and imprecise, it may be fall short of complete coverage of the problem. (E.g.: in the case of enterprise architectures.) The ontology-based approach offers the possibility to develop *automatisms* they can *aid the task of modeling* by automatic intelligent *recommendations* about the model's elements.

The tool allows intuitive and rapid modeling, due to its text-based nature and the usual IDE-functions (like intelliSense, or design-time syntactic validation). The results have been tested in the domain of IT-system monitoring, on an infrastructure emulating a typical enterprise environment, because of the explicit objective the solution to be effective and easy-to-adopt in the practice. The modeling language and the development environment are built on up-to-date Eclipse-based technologies. The IDE generates ready-to-run code from models for the open-source event processing platform Esper.

Tartalomjegyzék

1. Bevezetés

Az informatikai rendszerek fejlődésével és növekedésével arányosan nőtt azok komplexitása is. Egyre kevesebb homogén rendszerrel találkozunk, ehelyett az olyan megoldások kerülnek előtérbe, mint a szolgáltatás-orientált architektúrák, illetve az ezzel szorosan összekapcsolódó esemény-vezérelt architektúra, amelynek vezérlése a rendszerkomponensek állapotváltozásaira, illetve az azok által kiváltott eseményekre épül.

Motiváció

Az esemény-vezérelt architektúra eseményeinek megfelelő feldolgozásával olyan információkhoz juthatunk, amelyek vezérlési és döntési helyzetekben hasznosak bizonyulhatnak, valamint a munkafolyamatok ezen pontjainak automatizálhatóságát segítik. [4] (OMG)

Az információk kinyerésének módja a tervezési szempontból releváns események mintáinak azonosítása az eseményfolyamon.

Felismerve, hogy a nagy, elosztott rendszerek releváns eseményei általában nem korlátozhatók egyetlen önálló forrásra, hanem több, esetenként más formátumot és adatstruktúrát használó rendszeregység eseményeinek együttes bekövetkezéséből adódnak, előtérbe került az úgynevezett *komplex események* feldolgozásának technikája.[1] [1] [2] [5] [6] [7]

A komplexesemény-feldolgozásra alapozó megoldások mára a gyakorlati, ipari projektekben is meggyőző referenciákkal bírnak. Olyan területek hasznosítják eredményesen a módszert, mint az informatikai infrastruktúra monitorozása, a szenzorhálós előrejelző rendszerek (*Tsunami Warning System Japan*), forgalomirányítás, az orvosi informatika (járványok előrejelzése: *Google Flu Trends*), vagy az automatizált tőzsdei kereskedő alkalmazások (*algorithmic trading*). [8] [9] [10] [11]

Problémafelvetés

A felhasználási területek széles spektruma ellenére a komplex események feldolgozására szakosodott eszközök evolúciós útjának korai szakaszában járunk. A nagy gyártók (IBM, Microsoft, Oracle) természetesen rendelkeznek saját fejlesztésű eszközökkel, de ezek jellemzően saját platform-specifikus nyelvet kínálnak a fejlesztéshez (amelyek egyébként nem valósítanak meg közös nyelvi szabványt), aminek kézenfekvő hátránya, hogy a célterület és –infrastruktúra méretével arányosan növekvő érdekes eseményminták mennyisége egy alacsony szintű nyelv esetén nehezen karbantartható kódbázist eredményez. [12] [13]

Jelen dolgozat szempontjából azonban találunk relevánsabb problémákat is.

A fontos eseményminták egyértelmű megfogalmazása nagy pontosságú körülírást követel. A legegyszerűbb esetben is egy-egy minta esetén meg kell határoznunk, hogy mely eseményforrások eseményeit, milyen metrikák mely értékei alapján szeretnénk figyelni. Látható, hogy egyrészt sok olyan paraméter van a mintákban, amelyek a fejlesztési időben nem feltétlenül ismertek, vagy későbbi mérések visszacsatolásaként módosulhatnak; másrészt pedig felmerülhet, hogy mit is nevezünk érdekes eseménymintának, és hogy azokat milyen

[1] Fontos megérteni, hogy nem az események komplex feldolgozásáról, hanem a komplex eseményeknek a feldolgozásáról beszélünk. A dolgozat a továbbiakban ezért *A magyar helyesírás szabályai* legutóbbi érvényes, tizenegyedik kiadásának 138. pontja alapján a „komplexesemény-feldolgozás” írásmódot követi. [62]

metrikákkal mérjük. A problémára a megoldást a szakterület szabványaiban, mérnöki gyakorlatában és szabályozásában találjuk, ezek ugyanis magas szinten kijelölik a célrendszerek működésére vonatkozó irányelveket. (Az informatikai infrastruktúrák üzemeltetésének területén jó példa erre a COBIT szabvány által leírt követelmények. Például: Felhasználják-e a felügyeleti rendszer adatait valamelyik munkafolyamatban? Létezik-e minden kritikus szolgáltatásnak tartaléka?)
A jelenlegi ipari eszközök nem képesek az ilyen magas szinten definiált, szakterület-specifikus információk rugalmas kezelésére (a „*kritikus* szolgáltatás" túl általános ebben a formában), mert a platform-közeli nyelvhez képest pontosan definiált fogalmakkal, konkrét értékekkel kell körülírnunk az eseményeket. (Például: kritikus szolgáltatások felsorolása, egyértelmű metrika megkötése, stb.) Értelmezhetjük ezt úgy, mint a „*kódba égetett konfiguráció*" esetét, amikor bár a kezdeti állapotokat és célokat jól kielégíti a fejlesztett rendszer, a változó konfiguráció módosítása viszont már programozási munkát igényel.

A gyakorlatban jellemző, hogy egy-egy megcélzott szakterületen nagyszámú forrás eseményeit kell feldolgozni, nem ritka a több százas nagyságrendű mennyiség sem. A releváns minták leírása *ad-hoc* módon történik, senki nem garantálja azonban, hogy az így kialakított eseménymodell konzisztens lesz – például hogy a nagyszámú bonyolult minta nem fedi egymást, vagy nem mondanak egymásnak ellent. Továbbá a terjedelmes szabványoknak, esetleg jogszabályoknak való megfeleltethetőség (*conformance*) is csak nagyon körülményesen oldható meg.

Dolgozatomban egy *modellalapú megközelítést* mutatok be az említett problémák megoldására.

Célkitűzés
Dolgozatom célkitűzései a következők:

- **Modellező formalizmus kialakítása, követelmények azonosítása.** A komplex események modellezéséhez szükséges egy olyan modellező nyelv kialakítása, amely egy megfelelő leíróerővel rendelkező metamodellből és az események dinamikáját leírni képes operátorokból áll. Ennek segítségével olyan eseményminták definiálhatók, amelyeket a szakterület információi alapján konfigurálva pontos eseménymodell áll elő.
  Az események detektálása minden esetben valamilyen üzleti célt szolgál (monitorozás, kereskedés). Ezeket az üzleti célokat a szakterület magas szintű követelményei tartalmazzák (például szabványok, jogszabályok), amelyek feldolgozása segít abban, hogy a céljainknak leginkább megfelelő eseménymintákat fogalmazzuk meg.
- **Szakterület-specifikus konfigurációk kezelése és elemzése.** A magas szintű szakterület-specifikus információk tárolásának hatékony kialakításával (szemantikus tudásbázis) és megfelelő formális módszerek alkalmazásával kimutathatóvá válnak egyrészt az eseménymodell konfigurációinak ellentmondásai, másrészt ellenőrizhető a szabványokkal, jogszabályokkal való megfelelőség. Ezen a szinten tehát a generált forráskód *szemantikus* problémái is detektálhatók.
- **Tervezési minták definiálása.** A tervezési minták irányt mutatnak a gyakorlati problémák megoldására. Mint minden szakterületnek, a komplexesemény-feldolgozásnak is megvannak a maga általános tervezési mintái. A kifejlesztett

modellezőplatform és a fejlesztőeszköz hatékony használata és képességeinek teljes kihasználása érdekében szükséges a már létező tervezési mintákkal való összehasonlítás és a hozzáadott eredményeket kihasználó új tervezési minták definiálása.

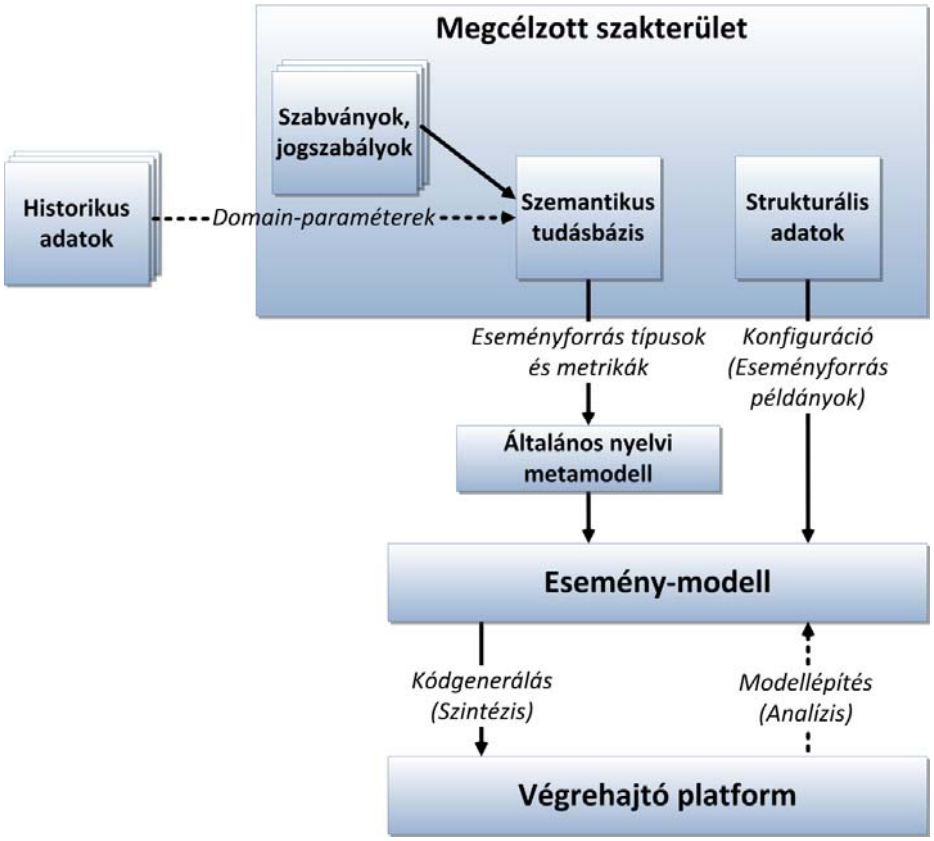
- **Gyakorlati alkalmazhatóság elemzése.** Az elméleti (kutatási) eredményeken túl fontosnak tartottam, hogy az eredmények a gyakorlatban is alkalmazhatók legyenek, ezért az alkalmazás módjára, illetve annak problémás pontjaira is rámutasson a dolgozat.
- **Fejlesztőeszköz implementálása.** A fejlesztőeszköz az eseményminták modellalapú leírását támogatja, majd a modellekből valamely eseményfeldolgozó platformon futtatható kódot generál. A modellalapú megközelítés lehetővé teszi magának az eseménymodellnek az ellenőrzését még a generált forráskód előállítása előtt.
- **Továbbfejlesztési irányok azonosítása.** A dolgozat jellegéből adódóan a fejlesztett eszközök csupán prototípus szinten készültek el, azonban ezek továbbfejlesztési lehetőségeire pontos iránymutatást adok.

Megközelítés

Az eseménymodellek kialakításához egy olyan általános modellezőplatformot hoztam létre, amely generikus és a szakterületet megcélzó elemekből áll össze nyelvi szinten. Utóbbiak egy szakterület-specifikus ontológiából (szemantikus tudásbázisból) nyerhetők ki. Az ontológia alapú megközelítés előnye az egyszerű adatbázissal szemben, hogy a komplex összefüggések felderítésére alkalmazhatók a szakértői rendszerekben is alkalmazott előreláncoló következtető módszerek, ezáltal kimutathatók az esetleges bonyolultabb összefüggések. Az így elkészült esemény-modell a célterület strukturális információival konfigurálva válik teljes értékű modellé.

A modellezés szöveg alapú megközelítéssel történik, aminek előnye a grafikus módszerrel szemben – ahogy erre a későbbi fejezetekben rámutatok – hogy gyorsabb modellalkotást tesz lehetővé.

A modellezést segítő eszköz olyan képességekkel rendelkezik, mint a szerkesztési-idejű szintaktikai ellenőrzés, az intelligens kódkiegészítés és a mentési-idejű automatikus kódgenerálás, amelyek nagyban megkönnyítik a tervező munkáját.



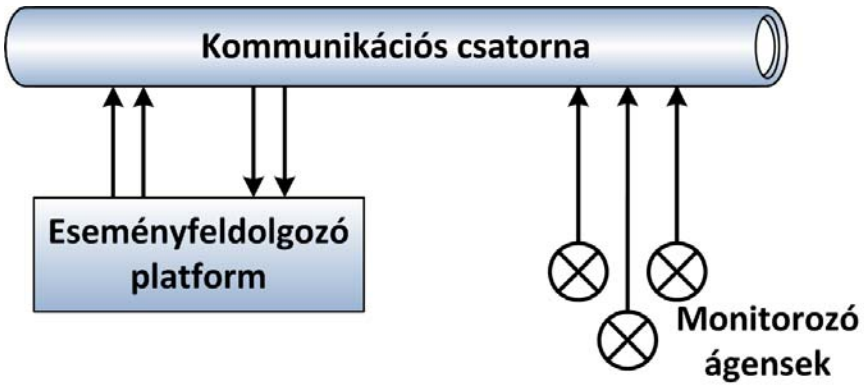
1. ábra – A dolgozatban bemutatásra kerülő megközelítés sematikus ábrája. A megcélzott szakterület historikus adatait, szabványait, jogszabályait egy szemantikus tudásbázisban fogjuk össze. Az eseménymodellt a strukturális adatokkal konfiguráljuk, amiből így a végrehajtó platformra telepíthető forráskód áll elő.

Technológiai szempontból a fejlesztőeszköz korszerű *Eclipse* alapú megoldásokra épít (*Xtext, XTend, Eclipse Modeling Framework*). A modellekből előállított forráskódokat a nyílt forráskódú *Esper* eseményfeldolgozó platformon teszteltem egy nagyvállalati infrastruktúrát emuláló környezetben, aminek aszinkron kommunikációját a *JBoss HornetQ* eszközzel biztosítottam. A dolgozat nem foglalkozik a megcélzott konkrét vállalati infrastruktúra adatainak a modellező platformra történő leképezésével, de hivatkozik a megfelelő dolgozatra, ami a HP konfigurációmenedzsment eszköze, az *UCMDB* adatainak programozott kinyerésével és transzformálásával foglalkozik, így a dolgozat az integrációs elemzés ipari referenciájaként ezt az eszközt jelöli meg a megfelelő helyen. [14] [15] [16] [17] [18] [19] [20]

2. Komplex események

Esemény alapú megoldásokkal, implementációkkal gyakran találkozhatunk informatikai rendszerekben, függetlenül azok méretétől, jellegétől, megvalósításától, vagy az alkalmazott szakterülettől. A beágyazott rendszerektől a személyi számítógépek operációs rendszerein át egészen a nagyvállalati és elosztott architektúrákig terjed a skála, amelyen az esemény-vezérelt megközelítés hasznos tervezési mintának bizonyul.

Az események jellemzően egy *forrás*hoz kapcsolódó információegységet jelentenek. Eseményforrás lehet minden olyan állapottal rendelkező rendszerkomponens, amelynek valamilyen mérhető paramétere van. A mérést automatizált ágensek végzik, az eredményeket a feldolgozóplatformra továbbítják.



2. ábra – Komplexesemény-feldolgozás sematikus modellje. A monitorozó ágensek eseményeket küldenek a csatornára, ami az eseményfeldolgozó platformra továbbítja azokat.

Szemantikus jelleg

Fontos viszont tudni az esemény szemantikus tartalmát, ami kétféle lehet:

- állapotleíró, vagy
- változás-vezérelt.

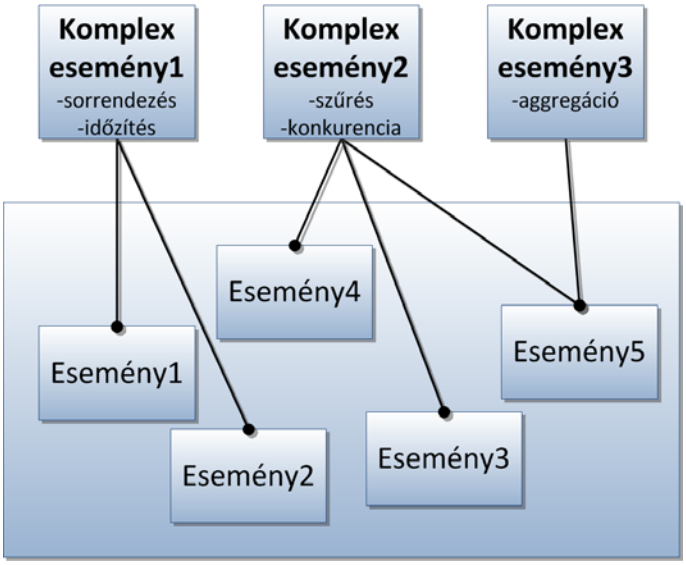
Az *állapotleíró esemény* a diszkrét idejű mintavételezés analógiájának tekinthető. A forrás meghatározott időközönként küldi az eseményeket. A mechanizmus legegyszerűbb ismert megvalósítása a *heartbeat*-jellegű monitorozás, ahol az esemény nem is tartalmaz több információt azon felül, hogy a forrás képes üzeneteket küldeni, tehát feltehetőleg el tudja látni a feladatát.
Változás-vezérelt eseményt akkor küld egy forrás, ha valamelyik releváns állapot-dimenziójában változás állt be.

Bár van rá explicit lehetőség, hogy e két típust megkülönböztessük, a dolgozatban erre *nem térek ki*. Ha nincs külön kiemelve, a dolgozat során eseményként hivatkozott atomi entitások olyan adategységeket jelentenek, amelyeket egy forrás *bizonyos időközönként*, *diszkrét idejű állapotleíró jelleggel* generál.

Komplex események

A *komplex események* az atomi események logikai összefogását jelentik, kiegészítve ezt a struktúrát az atomi események közötti *kapcsolatokkal*, mint például a sorrendiség, vagy az időzítés. Ezáltal sokkal absztraktabb és magasabb szintű esemény leírása is lehetővé válik, sok

esetben ugyanis egy-egy érdekes jelenség nem köthető csupán egyetlen forráshoz. E térbeli elosztottsághoz hasonlóan elképzelhető például, hogy bizonyos események sorozatára vagyunk kíváncsiak, adott időablakon belül. Természetesen semmi nem zárja ki a kettő kombinációját, azaz hogy térben és időben is elosztott eseményeket fogjunk össze komplex események formájában.



3. ábra – Atomi eseményekből felépített komplex események, kibővítve olyan metainformációkkal, mint az atomi események sorrendezése, időzítése, szűrése, konkurens megjelenése, aggregálása. Egy komplex esemény jellemzően több atomi eseményből épül fel, valamint egy atomi eseményt több komplex esemény is felhasználhat.

2.1. Motivációs példák

Online csalások felderítése – magas szintű követelmények automatizált feldolgozása

Az online (pénzügyi) csalások felderítése a komplexesemény-feldolgozás jellegzetes felhasználási területe. [21] [22] A cél olyan eseményminták megfogalmazása, amelyek *potenciálisan rosszindulatú tevékenységeket* írnak le. Ezt a magas szintű követelményt a biztonsági szabványok és kézikönyvek pontosan specifikálják. Ismert gyakorlati példa: egy bank online rendszerében a felhasználó a pénzügyi tranzakciókat kezelő modulhoz csak úgy férhet hozzá, ha előzőleg a biztonsági alrendszertől erre engedélyt kapott. Ez alapján rosszindulatú mintáknak ítélhetjük meg azt az esetet, amikor egy felhasználó pénzügyi tranzakciót próbál végrehajtani, de az azonosítójával nem történt belépés a rendszerbe egy adott időintervallumon belül.

A magas szintű szabályok formalizálásával, ontológiákba való szervezésével elérhető, hogy ebből a jólformázott struktúrából automatikus eljárások segítségével a megfelelő komplex esemény minták részben előállíthatók legyenek. Ezzel formálisan biztosítható a követelmények teljes lefedettsége.

Algoritmikus kereskedés – historikus adatok intelligens feldolgozása

A tőzsde világának izgalmas kérdése, hogyan lehet-e előre jelezni egy értékpapír, vagy részvény árfolyamát. Aki ugyanis képes erre, nyilvánvalóan előnybe kerül a többi üzletelővel szemben. Mivel tökéletes előrejelzési módszerek nem ismertek, az előny kialakításának egyetlen eszköze a tökéletes előrejelzés minél *pontosabb közelítése* (ugyanazon információk alapján) minél

rövidebb reakcióidőn belül. A komplexesemény-feldolgozás a reakcióidő csökkentésével támogatja a területet. [11] [23] Ahhoz, hogy egy döntés ne csak gyors, hanem (valamilyen értékrendszer szerint) *jó* is legyen, a döntést segítő komplex események paraméterei a historikus adatok feldolgozása nyomán előállt, visszacsatolt értékekkel kerülnek konfigurálásra. A konfigurálás automatizálásával elérhető, hogy az eseményminták gyakorlatilag minden időpillanatban a legpontosabban támogassák a döntéseket.

2.2. Esettanulmány

Biztonságtechnikai monitorozó rendszer

A példában egy banki információs rendszert veszek alapul, amelynek üzemeltetője a biztonsági monitorozó rendszer továbbfejlesztését tűzte ki célul. Az infrastruktúra több alrendszerből és komponensből áll, mindegyiket egy-egy automatizált ágens figyeli, amik a komponens rendszerdiagnosztikai eseményeit előre meghatározott formátumot követő adatcsomagokban egy szabványos, aszinkron üzenettovábbító csatornára küldik.

A monitorozó rendszer célja, hogy a biztonságtechnikai szempontból releváns eseményminták felismerhetők legyenek és automatizálni tudjuk azokat a tevékenységeket, amelyeket végre kell hajtani egy-egy ilyen eseményminta felbukkanásának esetén.
A biztonságtechnikai szempontból releváns eseményminták alapvetően kétfélék lehetnek. Származhatnak egyrészt az infrastruktúraelemek és komponensek szolgáltatásminőségi jellemzőiből (például: teljesítmény, áteresztőképesség, terheltség), vagy olyan felhasználói aktivitásból, amelyek potenciálisan rosszindulatú tevékenységre adnak gyanút.

