Molnár Géza Gábor

Számítástudományi tanszék

Kő Andrea, PhD

Budapesti Corvinus Egyetem

Közgazdasági és Gazdaságinformatikai Doktori Iskola

Szemantikus web technológiák alkalmazási lehetőségei az „exploratory” OLAP-ban

Doktori értekezés

Molnár Géza Gábor

Budapest, 2022

Tartalomjegyzék

Ábrajegyzék

Táblázatok jegyzéke

1 BEVEZETÉS

Az adatbázis rendszerek, mint információrendszerek, elsődleges célja az adatok információvá alakítása a döntéshozatalhoz felhasználható tudás érdekében. Az adatbázis rendszereknek eszközöket kell biztosítaniuk az adatok összesítéséhez és elemzéséhez, valamint az adatokat megfelelő formában (lekérdezések, riportok, kimutatások) kell előállítaniuk (Bourgeois, 2019) .

Amennyiben adataink megfelelően strukturált formában vannak jelen, akkor ennek a célnak a teljesítése általában nem okoz problémát, ugyanis számtalan jól bevált eszköz létezik a strukturált adatok feldolgozására. Ilyen például az SQL nyelv, amely a relációs adatbázisok „de facto" szabványos lekérdező nyelve. Ez viszonylag könnyen elsajátítható (angol nyelvhez hasonló) szintaktikát használ, és a lekérdezések létrehozása mellett lehetővé teszi az adatbázisok adminisztrálását is (Batra, 2018).

A kérdés az, hogy mi a helyzet a strukturálatlan adatokkal (pl. szövegek, képek, videók stb.). Ezek részesedése az összes adatmennyiségből korábban nem volt jelentős.

Az utóbbi évtizedben ez az arány jelentősen megváltozott, nem kis mértékben a közösségi médiának és az okos eszközöknek köszönhetően. A strukturálatlan adatok mennyisége sokkal nagyobb mértékben nőtt, mint a strukturált adatoké. Így fordulhat elő, hogy manapság már jóval több a nem strukturált adat a strukturáltnál: 2017-ben a részesedésük 80%, illetve 20% körül volt (Phoebe Wong, 2019), és várhatóan ez az arány egyre jobban eltolódik a strukturálatlan adatok felé. Az IDG (vezető cég a média-, adat- és marketing szolgáltatások terén) szerint 2022-re ez az arány 93%, illetve 7% körül lesz (Anonymous, 2020).

A nagy mennyiségű adat tárolása egyelőre nem okoz gondot, mivel a tároló eszközök ára az utóbbi időben jelentősen csökkent, kapacitásuk viszont közben növekedett. Egy GB adat tárolási költsége így a töredékére csökkent. A jelenleg alkalmazott tárolási technológiák ugyan néhány év múlva várhatóan elérik teljesítőképességük határát, de már most megjelentek olyan jövőbe mutató fejlesztések (pl. kvantum tárolók), amelyek képesek lesznek lépést tartani a megnövekedett tárolási kapacitás igényekkel (Klein, 2017).

A strukturálatlan adatok feldolgozásával viszont más a helyzet. Ezeket korábban manuálisan dolgozták fel, vagy egyszerűen figyelmen kívül hagyták. A növekedő részarányuknak köszönhetően viszont egyre többször merül fel ezek elemzésének igénye. Erre tipikus példa, amikor egy cég szeretné az ügyfeleinek visszajelzéseit vagy egy partneréről megjelenő szöveges információkat összesíteni, kiértékelni. A nagy adatmennyiség miatt sokszor reménytelen, hogy hasonló feladatokat manuálisan végezzünk el. Ezért többnyire az a törekvés, hogy olyan számítógépes eljárásokat dolgozzunk ki, amely lehetővé teszi a strukturálatlan – többnyire szöveges - adatok feldolgozását és elemzését.

Ezeknél az eljárásoknál több kihívással is szembe kell nézni. Ezek közül a legnagyobb annak eldöntése, hogy egy eljárás alkalmazása során kapott eredmény mennyire releváns. Az adatelemző modellek ugyanis nem tesznek különbséget a feltárt összefüggések között. Ha például egy elemzés során két változó között a modell alapján szoros kapcsolat adódik, az a gyakorlatban nem mindig jelent használható információt. A kapott eredmények megbízhatósága és pontossága sok esetben megkérdőjelezhető.

Egy másik kihívás az adatok minőségének kérdése. A strukturálatlan adatokat nehéz ellenőrizni, nem mindig pontosak. A minőség eloszlása, az adatok használhatósága sokszor ingadozó, egyenetlen (Dixon, 2019). További kérdés, hogy a kapott eredményeket hogyan lehet beilleszteni a strukturált adatok elemzése során kapott eredmények közé. Erre azért van szükség, mert ellenkező esetben az eltérő típusú adatok (strukturált és nem strukturált) miatt minden egyes döntés során két külön elemzést kellene figyelembe venni.

A strukturálatlan adatok feldolgozása, mint probléma sok részterületet érint, többek között a tudásreprezentáció, a szemantikus technológiák, az adat-, web-, és szövegbányászatot, a mesterséges intelligencia részterületeit ezen belül a tanuló algoritmusok témáját is. A struktúra alkotás fontos eszközei a szemantikus technológiák, ezek közül az ontológiák, amelyek egy adott terület fogalmi leírását adják meg (Gruber, 1993). Segítségükkel egy adott területre jellemző fogalmak és a közöttük lévő kapcsolatok definiálhatók.

Kutatási témám a szemantikus technológiák adattárházakban való felhasználásával kapcsolatos. Elsődleges célom olyan megoldások vizsgálata, amelyek lehetővé teszik a strukturált és nem strukturált adatok együttes elemzését az adattárház/üzleti intelligencia rendszerekben. Az ilyen rendszereket a szakirodalomban exploratory OLAP rendszereknek is nevezik. Az exploratory OLAP rendszerek területe meglehetősen új, a legelső modellek a szakirodalomban alig több, mint tíz éve jelentek meg. Fontos megjegyezni, hogy a legtöbb ilyen modell megmaradt koncepcionális szinten. Néhány esetben készült ugyan prototípus, viszont azok általános alkalmazhatósága még nem igazolt. Emiatt céljaim közé tartozik még legalább egy prototípus elkészítése, és értékelése. A működő prototípusok segítségével következtetéseket lehet levonni az adott exploratory OLAP modell megvalósíthatóságára és alkalmazhatóságára is. A prototípus elkészítésénél a szemantikus web technológiák közül elsősorban az ontológiákat szeretném használni. Mivel ezek az általam választott területen (ticketing rendszer) nem állnak rendelkezésre, ezért további cél a szükséges szakterületi ontológia kifejlesztése.

A disszertáció első része (második fejezet) egy szakirodalmi áttekintést tartalmaz. Ennek során kitérek az adattárházak fogalmi alapjaira, klasszikus modelljeire (Kimball és Inmon modellje) és az új megközelítésekre (pl. adattavak), ismertetem a jelenlegi kihívásokat. Kutatásom során a szöveges adatokban lévő információt szeretném OLAP struktúrájúvá alakítani, ezért a szakirodalom elemzésében foglalkozom a kapcsolódó részterületekkel is. Ismertetem az üzleti analitika fogalmi hátterét, az OLAP elemzések elméleti alapjait. Bemutatom az ontológiák elméleti alapjait, fejlesztési megközelítései. Végül a kapcsolódó, szakirodalomban közölt kutatási eredményeket elemzem, ismertetek több olyan rendszert, amely a szöveges adatokból OLAP struktúrát állít elő (Ibragimov, et al., 2015), (Abelló, 2015) és (Prasad, 2010).

A következő részben (harmadik fejezet) megfogalmazom a kutatást érintő kihívásokat, a kutatási kérdéseket, a kutatás hatókörét és ismertetem a kutatás fogalmi modelljét is. Kitérek a modell gyakorlati alkalmazási lehetőségére is egy lehetséges prototípus (ticketing rendszer) bemutatásával. Végül megemlítem az eredmények validálásával kapcsolatos problémákat is.

A negyedik fejezet a kutatási módszertant részletezi. Ide tartozik a design science, az adatgyűjtés és adatelemzés, az ontológia fejlesztés módszertanok, valamint a prototípus készítése során alkalmazandó fejlesztési módszertan bemutatása.

A disszertáció ötödik fejezetében az elért eredményeket mutatom be. Ennek során megvizsgálom két exploratory OLAP rendszer gyakorlati megvalósíthatóságát egy-egy prototípus elkészítésén keresztül. A prototípusok segítségével később fontos következtetéseket lehet levonni az adott rendszer használhatóságával, hatékonyságával kapcsolatban. Nem törekszem a teljességre annál is inkább, mivel egyre több ilyen rendszer kerül publikálásra. A hangsúlyt az ontológia-alapú megoldás(ok)ra helyezem, mivel ezekben látom a legtöbb lehetőséget.

Az utolsó fejezetben megvizsgálom, hogy az egyes kutatási kérdésekre – az eddigiek alapján – milyen válasz adható. Ugyanitt kitérek a kutatás korlátaira is, amelyek az exploratory OLAP rendszerrel kapcsolatosak.

2 SZAKIRODALMI ÁTTEKINTÉS

Ebben a fejezetben az exploratory OLAP rendszerek elméleti hátterét mutatom be. Először egy történeti áttekintés segítségével kitérek arra, hogy milyenek voltak a kezdeti riporting rendszerek. Utána arról lesz szó, hogy a változó riport igények, és a technikai fejlődés hogyan segítették elő az adattárházak kialakulását. Az adattárházaknál ismertetem a két legfontosabb architektúrát, majd azokat a kihívásokat, amelyek az adattárházakat napjainkban érintik. A fejezet végén az adattárházakra épülő OLAP rendszerekről lesz szó, ezen belül a kutatási témámhoz tartozó exploratory OLAP megoldásokat fogom bemutatni három konkrét fogalmi modell és rendszer leírásával. A három megoldás egymástól jelentősen eltér az alapelv, illetve a kivitelezés során alkalmazott technológia tekintetében. Az ide tartozó fogalmak (pl.: ontológia, csillag séma, multidimenzionális kifejezések stb.) is ismertetésre kerülnek.

Az 1960-as évekig a számítógépek többnyire elektronikus adatfeldolgozási feladatokat hajtottak végre (James A. O'Brien, 2010) (Sharda, et al., 2018). Ennek során különböző működési célú adatokat (pl.: pénzügyi adatok, gyártási adatok) rögzítettek és dolgoztak fel. Mivel az ilyen rendszerek általában adott üzleti folyamatokhoz kapcsolódó események, tranzakciók rögzítésével kapcsolatosak, ezért ezeket OLTP (Online Transaction Processing) rendszereknek is nevezik. Az OLTP rendszerek ma is léteznek, tipikusan ilyen rendszer például egy banki tranzakciókat vagy online vásárlásokat rögzítő és kezelő alkalmazás (Gábor, 1997).

Az OLTP rendszerek által rögzített és kezelt adatok mennyisége folyamatosan növekedett. Egyre nehezebbé vált a releváns információ kinyerése a tárolt adatokból. Erre elsősorban a vezetőknek volt szükségük a napi döntések meghozatalához. Ezek az igények vezettek a riporting rendszerek kialakulásához.

Az 1960-as években jelentek meg az első vezetői információs rendszerek (MIS – Management Information System) (James A. O'Brien, 2010). Ezek előre definiált riportok formájában támogatták a vezetői döntéseket. A riportok adott rendszerességgel (jellemzően hetente, havonta) futottak le, mindig azonos formában. A riportok jellemzően gyártási, eladási adatokat, költség trendeket jelenítettek meg. Ez a megoldás egyszerűen kivitelezhető volt, viszont meglehetősen rugalmatlan a riportok rögzített formája miatt.

Az 1970-es években a döntéstámogató rendszerek (DSS – Decision Support System) megjelenésével a vezetők számra készített riportok már jobban alkalmazkodtak az egyre növekvő igényekhez (Kő & Lovrics, 1997). Ezek már – korlátozott módon – képesek voltak interaktív és ad-hoc támogatást is biztosítani a vezetők és egyéb üzleti felhasználók döntési folyamataihoz tipikusan olyan területeken, mint a termék árazás, profit előrejelzés vagy a kockázat elemzés.

A felsővezetők döntéseit elősegítő felsővezetői információs rendszerek (EIS - Executive Information System) a következő évtizedben (1980-as évek) jelentek meg. Ezek testreszabott információkat biztosítottak a menedzsment számára az MIS és DSS rendszerekből és egyéb az igények szerint testreszabott formában. Jellemzően a kritikus, stratégiai döntéseket támogatták, például az üzleti teljesítmény elemzésével vagy a vetélytársak tevékenységeinek figyelésével (Kő & Lovrics, 2000) (Sharda, et al., 2018).

Közben a riport igények is folyamatosan változtak. A statikus, előre definiált és ütemezett riportok mellett a riportok új formái jelentek meg. Ilyenek a például a kivétel riportok, amelyek valamilyen nem várt eseménnyel kapcsolatosak (pl: egy ügyfél túllépi a hitelkeretét) vagy az un. push riportok, amikor az elérhető riportok közül kiválogatottak – megfelelő formára hozva – a vezetőhöz valamilyen módon, pl. hálózaton keresztül egyszerre eljutnak.

Az ad-hoc riportok létrehozását az un. riportgenerátorok is segítették, amelyek az átlagosnál kevesebb IT-ismerettel rendelkezők számára is lehetőséget adtak a riportok elkészítésére. Ezzel párhuzamosan természetesen megmaradt a riportkészítés hagyományos módja is, amely során a hozzáértő IT-szakember készíti el a riportot, és elérhetővé teszi a vezető számára.

A vezetői információs rendszerek a későbbiek során még sok változáson mentek keresztül. Alkalmazási területük egyre bővült, használhatóságuk folyamatosan javult. Ma már többnyire üzleti intelligencia rendszerek, illetve integrált vállalatirányítási rendszerek töltik be ezt a feladatot.

A kérdés az, hogy a vezetői információs rendszerek – vagy általánosabban a riporting rendszerek – megjelenése és fejlődése hogyan érintette a tranzakciós (OLTP) rendszereket?

A legelső riportok általában a működő OLTP rendszerek adatbázisaiból vették az adatokat. A riportok futtatása azonban – főleg az adatmennyiség gyors növekedésével - sok esetben jelentős többletterhelést okozott a rendszer számára. Ezen eleinte úgy segítettek, hogy a rendszer működési idején kívül – jellemzően éjszaka – futtatták le a riportokat.

A riportok állásidőben történő futtatása miatt keletkező időbeli késleltetés nem mindig volt elfogadható, ezért később gyakran másolatot készítettek az eredeti adatbázisokról, és a riportokat azokon futtatták. Ennek eredményeként a működési célú és az elemzési célú adatbázisok egyre inkább különváltak (Bánné Varga, 2012). Kezdetben a másolat felépítése megegyezett az eredeti adatbázissal. Később rájöttek, hogy érdemes a kimásolt adatokat a riport készítése előtt átalakítani abból a célból, hogy a riport elkészítése minél egyszerűbb legyen. A gond az volt, hogy a változó és eltérő riport igények miatt sokszor több másolat elkészítésére is szükség volt. Gyakran előfordult, hogy ugyanazokat az adatokat több másolat (kivonat, extrakt) is tartalmazta. Az extraktumok száma bizonyos esetekben olyan nagy volt, hogy elkészítési idejük nem fért bele abba az időbe, amíg az eredeti rendszer üzemidőn kívül volt („extraktum robbanás”, (Bánné Varga, 2012)).

Az adattárházak (Data Warehouses, DWH) az elemzési célú adatbázisokból fejlődtek ki. Létrejöttüket az a felismerés indukálta, az elemzés adatokat (megfelelő átalakítások után) célszerű egy helyen, egységes formában tárolni. Az adattárházakra épülő, elemzési célú rendszereket OLAP (Online Analitical Processing) rendszereknek is nevezzük. Ezek lehetővé teszik a vezetők és az elemzők számára nagy mennyiségű adat elemzését interaktív módon, különböző elemzési szempontok szerint.

Az OLAP-rendszerek alapvető működési szabályait Edgar F. Codd definiálta 1993-ban (Codd, et al., 1993). Ezen 12 szabály (axióma) szerint egy jól működő OLAP-rendszer, mint információs rendszer a következő tulajdonságokkal rendelkezik:

- Multidimenzionális nézet. A felhasználók/elemzők fogalmi nézetének többdimenziósnak kell lennie. A profitot lehet nézni például régiónként, termékcsoportonként, vagy akár az időben is. Az ilyen adatmodellek könnyebben és intuitívabban manipulálhatók, mint az egydimenziós adatmodellek. Jó példa erre a szeletelés (egy konkrét dimenzióérték rögzítése) művelete, amelynek megvalósítása egydimenziós modell esetén jelentősen több időt és erőfeszítést igényel, mint többdimenziós esetben.
- Transzparencia. Annak a ténynek, hogy az OLAP része vagy nem része a felhasználó kezelőfelületének, átláthatónak kell lennie a felhasználó számára. Ha az OLAP rendszer kliens-szerver architektúrára épül, akkor erre a tényre is vonatkozik a transzparencia követelménye. Az OLAP-ot egy valódi nyílt rendszerként kell biztosítani, amely lehetővé teszi az elemző eszköz beágyazását más rendszerekbe anélkül, hogy azok működését befolyásolná.
- Hozzáférhetőség. Az OLAP-rendszernek csak azokhoz az adatokhoz kell hozzáférnie, amelyek valóban szükségesek az elemzésekhez – felesleges input nem fordulhat elő. Az eszköz (és nem a felhasználó) feladata az OLAP logikai sémájának fizikai adatforrásokra való leképezése.
- Konzisztens jelentéskészítési teljesítmény. Az OLAP-eszköz teljesítményének nem szabad jelentősen csökkennie a dimenziók számának növekedése esetén.
- Kliens-szerver architektúra. A legtöbb adat, amely online analitikai elemzést igényel nagy teljesítményű számítógépeken (szerverek) tárolódik, és személyi számítógépeken (kliensek) keresztül érhető el. Ezért az OLAP-rendszereknek képeseknek kell lenniük a kliens-szerver környezetben való működésre. A szervereknek elég intelligensnek kell lenniük a különböző logikai és fizikai vállalati adatbázis sémák leképezésére és egyesítésére, valamint lehetővé kell tenniük, hogy a felhasználók minimális erőfeszítéssel kapcsolódhassanak hozzájuk.
- Általános dimenzió fogalom. Minden dimenziónak egyenértékűnek kell lennie a struktúráját és a műveleti képességeit illetően. A dimenziók szimmetrikusak, azaz egy adott művelet bármely dimenzióhoz hozzárendelhető. Az alapvető adatszerkezetet, képleteket és riport formátumokat nem szabad csak egy konkrét dimenzió irányában megváltoztatni.

- Dinamikus ritka mátrix kezelés. Az adatok eloszlásának egyik mérőszáma a ritkaság (hiányzó cellák aránya az összeshez képest), amely értéke gyakran változhat. Az OLAP-eszköznek optimális ritka mátrix kezelést kell biztosítani, azaz alkalmazkodnia kell a változó ritkaság értékekhez, ellenkező esetben az eszköz működése lassú lehet. A hozzáférési sebességnek állandónak kell lennie változó méretű adathalmazok esetén is.
- Több konkurens felhasználó támogatása. A gyakorlatban sokszor előfordul, hogy ugyanazon a modellen több felhasználó/elemző is dolgozik, vagy közülük egyszerre többen ugyanazon adatokból különböző modelleket hoznak létre. Az OLAP-eszköznek egyidejű hozzáférést, integritást és biztonságot kell nyújtania.
- Korlátozás nélküli dimenzióműveletek. Az eszköznek képesnek kell lennie az adatok közötti kapcsolatokból következő számítások elvégzésére. Azok a számítások, amelyek nem következnek a kapcsolatokbók, valamilyen nyelven megfogalmazott képletek alapján végezhetők el. Egy ilyen nyelvnek lehetővé kell tennie a számítást és adatkezelést bármilyen dimenzióban, függetlenül az egyes cellák által tartalmazott közös attribútumok számától.
- Intuitív adatkezelés. Az analitikai modellben meghatározott dimenziók felhasználói/elemzői nézetének tartalmaznia kell minden információt, amely szükséges az adatmanipulációkhoz (pl. lefúrás – adatok megtekintése részletesebb bontásban). Az adatmanipulációt a cellákon történő közvetlen művelettel kell végrehajtani, nem lehet szükség sem menü, sem pedig több elem használatára a felhasználói felületen.
- Rugalmas jelentéskészítés. A jelentéseknek (riportoknak) képesnek kell lenniük az adatmodellből származó információk vizuális szemléltetésére bármely orientáció szerint. Ez azt jelenti, hogy a soroknak, oszlopoknak és az oldalfejléceknek meg kell tudniuk jeleníteni tetszőleges számú dimenziót (nullától akár az összesig). Emellett az így megjelenített dimenzióknak képesnek kell lenniük a tagok bármely részhalmazának tetszőleges sorrendben való megjelenítésére.

- Korlátlan dimenziószám és aggregációs szint. Az OLAP-eszköznek képesnek kell lennie legalább 15, de lehetőség szerint 20 dimenzió kezelésére egy adatmodellen belül. Ezenkívül ezeknek a dimenzióknak korlátlan számú felhasználó/elemző által meghatározott összesítési szintet kell lehetővé tenniük az adott konszolidációs útvonalon belül.

Codd (1993) axiómái iránymutatóak az OLAP-rendszer definíciójával és működésével kapcsolatban. Az OLAP és az adattárház fogalmak erősen összefonódnak, ugyanis az adattárház döntéstámogatási szerepe a gyakorlatban szinte minden esetben OLAP-elemzések segítségével valósul meg.

Az adattárházak létrejöttét nagyban segítette az információtechnológia folyamatos fejlődése. A tároló eszközök (memóriák) kapacitásának növekedésével, a processzorok teljesítményének javulásával nem okozott problémát a működési adatok lemásolása, átalakítása, tárolása és feldolgozása. A hálózati eszközök és szoftverek lehetővé tették, hogy az adatbázis rendszerek - a nagygépes környezet mellett - elérhetők legyenek a felhasználók saját számítógépein is. Már csak az volt a kérdés, hogy milyen módon lehet megtervezni és megvalósítani egy olyan hatékony adattárház rendszert, amely megfelelő alapul szolgálhat az elemzési igényeknek?

A két legismertebb klasszikus adattárház tervezési megközelítés Kimball, illetve Inmon nevéhez fűződik (Kimball & Ross, 2013) (Inmon, 2002). A mai napig sok adattárház fejlesztés esetén ezek a modellek jelentik a kiindulópontot. Természetesen a 90-es évek óta sok idő eltelt, így a mai kor adattárházainak olyan kihívásokkal is szembe kell nézniük, amelyek Kimball és Inmon idejében még nem léteztek. Ezekről bővebben a 2.4-es alfejezetben lesz szó.

2.1 Kimball adattárház modellje

Az adattárház fogalmának Kimball szerinti meghatározása jól tükrözi az előzőekben ismertetett folyamatot, amely során a korábbi vezetői információs rendszerek átalakulása vezetett az adattárházak keletkezéséhez:

“Az adattárház a tranzakciós adatok lekérdezési és elemzési célokból speciálisan strukturált másolata” (Bánné Varga, 2012).

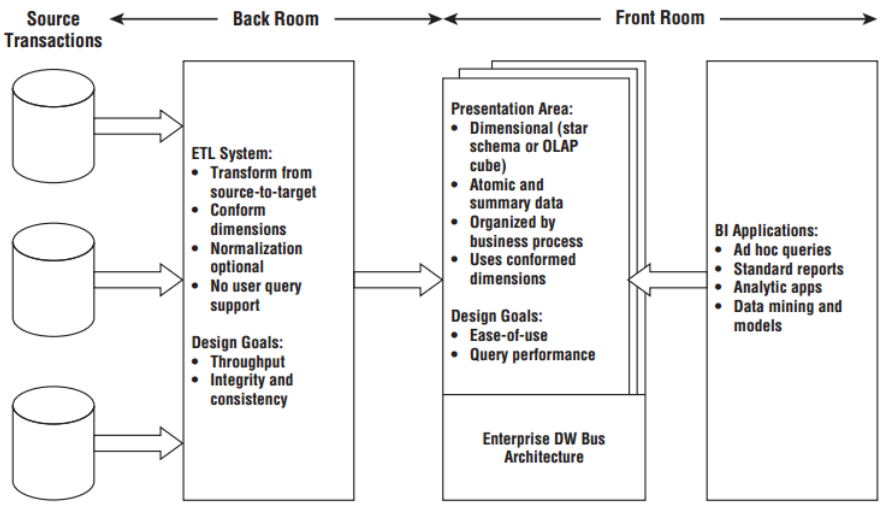
A modell alapján az adattárház rendszer négy különálló komponensből áll: a tranzakciós forrásrendszerek, ETL rendszer, adatmegjelenítési terület és az üzleti intelligencia (BI, Bussiness Intelligence) terület (Kimball & Ross, 2013).

- A tranzakciós forrásrendszerek tartalmazzák az üzleti tranzakciókat. Az esetek többségében ezek speciális alkalmazások, amelyek nem, vagy csak kis mértékben tartalmaznak történeti (historikus) adatokat. Az itt tárolt adatokat az adattárházból általában csak olvasni (lekérdezni) lehet, az adatok tartalma és formátuma rendszerint nem befolyásolható. Az adatok lekérdezése nem veszélyeztetheti a tranzakciós rendszerek működését, ezért a forrásrendszeri adatok lekérdezése csak tervezett módon és időben történhet. Ezért fontos, hogy a forrásrendszerek az adatok betöltésekor mindig elérhetők legyenek.
- Az ETL (Extract, Transform, Load) rendszer a tranzakciós rendszerek és a megjelenítési réteg között helyezkedik el. Felépítését tekintve egy munkaterületből, adatátviteli struktúrákból és folyamatok halmazából áll. Fontos elvi kérdés, hogy érdemes-e az adatokat normalizált formára hozni (ha nem ilyen formában érkeznek), mivel az megkönnyítené az adatok tisztítását és transzformációját. A megoldás hátránya, hogy így két ETL-folyamatra van szükség. Az első hozza létre a normalizált formát, a második pedig a később említett dimenzionális modellt. Kimball modellje nem javasolja, de nem is tiltja az adatok normalizálását.
    - Az ETL esetén az Extract jelenti az adatok kiolvasását a forrásrendszerekből és betöltését a rendszerbe további feldolgozás céljából. A betöltés után az adatok már az adattárházhoz tartoznak.
    - A Transform általában valamilyen adat manipulációt jelent, ami tipikusan lehet adattisztítás (elírások javítása, hiányzó elemek kezelése, egységes formátumra hozás stb.), a több forrásból származó adatok kombinációja, vagy az esetleges duplikációk megszűntetése. Az itt kezelt adatok az adattárház felhasználói számára még nem érhetők el.
    - Az ETL rendszer utolsó feladata az adatok áttöltése a megjelenítési területre megfelelő formában (Load). Ide a dimenzionális modellnek megfelelően – ténytáblák és dimenziótáblák formájában – töltődnek be az adatok. A dimenzionális modell létrehozása rendszerint sok átalakítást igényel. A dimenziótáblák általában kisebb méretűek, viszont az általában szükséges denormalizálás és a rekordokhoz megfelelő adattárházbeli kulcsok hozzárendelése nehezíti a feladatot. A ténytáblák esetén a folyamat kevésbé bonyolult, ellenben itt rendszerint nagy mennyiségű adatot kell feldolgozni, ez pedig

időigényes lehet. A Load alrendszer helyes működése kritikus az egész adattárház működése szempontjából, mivel az itt betöltött adatok már a felhasználók és a BI alkalmazások számára is láthatók lehetnek.

- A megjelenítési réteg szervezi és tárolja az adatokat olyan formában, amely lehetővé teszi az adatok hatékony lekérdezését a felhasználók, a riport készítők és a BI alkalmazások számára. Ez a dimenzionális tárolási forma relációs csillag séma vagy OLAP-kocka lehet. A megjelenítési réteg adatai lehetnek elemi (atomi) adatok és összegzett (aggregált) adatok is. Fontos, hogy az egyes dimenziók lehetőség szerint ne egyediek, hanem több ténytábla esetén is felhasználhatók legyenek. Eszerint az adatstruktúra létrehozásakor az adatokat nem az egyes osztályok (pl. marketing) igényei szerint kell strukturálni, hanem olyan dimenziókat kell alkalmazni, amelyek minden OLAP-kockában vagy csillag sémában ugyanolyan jelentéssel és szerkezettel rendelkeznek.
- Az üzleti intelligencia terület lehet akár egy lekérdező eszköz vagy egy komplex adatbányászati, adatmodellezési alkalmazás is. A lényeg mindegyik esetben a megjelenítési terület adatainak a döntéstámogatás során való felhasználásán van. A gyakorlatban a BI felhasználók többsége előre elkészített és megfelelően paraméterezett alkalmazásokkal és sablonokkal fér hozzá az adatokhoz, mivel a lekérdezések közvetlen létrehozása az átlagosnál több informatikai kompetenciát igényel.

Kimball adattárház/BI modelljének felépítését a következő ábra szemlélteti:



1. Ábra: Kimball DWH/BI architektúra (Kimball & Ross, 2013)

2.2 Inmon adattárház modellje

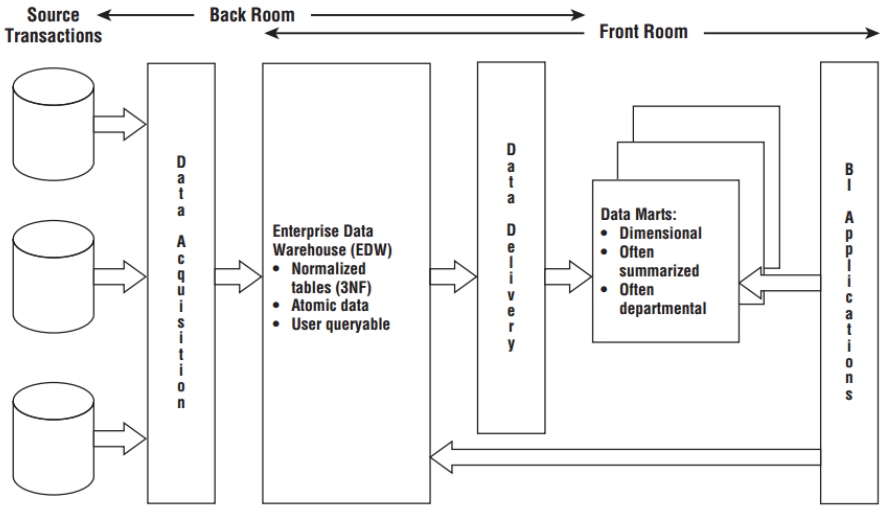
Inmon-t gyakran az adattárházak atyjaként is emlegetik. Ő a következőképpen definiálta az adattárház fogalmát:

"Az adattárház témakör-orientált, integrált, időfüggő, de időben nem változó adatok gyűjteménye, amelyet a cég vezetői döntéshozatalának támogatására használnak (Inmon, 2005)."

Inmon adattárház modelljében megtalálhatók a Kimball-modell bizonyos elemei (forrásrendszerek, ETL-rendszer, megjelenítési terület, üzleti intelligencia terület), de a koncepció és a megvalósítás attól teljesen eltérő (Kimball & Ross, 2013).

- Az architektúra középpontjában a vállalati adattárház (EDW, Enterprise Data Warehouse) található, amely egy normalizált, elemi adatokat tartalmazó adatbázis (repository). A normalizálás – Kimball modelljével ellentétben – itt kötelező.
- Az alábbi ábrán látható data acquisition jelenti az első ETL rendszert, amely kinyeri az adatokat a forrásrendszerekből, elhelyezi őket egy átmeneti területen (staging area), majd továbbítja őket a vállalati adattárház számára.
- A második ETL rendszer (Data Delivery) hozza létre a riportok és elemzések számára szükséges adatokat. Erre azért van szükség, mert a normalizált struktúrákat dimenzionális modellekké kell alakítani. Az átalakítás sok adat esetén idő- és erőforrás igényes lehet.
- A megjelenítési terület általában részleg-specifikus és dimenzionálisan strukturált, nem elemi (összegzett) adatokkal. Az itt alkalmazott adatstruktúrákat adatpiacoknak is nevezzük.
- Az üzleti intelligencia terület itt is a felhasználók által használt alkalmazásokat, lekérdező- és elemző eszközöket jelenti. Az ábráról leolvasható, hogy a BI felhasználók nemcsak a megjelenítési terület adataihoz férnek hozzá, hanem – többnyire részben – a vállalati adattárház adataihoz is.

Inmon adattárház modellje a következő ábrán látható.



2. Ábra: Inmon architektúra (Kimball & Ross, 2013) (Inmon, 2005)

2.3 Kimball vs. Inmon

Az adattárházak tervezésekor a mai napig az előzőekben ismertetett két klasszikus modell valamelyike a kiinduló pont (Abramson, 2004) (Sharda, et al., 2018). Egy konkrét adattárház esetén érdemes figyelembe venni, hogy a modellek közötti különbségeket:

- Inmon felülről lefelé (az adatpiacok oldaláról kiindulva) közelíti meg a problémát, ezzel szemben Kimball alulról felfelé (az adattárházból kiindulva).
- Inmon modellje komplexebb, nehezebben megvalósítható.
- Az Inmon-modell témavezérelt, míg Kimball modellje folymatvezérelt.
- Inmon adattárháza relációs felépítésű, míg Kimball esetén az adattárház modellje dimenzionális.

Ha választani kell Kimball, illetve Inmon modellje közül, akkor a következő szempontokat kell figyelembe venni (Abramson, 2004):

- Kimball modellje a megfelelő,
    - a felhasználók IT területről kerülnek ki
    - inkább taktikai döntések szükségesek
    - a forrásrendszerek viszonylag stabilak
    - minél előbbi eredményt szeretnénk elérni, kis kezdeti befektetéssel és csapattal

    - az adatok különálló üzleti területekről jönnek
    - a változások köre limitált
- Inmon modellje a megfelelő, ha
    - a felhasználók nem IT szakemberek
    - stratégiai döntések vannak túlsúlyban
    - a forrásrendszerek gyakran változnak
    - több idő, pénz és nagyobb létszámú csapat áll rendelkezésre
    - vállalati szintű adatintegráció szükséges
    - a változások köre bővülhet.