Megvalósítandók a következő kiemelt esetek:

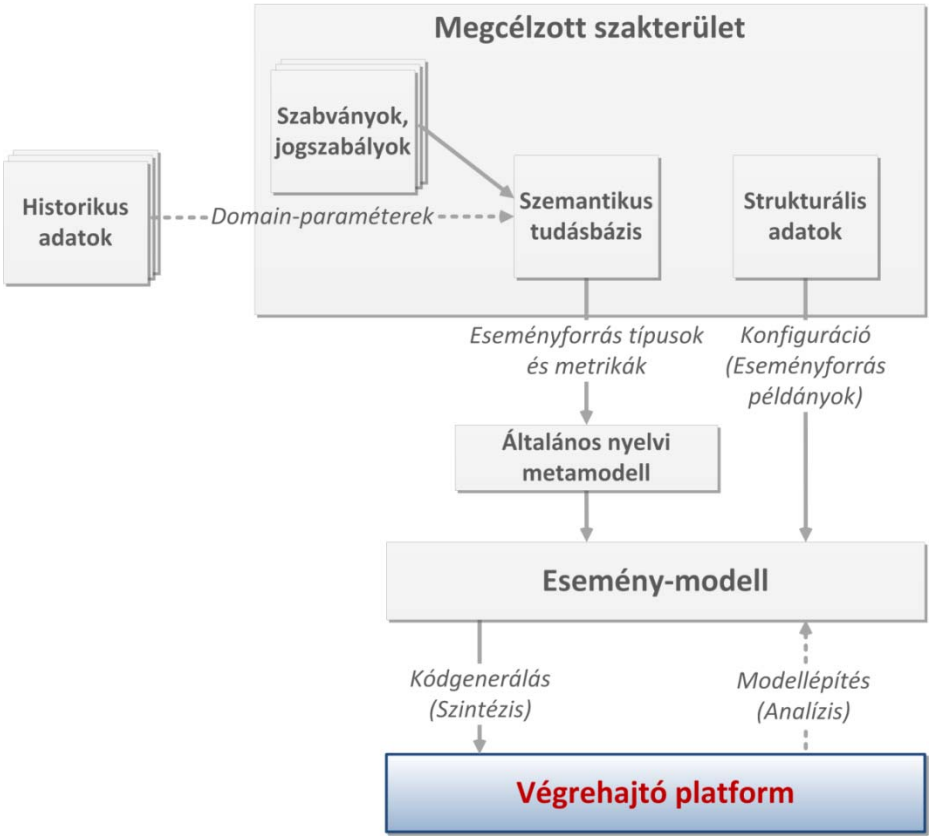
1. Több, egymással topológiai kapcsolatban álló szerver (pl. egy alkalmazás csoportot kiszolgáló web-és adatbázis szerver) terhelése tartósan a kritikus határ közelében van.
2. Egymás tartalékául szolgáló szerverek terhelése egyszerre nő meg.

Az 5. fejezetben ezen a példán keresztül mutatom be a komplex események modellezhetőségét a kifejlesztett módszerek és eszközök segítségével.

2.3. Eseményfeldolgozó platformok

A komplexesemény-feldolgozás ipari elterjedésének nyomán a támogató eszközök is gyors fejlődésnek indultak. Integrációjuk nem okoz különösebb nehézséget, ugyanis egyrészt egyértelmű helyük van a munkafolyamatokban, másrészt viszonylag korszerű technológiákra épülnek (Java, .NET), így az integráció technikai feladatai is leginkább a nem-funkcionális követelményeknek való megfelelésre összpontosul, kiemelten a teljesítményre.

A dolgozat során bemutatott példák és kódrészletek az Esper nyílt forráskódú eszközhöz készültek el, ezen a platformon alkalmazhatók. A választás oka az volt, hogy az nyílt forráskódú szoftverek között az Esper *de facto* standard megoldásnak tekinthető. [24]
(A dolgozat elkészítése során referenciaként felhasználtam még az iCEP projekt eseményfeldolgozó platformjának esettanulmányait és dokumentációit. [25].)



4. ábra – Az eseményfeldolgozó platform helye a kialakított megoldásban.

2.4. A modellalapú fejlesztés előnyei és hátrányai

Az eseményfeldolgozó eszközök platformközeli nyelvei szükségszerűen összetett formalizmussal dolgoznak, amelyek ráadásul nem követnek közös nyelvi szabványokat, így nehéz az áttérés egyik platformról a másikra. Szintén probléma lehet, hogy a nyelvek struktúrájából adódóan a kódbázis karbantartása problémás a növekvő komplexitás mellett. Ha új mintát kell implementálni, nem tudhatjuk biztosan, hogy az milyen kapcsolatban áll a korábban már implementáltakkal, mert egyik ismert platformhoz sem létezik olyan formális következtető logika, ami ellentmondásokat, korrelációkat, vagy fedéseket vizsgálna a minták között.
Ezzel szemben a modellalapú megközelítés magasabb logikai szinten képes megragadni a definiált eseményminták jólformáltsági kritériumait és azokat modellellenőrző eljárásokkal biztosítani.

A modellalapú fejlesztés főbb előnyei:

- elfedi a különböző nyelvek szintaktikai elemeit és egységes platformot biztosít az események, eseményminták leírására;
- a modelleken jólformáltsági ellenőrzések végezhetők, ezért szintaktikailag, de akár szemantikailag is helyes forráskódot tudunk előállítani gyakorlatilag bármelyik célplatformra;
- ugyanezen jólformáltsági ellenőrzések segítségével kimutathatók az esetleges ellentmondások, átfedések a definiált minták között.

Természetesen a módszernek hátrányai is vannak. A legfontosabb probléma egyértelműen az, hogy az eseményfeldolgozó platformok funkcionalitását általánosan kellene modellezni, hogy a lehető legtöbb platformmal tudjuk biztosítani az együttműködést. Egy ilyen általános modellel azonban a platformok specialitásait nem tudjuk leírni.

3. Komplex események modellezése

3.1. A modellezésről általában

Ahhoz, hogy modellalapú, vagy modellvezérelt megközelítésről beszélhessünk, tisztázni kell a szükséges alapfogalmakat és az egyes munkafolyamatokat.

A modell modellje: a metamodell

A formalizmust, amit egy modell követ, szigorú szabályok határozzák meg. A legegyszerűbb esetben ezek a szabályok közvetlenül egy szabványból származnak, például amikor UML osztálydiagramot készítünk egy alkalmazáshoz. Ezen szabványok szabályai önmaguk is egy modellt definiálnak, amely azt írja le, hogy a szabvánnyal összhangban lévő modellek hogy nézhetnek ki. Ezt az egy szinttel absztraktabb modellt hívjuk *metamodell*nek. Például az UML osztálydiagram formalizmusának metamodellje definiálja többek között az *osztály* és *attribútum* elemeket és a köztük lévő *egy-több kapcsolat*ot. Minden osztálydiagram típusú modell eszerint épül fel, ezt a metamodellt példányosítja.

Szakterület-specifikus (domén-specifikus) modellezőnyelvek[2]

Az általános szabványok fogalmi szintje gyakran túl általánosnak bizonyul és leíróereje nem megfelelő ahhoz, hogy releváns információk vesztése nélkül modellezzünk. Ekkor a tulajdonképpeni modellt nem közvetlenül a szabvány által definiált metamodellből származtatjuk, hanem előbb testre szabjuk a szabványt, kibővítjük azt a (szintén a szabvány által definiált) megfelelő módszerekkel. Az ilyen kibővített modellező nyelveket nevezzük *szakterület-specifikusnak*, hiszen egy adott célterületre fókuszálnak, megkönnyítve a terület modellezését.

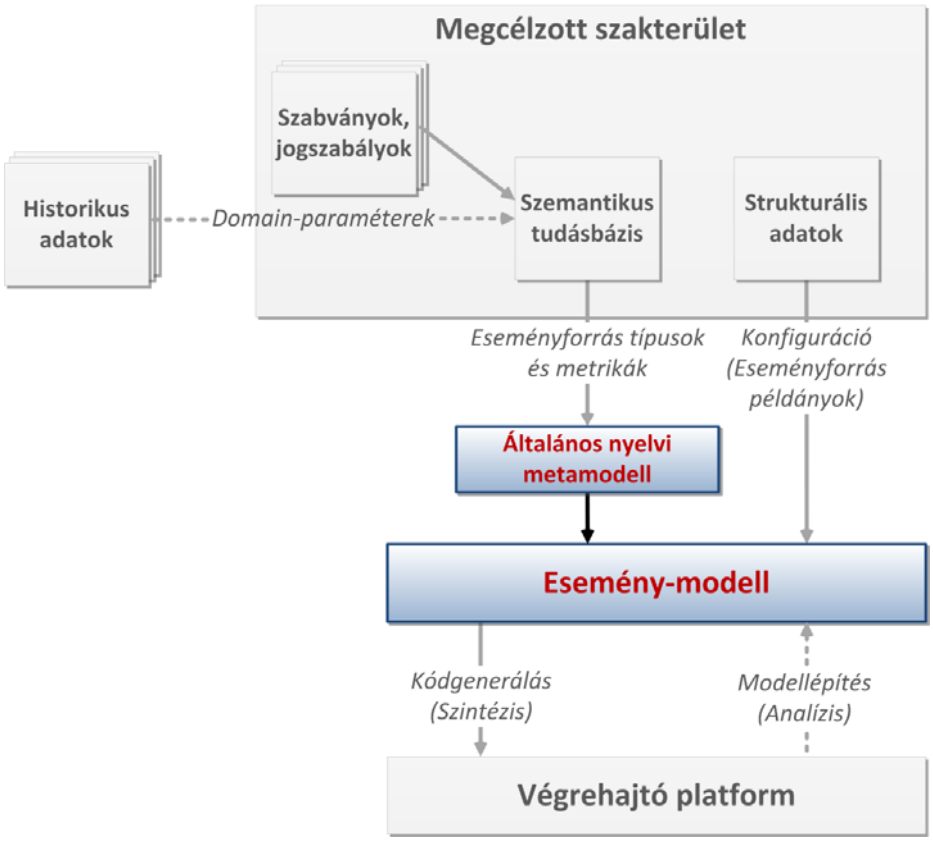
Szakterület-specifikus bővítésekkel egy modellezőnyelv tehát olyan speciális tulajdonságokkal ruházható fel, amelyek segítségével csökkenthetjük a modell számára kezelhetetlen információk számát, azaz *növelhetjük a nyelv leíróerejét*. A komplex események modellezéséhez – tekintve, hogy egy speciális és jól behatárolt szakterületről van szó – a dolgozat során egy általam kifejlesztett domén-specifikus nyelvet fogok bemutatni.

UML és EMF

A két legelterjedtebben alkalmazott modellezési szabvány, az UML (*Unified Modeling Language*) és az EMF (*Eclipse Modeling Framework*) is definiál domén-specifikus bővítési lehetőségeket. Míg az UML nagyon jól használható általános rendszertervezési problémákhoz, az EMF a Java-világ de facto szabványává vált, és a szakterület-specifikus modellezés elsődleges eszközévé, részben ennek is köszönhetően az EMF-fel való munka eszköz-támogatása nagyon fejlett és előrehaladott. Az EMF további előnye, hogy lehetővé teszi a szöveg alapú modellezést is, míg az UML csak grafikus modelleket definiál. A dolgozat során bemutatott modellezőnyelv az EMF-et veszi alapul, mert – mint később kitérek rá – a szöveg alapú megközelítés jóval hatékonyabbnak bizonyult a területen, mint a grafikus.

[2] Az angol *domain-specific* szóból származtatva a magyar szakirodalom is gyakran használja a *domén-specifikus* szakszót. *Domain* (eng): tartomány, tárgykör, szakterület.

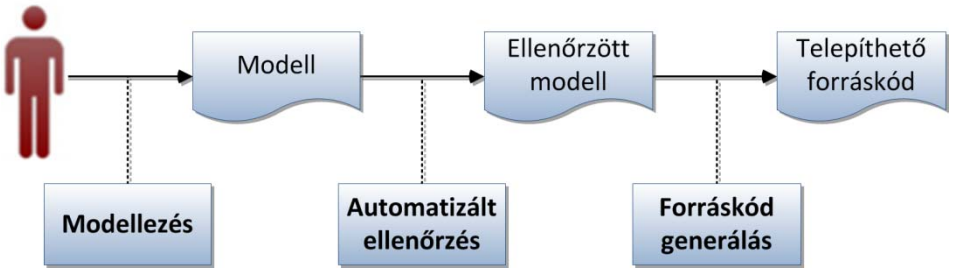
3.2. Komplexesemény-metamodell és modell



5. ábra – A metamodell helye a kialakított megoldásban.

A komplex események modellezésére alkalmas megközelítés a következőket követeli meg:

- Olyan *általános és szakterület-specifikus metamodellek* kialakítása, amelyek együttes használatával lehetőség nyílik az események platform-független megfogalmazására. A metamodell általános részét csak egyszer kell elkészíteni, hiszen az felhasználható bármilyen szakterületen, a szakterület-specifikus részt azonban minden szakterületre célzottan kell elkészíteni.
- A modell *jólformáltsági kényszereit ellenőrző eljárások* kialakítása, amelyek biztosítják a teljes szintaktikai integritás, és nagy részben szemantikai szempontból is nyújtanak bizonyos szintű garanciát.
- A modellelemek *egyértelmű megfeleltetése* a potenciális célplatform szintaktikai elemeinek. Ez alapján egy kódgeneráló logikát lehet implementálni, ami a modellekből a célplatform szintaktikájának megfelelő állományokat (forráskódokat és konfigurációkat) állítja elő.
- Valamint gondoskodni kell a modelleket konfiguráló *szakterület-specifikus strukturális adatok* kezeléséről és a tervezési időben a modellel való integrálásukról.



6. ábra – A modellezés folyamata. Miközben a tervező a modellt építi, automatizált eljárások ellenőrzik a modell jólformáltsági kényszereit, az ellenőrzött modellből pedig telepíthető forráskód generálható. (Az utóbbi lépés szintén automatizálható.)

Az elkészített modellekből egy konkrét platformra telepíthető állományokat állítunk elő (automatizálva), ezért természetesen célszerű volna, ha a metamodell kialakításánál már figyelembe vennénk olyan szabványokat, amelyeket egyaránt támogat minden eseményfeldolgozó platform. Ilyen szabványok azonban jelenleg nem léteznek, minden gyártó saját szintakszissal és szemantikával dolgozik, így a legjobb megközelítés a gyakorlati problémák alapján kialakítani a metamodellt, ezután leképezni a modellelemeket valamely konkrét platformra.

3.3. CEDL – Complex Event Description Language

Dolgozatom egyik lényegi pontja az általam kifejlesztett *Complex Event Description Language* (a továbbiakban: CEDL) nevű modellezőnyelv, amely az Eclipse Modeling Framework egy szakterület-specifikus kiterjesztése. A nyelv segítségével, alapvetően szöveges módon, komplex eseményeket modellezhetünk. Kialakítása során a 3.2. pontban felsorolt irányelveket követtem és gyakorlati problémák jellegzetes modellezési igényeire támaszkodva definiáltam a nyelvi elemeket.

A nyelv mögött egy *statikus nyelvi modell* áll, ami a nyelvi elemeket definiálja. Minden modell, amit a nyelvvel hozunk létre, ennek a modellnek egy példánya.

Példák CEDL szintakszissal definiált komplex eseményekre

A fejezet további részeiben bemutatott rövid példák az informatikai rendszerek monitorozásának szakterületéről származnak.[3]

A következő minta azt az atomi eseményt írja le, amikor *Server1* nevű forrás *CPULoad* paramétere legalább *90%* lesz.

```
Event CPULoadCritical {
      source Server1
      PercentageMeasurement CPULoad Minimum 90
}
```

A következő példa hasonló az előzőhöz: azt az esetet írja le, amikor a *Server1* forrásnak *kevesebb, mint 2* tartaléka van.

[3] A nyelvi specifikáció rövid kivonatát lásd a Függelékben.

```
Event BackupProblem {
	source Server1
	ScalarMeasurement AvailableBackups LessThan 2
}
```

A két atomi eseményből már előállíthatunk komplex eseményeket. Vegyük azt a komplex eseményt, amikor a két fenti atomi esemény egyszerre következik be és legalább 30 másodpercig fennáll!

```
ComplexEvent CriticalServer{
	CONCURRENT_T(CPULoadCritical BackupProblem; T: Minimum 30)
	action {
		Action of Type sendWarning
	}
}
```

Amennyiben a két atomi esemény együtt következik be és legalább 30 másodpercig fennáll, egy *sendWarning* típusú akciót hajtunk végre, azaz figyelmeztetést küldünk.

Látható, hogy a leírónyelv nagyon tömören, csupán pár sorban képes megragadni nagyon specifikus jellemzőket. A fenti példákból azonban nem derül ki, hogy azok paraméterei (a *Server1* forrás, annak *CPULoad* és *AvailableBackups* paraméterei és a *sendWarning* akciótípus) hol és milyen formában kerülnek definiálásra.
Az egyértelmű, hogy mindhárom elem szakterület-specifikus, de nem teljesen magától értetődő, hogy a szakterületnek melyik részéről származnak (a szemantikus tudásbázisból, vagy a strukturális adatok közül?), és hogy kerülnek a nyelvi elemek szintjére.

Általános felépítés és nyelvi elemek
Egy esemény-modell felépítéséhez alapvetően három dologra van szükség: az általános nyelvi modell által definiált elemekre, szakterület-specifikus elemekre és konfigurációra.

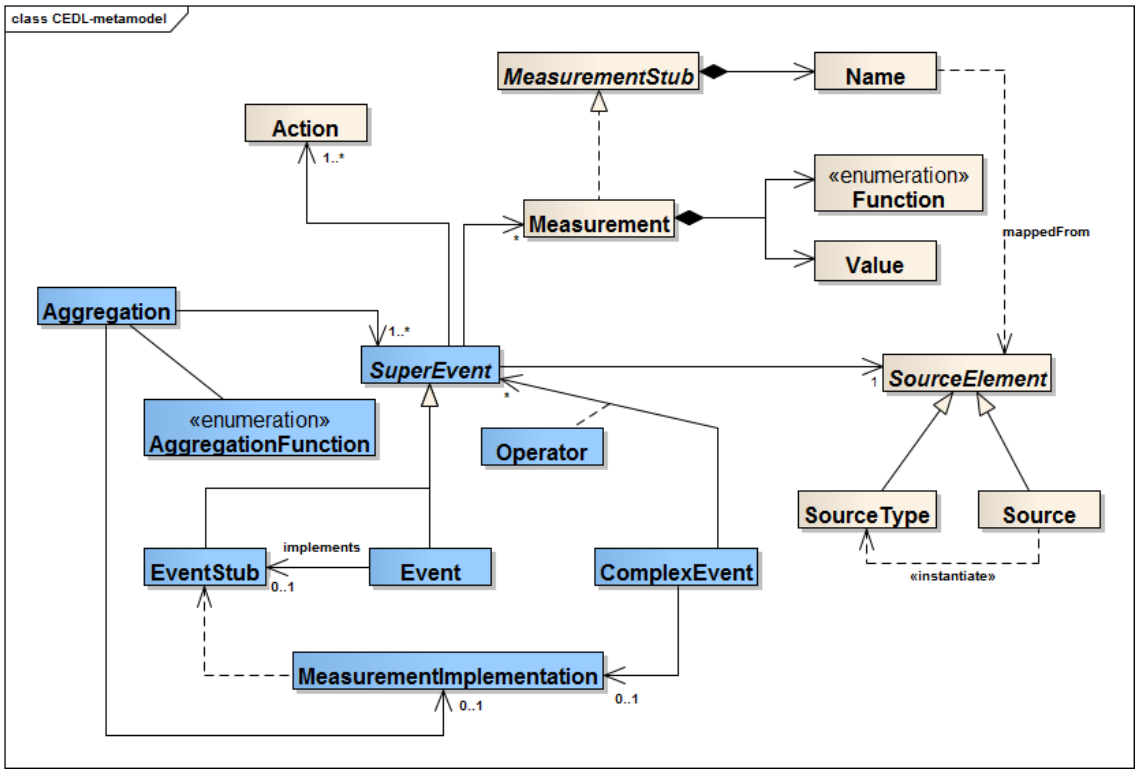
Az *általános nyelvi modell* a modellezőnyelv funkcionalitását határozza meg, tehát azokat az alapvető nyelvi elemeket, amelyeket minden modellezési feladatnál felhasználhatunk, például: *Esemény*, *Eseményforrás*, rendezési operátorok.
Az általános nyelvi modell még nem tartalmaz szakterület-specifikus fogalmakat, pedig épp ez kell ahhoz, hogy egy nyelv leíróereje kiemelkedő legyen. Ezért ezt a modellt a szakterületről származó információkkal kell kiegészíteni. Az információk logikailag két szinten találhatók meg:

- alapvető eseményforrás *típusok*, köztük lévő kapcsolatok és a hozzájuk kapcsolódó releváns metrikák („milyen típuson mit tudunk mérni?”), illetve
- konkrét eseményforrás *példányok* az eseményforrás típusok alapján.

Az első pont szükséges kiegészítése az általános nyelvi modellnek, aminek segítségével így már előállítható egy esemény-modell váz. Ahhoz, hogy az esemény-modell teljes értékű legyen, a második pont információival kell ellátni, azaz konfigurálni kell a modellt.
Előbbi elemeket szolgáltatja a szemantikus tudásbázis, a konfigurációt pedig a strukturális adatokból nyerhetjük ki.

Példa. Egy informatikai infrastruktúra esetében Eseményforrás típus lehet a *Webszerver*, az *Adatbázisszerver*, vagy a *Fájlrendszer* entitás, egy lehetséges konfiguráció pedig az „*X infrastruktúra elem egy Webszerver*” típusú információ.

3.4. A CEDL általános nyelvi modellje



7. ábra – A CEDL nyelvi modelljének egyszerűsített nézete.[4] Kékkel jelölve a szakterülettől független modellelemek. A többi elem a szakterület-specifikus információk becsatolására biztosít interfészt.

A nyelvi modell központi eleme az *Event* típus, ami egy atomi eseményt ír le. Alapvetően minden releváns eseményt előbb detektálni szeretnénk, majd végrehajtani valamilyen akciót (például egy automatizált eljárást). Ez utóbbi célt szolgálja az *Action* típus. (A 3.3. fejezetben látható példa az alkalmazásra.)