2.4 Az adattárházak napjainkban

Manapság az adatok értéke egyre gyorsabban nő, így egyre több cég épít adattárházakat. A mai adattárházaknak számos új kihívással kell szembenézniük, például a big data, a valós idejű válaszok, a gyakori változások az adatmodellben vagy a nem strukturált (szöveges) adatok kezelése.

A „big data” kifejezés olyan speciális adathalmazra utal, amely rendszerint nagy mennyiségű, változatos adatforrásból származó adatot tartalmaz a klasszikus relációs adatoktól a közösségi hálózati adatokig. További fontos jellemző, hogy a big data adathalmaz mérete általában nagyon gyorsan növekszik. Az adatok sok esetben valós időben, adatfolyam (stream) formájában érkeznek.

A szakirodalom a big data fontosabb jellemzőit v betűvel kezdődő szavakkal emeli ki. Ezen szavak közül a legfontosabbak a volume (mennyiség), variety (változatosság) és a velocity (sebesség). Ezek mellett további v-betűs big data jellemzők is megemlíthetők, pl.: value (érték) (Goes, 2014), visualization (vizualizáció), variability (inkonzisztens adatok) és veracity (adatok érvényessége). A v-betűs szavak számától függően big data 3V, 4V, vagy akár 10V modellről is beszélhetünk.

A big data kihívás kezelése az adattárházak és BI rendszerek számára sok problémát vet fel, főleg két területen. Először az ETL műveletek során, amelyek alacsony szintű, nyers adatokból állítanak elő strukturált információt. Másodszor pedig a számítások során, ugyanis még a legújabb megoldások sem képesek boldogulni az OLAP-kockák (lásd később) számítás igényével. Ez a számítási igény egyrészt a feldolgozandó adatok mennyiségének, másrészt komplexitásának köszönhető. A komplexitás ugyanis a big data adatkészletekben nagyon magas lehet többek között a szabálytalan hierarchiák, kardinális leképezések miatt (Cuzzocrea, et al., 2015).

A legtöbb adattárház esetén naponta egy-két alkalommal frissítik az adatokat. A frissítések többnyire a tranzakciós forrásrendszerek állásidejében (jellemzően éjszaka) történnek. A frissítés alatt általában az OLAP alkalmazások nem férnek hozzá az adatokhoz. A napi egy-két frissítés nem mindig elég. A modern adattárházak esetén gyakori követelmény, hogy az új vagy megváltozott adatok egy része minél előbb megjelenjen az adattárház riportokban is. Például egy banki rendszer vagy egy repülőjegy foglalásokat kezelő rendszer esetén (közel) valós idejű adatokra is szükség lehet. A szinte folytonos frissítések a hagyományostól eltérő architektúrát igényelnek.

Ennek lényege röviden, hogy az érintett tranzakciós forrásrendszeri adatok egy átmeneti adatbázisba is bekerülnek (replikálódnak). Amennyiben változás történik az átmeneti adatbázis adataiban (új rekord, módosított rekord vagy törölt rekord), akkor az adattisztítás és az adattranszformáció után a megfelelő változások bekerülnek az adattárházba is (Ali & Mohamed, 2016).

Az állandóan változó igényeknek köszönhetően a mai adattárházak – az adatmodellt tekintve - gyakori módosításoknak vannak kitéve. Az adatmodellben történő minden egyes módosítás változtatásokat igényelhet az ETL-folyamatokban, a megjelenítési rétegben és a BI alkalmazásokban is.

Ennek a problémának az egyik megoldási lehetősége az un. data vault metodológia. Ez lehetővé teszi az adatmodell dinamikus kiterjesztését anélkül, hogy komolyabb módosításokat kelljen végezni a többi érintett rendszer elemen. Ennek oka, hogy a data vault meta-model szinten kezeli az adatmodellt. Ez a gyakorlatban az üzleti kulcsok (hub), az üzleti kulcs tranzakciók (link) és az üzleti kulcs történet (sat) különválasztását jelenti.

Az üzleti kulcsok olyan objektum tulajdonságok, amelyek egyértelműen azonosítják az adott objektumot. Az üzleti kulcs tranzakciók olyan események leírásai, amelyek az üzleti kulccsal azonosított objektumok között történtek. Az üzleti kulcs történet olyan objektumtulajdonságok változásait jelenti, amelyek funkcionálisan függenek az adott üzleti kulcstól. A változások, események időbélyeggel vannak ellátva, így könnyen megállapítható, hogy egy adott időpontban pl. melyik objektumtulajdonság érték volt érvényes (Demidova, 2018).

Az adattárház fogalma elvileg platform független, azaz nem kötődik konkrét technológiához. Mégis, ha adattárházról van szó, akkor szinte mindig azt feltételezzük, hogy az adatok tárolása relációs módon történik. Ezen adatok SQL-nyelv segítségével kérdezhetők le.

Napjainkban viszont az újgenerációs (nem relációs, NoSQL) adatbázisok egyre nagyobb teret hódítanak (Bicevskaa & Oditis, 2017). Közös jellemzőjük a nagyfokú skálázhatóság, a rugalmas adatmodell, a magas írási/olvasási teljesítmény, valamint az, hogy az adatok lekérdezéséhez általában nem az SQL nyelvet használják. Ezeket az adatbázisokat négy csoportba sorolhatjuk:

- Dokumentum adatbázisok. Az adatokat címkézett elemekből álló dokumentumok segítségével tárolják. Pl: MongoDB
- Gráf adatbázisok. Csomópontokat és éleket használnak az adatok és a közöttük lévő kapcsolatok tárolásához. Pl: Neo4J
- Kulcs-érték tárolók. Kulcsokat és értékeket tartalmazó táblázatot használnak az adatok gyors eléréséhez. Pl: Dynamo
- Oszlopalapú tárolók. Az adatokat nem sorfolytonosan, hanem oszloponként tárolják. Pl: Cassandra

A NoSQL-adatbázisok elterjedése napjainkig nem befolyásolta jelentősen az adattárházaknál alkalmazott technológiákat. Továbbra is relációs adatbázis rendszereket alkalmaznak az adatok tárolása és lekérdezése során. Ennek két fő oka van:

- A cégek meglévő adatai általában relációs rendszerekből származnak
- Az adatfeldolgozási folyamatoknál még eléggé szokatlan a nemrelációs technológiák használata

Valószínűsíthető, hogy a NoSQL technológiák előbb-utóbb az adattárházaknál is megjelennek. Az a képességük, hogy képesek kezelni a félig strukturált és nem strukturált adatokat is, ellensúlyozhatja az előbb felsorolt hátrányaikat. Egy ilyen lehetőséget körvonalaz Bicevska és Oditic (Bicevskaa & Oditis, 2017), amely egy NoSQL adatpiacot valósít meg egy prototípus létrehozásával MongoDB és Clusterpoint DB környezetben.

2.5 Üzleti analitika – az adattárházakra épülő elemzési megoldások

Az adatelemzés (Data Analytics) magában foglalja az adatgyűjtés, átalakítás, tárolás, kezelés, kivonás, elemzés, modellépítés és vizualizáció összes folyamatát. Ez egy interdiszciplináris tudomány, amely felöleli többek között a statisztika, az adattudomány, valamint az operációkutatás területeit.

Az üzleti analitika (Business Analytics) az adatelemzés egy részterülete. Olyan eszközöket és technikákat (metrikák, modellek) foglal magában, amelyek segítenek a vezetőknek a minél hatékonyabb üzleti döntések meghozatalában (Sundararajan, 2020) (Kő & Gillani, 2019). Az alkalmazott modellek három kategóriába sorolhatók (Brinkmann, 2019):

- Leíró (descriptive) modellek. Arra a kérdésre adnak választ, hogy mi történt a múltban. Az adatokat statisztikai módszerekkel vizsgálják, és sok esetben vizuálisan jelenítik meg diagramok, jelentések vagy irányítópultok formájában. Fontos segítséget nyújtanak az adatok értelmezéséhez és az üzleti életben bekövetkezett változások megértéséhez.
- Előrejelző (predictive) modellek. Arra a kérdésre adnak választ, hogy mi fog történni a jövőben. A meglévő adatokban rejlő trendeket és mintákat pl. egy gépi tanulási modell segítségével alkalmazzák az aktuális adatokra, ezáltal képesek előrejelezni, hogy mi várható.
- Előíró (prescriptive) modellek. Arra a kérdésre adnak választ, hogy miért fog valami történni a jövőben. Különböző cselekvési irányokat javasolnak, és felvázolják, hogy ezeknek milyen következményei lehetnek.

A leíró adatelemzés során használt statisztikákat három kategóriába lehet besorolni (Bhandari, 2020):

- Gyakorisági eloszlások. Pl. abszolút gyakoriság, relatív gyakoriság.
- Szóródási mérőszámok. Pl. szórás, terjedelem.

- Központi tendencia (középérték) mérőszámai. Pl. átlag, medián, módusz.

Az előrejelző elemzések is három csoportra oszthatók (Bousdekis, 2020):

- Valószínűségi modellek. Ezek bizonytalan ok-okozati összefüggéseket ábrázolnak adott események bekövetkezési valószínűségeinek kiszámításával. Ilyen modellek pl. a Bayes-hálók, a Markov-lánc Monte Carlo vagy a rejtett Markov-modell.
- Gépi tanulási modellek. A mintaadatok egy matematikai modelljét az un. tréning adatokat építik fel abból a célból, hogy előrejelzéseket vagy döntéseket hozzanak anélkül, hogy explicit módon programozták volna őket az adott feladat megoldására. Ilyen modell pl. a döntési fa, a mintafelismerés vagy mesterséges neurális hálózat.
- Statisztikai elemzések. Az adatok minden aspektusával foglalkoznak és megoldják a statisztikai sokasággal vagy a statisztikai modell folyamatával kapcsolatos problémákat. Ide tartozik pl. a lineáris regresszió vagy a sztochasztikus idősorok elemzésénél használt integrált autoregresszív mozgóátlagolás (ARIMA).

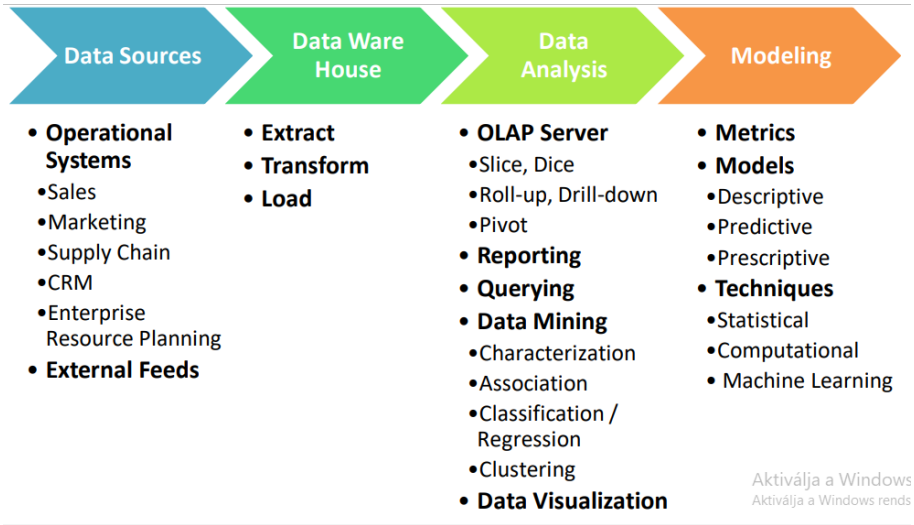
Az előíró elemzés esetén hat kategóriát lehet megkülönböztetni (Bousdekis, 2020):

- Valószínűségi modellek (ld. előző felsorolás).
- Gépi tanulási modellek (ld. előző felsorolás).
- Matematikai programozás. A korlátozottan rendelkezésre álló erőforrások egymással versengő tevékenységek közötti optimális elosztásával foglalkozik, figyelembe véve a probléma jellegéből adódó korlátozásokat. Ide tartozik pl. a lineáris programozás vagy a sztochasztikus optimalizálás.
- Evolúciós számítási modellek. A globális optimalizálás modelljei közé tartoznak, a biológiai evolúció analógiája alapján működnek. Egy kiinduló megoldáskészlet (generáció) iteratív módon való változtatásával (a kevésbé kívánt megoldások sztochasztikus eltávolításával és véletlenszerű módosításokkal) képesek olyan komplex feladatok megoldására, amelyek esetén nem lehet egzakt megoldásokat levezetni. Ilyen pl. a genetikus algoritmus.

- Szimulációs modellek. Egy valós vagy elképzelt helyzetet modelleznek, így segítségükkel tanulmányozni lehet, hogy hogyan működik a rendszer. A modellekben lévő változók értékeinek megfelelő megváltoztatásával előrejelzések adhatók a rendszer viselkedésére, illetve növelhető a humán döntés vagy az alkalmazásokba épített döntési logika hatékonysága. Ilyen szimulációs modell pl. a kockázatértékelés.
- Logikai alapú modellek. Okok és következmények feltételezett láncolatának segítségével proaktív módon támogatják a döntéshozatalt. Ide tartoznak pl. az asszociációs szabályok.

Az üzleti analitika megjelenése a 2000-es évek elejéhez köthető. Előtte kb. 10 évvel jelentek meg az üzleti intelligencia (Business Intelligence, BI) rendszerek, amelyek szintén a vezetői döntéstámogatásban játszanak szerepet. A különbség közöttük az, hogy az üzleti intelligencia gyakorlati megközelítésben elsősorban leíró modelleket használ, míg az üzleti analitika inkább előrejelző, illetve előíró modelleket.

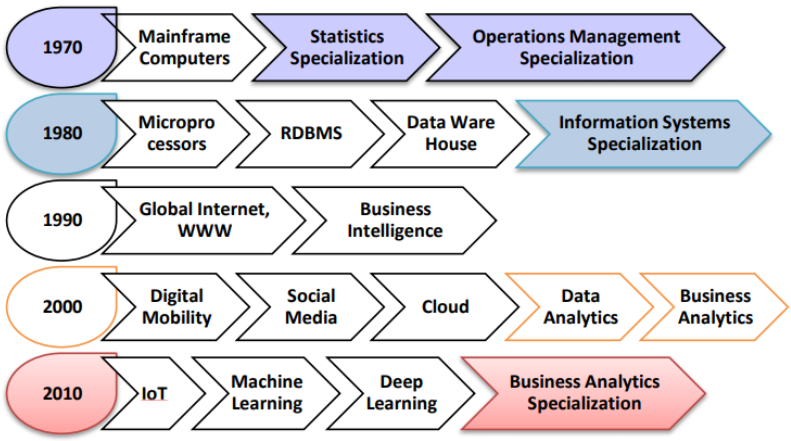
Az üzleti intelligencia kulcsterületei az adatbányászat (minták, összefüggések keresése nagy tömegű adathalmazból) és az OLAP technikák alkalmazása. Az elemzendő adatok sok esetben adattárházból származnak, mint ahogyan azt a következő ábra is mutatja:



3. Ábra: Adattárház és adatbányászat (Sundararajan, 2020)

Az üzleti analitika kialakulását és fejlődését a digitális mobilitás, a közösségi média és a felhőalapú technológiák elterjedése alapozta meg. Ehhez az időszakhoz köthető az adatmennyiség gyors növekedése (big data), és a hagyományos adatfeldolgozó rendszerek válsága.

A következő ábra az üzleti analitika kialakulásának folyamatát szemlélteti.



4. Ábra: Az üzleti analitika kialakulása (Sundararajan, 2020)

2.6 Ontológiák

Az Internet első korszakában megjelenő tartalmakra az volt a jellemző, hogy elsősorban humán felhasználásra tervezték őket. Emiatt a honlapokra feltöltött nem strukturált adatok kezdetben egyáltalán nem voltak alkalmasak számítógépes feldolgozásra.

Később a webes tartalmak egyre inkább olyan formában jelentek meg, amely a számítógépek számára jól olvasható. Ebben fontos szerepet játszott Berners-Lee, aki először írta le az un. szemantikus web fogalmát 2001-ben (Berners-Lee, 2001). Ő volt annak a World Wide Web Consortium-nak (W3C) a vezetője, amely hivatalosan is meghatározta ezt a fogalmat a következőképpen:

„A szemantikus web egy közös keretet biztosít az adatok határok nélküli megosztására és újrahasznosítására az alkalmazások, intézmények és közösségek között”

A szemantikus web felfogható a világháló egyfajta olyan kiterjesztésének, amely jelentést is tulajdonít az adatoknak.

A szemantikus web egyik legfontosabb leíró nyelve az RDF (Resource Description Framework) (Berners-Lee, 2001), amely a webes adatcsere szabványos modelljévé vált. Ez tulajdonképpen egy XML-alapú nyelv, amely hármasokat (tripletek) használ. A tripletek tárgy – alany – állítmány formában írják le a dolgokat. Az RDF struktúra lekérdezhető speciális lekérdező nyelvek segítségével. Ezek közül a legismertebb a SPARQL. Az RDF komponensek egy része az URI (Uniform Resource Identifier) segítségével azonosíthatók.

A szemantikus web szabványok lehetővé teszik az adatok interoperabilitását (átjárhatóságát) egy hatalmas, elosztott adatterület létrehozásával. A felhasználók és az alkalmazások elérhetik, integrálhatják és összesíthetik az itt tárolt információkat, függetlenül azok eredetétől és fizikai helyétől (d'Aquin & Noy, 2018). Az RDF és a hasonló szabványos formátumok az interoperabilitást csak szintaktikai szinten teszik lehetővé. A szemantikai szintű interoperabilitás eszközei az ontológiák, amelyek biztosítják az a fogalmi jelentés meghatározását. Az ontológiák megoszthatók, összehangolhatók és újra felhasználhatók.

Az ontológia az ókori filozófiában a lét tanulmányozásával kapcsolatos tudomány volt. A mesterséges intelligenciában az ontológiák a tudásreprezentáció lehetséges eszközei. Egy adott szakterülettel (domain) kapcsolatos fogalmak (többnyire) hierarchikus gyűjteményét jelentik, amely segítségével lehetővé válik a tudás megosztása és ismételt felhasználása (Biemann, 2005).

A szakirodalomban egy gyakran hivatkozott ontológia meghatározás Gruber nevéhez fűződik:

"Az ontológia a fogalmi modell (fogalomalkotás) világos és részletes leírása (Gruber, 1993).", ahol a fogalmi modell, illetve a fogalomalkotás egy adott szakterület gondolkodásmódját tükrözi.

Az ontológiák többféle csoportosítása ismert. Guarino négy kategóriát különböztetett meg (Guarino 1995):

- Magasszintű ontológia (referencia ontológia): olyan általános fogalmakat ír le, amelyek szakterület, feladat és alkalmazás függetlenek, mint pl. a tér, idő, tárgy, esemény stb. Az itt leírt fogalmakra hivatkozhatnak más ontológiák gyökérfogalmai, ezáltal a magasszintű ontológiák képezik alapját a rendszerben előforduló többi ontológiának. Példaként a Sowa által fejlesztett ontológiát lehet említeni (Sowa, 2000).
- Szakterületi ontológia (domén ontológia, tartományi ontológia): valamely szakterület fogalomkészletének leírását tartalmazza. A szakterületi ontológiák újra felhasználhatják vagy specializálhatják a magasszintű ontológiákból származó fogalmakat. Ilyen szakterületek, pl. az orvostudomány, geológia, gazdálkodás, pénzügyek, amelyeket az olyan feladatoktól és problémáktól függetlenül kezelnek, amelyek pedig a szakterülettel kapcsolatban felbukkanhatnak. Egy konkrét szakterületi ontológia pl. az OBO (Open Biological and Biomedical Ontologies), amely orvosi és biológiai fogalmak leírását tartalmazza.
- Feladat ontológia: egy tevékenység, vagy feladat leírását tartalmazza, a magasszintű ontológia specializálásának megfelelően. Tárgya a problémamegoldás. Például egy ilyen tevékenység lehet a képfeldolgozás.
- Alkalmazás ontológia: a legspeciálisabb ontológia, amely a szakterületi, illetve a feladat ontológia egy specializálását jelenti valamely konkrét alkalmazásra. Az alkalmazás ontológiák általában nem használhatók fel más alkalmazásokhoz.

A fentiek alapján az ontológiák jellemzésére használatos főbb dimenziók a következők:

- Formalizáltság: a formalizáltság foka, amely a terminológia jegyzékre és a szavak jelentésének megfogalmazására használnak;
- Cél: mire kívánják az ontológiát használni;
- Szakterület: a szakterület természete, amelyet az ontológia leír.

A formalizáltság mértéke az informális természetes nyelvtől a szigorú formális nyelvig terjedően négyféle lehet:

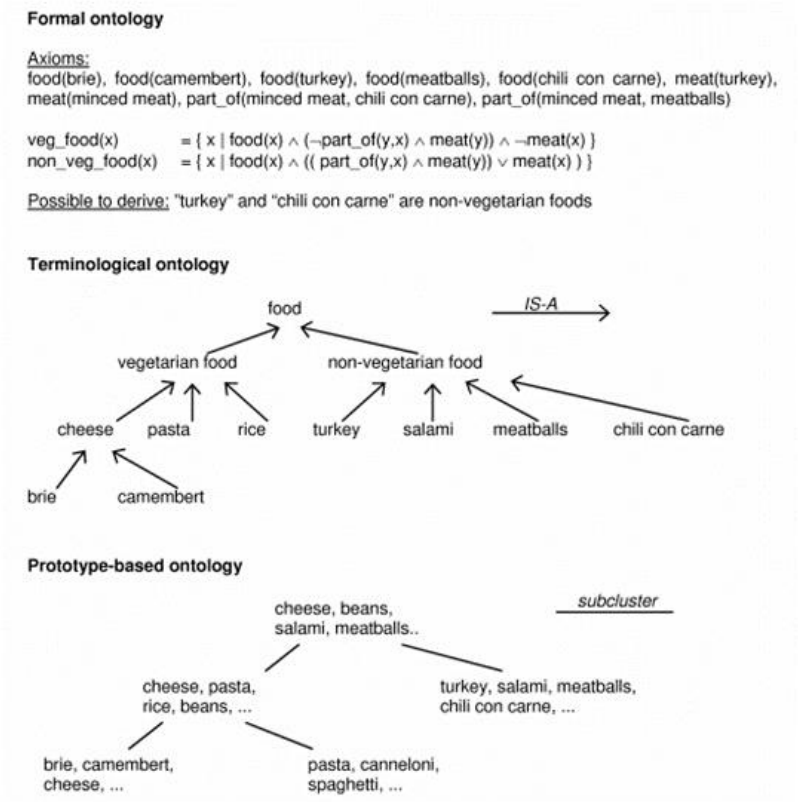
- Nem formalizált: informálisan kifejtett, természetes nyelven megfogalmazott;

- Strukturált informális: strukturált és korlátozott természetes nyelven megírt, amely nagymértékben növeli az érthetőséget és csökkenti a kétértelműséget (pl. a ’szervezeti ontológia’ szöveges változata);
- Félig-formális: egy ilyen célra kialakított, mesterséges specifikációs nyelvben kifejtett leírás (pl. ’szervezeti ontológia’ Ontolingua leírása);
- Szigorúan formális, szabatos: meghatározott alapfogalmak, formális szemantikai leírással, tételek és bizonyítások, többek között az elmélet konzisztenciájára és teljességére vonatkozóan (TOVE).

Az ontológiák egy másik lehetséges csoportosítása (Biemann, 2005):

- Formális ontológiák (formal ontology) esetén a fogalmak megkülönböztetése és kategorizálása definíciókkal és axiómákkal történik
- Prototípus-alapú ontológiáknál (prototype-based ontology) tipikus példányok (prototípusok) megadása szolgál a fogalmak leírására. A tipikus példányok kiválasztásához un. hasonlósági metrikákat kell definiálni
- Szaknyelvi (terminological) ontológiák jellemzői a típus-altípus kapcsolatok és a szinonimák használata a fogalmak leírására. Jól ismert példa ezekre a WordNet.

Az egyes típusok közötti különbséget egy egyszerű példán keresztül lehet szemléltetni. A példa az ételekkel (vegetáriánus és nem vegetáriánus) kapcsolatos, a következő ábrán látható:



5. Ábra: Ontológia típusok (Biemann, 2005)

Formális ontológiák segítségével a leírás a következőképpen történik:

- axiómák: a sajt egy étel, a kacsa egy hús, a darált hús része a fasírtnak stb.
- definíciók: vegetáriánus étel = olyan étel, ami nem hús és nem része húsnak

Szaknyelvi ontológiákkal ugyanez: a camembert egy sajt, a sajt vegetáriánus étel, ami az ételek közé tartozik stb. Prototípus-alapú ontológiák esetén: a sajt, bab, szalámi, fasírt… példányok alcsoportja a kacsa, szalámi, chilis bab stb.

Az 1990-es évek kezdetétől az ontológiák megvalósítására, kezelésére számos nyelv jött létre (Gomez, et al., 2002). Ezek a legtöbb esetben a matematikai logikát vagy a kereteket (frame, egy olyan adatstruktúra, amely egy fogalmat ábrázol egy adott területen) használják a nyelv leírására.

- A KIF az elsőrendű logikára épülő nyelv, amelynek elsődleges célja különböző tudásábrázoló rendszerek közötti adatcsere megvalósítása volt.
- A Loom elsősorban tudásábrázolásra használt nyelv, amely leíró logika és következtetési szabályok segítségével képes volt a fogalmak automatikus osztályozására.

- Az OCML nyelv további komponenseket is tartalmazott (pl. függvény műveletek definícióit), és végrehajtható ontológiákat tudott generálni a problémamegoldó módszerek számára.
- Az Flogic az elsőrendű logika és a keretek kombinációjával lehetővé tette fogalmak, taxonómiák (osztályozás), bináris kapcsolatok, függvények, példányok, következtetések és axiómák ábrázolását.
- A SHOE a HTML nyelv kiterjesztése, amely tag-eket használ, így lehetővé teszi ontológiák beépítését HTML dokumentumokba. Később az XML elterjedésével a SHOE is XML-alapú nyelv lett.
- Az XOL egy nagyon egyszerű ontológia nyelv, amely csak fogalmakat, taxonómiákat és bináris kapcsolatokat képes kezelni
- Az RDF (lásd következő alfejezet) webes erőforrások leírására jött létre. Ennek kiterjesztése az RDF Schema (RDF(S)), amely keret-alapú elemeket is tartalmazott.
- Az OIL, DAML és OWL nyelvek az RDF(S)-ből fejlődtek ki.

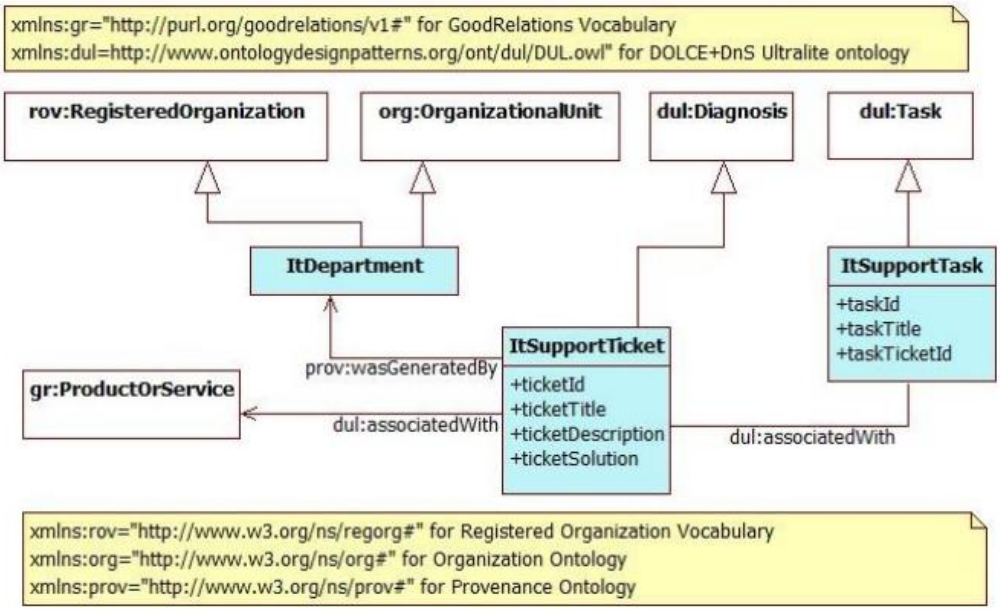
Az ontológiák létrehozását számos szoftveres eszköz támogatja (Slimani, 2015).

- A DUET és az UBOT UML-alapúak, ontológiák tárolására nem képesek
- Az OntoEdit alapnyelve az FLogic, képes ontológiák tárolására is
- A Protege az egyik legnépszerűbb eszköz. Képes kezelni az összes gyakoribb ontológia formátum kezelésére és ontológia könyvtárak építésére
- Az OiLED egyszerű ontológia editor, amely OIL nyelvű ontológiák létrehozására alkalmas
- Az Ontolingua egy webes felületű program, amely számos ontológia nyelvet támogat
- Az OntoSaurus Loom nyelvű tudásbázisok létrehozására szolgál
- A WebODE HTML és Java alapú program, amely különlegessége, hogy az ontológiákat egy adatbázisban tárolja
- A WebOnto az OCML nyelvű ontológiák létrehozására szolgál
- A Swoop, Hozo, Dogma Studio Workbench, TopBraid Composer, OWLGrEd és a Graffoo mindegyike OWL nyelvre épül.

A szemantikus weben közzétett adatok növekedése maga után vonta a kifejlesztett ontológiák számának gyors növekedését is. Az ontológia könyvtárak olyan rendszerek, amelyek különböző forrásokból gyűjtenek ontológiákat, és lehetővé teszik azok keresését, szűrését és felhasználását. Ilyen ontológia könyvtár például a BioPortal, az OntoSelect vagy az ONKI (d'Aquin & Noy, 2018).

A meglévő ontológia könyvtárak egyre több témát fednek le. Természetesen lehetnek olyan szakterületek is, amelyeknél nem állnak rendelkezésre ontológiák, vagy azok nem megfelelőek egy adott feladatra. Ilyen például a prototípus elkészítésénél használt helpdesk (ticketing) rendszer is, ahol a szakirodalomban meglehetősen kevés használható forrás található. Ezek közül a disszertáció szempontjából a következők lehetnek relevánsak:

- Leung (2012) többszintű ontológiát készített a helpdesk kérések kategorizálására. Ennek legfelső szintjén a kérés (Helpdesk Enquiry) osztály van, amely négy alosztályra bomlik: Hardware Problem, IT Admin Issue, Software Problem és Other Problem. Ezek az alosztályok további 3-4 szintre bonthatók, pl. egy okostelefonnal kapcsolatos probléma kategóriája a Helpdesk Enquiry – Hardware Problem – Non Standard Hardware Problem – Mobile Gadget Problem – Smartphone Problem útvonalon érhető el. Ez az ontológia csak osztály-alosztály kapcsolatokat tartalmaz.
- Kotis, et al., (2014) esettanulmányukban létrehoztak egy IT Helpdesk Support tématerületi ontológiát. Ennek fő osztályai: ItDepartment, ItSupportTicket, ItSupportTask és ProductOrService. A létrehozás főleg meglévő ontológiák és szótárak újrahasznosításával történt. Ilyenek pl. a W3C-OrgOnto vagy V3C-ROV. Az elkészült helpDeskOnto ontológia sémáját a következő ábra szemlélteti.

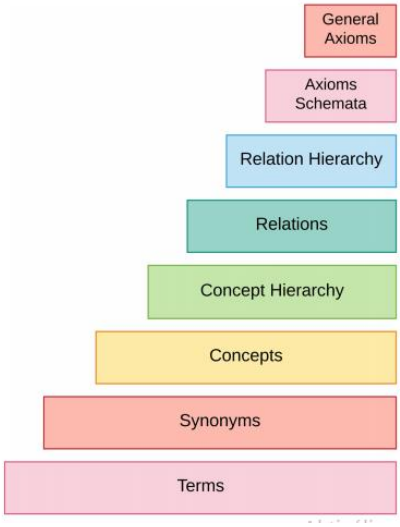


6. ábra: A helpDeskOnto ontológia (Kotis, et al., 2014)

Az előző ontológiához képest itt már többféle kapcsolattípus is megjelenik, emellett az osztályoknál tulajdonságok is definiálásra kerülnek. Például az ItSupportTicket osztály társítási kapcsolatban áll az ItSupportTask, az ItDepartment és a ProductOrService osztályokkal, valamint rendelkezik négy tulajdonsággal: hibajegy azonosítója (ticketId), címe (ticketTitle), leírása (ticketDescription) és megoldása (ticketSolution).

Ontológiákat fejleszteni lehet akár a meglévő ontológiák felhasználásával, akár azok nélkül is. Mindkét esetben szükség lehet valamilyen módszertan alkalmazására. Az ontológia fejlesztési módszertanokról a 4.3 alfejezetben esik szó.

Az ontológiák létrehozása nem csak manuális módon történhet. A félig automatikus vagy automatikus létrehozást ontológia tanulásnak nevezzük. Az ontológia tanulás első lépése a kifejezések és szinonimák kivonása a rendelkezésre álló szövegből. Ezután a megfelelő kifejezésekből és szinonimákból fogalmak lesznek, majd a fogalmak közötti összefüggések (relációk) megtalálása következik. A folyamat végét az axióma sémák példányosítása és az általános axiómák kinyerése jelenti (Asim, et al., 2018).



7. Ábra: Az ontológia tanulás lépései (Asim, et al., 2018)

Az ontológia tanuló rendszerek által használt technológiákat három csoportra oszthatjuk:

- Nyelvi technológiák. Elsősorban a szöveg előfeldolgozása során, valamint a kifejezések, fogalmak, és relációk kinyerése során használatosak. A szöveg előfeldolgozás tipikus technikái a szöveg címkézés (bizonyos szövegegységek megjegyzésekkel való ellátása, pl. szófaj), a szöveg elemzése és a lemmatizálás (lemmákra bontás, lemmák - azonos szótőből származó szóalakok). A kifejezések és fogalmak kinyerésének három gyakori algoritmusa a szintaktikai elemzés, az alkategóriák kialakítása és az alapszavak (adott téma/terület releváns szavai) használata.
- Statisztikai technológiák. Ezek nem veszik figyelembe a szemantikát, kizárólag a szöveg statisztikáin alapulnak. Legtöbb esetben valószínűségeket használnak, és általában az ontológia tanulás korai szintjén használatosak. Ilyen technológia például a C/NC érték (többszavas kifejezésekhez rendelt pontszámok), a klaszterezés (a hasonló kifejezések csoportokba sorolása valamilyen távolság- vagy hasonlóság mérték alapján úgy, hogy az egyes csoportok homogének legyenek, de egymástól különbözzenek), vagy az együttes előfordulás elemzése.