Az esemény detektálásához szükséges esemény mintát a *Measurement* elemmel definiálhatjuk. Egy ilyen típus rendelkezik névvel (*Name*), egy matematikai, vagy logikai összehasonlító függvénnyel (*Function*), valamint egy értékkel (*Value*). A név az esemény forrásához kapcsolódik: annak egy mérhető paraméterére hivatkozik – ilyen lehet például egy webszerver forrás esetén a processzor terheltsége (*CPULoad*, a 3.3. fejezet példájában). Ilyen módon tehát megadható, hogy egy adott eseményhez milyen forráson milyen mérhető paramétereket milyen értékkel figyelünk – ezáltal leírható váltak azok a tulajdonképpeni *minták*, amik alapján egy-egy esemény bekövetkezését érzékelni tudjuk.

A *ComplexEvent* típus több *Event* (vagy *ComplexEvent*) típust kapcsol össze és egy *Operator* entitást ad a struktúrához. A 3.3. fejezetben láthattunk erre példát, amikor a *CriticalServer* komplex esemény a *CPULoadCritical* és a *BackupProblem* eseményekből épült fel, a *CONCURRENT_T* pedig az operátor szerepkört töltötte be.

Lehetőség van az *Event* típusnál egy általánosabb struktúrát definiálni, ez az *EventStub*, ami a nyelv egy esszenciális eleme, kifejezetten azzal a céllal létrehozva, hogy részleges, hiányos információk segítségével is tudjunk eseménymintákat definiálni, vagy hogy több, logikailag

[4] A teljes modellt lásd a Függelékben.

azonos eseménytípust egy közös őssel összefogjunk. A megközelítés az objektum-orientált paradigma interfész és ősosztály elemeit ötvözi. Erről bővebben a 3.6. fejezetben.
Az *EventStub* lehetővé teszi, hogy egy eseménymintához csak jelezzük, hogy egy akciót kell hozzá rendelni, de ezt az őt implementáló komplex eseményekre bízza – ezt a célt szolgálja a *MeasurementImplementation* osztály.

A *Measurement* típushoz kapcsolódó, a minták leírásának kifejezőerejét növelő entitás az *Aggregation*, ami a komplex események által összefogott atomi események paramétereinek bizonyos szempontok alapján történő aggregálására használhatók. Minden aggregációhoz egy aggregációs függvényt kell megadni, amelyek egy enumeráció típusban vannak definiálva. Egy ilyen aggregáció lehet például, amikor azt szeretnénk leírni, hogy a *CPULoadCritical* esemény *CPULoad* paraméterének utóbbi 60 másodpercben vett átlaga elérte a 85%-ot:

```
AVG(event CPULoadCritical.'CPULoad') Minimum 85 TIMEWIN(60)
```

Természetesen minden eseményhez egy forrás is tartozik (*SourceElement*), ami lehet egy konkrét forrás példány, vagy csak egy típus.

Komplexesemény-operátorok

Az események egymással való kapcsolatának leírásához, a sorrendiség és időzítés kifejezéséhez a nyelv három alap operátort definiál:

- `EXISTS(Event e1, Event e2… Event en)`
  A paraméterül kapott események *létezését* írja elő.
- `FOLLOWS(Event e1 as Alias1 (N1), Event e2 as Alias2 (N2)… Event en as Aliasn (Nn))`
  A paraméterül kapott események *sorrendiségét* írja elő. Az egyes eseményeknek alternatív neveket (*alias*) is adhatunk, valamint számosságot (*N*). Ilyen módon két e1 esemény után következő két e2 esemény a következő módon írható le: `FOLLOWS(e1 (2), e2(2))`.
- `CONCURRENT(Event e1 as Alias1 (N1), Event e2 as Alias2 (N2)… Event en as Aliasn (Nn))`
  A paraméterül kapott események *egyidejű bekövetkezését* írja elő. Az egyes eseményeknek itt is adhatunk alternatív neveket, valamint számosságot, hasonlóan a FOLLOWS operátorhoz.

A fenti operátorok leíró erejét növeli, az eseményfeldolgozó platformok oldaláról pedig könnyen támogatható az *időablakok* definiálása, amit mindhárom operátor esetén megtehetünk:

- `EXISTS(Event e1, Event e2… Event en).timewin=10`
- `FOLLOWS_T(Event e1 as Alias1 (N1), Event e2 as Alias2 (N2)… Event en as Aliasn (Nn); T: 10)`
- `CONCURRENT_T(Event e1 as Alias1 (N1), Event e2 as Alias2 (N2)… Event en as Aliasn (Nn); T:10)`

Az eseményminták jelentése ugyanaz, mint az előbbi esetben, azonban mindig egy 10 másodperces csúszó időablakon belül vizsgálva.

Az *egyidejűség* értelmezése egy valós rendszerben nem egyszerű, hiszen a valóban egy időben bekövetkező események száma elhanyagolható, akkor viszont ennek az operátornak a

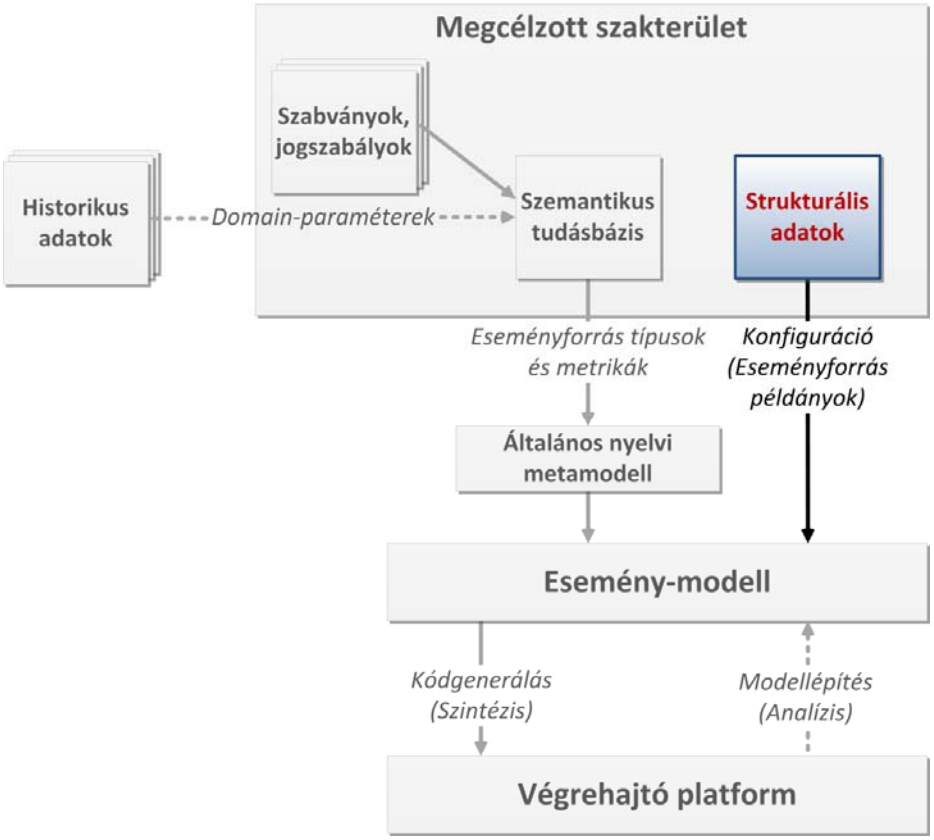
használata sem lenne célszerű. A CEDL által definiált CONCURRENT és CONCURRENT_T operátorok közül utóbbi egy expliciten definiált időablakon belüli előfordulást ír elő a paraméterként kapott események számára. Végeredményben a CONCURRENT operátor is hasonló elven működik, azonban az időablakot implicit módon definiálja egy, a tervező által meghatározott *epsilon* nevű paraméter alapján, ami a célrendszerre jellemző minőségi jellemzők alapján számolható ki, mint például a teljesítőképesség, a megbízhatóság, vagy a mérési pontosság. Az *epsilon* paraméter, ahogy a neve is utal rá, egy kis szám, ami a gyakorlatban azt mondja meg, mekkora az a legkisebb időablak, amin belül észlelt eseményeket egyidejűnek tekinthetünk. A tesztelések során én ezt az értéket 0.5-nek vettem, azaz a 0.5 másodpercen belül érkező eseményeket tekintettem egyidejűnek.

Bizonyos minták megfogalmazása esetén célszerűbb a problémát ellenpéldák, úgynevezett *negatív minták* segítségével definiálni. Ezt támogatja a negáció nyelvi elem, amit egy-egy eseménytípusra írhatunk elő a komplexesemény operátorokon belül. A következő kifejezés például egy olyan mintát definiál, ahol az e1 esemény után nem e2 következik:

```
FOLLOWS(e1, neg e2).
```

Az operátor jelentése az SQL-szabvány *NOT* kulcsszavának feleltethető meg.

3.5. Strukturális információk alkalmazása a modellben



8. ábra – A strukturális adatok a megcélzott szakterületről származnak, és az eseménymodellt konfigurálják (például a szemantikus tudásbázisból származó entitástípusok példányaival).

Mértékek (*Measurement*) neveinek leképezése

Az előző fejezetben szó volt róla, hogy a *Measurement* (pontosabban a *MeasurementStub*) típus által tartalmazott név (*Name*) attribútum az esemény forrásának egy mérhető

paraméterére hivatkozik, viszont nem tisztázódott, hogyan jön létre a leképezés a nyelvi elem és a strukturális adatok között. Azaz, például, a következő esetben honnan tudjuk a modellezés során, hogy a *Server1* rendelkezik egy *CPULoad* nevű mérhető paraméterrel?

```
Event CPULoadCritical {
      source Server1
      PercentageMeasurement CPULoad Minimum 90
}
```

Már egy kisméretű monitorozott IT-infrastruktúránál is problémás lehet a manuális leképezés, tekintettel a mért paraméterek potenciálisan nagy számára. Mindezt emberi erőforrással leképezni és karban tartani kifejezetten nehéz feladat.

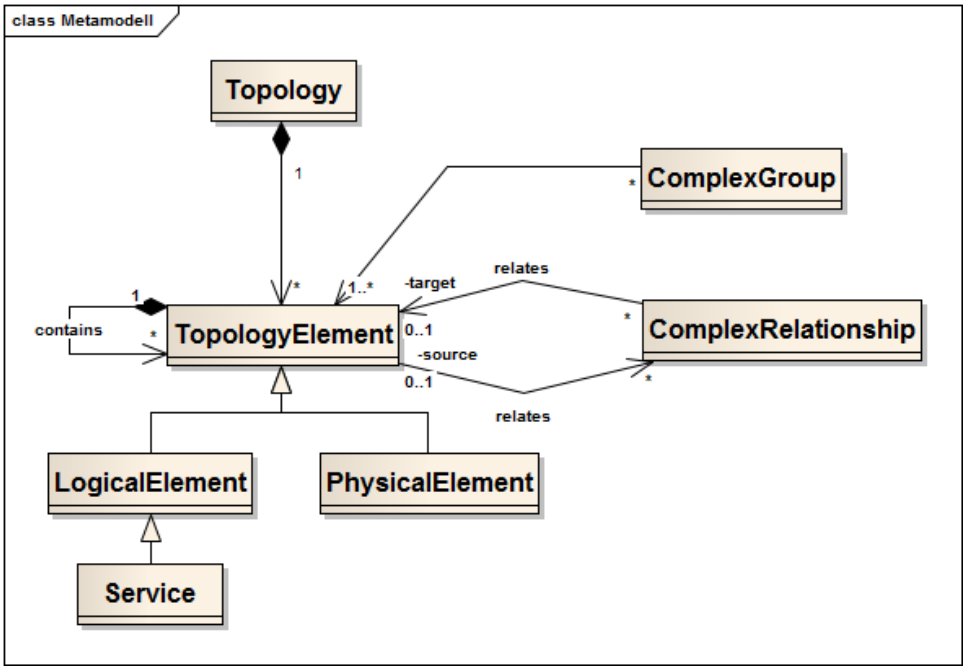
A megoldást a monitorozó eszközök, vagy ágensek és a nyelv közötti automatizált leképezés jelenti. A dolgozat nem foglalkozik mélyebben a problémával, mert arra korábban már született publikált megoldás. [26]

Infrastruktúra (topológia) elemek leképezése

Hasonlóan problémás lenne az infrastruktúra elemeket (eseményforrás példányok) manuális módszerrel a modellbe felvenni – bár erre a nyelv, természetesen, lehetőséget biztosít. Mégis célszerűbb automatizálni ezt a lépést.

Általában élhetünk azzal a feltételezéssel, hogy a célinfrastruktúráról valamilyen konfigurációs adatbázisban (CMDB – *Configuration Management Database*) találunk információkat, vagy az információk könnyen begyűjthetők (például automatizált ágensek segítségével). Ha rendelkezésre állnak az erőforrás típusok és a köztük lehetséges kapcsolatok típusai, akkor a kereskedelmi CMDB megoldások API-ján keresztül viszonylag egyszerűen le tudjuk képezni a CEDL szintjére az infrastruktúra elemeket, azaz a forrás példányokat.

Az alábbi ábrán a CEDL nyelvi modellje által a *SourceElement* entitáson keresztül hivatkozott *topológia metamodell* látható. A 3.4. fejezetben tárgyalt források tehát egy IT infrastruktúra esetében tulajdonképpen topológia elemeknek feleltethetők meg, azaz egy *SourceElement* a CEDL nyelvi modellben egy *TopologyElement* a topológia metamodellben.
A metamodell kialakítása során a CIM metasémát követtem, így biztosítva a lehető legnagyobb konformitást a rendelkezésre álló szabványokkal és az azokat követő infrastruktúrákkal. [27] [28]



9. ábra – Topológia metamodell.

A CMDB-ből történő leképezések a topológia metamodell alapján történnek. Minden eseményforrás példányból egy *Source* típusú entitás áll elő, a köztük lévő kapcsolatok pedig *ComplexGroup*, vagy *ComplexRelationship* típusúak lesznek. Előbbi írja le azt az általános csoportosító kapcsolatot, amikor minden forrás egyforma rangú a csoporton belül, utóbbi viszont egy irányított forrás-cél kapcsolatot definiál. (Lásd: 3.4. fejezet, 7. ábra.)
Például *ComplexGroup* típusú az „*X alkalmazást kiszolgáló összes erőforrás*”, *ComplexRelationship* típusú az „*Y webszerver és annak Z tartaléka közti kapcsolat*”, a „*tartaléka*” kapcsolat egyértelműen irányított jellege miatt.

A [29] irodalom egy mintaimplementációt mutat be a HP UCMDB termékéhez kapcsolódóan.

3.6. A CEDL által definiált tervezési minták

A tervezési minták célszerű alkalmazása minden informatikai fejlesztési területen hasznosnak bizonyul, hiszen a gyakori problémákra adnak *best-practice* jellegű, kipróbált megoldást. Természetesen a komplex események feldolgozásához, mint folyamathoz is léteznek ilyen tervezési minták, ezek azonban jellemzően magára a feldolgozás folyamatára adnak iránymutatásokat. [30] [31]

Az előző pontban bemutatott komplexesemény-leíró nyelv modellje gyakorlati problémák feldolgozásával került kialakításra, azzal a kiemelt céllal, hogy a tervezés ne legyen túl bonyolult (minél kevesebb elem felhasználásával tudjunk pontos modelleket alkotni) és ne legyen nagyon időigényes. A kialakítás közben a modellezési problémák bizonyos pontokon nagy hasonlóságot mutattak, ezeket a komplexesemények-modellek tervezési mintáiként formalizáltam. A szokásos módon, a mintákat a következőkkel definiálom: név (angol név), probléma, megoldás, következmények.

Eseménycsonk (Event Stub)

Probléma:

1. Több esemény logikailag egy másik eseményt terjeszt ki új paraméterek hozzáadásával, vagy az eredeti paramétereinek felüldefiniálásával. Úgy szeretnénk az ősben attribútumokat definiálni, hogy annak ne kelljen értéket adni, ha egyébként is minden öröklő eseménymintában felül kell definiálni azt.
2. Az események formátuma csak részben ismert, vagy az öröklők formátuma csak részben azonos.
3. Az eseményforrások típusára szeretnénk előírni eseménymintát, amit akár az összes adott típusú forrásra felhasználhatunk.

Megoldás:

Az objektum-orientált terminológiát használva, a problémát részben az *interfész*, részben pedig az *ősosztály* használata oldja meg. A 2. pont tisztán az ősosztály, a 3. tisztán az interfész esete. Az 1. pontban leírtak viszont a kettőt egyesítik: definiálni akarunk valamiféle ős-eseménymintát, de szeretnénk fenntartani a lehetőséget arra, hogy bizonyos attribútumoknak ne adjunk értéket, csak definiáljuk azok jelenlétét (mint az interfész metódusdeklarációi).

A megoldás az Eseménycsonk használata. Az Eseménycsonk formális definíciója szerint egy Esemény típusból kapható meg, annak csonkolásával, ahol a csonkolás funkció a következőket jelentheti:

- struktúra-csonkolás: Az Eseményből eltávolítunk bizonyos strukturális elemeket: attribútumot, akció-leírást, vagy metrikát.
- értékcsonkolás: Az Esemény egy strukturális elemének értékét távolítjuk el, így az Eseménycsonkban csak a strukturális elem jelenlétére utaló definíció marad, az értékadást a csonkot megvalósító Esemény köteles elvégezni.

Az Eseménycsonk definíció szerint csak Eseményforrás típusra hivatkozhat, tehát konkrét forrásra nem. A csonkot megvalósító Eseményben már konkrét forráspéldány szerepel, aminek típusa a csonkban definiálttal egyezik.

Következmények:

A minta használatával elkerülhető a felesleges modell-szintű redundancia, átláthatóbbá válik a struktúra. Lehetőség nyílik rá, hogy csak egy forrás típushoz kapcsoljunk eseménymintákat, ne konkrét forráshoz.

Példa:

„Két webszerverünk van (ws1 és ws2). Mindkettőn szeretnénk jelezni, ha a processzor terheltsége kritikus szint fölé emelkedik. A ws1 szerver esetén ez 90%, a ws2 esetén 95%. Emellett szeretnénk definiálni egy eseménymintát, ami azt az esetet írja le, amikor bármelyik webszerver processzor-terheltsége 50% fölé emelkedik."

A csonk a webszerver típusú forrásra vonatkozik, egy százalékos mértéket ad meg a processzorterheltségre és egy figyelmeztetés akciót definiál alapértelmezetten.

```
EventStub CriticalCPULoadOnWebServer {
    sourceType WebServer
    PercentageMeasurement CPULoad
    actions{
           Action of Type sendWarning
    }
}
```

A két konkrét esemény beállítja a forráspéldányt és a processzorterheltséget, az akciót pedig örökli, de nem módosítja:

```
Event CriticalCPULoadOnWS1 implements CriticalCPULoadOnWebServer {
    source ws1
    @ImplementMeasurement{
           PercentageMeasurement CPULoad Minimum 90
    }
}

Event CriticalCPULoadOnWS2 implements CriticalCPULoadOnWebServer {
    source ws2
    @ImplementMeasurement{
           PercentageMeasurement CPULoad Minimum 95
    }
}
```

A harmadik esemény az összes webszervert figyeli és jelez, ha bármelyik terheltsége 50% felett van:

```
Event SuspiciousCPULoadOnAnyWS implements CriticalCPULoadOnWebServer {
    source OFTYPE WebServer
    @ImplementMeasurement{
           PercentageMeasurement CPULoad Minimum 0.50
    }
}
```

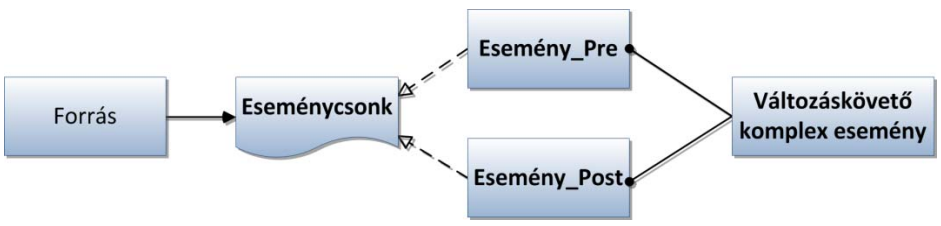
Változáskövetés (Change tracking)

Probléma:

Olyan eseményeket szeretnénk figyelni, amely egy adott forrás valamely paraméterében történt változást ír le.

Megoldás:

A forráshoz egy eseménycsonkot hozunk létre, amely a változó paramétereket definiálja – a csonkot két atomi esemény implementálja: az elő-, illetve az utófeltétel-esemény. A két eseményt egy komplex eseményben fogjuk össze, opcionálisan időablak paraméterrel kiegészítve.



10. ábra – Változáskövető komplex esemény. Az elő- és utófeltétel események egy közös csonkot implementálnak, majd egy komplex eseménybe szervezzük őket.

Következmények:

A tervezési minta alkalmazásával egy adott forrásból származó eseményeket foghatunk össze olyan módon, hogy az adott forrás kiválasztott paramétereinek változásait leíró eseményekre fogalmazunk meg mintákat.

Példa:

„A szerver terheltsége rövid időn belül (5 másodperc) nőtt a kritikus értékre (90%).”

Először létrehozzuk az eseménycsonkot a releváns paraméter definiálásával:

```
EventStub CriticalCPULoad {
   sourceType WebServer
   PercentageMeasurement CPULoad
}
```

Ezután az implementáló eseményeket definiáljuk:

```
Event Precondition implements CriticalCPULoad {
   source WebServer1
   @ImplementMeasurement{
         PercentageMeasurement CPULoad LessThan 90
   }
}

Event Postcondition implements CriticalCPULoad {
   source WebServer1
   @ImplementMeasurement{
         PercentageMeasurement CPULoad Minimum 90
   }
}
```

Végül a két atomi eseményt összefogjuk egy komplex eseményben és megadjuk az 5 másodperces időablakot:

```
ComplexEvent CriticalCPULoadChange {
   FOLLOWS(Precondition Postcondition; T: Maximum 5)
}
```

3.7. A modellezőnyelv leíróereje

Egy nyelv leíróerejének meghatározásához nincs abszolút, de facto mértékrendszer. A legjobb közelítések általában összehasonlító módszereket használnak két, vagy több nyelv között, olyan szempontok alapján, mint az azonos funkcionalitást megvalósító kódrészletek utasításainak, vagy kódsorainak száma.

A tesztelés során keletkezett forráskódok alapján összehasonlítottam, hogy hány utasítást és kódsort tartalmaznak a CEDL segítségével megfogalmazott eseményminták, valamint az ebből generált Esper szintaktikájú forráskódok. Ezzel egy jó közelítő képet kapunk arról, hogy mennyire gyorsítja a munkát a CEDL nyelv, valamint általában egy szakterület-specifikus modellezőnyelv.

A tesztek során mért adatokat az alábbi táblázatok foglalják össze.

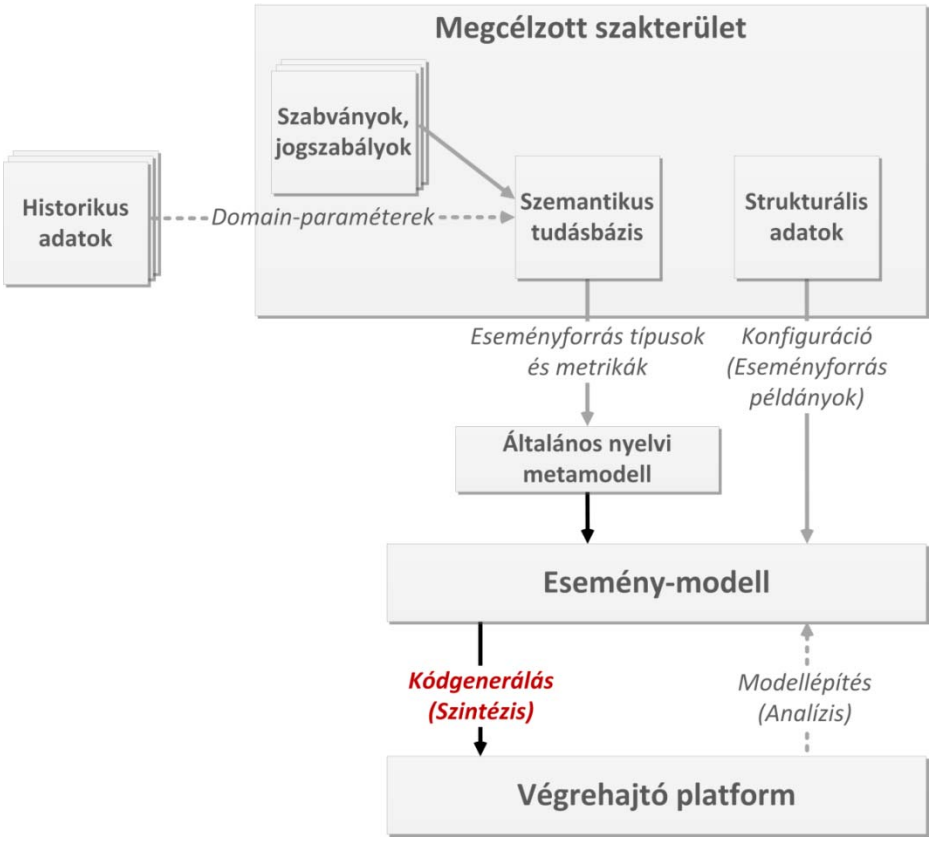
CEDL		Generált kód (az Esper platformra)	
Kódsorok száma	**Utasítások száma**	**Kódsorok száma**	**Utasítások száma**
116	121	341	351

1. táblázat – A tesztek során mért adatok: kódsorok, illetve utasítások száma a CEDL, illetve az ESPER platformon.

Látható, hogy mind a kódsorok számának tekintetében, mind az utasítások tekintetében a CEDL közel *háromszor* hatékonyabbnak[5] bizonyult, azaz harmadannyi „kódolással" ugyanaz a funkcionalitás implementálható, mint az Esper platformon, az üzleti követelményekhez közelebbi absztrakciós szinten.

A táblázatban hivatkozott generált kód nem csupán a mintafelismerést valósítja meg, hanem integráltan kezeli a mintafelismerést, a szűrést és a feldolgozást megvalósító osztályokat, valamint konfigurációs és interfészleíró segédállományokat

3.8. A modellezőeszköz megvalósítása



11. ábra – A kódgenerálás helye a kialakított megoldásban.

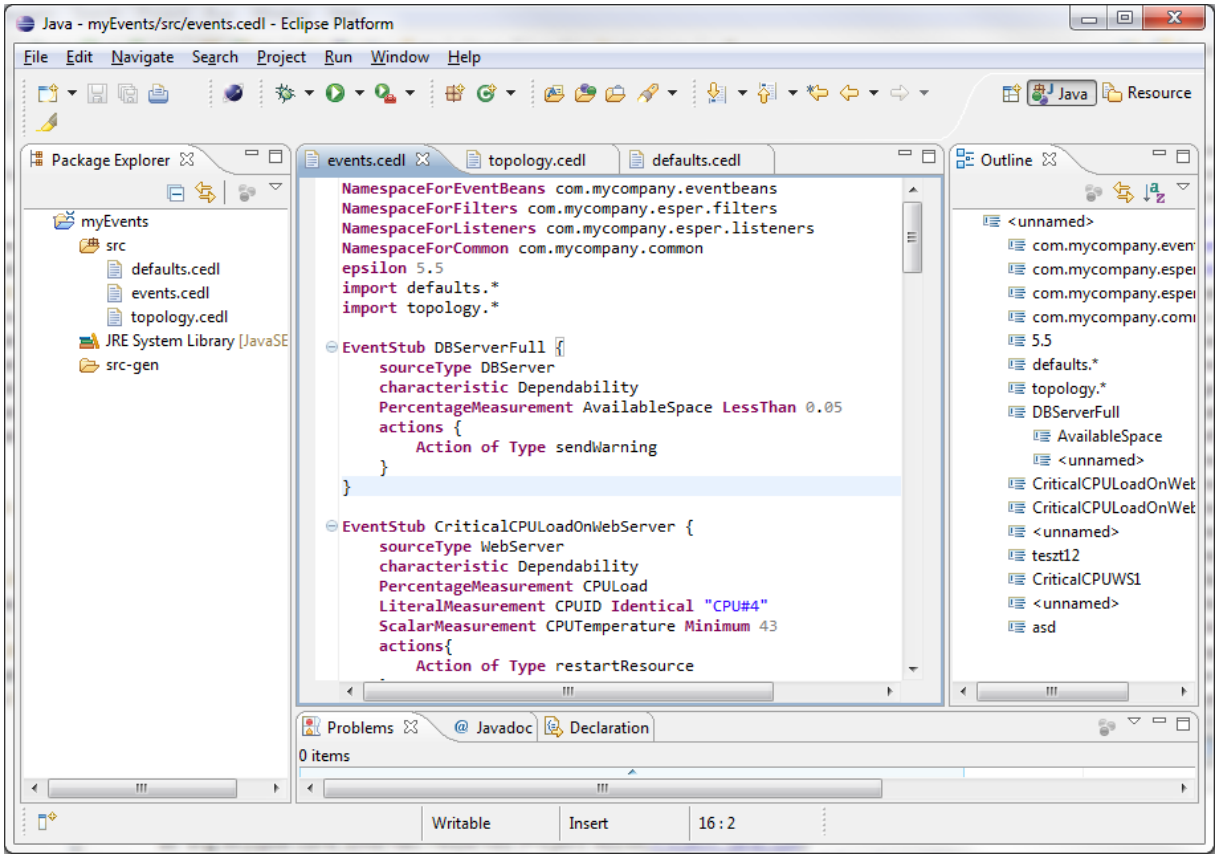
A modellezéshez egy saját fejlesztésű eszközt használható, ami korszerű Eclipse alapú technológiákra épül. A szöveg alapú modellezés támogatásához az Xtext2.0 keretrendszert használtam, ennek segítségével definiáltam magát a CEDL nyelvet. Az Xtext erőteljes megoldás a területen: az Eclipse platformmal integrálva a szakterület-specifikus nyelv definiálása után automatikusan előállít egy egyszerű fejlesztőkörnyezetet, amit igény szerint lehet továbbfejleszteni. Az alap környezetet *modellvalidációs szabályokkal*, *kódkiegészítéssel* és

[5] A pontos számok: 2.94 kódsorok, valamint 2.90 utasítások tekintetében.

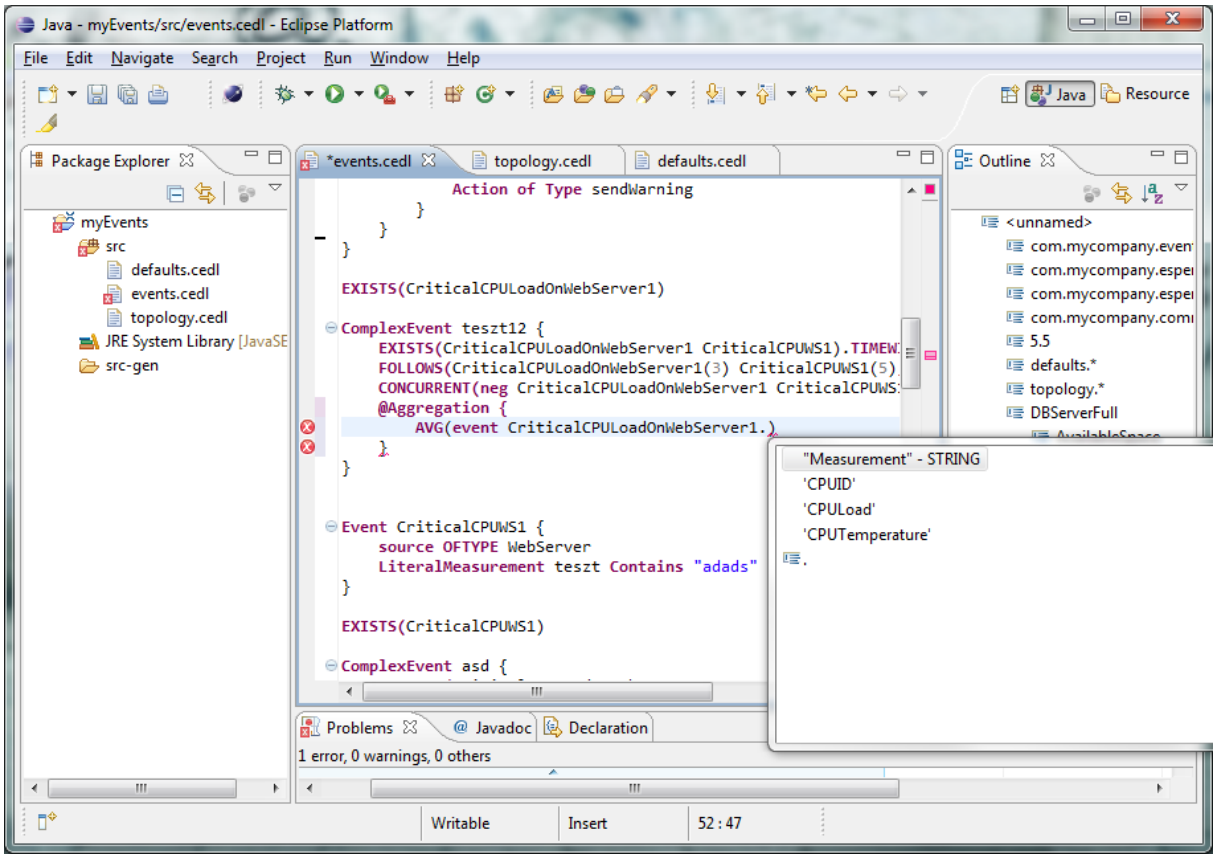
szerkesztési idejű *automatikus forráskód-generálással* egészítettem ki. A kódgeneráláshoz az XTend technológiát alkalmaztam.

A fejlesztés során felhasznált források: [32] [14] [15] [16] [33].

A 12. ábraés 13. ábra látható az eszköz felhasználói felülete. A bal oldali *Package Explorer* nézetben látható, hogy egy általános projekt létrehozása után *.cedl* kiterjesztésű állományokat tudunk létrehozni, majd abban modellezni. A *src-gen* mappában a modellezéssel párhuzamosan, automatikusan előállított forrásfájlok találhatók, amelyek ebben az esetben az események vázát leíró *EventBean* osztályok, a mintákat leíró *Filter* osztályok, valamint az akciókat definiáló *UpdateListener* osztályok lesznek, az Esper konvenciói szerint.



12. ábra – A komplex események modellezésére kifejlesztett eszköz felhasználói felülete.



13. ábra – A kódkiegészítő funkcionalitás használat közben: hatékony segítség a tervező munkájához.

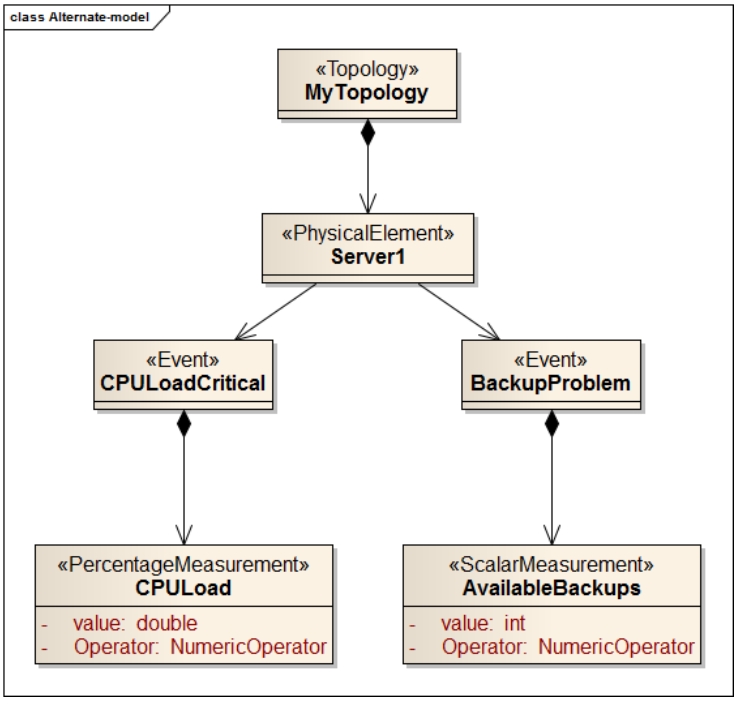
3.9. Egy modellezési alternatíva: az UML osztálydiagram

A 3.1. fejezet rámutatott, hogy a szöveg alapú modellezési megközelítés a ráfordított időt tekintve hatékonyabb alternatívának bizonyul a grafikus modellezéssel szemben. Ebben a fejezetben bemutatom a 3.3. szakaszban már látott modell UML osztálydiagram alapú, grafikus változatát, ami mögött ugyanaz a nyelvi modell található, tehát leíróerejük praktikusan azonos.

A megvalósítandó modell:

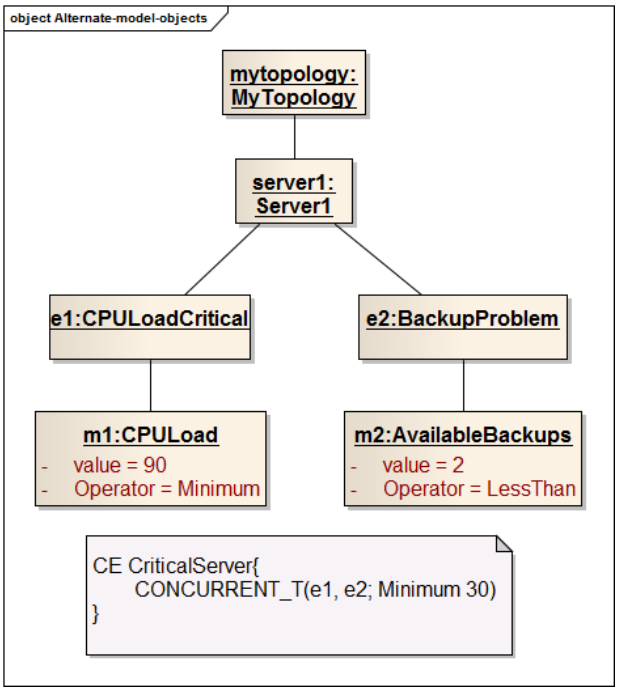
```
Event CPULoadCritical {
	source Server1
	PercentageMeasurement CPULoad Minimum 90
}

Event BackupProblem {
	source Server1
	ScalarMeasurement AvailableBackups LessThan 2
}

ComplexEvent CriticalServer{
	CONCURRENT_T(CPULoadCritical BackupProblem; T: Minimum 30)
	action {
		Action of Type sendWarning
	}
}
```

A megvalósítás *UML profiling* segítségével történik, azaz a metamodell elemeket a megfelelő sztereotípiák alkalmazásával tudjuk használni. [34] Először egy osztálymodellt kell létrehozni az egyes entitások megadásával, majd ebből egy objektummodellt kell példányosítani.



14. ábra – UML osztály diagram formalizmussal megfogalmazott esemény-metamodell.

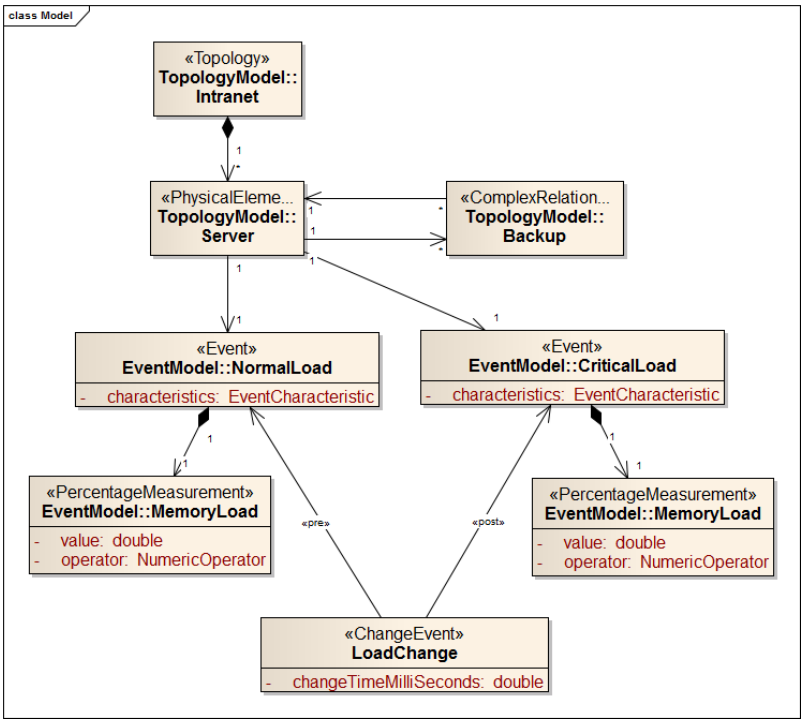
A 14. ábra látható, hogy a topológia alá létrehoztuk a *Server1* entitást, aminek két eseményét vizsgáljuk. Hogy ezeknek mik a paraméterei, az objektum modellben kerülnek megadásra, ahogy a komplex esemény is, egy OCL-szerű kényszer segítségével.



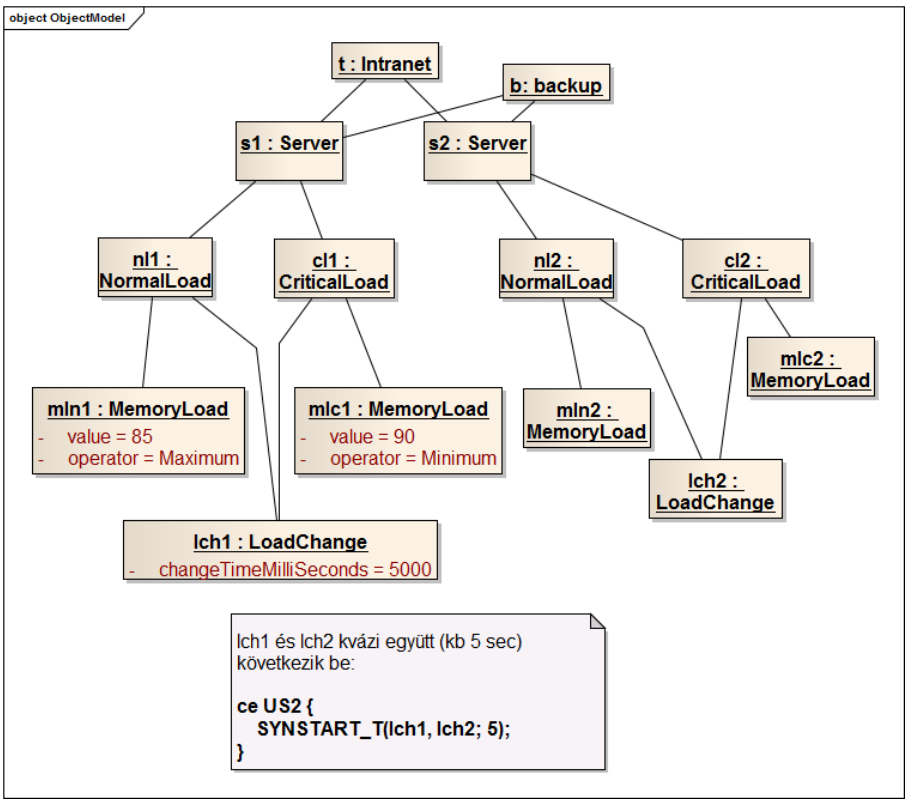
15. ábra – UML objektum diagram formalizmussal definiált eseménymodell.

Az esettanulmány egy modellje

A 2.2. fejezetben bemutatott esettanulmány 2. követelményének CEDL segítségével történő megvalósítása látható az 5. fejezetben. Összevetési alapként, ugyanennek a követelmények egy UML alapú megfogalmazása látható a 16. ábraés 17. ábra.



16. ábra – Az esettanulmány 2. kritériumának metamodellje.



17. ábra – Az esettanulmány 2. kritériumának eseménymodellje.

Következtetés

A fentiekből azt a következtetést vonhatjuk le, hogy már ilyen egyszerű, kisméretű mintáknál is nagy különbség van a szöveges modellezés és a rendelkezésre álló standard grafikus modellező módszerek időigényében, mégpedig mindenképpen előbbi javára.
A grafikus modellezés előnye, az áttekinthetősége nem érvényesül bizonyos méretű minták felett, így egyértelműen a szöveg alapú modellezés a jó megoldás a területen.

Fontos megjegyezni, hogy itt csak a standard, ipari szabványok által definiált grafikus modellezési lehetőségeket vizsgáltam. A továbbfejlesztési irányok között (7. fejezet) szerepel egy olyan szakterület-specifikus grafikus szerkesztő terve, ami a CEDL szöveges funkcionalitását grafikus lépésekkel tudja támogatni.

4. Szakterület-specifikus információk: a szemantikus tudásbázis

A gyakorlati esetekben jellemző probléma, hogy a megcélzott rendszerek mérhetői metrikái több százas, akár ezres nagyságrendűek lehetnek. (Lásd: VMWare ESX szervermetrikák. [35]) Ennyi metrikát (potenciális eseménytípust) átlátni nehéz feladat, így ezek kezelése nehezíti a modellezés folyamatát. Az sem elhanyagolható, hogy minél több alkalmazható metrika áll rendelkezésre, annál pontosabban írhatók körül az eseményminták, így azok definícióinak bonyolultsága növekszik.