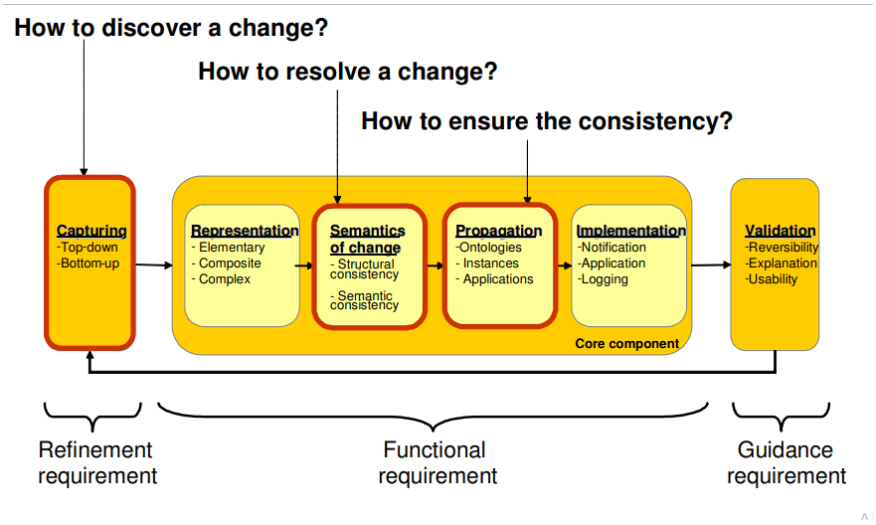
- Logikai technológiák. Az induktív logikai programozás a gépi tanulás témaköréhez tartozik, amely a háttérismereteken és logikai programozási példákon alapuló hipotéziseken alapul. Az ontológia tanulás utolsó szakaszában van szerepe, ahol a sematikus axiómákból általános axiómák jönnek létre. Logikai egyszerűsítéssel és formális reprezentációs algoritmusok segítségével lehetővé teszik a kifejlesztett ontológiák ábrázolását.

Az ontológia alapú alkalmazások folyamatos változásnak vannak kitéve. A változást okozhatja a rendszer működési környezetének megváltozása (új üzleti területek, funkciók), a felhasználói igények változása (új alkalmazottak, új kompetenciák), vagy a belső folyamatok újratervezése.

Ha az ontológia nem naprakész, akkor a rendszer megbízhatósága, pontossága és hatékonysága jelentősen csökken. Ezért a rendszerben bekövetkezett változásoknak tükröződniük kell az alapul szolgáló ontológiákon. Az ontológiák karbantartása időigényes folyamat, amelynek költsége akár meg is haladhatja a fejlesztés költségeit. Az ontológiák karbantartása során ugyanis a következő kihívásokkal kell szembenézni (Stojanovic, 2004):

- Komplexitás. Az ontológiai modell általában összetett szerkezettel rendelkezik. Még viszonylag kis változások esetén is az összes változás kumulatív hatása nagyon nagy lehet
- Függőségek. Az ontológiák gyakran felhasználják és kiterjesztik az alapul szolgáló ontológiákat, ezért az ontológiákban bekövetkezett változásokat szinkronizálni kell a függésben lévő ontológiákkal.
- Fizikai eloszlás. A függő ontológiák fizikai eloszlását is figyelembe kell venni az ontológia karbantartás során.

Az ontológiák karbantartásának folyamatát a következő ábra szemlélteti:



8. ábra: Az ontológiák karbantartási folyamata (Stojanovic, 2004)

Az első lépés a változások azonosítása (Capturing), amelyeket utána megfelelő formára kell hozni (Representation). Egy változás lehet elemi (csak egy ontológia entitást érint), kompozit (az adott ontológia egység szomszédait is érinti) és összetett (legalább két elemi vagy kompozit változást foglal magában).

A következő lépés a változások miatt fellépő inkonzisztenciák kezelése (Semantics of change). Ez lehet egyrészt a strukturális, másrészt a szemantikus inkonzisztencia. Előbbire példa, ha törlődik egy öröklési kapcsolat, akkor a gyermek entitás elveszítheti tulajdonságainak egy részét. Utóbbira példa, ha törlünk egy fogalmat, akkor a kapcsolódó fogalmak jelentése is megváltozhat.

A harmadik lépés a változásterjesztési szakasz (Propagation), amely a változások hatásait figyelembe véve az ontológia rendszert frissíti úgy, hogy ismét konzisztens állapotba kerüljön.

A negyedik lépés a végrehajtási szakasz (Implementation), amely összesíti a változás minden lehetséges hatását, végrehajtja a változást, majd nyomon követi annak minden lépését.

Az utolsó lépés a változás érvényesítési fázis (Validation), amely lehetővé teszi a végrehajtott változások érvényesítését, vagy szükség esetén a felhasználók általi visszavonását.

Az ontológiák a szemantikus weben betöltött szerepük mellett sok más területen is felhasználhatók (Taye, 2010), pl.:

- Mesterséges intelligencia. Az ontológiák megkönnyítik a tudás megosztását és újra felhasználását szolgáltatások, ügynökök és szervezetek között egy adott területet illetően.
- Keresőmotorok. Az ontológiák segítenek az internetes keresésben a keresett kifejezés szinonimáinak felhasználásával.
- E-kereskedelem. Az ontológiák egyik lehetséges felhasználási módja az áruk leírása, amely nagyon fontos az eladó és a vevő közötti kommunikációban. Ugyancsak az ontológiák tudnak segíteni abban, hogy a vevő megtalálja a kereséshez legjobban illő árucikkeket. Az ontológiák a gépi kommunikáció során is felhasználhatók, pl. online utazások szervezésénél a vevőknek küldött automatikus válaszoknál is alkalmazhatók.

2.7 Exploratory OLAP - áttekintés

Az adattárházakban tárolt nem strukturált vagy félig strukturált adatok egy részét (pl: megjegyzések, leírások) gyakran nem dolgozzák fel, figyelmen kívül hagyják. Ennek következtében információvesztés alakulhat ki, pedig van több megoldási lehetőség arra, hogy ezekből a - többnyire szöveges – adatokból is értékes információhoz jussunk.

Elterjedt megoldás az üzleti analitikai elemzésekben, riportokban, hogy az adatforrásokból lekérdezett, majd összegzett (aggregált) adatokból egy speciális struktúrát, kockát építünk. Az OLAP jelenti azt a számítási módszert, amely lehetővé teszi a felhasználók számára az adatok hatékony lekérdezését és elemzését különböző szempontok szerint. Más megközelítésben az OLAP egy multidimenzionális séma.

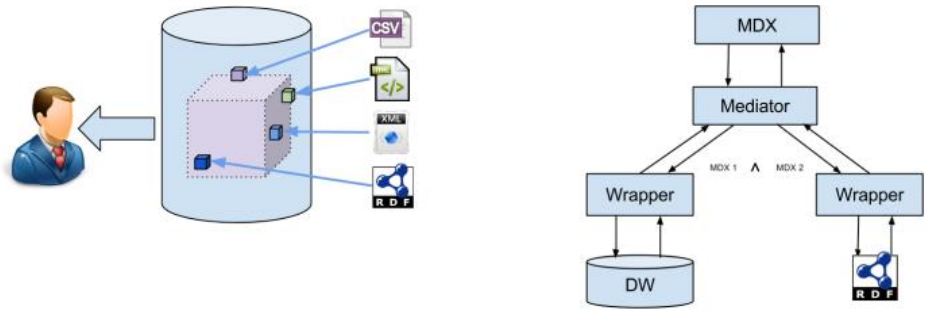
Azokat a technológiákat, pontosabban keretrendszereket (framework), amelyek lehetővé teszik, hogy nem strukturált adatokat is tartalmazó adattárház adataiból multidimenzionális sémát hozzunk létre, felfedező OLAP-nak (Exploratory OLAP) nevezzük (Abelló, 2015).

Az adattárház/OLAP technológiák jól működnek megfelelően strukturált adatok esetén. Kutatási kihívást jelent az, hogy mi a helyzet, ha nem, vagy félig strukturált adatokkal is kell dolgozni? Például, ha egy termék kampánnyal kapcsolatos riportot szeretnénk készíteni, akkor kombinálni kell a strukturált információkat (pl. termék eladási adatok) és a nem strukturált információkat (felhasználói vélemények).

Másképpen megközelítve, egyre több cég szeretni kihasználni a szövegesen tárolt adatokban lévő lehetőségeket, rejtett információt és beépíteni azokat a meglévő OLAP elemzésekbe. Ezt az újfajta OLAP koncepciót írja le az exploratory OLAP (Ibragimov, et al., 2015).

A szakirodalom nem egységes az exploratory OLAP fogalmi keretrendszerében, megvalósítási elveiben.

Az exploratory OLAP rendszerek egy lehetséges példája az Ibragimov által javasolt architektúra, amely elsősorban webes adatokkal dolgozik (Ibragimov, et al., 2015). Ibragimov és társai elvi megoldását az exploratory OLAP keretrendszerre a következő ábra szemlélteti.



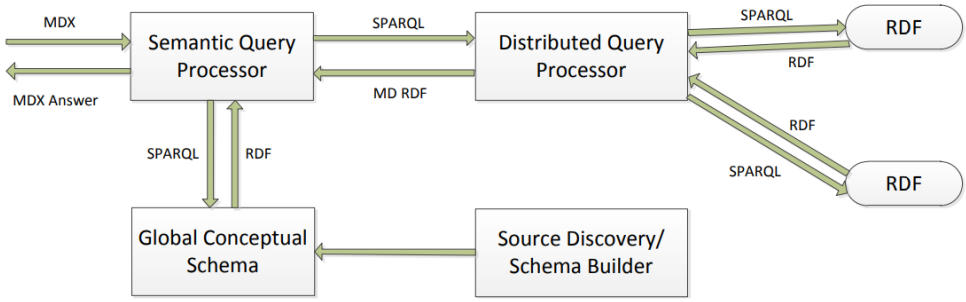
9. Ábra: Exploratory OLAP (Ibragimov, et al., 2015)

Ibragimov adatforrásként Linked Open Data (LOD) adatokat használt, amelyek RDF formátumban vannak tárolva.

A javasolt rendszer négy modulból áll (lásd. következő ábra):

- Global Conceptual Schema – az adatkockáról tárol információkat
- Semantic Query Processor – MDX lekérdezésekből állít elő SPARQL lekérdezéseket
- Distributed Query Processor – lekérdezi a végpontokat, összegyűjti az adatokat

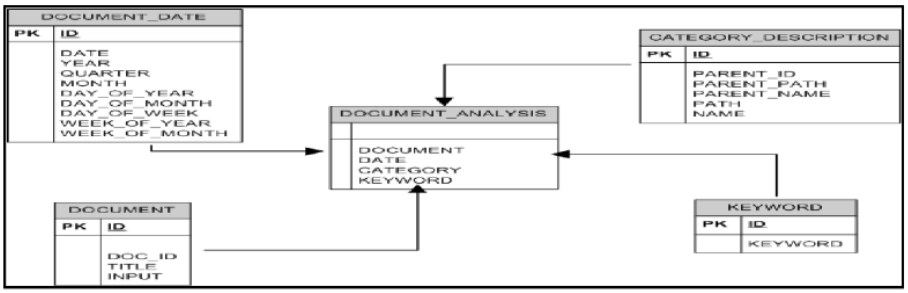
- Source Discovery Schema Builder – a felhasználókkal tartja a kapcsolatot a séma létrejötte alatt.



10. Ábra: Ibragimov architektúra (Ibragimov, et al., 2015)

Prasad és társai egy másik megközelítést javasoltak (Prasad, 2010). Az ő megoldásuk egy olyan szövegfeldolgozó és elemző technológián alapul, amely segítségével integrálhatók a strukturált és nem strukturált adatok az adattárházba. A megoldás a szöveg elemzéséből (statisztikai és szemantikus elemzés), a kulcs kifejezések azonosításából és szöveg címkék létrehozásából (text tagging) áll.

A szövegelemzés után az adatok betöltődnek az adattárházba. Ehhez használatunk XML-formátumot, amelyet minden adattárház rendszer képes kezelni. Ez tartalmazhatja a dokumentum tulajdonságait (azonosító, szerző, téma stb.), kulcsszavait és más fontos információkat. Ez a struktúra csillag sémára alakítható, ahol egy tény táblához kapcsolódnak a dimenzió táblák, amint ezt a következő ábrán is láthatjuk.



11. Ábra: A csillag séma (Prasad, 2010)

Abello és társai exploratory OLAP koncepciója további szemantikus web technológiákat használ (Abello & Romero, 2010). Ennek legfontosabb része az ontológia-alapú tudásreprezentáció.

Abello (2015) az ontológiákat használja a tények azonosítására. Mindegyik tény esetén azonosítja a lehetséges dimenzionális fogalmakat, amelyek egy jól elrendezett hierarchikus formában vannak tárolva funkcionális függőségek segítségével.

Az előzőekben bemutatott exploratory OLAP keretrendszer javaslatok mindegyike más és más módon kezeli a szöveges adatok feldolgozását. A következő táblázat az előzőekben említett három megközelítési módot hasonlítja össze:

Szempont	Abello	Ibragimov	Prasad
Adatforrások (nem strukturált)	**tetszőleges**	**Linked Open Data**	**tetszőleges**
Eszközök, technológiák	**ontológiák**	**RDF, MDX, SPARQL**	**szövegelemzés, XML**
Legnagyobb kihívás	**ontológia –-> MD**	**MDX → SPARQL**	**szövegelemzés**
Létezik-e prototípus terv	**nem**	**igen**	**igen**
Kimenet helye	**kocka**	**kocka**	**adattárház**
Kimenet formája	**OLAP séma**	**OLAP séma**	**Csillag séma**

1. táblázat: A bemutatott exploratory OLAP rendszerek összehasonlítása

Az itt bemutatott három exploratory OLAP megközelítés közül kettő esetén létezik ugyan prototípus terv, de nem tartozik hozzá konkrét megvalósítás.

Emiatt ismertetésre kerül egy negyedik ilyen rendszer is (Nebot, et al., 2009), amelyhez tartozik egy (orvosbiológiai) esettanulmány. Mivel ez – a szintén ontológiákat használó – rendszer elsősorban nem az általános működés miatt releváns, hanem a prototípus elkészítéséhez ad majd hasznos segítséget, ezért ennek bemutatására a 3.6-os alfejezetben kerül sor.

3 KUTATÁSI KÉRDÉSEK ÉS A KAPCSOLÓDÓ FOGALMI MODELL

Ebben a fejezetben részletezem a kutatási témát illető kihívásokat, majd megfogalmazom a kutatás hatókörét és a kutatási kérdéseket. Végül ismertetem azt az exploratory OLAP fogalmi modellt, amellyel a kutatás során dolgozni fogok.

Kutatásom központi kérdése az, hogy hogyan lehet a hagyományos OLAP és adattárház megoldásokat szemantikussá tenni, a nem strukturált adatokból származó tudáskinyerési megoldással kiegészíteni?

A 2.7 alfejezetben ismertetett három megközelítés közül Prasad (2010) módszerét választottam kiindulási koncepcióként az exploratory OLAP fogalmi megtervezésére és megvalósítására.

A módszer előnye, hogy az alkalmazandó szövegelemzési technológiák (szöveg cimkék, kulcsszavak keresése) megvalósítására számos lehetőség van akár R-ben, akár Python-ban. A hátrány, hogy az eredmény (csillag séma) az adattárházba kerül. Ezért még egy kiegészítő lépés szükséges, hogy ebből OLAP kockát előállítsunk.

Abello (2015) módszere nehezebben megvalósítható, de a három módszer közül ez tűnik a legalkalmasabbnak az exploratory OLAP fogalmi megtervezésére és megvalósítására, ugyanis

- A nem strukturált adatforrások széles körében alkalmazható
- Az adatok lekérdezése és átalakítása mellett a tudásábrázolást is megoldja
- Az ontológiák nagyon sok terület (domén) esetén rendelkezésre állnak, ezért ilyenkor csak az ontológia mappelésre van szükség, az ontológiák előállítására nem
- A cél elsősorban a nem strukturált adatok bevonása az elemzésekbe, riportokba. Ennek jobban megfelel az OLAP séma, mintha az adattárházba töltenénk be ezeket az adatokat is.

Az előzőek alapján Abello (2015) módszere lesz a fő koncepció az exploratory OLAP keretrendszer megvalósítására. Hátrányként figyelembe vettem azt is, hogy Abello (2015) módszerére nem találtam kellően részletes prototípus leírást. További nehézség lehet, hogy az általam választott szakterület (ticketing rendszer) esetén nincsen olyan meglévő, egy az egyben felhasználható ontológia, amely használható lenne a prototípus elkészítéséhez.

Ibragimov módszere szintén az ontológiákra épül, viszont más megfeleltetést használ (MDX-SPARQL). A módszer legnagyobb hátránya, hogy elsősorban webes adatforrásokra (Linked Open Data) alkalmazható, így felhasználhatósága emiatt korlátozott. Mivel a kutatás során kidolgozandó prototípus (ticketing rendszer) nem ilyen, ezért az Ibragimov által javasolt megoldást a gyakorlatban nem tervezem megvalósítani.

Az exploratory OLAP rendszerek prototípusainak értékelése során érdekes lehet majd megnézni, hogy ugyanarra az adatforrásra alkalmazva Prasad (2010), illetve Abello (2015) ötletén alapuló prototípusokat, milyen eredményre jutunk. Azaz, melyik módszer segítségével lehet több rejtett információ kinyerni, és azt a riportokban megjeleníteni.

3.1 Kihívások az exploratory OLAP megoldásokkal kapcsolatosan

A közeljövő (exploratory) OLAP rendszereinek számos kihívással kell szembenézniük.

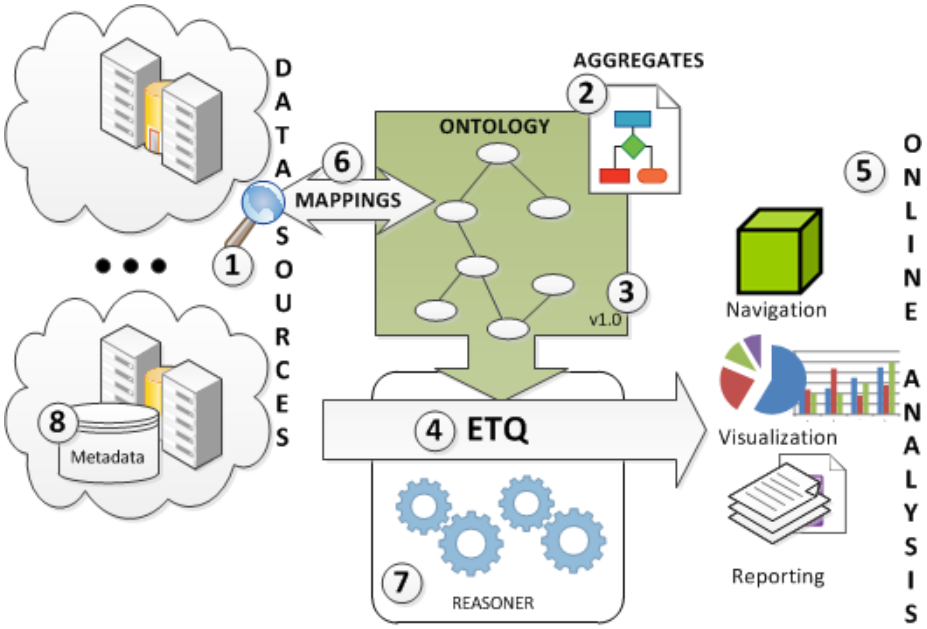
Prasad szerint (Prasad, 2010) a problémák két csoportra oszthatók:

- A szöveges adatok – lehetőleg automatikus - kinyerése az adatforrásokból, majd azok elemzésre alkalmas formára hozása nem egyszerű feladat. Megoldása rendszerint sok programozást igényel.
- A nem strukturált adatforrásokból kinyert “master data” adatok (pl.: a kulcsszavak) menedzselését, aktualizálását is meg kell oldani.

Abello szerint (Abelló, 2015) a kihívások részben a séma tervezésével, részben az adatszolgáltatással, valamint a szemantikus és számítási kapacitással kapcsolatosak. A következő ábra 1-8-ig való számozással jelzi az egyes kihívásokat érintő rendszerelemeket.

- Első probléma, hogy az adott területet érintő szakterületi ontológiák általában nem állnak rendelkezésre. Ezek – lehetőleg automatikus – előállítása a nem strukturált forrásrendszerekből nem triviális feladat. Vannak ugyan általános séma megfeleltetési és adatcsere módszerek, de ezeket még nem alkalmazzák az adattárházak tervezése során. Az ontológiák kinyerése többnyire offline módon történik, és bonyolult transzformációkat igényel.

- A második nehézség az OLAP kocka tervezésének egy fontos lépésénél, az aggregációk (összegzett adatok) tervezésénél adódik. Az ontológia leíró nyelvek általában ezt nem támogatják, ennek megoldására egyelőre csak előzetes kutatási eredmények vannak (Calvanese, et al., 2008).
- A harmadik probléma az ontológiák változásával kapcsolatos (evolúció, verziókezelés). Jelenleg ennek kezelése még gyerekcipőben jár. Az viszont biztos, hogy az időbeliség kezelése nehéz, nem szabványos ontológia érvelésekhez vezet (Motik, 2012).
- A következő kihívás az ETL folyamatot érinti. Van ugyan számos megoldás a félig strukturált adatok betöltésére, de ezek többnyire csak fogalmi szinten vannak megvalósítva. Ráadásul ezek integrálása az adattárház ETL folyamatai közé sem egyszerű, ugyanis számos olyan elemet tartalmazhat, amelyet a strukturált adatokat kezelő folyamatok nem ismernek. Például nem relációs operátorok, gépi tanulást alkalmazó számítások, összetett adattípusok stb.
- A szemantikai réteg integrálása az ontológiák segítségével performancia problémákat vethet fel, amelyeket figyelembe kell venni az elkészült terv optimalizálásánál is.
- Az exploratory OLAP rendszernek képesnek kell lennie dinamikusan integrálni az adatforrásokat anélkül, hogy a globális sémát és az ontológia mappelést előre definiálnánk. A dinamikus integráció olyan magas számítási költségeket eredményezhet, amelyek akár a kivitelezhetőséget is veszélyeztethetik.
- Az exploratory OLAP automatikus hozzáférést jelent nem csak a sémához, hanem bizonyos mértékben az adatokhoz is. Ez maga után vonja azt, hogy példány szinten is képesnek kell lenni az adatok értelmezésére és az érvelésre. Ontológiák használata esetén könnyebb (kevésbé számításigényes) az érvelés, ha az a társított példányok nélkül történik. Mindazonáltal az újabb megközelítések pl. ODBA (ontology-based data access) új utakat nyithatnak (Poggi, et al., 2008).
- Végül az utolsó nehéz kérdés az, hogy az alkalmazott szemantikus web technológia mennyiben alkalmas a vezetői döntés támogatásra. Itt a nemstrukturált adatok integrálása és az egyes adatforrások függetlensége okozza a fő problémát.



12. Ábra: a következő generációs OLAP rendszerek kihívásai (Abelló, 2015)

A kutatás során azt vizsgálom, hogy hogyan támogathatják a szemantikus technológiák az „exploratory OLAP" alapjául szolgáló adatgyűjtést, adatfeltárást, megismerést, integrációt, adatfeldolgozási és elemzési feladatokat a nem strukturált adatok esetében.

A fókusz az ontológia – OLAP megfeleltetésen van, azaz hogyan lehet a nem strukturált adatokból előállított ontológiák segítségével eljutni az elemzések alapjául szolgáló multidimenzionális sémáig.

3.2 Kutatási kérdések

A fentiek alapján a következő kutatási kérdéseket fogalmaztam meg, amelyekre a kutatás során szeretnék választ adni:

- Hogyan tehető szemantikussá a hagyományos OLAP és az adattárház, vagyis hogyan lehet megtervezni a szemantikus réteg integrációját az adattárházba?
- Hogyan lehet a megtervezett modellt a gyakorlatban megvalósítani egy prototípus segítségével?
- Mi lehet egy hatékony validálási módszer az „exploratory OLAP" modell esetében?

Az első kérdésre a szakirodalomban megtalálható exploratory OLAP megoldási javaslatok (Abello (2015), Prasad (2010)) figyelembevételével megtervezett modellek megalkotása lehet a válasz. Olyan modellt szeretnék létrehozni, amely minél univerzálisabban alkalmazható, és lehetőség szerint a gyakorlatban is működőképes.

A második kérdésre adott válasz működő prototípusok létrehozása az említett modellek alapján. Ennek fontos szerepe lehet az első kutatási kérdésben említett exploratory OLAP fogalmi modell működőképességének igazolásában, valamint alapjául szolgálhat újabb – más területeken használható – prototípusok létrehozásának.

A harmadik kérdéssel elméletben a 3.5 alfejezetben foglalkozom. Az ott leírtakat az egyes exploratory OLAP prototípusok elkészítése során gyakorlatban is alkalmazom

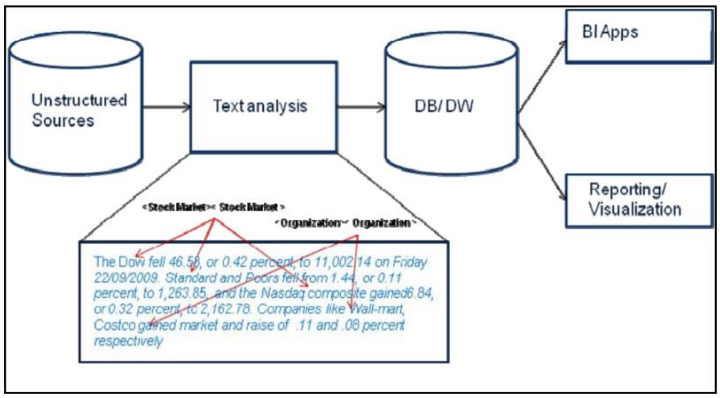
3.3 Szemantikus réteggel kiegészített exploratory OLAP modell

Ebben az alfejezetben ismertetem a megvalósítandó exploratory OLAP modellek fogalmi felépítését és működését. Az OLAP-séma létrehozásához szükség van az elemzés tárgyának (a tényeknek) az azonosítására. A tényeket különböző szempontok (dimenziók) alapján elemezhetjük. Az elemzés számszerűsítésére numerikus értékeket (mértékek) használunk. Ezek értékeit különböző összesítési (aggregálási) szinten lehet megjeleníteni. A tények egy-egy kapcsolat (azaz funkcionális függőség) révén kapcsolódnak a dimenziókhoz. A tényadatokat ennek megfelelően egyértelműen azonosítja a dimenzióértékek egy halmaza. Ezt a halmazt multidimenzionális azonosítóknak (MD ids) nevezzük. Mivel egyes dimenziók között hierarchikus (összesítési) kapcsolatok is lehetnek, ezért ezek felfedezése is szükséges az OLAP-kockák kialakításához (Abelló, 2015).

Az OLAP-kocka szerkezete, illetve az ennek többnyire alapjául szolgáló csillag séma manuálisan is definiálható (Prasad, 2010). Természetesen ennél általánosabb érvényű megoldás a dimenzionális séma (fél)automatikus generálása (Abelló, 2015).

3.3.1 Fogalmi modell Prasad ötlete alapján

Prasad (2010) modelljében a nem strukturált adatokat tartalmazó adatforrásokból kinyert szövegre először szövegelemzési eljárásokat alkalmaz, majd az elemzések eredményét az adatbázis/adattárház megfelelően kialakított tábláiban helyezi el. A ténytábla és a dimenziótáblák csillag elrendezésűek, ezért alkalmasak OLAP kockák, illetve riportok létrehozására.



13. Ábra: Prasad exploratory OLAP rendszere (Prasad, 2010)

Prasad (2010) modellje két ismert szövegelemzési technikára épül: kulcsszavak kinyerése és címkézése (keyword extraction, text tagging).

A kulcsszavak kinyerésére számos algoritmus ismert. Ezek közül én a TextRank algoritmust választottam, amelyet Python-ban valósítottam meg Xu Liang leírása alapján (Liang, 2018). Az algoritmus ötlete a PageRank algoritmus, amely eredetileg weboldalak rangsorolására dolgoztak ki.

A PageRank algoritmus a weboldalakhoz egy irányított gráfot rendel. A gráf csomópontjai a weboldalak. Amennyiben egy weboldal kapcsolatban van egy másik oldallal, akkor a kapcsolatot a megfelelő csomópontok közötti él jelöli.

Az egyes csomópontokhoz súlyokat is rendelhetünk az alábbi képlet alapján:

$$S(V_i) = (1-d) + d * \sum_{j \in In(v_i)} \frac{1}{|Out(V_j)|} S(V_j) \qquad (1)$$

ahol

$S(V_i)$ - az egyes súlyok értéke,

d – csillapító tényező (dumping factor)

$In(V_i)$ – a csomóponthoz kapcsolódó csúcsok halmaza (bejövő élek)

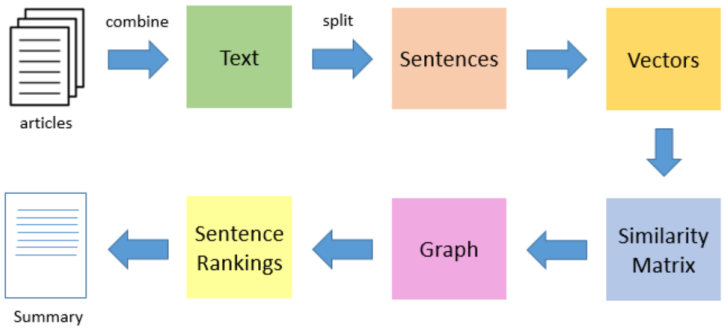
$Out(V_i)$ – a csomóponthoz kapcsolódó csúcsok halmaza (kiinduló élek).

A gráf matematikailag egy n*n-es mátrix segítségével ábrázolható, ahol n a csúcsok száma, és az i-ik sor j-ik eleme 1, ha az i-ik csúcstól megy él a j-ik csúcsig. Ha nem indul ilyen él, akkor az adott elem 0. Az adott mátrix oszlopait normalizáljuk, majd megszorozzuk az egyes csomópontok súlyait tartalmazó vektorral. Ezzel megkapjuk a PageRank kezdeti értékeit.

Ezután egy iteráció következik, melynek minden lépésében rekurzívan módosulnak a PageRank értékek. Ezek tulajdonképpen valószínűségi változók egy sorozatát alkotják (irreducibilis, aperiodikus Markov-lánc), amelyeknek a d csillapítási tényező miatt létezik határeloszlása.

A PageRank algoritmus a Google keresőmotor egyik legfontosabb eleme. A PageRank értékek felfoghatók az adott oldalra való kattintás valószínűségének is, a gráf pedig egy böngészés során bejárható oldalakat tartalmazza.

A TextRank algoritmus (Joshi, 2018) weboldalak helyett mondatokkal dolgozik. Itt egy adott weboldalról egy másikra való átmenet két mondat hasonlóságának felel meg. A hasonlósági adatok itt is egy mátrixban tárolódnak. Az algoritmust a következő ábra szemlélteti:



14. Ábra: A TextRank algoritmus működése (Joshi, 2018)

A TextRank algoritmus alkalmazása során a mondatokat egységekre (szavakra) bontjuk, és minden egységhez hozzárendeljük a szófaját, mint a neki nyelvtanilag megfelelő kategóriát, mint címkét. A módszer neve POS tagging (Part of speech tagging), amelynek eredményeképpen a szavakhoz hozzárendelt cimkék lehetnek: NOUN (főnév), PROPN (tulajdonnév), VERB (ige) stb. A kulcsszavak keresése során csak főnévre és tulajdonnévre keresünk. A kulcsszavak ezek közül azok lesznek, amelyek súlyértéke a legnagyobb.

A prototípushoz kapcsolódóan (ticketing ontológiák) nem sikerült megfelelőt találni, ezért az ontológiák előállítása egy további feladat lesz. Az ontológiák előállításának egy lehetséges módja a szövegbányászat segítségével történő fejlesztés. Ebben az esetben a szöveg előfeldolgozását az un. szózsák modell (bag of words) alkalmazása követheti (Browniee, 2017).

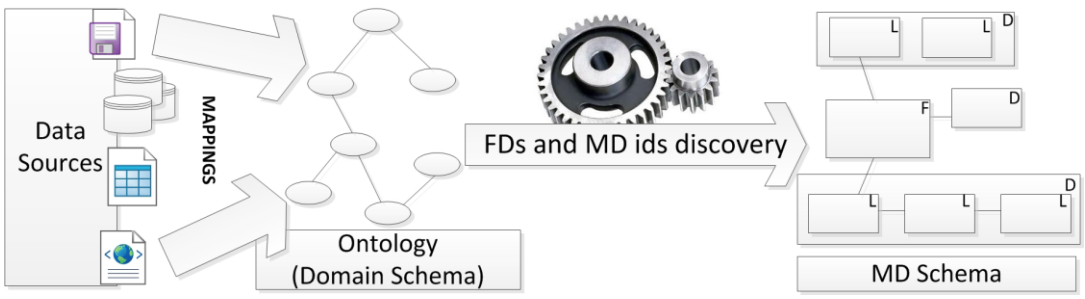
A szöveg előfeldolgozása a következő lépéseket jelenti:

- szavakra és írásjelekre tördelés (tokenizálás)
- szótövezés (azaz a ragok, toldalékok leválasztása)
- és/vagy a szavak szótári alakra hozása
- az irreleváns, jelentést nem tartalmazó szavak (stop szavak) elhagyása
- gyakorisági vizsgálat (a leggyakoribb szavak azonosítása)

A szózsák modell a szavak dokumentumon belüli előfordulási gyakorisága alapján a feldolgozott szöveghez számokat rendel (vektor), amely a további feldolgozási műveletek alapját képezi.

3.3.2 Fogalmi modell Abello ötlete alapján

Abello (2015) az exploratory OLAP megvalósítására két elvi lehetőséget is vázol, ezek közül azt a modellt választottam kiindulásként, ahol az ontológiák aktív szerepet játszanak a heterogén adatforrások adatainak közös sémára hozásában. Ennek oka, hogy egyrész az ontológiák az eszközei az üzleti fogalmak és a szakterület (domain) formális megfogalmazásának, másrészt a szemantikai megjegyzések segítségével az ontológiába bevonhatók a különböző (heterogén) adatforrások. Ez egy olyan homogén fogalmi teret eredményez, amely tartalmazza a benne lévő elemek jelentését is (Abelló, 2015).