Az szakterület szabványai és jogszabályai kijelölik azokat a magas szintű üzleti kritériumokat, amelyeket egy-egy komplexesemény-mintával ellenőrizni kell.[6] Ezek formalizálásával és elemzésével azonosíthatóvá válnak a lényeges metrikák, amelyeket automatizált leképezések segítségével a CEDL szintjén is elérhetővé tehetünk a tervező számára.

Példák magas szintű üzleti kritériumokra

Az információs rendszerek biztonságos üzemeltetésének legismertebb *szabványgyűjteménye* a COBIT. [36] Néhány ebben definiált kritérium:

- Gondoskodni arról, hogy a kritikus és bizalmas információkhoz ne lehessen jogosulatlanul hozzáférni.
- Minden informatikai vagyonelem nyilvántartása es védelme.
- Gondoskodni arról, hogy az informatika betartja a törvényeket, jogszabályokat és szerződéseket.

A tőzsde a piac szigorúan szervezett és szabályozott megjelenési formája. Számos kapcsolódó *törvényt* lehetne említeni, amelyek a kereskedés formai kereteit, a szereplők jogköreit, vagy épp az informatikai kiszolgáló infrastruktúrákkal szemben támasztott követelményeket rögzítik. [37] [38]

Egy példa az Budapesti Értéktőzsde informatikai rendszerekre vonatkozó szabályaiból:

- Az alkalmazott kontrolloknak, a Tőzsde Üzletmenet-folytonossági és Katasztrófa utáni helyreállítási terveit is beleértve, az általános értelemben vett működési kockázatokat is figyelme kell venniük. [39]

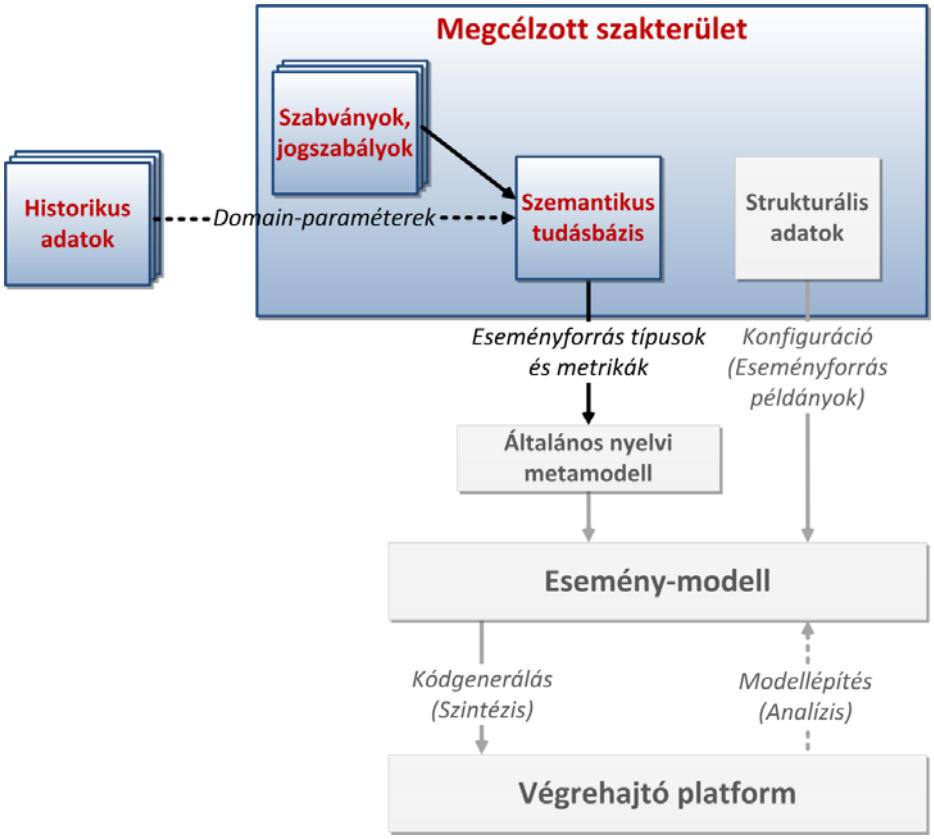
A felsorolt követelmények láthatóan magas szintűek, az informatikai rendszerek tervezésének és megfigyelésének alapját képezik.

A szemantikus tudásbázis

A 3.5. fejezet bemutatta, milyen módon lehet hasznosítani a megcélzott szakterület strukturális adatait az esemény-modell konfigurálására. A strukturális adatok leképezése előtt azonban szükséges bizonyos szakterület-specifikus információk kinyerése, mint például a strukturális adatok típusai. Ezeknek forrása a *szemantikus tudásbázis* (Semantic Knowledge Base – SKB), amelynek feladatai a CEDL szempontjából a következők:

[6] Fontos megjegyezni, hogy az *üzleti kritérium* fogalom ebben a kontextusban az angol *business requirement* fogalomnak felel meg, aminek jelentése némileg eltér a magyarban megszokottól.

- **A megcélzott szakterület alapvető fogalmainak definiálása**
  A szemantikus Tudásbázisból nyerhetők ki a *megcélzott szakterület alapvető fogalmai*, amelyeket a strukturális adatok példányosítanak. (Például egy szakterület-specifikus fogalom a *naplózást végző szerver*, ennek egy példánya az *X szerver, ami az infrastruktúrában található*.)
- **Metrikák definiálása**
  A szemantikus Tudásbázis az alapfogalmakkal kapcsolatban definiálja, hogy mik az *érdekes és releváns metrikák*, amelyeket mérni szeretnénk. A metrikák kapcsán definiálja a formátumot (skalár, százalékos, stb.), valamint a konkrét értékeket. (Például: a *processzor kritikus terheltsége* egy százalékos metrika, ami azt jelenti, hogy a processzor *90%* feletti terheltséget mutat.)
  A metrikák definiálásához korábban mért, historikus adatok használhatók fel.



18. ábra – A szakterület-specifikus információkat tároló szemantikus tudásbázis és a kapcsolódó források.

A szemantikus tudásbázis implementációs szempontból egy *ontológia*. A szemantikus megközelítés előnye a relációs adatbázisokhoz képest, hogy az ontológiakezelő eszközök következtető logikáinak segítségével a tárolt tudáshalmazon automatizáltan mutathatók ki komplex összefüggések, aminek előnyei akkor mutatkoznak meg a gyakorlati alkalmazásokban, ha metaszinteken átívelő szabályokat kell ellenőrizni. (Például: *„Létezik-e minden X erőforrástípushoz legalább két darab példány, amelyek között a kapcsolat a Tartalék típus példánya?”*)

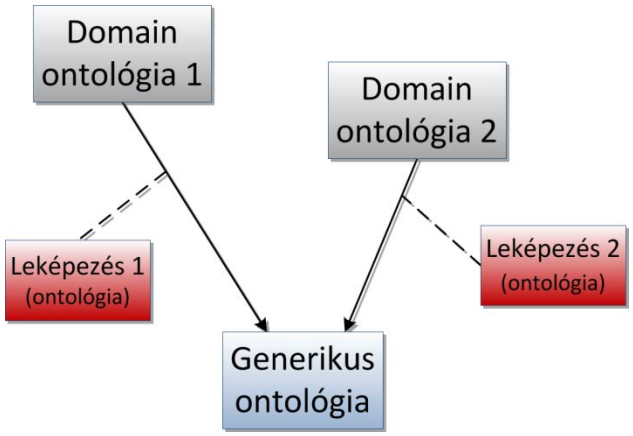
Az ontológiákban tárolt tudás más platformra történő transzformálásával többek között a [40] és [41] munkák foglalkoznak.

4.1. A szemantikus tudásbázis struktúrája és felhasználása

A szakterület-specifikus tudás tárolása strukturálisan több összekapcsolt ontológia segítségével valósul meg. A megközelítés hátterében az áll, hogy nem tudunk általános ontológia-tervezési mintákat és szabályokat adni minden szakterület információinak definiálásra (hiszen azok karakterisztikusan eltérőek lehetnek), valamint rendszerint már létező ontológiákkal kell dolgozni, melyek struktúrája kötött.
Célszerű ezért kettéválasztani az eddig szemantikus Tudásbázisként hivatkozott logikai egységet a következő két részre:

- **Domain (szakterületi) ontológia**
  Az adott szakterületet írja le, tulajdonképpen tetszőleges formában. Típusok, fogalmak, kapcsolatok, szabályok szerepelnek benne. Minden szakterület saját Domain ontológiával rendelkezik. (Például: biztonsági ontológiák [42], tőzsdei kereskedés jogszabályainak ontológiája.)
- **Generikus ontológia**
  A CEDL platform támogatásának szem előtt tartásával kifejlesztett ontológia, amelyre a megcélzott terület Domain ontológiája leképezhető. A Generikus ontológia minden esetben ugyanaz, függetlenül a megcélzott területtől. A dolgozatban bemutatok egy példát ilyen ontológiára.

A két egység összekapcsolása a Domain ontológia elemeinek a Generikus ontológia elemeire történő leképezéssel történik, amit egy újabb ontológia segítségével definiálhatunk. A szakirodalom ezt a megközelítést *szemantikus integrációnak* nevezi. [43] [44]



19. ábra – Két Domain ontológia leképezése a Generikus ontológiára további két leképező ontológia segítségével.

A szétválasztás előnye, hogy a 18. ábra látható transzformációt a szemantikus tudásbázis és az Általános nyelvi metamodell között csak egyszer kell implementálni, ezután pedig elég a megfelelő Domain ontológiát integrálni a Generikus ontológiával.

A Domain ontológiák struktúrája

A Domain ontológia struktúrájáról nem tudunk semmi biztosat kijelenteni, hiszen ha készen kapjuk, akkor felépítése bármilyen lehet, ha viszont nekünk kell elkészíteni, akkor célszerűen azonnal a Generikus ontológia típusai alapján hozhatjuk létre. (Ha nincs más célunk vele, mint a CEDL platformon történő modellezést támogatni.) A továbbiakban azzal az, egyébként életszerű, feltételezéssel élünk, hogy a Domain ontológiában a Generikus ontológia elemeihez

hasonló jellegű elemek találhatóak, így a leképezés komolyabb nehézségek nélkül megoldható; magát a Domain ontológiát pedig fekete doboznak tekintem.

A megközelítés sikeres alkalmazhatóságára a [44] munka mutat be esettanulmányt.

A Generikus ontológia struktúrája

A Generikus ontológia struktúráját pontosan ismerjük. A CEDL platform nézőpontjából ez biztosítja az interfészeket a szakterület adataihoz.

A dolgozathoz elkészítettem a Generikus ontológiának egy prototípusát, amit az alábbi táblázatban ismeretetek. A jobb érthetőség kedvéért és az általánosság szintjének bemutatása végett két lehetséges szakterület (IT-rendszerek monitorozása, illetve Tőzsdei kereskedés) fogalmaival mutatom be az egyes elemeket.

(A generikus elemek elnevezésénél a prototípusban szereplő angol neveket használom. Az értelmezést lásd a táblázat alatt.)

Generikus ontológia elem	Példák: IT	Példák: Tőzsde
SourceType	*Webszerver*, *Adatbázisszerver…*	*Részvény, Kötvény…*
Quantifier	*terheltség, megbízhatóság…*	*variancia, szórás…*
Metrics	*m1*(Webszerver, terheltség)	*m2*(Részvény, variancia)
Range	*r1*(0.9, 1)	*r2*(0, 0.1)
Qualifier	*kritikus*(m1, r1)	*megbízható*(m2, r2)
ComplexRelationship	*tartalék*(Webszerver, Webszerver)	-
ComplexGroup	*fürt*(Webszerver, Adatbázisszerver)	*portfólió*(Részvény, Kötvény)
Constraint	*c1*(fürt, legalább_egy_Webszerver)	*c2*(portfólió, szuboptimális)

2. táblázat – A Generikus ontológia elemei, valamint egy-egy példa különböző szakterületekről (IT-rendszerek monitorozása, valamint Tőzsdei kereskedés).

A *SourceType* elem a szakterület azon típusait tartalmazza, amelyek események forrásaként funkcionálhatnak. A *Qualifier* egy minőségi mutató, amit a *Metrics* (azaz Metrika) kapcsol össze az eseményforrásokkal. Ezzel a hármassal leírható például, hogy létezik metrikánk a Webszerverek terheltségének, vagy a Részvények varianciájának mérésére. (Lásd a táblázat példáit.) A prototípusban csak numerikus mennyiségeket alkalmazó metrikákat definiáltam, de elképzelhetőek szöveges, vagy logikai (Boole) metrikák is.

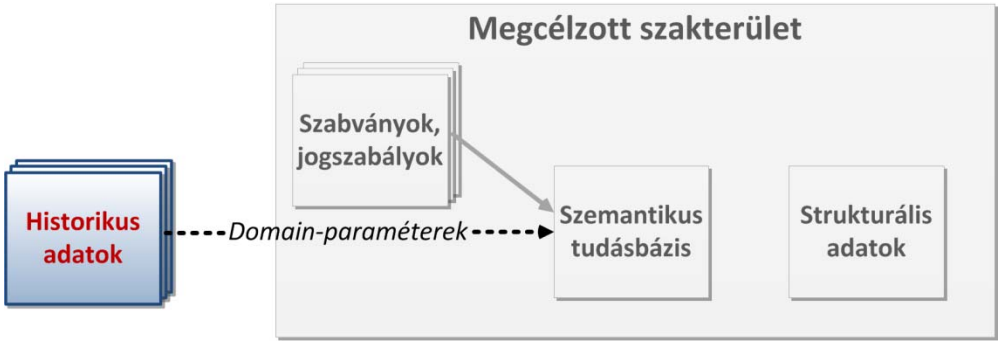
A *Range* elem egy olyan tartományt definiál, amit a *Qualifier* (azaz Minősítő) kapcsol össze egy Metrikával. Így már azt is le tudjuk írni, hogy egy Webszerver terheltsége milyen értéktartományban kritikus, vagy hogy egy Részvény mikor mondható megbízhatónak varianciája alapján. (Lásd a táblázat példáit.)

A forrástípusok (*SourceType*) közötti kapcsolatok, ahogy a CEDL-ben, itt is kétfélék lehetnek: irányított egy-egy (*ComplexRelationship*), vagy tartalmazási több-több (*ComplexGroup*) kapcsolatok.

Végül a *Constraint* egy olyan szabályt definiál, amit jellemzően kapcsolatokra, forrástípusokra, vagy metrikákra alkalmazhatunk. A táblázat példáinak jelentése: minden *fürt* kapcsolatban *legalább egy Webszerver típus található*; illetve: a *portfólió* típus *szuboptimális*.

A *„legalább egy Webszerver”* és *„szuboptimális”* fogalmi elemek két helyen kerülhetnek definiálásra: a Domain ontológiában, valamint a leképezést biztosító ontológiában. Korábban kikötöttük, hogy a Domain ontológiát fekete dobozként kezeljük, így a választás számunkra most érdektelen.
További lehetőség a kényszerek definiálására az OMG SBVR szabványa. [45] Használatára esettanulmányt a [40] munka mutat be.

Historikus adatok ontológiája
A 2. táblázatban szereplő *Range* elem a metrikákat minősítő tartományokat jelöli. Például azt, hogy egy Webszerver terheltsége mikor kritikus, vagy mikor van rendben. A tartomány határait első közelítésben természetesen becsléssel, vagy ismert bechmarkok alapján is megadhatjuk, ennél azonban jobb módszer a rendszer eddig mért, historikus adataiból ontológiát építeni, majd azt a Domain ontológiához hasonló módon becsatolni a szemantikus tudásbázisba.



20. ábra – Historikus adatok ontológiájának leképezése a szemantikus tudásbázisba.

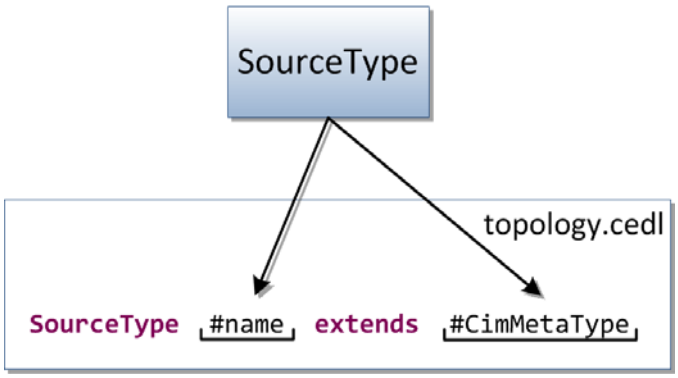
Ezzel elérhető, hogy a metrikákat minősítő tartományok a mérési adatok visszacsatolásával a lehető legpontosabban jellemezzék az adott célterületet.

4.2. Típusok és metrikák kinyerése a CEDL platform számára

A 18. ábra látható módon a szemantikus tudásbázisból metrikákat és forrástípusokat tudunk kinyerni, ezáltal a modellezés bizonyos lépéseit automatizálni.

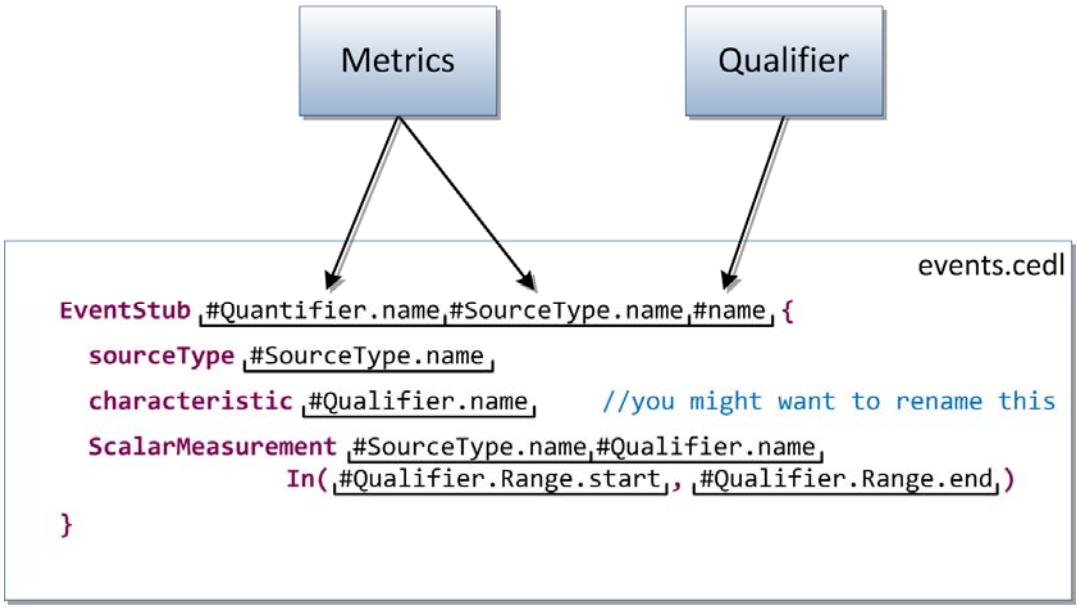
A CEDL platformon használható forrástípusok előállítását mutatja a 21. ábra. A *SourceType* elem nevéből és CIM metatípusából előáll a definíció.

A CIM metatípus a 9. ábra látható Topológia metamodell alapján fizikai, vagy logikai egység, illetve szolgáltatás lehet. Ezt a paramétert egyaránt kezelhetjük a leképezést megvalósító logikában, vagy ontológia-szinten. Mivel érdemi szempontból ez nem lényeges, a prototípus során ezzel a kérdéssel nem foglalkoztam.



21. ábra – Eseményforrások leképezése a CEDL platformra.

A metrikákból Eseménycsonkokat állíthatunk elő, amit Események és Komplex események implementálhatnak. Ezt a megközelítést mutatja be a 22. ábra.



22. ábra – A metrikákból az Eseménycsonkokat állítunk elő, amelyet Események és Komplex Események implementálhatnak.

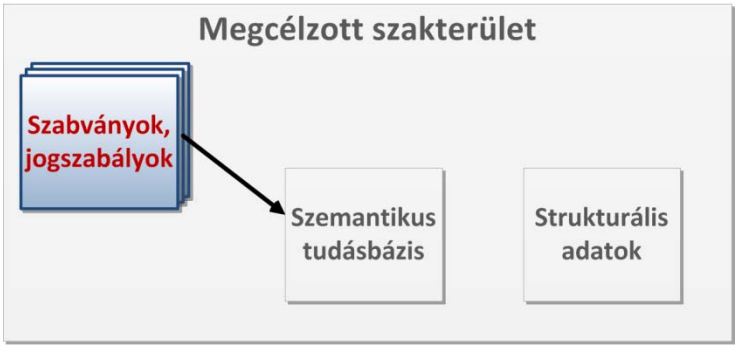
A 2. táblázat alapján az IT-szakterületre a következő forráskódot lehet előállítani az ontológiából a CEDL platformra:

```
SourceType Webszerver extends PhysicalElement

EventStub TerheltségWebszerverKritikus {
       sourceType Webszerver
       characteristic Kritikus
       ScalarMeasurement WebszerverKritikus In (0.9, 1)
}
```

4.3. Szabványkonformitás-ellenőrzés

A következtető eszközök segítségével lehetőség van saját verifikációs lekérdezéseket definiálni és futtatni. Egy ilyen lekérdezés lehet például, hogy létezik-e minden kritikus szerverhez tartalék. A következtető motor a fogalmi hierarchiát végigjárva képes feldolgozni a *kritikus*, a

szerver és a *tartalék* fogalmakat, valamint a *létezik(kritikus_szerver, tartalék_szerver, tartalék)* relációt.



23. ábra – Szabványokat és jogszabályokat leíró ontológiák leképezése a szemantikus tudásbázisba.

Az ilyen verifikációs lekérdezéseket a Generikus ontológia *Constraint* elemei tárolják. Így természetesen arra is van lehetőség, hogy a Domain ontológia mellett a *Constraint* elemekhez egy olyan ontológiából képezzünk le szabályokat, amelyek a szakterület szabványait, irányelveit, jogszabályait írják le. Ilyenek találhatók például a [46] és [47] munkákban.

Miután a szabályozó ontológiát becsatoltuk, a Generikusra korábban a Domain ontológiából leképzett elemeken végezhetünk el verifikációt, ezáltal kimutathatóvá válik az egyes szakterület-specifikus szabványok és jogszabályoknak való megfelelőség.

4.4. Összefoglalás

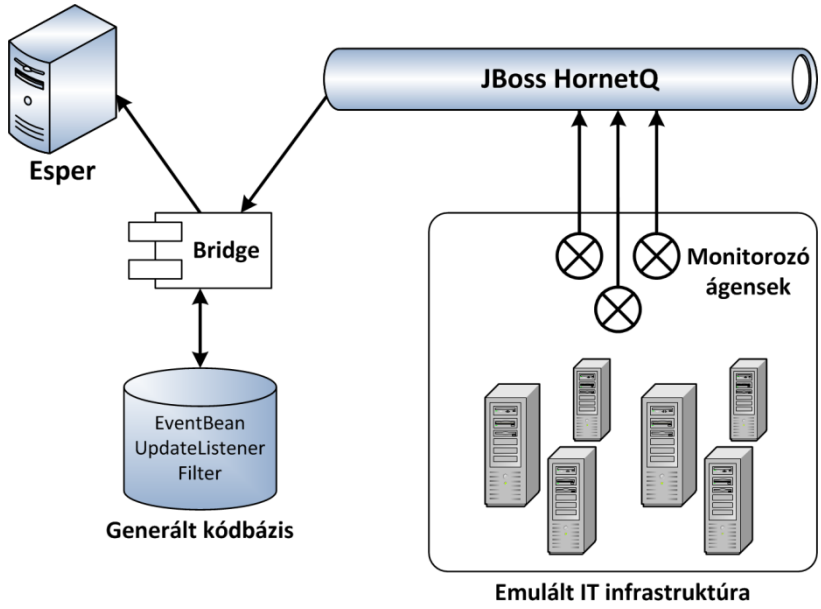
A fejezet során láthattuk, hogy ontológiák segítségével nagyon rugalmasan tudjuk kezelni a szakterület információit, a következtető motorok segítségével magas szintű követelményeket tudunk ellenőrizni a szemantikus modelleken, a szemantikus tudásbázisból származó információkkal pedig a komplex események modellezésének bizonyos lépéseinek automatizálására is lehetőség van.

5. Megvalósítás

Ebben a részben a korábban (2.2. fejezet) már ismertetett esettanulmányon keresztül mutatom be a munkafolyamatokat a modellezéstől a generált forráskódok telepítéséig.

5.1. Tesztkörnyezet

A módszerek és eszközök teszteléséhez egy olyan tesztkörnyezetet alakítottam ki, ami egy jellemző nagyvállalati infrastruktúrát emulál. Ennek felépítése látható az alábbi ábrán.



24. ábra – A tesztkörnyezet.

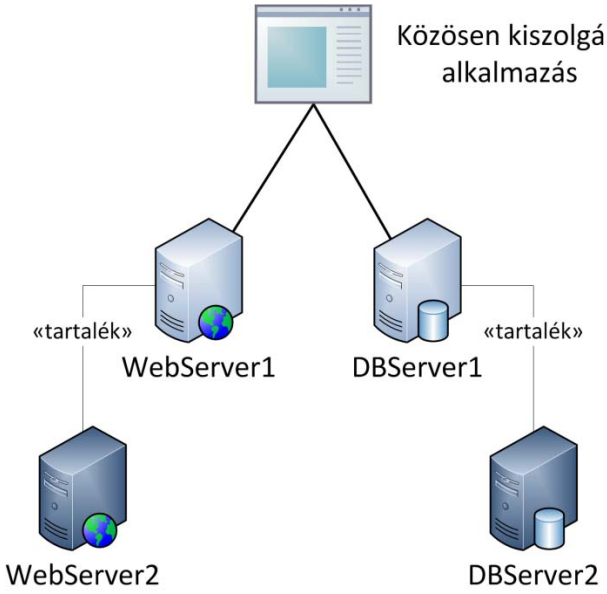
Az események feldolgozására az Esper platformot választottam, ami egy nyílt forráskódú, Java alapú szoftver, jól használható külső interfészekkel (*Esper API*). Az eseményeket a *Bridge* nevű komponens továbbítja a platformra, amelynek feladata, hogy a nyers formában (*raw data*) érkező eseményfolyamból előállítsa az Esper által támogatott formátumú esemény objektumokat. Ez a formátum többféle is lehet: XML, JSON, vagy a Bean-konvenciókat követő Java osztály (*POJO – Plain Old Java Object*), amelyek közül utóbbit választottam, ezért a továbbiakban az eseményeket leíró osztályokra *EventBean* néven is hivatkozom a dolgozat során.

Az események az emulált IT infrastruktúrából származnak, amiket a monitorozó ágensek gyűjtenek össze és egy *JBoss HornetQ* alapú aszinkron kommunikációs csatornára (üzenetsor) küldik azokat. Az üzenetsor feladata biztosítani, hogy minden esemény megérkezzen a *Bridge* komponenshez, valamint hogy az események sorrendezhetők legyenek. (Az ipari kommunikációs köztesréteg megoldások ezekre a funkciókra nagy hangsúlyt fektetnek.)

A *Bridge* komponens úgy kapcsolja össze az eseményfolyamot a feldolgozó platformmal, hogy közben adatkonverziót is végez. Megvalósítását tekintve ez egy saját fejlesztésű, Java alapú megoldás. Feladatát tekintve, éles környezetben értelemszerűen kritikus, hogy a komponens mennyire párhuzamosítható, hiszen mind az Esper, mint a HornetQ kifejezetten párhuzamos működésre lett tervezve, így a *Bridge* ennek az infrastruktúrának a szűk keresztmetszete.

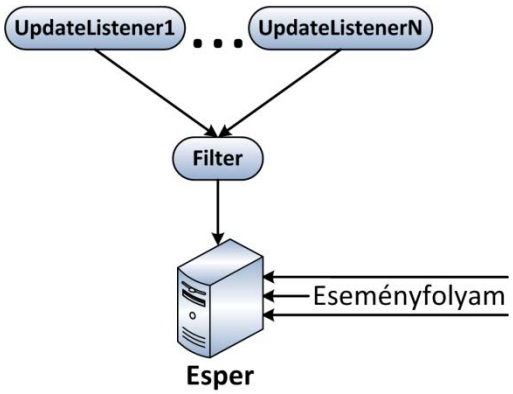
A dolgozat fókuszának szempontjából nem volt kifejezetten releváns a teljesítőképesség értékelése, ezért ezzel a dolgozat nem is foglalkozik.
A tesztkörnyezet összeállításához a következő forrásokat használtam fel: [48] [18] [19]

Az Esettanulmány megvalósításához a 24. ábra emulált IT infrastruktúráját a következőképpen kell összeállítani.



25. ábra – Az Esettanulmány IT infrastruktúrája.

Az alábbi ábra azt mutatja be, hogy történik az események feldolgozása az Esper platformon.



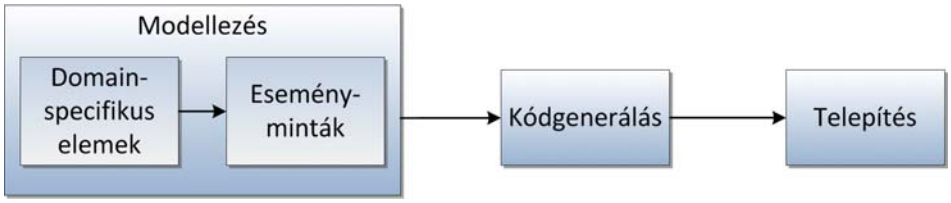
26. ábra – *Filter* és *Listener* komponensek a feldolgozóplatformon.

A beérkező eseményeket az Esper feldolgozza, mégpedig az úgynevezett *Filter* példányokban megfogalmazott *Statement* kifejezések alapján, amelyek az SQL nyelvhez hasonló lekérdezéseket definiálnak az eseményfolyamon az Esper mintaleíró nyelvének segítségével (EPL – *Event Processing Language*). A modellekben definiált eseményminták ide képződnek le a kódgenerálás során.

Ha egy *Filter* példányban foglalt eseményminta bekövetkezik, akkor a feliratkozott *UpdateListener* példányok végrehajtják a bennük definiált eljárásokat. Ez utóbbi osztályokba képződnek le a modell *Action* elemei, ezek definiálják a végrehajtható eljárásokat.

5.2. Munkafolyamat

A továbbiakban a következő munkafolyamatot követem.



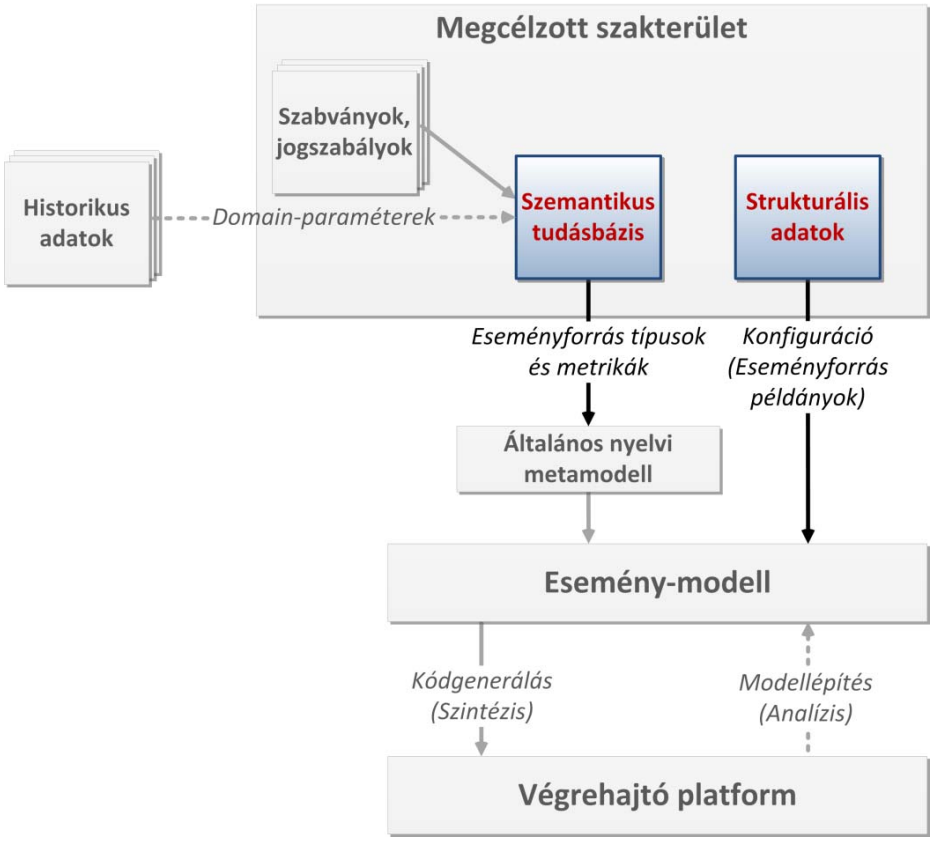
27. ábra – A munkafolyamat.

Az első lépésben az esettanulmányban foglalt követelmények alapján, a korábban már bemutatott módszerekkel elkészítem az eseménymodellt. Ezután megvizsgálom a fejlesztőeszköz által előállított forráskódokat, majd telepítem azokat.

5.3. Modellezés

A modellezés során nem áll rendelkezésünkre konfigurációmenedzsment adatbázis, sem szakterületi ontológia, ezért szakterület-specifikus információkat és a konfigurációt manuálisan veszem fel a modellbe.

Szakterület-specifikus elemek



28. ábra – A modellezés első lépése a szakterület-specifikus információk becsatolása.

A 25. ábra alapján vegyük fel az erőforrástípusokat, az erőforrások közötti kapcsolattípusokat, valamint az akciótípusokat! Ezek az információk a szakterület-specifikus ontológiából származnak.

```
SourceType WebServer extends PhysicalElement
SourceType DBServer extends PhysicalElement

ComplexGroupType commonlyServedApplication

ActionType sendWarning {
	description "The action calls a message sender service."
	consoleInfo "Warning sent!"
}
```

A fenti definíciók szerint egy *WebServer* és egy *DBServer* típusunk van, amelyek a *PhysicalElement* típusból származnak. Ez egy olyan alaptípus, ami a 3.5. fejezetben már bemutatott topológia metamodellben van definiálva, és amely metamodell a CIM metasémát veszi alapul. [27]

A következő lépés a példányok felvétele: erőforráspéldányok és a köztük lévő kapcsolatok példányai. Ezek az információk egy konfigurációmenedzsment adatbázisból (CMDB – *Configuration Management Database*) származnak éles környezetben.

```
Source WebServer WebServer1
Source WebServer WebServer2
Source DBServer DBServer1
Source DBServer DBServer2

ComplexGroup ApplicationGroup1{
	type commonlyServedApplication
	topologyElements WebServer1 DBServer1
}

ComplexRelationship {
	type backup
	sourceTopologyElement WebServer1
	targetTopologyElement(s) WebServer2
}

ComplexRelationship {
	type backup
	sourceTopologyElement DBServer1
	targetTopologyElement(s) DBServer2
}
```

Példányosítottunk egy-egy web- és adatbázis szervert, egy web szervert és egy adatbázis szervert logikai kapcsolatba kötöttünk egy közösen kiszolgált alkalmazás által, valamint a szervertípusokat páronként egymás tartalékául jelöltük meg.

Ezek az alapvető információk most manuálisan kerültek be a modellbe, azonban automatizált eljárásokkal könnyen kivitelezhető az importálásuk a szakterület tudásbázisaiból (adatbázisok, ontológiák).

Eseményminták

A következő feladat az eseményminták definiálása. Az esettanulmány által ismertetett két eset közül a másodikat fogom modellezni (és az egyszerűség kedvéért ezt is csak a web szerver típusra), de természetesen ehhez hasonlóan az esettanulmány további részei is elkészültek.

Egymás tartalékául szolgáló szerverek terhelése egyszerre nő meg.

Az esemény leírásában csak annyit köt meg az esettanulmány, hogy azokat az eseményeket jelezzük, amelyek során *egyszerre nő meg* a szerverek terhelése, azt nem fejti ki, hogy pontosan mennyit is nő a terhelés. Az ilyen problémák feloldásával a 4. fejezet foglalkozik, itt most elégséges, ha megkötünk egy ésszerű értéket és azzal modellezünk tovább.
A továbbiakban az „egyszerre nő meg" események azt az esetét vizsgálom, amikor legalább *35%*-os a terhelésnövekedés.

Az persze továbbra is kérdés, hogy mit jelent, ha két esemény „egyszerre" következik be. Ezt a problémát a CEDL nyelvi szintjén kezeltem, ahol az egyidejűség leírására a CONCURRENT és a CONCURRENT_T operátorok használhatók. Utóbbi egy időablakot definiál, amin belül bekövetkező két eseményt kvázi egyidejűnek értékelünk, előbbi azonban nem definiál explicit időablakot, helyette az infrastruktúra kommunikációs minőségi jellemzőiből (késleltetés, csomagvesztés, stb.) származtatott értéket (*epsilon*) használja egy implicit időablak definiálásához. Ez az érték a tesztkörnyezetben 0.5 másodperc - az ezen belül bekövetkező két esemény egyidejűnek tekinthető.
(Bővebben lásd a 3.4. fejezetben, a *Komplexesemény-operátorok* szakaszban.)

Minden adott, hogy az eseménymintákat megfogalmazzuk.

```
EventStub CPULoadOnWebServer {
	sourceType WebServer
	PercentageMeasurement CPULoad
	LiteralMeasurement ID
}
```

Definiáltunk egy eseménycsonkot, amely a *WebServer* típusra előír egy *CPULoad* nevű mértéket, valamint egy azonosítót (*ID*) is definiál – ez utóbbinak segítségével tudjuk megkülönböztetni, hogy melyik szerver eseményéről van szó, hiszen a forrásnál ezt nem különítettük el most.

Ezután egy komplex eseményben olyan mintát fogalmazunk meg, ahol ezt a csonkot implementáló események együtt következnek be és *CPULoad* paraméterük különbsége legalább 35. Azt is előírjuk, hogy csak azok az események érdekelnek minket, amikor a két esemény más-más web szervertől származik.

```
ComplexEvent CPULoadOnBackups {
	FOLLOWS(CPULoadOnWebServer as e11 CPULoadOnWebServer as e12)
	FOLLOWS(CPULoadOnWebServer as e21 CPULoadOnWebServer as e22)
	CONCURRENT(e12 e22)
	ComplexRelationshipType backup
	@ImplementMeasurement{
		implementation {
			e11.'ID' Identical e12.'ID';
			e21.'ID' Identical e22.'ID';
			e11.'ID' NotIdentical e21.'ID';
		}
	}
	@Aggregation {
		DIF(eventStub e12.'CPULoad' eventStub e11.'CPULoad') MoreThan 35
		DIF(eventStub e22.'CPULoad' eventStub e21.'CPULoad') MoreThan 35
	}
	action {
		Action of Type sendWarning
	}
}
```

Az komplex esemény megfogalmazásánál először azt írtuk elő, hogy a létezzen az eseménycsonkot implementáló két-két esemény, amelyek páronként egymást követik (*e11* és *e12*, valamint *e21* és *e22*). Ez után előírtuk, hogy *e12* és *e22* események egyidejűleg következzenek be.

Megadtuk, hogy csak olyan esetekben releváns ez a minta, amikor az eseménycsonkot implementáló események olyan *WebServer* típustól származnak, amelyek backup kapcsolatban állnak.
Az *@ImplementMeasurement* blokkban definiáltuk, hogy mely események származnak ugyanattól a szervertől és melyek különböznek.
Az *@Aggregation* blokk írja le, hogy az azonos szervertől származó események *CPULoad* paramétere 35%-kal különbözik. A sorrendiség miatt ez itt +35%-ot jelent, azaz növekedést.
Végül az *action* blokk a végrehajtandó eljárást definiálja.

5.4. Az előállított forráskód

A modellek építésével párhuzamosan előálltak a telepíthető forráskódok állományai is.[7]

Az állományok telepítésének helye a 24. ábra látható *generált kódbázis*. A feldolgozás folyamatát menedzselő komponensek innen olvassák ki az osztályokat. A táblázatban szereplő „Telepítés helye” mezőben mindig a generált kódbázis gyökérkönyvtárához viszonyított relatív elérési útvonalat kell érteni. (Például ha a kódbázis az *org.mycompany.eventprocessing* névtérben helyezkedik el, akkor ezt minden esetben hozzá kell érteni a táblázat értékeihez.)[8]

[7] A generált forráskódok egy része megtekinthető a következő linken: http://dl.dropbox.com/u/44011277/cedl/src-gen.zip.

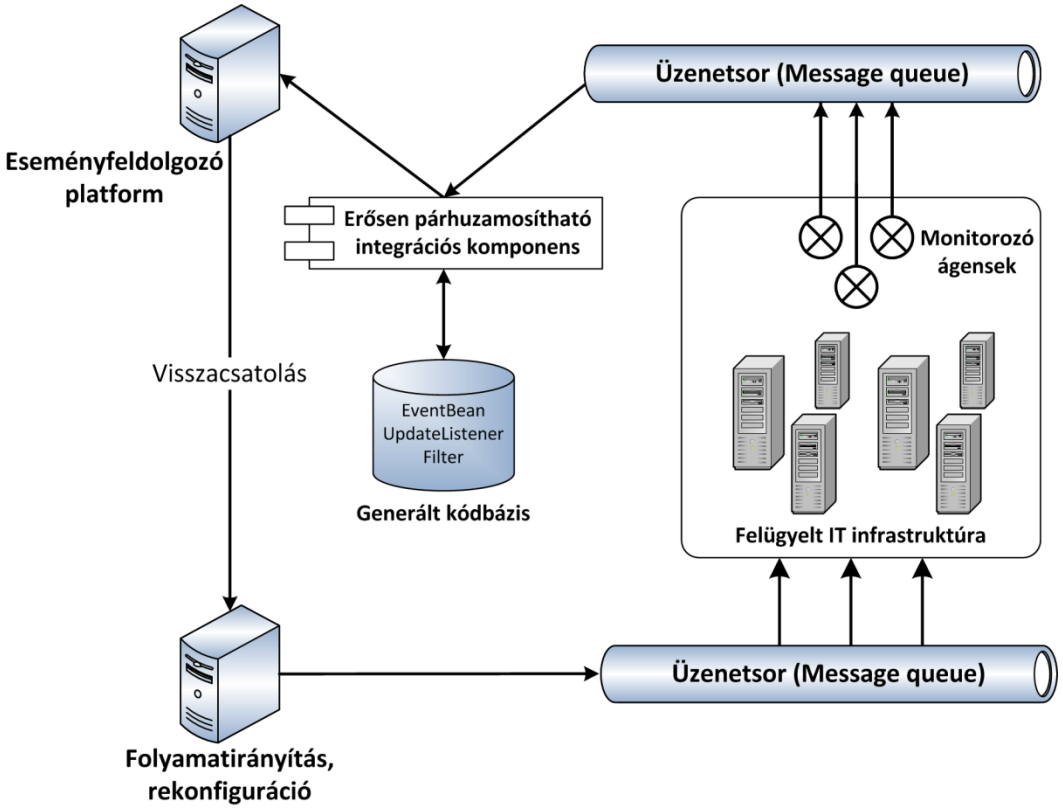
[8] Ezek a névterek (és így az elérési útvonalak) a modellek szintjén definiálhatóak a *NamespaceForEventBeans*, *NamespaceForFilters*, *NamespaceForListeners* és *NamespaceForCommon* nyelvi elemekkel.

Generált állomány		
Név [.java]	**Telepítés helye**	**Leírás**
IEventFilter	/common/	A modelltől függetlenül előállított interfész, ami a Filter osztályoknak biztosítja a Listener osztályok berögzítését lehetővé tevő funkcionalitást.
CPULoadOnWebServer	/eventbeans/ WebServer/multisource/	Az esemény reprezentálására alkalmas osztály. A névtérből kiderül, hogy a WebServer típushoz tartozik, azon belül azonban több forrásból is érkezhet, nem köthető exkluzívan csak egyhez.
CPULoadOnBackupsFilter	/esper/filters/complex/	A eseménymintát leíró Esper EPL (*Event Processing Language*) kifejezést tartalmazó osztály. Implementálja az IEventFilter interfészt.
CPULoadOnBackupsListener	/esper/listeners/complex/	Az akciót tartalmazó Listener osztály, amely ismeri az IEventFilter interfészt és ezen keresztül jegyzi be magát a CPULoadOnBackupsFilter eseményeire.

3. táblázat – A modellezés végén előállt forrásállományok, telepítési helyük és leírásuk.

5.5. Gyakorlati alkalmazhatóság és korlátok

Integráció nagyvállalati infrastruktúrákkal



29. ábra – Eseményfeldolgozás integrálása nagyvállalati környezetben.

Az eseményfeldolgozó rendszerek nagyvállalati alkalmazása jellemzően az automatizált döntéshozatal, vagy szabályozás támogatását célozza. Minden esetben kiemelt fontossággal kell kezelni az események megfelelő sorrendezését és az időzítés pontos kezelését, hiszen ezek az információk – mint az eddigiekben már láthattuk – a komplex események mintáinak

jelentését befolyásolják. A sorrendhelyes átvitel és a torzítatlan időzítés funkcionalitását biztosíthatjuk üzenetsorok alkalmazásával, amelyek ráadásul, aszinkron jelegüknél fogva teljesítmény szempontjából is rendszerint jó választásnak bizonyulnak. Hasonlóképpen: a szabályozás folyamatának kommunikációs csatornája számára is választható egy aszinkron csatorna (célszerűen ugyanaz az üzenetsor).