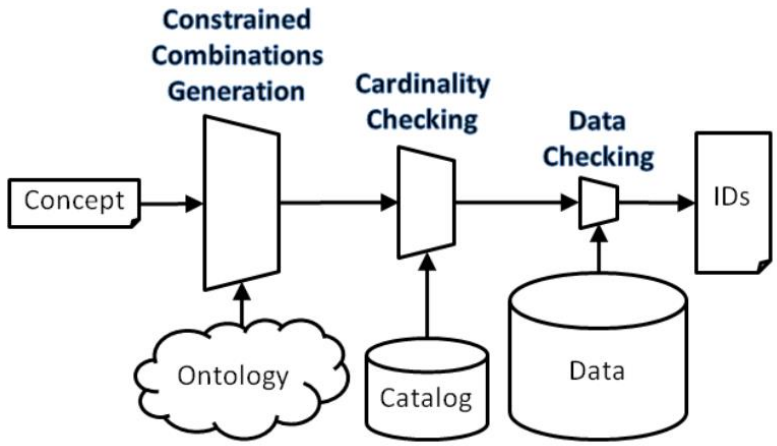
15. Ábra: Ontológiák a domain modellezésben (Abelló, 2015)

Az adatforrásokból (data sources) jövő adatok egy referencia ontológiára vannak leképezve. Az ontológiára épülő leképezések segítségével azonosíthatók a funkcionális függőségek (FD, Functional Dependency) és a multidimenzionális azonosítók (MD ids). Ezek után a tények és a dimenziók azonosításával előáll az MD séma.

Abello (2015) modellje feltételezi, hogy az ontológia már rendelkezésre áll. A gyakorlatban ez természetesen adatforrás függő, pl. webes adatok esetén sok esetben létezik az adott területet (domént) leíró ontológia rendszer.

Az MD azonosítók automatikus előállítására alkalmazott módszerek általában példány szinten oldják meg a feladatot. Ezeknek egyik hátránya, hogy ezek hajlamosak figyelmen kívül hagyni az összetett azonosítókat. A másik probléma, hogy nagy számú példány, illetve attribútum esetén az alkalmazott módszerek meglehetősen számításigényesek.

A számítási igény csökkenthető a következő ábrán látható módszerrel csökkenthető (Romero & Abello, 2012).



16. Ábra: A multidimenzionális azonosítók előállításának folyamata (Romero & Abello, 2012)

A módszer lényege, hogy a referencia ontológia segítségével előzetesen megadhatók az MD azonosító jelöltek, ezzel jelentősen csökkentve a tesztesetek számát.

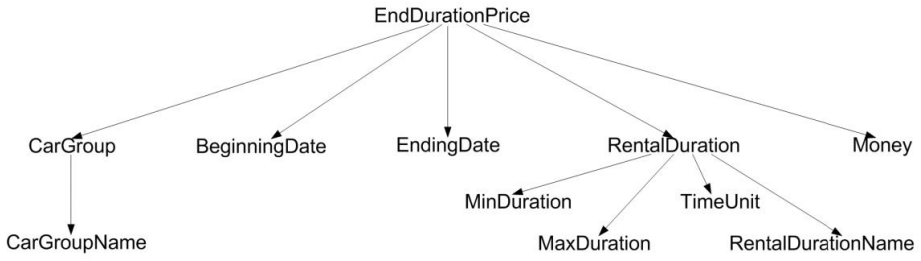
A folyamat első szakasza olyan fogalomkészleteket állít elő, amelyek a felhasználó által választott fogalom azonosítóját eredményezhetik. A fogalomkészletek csak olyan ontológia-alapú fogalmakból állhatnak, amelyeket a bemeneti fogalmat funkcionálisan meghatározzák. Ezenkívül a fogalomkészleteknek minimális halmazoknak kell lenniük, azaz nem tartalmazhatnak az azonosításhoz nem szükséges (felesleges) fogalmakat.

A második lépésben a rendszer katalógus (metaadatok) lekérdezése után összehasonlítjuk az azonosítók maximális kardinalitását (előfordulások száma) az adott fogaloméval.

A harmadik lépésben végül ellenőrizzük, hogy a kapott azonosítók segítségével elérhetők-e az adatok. Minél közelebb van egy azonosító kardinalitása az azonosítani kívánt fogaloméhoz, annál valószínűbb, hogy megfelel egy valóban értelmes azonosítónak. A lépés után előáll a lehetséges azonosítók halmaza.

Példaképpen nézzük meg a multidimenzionális azonosítók előállítását egy konkrét szakterületre vonatkozó példa (autókölcsönzés) esetén az EU-Car Rental ontológia segítségével (Frias, et al., 2003). Ez az ontológia 65 fogalmat és 170 kapcsolatot tartalmaz.

Először a szakterület minden fogalma esetén elő kell állítani a funkcionális függőségi fát (FD-fa). Pl. az EndDurationPrice (az autóbérlés végén fizetendő összeg) fogalom esetén az FD-fa felépítése a következő ábrán látható:



17. Ábra: Az EndDurationPrice fogalomhoz tartozó FD-fa (Romero & Abello, 2012)

Az ábráról leolvasható, hogy az adott fogalmat (EndDurationPrice) meghatározó fogalomkészlet mely fogalmak közül kerülhet ki. Ez összesen tíz fogalmat jelent, ezekből összesen 2^{10} részhalmaz képezhető. Ez még nem mondható soknak (nagyságrendileg ezer), de más fogalmak esetén ez az érték jóval nagyobb is lehet. Pl: DamageCost (kár esetén fizetendő összeg) fogalomnál a lehetséges esetek száma már 2^{81}, amely nagyságrendileg a tíz huszonnegyedik hatványával egyenlő.

A lehetséges esetek számát jelentősen csökkenti az a körülmény, hogy a keresett fogalomkészleteknek minimális halmazoknak kell lenniük. Ha pl. azt találjuk, hogy a bérlés kezdeti dátuma és időtartama (BeginningDate és RentalDuration) funkcionálisan meghatározza a fizetendő összeget (EndDurationPrice), akkor a keresés során az összes olyan fogalomhalmaz elhagyható, amely tartalmazza a BeginningDate és RentalDuration fogalmakat.

A vizsgálandó esetek számát az is csökkenti, hogy egy adott fogalomhalmazon belül sem lehetnek funkcionális függőségek. Pl. ilyen függőségi kapcsolat van a RentalDuration és a MinDuration (minimális bérleti időtartam) fogalmak között is, ezért a kettő együtt nem fordulhat elő.

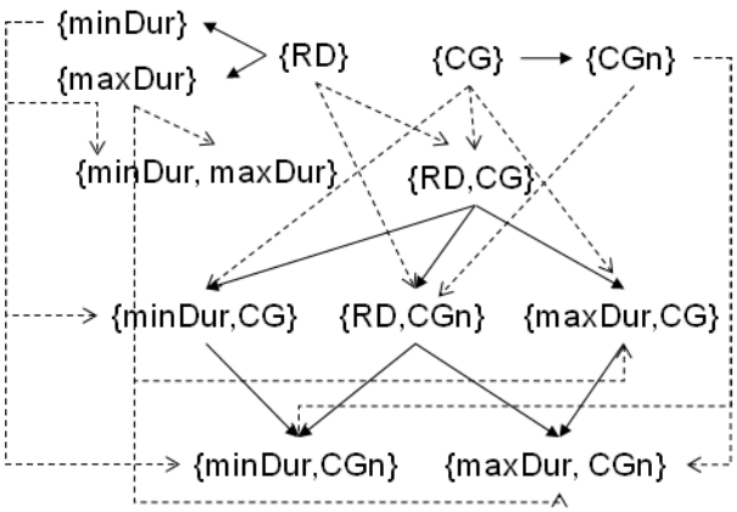
Végül további csökkentési lehetőség adódik abból az egyszerű következtetésből, hogy ha egy fogalomhalmaz nem azonosítja egyértelműen az eredeti fogalmat, akkor az összes olyan fogalomhalmaz is kizárható a keresésből, amely az adott fogalomhalmaztól funkcionálisan függő fogalmakat tartalmaz. Pl.: ha feltételezzük, hogy a bérlet időtartama és a befejezés dátuma (RentalDuration és EndingDate) nem határozza meg a fizetendő összeget, akkor a bérlet időtartama és a minimális bérleti időtartam (MinDuration) sem, mivel a MinDuration és a RentalDuration között függőségi kapcsolat van.

Az MD azonosítókat előállító algoritmus a következő elven működik:

- Input paraméterek: az a fogalom, amelyhez az MD azonosítót keressük, valamint a fogalomhoz tartozó FD-fa. Példánkban az input paraméterek az EndDurationPrice fogalom és annak FD-fája.
- Definiáljunk három halmazt a következőképpen:

- L – Lehetséges ID-k. Ez az algoritmus i-ik lépésében a fogalmak olyan i elemű kombinációit jelenti, amelyek eleget tesznek a számítási igényt csökkentő feltételeknek. Kezdetben az L halmaz álljon az FD-fa gyökér elemeinek gyermekeiből, azaz L={{CarGroup}, {BeginningDate}, {EndingDate}, {RentalDuration}, {Money}} (ld. 17. Ábra).
- C – nem megfelelő ID jelöltek. Az L halmaz azon i elemű kombinációi, amelyek az adatok lekérdezésekor kizárunk az MD azonosítók közül.
- I – lehetséges MD azonosítók. Az L halmaz azon i elemű kombinációi, amelyek az adatok lekérdezése után MD azonosítók lehetnek.

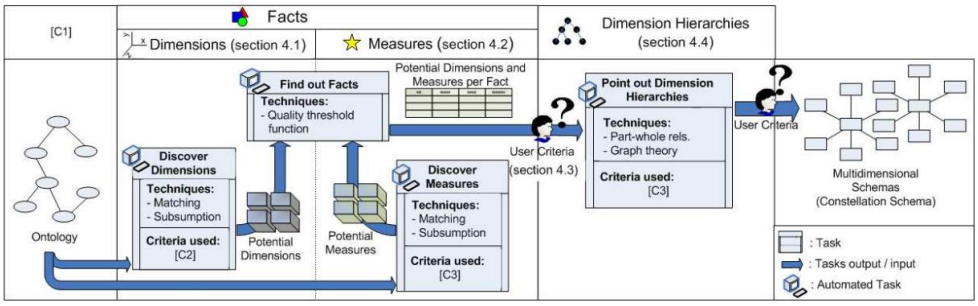
- Ezután hajtsunk végre egy ciklust, amíg az L halmaz üres nem lesz. A ciklus során egy keresési fát építünk fel, amelynek egy részletét a 18. Ábra mutatja. Az ábrán normál nyilak mutatják a függőségi kapcsolatokat, szaggatott nyilak pedig a részhalmaz kapcsolatokat.
  - A cikluson belül képezzük a lehetséges i-elemű kombinációkat az L halmazból. Minden egyes kombináció esetén először a C halmaz legyen üres, majd
    - Ha egy kombináció megfelelő, akkor adjuk hozzá az I halmazhoz. Ezután, ha a kombinációnak vannak leszármazottai, akkor adjuk hozzá közülük azokat az L halmazhoz, amelyek elemszáma azonos, és a kombináció egy fogalma funkcionálisan függ a szülő kombináció egy fogalmától. Pl. a {RentalDuration, CarGroup} esetén ilyen leszármazott a {RentalDuration, CarGroupName}, mivel a CarGroupName és a CarGroup között funkcionális függés van.
    - Ha egy kombináció nem megfelelő, akkor adjuk hozzá a C halmazhoz.
    - Végül távolítsuk el a vizsgált kombinációt az L halmazból.
  - A ciklus végén az L halmaz álljon a C halmaz olyan kombinációiból, amelyek elemszáma 1-gyel nagyobb, mint a korábbi elemszám. Pl. ha a C halmaz {RentalDuration}, akkor L={RentalDuration, CarGroup}, {RentalDuration, CarGroupName}}



18. Ábra: Az EndDurationPrice MD azonosítójának keresése (Romero & Abello, 2012)

Az MD azonosító algoritmusban egy kombináció megfelelőségét az adatok lekérdezésével ellenőrizhetjük. A lekérdezés módja függ az adatok tárolási módjától, pl. relációs adatbázis esetén SQL-lekérdezéseket lehet alkalmazni. A megfelelőség feltétele, hogy az azonosítandó fogalom előfordulásainak száma (számosság, kardinalitás) kisebb vagy egyenlő legyen, mint a kombinációban lévő fogalmak kardinalitásainak szorzata.

A tények és dimenziók előállításának lépéseit a következő ábra szemlélteti (Abello & Romero, 2010).



19. Ábra: A tények és dimenziók előállításának folyamata (Abello & Romero, 2010)

Az ábrán C1, C2 és C3 jelöli azokat a kritériumokat, amelyeket az egyes lépéseknél figyelembe veszünk:

- C1: Multidimenzionális modell alkalmazása (tények és dimenziók).
- C2: Egy ténynek az egyes elemzési dimenziókhoz több az egyhez kapcsolattal kell kapcsolódnia.

- C3: Összesítési integritás kényszer. Az adatösszesítéseknek korrektnek kell lenniük. Ez a következő három feltétel segítségével biztosítható:
    - Diszjunktivitás. Az összesítendő objektumok halmazainak diszjunktnak kell lenniük.
    - Teljesség. A részhalmazok uniójának a teljes halmaznak kell lennie.
    - Kompatibilitás. A dimenziónak, az összesítendő mértéknek és az összesítő függvénynek egymással összhangban kell lennie. Pl. nem összesíthetjük a készlet nagyságát (mint mértéket) az idő dimenzió mentén.

A dimenziók felfedezését elősegíti az a tény, hogy azok rendszerint nem numerikus adatok, és lehetséges elemeiknek száma általában alacsony (Abello & Romero, 2010). Emiatt pl. a száz feletti elemszámú fogalmak többnyire nem dimenziójelöltek.

A tények előállításához a lehetséges dimenzionális fogalmak mellett szükséges a mértékek felfedezése.

A tényjelöltek minősítésére definiálható egy függvény, amely minden egyes ontológia fogalomhoz egy számértéket rendel. Ez a függvény a következő módon fejezhető ki:

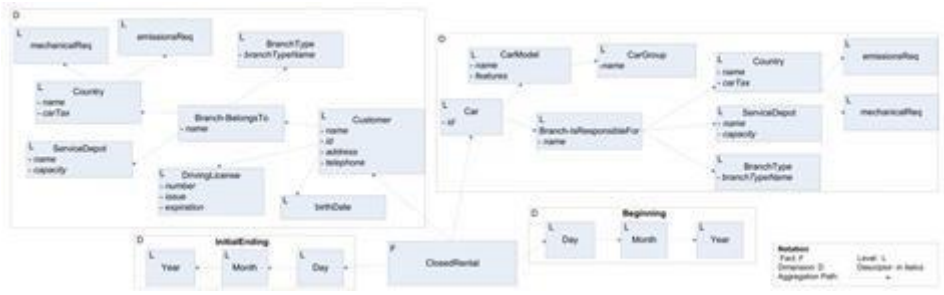
$$f = D + 2M \qquad (2)$$

ahol D az adott fogalomhoz társítható dimenzionális fogalmak száma, M pedig a lehetséges mértékek száma. Például a ClosedRental (lezárt kölcsönzések) fogalomhoz 74 dimenzionális fogalom és 4 mérték tartozik, így ez esetben a függvény értéke 74+2*4 = 82 lesz.

Amely fogalmak esetén ez a függvény a legnagyobb értékeket veszi fel, azok lesznek a konkrét tényjelöltek. Esetünkben ezek a következők: LateReturn, DamageCost, Prepared, AssignedCar, ClosedRental, EarlyReturn. Ezen jelöltek közül a felhasználó választja ki a végleges tényeket. Legyen például kiválasztva a ClosedRental fogalom. Erre lefuttatva az MD azonosító keresésének algoritmusát a következő találatok adódnak: {Beginning, Customer}, {Beginning, Car}, {InitialEnding, Customer}, {InitialEnding, Car}, {AgreedEnding, Customer}, {AgreedEnding, Car}.

A dimenzió hierarchiák meghatározásához ezután azonosítani kell a releváns összesítési utakat, amelyek tipikus rész-egész összefüggéseket keresnek. Tegyük fel, hogy a felhasználó az előbb felsorolt MD azonosítók közül a következőket választotta: {Beginning, Customer}, {Beginning, Car}, {InitialEnding, Customer}, {InitialEnding, Car}. Ezek alapján négy dimenzió határozható meg: Customer, Car, Beginning, és InitialEndig.

A végeredményként kapott dimenzionális sémát a következő ábra szemlélteti.



20. ábra: A ClosedRental tény és a hozzá tartozó dimenziók (Romero & Abello, 2012)

Az ábrán a dimenziókat D, a tényt F, az egyes szinteket L, az összegzési utakat pedig nyilak jelölik. Ezek egy része természetesen adódik, pl. Év-hónap-nap, más részük a felhasználó által definiált. Abello (2015) exploratory OLAP modellje esetén az egyik fő problémát a tranzitív funkcionális függőségek kezelése jelenti, amelyek nem, vagy csak bizonyos mértékben következtethetők ki ontológia érvelésekkel. Neumayr javasolt erre egy megoldást Datalog ontológia metamodell használatával (Neumayr, et al., 2012).

3.4 A prototípus

A kutatásom során szeretnék létrehozni egy-egy prototípust is, amely segítségével Abelló, illetve Prasad (2010) modellekre épülő fogalmi modelljeim szakterületi verziói megvalósíthatók.

A prototípusokhoz kapcsolódó adattárház egy szoftver fejlesztő céghez köthető, amely különböző IT szolgáltatásokat nyújt az ügyfelei számára. A leggyakoribb szolgáltatások a szoftver fejlesztés, üzemeltetés, hibaelhárítás és támogatás. A szolgáltatásokkal kapcsolatban keletkezett adatok kezelésére a cég speciális, un. servicedesk alkalmazásokat (ticketing rendszerek) használ.

Egy ilyen ticketing rendszernek négy fő funkciója van. Az első új ticket létrehozása. Ez lehetővé teszi a felhasználó számára, hogy megadja a ticket (hibajegy) típusát (igény, változás vagy incidens), részletes leírás csatolását és sok más információt, pl. prioritás vagy a tickethez rendelt személy megadását.

A második funkció a logolás. Ez annak tárolását jelenti, hogy melyik elemző mennyi időt töltött a ticket-hez kapcsolódó feladatok megoldásával.

A harmadik a ticketek állapotváltozásainak követése. Amikor egy ticket létrejön, akkor a kezdeti státusza nyitott (open). Ezután – az esettől függően – számos más állapotba is kerület, pl: waiting vagy in progress. A folyamat végén a ticket lezárásra kerül (closed).

Az utolsó funkció az SLA-k (Service Level Agreements) kezelése. Ebből a szempontból a ticket-ek SLA státusza lehet OK, Warned vagy Violated.

Az ügyfelek nagy száma és eltérő igényei miatt a cég több ilyen ticketing rendszert használ. Ezek egymástól akár fizikailag is el vannak különítve, így a rendszerek közötti adatcsere több esetben csak közvetett úton valósítható meg (pl: export a forrásrendszerből Excel-be →továbbítás az adattárház felé → import Excel-ből). A nehézséget az is növeli, hogy a forrásrendszerek nem teljesen egyformák. Ez kisebb részben az eltérő verziójú programok miatt van, nagyobb részben amiatt, hogy ezek a rendszerek tartalmazhatnak saját fejlesztésből származó adatokat is.

A hibajegyek kezelése során rengeteg szöveges adat keletkezik. Például csak a probléma leírása akár több oldalas is lehet. A ticketek nagy száma miatt az ilyen szöveges adatok teljes körű manuális feldolgozása szinte lehetetlen. Az exploratory OLAP megvalósításával és alkalmazásával viszont elérhető, hogy a nem strukturált (szöveges) adatok feldolgozása automatikus legyen. Az így kapott plusz információk fontos szerepet játszhatnak a vezetői döntéstámogatásban.

3.5 Az eredmények értékelése

Az exploratory OLAP rendszerek validálására a gyakorlatban nincs egységesen elfogadott és alkalmazott módszer, ezért a validálás során a hagyományos OLAP rendszerek értékelése jelenti a kiindulópontot. Mivel az exploratory OLAP rendszerek szemantikus technológiákat (többnyire ontológiákat) is használnak, ezért validáláskor ezeket is értékelni kell.

Ennek megfelelően az exploratory OLAP kutatás eredményeinek validálása, értékelése során a két legfontosabb értékelési szempont a következő:

- Hogyan értékelhetők a forrásrendszerekből kinyert ontológiák? (amennyiben az exploratory OLAP rendszer ontológiákat használ)
- Hogyan értékelhetők az ontológia-MD megfeleltetésből vagy más módon kapott multidimenzionális sémák?

Az első kérdés megválaszolására, azaz az ontológiák értékelésére a szakirodalom több módszert is megemlít. Ezek a módszerek a következő módon csoportosíthatók: (Haghighi, et al., 2013):

- „Gold standard" értékelés. Ennek lényege, hogy összevetjük az ontológiával kapcsolatos előzetes feltételezéseinket a kapott megoldással.
- Adatvezérelt értékelés. Ez az alapvető adatokkal leírt célmegoldással (pl. szemantikus web szolgáltatás) való összevetést jelenti.
- Humán értékelés. Ez egy előre meghatározott kritériumrendszer alapján történő értékelést jelent. Például Gómez Pérez definiálta az un. ontometriát, amely nyelvből, tartalomból, módszertanból és költségből álló kritériumokat tartalmaz.
- Alkalmazás alapú értékelés. Ennek során megpróbáljuk az ontológiát az adott célra felhasználni, és a kapott tapasztalatok alapján értékelünk.
- Feladat alapú értékelés. Az előző módszerhez hasonlóan itt is az ontológiára épülő alkalmazáson van a hangsúly. A különbség, hogy itt a létrejött megoldás teljesítményét előre definiált feladatok végrehajtása alapján értékeljük.
- Kritérium alapú értékelés. A humán értékeléshez hasonló, de annál rugalmasabb megközelítés. A kritériumok itt kevésbé formálisak.

A konkrét ontológia értékelő módszer a 4.4 alfejezetben kerül megnevezésre.

A második kérdés megválaszolásához a következő értékelési szempontokat kell figyelembe venni (Prat & Cherfi, 2003):

- Olvashatóság. Mennyire könnyen olvasható a séma? Ennek objektív mérésére két kritérium használható:
    - Minimalitás. Batini, et al., (1992) szerint „Egy séma minimális, ha a követelmények minden aspektusa csak egyszer jelenik meg". Másképpen kifejezve, a minimális sémából bármely attribútumot

törölve információvesztés lép fel. A minimalitás többféle mérőszámmal is mérhető, ezek közül a legfontosabb a rendundanciamentesség (NR). Ez a következő összefüggés alapján határozható meg:

$$NR = \frac{\sum_S w_i * N(C_i) - w_i * NR(C_i)}{\sum_S w_i * N(C_i)} \quad (3)$$

ahol C_i az S séma ténytáblája, dimenziója vagy dimenzióhierarchia szintje, $N(C_i)$ a C_i típusú elemek száma, $NR(C_i)$ a C_i típusú redundáns elemek száma a sémában, w_i pedig a C_i-hez rendelt súly. A rendundanciamentesség értéke 1, ha a sémában nincsen redundancia, mivel ilyenkor az NR(Ci) értéke minden esetben 0 lesz, így a képletben a számláló és a nevező értéke megegyezik. Rendundancia esetén nyilván 0 és egy közötti számot kapunk eredményül.

- Kicsinyítési-nagyítási képesség. Ez azt mutatja meg, hogy mennyire tekinthető meg a séma különböző szintű részletezettségben (granularitásban). Ehhez konkrét mérőszám (Z) a következő képlet alapján rendelhető:

$$Z = (\sum_{i=1}^{N} 1 - \frac{1}{D(i)})/N \quad (4)$$

ahol N a ténytáblák száma, és D(i) az i-ik ténytáblához tartozó dimenziók maximális mélysége. Ennek értéke egyszintű dimenzióhierarchiák esetény nyilván 0, mivel ilyenkor D(i) értéke mindenhol 1 lesz. Ez megfelel annak az elvárásnak, hogy ha nincsenek legalább kétszintű dimenzióhierarchiák, akkor az adatokat csak egyféle részletezettségben látjuk. Azaz, ilyenkor nem lehet sem kicsinyíteni, sem nagyítani a részletezettségen.

- Kifejezőképesség (szemléletesség). Batini, et al., (1992) szerint „Egy sémát kifejezőnek mondunk, ha természetes módon reprezentálja a felhasználói igényeket, és további magyarázat nélkül könnyen megérthető”. A kifejezőképesség tulajdonképpen a sémára épülő elemzési lehetőségek változatosságát minősíti. Egy séma kifejezőképessége elsősorban a ténytáblákon múlik. Emiatt célszerű a ténytáblákra definiálni mérőszámokat, és utána venni ezek átlagát. A mérőszámok függhetnek a ténytáblában lévő mértékek számától, a ténytáblához csatlakozó dimenziók számától, a ténytáblához csatlakozó dimenzió hierarchia mélységétől vagy a mértékeken alkalmazható összegzési (aggregálási) lehetőségektől. Ilyen mérőszám pl. egy ténytábla lokális gazdagsága (LR):

$$LR = \frac{N(F)}{N(S)} \tag{5}$$

  ahol N(F) az F ténytábla mértékeinek száma, és N(S) a sémában lévő összes mérték száma. Nyilvánvaló, hogy egyetlen, legalább egy mértéket tartalmazó ténytábla esetén ennek értéke mindig 1 lesz.

- Egyszerűség. Egy multidimenzionális séma annál egyszerűbb, minél kevesebb ténytáblát és dimenziót tartalmaz. Az egyszerűség (S) objektív mérőszáma a következőképpen definiálható:

$$S = \frac{N(F) + N(DL)}{N(F) + N(DL) + N(L)} \tag{6}$$

  ahol N(F) a ténytáblák száma a, N(DL) a dimenzióhierarchia szintek száma, N(L) pedig a kapcsolatok száma a sémában. Látható, hogy az egyszerűség értéke 1-nél kisebb szám (az 1-et csak abban az esetben érhetné el, ha a sémában egyáltalán nem lennének kapcsolatok).

- Helyesség. Ez a tények és dimenziók korrekt definícióját értékeli a sémában. A korrektség azt jelenti, hogy a definíció megfelel egy vagy több ellenőrzési szempontnak. A helyesség mérőszáma (C) a következő összefüggéssel definiálható:

$$C = \frac{\sum V(C_i) - E(C_i)}{\sum V(C_i)} \tag{7}$$

  ahol C_i a multidimenzionális séma egy eleme, $V(C_i)$ a C_i-n elvégzendő ellenőrzési szempontok száma, $E(C_i)$ pedig a C_i ellenőrzésekor kapott hibák száma. Hibamentes séma esetén a helyesség értéke 1.

Az OLAP-rendszerek értékelésekor egyéb szempontokat is mérlegelhetünk, pl:

- Mennyire egyszerű lekérdezni a sémából?
- A lekérdezések mennyire gyorsak?
- Az aggregációk milyen számítási kapacitást igényelnek?

Fontos megjegyezni, hogy az OLAP-rendszerek általában strukturált adatokra épülnek. Ezzel szemben viszont az exploratory OLAP rendszerek nem strukturált adatokkal (is) dolgoznak. Ezt a tényezőt figyelembe kell venni az OLAP-hoz köthető kérdések megválaszolásánál.

Az exploratory OLAP modellek validálásához nélkülözhetetlen egy-egy működő prototípus elkészítése. Természetesen a prototípusok alapján levont következtetésekből még nem lehet általánosítani, viszont az exploratory OLAP modell más területen is alkalmazható, így a később kapott eredmények összevethetők a kutatási terv megállapításaival. A kutatásom során Prasad (2010), illetve Abello (2015) ötletét felhasználva készítettem exploratory OLAP prototípusokat, amelyeket az 5.1, illetve 5.4 alfejezetekben mutatok be. Ugyanitt – a prototípusokat is felhasználva – értékelem a vonatkozó fogalmi modelleket is.

3.6 Kapcsolódó munkák

Abello (2015) ontológiákat használ az exploratory OLAP rendszer létrehozása során, amellyel tulajdonképpen egy szemantikus réteget hoz létre az adattárházban. A szemantikus adattárház létrehozásának ötlete nem ismeretlen a szakirodalomban. Ebben az alfejezetben egy ilyen keretrendszer kerül bemutatásra (Nebot, et al., 2009), amely Abello (2015) -hoz hasonlóan szintén ontológiákat használ. Bár konkrét prototípus itt sem készül, de a cikk a benne lévő hasznos megállapításokkal, és egy konkrét használati eset leírásával megfelelő hátteret adhat az exploratory OLAP prototípus elkészítéséhez.

Az ontológiák alkalmazásának fő oka, hogy közös terminológiát és logikát állítson fel az adott területet (tartományt, domént) érintő fogalmak számára. Manapság sok alkalmazás (pl. orvosi alkalmazások) metaadatokat és szemantikus annotációkat (kommentárok, jegyzetek) is csatol az általa előállított információkhoz. Például orvosi terület esetén ilyen lehet a röntgen kép vagy a labor lelet. A szemantikus annotációk különösen hasznosak a strukturálatlan, félig strukturált és szöveges adatok leírására, amelyet a jelenlegi adatbázis rendszerek nem tudnak megfelelően kezelni.

A szemantikus adattárház ezek alapján webes erőforrásokat, ontológiákat és szemantikus annotációkat tartalmaz.

A keretrendszernek a következő követelményeket kell kielégítenie:

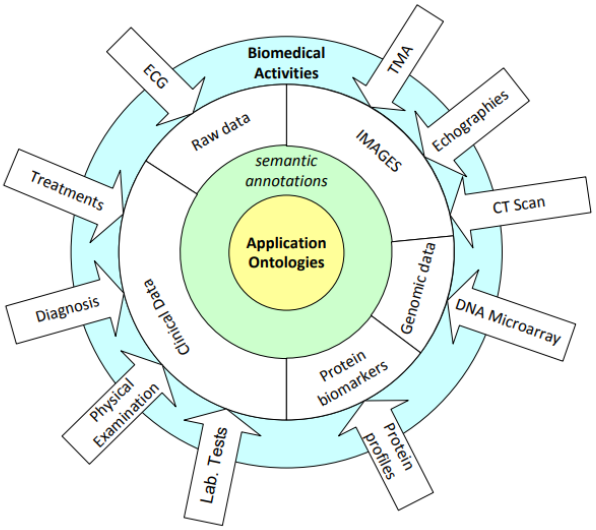
- Az elemzéshez szükséges fogalmak kiválasztása különböző ontológiákból (multiontológia tervezés)
- Skálázhatóság
- Formálisan megalapozott megközelítés

A multiontológia tervezésre azért van szükség, mert egy komplex rendszer esetén sok szemantikai adat keletkezik, amelyek gyakran egynél több (tartományi) ontológiához köthetők. Mivel a tartományi ontológiák általában nagy méretűek, ezért ezek közül csak a szükséges modulokat vagy töredékeket szabad felhasználni a tervezés során. A tervezési folyamatnak a szemantikus web számára széles körben elfogadott formalizmusokra kell támaszkodnia, ilyen például a leíró logika.

A keretrendszer képes – többek között - a következő feladatok megvalósítására:

- Az ontológiák multidimenzionális sémába való integrálása. A fogalmak és tulajdonságok integrálásával a multidimenzionális elemzéshez szükséges tények, dimenziók, mértékek és hierarchiák automatikusan létrehozhatók.
- A szemantikus adattárházban tárolt adatok felhasználásával OLAP-kocka automatikus elkészítése. Ez lehetővé teszi majd az adatok elemzését hagyományos OLAP-operátorok segítségével.

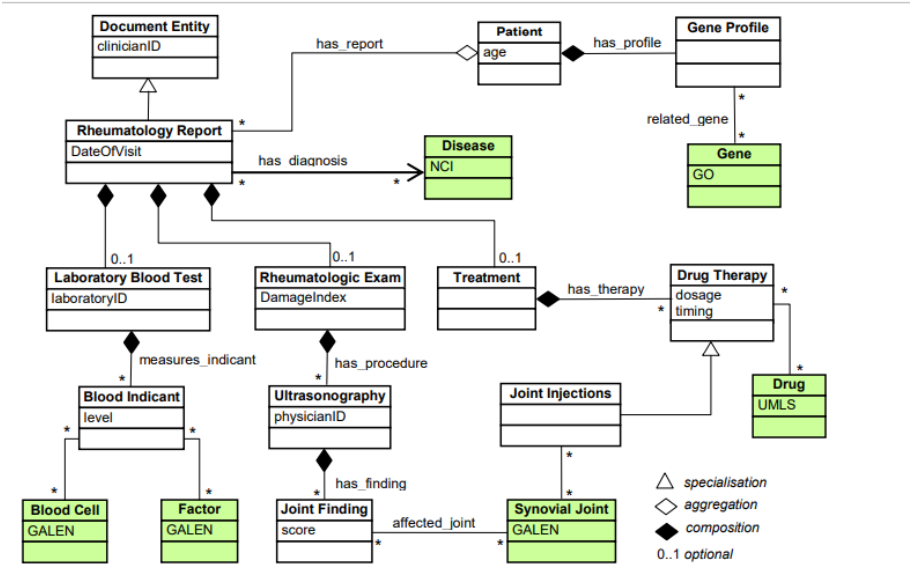
A keretrendszer egy alkalmazási lehetősége az orvosbiológia területe, ahol sokféle szemantikailag annotált adat keletkezik. Ezt a következő ábra szemlélteti:



21. Ábra: Szemantikus annotációk generálása az orvosbiológia területén (Nebot, et al., 2009)

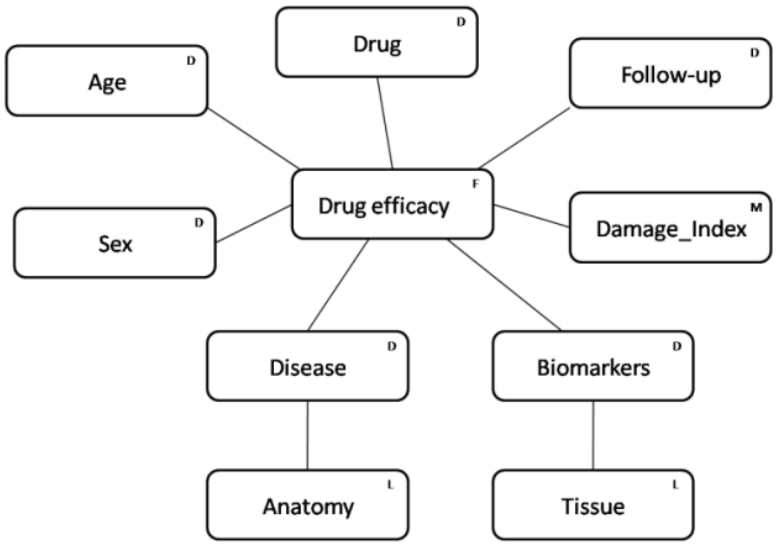
A különböző orvosi tevékenységek (kezelések, vizsgálatok) során keletkező strukturálatlan adatokat megfelelő módon annotálva kell a szemantikus adattárházban elhelyezni. Az annotációk legegyszerűbben XML vagy RDF formátum segítségével tárolhatók.

Az annotációk leírása ontológiák segítségével történik. A következő ábra a reumatológia terület ontológiáinak egy részét mutatja. Látható, hogy a páciensek különböző reumatológiai jelentéseket kaphatnak, amelyek az elvégzett vizsgálatok eredményeit, illetve a javasolt kezeléseket írják le.



22. Ábra: Az alkalmazás ontológiák egy részlete (reumatológia terület) (Nebot, et al., 2009)

Az elemzésekben részt vevő ontológiák segítségével a keretrendszer a következő dimenziókat állítja elő: a páciens kora és neme, a betegség típusa, a biomarker (biológiai jelzőanyag, amely a betegség jelenlétét, súlyosságát jelzi), a károsodási index, és a kezelés, illetve az utólagos vizsgálatok során alkalmazott gyógyszerek.

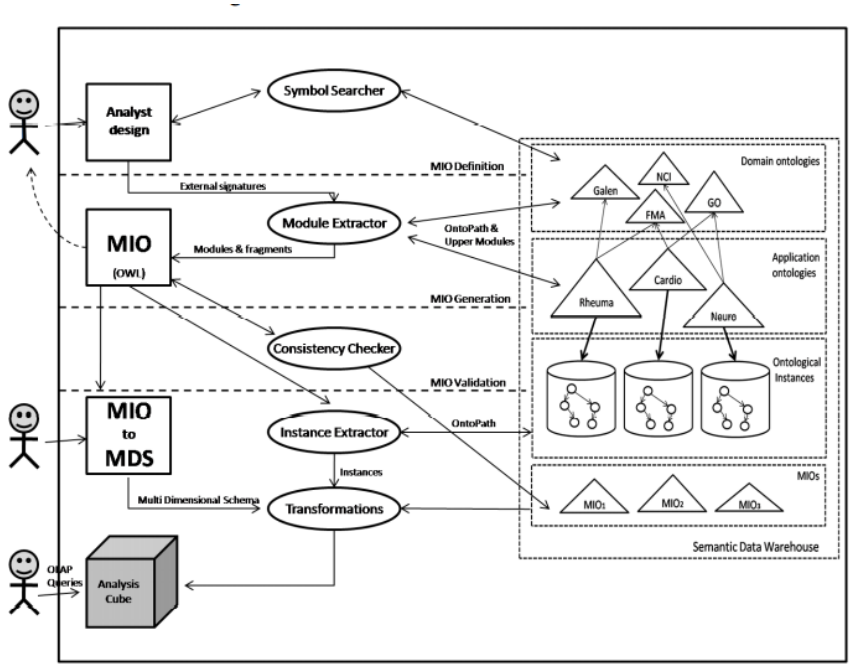


23. ábra: Dimenziók (D), tények (F) és mértékek (M) a reumatológiai elemzésekhez (Nebot, et al., 2009)

A keretrendszer logikailag három részből áll:

- Megfelelő módon elrendezett adatok és kapcsolataik (tartományi ontológiák, alkalmazás ontológiák, ontológia példányok, multidimenzionálisan integrált ontológiák)
- A többdimenziós kocka elkészítéséhez szükséges eszközök (pl. szimbólum kereső)
- A multidimenzionális sémába integrálható ontológiák (MIO) tervezéséhez szükséges folyamatok (pl. MIO-tervezés)

A keretrendszer működését a következő ábra szemlélteti:



24. Ábra: A szemantikus adattárházat megvalósító keretrendszer (Nebot, et al., 2009)

Az ábráról leolvasható, hogy az alkalmazás ontológiák különböző tartományi ontológiákat használhatnak fel hasonló fogalmak jelölésére. Emiatt szükség van olyan leképezésekre, amelyek összeegyeztetik a fogalmi átfedéseket.

Az OLAP-kocka létrehozásának első lépése az MIO-definíció. Ennek során az elemző (Analyst) egy szimbólum keresőt (Symbol Searcher) alkalmaz, amely az elemzés során használandó szimbólumokat (fogalmak, tulajdonságok) lekéri a szemantikus adattárház tartományi ontológiáitól. A lépés célja az elemzéshez szükséges dimenziók és mértékek meghatározása az ontológiák segítségével.

Ez a gyakorlatban öt egymás utáni tevékenység végrehajtását jelenti: az elemzés témájának kiválasztása, az elemzési dimenziók meghatározása, a mértékek kiválasztása, az összesítési (roll-up) kapcsolatok definiálása, végül az elemzendő példányok specifikálása.

A második lépés az MIO-k létrehozása. Ezeket a modul kivonatoló (Module Extractor) automatikusan hozza létre a következő elemekkel:

- A külső szimbólumokkal történő következtetések levonásához szükséges ismeretekkel rendelkező modulok összessége. Például egy betegség diagnózisának meghatározásához szükséges ontológiák tartalmazhatnak olyan axiómákat, amelyek kellenek az érvelések során (külső axiómák). Ilyenkor a megfelelő modul kivonatolása szükséges.
- A legfelső szintű ontológia - amely az előző modulok (UM – Upper Modules) egyesítésének eredménye -, valamint az ontológia leképezésekből származó axiómák halmaza (tematikus axiómák).
- Lokális axiómák, amelyek az elemző által megadott MIO-definíciókból származnak.

Az MIO-k végül a lokális axiómák, a tematikus axiómák és a külső axiómák uniójaként állnak elő.

A harmadik lépés az MIO-validálás, amely a következetesség ellenőrző (Consistency Checker) eszköz segítségével történik. A validálás két szinten történik: a séma, illetve a példány szintjén. Ha az eszköz következetlenséget tapasztal, akkor a felhasználó megváltoztathatja az MIO-axiómákat abból a célból, hogy az OLAP-kocka érvényes (valid) legyen. A validálást megnehezíti, hogy néhány tulajdonság (pl. összegezhetőség) csak a kocka létrehozása után ellenőrizhető.

Az utolsó lépés az elemzés elkészítése. Ebben a szakaszban az OLAP-kocka tényeinek és dimenzióinak előállításához szükséges példányok állnak elő a példány kivonatoló (Instance Extractor) által. Ezután következik az OWL to MDS folyamat, amely az ontológiákat elemzésre alkalmas formára hozza (Multi Dimensional Schema). Végül a tények generálása következik, amely a kivonatolt példányokra alkalmazott transzformációk (Transformations) segítségével valósul meg. A folyamat eredményeként létrejön az elemzéshez szükséges OLAP-kocka.

4 KUTATÁSI MÓDSZERTAN

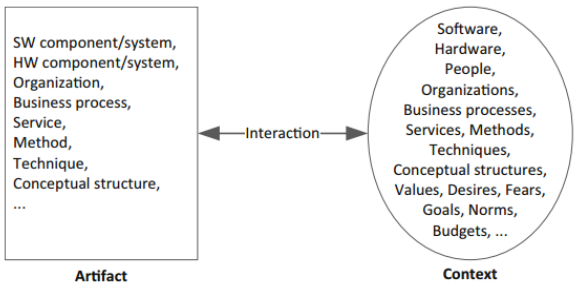
Ebben a fejezetben a kutatás során alkalmazandó módszertan kerül ismertetésre. Ide tartozik az információs rendszerek tervezése és a szoftverfejlesztés során használt design science, az adatgyűjtés és adatelemzés, az ontológia fejlesztési módszertan, valamint a prototípus készítése során alkalmazott fejlesztési módszertan.

4.1 Design Science

A design science („tervezéstudomány") (Vaishnavi, et al., 2004) olyan összegző és elemző technikák, nézőpontok halmaza, amelyek IT-kutatások során hatékonyan alkalmazhatók. A cél az adott probléma megértése az információs rendszerek nézőpontjából. Ez rendszerint két alapvető tevékenység segítségével történik:

- Új ismeretek szerzése egy innovatív alkotás (artifaktum) létrehozásán keresztül
- Az artifaktum használata során kapott visszajelzések elemzése

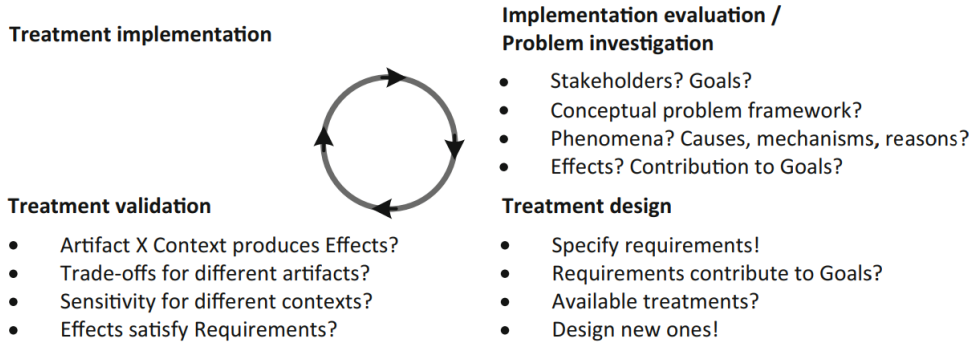
Az artifaktum lehet például algoritmus, nyelv, rendszertervezési módszer stb. Kutatásom esetén az artifaktum az exploratory OLAP prototípus, illetve a ticketing ontológia. Az artifaktum tervezése és vizsgálata mindig egy adott összefüggésben, környezetben (kontextus) történik. A kontextus esetünkben a ticketing rendszer.



25. Ábra: Az artifaktum és a kontextus (Wieringa, 2014)

Egy design science projekt mindig iteratív tevékenység, amely két fő tevékenysége a tervezés és a vizsgálat. A tervezési tevékenység három részre osztható. Ezek a következők: a probléma vizsgálata, a probléma kezelésének megtervezése és a kezelés validálása. Ezen három résztevékenység ismétlését tervezési ciklusnak (design cycle) nevezzük.

A tervezési ciklus egy nagyobb ciklus része, amelynek során a tervezési ciklus eredményét (a validált probléma kezelést) átadják a felhasználóknak, akik ezt használják, illetve értékelik. Ezt a nagyobb ciklust fejlesztési ciklusnak (engineering cycle) nevezzük (Wieringa, 2014). Ennek működése a következő ábrán látható.



26. Ábra: A fejlesztési ciklus (Wieringa, 2014)

A ciklus a következő lépésekből áll:

- A probléma vizsgálata. Milyen jelenségeken kell javítani? (Pl.: hogyan lehet növelni a szöveges adatok kezelésének hatékonyságát az adattárházakban)
- A probléma kezelő eljárás tervezése. Egy vagy több artifaktum tervezése a probléma kezelésére. (Pl.: exploratory OLAP prototípus tervezés)
- A probléma kezelő eljárás validálása. Ezek a tervek megoldják a problémát?
- A probléma kezelő eljárás implementálása. A probléma kezelése az egyik artifaktummal. (Pl.: egy exploratory OLAP prototípus megvalósítása)
- A megvalósított eljárás értékelése. Milyen sikeres a probléma kezelése? (Pl.: az elkészült prototípus mennyire felel meg a követelményeknek?) Ez a lépés sok esetben egy új iteráció kezdetét jelenti.

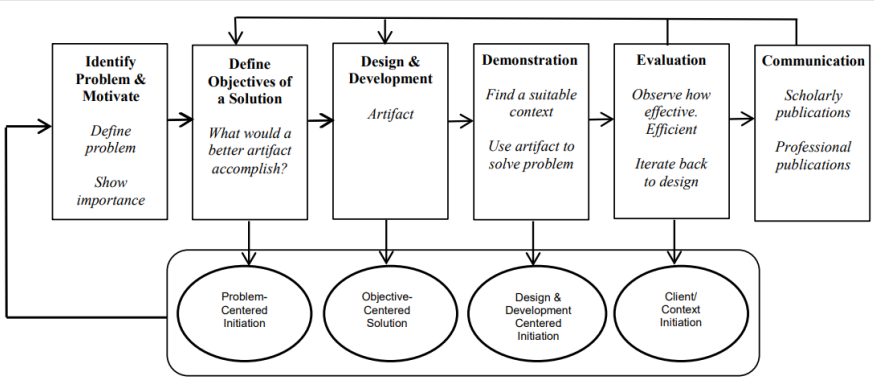
Ezt a ciklust addig kell ismételni, amíg el nem jutunk a kívánt eredményig.

A tervezési és fejlesztési ciklusok tartalmazzák a végrehajtandó feladatokat, de nem írják elő a tervezés vagy fejlesztés folyamatát (Pl. nem szerepel bennük a lépések szigorú sorrendje). Ugyanígy arról sem informálnak, hogy hogyan lehet kezelni ezeket a folyamatokat.

A tervezési ciklus lépéseinek sorrendjét többféleképpen is lehet rögzíteni, néhány lehetőség ezek közül:

- Vízesés-jellegű fejlesztési folyamat esetén a tervezési ciklus lépései szekvenciálisan hajtódnak végre, visszalépés nélkül. Erre akkor van lehetőség, ha a problémát annak kivizsgálása után teljes mértékben sikerül megérteni, és a probléma kezelés validálása vagy a kezelés megvalósítása nem ad okot a probléma kezelés újratervezésére.
- Agilis fejlesztési folyamat esetén a cikluson sok egymást követő áthaladás történik. Minden egyes menetben a kezelt problémának elég kicsinek kell lennie ahhoz, hogy az adott menet egy-két hét alatt teljesíthető legyen.
- Egy másik iteratív megoldási lehetőség, hogy kezdetben kiindulunk egy leegyszerűsített problémából, amelyet ideális körülmények között kezelünk. Ezután minden lépésben szigorítunk a feltételeken úgy, hogy egyre inkább közelítsünk a valós körülménykehez. Ezáltal minden iterációs lépésben a problémát egy adott idealizálási szinten oldjuk meg.
- Lehetséges megoldás az is, hogy először csak a kisebb ciklust (a tervezési ciklust) hajtjuk végre iteratívan. Ezekben a lépésekben csak a probléma leírása, annak lehetséges kezelése, és a kezelés konceptuális validálása történik, implementálás nélkül. A tervezési ciklus sikeres befejezése után következhet a fejlesztési ciklus többszöri végrehajtása. Egy adott tervezési ciklust még a korább ciklus befejeződése előtt is el lehet indítani. Ez a tervezési ciklus egyidejű, de aszinkron végrehajtásaihoz vezet. Ennek célja a minél szofisztikáltabb artifaktum létrehozása.

A gyakorlatban számos konkrét design science kutatási folyamat modell létezik (Vaishnavi, et al., 2004). Ezek közül Peffers modelljét a következő ábra szemlélteti.



27. Ábra: Peffers design science folyamat modellje (Vaishnavi, et al., 2004)

4.2 Adatgyűjtés és elemzés

Az első adatgyűjtési metódus a szisztematikus szakirodalmi áttekintés kiegészítése az adott területre (ticketing rendszerek) vonatkozóan. Ez segít a kutatási kihívások megfogalmazásának pontosításában, az exploratory OLAP fogalmi keretrendszerének kialakításában, valamint a prototípus létrehozásában is. Az adatgyűjtés során azonosítom a nem strukturált szakterületi adatforrásokat és létrehozom a kezdeti ontológiát.

A következő módszer az összegyűjtött adatok vizsgálata. Ide tartozik az adatminőség elemzése, amelynek során elsősorban a következőket lehet ellenőrizni:

- Hiányzó értékek (pl.: nem kitöltött mezők)
- Kiugró értékek (pl.: az átlagosnál jóval hosszabb vagy esetleg csak két karakterből álló megjegyzés mező)
- Duplikációk
- Nem megfelelő értékek (pl.: szótárban nem szereplő szavak, rövidítések stb.).

Az adatminőség és az adatok vizsgálata alapján különböző statisztikai riportok készíthetők a nem strukturált adatforrásokra vonatkozóan. Például egy nem strukturált adatforrás esetén megadható, hogy az adatmezők kitöltöttségének százalékos mértéke vagy a duplikált sorok száma. Ezekhez a megállapításokhoz szükség lehet különböző szöveg előfeldolgozási műveletekre, pl.: felesleges szóközök eltávolítása, szótári szavak keresésére stb.

A riportokból kapott eredmények alapján szükséges lehet az adatok tisztítása abból a célból, hogy a rendszerbe ne kerülhessenek be nem megfelelő minőségű adatok.

4.3 A ticketing ontológia fejlesztése

Bár napjainkban számos ontológia fejlesztési módszertan ismert, ezek közül kevés a széles körűen elfogadott és kellően érett megoldás (Iqbal, et al., 2013). Kutatásom során ezek közül – figyelembe véve előnyeiket és hátrányaikat- választom ki az alkalmazandó ontológia fejlesztési módszertant.

Az ontológia fejlesztési módszertanok összehasonlítására Iqbal és társai (Iqbal, et al., 2013) a következő szempontokat említik:

- A fejlesztés típusa (szakaszos, prototípus fejlesztő, moduláris vagy iránymutató)

- Közösségi fejlesztés-e, azaz más felhasználók hozzájárulhatnak-e egy adott fejlesztő által létrehozott ontológia továbbfejlesztéséhez, fűzhetnek-e hozzá megjegyzéseket vagy vitatkozhatnak-e róla.
- Az alkalmazástól való függetlenség foka (teljesen független vagy félig független)
- Életciklus ajánlásokat tartalmaz-e? (várható élettartam, frissítés stb.)
- Milyen startégiát követ a fogalmak azonosítására? (fentről lefelé, alulról felfele, szabály alapú, vegyes vagy nem világos)
- Milyen a módszertan részletessége? (elégtelen, elégséges vagy néhol elégséges)
- Támogatja-e a más rendszerekbe való áthelyezést?

Ezen szempontok alapján Iqbal 15 gyakori ontológia fejlesztési módszert hasonlított össze (Iqbal, et al., 2013). Ezek a következők voltak: TOVE, Enterprise model approach, METHONTOLOGY, KBSI IDEF5, Ontolingua, CommonKADS and KAKTUS, PLINIUS, ONIONS, Mikrokosmos, MENELAS, Sensus, Cyc methodology, Upon, 101 method és On-To-Knowledge. Az összehasonlítás alapján a következő következtetések vonhatók le:

- A fejlesztés típusánál nincs domináns módszer, mindegyik lehetőség többször is előfordul. A Sensus annyiban tér el a többi módszertantól, hogy esetében nincs megadva preferált fejlesztési típus.
- Az összehasonlított módszerek közül csak az Ontolingua és a Sensus támogatja a közösségi fejlesztést.
- Az Enterprise model, METHONTOLOGY, KBSI IDEF5, Ontolingua, PLINIUS és Cyc methodology alkalmazás függetlenek. A többi említett módszertan részben vagy teljesen alkalmazás függő.
- Életciklus ajánlásokat egyedül a METHONTOLOGY módszertan tartalmaz.
- A fogalmak azonosítására alkalmazott stratégia változatos, mindegyik elvi lehetőség előfordul. Néhány módszer esetén (KBSI IDEF5, Ontolingua, ONIONS és Cyc methodology) nincs egyértelműen (világosan) megfogalmazott stratégia.
- Az egyes módszerek részletezettsége általában csak többé-kevésbé elfogadható mértékű, néhány esetben (CommonKADS és KAKTUS, ONIONS és MENELAS) elégtelennek mondható.

- Más rendszerekbe való áthelyezést csak az Ontolingua, az ONIONS, és a SENSUS módszertanok támogatják.

Gomez és társai (Gomez, et al., 2002) más szempontokat tartanak fontosnak. Ők elsősorban azt vizsgálták, hogy egyes projektmenedzsmenttel kapcsolatos, ontológia fejlesztési és integrációs folyamatok esetén javasolt-e az adott ontológia fejlesztési módszertan. Az ontológia fejlesztés illetően a következő folyamatokat különböztetik meg:

- Fogalmak felfedezése
- Követelmények
- Tervezés
- Implementáció
- Telepítés
- Műveletek
- Támogatás
- Fenntartás
- Megszűntetés.

A felsorolt módszerek alapján hét módszertan értékelésére került sor, ezek közül öt az előző szempontrendszer esetén is szerepelt (Cyc, KACTUS, METHONTOLOGY, SENSUS, On-To-Knowledge).

Az értékelés eredményei a következő módon foglalhatók össze:

- A vizsgált módszertanok közül a fogalmak felfedezésére egyedül az On-To-Knowledge módszertan javasolt.
- A követelmények és a tervezés folyamatai esetén egyedül a Cyc módszertan nem ajánlott.
- Az ontológiák implementációja minden módszertan esetén megfelelőnek bizonyult.
- A telepítés, műveletek, támogatás és megszűntetés folyamatok esetén egyik vizsgált módszertan sem bizonyult ajánlottnak.
- A fenntartás folyamata csak a METHONTOLOGY és az On-To-Knowledge módszertanok esetén volt megfelelő.

Az előzőek alapján látható, hogy főleg az ontológia fejlesztés előkészítő és az utólagos lépéseinél vannak hiányosságok. Több esetben is előfordul, hogy egyik módszert sem tartják alkalmasnak az éppen vizsgált folyamat esetén.

Az elemzések alapján a kutatásom támogatása szempontjából a tárgyalt módszertanok közül a következőket emelném ki (Iqbal, et al., 2013) (Gomez, et al., 2002):

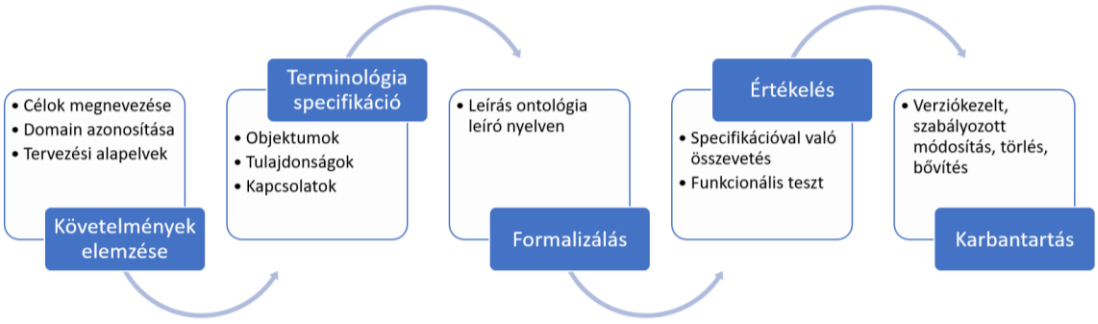
- TOVE (TOronto Virtual Enterprise): egy projekt keretében készült. A célja a vállalati működés modellezése és a vállalati integráció leírása volt. Az eredmény felfogható egy második generációs szakértői rendszernek is. A projektből kialakult ontológia módszertant később más célokra (pl: ellátási láncok) is felhasználták.
- CommonKADS: gyakran használt módszertan tudásalapú rendszerek létrehozására. Más módszertanokkal ellentétben ez a megközelítés szorosan kapcsolódik az objektumorientált programozásban használt UML jelölési módszertanhoz.
- SENSUS: a WordNet (az angol nyelv lexikai adatbázisa) kibővítésével készült ontológiára épülő módszertan.
- On-To-Knowledge: egy EU-s projekt keretében létrejött módszertan. A cél nagyszámú, heterogén strukturálatlan vagy félig strukturált dokumentumra épülő tudásbázis létrehozása volt. A dokumentumok nagyvállallatok intranetjeiről és a világhálóról származtak.

A ticketing ontológia fejlesztésénél (ld 4.3 alfejezet) az On-To-Knowledge módszertan leegyszerűsített változatát alkalmazom, mivel esetünkben a fogalmak felfedezése van a fókuszban. A módszertan főbb lépései a következők (Fensel, et al., 2003):

- Követelményelemzés. Az ontológiák fejlesztése, mint a legtöbb szoftverfejlesztési feladat a követelmények elemzésével, specifikálásával kezdődik. A követelményelemzés első lépése a célok megnevezése (miért hozzuk létre az ontológiát) és a tématerület (domain) azonosítása (milyen területet szeretnénk leírni). Például a domain lehet a ticketing rendszer, a cél pedig szemantikus keresés. Ezután az ontológia tervezés alapelveinek lefektetése következik. Az alapelvek függenek az ontológia típusától, valamint a megvalósítás módjától és eszközétől. Ide tartozik pl. a fogalmak elnevezésének konvenciója vagy a létrehozandó ontológia komplexitásának becslése (nagyságrendileg hány fogalom várható, milyen szintű lesz a granularitás). A következő részfeladat az adott területen rendelkezésre álló (felhasználható, újrahasznosítható) források felkutatása. Ilyen forrás lehet akár egy szótár, egy szabály gyűjtemény vagy akár egy meglévő szakterületi ontológia. Ezután fel kell térképezni, hogy kik (milyen felhasználók) és mire fogják használni a rendszert (használat esetek, forgatókönyvek). Ezt leginkább a felhasználókkal való kommunikációval (interjú, kérdőív) lehet megvalósítani.
- Terminológia specifikáció. A fogalmak (objektumok), tulajdonságok és a fogalmak közötti kapcsolatok leírását jelenti. Ennek kivitelezésére két fő megközelítés létezik. A fentről le (top-down) módszer esetén a modellezést a legmagasabb szintről kezdjük, és a modellt fokozatosan finomítjuk. Ilyenkor tipikusan az előző lépésben feltárt használati esetekből indulunk ki. Az alulról felfele (bottom-up) megközelítés ezzel ellentétes, mivel itt a dokumentumokból többnyire automatikusan kinyerhető információk jelentik a kiindulópontot.
- Formalizálás. Ez a lépés az ontológiák félig formális (semi-formal) leírását jelenti valamilyen ontológia leíró nyelven. Tipikusan ilyen nyelv az OWL (Web Ontology Language). A konkrét ontológia leíró nyelvet az adott feladat sajátosságait figyelembe véve kell kiválasztani, számításba véve az adott nyelv előnyeit és hátrányait.

- Értékelés. Ennek során ellenőrizni kell, hogy az ontológia a specifikációnak megfelelően készült-e el, teljesíti-e az ott leírt követelményeket. Ezen kívül funkcionális teszt elvégzése is szükséges, azaz meg kell nézni, hogy a tipikus használat esetekben a rendszer megfelelően működik-e. Fontos a (leendő) felhasználók visszajelzése. Ők tudják a leginkább megmondani, hogy az elkészült ontológia prototípus megfelel-e a felhasználói elvárásoknak. Ha nem, akkor a prototípus felülvizsgálata, finomítása szükséges. Az értékelés konkrét módjának leírását a 4.4 alfejezet tartalmazza.
- Karbantartás, továbbfejlesztés. Az elkészült és elfogadott ontológia természetesen rendszeres karbantartást igényel. Ez általában egy pontosan definiált szervezeti folyamat részeként történik, amely részletesen szabályozza a teendőket változások (módosítás, törlés, bővítés) esetén. A karbantartási feladatot rendszerint egy külön felhasználó (ontológia mérnök) végzi el. Megszokott gyakorlat a verziókezelés, amely segítségével az elvégzett változtatások nyomon követhetők vagy akár vissza is vonhatók.

A disszertációban alkalmazott ontológiafejlesztés lépéseit a következő összefoglaló ábra szemlélteti. Az egyes lépések között – szükség esetén – visszalépés is lehetséges.



28. ábra: Az ontológia fejlesztés lépései az On-To-Knowledge módszertan alapján (saját szerkesztés)

4.4 A ticketing ontológia értékelése

A 3.5 alfejezetben leírt lehetőségek közül egy kritérium alapú értékelési módszert választottam. Az értékeléshez, azaz a konkrét kritériumok teljesülésének megvizsgálásához itt a következő kérdéseket kell feltenni (Pâslaru-Bontaş, 2007):

- Tartalmi szempontból releváns-e a kapott ontológia modell az adott területen (esetünkben a ticketing rendszer)?

- Milyen minőségű a kapott ontológia mögötti fogalmi modell terület független nézőpontból, azaz milyen hatékony a tudás ábrázolás?
- Technikai szempontból mennyire felel meg a kapott ontológia figyelembe véve a legújabb technológiákat és eszközöket?
- Mennyire használható a kapott ontológia egy adott gyakorlati feladat esetén?
- Felhasználható-e az ontológia más alkalmazásokban?

Ezen kritériumokat fogom ellenőrizni a ticketing ontológia második verziójának elkészítése után (ld. 5.4.5 alfejezet).

4.5 Rendszerfejlesztés

A prototípus kifejlesztése felfogható egy rendszerfejlesztési, azon belül szoftverfejlesztési feladatnak is, emiatt célszerű ezt egy megfelelően kiválasztott szoftverfejlesztési módszertan szerint végezni (Bánné Varga, 2012).

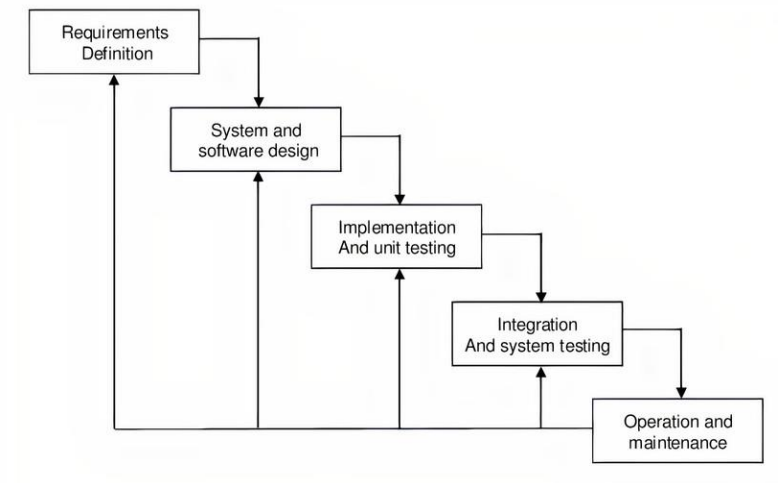
Ezekben a módszertanokban az a közös, hogy a fejlesztés során ugyanazokat a tevékenységeket (részfolyamatok) hajtják végre, csak más módon és megközelítésben (Sommerville, 2011).

A szoftverfejlesztés részfolyamatai a következők:

- Specifikáció. Ennek során az ügyfelek és a szoftver fejlesztők pontosan definiálják az előállítandó szoftver feladatait és a működésére vonatkozó korlátozásokat.
- Szoftverfejlesztés. Ez a részfolyamat a szoftver tervezését és elkészítését jelenti.
- Validálás. Annak ellenőrzése, hogy a szoftver a felhasználó igényeinek megfelelően készült-e el.
- Evolúció. A szoftver utólagos módosítása a változó felhasználói vagy piaci igényeknek megfelelően.

Számos szoftver fejlesztési módszertan létezik, ezek közül a fejlesztendő szoftver típusát figyelembe véve célszerű választani.

A klasszikus fejlesztési módszerek terv-vezéreltek (Sommerville, 2011), azaz minden egyes tevékenység előre megtervezett, és az eredményt a tervhez viszonyítjuk. A legegyszerűbb ilyen módszertan a vízesés modell, amely az előbbi tevékenységeket (több lépésre bontva) hajtja végre egymás utáni sorrendben. Minden lépés csak akkor kezdődhet el, ha az előző már befejeződött. Ez a modell csak akkor lehet eredményes, ha a feladat nagyon pontosan definiált, és a lépések jól elkülöníthetők.



29. Ábra: A vízesés modell (Sommerville, 2011)

A vízesés modell hatékonysága javítható, ha bizonyos részfolyamatokat párhuzamosítunk vagy ha több ciklusban végezzük el. Ilyen fejlesztési modell pl. az inkrementális fejlesztés modellje vagy az evolúciós fejlesztési modell.

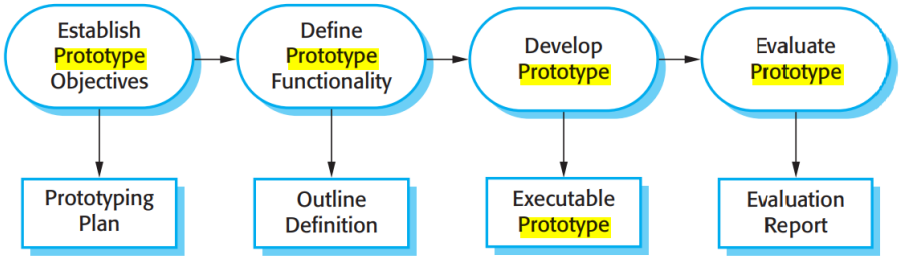
A klasszikus fejlesztési módszerekkel szemben az agilis fejlesztési módszertanok rugalmasan kezelik a specifikációt (Sommerville, 2011). Elsősorban a felhasználó igényeit tartják szem előtt, és megpróbálnak minél gyorsabban valamilyen kézzel fogható eredményt bemutatni. A fejlesztést ciklusokra (pl.: sprintek) osztják, és egy ciklus alatt csak a ciklus elején megfogalmazott célokra fókuszálnak.

Az exploratory OLAP rendszer fejlesztése során kézenfekvőnek tűnhet a vízesés modell alkalmazása, mivel megfelelő szakterületi ontológiák és a modell (Prasad (2010), Abello (2015)) megléte esetén a megoldandó feladat elég pontosan specifikálható, és így a kivitelezés egymástól elkülönülő, lineáris lépésekre bontható.

A vízesés modell alkalmazása esetén szükség lehet visszalépésre is. Például, ha az ontológiák értékelése nem megfelelő, akkor vissza kell lépni az ontológia létrehozására. Előfordulhat, hogy az alkalmazott ontológia fejlesztési módszertan felülvizsgálatra szorul, de az is lehet, hogy egy még korábbi lépést kell ismét végrehajtani.

Az exploratory OLAP rendszer esetén a vízesés modell mégsem jó választás. A fókusz ugyanis itt a szoftver kezdeti változatának (prototípusának) elkészítésén van, amely lehetővé teszi a koncepciók bemutatását, a tervezési lehetőségek kipróbálását és további információkat szolgáltat a problémáról és annak lehetséges megoldásairól.