Az üzenetsor (*message queue*) teljesítményének optimalizálása
Az üzenetsor megoldások alapvetően két *kézbesítési modellt* kínálnak: sor (*queue*) és téma (*topic*). Előbbi annyit garantál, hogy az üzenetet továbbítja a kimenetre, ahol olvasásra kész állapotba kerül, utóbbi viszont az üzeneteket több címzetthez is el tudja juttatni: azokhoz, akik az adott témára feliratkoztak.
Technológia-specifikus nézőpontból vizsgálva: a JBoss HornetQ az *üzenet formátumára* is több lehetőséget nyújt. Továbbíthatunk üzenetet például objektumként, de akár byte-folyamként is.

Valós környezetben, gyakorlati problémáknál kritikus fontosságú az üzenetsort a fenti lehetőségek mentén optimalizálni. A dolgozat során ezzel nem foglalkoztam, mert a tesztkörnyezet nem produkált akkora teljesítménybeli terhelést, ami valós környezetben fordul elő, ezért a sor alapú modell és az objektumként továbbított üzenetek megfelelő választásnak bizonyultak.

Generált forráskódok futás közbeni telepítése
Az így előállított állományokat már telepíthetjük az eseményfeldolgozó platformra. A tesztkörnyezethez hasonlóan kialakított megoldás azonban nem képes futás közben, újraindítás nélkül új komponensek becsatolására. Erre egyszerű megoldást nyújt például az *Interceptor* tervezési minta. [49]

5.6. Összefoglalás

A fejezet során egy valós esettanulmány alapján bemutattam a komplex események modellezésének folyamatát az általam kifejlesztett modellezési platformmal és eszközzel. Ismertettem a modellezés során előálló állományokat és azok szerepét az Esper célplatformon.

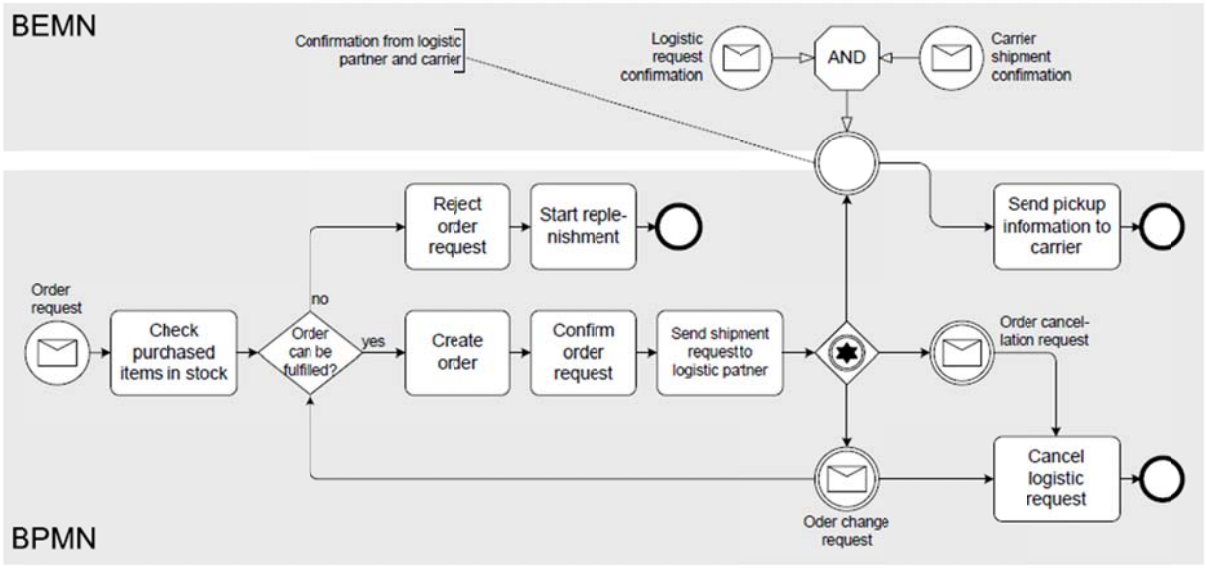
Láthattuk, hogy a fejlesztés valóban hatékony időben, hiszen a 3.9. fejezetben bemutatott UML alapú módszerrel elképzelhető modellekhez képest sokkal tömörebb modellek kerültek kialakításra.

A fejezet során továbbá bemutattam egy tesztkörnyezetet, ami irányt mutat a komplexesemény-feldolgozó eszközök integrációjára valós, nagyvállalati környezetekben.

6. Kapcsolódó munkák

6.1. BEMN – Business Event Modeling Notation

A [50] cikkben a szerzők grafikus modellezőnyelvet kínálnak komplex események definiálásához. A BEMN (Business Event Modeling Notation) nyelv az eseményminták dinamikájára koncentrál, a BPMN és a BPEL formalizmusát veszi alapul, valamint a BPMN eszköztárának kibővítését célozzák vele.



30. ábra – BEMN formalizmussal definiált komplex eseményt felhasználó BPMN folyamat. [50]

A grafikus felület nyújtotta előnyök mellett a BEMN a következő hátrányokkal rendelkezik az általam kifejlesztett CEDL nyelvhez képest:

- Nem biztosít egyértelmű leképezést valós eseményfeldolgozó platformhoz. Továbbá nem állítható elő belőle végrehajtható szemantika (mint például a BPEL).
- Az eseményminták dinamikáját ragadja meg, ugyanakkor a statikus struktúrával csak érintőlegesen foglalkozik. Eseményforrások a CEDL által definiált értelemben nem jelennek meg.
- Az előző pontból adódóan nincs lehetőség a szakterület-specifikus és statikus tudásbázisokból kinyert információk hasznosítására, ezért a modellezés nem támogatható ilyen vonatkozású automatizálással.
- A modellek manuálisan épülnek, ezért nagy problématér esetén fennállhat annak veszélye, hogy maga a modell hiányos, ezért hibásan írja le a valós problémát.

A megközelítés mindazonáltal jó irány lehet a CEDL grafikus továbbfejlesztése esetén.

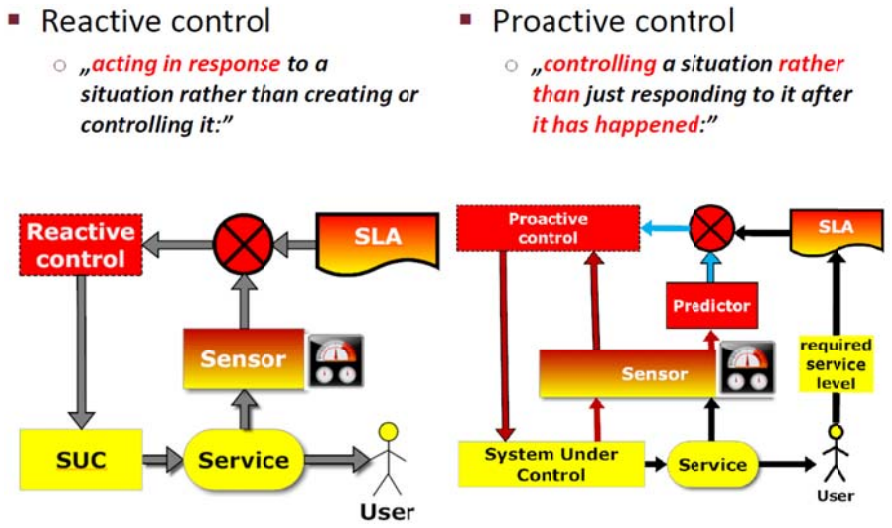
6.2. CoMiFin – Communication Middleware for Monitoring Financial Critical Infrastructure

A CoMiFin EU projekt kritikus pénzügyi infrastruktúrák biztonságtechnikai monitorozására fókuszált. A megvalósítás során olyan Java alapú köztesréteg implementációk készültek el, amelyek az eseményfeldolgozás elvét alkalmazzák, ezáltal képesek például online csalások felderítésére, vagy a *WS-Trust* szabvány által leírt partner-megbízhatósági előírások biztosítására. [51]

Dolgozatom végig azzal a feltételezéssel élt, hogy a célinfrastruktúrán vannak *valóban* megfigyelhető események és ezeket bizonyos monitorozó ágensek segítségével aggregálni tudjuk. A CoMiFin projekt fontos ide kapcsolódó eredménye, hogy irányt mutat az infrastruktúrák monitorozásának kialakítására, és a megfigyelhető mérési pontok biztosítására. [21] [22] [52] [53] [26]

6.3. Proaktív rendszerek tervezése

A [54] munka a reaktív és proaktív irányítási rendszereket állítja szembe és azt a kérdést vizsgálja, milyen hozzáadott eszközökkel és információkkal fejleszthető egy reaktív rendszer proaktívvá.



31. ábra – Reaktív és proaktív irányítási rendszerek. [54]

A cikk a megoldást egy predikciós logika bevezetésében adja meg, amely az aktuálisan mért adatok és a már rendelkezésre álló historikus adatok alkalmas felhasználásával valósidejű előrejelzésre képes az irányított folyamattal mutatóival kapcsolatban.

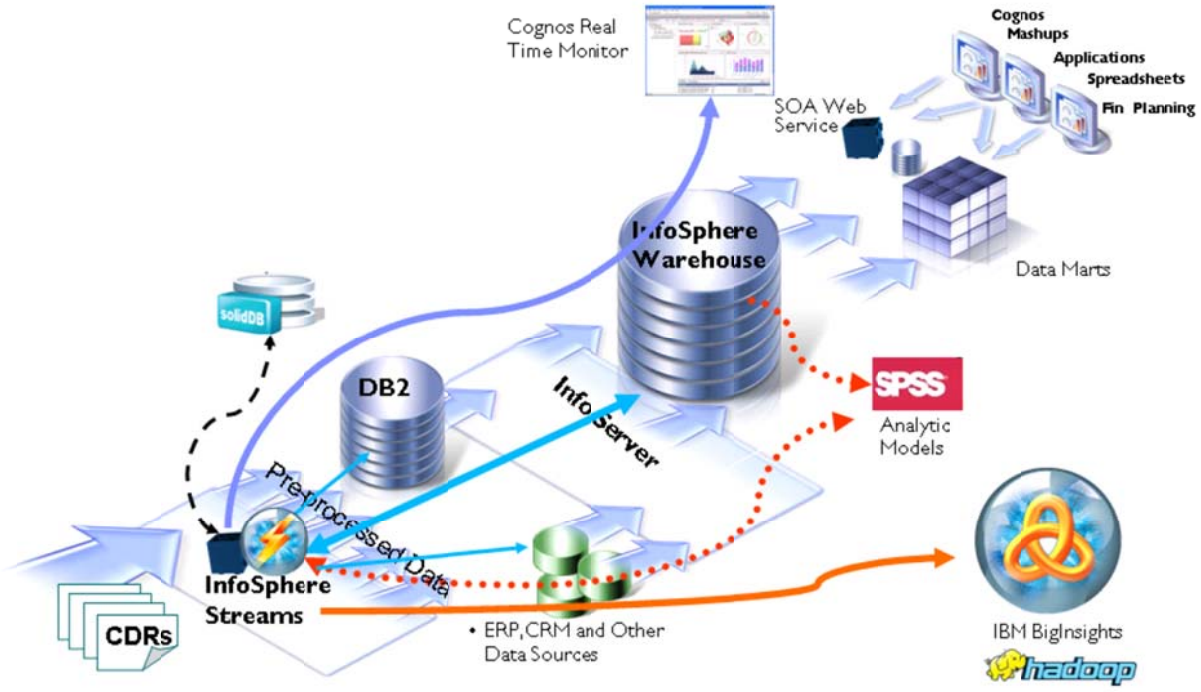
A dolgozatom során bemutatott ontológia alapú megközelítés képes a komplexesemény-minták rugalmas paraméterezését biztosítani, így ezzel a konstrukcióval egyfajta predikciós logikát definiálhatunk, hiszen az ontológiában a visszacsatolás nyomán olyan új eseményekre készülhetünk fel, amelyeket a kezdeti állapotban még kezeltünk, és amely események szignifikáns jövőbeli mintákat jeleznek előre.

A reaktív rendszerek válaszidejének csökkentésére is számos dolgozat született a korai előrejelzés (*early classification*) matematikai hátterének témakörében, amelynek segítségével a bekövetkező káros események minél előbbi felismerése válik lehetővé. [55] [56] [57] Természetesen az előbb ismertetett módon a reaktív rendszerek eseményeinek modellezése is lehetséges, ezáltal a korai előrejelzés komplexesemény alapú modellvezérelt támogatása.

6.4. IBM InfoSphere Streams

Az *InfoSphere Streams* IBM „nagyadat-platformjának" (*Platform for Big Data*) komplexesemény-feldolgozó eszköze. [58] [59] A Redbooks sorozat [60] kiadványa a lehetséges alkalmazásokat vizsgálja és többek között a 6.3. fejezetben bemutatott koncepció egy megvalósítását is bemutatja.

Az SQLServer 2008-ra épül a Microsoft StreamInsight megoldása, hasonló funkcionalitással. [12]



32. ábra – Az IBM Continuous Insight modellje. [60]

7. Továbbfejlesztési lehetőségek

Továbbfejlesztési lehetőségekről elsősorban a modellezőeszköz, valamint a kapcsolódó technológiai támogatás kapcsán van értelme beszélni. Az alábbiakban a fejlesztés két legvalószínűbb folytatását mutatom be.

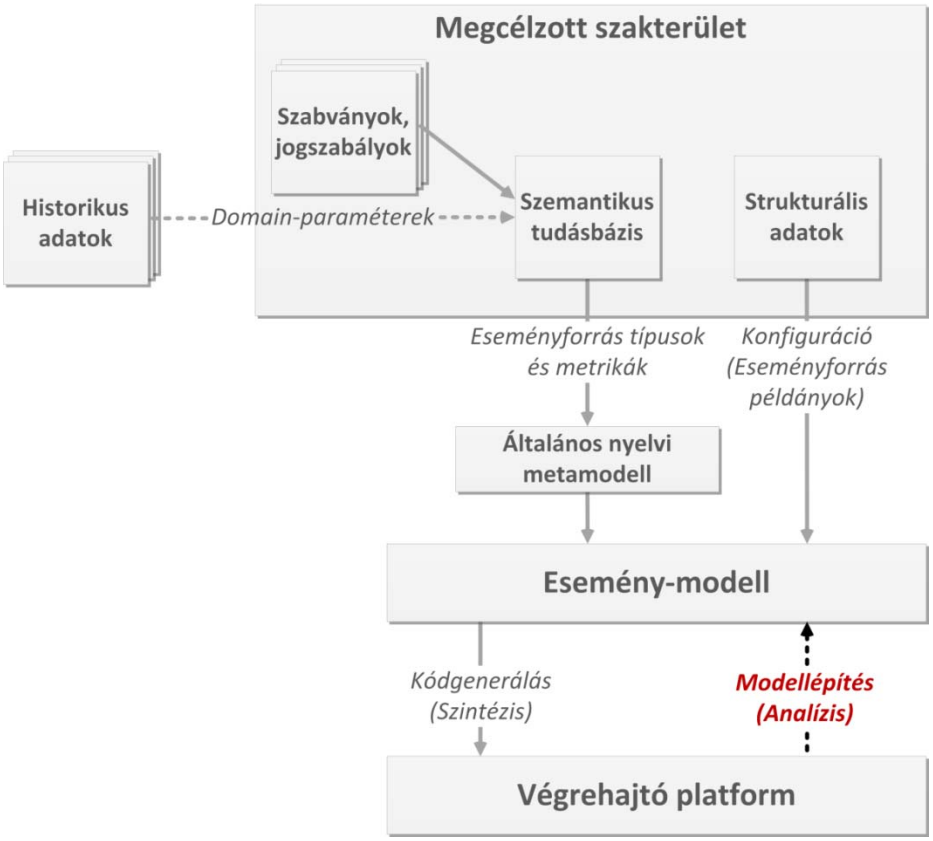
7.1. Proaktív irányítórendszerek fejlesztésének támogatása

A proaktív irányítás jellemzője, hogy a rendszer hibás állapotait historikus adatok és az aktuális mérési adatok kombinációjából előre tudja jelezni és elkerülni azokat. (Ezzel szemben a reaktív rendszerek a hiba bekövetkezése után kezelik a helyzetet.)
A proaktív viselkedéshez szükséges, hogy a historikus adatok megfelelő adatbányászati elemzésen essenek túl: egyszerű esetben klaszterezési technikákkal csoportosítani lehet a rendszer számára ártalmas és ártalmatlan jelenségeket. Egy-egy ilyen jelenséget könnyen felírhatunk komplexesemény-mintákkal és így lehetőség nyílik proaktív jellegű irányítási folyamatok kialakítására.

A témával bővebben a kapcsolódó munkák között, a 6.3. fejezet foglalkozik.

Az eszköz és a módszer továbbfejlesztéseként a historikus adatokon keresztül történő visszacsatolás, valamint a pontos adatelemzési módszerek kialakítása képzelhető el.

7.2. Analízis és integráció támogatása



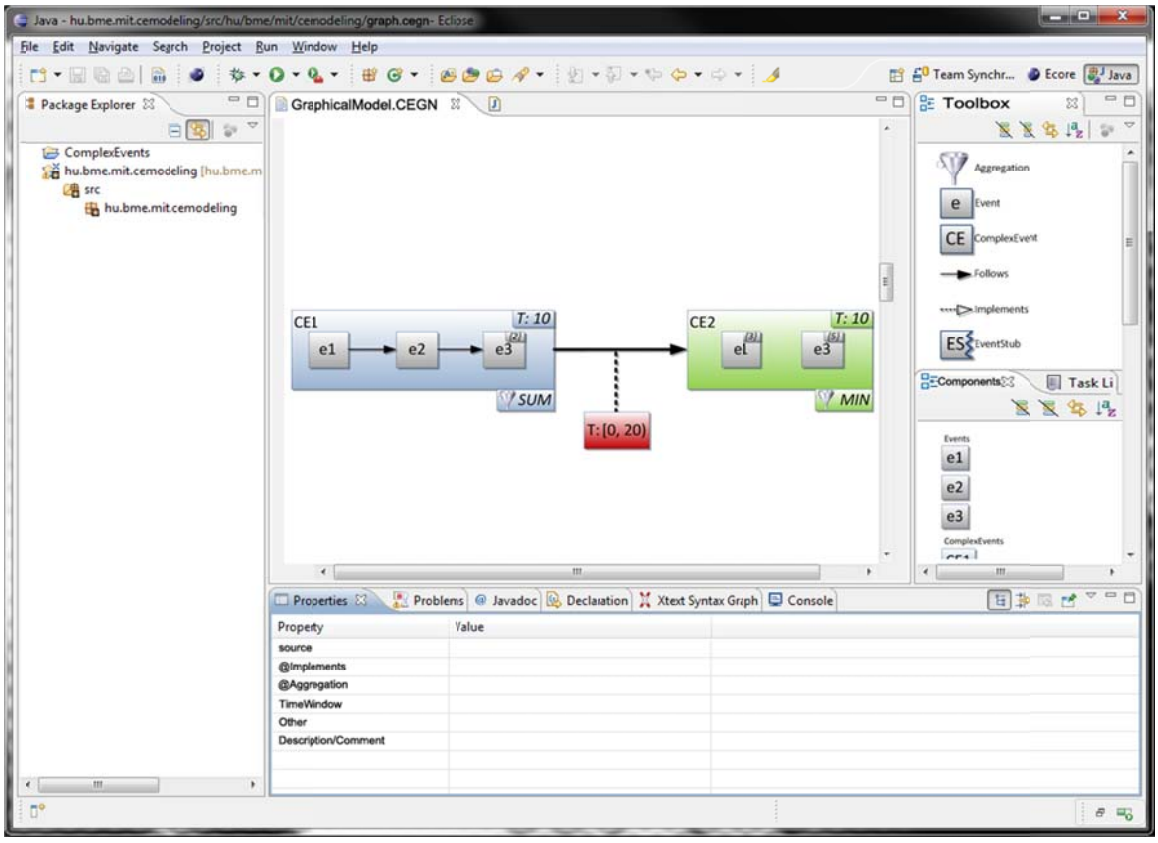
33. ábra – Analízis: fordított irányú modell-kód szinkronizáció.

Az automatikus kódgenerálás segítségével elértük, hogy az összeállított esemény-modellből szintaktikailag helyes és bizonyos szemantikai szempontok alapján is ellenőrzött forráskód álljon elő egy adott platformhoz.
Bizonyos esetekben hasznosak lehetnek a forráskódokból modelleket előállító automatizmusok, például az integrációs feladatoknál, vagy ha egy korábbi fejlesztési fázisból örökölt kódbázissal kell együttműködni.

Az automatizmus kialakítása viszonylag egyszerű, tekintve, hogy egy, az iparban nagyon elterjedt és hétköznapinak számító módszerről van szó: elegendő a fájlrendszer megfelelő mappáinak tartalmát szekvenciálisan végigolvasva és a CEDL metamodellje alapján a nyelv szintjére leképezni. Tekintve, hogy a forrásfájlok az adott platform szintaktikáját és szemantikáját követik, megfelelő dokumentáltság mellett ez egy jól kézben tartható feladat.

7.3. Grafikus felület

A dolgozat a 3. fejezetben rámutat, hogy a komplex események területén a *standard* grafikus modellező formalizmusok kevésbé hatékonyak, mint a szöveg alapúak. Azonban egy erre a célra kifejlesztett grafikus eszköz jó támogatást nyújthat a modellezéshez, jellemzően azokon a pontokon, ahol folyamat alapú megközelítést kell követni – mint például az egymást követő atomi események komplex eseménnyé történő összefogása esetén, időablakos definícióknál, vagy ok-okozati kapcsolatok megjelenítésekor.
A grafikus kiegészítés jelenleg tervezési fázisban van.



34. ábra – A grafikus szerkesztőfelület egy lehetséges megvalósítása. (Tervezet.)

7.4. Algoritmikus kereskedés modellalapú támogatása

A komplexesemény-feldolgozás felhasználási területeinek alapvető példája az algoritmikus kereskedés. [11] Az eszköz és módszer egyik továbbfejlesztési iránya lehet ennek a szakterületnek a támogatása. Az automatizálásra kifejlesztett automatizáló algoritmusok elemzésével várhatóan egy modellalapú módszer fejleszthető ki a kereskedés automatizálásnak támogatásához, ami intelligens, ontológia alapú visszacsatolással ellátva az algoritmusok hatásfokának, pontosságának javulásához vezethet.

8. Értékelés

Dolgozatom célja az volt, hogy megalkossak egy modellalapú módszert a komplex események feldolgozásához, ami a megcélzott szakterület követelményei alapján bizonyos mértékig automatizálható, valamint hogy ezt a módszert a megfelelő eszközökkel támogassam. Az eredmények önmagukban is relevánsak, de értéküket tovább növeli, hogy egy nagy ívű, jelenleg is sokak által aktívan kutatott témában képes kijelölni a további tudományos munka lehetséges irányait és jó alapot nyújtani azokhoz.

8.1. Eredmények

Koncepcionális eredmények

A dolgozat során elért koncepcionális eredmények a következők.

- Elkészítettem egy szakterülettől független eseménymodellt, amely a szakterületről származó speciális információkkal bővíthető ki, így erőteljes megoldást biztosít a komplex események modellezéséhez. (3. fejezet)
- Az eseménymodellre építve elkészítettem egy modellezőnyelvet, a CEDL-t (*Complex Event Description Language*), amely segítségével közvetlenül leírhatók a komplexesemény-minták. A nyelvhez tervezési mintákat is definiáltam a megfelelő használat megkönnyítése céljából. (3. fejezet)
- Módszert adtam a szakterület-specifikus információk hatékony kezelésére, ontológiák segítségével. Rámutattam az szemantikus tudásbázis célszerű szétválasztására, ami által könnyedén becsatolhatóvá válik bármilyen szakterület leíró ontológiája. (4. fejezet)
- Módszert adtam a megcélzott szakterület magas szintű ellenőrizhetőségének biztosítására, amivel akár olyan komplex követelményeknek való megfelelőség is kimutatható, mint a szakterület szabványai, jogszabályai. (4. fejezet)
- Összehasonlítottam a nyelv leíróerejét az alapul vett gyakorlati megvalósításhoz képest (Esper). Példán bemutattam, hogy az általam definiált nyelv mind az eseményminták leírásához szükséges kódmennyiség, mind utasításmennyiség tekintetében hatékonyabb. (3. fejezet)
- Bemutattam a komplexesemény-feldolgozó rendszerek gyakorlati alkalmazhatóságát, magával a tesztkörnyezet megvalósításával, valamint annak továbbfejlesztési és integrációs irányaival. (5. fejezet)

Implementációs eredmények

A dolgozat során kialakított módszert a megfelelő eszköztámogatással is elláttam, alapvetően az Eclipse platform nyújtotta eszközökre építve.

- Elkészítettem egy, a komplex események modellezésére alkalmas metamodellt az Eclipse Modeling Framework segítségével.
- Erre építve implementáltam egy teljes értékű fejlesztőkörnyezetet, ami a modellezés munkafolyamatát segíti olyan funkciókkal, mint az automatikus kódkiegészítés, vagy a szintaktikai validáció. Az eszköz Eclipse-plugin formájában érhető el, amit Java1.6SE-t futtatni képes környezetben az *Eclipse Indigo* verziójára lehet telepíteni. [61]

- A fejlesztőkörnyezethez elkészítettem egy automatikus kódgeneráló logikát, ami a modellek szerkesztésével párhuzamosan állít elő az Esper eseményfeldolgozó platformon telepíthető forráskódot: a mintafelismerést, a szűrést és a feldolgozást megvalósító osztályokat, valamint konfigurációs és interfészleíró segédállományokat.
- Az módszer és az eszközök tesztelésére egy jellegzetes nagyvállalati rendszert emuláló tesztkörnyezetet hoztam létre az iparban elterjedt, valamint saját fejlesztésű komponensek felhasználásával. Az eredményeket sikeresen teszteltem.

8.2. Az eredmények felhasználási területei és módjai

Az eredmények felhasználása tulajdonképpen minden olyan területen lehetséges, ahol komplexesemény-feldolgozó rendszereket alkalmaznak gyakorlati problémák megoldására. A modellalapú módszer használatának előnye jellemzően azokban az esetekben mutatkozik meg, amikor nagyméretű, nagyszámú, vagy nagy komplexitású eseménymintákat kell felhasználni a követelmények lefedéséhez. Ezek közül azokat a szakterületeket említem meg, amelyeken a megkezdett munka folytatása valószínűnek tűnik a közeljövőben.

Kritikus IT-rendszerek monitorozása

Az informatikai rendszerek monitorozása leginkább az üzemeltetési és biztonsági szabványokkal való konformitás biztosítása miatt fontos terület. Az ontológiaalapú megközelítés (4. fejezet), valamint az analízis irányú modell-kód szinkronizáció alkalmas támogatásával lehetőség nyílik formálisan is kimutatni a vonatkozó szabványoknak, vagy jogszabályoknak való megfelelőséget.

Elosztott piaci elemzések

A modern piac alapvetően a tőzsdével azonosítható. Az algoritmikus kereskedést támogató eszközök előre definiált algoritmusok alapján tesznek ajánlatokat vásárlásra, vagy adnak el. Vannak azonban a tőzsdén kívüli események, amelyek hatása a tőzsdén is megjelenik, hiszen a kereskedés nagyrészt hangulatfüggő, főleg a spekuláns ügyletek esetében. Ezek a külső események jellemzően kis késleltetéssel jelennek meg a tőzsdén, jóval gyorsabban és nagyobb hatással, mint például egy ország GDP-jének görbéjében.

Az ontológia alapú megközelítés lehetővé teszi, hogy bizonyos külső információs csatornákat csoportosítva, minősítve csatoljunk be a tőzsdei elemző eszközökhöz, ezáltal olyan piaci elemzések is lehetővé válnak, amelyek valódi prediktív erővel bírnak: már azelőtt prognosztizálni lehet egy-egy tőzsdei eseményt, mielőtt annak nyomai a tőzsdei adatokban megjelennének.

Irodalomjegyzék

[1] Supreet Oberoi, Introduction to Complex Event Processing & Data Streams, 2007, soa.sys-con.com.

[2] Michael Eckert and François Bry, "Complex Event Processing," *Informatik Spektrum, Springer*, vol. 32, no. 2, pp. 163-167, április 2009.

[3] IBM Official Website. Complex Event Processing. [Online]. http://domino.watson.ibm.com/comm/research.nsf/pages/r.datamgmt.innovation.cep.html

[4] Brenda M. Michelson. (2006) Event-Driven Architecture Overview. http://www.omg.org/soa/Uploaded%20Docs/EDA/bda2-2-06cc.pdf.

[5] Alejandro Buchmann and Boris Koldehofe, "Complex Event Processing," *it - Information Technology: Vol. 51*, no. 5, pp. 241-242, 2009.

[6] David B. Robins, Complex Event Processing, 2010.

[7] The event processing manifesto by the participants of the 2010 Dagstuhl seminar on event processing, 2010.

[8] Jamie Miyazaki. (2005) How Japan handles tsunami threat. http://news.bbc.co.uk/2/hi/asia-pacific/4149009.stm. [Online]. http://news.bbc.co.uk/2/hi/asia-pacific/4149009.stm

[9] Google Inc. Google Flu Trends. [Online]. http://www.google.org/flutrends/

[10] Foley J. and Churcher G.E., "Applying Complex Event Processing and Extending Sensor Web Enablement to a Health Care Sensor Network Architecture," in *Communications Symposium*, London, 2009.

[11] Melanie Rodier. (2007, november) Wall Street & Technology. [Online]. http://www.wallstreetandtech.com/technology-risk-management/204203344

[12] Microsoft Developer Network Web Site. [Online]. http://msdn.microsoft.com/en-us/library/ee229547%28v=SQL.10%29.aspx

[13] IBM Corporation, IBM Streams Processing Language Introductory Tutorial, 2011.

[14] Eclipse Foundation, Inc, Xtext Reference Documentation, 2011.

[15] Eclipse Foundation, Inc, Xtext Documentation (Detailed), 2011.

[16] Eclipse Foundation, Inc., Xtext User Guide v0.72.

[17] EsperTech Inc., ESPER Reference Documentation 4.3.0, 2011.

[18] Clebert Suconic, Andy Taylor, Tim Fox, Jeff Mesnil, and Howard Gao, HornetQ QuickStart Guide - Putting the buzz in messaging, 2010.

[19] Clebert Suconic, Andy Taylor, Tim Fox, Jeff Mesnil, and Howard Gao, HornetQ User Manual - Putting the buzz in messaging, 2010.

[20] Hewlett-Packard Company. HP Universal CMDB Software. [Online]. http://www8.hp.com/us/en/software/software-product.html?compURI=tcm:245-936985

[21] Giorgia Lodi et al., CoMiFin Deliverable D4.2 Components of the secure middleware (final), 2010.

[22] Gennady Laventman et al., CoMiFin Deliverable D3.3 Final Framework Architecture, 2010.

[23] StreamBase Web site. [Online]. http://www.streambase.com/algorithmic-trading.htm

[24] EsperTech Inc. Official Esper Website. [Online]. http://www.espertech.com/

[25] FZI Forschungszentrum Informatik, intelligent Complex Event Processing (iCEP) at FZI, http://icep.fzi.de; Utoljára megtekintve: 2011. október 4.

[26] Hamza Ghani et al., "Assessing the Security of Internet Connected Critical Infrastructures," *Security and Communication Networks Special Issue: Internet of things*, pp. 1-12, 2011.

[27] Distributed Management Task Force, Inc. (DMTF). (2004, december) Distributed Management Task Force, Inc. Official Web site. [Online]. http://www.dmtf.org/sites/default/files/cim/cim_schema_v29/CIM_Core.pdf

[28] Lajos Róbert Erdélyi, Eseményfeldolgozás modellalapú támogatása, 2010.

[29] János Balázs Ribli, Informatikai infrastruktúra konfigurációjának szabályalapú ellenőrzése, 2010.

[30] Adrian Paschke, "Design Patterns for Complex Event Processing," , 2008.

[31] Adrian Paschke, Paul Vincent, Catherine Moxey, and Alex Alves, Event Processing Reference Architecture and Design Patterns, 2011, The 5th ACM International Conference on Distributed Event-Based Systems (DEBS).

[32] Balázs Simon, Metamodellek a szoftverfejlesztésben (VIIIM228) : Szintaktikai elemzés: XText, Laboratóriumi segédlet.

[33] Johannes Kepler University of Linz and Vienna University of Technology. EMF Profiles. [Online]. http://www.modelversioning.org/emf-profiles

[34] Wayne Diu. (2009, február) Custom Domain Modeling with UML Profiles: The Basics of Generating Tooling for Elements from a UML Profile. https://www-304.ibm.com/support/docview.wss?uid=swg27015163&aid=1.

[35] Oracle Corporation. (2007) VMWare ESX Server Metrics (Oracle Enterprise Manager System Monitoring Plug-in Metric Reference Manual for Virtualization Management). http://download.oracle.com/docs/cd/E11857_01/em.111/e10709/vmware.htm.

[36] ISACA Organization, CobiT 4.1 szabványspecifikáció - Magyar változat, 2010.

[37] Securities and Exchange Commission, Securities Act of 1933, as amended through P.L. 111-229, approved August 11, 2010.

[38] Budapesti Értéktőzsde Részvénytársaság, A Budapesti Értéktőzsde zártkörűen működő részvénytársaság alapszabálya a változásokkal egységes szerkezetben, április 29., 2011.

[39] Budapesti Értéktőzsde Részvénytársaság, A Budapesti Értéktőzsde zártkörűen működő részvénytársaság működési kockázatkezelési szabályzata, 2010.

[40] Benedek Izsó, Szakterület specifikus mérnöki modellek ontológia alapú ellenőrzése, 2009, BME Tudományos Diákköri Konferencia.

[41] Diana Kalibatiene, Olegas Vasilecas, and Giancarlo Guizzardi, Transforming Ontology Axioms to Information Processing Rules – An MDA Based Approach, 2010.

[42] Paul Cichonski and Booz Allen Hamilton. (2010) Semantic Interoperability in IT Security: Ontology for IT Product Representation. http://scap.nist.gov/events/2010/cichonski_paul-semtech2010_0620-final.pdf.

[43] Loraine Lawson. (2011, augusztus) Semantic Integration Is Emerging Field - Should You Care? http://www.itbusinessedge.com/cm/blogs/lawson/semantic-integration-is-emerging-field---should-you-care/?cs=16192. [Online].

http://www.itbusinessedge.com/cm/blogs/lawson/semantic-integration-is-emerging-field---should-you-care/?cs=16192

[44] András Pataricza, László Gönczy, András Kövi, and Zoltán Szatmári, A methodology for standards-driven metamodel fusion, 2011.

[45] OMG Org. (2008, január) Documents associated with Semantics of Business Vocabulary and Business Rules (SBVR), Version 1.0. http://www.omg.org/spec/SBVR/1.0/.

[46] Attila Kirner, Az üzletmenet-folytonossággal kapcsolatos jogszabályi elvárások és IT felügyeleti tapasztalatok a pénzügyi szervezeteknél, 2007.

[47] Pénzügyi szervezetek állami felügyelete, A Pénzügyi Szervezetek Állami Felügyeletének 1/2007. számú módszertani útmutatója a pénzügyi szervezetek informatikai rendszerének védelméről, 2007.

[48] Ábel Hegedüs, Rendszerintegráció és -felügyelet laboratórium (VIMIM309): Kommunikáció JMS és JMX technológia segítségével, 2011, Laboratóriumi segédlet.

[49] Michael Dr.-Ing. Eichberg, "The Interceptor Architectural Pattern," in *Pattern-oriented Software Architecture: Patterns for Concurrent and Networked Objects, Volume 2*.: Wiley, 2000, http://www.st.informatik.tu-darmstadt.de:8080/ctfda/downloads/files/9%20-%20Interceptor%20Pattern.pdf.

[50] Gero Decker, Alexander Grosskopf, and Alistair Barros, A Graphical Notation for Modeling Complex Events in Business Processes, 2007.

[51] OASIS - Organization for the Advancement of Structured Information Standards. (2007, március) WS-Trust 1.3 OASIS Standard. http://docs.oasis-open.org/ws-sx/ws-trust/200512/ws-trust-1.3-os.html.

[52] Roberto Baldoni, CoMiFin Scientific View, 2011.

[53] Giorgia Lodi et al., "Trust Management in Monitoring Financial Critical Information Infrastructures," *Mobile Lightweight Wireless Systems*, vol. 45, no. 7, pp. 427-439, 2010.

[54] András Pataricza, Proactivity = Observation + Analysis + Knowledge extraction + Action planning?, 2011.

[55] Zhengzheng Xing, Guozhu Dong, Jian Pei, and Philip S. Yu, Mining Sequence Classi¯ers for Early Prediction, 2008.

[56] Krisztián Antal Buza, Fusion Methods for Time-Series Classification, 2011.

[57] Jian Pei, Early Classification: Problems, Preliminary Results, and Opportunities, 2011.

[58] IBM Corporation. Bringing big data to the Enterprise. [Online]. http://www-01.ibm.com/software/data/bigdata/

[59] IBM Corporation. InfoSphere Streams. [Online]. http://www-01.ibm.com/software/data/infosphere/streams/

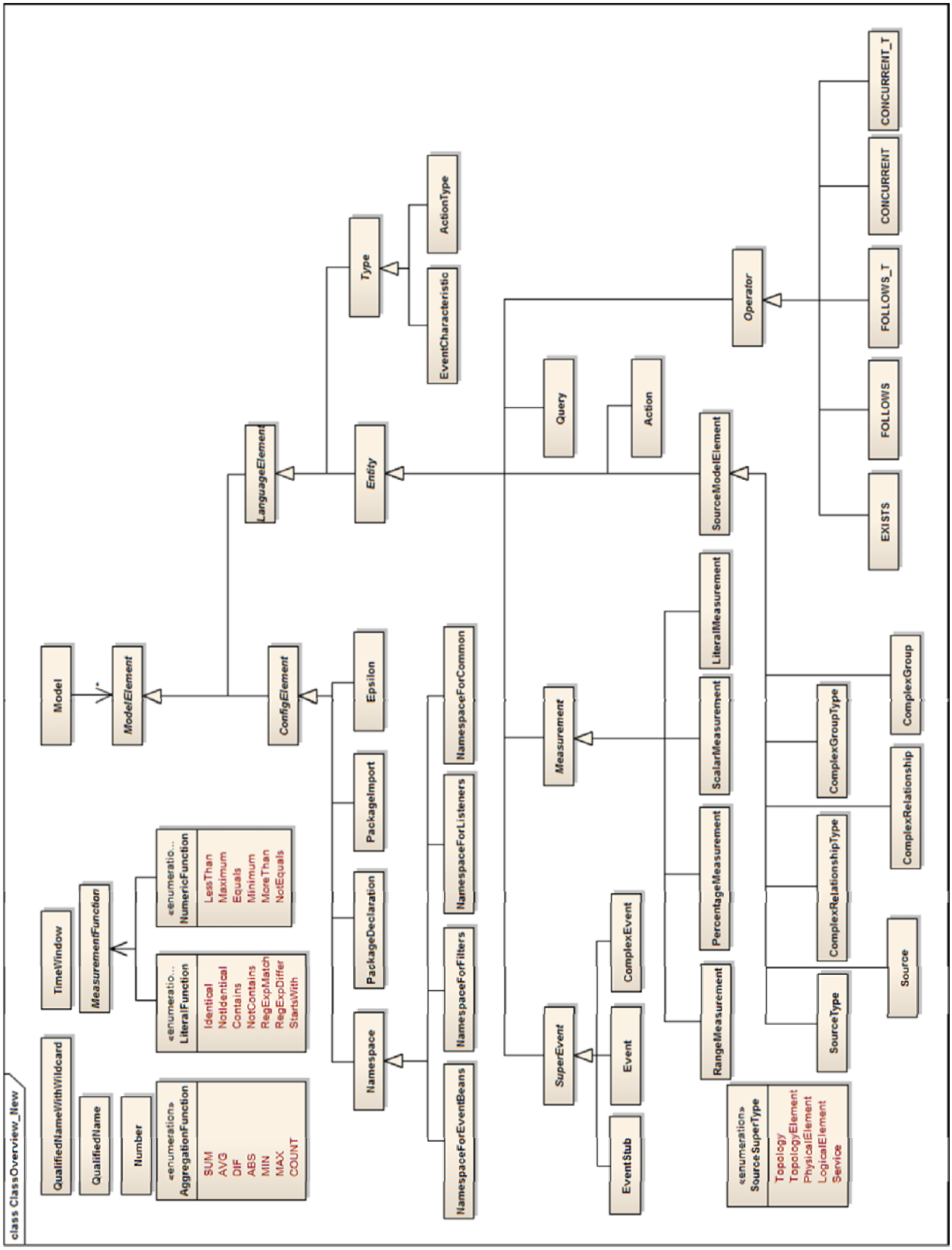
[60] Chuck Ballard et al., *IBM InfoSphere Streams: Assembling Continuous Insight in the Information Revolution*, első ed.: IBM, 2011.

[61] Eclipse Foundation, Inc. Eclipse Indigo Official Website. [Online]. http://www.eclipse.org/indigo/

[62] Magyar Tudományos Akadémia, *A magyar helyesírás szabályai*, tizenegyedik kiadás ed. Budapest, Magyarország: Akadémiai Kiadó, 1985.

Függelék

I. Teljes CEDL metamodell



35. ábra – Teljes CEDL nyelvi metamodell.

II. CEDL Nyelvi kivonat

import
A Java nyelvhez hasonlóan más névtereket tesz közvetlenül használhatóvá az adott névtérben.

NamespaceForEventBeans (-Filters, -Listeners, -Common)
A kódgenerálás kimeneti útvonalait lehet konfigurálni ezekkel a beállításokkal a különböző kimeneti típusokra.

epsilon
A célrendszer minőségi paramétereiből származtatott időablak, amin belül két esemény abszolút egyidejűnek tekintünk feldolgozás közben.

Action
Jobbkéz-szabály. A bal oldalon felismert minta után végrehajtott szekvencia.

ActionType
Az egyszerűbb használathoz az akciókat akciótípusba csoportosítva használhatjuk.

LiteralMeasurement, ScalarMeasurement, PercentageMeasurement, RangeMeasurement
A nyelvben definiált mértékek. Az események attribútumaira ezek segítségével illeszthetünk mintákat.

EXISTS
A paraméterül kapott események létezését írja elő.

.TIMEWIN(t)
Az EXISTS operátorhoz kapcsolható módosító, ami az EXISTS eseményeinek létezését a *t* időintervallumon belül követeli meg.

FOLLOWS
A paraméterlistában átadott események, az ott definiált sorrendben követik egymást, esetleg számossággal kiegészítve. Az EXISTS operátort a sorrendezés relációjával egészíti ki.

FOLLOWS_T
Ugyanaz, mint a FOLLOWS, de követési szabályoknak egy *t* időintervallumon belül kell lenniük.

CONCURRENT
A paraméterül kapott események egyidejűleg jelennek meg. (Nem definiál explicit időablakot, ezért az implicit Epsilon paraméter a mérvadó.)

CONCURRENT_T
Ugyanaz, mint a CONCURRENT, de az események egyidejűsége egy *t* időintervallumon belül *bármikor* bekövetkezhet.

Event
Esemény definiálására szolgáló kulcsszó.

ComplexEvent
Komplex esemény definiálására szolgáló kulcsszó.

EventStub
Eseménycsonk definiálására szolgáló kulcsszó.

characteristic
Az esemény karakterisztikáját állíthatjuk be vele. Nem kötelező paraméter, használata validációs szempontokból hasznos nagyobb modelleknél. (Például meg lehet nézni, létezik-e minden kritikus szerverhez „Szolgáltatásbiztonság” karakterisztikájú eseményminta.

implements
Az esemény, ha csonkot valósít meg, ezzel a kulcsszóval adhatjuk meg a kapcsolatot.

@Override
Az Eseménycsonkot implementáló Eseményben ezzel a kulcsszóval definiálhatjuk felül a csonkban definiált attribútumokat. (Csak azokat, amelyeket nem kell implementálni, tehát érték is van kötve hozzájuk.)

@ImplementMeasurement
Az Eseménycsonkot implementáló Eseményben ezzel a kulcsszóval implementálhatjuk a csonkban csak létrehozott, de értékkel nem kötött attribútumokat. (Ezeket kötelező implementálni.)

@Aggregation
Ezzel a kulcsszóval egy komplex eseményen belül adhatunk meg aggergációs szabályokat az események bizonyos attribútumaira.

source
Eseményforrás definiálására szolgáló kulcsszó.

sourceType
Eseményforrás típusának definiálására szolgáló kulcsszó.

ComplexRelationship
Két eseményforrás közötti irányított, 1-1 kapcsolat. Például: tartalék erőforrás.

ComplexGroup
Több eseményforrás közötti irányítatlan, *-* kapcsolat. Például: szerverfürt.