A prototípus elkészítéséhez ezért a vízesés modell helyett az un. rapid prototípus módszert (Luqi, 2002) fogom alkalmazni, amely a prototípus gyors, iteratív előállítását helyezi előtérbe, ezáltal kézzelfogható eredményeket szolgáltat már a szoftverfejlesztés korai fázisában. A módszernek megfelelően, Prasad (2010), illetve Abello (2015) ötletei alapján több prototípus létrehozását és értékelését is tervezem. Egy adott prototípus létrehozásának lépéseit a következő ábra mutatja:



30. Ábra: A prototípus elkészítésének lépései (Sommerville, 2011)

Az első lépés a prototípus tervezése, amelynek során egyértelművé kell tenni a prototípus által megvalósítandó célkitűzéseket. Esetünkben a legfőbb célkitűzés az exploratory OLAP modell megvalósíthatóságának demonstrálása. Ezután következik a prototípus definiálása. Itt kell eldönteni, hogy mi kerüljön bele prototípusba a rendszer funkciók közül, illetve mit hagyjon ki a prototípus ezekből. A harmadik lépés a működő (kipróbálható) prototípus fejlesztése, amelyet annak értékelése (validálása) követ.

5 EREDMÉNYEK

Ebben a fejezetben a kutatás jelenlegi állását, meglévő eredményeit ismertetem. A 4. fejezetben leírtak alapján először az exploratory OLAP modell megvalósíthatóságát vizsgálom egy olyan prototípus elkészítésével, amely Prasad (2010) ötletének felhasználásával készült, de attól eltér. A prototípus a már bemutatott ticketing rendszer szöveges adatait használja adatforrásként.

Ezután a ticketing ontológia első verziójának elkészítését mutatom be, amelyre majd az Abello (2015) által javasolt exploratory OLAP prototípus elkészítésénél lesz szükség.
A ticketing ontológia felhasználásával manuálisan definiálok egy OLAP-struktúrát is, amelynek a későbbiekben lesz majd jelentősége. Az Abello (2015) ötletén alapuló prototípus ugyanis (félig) automatikusan generálja majd az OLAP-sémát. Így lehetőség nyílik a kétféle úton előállított OLAP-kockák összehasonlítására. Ezáltal a manuálisan előállított OLAP-kocka felfogható egyfajta referencia eredménynek, vagy akár az Abello (2015) ötletén alapuló exploratory OLAP prototípus nulladik verziójának is.

Az elkészült rapid prototípusok a későbbiekben további finomításokra szorulnak a kapott eredmények értékelésének függvényében.

5.1 Exploratory OLAP prototípus első verzió

A prototípus elsődleges célja, hogy a ticketing rendszerből származó szöveges adatok feldolgozásával előálljon a Prasad (2010) által javasolt csillagséma.

A prototípussal szemben további elvárás, hogy valósítsa meg a következő funkciókat:

- Szövegfeldolgozási műveletek (kulcsszavak keresése, és címkézése)
- A kulcsszavak tanulási képessége
- OLAP-séma elkészítése a csillagsémát felhasználva
- Dashboard készítése a kapott eredmény szemléltetésére.

Egyelőre nem cél a félig vagy teljesen automatizált működés.

A prototípus létrehozása a Prasad (2010) -féle megközelítésre épülő, de attól eltérő fogalmi modell alapján négy lépésben történt:

- Adatok előkészítése

- Szövegelemzések elvégzése
- Csillag séma létrehozása
- OLAP-kocka létrehozása, tesztelése.

A kivitelezés során a következő szoftvereket használtam:

- Service Desk (ticketing) program a forrásadatok exportálására
- Python 3.7 (Jupyter, Anaconda): a szövegfeldolgozó rutinok elkészítésére
- MS SQL Server 2017 Express, SQL Server Management Studio és SQL Server Import és Export varázsló a csillag séma létrehozására
- Power BI Desktop az OLAP-kocka elkészítésére, tesztelésére.

5.1.1 Adatok előkészítése

Az adatforrás a ticketing rendszer jegyei közül a hibabejelentéseket (Incident) vizsgálja egy kb. féléves intervallumban. Minden egyes hibához egy-egy rekord tartozik több tucatnyi mezővel. A prototípus elkészítése során csak a leglényegesebb mezőkkel dolgozunk.

A hibabejelentések alapvető szöveges adatai a hiba összefoglalása (Summary), és a hiba részletes leírása (Description). Ezeken kívül fontos adat még a hibajegy keletkezésének dátuma (Open_Date). Ez az adat természetesen DateTime formátumban is rendelkezésre áll, de az egyszerűség kedvéért most csak a dátum részt vesszük figyelembe. Ugyanúgy szükséges a hibajegy azonosító száma (Ticket Number). Ezt adatvédelmi megfontolások miatt egy számláló típusú mezővel helyettesítettem.

Az adatok előkészítése során először a ticketing rendszerből kiexportáltam a szükséges adatokat (Ticket Number, Open_Date, Summary, Description) az említett szűrőfeltételeket alkalmazva. Az exportálás során – az egyszerűség kedvéért - a .CSV formátumot választottam. Ennek oka, hogy ez a formátum a szövegelemzések lehetséges kimeneti formátumának is megfelel, másrészt az SQL-alapú adatbázis/adattárház rendszerek is támogatják ezt a formátumot. Ez eltér a Prasad (2010) által javasolt XML-formátumú adatkezeléstől, de a kivitelezés során ennek nincs jelentősége.

A ticketing rendszer elvileg csak konzisztens adatokat tartalmaz, ezért csak minimális adattisztításra (duplikációk megszűntetése) volt szükség. Erre azért volt szükség, mert gyakran előfordul, hogy ugyanazt a hibát, azonos tartalommal, egyidőben több felhasználó is bejelenti.

Az adatok előkészítésének második lépése a kulcsszavak meghatározása. Ennek egyik lehetséges módja, hogy manuálisan megadjuk a kulcsszavak listáját. Ennél jobb módszer, hogy megkeressük a szöveges adatforrás leggyakoribb szavait (esetleg szókapcsolatait), majd ezek közül kiválasztjuk a számunkra relevánsakat. A kulcsszavakat itt is .CSV formátumban tárolom.

A prototípus elkészítése során a két módszer kombinációját alkalmaztam a következő módon:

- Leírtam néhány kulcsszót, amely a hibajegyekkel kapcsolatban gyakran előfordul (pl: hardver, client, application stb.)
- Készítettem egy Python-scriptet, amely
    - megnyitja a forrásadatokat
    - eltűnteti a felesleges szóközöket
    - mindent kisbetűssé alakít
    - a szöveget szavakra bontja
    - elhagyja a töltelékszavakat (stopwords) és az angol szótárban nem megtalálható szavakat
    - a szavak közül csak a főneveket tartja meg
    - végül azokra a szavakra szűr, amelyek legalább 100-szor előfordulnak
- Ezután a két lista összefésülésével megkaptam a kulcsszavak kezdeti listáját. Itt szükség volt manuális „beavatkozásra” is, amely során a nem releváns találatok a listából eltávolításra kerültek.

A script készítése során felhasználtam egy szöveges állományt (words_alpha.txt), amely egy több tízezer szóból álló angol szótár szavait tartalmazza abc sorrendben. A sorrendet kihasználva bináris kereséssel ellenőriztem, hogy az egyes kulcsszavak benne vannak-e a szótárban.

Az adatok előkészítése után kapott két .CSV fájl felépítése a következő:

INCIDENTS.CSV (ID, OPEN_DATE, SUMMARY, DESCRIPTION)

KEYWORDS.CSV (ID, KEYWORD)

A kulcsszavak listája bővíthető akár manuálisan, akár a ticketing rendszer újabb szöveges adatainak feldolgozásával.

5.1.2 Szövegelemzések elvégzése

A szövegelemzéseket szintén Python-script segítségével végeztem el. A script a korábban tárgyalt TextRank algoritmust valósítja meg, ezenkívül a kulcsszavakat címkékkel is ellátja. Az eredményt a Prasad (2010) által javasolt felépítésben hozza létre .CSV formátumban.

A script a megfelelő csomagok importálásával kezdődik, ahol az egyes csomagok szerepe a következő:

- numpy: alapvető csomag mindenféle számítási feladathoz
- pandas: az adatfeldolgozást támogató könyvtár
- spacy: szövegelemzési rutinokat tartalmazó csomag
- pprint: a beépített print utasításnál „okosabb” kiíró utasítás
- collection: konténer jellegű adattípusokat tartalmazó csomag

Ezután a TextRank algoritmus megvalósítására szolgáló osztály elkészítése következik (Joshi, 2018). Az osztály felépítése a következő:

TextRank4Keyword
kw: List
+ init() **+ getkw(): List** **+ setkw_empty()** **+ set_stopwords()** **+ sentence_segment()** **+ get_vocab()** **+ get_token_pairs()** **+ symmetrize()** **+ get_matrix()** **+ get_keywords()** **+ analyze()**

Ezek közül a legfontosabb eljárások

- init(): beállítja a kezdeti adatokat (iteráció lépésszáma, csillapítási tényező, konvergencia küszöb)
- analyze():
    - a szöveget szavakra tördeli
    - a paraméterként megkapott szövegből eltávolítja a töltelékszavakat
    - kiszűri a szavak közül a főneveket és tulajdonneveket

    - elkészíti a TextRank algoritmusban szereplő mátrixot
    - egy iteráció segítségével meghatározza az egyes szavakhoz tartozó súlyokat
- get_keywords(): visszaadja a szövegben lévő 10 legnagyobb súllyal rendelkező kulcsszó nevét és súlyát
- getkw(): visszaadja a szövegben lévő kulcsszavak listáját

Ezután következik az adatok feldolgozása, amelynek során előáll a szöveges adatforrásból kinyert információ a következő formában:

DOCUMENT(DOC_ID, DATE, TITLE, INPUT, KEYWORD, CAT), ahol

- DOC_ID a dokumentum azonosítója
- DATE a hibajegy megnyitásának dátuma
- TITLE a dokumentum címe (az egyszerűség kedvéért ide az eredeti SUMMARY mező tartalma kerül, amely röviden összefoglalja a hibabejelentés célját)
- INPUT az a mondat, amely az algoritmus által feltárt kulcsszót tartalmazza
- KEWWORD az algoritmus által feltárt kulcsszó
- CAT a kulcsszó kategóriája (címkéje), amely lehet:
    - NN: főnév
    - NNP: tulajdonnév
    - VBN: ige
    - XX: egyéb

Az adatok feldolgozásának algoritmusa a következő:

```
document = üres lista
keywords = megnyit(keywords.csv)
szöveg = megnyit(text.csv)

ciklus a szöveg rekordjain
        ciklus a rekordok mondatain
                a mondat szövegének elemzése
                kulcsszavak kinyerése
                a kinyert kulcsszavak keresése a keywords listában
                ha van találat, akkor
                        ciklus a találati lista szavain
                                szövegcímke létrehozása az adott szóhoz
                                a document lista bővítése a kapott adatokkal
                        ciklus vége
```

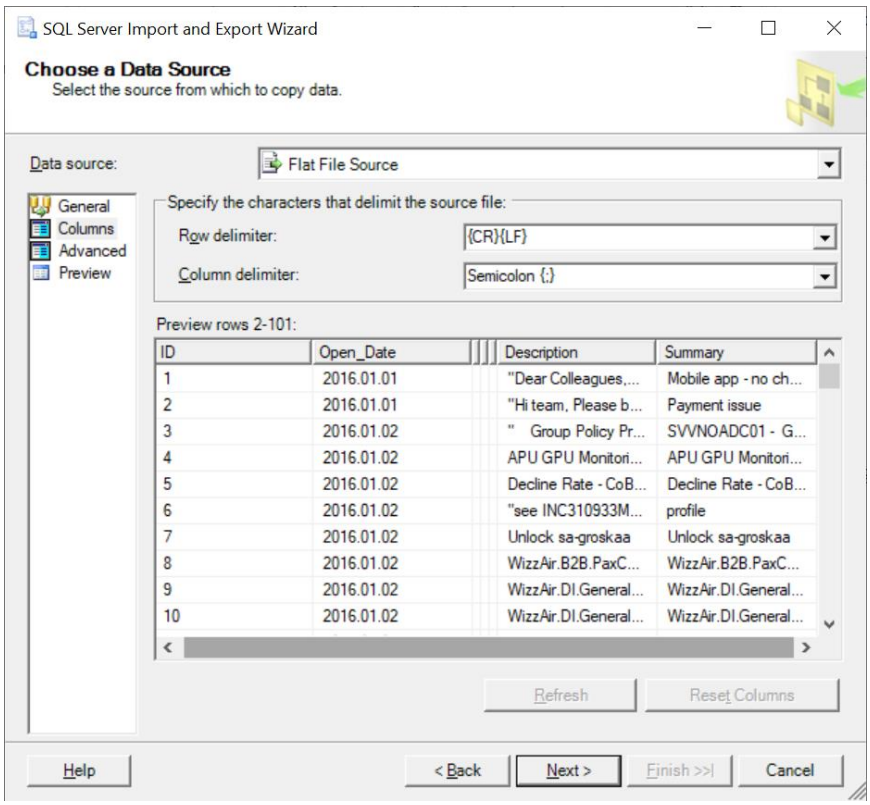
elágazás vége
ciklus vége
ciklus vége
KI: document (csv)

5.1.3 Csillag séma létrehozása

A csillag séma létrehozásához a következő feladatokat kell megoldani:

- A szövegelemzés során kapott eredmények adatbázisba töltése
- Prasad (2010) által javasolt struktúra kialakítása

Az adatbázisba való betöltés legegyszerűbb módja az SQL Server Import és Export varázsló használata, amelynek segítségével a .CSV fájl tartalmát egy adatbázisba tölthetjük.



31. Ábra: Az SQL Server Import és Export varázslója

A betöltött adatokat tartalmazó tábla legyen a következő nevű és felépítésű:

DOCUMENT_ANALYSIS(DOC_ID, DATE, TITLE, INPUT, KEYWORD, CAT)

Az adatok betöltése után a megfelelő struktúrát a legegyszerűbben nézetek segítségével alakíthatjuk ki. Az egyes nézetek létrehozására a következő utasítások szolgálnak:

```
CREATE VIEW [dbo].[DOCUMENT]
AS
SELECT        ROW_NUMBER() OVER(ORDER BY DOC_ID) AS DOC_ID,
              DOC_ID AS DOC_ORIG_ID,
              TITLE, INPUT

FROM            dbo.DOCUMENT_ANALYSIS
```

Itt a DOC_ID mező automatikus sorszámozású, de megtartjuk a korábbi azonosítót is DOC_ORIG_ID néven.

```
CREATE VIEW [dbo].[CATEGORY]
AS
SELECT        ROW_NUMBER() OVER (ORDER BY CAT) AS CAT_ID,
              d.CAT
FROM
 (SELECT DISTINCT CAT FROM DOCUMENT_ANALYSIS) d
```

Ennél a lekérdezésnél a szövegelemzésnél kapott kategóriákat helyezzük el egy új nézetbe.

```
CREATE VIEW [dbo].[KEYWORDS]
AS
SELECT        ROW_NUMBER() OVER (ORDER BY KEYWORD) AS KEYWORD_ID,
              d.KEYWORD
FROM
 (SELECT DISTINCT KEYWORD FROM DOCUMENT_ANALYSIS) d
```

Itt a kulcsszavakat töltjük be egy új nézetbe, szintén automatikus számozással.

A Dátum dimenzió tábláját egy script segítségével generáljuk le 2006.01.01-től 2020.12.31-ig.

```
Declare  @FromDate    Date = '2016-01-01',
         @ToDate      Date = '2020-12-31'

;With DateCte (Date) As
(
    Select  @FromDate Union All
    Select  DateAdd(Day, 1, Date)
    From    DateCte
    Where   Date < @ToDate
)
insert into date
Select  Date
From    DateCte
```

Ezek után – elsősorban az adatelemzések kedvéért – létrehozunk számított mezőket a táblához:

YEAR, QUARTER, MONTH, DAY_OF_YEAR, DAY_OF_MONTH, DAY_OF_WEEK, WEEK_OF_YEAR, WEEK_OF_MONTH

Végül a ténytáblának megfelelő nézetet generáljuk le:

```
CREATE VIEW [dbo].[ANALYSIS] AS
SELECT d.ID as DATE_ID, doc.DOC_ID, c.CAT_ID, k.KEYWORD_ID
FROM document_analysis p

JOIN DATE d on p.DATE = d.DATE
JOIN DOCUMENT doc on p.DOC_ID = doc.DOC_ORIG_ID AND p.INPUT = doc.INPUT
JOIN CATEGORY c on p.CAT = c.CAT
JOIN KEYWORDS k on p.KEYWORD = k.KEYWORD
```

A ténytábla érdekessége, hogy nem tartalmaz mértékeket, csak dimenzió azonosítókat.

A kialakított struktúra másik jellegzetessége, hogy a táblák közötti kapcsolatok nincsenek definiálva. Ennek oka, hogy ezt a feladatot a következő lépésben fogjuk elvégezni az adatmodell létrehozásával.

Fontos megjegyezni, hogy mind a betöltési folyamat, mind pedig a táblák és nézetek kialakítása automatizálható, ütemezhető. Az automatizálás a későbbiek során lesz megvalósítva a következő módon:

- A Python-script futtatható a Windows beépített feladat ütemezőjével (Windows Task Scheduler)
- A .CSV állományok az adatbázisba betölthetők a Microsoft legismertebb ETL-eszközével (SQL Server Integeration Services)
- A táblák és nézetek létrehozása automatizálható egy tárolt eljárás segítségével
- Az adatok betöltése és a tárolt eljárás futtatása megoldható az SQl Server Job Scheduler használatával

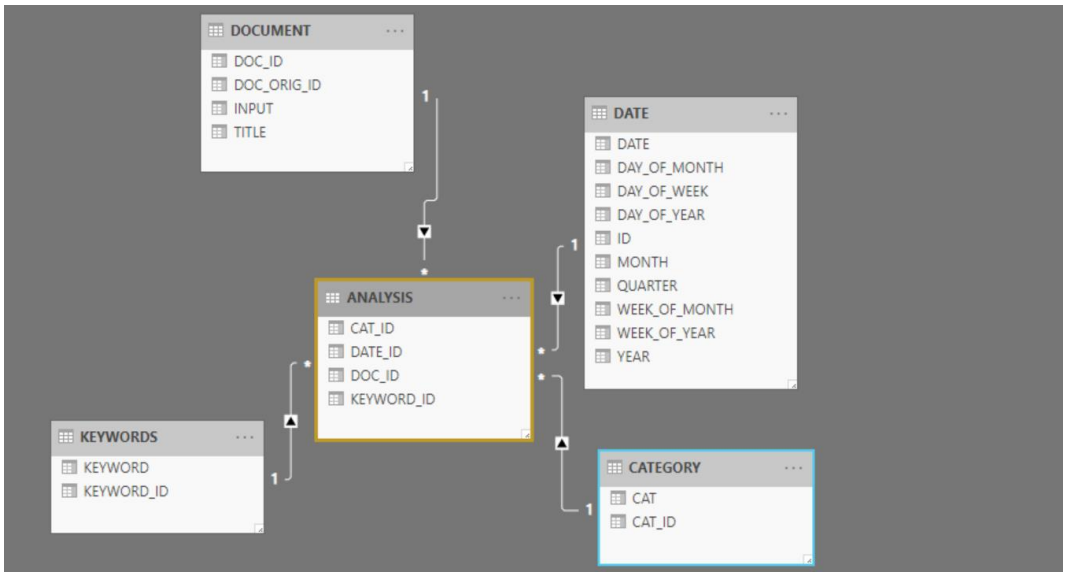
5.1.4 OLAP-kocka létrehozása, tesztelése

Az OLAP-kocka létrehozása általában egy szerver oldali eszközzel, pl: az SQL Server Analysis Services segítségével történik.

A feladat azonban egyszerűbb módon – kliens oldalról - is megoldható, ugyanis a Power BI Desktop program segítségével előállított adatmodell ezzel kompatibilis. Ez azt jelenti, hogy a Power BI-ban a modell kipróbálható, majd a későbbiekben az Analyis Services szolgáltatásba változtatás nélkül megnyitható.

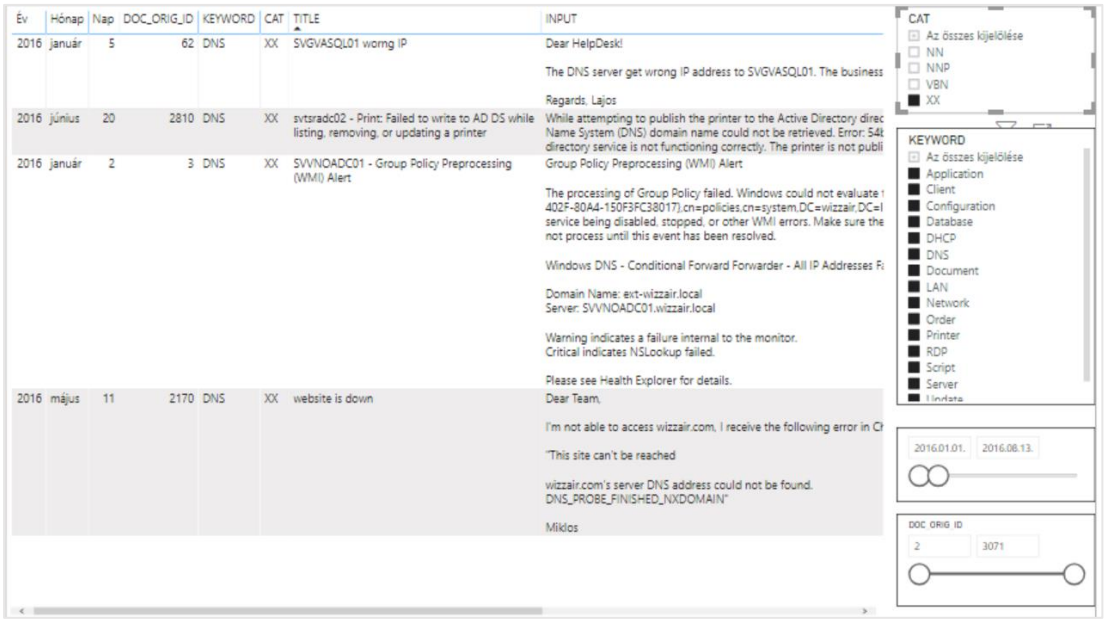
Az adatmodell létrehozásához csatlakozzunk az SQL Server-hez a hitelesítési adatok (szerver neve, adatbázis neve, hitelesítési mód) megadásával. Ezután importáljuk be a korábban elkészített táblákat és nézeteket: DOCUMENT, CATEGORY, KEYWORDS, DATE és ANALYSIS.

A modell nézetben rendezzük el a táblákat csillagszerűen az alábbi módon:



32. Ábra: Az OLAP-kocka modellje

A Power BI felismeri és automatikusan létrehozza a táblák közötti kapcsolatokat. Ezután vizuális nézetbe váltva a modell („OLAP-kocka") tesztelhető, pl. az alábbi dashboard létrehozásával:



33. Ábra: A Prasad-modell alapján létrejött dashboard

A dashboard táblázatos elrendezésben mutatja az egyes hibajegy leírások szövegének kulcs mondatait, azok kulcsszavait és kategóriáit, valamint a hibajegy létrehozásának dátumát. A lista szűrhető kategória, kulcsszó, dátum és dokumentum azonosító segítségével.

A kapott eredményeket (ezáltal a prototípust is) értékelve látszik, hogy a folyamat még optimalizálásra szorul. Erre több lehetőség is van:

- A scriptek többszöri futtatása változó paraméterekkel (pl.: a legnagyobb súllyal rendelkező kulcsszavak közül hányat veszünk figyelembe)
- A manuálisan megadott kulcsszavak számának növelése
- A kulcsszavak egyezése helyett a hasonlóságuk vizsgálata valamilyen metrika segítségével
- A kulcsszavak mellett több szóból álló kulcskifejezések figyelembevétele
- A nem releváns kulcsszavak arányának csökkentése (pl. szakterületi szótár használatával).

Fontos megjegyezni, hogy a Category dimenziót elvileg a Keywords dimenzióhoz kellett volna kapcsolni az Analyisis ténytábla helyett. Ekkor viszont hópehely séma jött volna létre, ami nem felel meg a célkitűzésnek.

5.1.5 Az elkészült exploratory OLAP rendszer validálása

A validálás a 3.5 alfejezetben megfogalmazott szempontok figyelembevételével történik. Mivel Prasad (2010) exploratory OLAP rendszere nem használ ontológiákat, ezért a validálás ebben az esetben jóval egyszerűbb, mint Abello (2015) rendszere esetén. A validálás során a legfontosabb feladat a multidimenzionális séma értékelése.

A sémában nincsen információvesztés nélkül törölhető (felesleges) attribútum, ezért a minimalitást jellemző redundanciamentesség értéke 1. Mivel nincsenek legalább kétszintű hierarchiák, ezért a kicsinyítési-nagyítási képesség értéke 0. Emiatt az adatok csak egyféle részletezettségben tekinthetők meg. Abban az esetben, ha a DATE dimenziót átalakítanánk YEAR-MONTH-DAY formájú hierarchiává, akkor ennek a mérőszámnak az értéke (4) alapján Z= $\frac{\left(1-\frac{1}{3}\right)}{1} = \frac{2}{3}$ -ra változna.

A séma egyetlen ténytáblát tartalmaz, amely nem tartalmaz mértéket. Emiatt a kifejezőképességét mérő mutatók közül a ténytábla lokális gazdagságának értéke (5) szerint itt nem értelmezhető.

A séma egyszerűségét (6) alapján jellemezhetjük a következő értékkel: $S = \frac{1+0}{1+0+4} = \frac{1}{5}$. Amennyiben a DATE dimenzió helyett itt is létrehoznánk a háromszintes YEAR-MONTH-DAY hierarchiát, akkor ez az érték a következőképpen módosulna:

$S = \frac{1+3}{1+3+6} = \frac{2}{5}$

A séma helyességét a próba lekérdezések alapján lehet vizsgálni. Mivel hiba nem adódott, ezért annak értéke (7) miatt C=1.

A kapott multidimenzionális séma egyszerűen lekérdezhető. Mivel aggregációk nincsenek, ezért a jelenlegi adatmennyiség sokszorosa esetén sem várhatók performancia problémák. Ez alól kivételt csak az INPUT mezővel kapcsolatos lekérdezések jelenthetnek. Ez a szöveges mező tartalmazza a hibajelentések teljes szövegét. A szöveg nem strukturált, ezért a benne való keresés lassú lehet.

5.2 A ticketing ontológia

Az ontológiák fejlesztése a 4.3 alfejezetben leírtak alapján történik. Ebben az alfejezetben a fejlesztés első három lépése kerül bemutatásra.

A ticketing ontológiákból két verziót is készítek. Az első verzió egy kétszintű taxonómia, amely könnyen és gyorsan elkészíthető. Hátránya, hogy csak korlátozottan használható az exploratory OLAP prototípus elkészítésénél. Ezt a verziót fogom használni az 5.3.3 alfejezetben bemutatott dimenzionális tervezésnél.

Ezután bemutatok egy második (fejlettebb) verziót is, amelynek elkészítése nehezebb, viszont a gyakorlatban jobban használható. Az Abello (2015) ötletét felhasználó exploratory OLAP prototípus (ld. 5.4) ezt az ontológiát használja.

Az ontológia fejlesztés negyedik lépése (az elkészült ontológia értékelése) az 5.4.5 alfejezetben található. Itt a ticketing ontológia második verziója kerül validálásra. Az ötödik lépéssel (karbantartás) a disszertációban nem foglalkozom, ez további kutatási feladatok tárgyát képezheti.

5.2.1 Első verzió

Az első lépés a követelmények elemzése. Az elkészítendő ontológia szakterülete a ticketing rendszer, azon belül a hibabejelentések (incidensek) alrendszere. A cél egy olyan ontológia létrehozása, amely elősegíti a szemantikus keresést az incidensekben. Első körben az incidens témájának azonosítása, kategorizálása a leglényegesebb feladat. Kezdetben elég néhány (3-5) fő kategória, és max. 50 alkategória megadása.

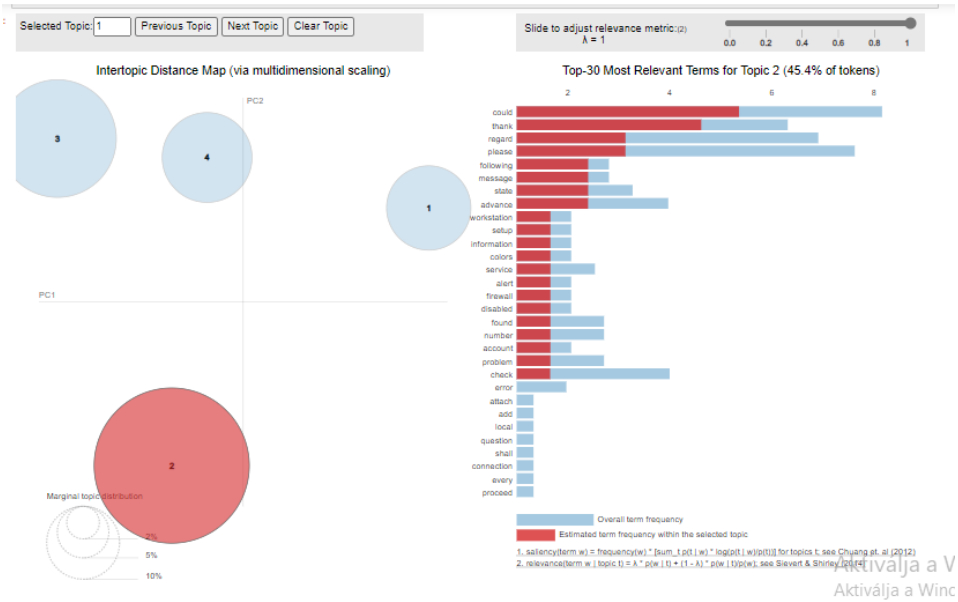
A második lépés a terminológia specifikáció, amely során meg kell adni az objektumokat (osztályokat), tulajdonságaikat és a kapcsolatokat. A követelmények miatt elég egy kétszintű taxonómiát létrehozni a szöveges adatokban lévő témák és altémák azonosításával. Ezt a feladatot a LDA (Latent Dirichlet Allocation) modell segítségével oldom meg (Revert, 2018). Ez egy olyan felügyelet nélküli gépi tanulási modell, amely szöveges dokumentumokhoz témákat rendel. Ezen kívül a modell azt is megmutatja, hogy a témák az egyes dokumentumokban milyen mértékben (százalékban) vannak jelen.

Egy-egy téma szavak súlyozott listáját (tömbjét) jelenti, pl: $téma_1$ = [0,15*"hdd", 0,2*"mouse", 0,3*"printer", …]. A modellnek 3 fő paramétere van: a témák száma (összesen), a szavak száma egy témában, és a témák száma dokumentumonként.

A témák azonosítását végző Python-script a Gensim könyvtárat használja, amely felügyelet nélküli gépi tanulási modellek – többek között az LDA modell - megvalósítását is tartalmazza. A kapott eredmény vizuálisan is szemléltethető a pyLDAvis csomag segítségével.

A script futtatása előtt szükséges a szöveg előkészítése, amely a következő lépéseket foglalja magában: a szöveg egységekre bontása (tokenizálás), a szavak elő- és utótagjainak eltávolítása (lemmatizálás), a szótárban nem megtalálható szavak elhagyása, kisbetűssé alakítás, valamint a külön jelentéssel nem bíró szavak (stopwords) elhagyása.

A script nagy mennyiségű szöveges adat esetén is gyorsan lefut. Egy ilyen futtatás eredményét a következő ábra szemlélteti:

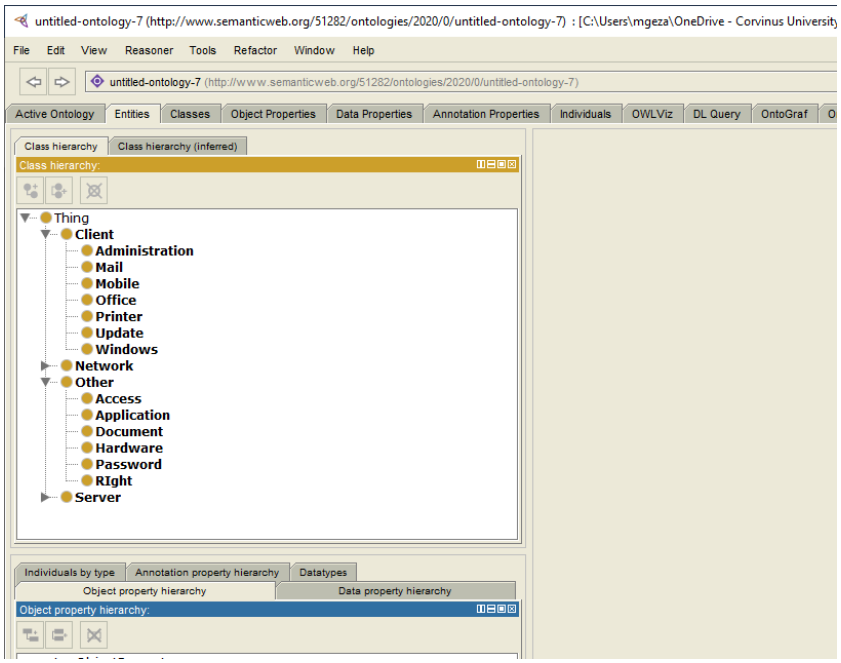


34. Ábra: Az LDA modell segítségével azonosított fő témák

Az altémák meghatározásához nem használtam külön módszert/algoritmust, hanem az egyszerűség kedvévért az LDA modell által azonosított témákból hoztam létre a kétszintű taxonómiát.

A taxonómia elkészítéséhez tehát manuális beavatkozásra is szükség volt. Ennek egyik oka, hogy nem releváns szavak is bekerülhetnek a témák szavai közé, ezeket az algoritmus nem tudja eltávolítani. Másrészt, a többszöri, különböző paraméterekkel történő futási eredmények közül a legjobb kiválasztása, és a kapott témák, valamint altémák felcímkézése szintén igényli az emberi közreműködést.

A taxonómia a Protegé szoftver segítségével készült el. Ennek egy részletét mutatja a következő ábra:



35. Ábra: A kezdeti ontológia (taxonómia) a Protege-ben

A taxonómia, azaz a témák és az altémák azonosítása alapján a hibajegyek 4 fő kategóriába sorolhatók:

- Server: szerveroldali problémák, pl.: a szerver nem elérhető
- Client: kliensoldali problémák, pl.: az Office Word alkalmazás nem indul el
- Network: hálózati problémák, pl.: nem működik a VPN
- Other: az előző három kategóriába be nem sorolható problémák, pl.: a jelszó lejárt

Ezek a kategóriák további alkategóriákra bonthatók. A következő felsorolások ezek megnevezését és rövid leírását tartalmazzák.

A Server kategória esetén jelenleg a következő alkategóriák vannak definiálva:

- Database: a szerveren tárolt adatbázisokkal kapcsolatos problémák
- DHCP: a cég helyi hálózatának IP-konfigurációs problémái
- DNS: a cég helyi hálózatának névfeloldási problémái
- Share: a cég megosztott tárolójának elérésével kapcsolatos problémák
- Storage: a szervereken tárolt adatokkal kapcsolatos problémák

A Client kategória jelenleg három alkategóriát tartalmaz:

- Mail: a levelezőrendszerrel kapcsolatos problémák
- Office: az irodai programcsomag problémái

- Printer: nyomtatási problémák

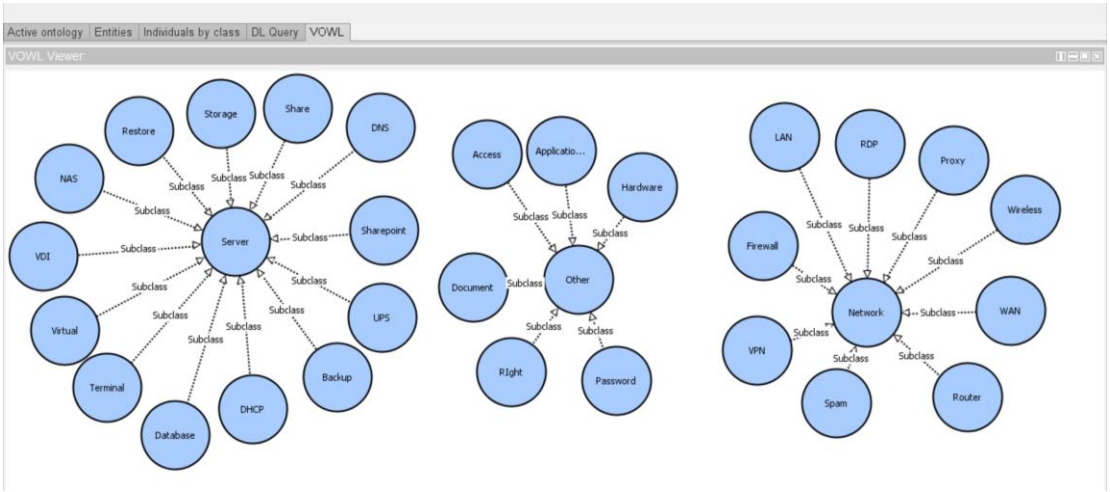
A Network kategória jelenleg a következő alkategóriákat tartalmazza:

- Firewall: minden olyan probléma, amely a cég tűzfalával kapcsolatos
- LAN: helyi hálózati problémák
- Proxy: internetes oldalak elérési problémái
- Router: internetkapcsolat problémák, amelyek a céges hálózattal összefüggésben lehetnek
- WAN: internetkapcsolat problémák, amelyek a céges hálózattól függetlenek
- Wireless: vezetéknélküli hálózati problémák

Az Other kategória jelenlegi alkategóriái a következők:

- Application: felhasználói szoftveres problémák (kivéve irodai programok)
- Document: dokumentumokkal kapcsolatos hibák
- Hardware: hardver eszközök problémái

A kategóriák és alkategóriák (másképpen: osztályok és alosztályok) segítségével létrehozott taxonómiák lesznek a kezdeti ontológiák. A következő ábra a Server, Network és Other osztályokat és azok alosztályait szemlélteti:



36.Ábra: A ticketing ontológiák kezdeti rendszere (részlet)

A harmadik lépés a formalizálás, azaz a kapott ontológiák leírása ontológia leíró nyelven. A leírás OWL nyelven történik (https://www.w3.org/TR/owl-features/), amely segítségével a kapott ontológia feldolgozásra alkalmas formában leírható. Ennek első verziója 2004-ben, a második 2009-ben jelent meg. A nyelv maga három résznyelvből áll, és többféle szintaxissal rendelkezik. Itt most csak a kapott taxonómia leírására szolgáló nyelvi elemekkel foglalkozom az OWL/XML szintaxist használva.

Egy osztály a <Declariation> és a <Class> tagek segítségével adható meg:

```
<Declaration>
        <Class IRI=”#Osztálynév” />
</Declaration>
```

Például a következő kódrészlet definiálja az Ohter, Application, Document és Hardware osztályokat:

```
<Declaration>
        <Class IRI=”#Other” />
</Declaration>
<Declaration>
        <Class IRI=”#Application” />
</Declaration>
<Declaration>
        <Class IRI=”#Document” />
</Declaration>
<Declaration>
        <Class IRI=”#Hardware” />
</Declaration>
```

Az osztályok deklarálása után meg kell adni a közöttük lévő kapcsolatokat is. Esetünkben az Application, Document és Hardware osztályok az Other osztály alosztályai. Ezt a <SubClassOf> tagek segítségével lehet megadni a következő módon:

```
<SubClassOf>
    <Class IRI="#Application"/>
    <Class IRI="#Other"/>
 </SubClassOf>
<SubClassOf>
    <Class IRI="#Document"/>
    <Class IRI="#Other"/>
 </SubClassOf>
<SubClassOf>
    <Class IRI="#Hardware"/>
    <Class IRI="#Other"/>
 </SubClassOf>
```

Az elkészült ontológia többi része is hasonló módon leírható.

5.2.2 Második verzió

A követelmények az első verzióhoz képest bővültek. A legfontosabb új követelmény, hogy az ontológia ne csak a hibajegyek kategorizálására legyen alkalmas. Lehetőség szerint minél jobban írja le a ticketing rendszer működését, azon belül is a hibajelentések (incidensek) kezelését az alábbi pontok alapján:

- Az ügyfél szöveges módon jelezi a problémát a helpdesknek, amivel megnyit egy hibajegyet.
- A hibajegy szövege egyrészt egy rövid megfogalmazásból (tárgy), másrészt egy részletesebb leírásból áll. A hibajegy emellett egyéb jellemzőkkel bír. Ilyen például a ticket létrehozásának ideje, vagy a hiba sürgőssége (prioritás).
- A ticket egy hozzáértő személyhez (assignee, analyst) lesz rendelve, aki megoldja a problémát. A megoldás során végzett tevékenységek rögzítésre (logolás) kerülnek.
- A megoldás során a ticket bizonyos jellemzői (pl. a státusza) többször is megváltozhatnak.
- Nem cél a ticket teljes életútjának leírása, itt csak a ticket kezdeti és végső állapota lényeges.
- Legyen lehetőség néhány mérőszám (mérték, KPI) ábrázolására is, pl. mennyi egy hibajegy átlagos megoldási ideje stb.

A ticketing ontológia második verziójában fontos, hogy az osztály-alosztály kapcsolaton kívül más kapcsolatok, illetve az osztály tulajdonságok is megjelenjenek. Konkrét példányok létrehozása az exploratory OLAP prototípus elkészítéséhez nem szükséges.

A fogalmak meghatározása során Skoutas & Simitsis (2007) ötleteit használom fel némileg átalakítva és leegyszerűsítve. Az alkalmazott módszer („reverse engineering”) lényege az adatforrásokból kinyerhető fogalmak és az adott szakterület (ticketing incidensek) szótárának integrálása. Az eredeti megoldás a strukturált és nem strukturált adatokat egyaránt gráfként ábrázolta, majd ezekből egy saját algoritmus segítségével (félautomatikusan) hozta létre az ontológiát. Az így kapott ontológia un. felhasználó-centrikus szakterületi ontológia, azaz az adott szakterületről csak a vizsgált adatforrások szempontjából szükséges ismereteket tartalmazza.

A szöveges adatforrásokból származó fogalmak (kulcsszavak) listája az 5.1.2 pontban már elkészült, így rendelkezésre áll. A szakterületi fogalomgyűjteményt az Internetről letöltött tematikus források segítségével állítottam össze (ld. következő táblázat). A több helyen is előforduló szavakat természetesen csak egyszer vettem figyelembe, így a fogalomgyűjtemény összesen 526 fogalmat/szót tartalmaz.

Sorszám	Adatforrás linkje	Szavak száma
1.	https://www.proprofsdesk.com/blog/help-desk-glossary/	60
2.	https://buildahelpdesk.com/help-desk-glossary/	80
3.	https://freshdesk.com/customer-support-glossary	115
4.	https://www.liveagent.com/customer-support-glossary/	371

2. táblázat: A szakterületi fogalomgyűjtemény forrása

Mivel nem cél a helpdesk/ticketing terület teljes leírása, ezért először az ontológia szempontjából releváns fogalmak kiválasztásához egy Python-script segítségével szemantikus összehasonlítást végeztem a szöveges adatforrásból kinyert kulcsszavak és a szakterületi fogalomgyűjtemény szavai között.

Ezután a kapott eredményből leszűrtem a szakterületi fogalomgyűjtemény azon szavait, amelyekhez található magas (elsősorban 0.75 feletti) hasonlósági indexű kulcsszó az adatforrásokból. A kapott szavakból manuálisan választottam ki, illetve határoztam meg az ontológiában szereplő fogalmakat (osztályokat), és a tulajdonságokat. A kapcsolatokat a követelményekből kiindulva hoztam létre.

Az ontológia fejlesztés második lépése (terminológia specifikáció) során a következőket vettem figyelembe:

- Az eredeti elnevezéseket néhány kivételtől eltekintve meghagytam. Néhány esetben a nevet egy előtaggal kiegészítettem, pl: Category helyett Ticket_Category.
- A lehetséges szinonimákkal egyelőre nem foglalkoztam.
- Az Activity log esetén az Activity-t osztálynak, a log-ot kapcsolatnak vettem fel.
- Az Activity osztályhoz manuálisan rögzítettem a Begin Time és End Time alosztályokat, mivel ezek nem voltak benne a szógyűjteményben.
- A kapcsolatok egy részét a szavak között nem szereplő „Has” kapcsolat segítségével definiáltam.

- Elhagytam azokat a fogalmakat, amelyek az esetünkben nem relevánsak, pl: Parent Ticket, Reopen Ticket, Due by time stb.

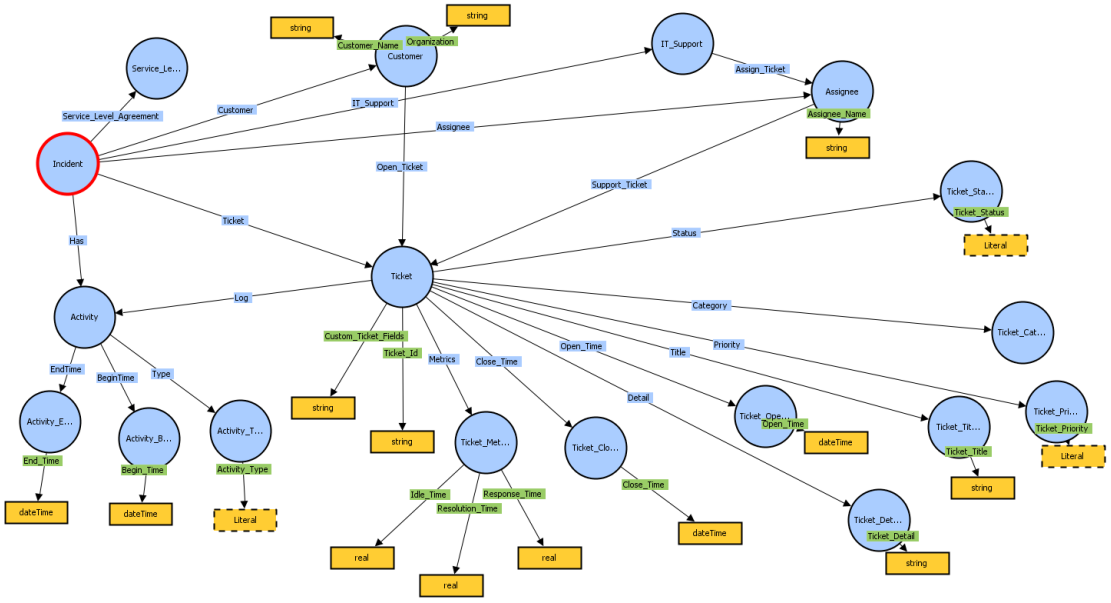
A kapott eredményt a következő táblázat szemlélteti:

Típus	**Név**	**Megjegyzés**
Osztály	Incident	
Osztály	Activity	
Osztály	Activity_Type	
Osztály	Activity_Begin_Time	
Osztály	Activity_End_Time	
Osztály	Assignee	
Osztály	Customer	
Osztály	IT Support	
Osztály	Service Level Agreement	
Osztály	Ticket	
Osztály	Ticket_Open_Time	
Osztály	Ticket_Close_Time	
Osztály	Ticket_Title	
Osztály	Ticket_Detail	
Osztály	Ticket_Status	
Osztály	Ticket_Priority	
Osztály	Ticket_Category	
Osztály	Ticket_Metrics	
Kapcsolat	Has	Incident és Activity osztályok esetén
Kapcsolat	Assign Ticket	IT_Support és Assignee osztályok között
Kapcsolat	Log	Ticket és Activity osztályok között
Kapcsolat	Open Ticket	Customer és Ticket osztályok között
Kapcsolat	Support Ticket	Assignee és Ticket osztályok között
Tulajdonság	Activity_Type →Activity_Type	Lehetséges értékek: Close, Comment, Forward, Resolve, Response, Schedule
Tulajdonság	Activity_Begin_Time→ Begin Time	
Tulajdonság	Activity_End_Time → End Time	
Tulajdonság	Assignee → Assignee Name	
Tulajdonság	Customer → Customer Name	
Tulajdonság	Customer → Organization	
Tulajdonság	Ticket → Ticket Id	
Tulajdonság	Ticket_Open_Time → Open Time	
Tulajdonság	Ticket_Close_Time→ Close Time	
Tulajdonság	Ticket_Detail → Ticket Detail	A ticket szöveges leírása

Típus	Név	Megjegyzés
Tulajdonság	Ticket_Title → Ticket Title	A ticket címe, tárgya
Tulajdonság	Ticket → Resolution Time, Response Time, Idle Time	
Tulajdonság	Ticket →Custom Ticket Fields	
Tulajdonság	Ticket_Priority → Ticket_Priority	Lehetséges értékek: High, Low, Medium
Tulajdonság	Ticket_Status → Ticket_Status	Lehetséges értékek: Open, Closed, Waiting, Resolved, Unresolved

3. táblázat: Az ontológiában szereplő osztályok, kapcsolatok és tulajdonságok

A kapott ontológia grafikus formában a következő ábrán látható:



37. ábra: A ticketing ontológia második verziója

Az ontológiát ezután még kiegészítettem az első verziójánál kapott hibajegy kategóriákkal, amelyek a Ticket_Category alosztályai lettek.

Az ontológia fejlesztés harmadik lépése a kapott ontológia leírása OWL/XML szintaxis szerint. Az első verzióhoz hasonlóan itt is - terjedelmi okok miatt – a leírásnak egy részlete szerepel, konkrétan az Activity osztály és annak tulajdonságai, kapcsolatai.

```
<Declaration>
    <Class IRI="#Activity"/>
  </Declaration>
...
<ObjectPropertyDomain>
```

```
      <ObjectProperty IRI="#BeginTime"/>
      <Class IRI="#Activity"/>
   </ObjectPropertyDomain>
<ObjectPropertyDomain>
      <ObjectProperty IRI="#EndTime"/>
      <Class IRI="#Activity"/>
   </ObjectPropertyDomain>
<ObjectPropertyDomain>
      <ObjectProperty IRI="#Type"/>
      <Class IRI="#Activity"/>
   </ObjectPropertyDomain>
…
<ObjectPropertyRange>
      <ObjectProperty IRI="#BeginTime"/>
      <Class IRI="#Activity_Begin_Time"/>
   </ObjectPropertyRange>
…
<DataPropertyDomain>
      <DataProperty IRI="#Activity_Type"/>
      <Class IRI="#Activity_Type"/>
   </DataPropertyDomain>
…
<DataPropertyRange>
      <DataProperty IRI="#Activity_Type"/>
      <DataOneOf>
         <Literal>Close</Literal>
         <Literal>Comment</Literal>
         <Literal>Forward</Literal>
         <Literal>Resolve</Literal>
         <Literal>Response</Literal>
         <Literal>Schedule</Literal>
```

```
    </DataOneOf>
  </DataPropertyRange>
```

A többi osztály, tulajdonság, illetve kapcsolat hasonló módon írható le.

5.3 OLAP-kocka tervezése a ticketing ontológiák felhasználásával

Ez az OLAP-kocka későbbiekben az Abello (2015) exploratory OLAP fogalmi modelljére épülő prototípussal való összehasonlításra fog szolgálni. A kocka tervezése a következő lépésekben történik:

- Adatok előkészítése
- Szövegelemzés és szövegfeldolgozás
- Dimenzionális tervezés.

A tervezés során a következő szoftvereket használtam:

- Service Desk (ticketing) program a forrásadatok exportálására
- Python 3.7 (Jupyter, Anaconda): a szövegfeldolgozó rutinok elkészítésére.

5.3.1 Adatok előkészítése

Az adatok előkészítése itt is az 5.1.1 alfejezetben leírtak alapján történik ugyanazt a .CSV adatforrást (ticketing rendszer hibajegyeinek szövege) felhasználva.

Az elemzési lehetőségek kibővítése érdekében az adatforrás még tartalmazza a hibajegy prioritását, a létrehozójának a nevét, és az esetleges lezárás dátumát is. Az eredeti adatforrásból sok más adat is releváns lenne, de ennél az OLAP-kockánál elsősorban a nem strukturált (szöveges) adatok elemzésén van a hangsúly. Emiatt a kockából kimarad a hibajegyet kezelő(k) neve, a megoldására felhasznált munkaórák száma stb.

A különbség ezenkívül még annyi, hogy itt nincs szükség a kulcsszavak előzetes megadására. Ennek oka, hogy itt a kulcsszavak az ontológiák osztályainak, illetve alosztályainak feleltethetők meg. A szövegelemzés és feldolgozás során a ticketing ontológia első verziója kerül felhasználásra (ld. 5.2.1 alfejezet).

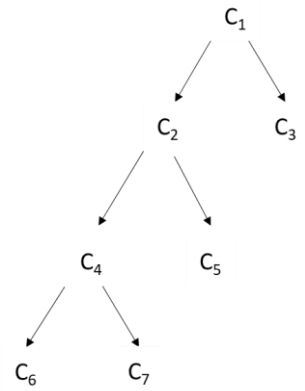
5.3.2 Szövegelemzés és szövegfeldolgozás

A szövegelemzés során az 5.1.2 részben tárgyalt TextRank algoritmus segítségével kinyerjük a ticketing hibabejelentésekben lévő kulcsszavakat, majd azok listáját összevetjük az ontológia osztályok, illetve alosztályok elnevezésével (Liu & Wang, 2014).

Az ontológia osztályokat egy listában (rendezett vektorban) tároljuk a következő formában:

['Access', 'Administration', 'Application', 'Backup', 'Client', 'DHCP', 'DNS', 'Database', 'Document', 'Firewall', 'Hardware', 'LAN', 'Mail', 'Mobile', 'NAS', 'Network', 'Office', 'Other', 'Password', 'Printer', 'Proxy', 'RDP', 'RIght', 'Restore', 'Router', 'Server', 'Share', 'Sharepoint', 'Spam', 'Storage', 'Terminal', 'UPS', 'Update', 'VDI', 'VPN', 'Virtual', 'WAN', 'Windows', 'Wireless']

Az ontológia osztályoknak valójában egy hierarchikus szerkezet (fa) felel meg, amelynek csúcsai az osztályok (fogalmak), pl. egy ilyen fa:



38. ábra: Osztályok és alosztályok hierarchiája

A kulcsszavakkal való összehasonlításkor a fa bizonyos csúcsait megkülönböztetjük aszerint, hogy a csúcs megegyezik a kulcsszóval (közvetlenül csatlakozó csúcsok), vagy a kulcsszóval egyező csúcs valamelyik őse, vagy leszármazottja (releváns csúcsok). Például az előző ábrán legyen C_2 az a csúcs, amely az egyik kulcsszóval azonos. Ekkor C_2 közvetlenül csatlakozó csúcs, C_1, C_4, C_5, C_6 és C_7 releváns csúcsok.

A mindkét listában előforduló szavak segítségével vektorokat (un. concept vektorok) hozunk létre. Egy adott kulcsszóval való egyezés esetén a concept vektor minden zérustól különböző eleme a közvetlenül csatlakozó csúcs és a releváns csúcsok figyelembevételével jön létre. Az előző példában (C_2 csúccsal való egyezés) a concept vektor nem nulla elemei esetén figyelembe vett csúcsok: C_2, C_1, C_4, C_5, C_6 és C_7.

A concept vektor adott eleme

- 0, ha az adott osztály vagy alosztály neve nem szerepel a kulcsszavak között
- $w \in (0,1)$, ha egy alosztály neve szerepel a kulcsszavak között (w értéke konstans)

- 1, ha egy osztály neve szerepel a kulcsszavak között

A w súly szerepe, hogy megkülönböztesse a közvetlenül csatlakozó csúcsokat a releváns csúcsoktól. Ennek oka, hogy ez utóbbiak nem annyira fontosak a kulcsszóval való egyezésnél, ezért kisebb súllyal szerepelnek.
Ennek segítségével a concept vektor a következő lesz: [w, 1, 0, w, 0, w, w]

A concept vektorok körében értelmezhető a hasonlóság fogalma. Ennek legegyszerűbb mérőszáma a vektorok által bezárt szög: ha ez kicsi, akkor a vektorok „hasonlóak", ha nagy (90º közeli), akkor a vektorok teljesen eltérnek (merőlegesek).

Két vektor „Koszinusos hasonlóságának" (cosine similarity) kiszámításának módja:

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}},$$

Egy mondatban több kulcsszó is lehet, ezért mondatok hasonlósága is definiálható a következőképpen:

- Keressük meg minden mondatban a kulcsszavakat
- Keressük meg a kulcsszavakhoz közvetlenül csatlakozó csúcsokat
- Terjesszük ki a közvetlenül csatlakozó csúcsok listáját a hozzájuk tartozó releváns csúcsokkal
- Definiáljuk a concept vektort az egy kulcsszóval való egyezéshez hasonló módon
- Képezzük a mondatok hasonlóságát az előző formula (koszinuszos hasonlóság) segítségével

Például, legyenek egy mondat kulcsszavai C_4 és C_5 (lásd. Osztályok és alosztályok hierarchiája ábra). Ekkor a releváns csúcsokkal kiegészített lista a következő lesz: C_4, C_5, C_1, C_2, C_6, C_7.

Ebből kiindulva a mondathoz tartozó concept vektor: [w, w, 0, 1, 1, w, w]

Egy adott szöveg több mondatból áll, ezért van értelme beszélni a szövegek (dokumentumok – esetünkben a ticketing rendszerből származó hibajelentések) hasonlóságáról is. A mondatok és a dokumentumok között fontos különbség, hogy a dokumentumok jóval több szót tartalmaznak. Emiatt az egyes szavak fontossága másképpen értelmezendő egy dokumentumban, mint egy mondat esetén. A mondatoktól eltérően itt a kulcsszavakhoz közvetlenül csatlakozó csúcsokat nem feltétlenül egyforma súllyal kell figyelembe venni a hasonlóság számításánál.

Az ontológia osztályoknak megfelelő fa struktúrában minél magasabb szinten van egy fogalom, a jelentése annál általánosabb. Például az első taxonómia szinten lévő Client szó jelentése elég általános, a második szinten lévő Windows szó viszont jóval konkrétabb jelentéssel bír. Ezért a kulcsszavak megfelelő súlyozásához vezessünk be egy α együtthatót (súly terjedési faktor), amely segítségével az ontológiák fa struktúrájában minden szülő csomóponthoz tartozó súly rekurzívan definiálható a gyermek csomóponthoz tartozó súly és a terjedési faktor szorzataként:

$$w_{Parent} = \alpha \cdot w_{Child}$$

A terjedési faktor szemléletesen azt jelenti, hogy a szülő csomópont súlya hányadrésze a gyermek csomópont súlyának.

A fentiek alapján a ticketing rendszerből származó adatok feldolgozása a következő algoritmus alapján történik:

```
document = üres lista
szöveg = megnyit(text.csv)
ciklus a szöveg rekordjain
        a rekord szövegének elemzése
        kulcsszavak kinyerése
        a kinyert kulcsszavak keresése a taxonómia osztályok között
        ha van találat, akkor
                ciklus a találati lista szavain
                        releváns csúcsok hozzáfűzése
                ciklus vége
                concept vektor létrehozása
                a document lista bővítése az aktuális adatokkal
        elágazás vége
ciklus vége
KI: document (csv)
```

Ezt végrehajtva előáll a szöveges adatforrásból kinyert információ a következő formában:

DOCUMENT(DOC_ID, OPEN_DATETIME, CLOSE_DATETIME, PRIORITY, CREATOR, KEYWORDS, NODES, VECTOR, INPUT, TITLE), ahol

- DOC_ID a hibajegy (dokumentum) azonosítója
- OPEN_DATETIME a hibajegy létrehozásának dátuma és ideje
- CLOSE_DATETIME a hibajegy lezárásának dátuma és ideje (ha még nincs lezárva, akkor NULL értékű)
- PRIORITY a hibajegy prioritása, amely lehet alacsony, közepes vagy magas
- CREATOR a hibajegy létrehozója
- KEYWORDS a hibajegyben lévő kulcsszavak listája
- NODES a kulcsszavak leszármazottai
- VECTOR a hibejegyhez tartozó concept vektor
- INPUT a hibajegy szövege
- TITLE a hibajegy tárgya – általában egy soros szöveg

Ezen adatok alapján meghatározhatjuk bármely két hibajegy hasonlóságát, és az egymáshoz hasonló hibajegyeket osztályokba sorolhatjuk.

5.3.3 Dimenzionális tervezés

A lehetséges dimenziókat a meglévő adatok és a várható elemzési igények alapján kell meghatározni. Az előző prototípushoz hasonlóan itt is csak a hibajegy legfontosabb adataival dolgozunk. Később az adatok köre – így a létrehozott dimenzionális modell – tetszőlegesen bővíthető.

A hibajegy létrehozásának időpontja, mint dimenzió triviálisan adódik, mivel az egyes hibajegyeket tartalmát, vagy a hibajegyekkel kapcsolatos bármely mérőszámot (pl. hibajegyek száma) vizsgálhatunk az idő függvényében is. A létrehozási idő év.hónap.nap óra:perc formában áll rendelkezésre (pl: 2017.07.12 11:55). Ezt érdemes később két részre bontani, leválasztva ezzel a dátum (év.hónap.nap) és az idő (óra:perc) részeket. A dátum dimenzióhoz érdemes hierarchiát is létrehozni, legegyszerűben év-hónap-nap formában.

A második dimenzió lehet a hibajegy kategóriája, amely megfeleltethető a taxonómia fogalmaknak. A jelenlegi megoldás esetén egy hibajegyhez (mint dokumentumhoz) több kulcsszó is tartozhat, ezért a hibajegyek és a kategóriák között több a többhöz kapcsolat van, amely nem túl szerencsés egy OLAP-kocka esetén.

Ennek feloldására az egyik lehetőség a több a többhöz kapcsolat felbontása két egy a többhöz kapcsolatra egy új tábla közbeiktatásával. Ennél egyszerűbb megoldás (preferált), hogy a leendő ténytáblába a hibajegyet annyi példányban vesszük fel, ahány kulcsszót tartalmaz a megfelelő adatsor. Minden ilyen példány pontosan egy kulcsszót tartalmaz, így a kulcsszavaknak megfeleltetett kategória és a hibajegyek között egy a többhöz kapcsolat alakítható ki.

Alternatív lehetőség egy új kategória rendszer kialakítása a dokumentumok osztályba sorolásával (a concept vektor adatokat felhasználva). A jelenleg kétszintű taxonómia felhasználása esetén a kategória dimenzióhoz is lehet hierarchiát definiálni, pl. főkategória-alkategória felépítésben. Természetesen a hasonlóság alapú osztályba sorolás esetén is lehetséges a két- vagy többszintű kategória hierarchia kialakítása.

A harmadik dimenzió lehet az ügyfél, aki a hibát bejelentette. Első körben ez a dimenzió nem tartalmaz hierarchiát, de később ez még bővíthető, pl. cég-ügyfél séma szerint.

A negyedik dimenzió a hibajegy bizonyos adatait tartalmazza, konkrétan a hibajegy tárgyát és szövegét. Ez a dimenzió a tervek szerint nem tartalmaz hierarchiát. A későbbiekben kiegészíthető még számított adatokkal, pl. a hibajegy besorolása a szöveg hosszúsága szerint (rövid – közepes – hosszú), vagy (szemantikusan) hasonló hibajegyek száma stb.

Az ötödik dimenzió a hibajegy prioritása, amely a hibajegyek fontosság szerinti elemzéséhez szükséges. A prioritások rendszere egyszintű, ezért ez a dimenzió sem tartalmaz hierarchiát.

Az OLAP-kocka kezdeti verziójában csak két egyszerű mérték kerül definiálásra:

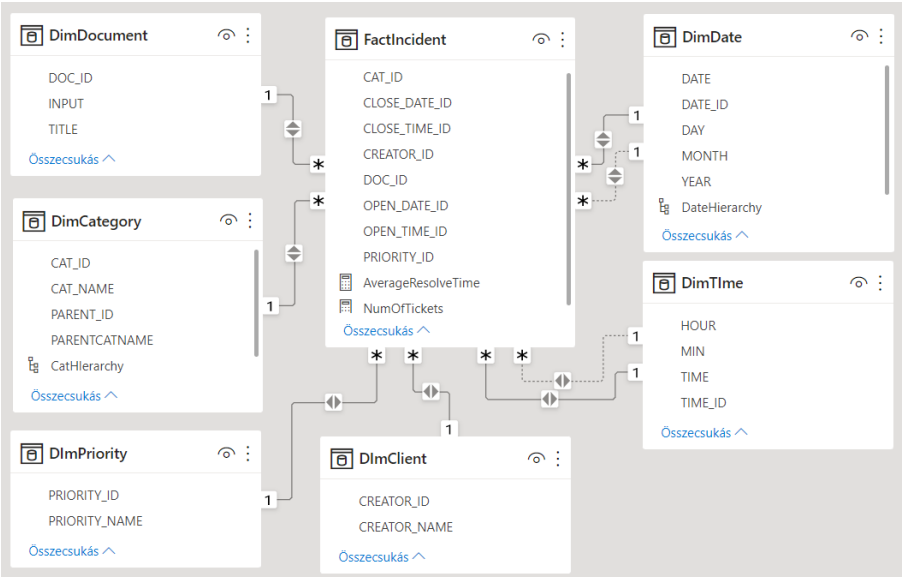
- Hibajegyek száma. Ez a mérték mindegyik dimenzió alapján összegezhető.
- Átlagos megoldási idő. Ez a mérték nem összegezhető.

Ezen mértékek minimális módosításával később több másik is definiálható, pl:

- Nyitott hibajegyek száma. Ez olyan incidenseket jelent, amelyekhez még nem tartozik lezárási dátum
- Lezárt hibajegyek száma. Ez a már megoldott eseteket számolja meg.

- Backlog-ban lévő hibajegyek száma. Ez egy bizonyos időnél régebbi (pl. több, mint kéthetes) megoldatlan eseteket számolja meg. Ha ez az érték magas, akkor feltehetően rendszerszintű problémák vannak, pl. a hibákat kezelő humán erőforrás kapacitás kevés a feladatok ellátásához.
- Minimális, illetve maximális megoldási idő. Ennek segítségével megadható egy intervallum, amelybe az adott szűrési feltételek mellett a megoldási idő beleesik.

A ténytábla az egyes dimenzió azonosítókat és a mértéket tartalmazza. Az elkészült csillagsémát a következő ábra szemlélteti.



39. Ábra: A ticketing ontológiák felhasználásával létrehozott csillagséma

Az OLAP-kocka ezen információk alapján – az 5.1.4-es részhez hasonló módon – már létrehozható.

5.4 Exploratory OLAP prototípus második verziójának elkészítése

Ebben az alfejezetben az Abello (2015) ötletét felhasználó exploratory OLAP prototípus megtervezéséről és kifejlesztéséről lesz szó. A kivitelezés a 3.3 pontban leírtak alapján történik a 4.3 pontban kifejlesztett ontológia 2. verziójának felhasználásával.

5.4.1 Adatok előkészítése

A probléma szempontjából releváns forrásadatok a következők:

- A hibajegyek adatai (Azonosító, megnyitás ideje, prioritás stb.)
- Hibajegy kategória adatok

- A hibajegyekhez köthető tevékenységek adatai (Tevékenység kezdetének dátuma, tevékenység típusa stb.)
- A tevékenységeket elvégző informatikusok adatai
- A hibajegyet létrehozó ügyfél adatai
- SLA adatok

Az eredetileg Excel/CSV formátumú adatokat egy relációs sémában helyeztem el. A séma tartalmazza a szöveges adatokat is (hibajegyek tárgya és részletes leírása).

5.4.2 A funkcionális függőségi fa

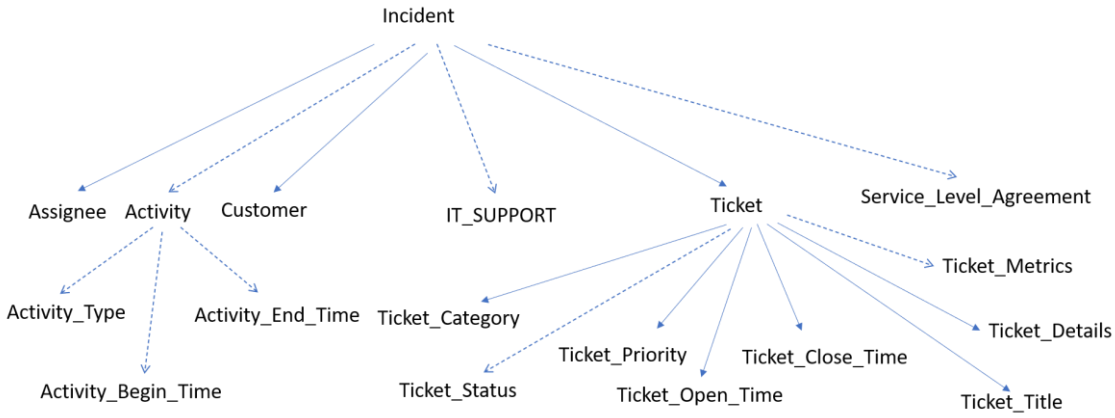
A ticketing ontológia fogalmai közül a legfontosabb az Incident fogalom. A prototípus elkészítésénél csak ezzel foglalkozom, a többi fogalom esete ehhez hasonlóan tárgyalható. Az Incident fogalomhoz tartozó FD-fa előállításához azt kell meggondolni, hogy egy adott hibabejelentés (incident) - az ontológiát figyelembe véve - milyen más fogalmakkal van függőségi viszonyban. Ez alapján tizenhét fogalmat találtam: Ticket, Ticket Status, Ticket Priority, Ticket Open Time, Ticket Close Time, Ticket Details, Ticket Metrics, Ticket Title, Assignee, Activity, Activity Type, Activity Begin Time, Activity End Time, Customer, IT Support, Service Level Agreement.

A legfontosabb függőségi kapcsolatok a következők:

- Az egyes esetekhez az IT Support hibajegyet rendel, azaz {Incident} → {Ticket}
- Minden esethez tartozik egy ügyfél (a bejelentő), vagyis {Incident} → {Customer}
- Minden esethez hozzárendelhető (az egyszerűség kedvéért csak egy) személy, aki foglalkozik az esettel, emiatt {Incident} → {Assignee}
- Minden esethez társíthatók a kapcsolódó tevékenységek, amelyeket a hibát javító informatikus végez, ezért {Incident} → {Activity}
- Végül minden esetben van egy szolgáltatási szint megállapodás az ügyfél és a cég között, azaz {Incident} → {Service Level Agreement}

Az előzőek alapján a fa már könnyen felépíthető. Az első szinten találhatók a legfontosabb függőségek, a második szint az első szinttől funkcionálisan függő fogalmakat tartalmazza. Pl. az Activity esetén ilyenek: Activity Type, Activity Begin Time és Activity End Time.

Az egyszerűség kedvéért csak a fa első két szintjét ábrázoltam.



40. ábra: Az Incident fogalomhoz tartozó FD-fa

A dimenzionális tervezéssel való összehasonlíthatóság miatt (ld. 5.3.3 alfejezet) azonban a szaggatottal jelölt fogalmakat nem vettem figyelembe. Ezen kívül a Ticket fogalom is kiesett, mivel nem volt megfeleltethető az adatforrás adatainak. Emiatt a Ticket_Category, Ticket_Priority … Ticket_Details fogalmak a második szintről az elsőre kerültek. Az Incident fogalommal kapcsolatba hozható fogalmak száma így nyolcra csökkent.

5.4.3 A multidimenzionális azonosítók előállítása

A lehetséges multidimenzionális azonosítók megtalálásához a 16. ábrán látható módszert alkalmazom. Legyen L halmaz a lehetséges azonosítók halmaza. Definiáljunk még két (kezdetben üres) halmazt. Nevezetesen I legyen a lehetséges MD azonosítók halmaza, C pedig az MD azonosítóknak nem megfelelő azonosítók halmaza. A lehetséges esetek számának csökkentéséhez vegyük figyelembe a következőket:

- a keresett fogalomkészleteknek minimális halmazoknak kell lenniük.
- egy adott fogalomhalmazon belül sem lehetnek funkcionális függőségek.
- ha egy fogalomhalmaz nem azonosítja egyértelműen az eredeti fogalmat, akkor az összes olyan fogalomhalmaz is kizárható a keresésből, amely az adott fogalomhalmaztól funkcionálisan függő fogalmakat tartalmaz

A fennmaradó azonosító jelöltek közül azokat választjuk ki, amelyek esetén az azonosítandó fogalom (Incident) előfordulásainak száma (számosság, kardinalitás) kisebb vagy egyenlő, mint a kombinációban lévő fogalmak kardinalitásainak szorzata.

Az első lépésben L = {{Assignee}, {Customer}, {Ticket_Category}, {Ticket_Priority}, {Ticket_Open_Time}, {Ticket_Close_Time}, {Ticket_Title}, {Ticket_Details}}. Ezek mindegyike triviálisan megfelel az előbb felsorolt három követelménynek. A lépés végén az I halmaz üres lesz, mivel egyik fogalom sem felel meg MD azonosító jelöltnek. Ez utóbbit a kardinalitásra vonatkozó feltétel vizsgálatával láthatjuk be. Például az Assignee fogalom esetén a következő lekérdezést kell végrehajtani:

```
SELECT COUNT(DISTINCT Assignee)
FROM forras_adatok
```

A kapott eredmény (220) jóval kisebb, mint az Incidens kardinalitása (3073).

Az első lépés végén a C halmaz a következő elemekből fog állni: C = {{Assignee}, {Customer}, {Ticket_Category}, {Ticket_Priority}, {Ticket_Open_Time}, {Ticket_Close_Time}, {Ticket_Title}, {Ticket_Details}}.

Az L halmaz a C elemeinek olyan kételemű kombinációit tartalmazza, amelyek között nincsen funkcionális függőségi kapcsolat. Ez utóbbi feltétel teljesülése az FD-fáról leolvasható.

A második lépésben alkalmazott kételemű kombinációkat egy sql script segítségével lehet előállítani. A script végigfut az összes kombináción, és kiválasztja közülük a lehetséges MD azonosító jelölteket.

Fontos, hogy a függőségi fa fogalmait itt a megfelelő forrásadatokkal helyettesítsük.

A forrásadatok felépítése korábban a következő volt:

DOCUMENT(DOC_ID, OPEN_DATETIME, CLOSE_DATETIME, PRIORITY, CREATOR, KEYWORDS, NODES, VECTOR, INPUT, TITLE).

Itt a fogalom – forrásadat leképezés miatt nincs párja a DOC_ID, a KEYWORDS, NODES és a VECTOR mezőknek, ezért ezek most kimaradnak. Ezenkívül hiányzik a Category és az Assignee mező is, viszont a forrásadatok kibővített verziójában ezek rendelkezésre állnak.

Az egyes kombinációk előállítása a következő script segítségével történik:

```
create table #fogalmak
(
   fogalom nvarchar(50)
)
insert into #fogalmak values ('Assignee')
insert into #fogalmak values ('Creator')
insert into #fogalmak values ('Category')
insert into #fogalmak values ('Priority')
```

```
insert into #fogalmak values ('Open_Datetime')
insert into #fogalmak values ('Close_Datetime')
insert into #fogalmak values ('Title')
insert into #fogalmak values ('Input')

go

select f.fogalom as 'fogalom1',
       f2.fogalom as 'fogalom2'
into kombinaciok
from #fogalmak f cross join #fogalmak f2
where f.fogalom < f2.fogalom


drop table #fogalmak
```

A kombinációk számát 7 fogalom esetén könnyedén meghatározhatjuk, ennek értéke

$$\binom{7}{2} = 28$$

Majd következik egy ciklus, amelyben kiszűrjük a kardinalitási feltételnek megfelelő kombinációkat:

```
declare @f1 nvarchar(100)
declare @f2 nvarchar(100)
declare @sql nvarchar(1000) = ''
declare @outputparam1 int
declare @outputparam2 int
declare @count int

select @count = COUNT(*) from forras_adatok

declare cs cursor for
select fogalom1, fogalom2 from kombinaciok

open cs
fetch next from cs into @f1, @f2
while @@FETCH_STATUS = 0
begin
  select @f1, @f2
  set @sql = N'SELECT @id_out = count(*) from
             (select distinct ' + @f1 +' from forras_adatok) f'
  EXEC sys.sp_executesql @sql, N'@id_out INT OUT', @outputParam1 OUT;
  set @sql = N'SELECT @id_out = count(*) from
             (select distinct ' + @f2 +' from forras_adatok) f'
  EXEC sys.sp_executesql @sql, N'@id_out INT OUT', @outputParam2 OUT;
  select iif(@outputparam1 * @outputparam2 >= @count, 'Lehetséges', 'Nem
   lehetséges')
  fetch next from cs into @f1, @f2
end

close cs

deallocate cs
```

A kapott eredmény alapján a 28 kombinációból 18 MD azonosító jelöltet kapunk. A lehetséges kombinációk száma tovább csökkenthető, ha csak azokat a fogalompárokat vesszük figyelembe, amelyek együttes kardinalitása közel áll az Incident fogalom kardinalitásához (3073). Az együttes kardinalitás értéke az előző számításhoz hasonlóan adódik, csak a select utasításnál a distinct részben mindkét fogalomnak szerepelnie kell, pl:

```
select count(*)
from (
select distinct Assignee, Open_DateTime
from forras_adatok) f
```

A legnagyobb kardinalitású fogalompárokat (az eredeti fogalmaknak megfelelően) a következő táblázat mutatja:

Fogalom1	Fogalom2	Együttes kardinalitás
Ticket_Details	Ticket_Open_Date	3073
Assignee	Ticket_Open_Date	3073
Customer	Ticket_Open_Date	3073
Ticket_Open_Date	Ticket_Close_Date	3073
Ticket_Category	Ticket_Open_Date	3072
Ticket_Title	Ticket_Close_Date	3071
Ticket_Title	Ticket_Details	3054
Ticket_Details	Ticket_Close_Date	3053

4. táblázat: Lehetséges MD azonosítók Datetime adattípus esetén

Látható, hogy a legtöbb esetben a Ticket_Open_Date szerepel a fogalmak között. Ez nem meglepő, mivel ez DateTime típusú adat (másodperc pontossággal). Egy adott időpontban kis valószínűséggel fordulhat elő, hogy két hibajegyet nyitnak.

Az előző táblázatból kiválogatva a szemantikusan is megfelelőket az MD azonosító jelöltek halmaza az eredeti 28 elemről 4-re szűkült:

I = {{Ticket_Open_Date, Ticket_Details}, {Ticket_Open_Date, Assignee}, {Ticket_Open_Date, Customer}, {Ticket_Open_Date, Ticket_Close_Date}}

A kereső algoritmus itt (a második lépésben) véget is ér, mivel a háromelemű kombinációkra már nincsen szükség. Mivel ezen kombinációk együttes kardinalitása megegyezik az adatforrás sorainak számával, emiatt ezek megfelelnek azonosítóknak.

Megjegyzések:

- A Ticket_Open_Datetime elvileg önmagában is lehetne MD azonosító. A gyakorlatban viszont – a nagyszámú hibajegy miatt – mindig vannak olyan esetek, ahol a jegy nyitásának időpontja másodpercre megegyezik. Egy másik fogalommal párosítva viszont a datetime típusú adat már megfelel MD azonosítónak. A másik lehetőség a pontosság növelése, azaz a másodperc helyett nagyobb pontossággal vennénk figyelembe az időt.
- A Ticket_Close_Date nincs mindig kitöltve. Emiatt csak a lezárt hibajegyeknél lehet része az MD azonosítónak.
- A {Ticket_Details, Ticket_Title}is ígéretesnek tűnik, de nem teljesíti az azonosítónak szükséges feltételt. Vannak ugyanis olyan sorok, ahol ezek értékei egyformák. Ez nem duplikációból adódik, hanem gyakori (rövid) hibáknál jöhet létre. Pl. Title = ’Nyomtatási probléma’, Details = ’Nem tudok nyomtatni’.
- Ha csak Date típusú ticket adataink lennének, akkor szükséges lenne a háromelemű (esetleg négyelemű) kombinációk vizsgálata is. DateTime helyett Date típusú adatokkal is lefuttattam a lekérdezést. Ebben az esetben a lehetséges MD azonosítók a következő táblázatban láthatók:

Fogalom1	Fogalom2	Együttes kardinalitás
Ticket_Details	Ticket_Title	3054
Assignee	Ticket_Title	2995
Assignee	Ticket_Details	2695
Customer	Ticket_Details	2963
Customer	Ticket_Open_Date	2952
Assignee	Ticket_Open_Date	2952
Ticket_Open_Date	Ticket_Close_Date	2952

5. táblázat: A lehetséges MD azonosítók Date adattípus esetén

5.4.4 Tények és dimenziók előállítása

A dimenziójelöltek meghatározásához érdemes lekérdezni, hogy melyik fogalomnak milyen elemszámú az értékkészlete. Dimenziónak a legkisebb elemszámú, nem numerikus fogalmak lehetnek alkalmasak.

A lekérdezéshez az együttes kardinalitásokat meghatározó script leegyszerűsített változata alkalmazható.

A következő táblázat elemszám szerint növekvő sorrendben mutatja a fogalmakat.

Fogalom	Elemszám
Ticket_Priority	5
Ticket_Category	35
Assignee	63
Customer	242
Ticket_Close_Time	2556
Ticket_Title	2926
Ticket_Details	2947
Ticket_Open_Time	3072

6. táblázat: Az egyes fogalmak értékkészletének elemszáma

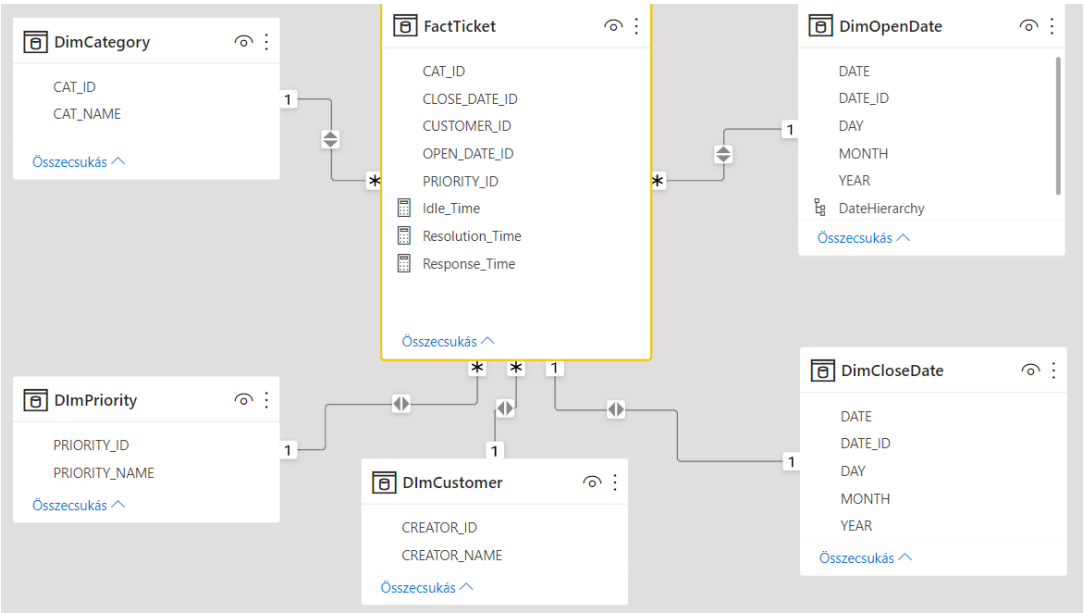
A táblázat alapján dimenziójelöltek lehetnek: Ticket_Priority, Ticket_Assingee, Category és Customer.

A mértékek elsősorban a numerikus fogalmak közül kerülnek ki. Esetünkben a függőségi fa első két szintjén csak egy ilyen fogalom van, ez pedig a Ticket_Metrics. Ez nyilván a hibajegyhez rendelhető mértékek összességére utal. A ticketing ontológia második verziójában megtalálhatók a Ticketing_Metrics alosztályai, ezek: Idle_Time, Resolution_Time, Response_Time.

A mértékekre szükség van a tényjelöltek felfedezésekor. A tényjelöltek azonosításához minden fogalomra ki kell számítani az f = D +2M függvény értékét, ahol D a fogalomhoz köthető dimenziók, M pedig a mértékek száma. A függőségi fa (ld. 40. ábra) alapján egyértelmű, hogy a Ticket fogalomhoz tartozik a legtöbb dimenzionális fogalom, ráadásul mindegyik mérték is ehhez a fogalomhoz rendelhető. Emiatt esetünkben a Ticket lesz az elsődleges tényjelölt.

Az OLAP-séma felépítéséhez szükség van még az összegzési utak, ezáltal a dimenzió hierarchiák meghatározására. A kijelölt MD azonosítók alapján ({{Ticket_Open_Date, Ticket_Details}, {Ticket_Open_Date, Assignee}, {Ticket_Open_Date, Customer}, {Ticket_Open_Date, Ticket_Close_Date}}) dimenzió hierarchia képezhető a Ticket_Open_Date és a Ticket_Close_Date fogalmakból. Ezek természetesen adódnak pl. év-hónap-nap felépítésben. A gyakorlatban egyetlen dátum (esetleg külön idő) dimenzió is betöltheti ugyanezt a szerepet.

A következő ábra a fentiek alapján elkészített OLAP-sémát szemlélteti.



41. ábra: A 2. exploratory OLAP prototípus sémája

Az elkészült sémát a következő alfejezet második részében értékelem.

5.4.5 Az elkészült exploratory OLAP rendszer értékelése

Először a ticketing ontológia második verzióját értékelem a 3.5 és a 4.4 alfejezetben leírtak alapján.

- Tartalmi szempontból az elkészült ontológia csak részben írja le a helpdesk/ticketing területet. Elsősorban a hibabejelentések (incidensek) vannak a fókuszban, azon belül is az egyes hibajegyek tulajdonságai, valamint az életciklusuk fontosabb állomásai.
- A tudásábrázolást tekintve az ontológia a megcélzott területen képes a legfontosabb tényeket kifejezni fogalmak, kapcsolatok, axiómák és szabályok formájában. Az ontológia mérete és komplexitása alapján könnyen értelmezhető a felhasználók számára.
- Technikai szempontból az alkalmazott fejlesztési módszertan (On-To-Knowledge) az ajánlott módszertanok között szerepel (ld. 4.3). Az ontológia fejlesztésénél használt eszköz (Protégé) is kellően korszerűnek mondható (Singh & Anand, 2013). A kivitelezés módja lehetett volna talán automatizáltabb, azaz kevesebb emberi beavatkozást igénylő.

- Az ontológia teljes mértékben használhatónak bizonyult az exploratory OLAP prototípus fejlesztésénél, ahol elsődleges szerepe a dimenzionális fogalmak azonosításának elősegítése volt.
- Az ontológia jelenlegi formájában nem újrahasznosítható, mivel egy konkrét alkalmazáshoz készült. Alapjául szolgáltat viszont egy későbbiekben kifejlesztendő szakterületi ontológiának a helpdesk/ticketing területen.

A multidimenzionális sémát először a 3.5 alfejezetben[1] leírt szempontok alapján vizsgáltam meg. Ennek során a következő megállapításokra jutottam:

- Ez a séma is redundanciamentes, nem tartalmaz felesleges attribútumokat.
- A kicsinyítési-nagyítási képesség értéke a dátum jellegű dimenziók esetén (4) alapján Z= $\frac{\left(1-\frac{1}{3}\right)}{1} = \frac{2}{3}$.
- A séma egyetlen ténytáblát és három mértéket tartalmaz. Mivel mindegyik mérték egy ténytáblához tartozik, ezért a ténytábla lokális gazdagságának értéke (5) szerint 1.
- A séma egyszerűségét (6) alapján határozhatjuk meg. Mivel egy ténytáblánk van, és a dimenzió hierarchiák 3 szintesek, valamint 5 kapcsolat van a modellben, ezért
  $S = \frac{1+3}{1+3+5} = \frac{4}{9}$
- A lekérdezések során hiba nem adódott, ezért (7) miatt a helyesség értéke C=1.
- A kapott sémában vannak aggregációk, ezért nagyobb adatmennyiségnél előfordulhatnak teljesítmény gondok. A séma viszont nem tartalmaz strukturálatlan szöveget, így annak lekérdezése nem okoz külön problémát.

Tehát egy viszonylag egyszerű sémát kaptunk, amelyet érdemes összehasonlítani az 5.3.3 pontban elkészített referencia prototípussal. A referencia prototípus sémája ugyanis dimenzionális tervezéssel jött létre, ezért ideális esetben a két sémának közel azonosnak kellene lennie. A sémákat összehasonlítva viszont a következő eltéréseket lehet észrevenni:

- A ténytáblák közötti legnagyobb különbség abban van, hogy azok a referencia séma esetén az Incident, míg az exploratory OLAP séma esetén a Ticket fogalomhoz köthetők.

[1] a zárójelben a 3.5 fejezet egyes kifejezéseinek azonosítói láthatók

- A dimenziótábláknál az exploratory OLAP modell nem találta meg a Category dimenziót. Ezzel egy lehetséges összesítési út és egy hierarchia is elveszett (kategória – alkategória). Dátum dimenzióból viszont kettő is adódott.
- A kereső algoritmus szintén nem azonosított olyan dimenziót (pl. dimTicket), amely a hibajegy leíró adatait tartalmazta volna. Emiatt fontos szöveges információk maradtak ki a sémából.

Az elkészült exploratory OLAP prototípus hiányosságainak több oka is van. Az első, hogy ez még csak a legelső változat, amelynél a kivitelezhetőség volt az elsődleges szempont. A prototípus továbbfejlesztésével várható, hogy a kapott eredmény is relevánsabb lesz. A következő hibalehetőség a folyamat azon lépéseinél van, amelyek manuális beavatkozást (döntést) igényelnek. Ilyen például a multidimenzionális azonosítók előállítása, ahol a viszonylag magas együttes kardinalitású {Ticket_Title, Ticket_Details} fogalompárt elvetettük. Emiatt ezek az adatok nem épültek be a modellbe. A harmadik ok lehet, hogy a felhasznált ontológia nem teljes mértékben írja le a ticketing rendszer incidenseit. Ezáltal a forrásadat – ontológia fogalom leképezés nem minden adat esetén valósítható meg.

A hiányosságok ellenére úgy látom, hogy az elkészített prototípus megfelel egy „nulladik” dimenzionális modellnek. Azaz, egy olyan alapot adhat, amelyre építve az OLAP-séma lényegesen könnyebben létrehozható, mintha a folyamatot teljesen az alapoktól kezdenénk.

6 ÖSSZEFOGLALÁS

Az adatok mennyisége manapság soha nem látott mértékben nő. Ez jórészt a nem vagy félig strukturált adatoknak köszönhető. Ezek közül is fontos szerepet játszanak a szöveges adatok. Az adattárházakra épülő elemzések hatékonyabbá tehetők, ha a szöveges adatokban lévő információtartalmat is figyelembe vesszük. Ezt a gyakorlatban úgy lehet megvalósítani, hogy egy szemantikus réteget építünk be az adattárházba, amely segítségével a szöveges adatok jelentése strukturált formában feldolgozható.

Az exploratory OLAP rendszerek erre a problémára adnak megoldást. Bár fogalmi szinten több ilyen koncepció létezik, konkrét megvalósítás általában nem tartozik hozzájuk. Prototípus leírás vagy esettanulmány néhány esetben elérhető, de kérdéses, hogy ezek alapján lehet-e általános következtetéseket levonni ezekkel a rendszerekkel kapcsolatban.

Kutatásom fő célkitűzése az exploratory OLAP fogalmi modellezése, lehetséges prototípusos megvalósítása és a validálási eljárások kidolgozása volt.

A kutatás első lépéseként a probléma elméleti hátterének feltérképezésével foglalkoztam. Ebben a részben először az adattárházakkal kapcsolatos alapfogalmakat, illetve modelleket és technológiákat mutattam be, részletesen kitérve a mai adattárházakat érintő problémákra is. Utána a szemantikus adattárház réteg létrehozásában fontos szerepet játszó ontológiák felépítését, létrehozási és alkalmazási lehetőségeit ismertettem.

A következő lépésben elemeztem a szakirodalomban jelenleg elérhető exploratory OLAP rendszerek felépítését. Ezek közül Prasad (2010) és Abello (2015) javaslatait emelném ki, amelyek eltérő módon közelítik meg ugyanazt a problémát. Prasad (2010) megoldása egyszerűbb, viszont a csillagséma és ezáltal az OLAP-kocka szerkezete manuálisan definiált. Abello (2015) rendszere jóval bonyolultabb, viszont a megoldás is általánosabb érvényű, mivel ebben a megoldásban az OLAP-séma (részben) automatikusan állítható elő.

A nehézségeket növeli, hogy előfeltételként szükség van az adott területi ontológiákra, amely nem minden esetben áll rendelkezésre. Az Abello (2015) -féle rendszer megvalósíthatóságának kritikus pontja az ontológia és a multidimenziós séma megfeleltetése. További vizsgálat tárgyát képezi, hogy ez milyen mértékben és milyen korlátozásokkal valósítható meg.

A kutatási kérdések megfogalmazása és az alkalmazandó módszerek bemutatása után először egy prototípust készítettem el Prasad (2010) ötlete alapján. Ezután az alkalmazott adatforráshoz (ticketing rendszer) tartozó ontológiák nulladik verziójának elkészítése következett. Végül egy manuálisan definiált OLAP-kockát terveztem meg, amely referenciaként szolgált az Abello (2015) megoldása alapján elkészítendő prototípushoz.

Az első kutatási kérdésre (hogyan integrálható szemantikus réteg az adattárházba) a 3.3 alfejezetben leírt modellek adhatják meg a választ, amennyiben a gyakorlatban is működőképesnek bizonyulnak. A jelenlegi állapot szerint a Prasad (2010) ötletén alapuló modell működőképesnek látszik, de mindezt további kutatásokkal kell még alátámasztani. Ennek a modellnek viszont a hatóköre meglehetősen korlátozott, mivel egy előre definiált OLAP-sémát használ. Az Abello (2015) koncepciójára épített modellnél a kutatás eddigi eredményei alapján feltételezhető, hogy az bizonyos egyszerűsítésekkel és megszorításokkal megvalósítható. A modell határainak pontos feltérképezése szintén további kutatásokat igényel.

A második kutatási kérdésre (hogyan lehet a modellt a gyakorlatban megvalósítani) két prototípus elkészítésével adtam meg a választ. Ezek közül az első Prasad (2010), a második pedig Abello (2015) ötletét felhasználva készült el. Az első prototípus felépítése előre definiált, emiatt ennek megvalósítása egyszerűbb volt. A második prototípus elkészítése komplexeb, számos nyitott kérdéssel, viszont a dinamikusan létrejövő séma miatt a gyakorlatban jobban használható.

A harmadik kutatási kérdésre (exploratory OLAP rendszerek validálása) a 3.5 fejezetben adtam meg a választ egy validáló/értékelő szempontrendszer megadásával. Ezeket a szempontokat alkalmaztam az elkészült exploratory OLAP prototípusok validálásánál is.

Fontos megjegyezni, hogy a kutatás során elsősorban a megvalósíthatósági szempontokra helyeztem a hangsúlyt. Emiatt szükség esetén egyszerűsítettem, módosítottam a kiinduló modelleken. Szintén nem volt cél a teljesen automatizált működés.

A kutatást a későbbiekben elsősorban ezen két korlátozás csökkentésével szeretném folytatni. Egyrészt szeretném a meglévő prototípusokat továbbfejleszteni, és tesztelni őket különféle tartalmú és méretű adatforrásokon. Másrészt fontos lenne a prototípusok elkészítéséhez szükséges emberi beavatkozások minimalizálása. Ezeken kívül további kutatási irány lehet az elkészített ticketing ontológia karbantartásának problémája.

7 IRODALOMJEGYZÉK

Abelló, A., 2015. In: *Using Semantic Web Technologies for Exploratory OLAP: A Survey.* hely nélk.:IEEE.

Abello, A. & Romero, O., 2010. A framework for multidimensional design of data warehoses from ontologies. In: hely nélk.:Data Knowl. Eng. Vol 69, no. 11, pp. 1138-1157.

Abramson, I., 2004. In: *Data Warehouse: The Choice of Inmon versus Kimball.* hely nélk.:IAS Inc, pp. 28-30.

Ali, A. A. & Mohamed, W. M., 2016. In: *Monitoring Business Transactions for a Real-time Data Warehouses.* hely nélk.:International Journal of Computer Applications.

Anon., dátum nélk. CA Service Desk Manager. In: hely nélk.:https://www.ca.com/us/products/ca-service-desk-manager.html.

Anonymous, 2020. *Structured vs. Unstructured Data – Best Thing You Need To Know.* [Online]
Available at: https://prowebscraper.com/blog/structured-vs-unstructured-data-best-thing-you-need-to-know/

Asim, M. N., Wasim, M. & Usman, M., 2018. A survey of ontology learning techniques and applications. *Database,* 2018. kötet.

Bánné Varga, G., 2012. In: *Az adattárház készítés technológiája.* hely nélk.:Typotext, pp. 19-20, 23, 3.fejezet.

Batini, C., Ceri, S. & Navathe, S., 1992. *CONCEPTUAL DATABASE DESIGN: An Entitiy-Relationship Approach.* Redwood City, California: The Benjamin/Cummings Publishing Company, Inc..

Batra, R., 2018. *SQL Primer - An Accelerated Introduction to SQL Basics.* hely nélk.:Apress.

Berners-Lee, T., 2001. The Semantic Web. In: hely nélk.:Scientific American.

Bhandari, P., 2020. *Scribbr.* [Online]
Available at: https://www.scribbr.com/statistics/descriptive-statistics/

Bicevskaa, Z. & Oditis, I., 2017. Towards NoSQL-based Data Warehouse Solutions. *Procedia Computer Science,* 104. kötet, pp. 104-111.

Biemann, C., 2005. Ontology Learning from Text: A Survey of Methods. In: hely nélk.:LDV Forum, pp. 1-5.

Bourgeois, D., 2019. *Information Systems for Business and Beyond (2019).* hely nélk.:Pressbooks.

Bousdekis, A., 2020. Prescriptive analytics: Literature review and research challenge. *International Journal of Information Management.*

Brinkmann, B., 2019. *Descriptive, Predictive, and Prescriptive Analytics: A Practical Introduction.* [Online]
Available at: https://www.logianalytics.com/predictive-analytics/comparing-descriptive-predictive-prescriptive-and-diagnostic-analytics/

Browniee, J., 2017. A Gentle Introduction to the Bag-of-Words Model. In: hely nélk.:Deep Learning for Natural Language Processing.

C. Salley, E. F. C., 1998. Providing OLAP to User-Analysts: An IT Mandate. *Computer Science,* pp. 12-17.

Calvanese, D., Kíharlamov, E. & Thorne, C., 2008. Aggregate queries over ontologies. In: hely nélk.:Proc. Int. Workshop Ontologies Inform. Syst. Semantic Web, pp. 97-104.

Codd, E., Codd, S. & Salley, C., 1993. *Providing OLAP (On-line Analytical Processing).* hely nélk.:Codd & Associates.

Cuzzocrea, A., Bellatreche, L. & Song, I.-Y., 2015. In: *Data Warehousing and OLAP over Big Data: Current Challenges and Future Research Directions.* hely nélk.:International Journal of Business Process Integration and Management.

d'Aquin, M. & Noy, N. F., 2018. Where to Publish and Find Ontologies?. *Journal of Web Semantics,* pp. 1-10.

Demidova, A. V., 2018. In: *Anastasiya V. Demidova, Yevgeny A. Kuznetsov, MDesigning Multidimensional Information Systems Using the Data Vault Methodology.* hely nélk.:ismeretlen szerző

Dixon, M., 2019. *The challenges of analysing unstructured data.* [Online]
Available at: https://seleritysas.com/blog/2019/08/27/the-challenges-of-analysing-unstructured-data/

Fensel, D., Davies, J. & Van Harmelden, F., 2003. *TOWARDS THE SEMANTIC WEB: Ontology-driven knowledge management.* Chichester: JOHN WILEY & SONS, LTD.

Frias, L., Queralt, C. & Olivé Ramon, A., 2003. *EU-Rent car rentals specification,* hely nélk.: ismeretlen szerző

Gábor, A., szerk., 1997. *Információmenedzsment.* hely nélk.:Aula kiadó.

Goes, P., 2014. Editor's comments: Big data and IS research.. In: hely nélk.:MIS Quarterly,, pp. 38(3), iii-viii.

Gomez, A., Corcho, O. & Fernandez-Lopez, M., 2002. Methodologies, tools and languages for building ontologies. In: hely nélk.:Data & Knowledge Engineering 46 (2003), pp. 41-64.

Gruber, T. R., 1993. A Translation Approach to Portable Ontology Specifications. In: hely nélk.:Knowledge acquisition, Vol. 5 No. 2 , pp. 199-200.

Guarino, N., 1995. Formal Ontology, Conceptual Analysis and Knowledge Representation: The Role of Formal Ontology in the Information Technology. *International Journal of Human-Computer Studies,* 43. kötet, pp. 625-640.

Haghighi, P. D., Burstein, F., Zaslavsky, A. & Arbon, P., 2013. Development and evaluation of ontology for intelligent decision support in medical emergency management for mass gatherings.. *Decision Support Systems,* 54. kötet.

Ibragimov, D., Hose, K., Pedersen, T. B. & Zimány, E., 2015. In: *Towards Exploratory OLAP Over Linked Open Data – A Case Study.* hely nélk.:International Workshop on Business Intelligence for the Real-Time Enterprise.

Inmon, W., 2005. *Building the data warehouse.* hely nélk.:John Wiley & sons.

Inmon, W. H., 2002. In: *Building the Data Warehouse.* hely nélk.:Wiley.

Iqbal, R., Murad, A. & Mustapha, A., 2013. An Analysis of Ontology Engineering Methodologies: A Literature Review. In: hely nélk.:Faculty of Computer Science and Information Technology, Universiti Putra Malaysia, pp. 1-8.

James A. O'Brien, G. M. M., 2010. Management Information Systems. In: hely nélk.:McGraw-Hill/Irwin, pp. 41-46, 431-441.

Joshi, P., 2018. [Online]
Available at: https://www.analyticsvidhya.com/blog/2018/11/introduction-text-summarization-textrank-python/

Kimball, R. & Ross, M., 2013. In: *The Data Warehouse Toolkit Third Edition.* hely nélk.:Wiley, pp. 18-20, 28-29.

Klein, A., 2017. *Hard Drive Cost Per Gigabyte.* [Online]
Available at: https://www.backblaze.com/blog/hard-drive-cost-per-gigabyte/#:~:text=From%20January%202015%20to%20January,the%20cost%20of%20providing%20storage.

Kotis, K. I., Athanasakis, I. & Vouros, G. A., 2014. Semantic Integration & Single-Site Opening of Multiple Governmental Data Sources. *Computer Science,* 02 06.

Kő, A. & Gillani, S., 2019. A Research Review and Taxonomy Development for Decision Support and Business Analytics Using Semantic Text Mining. *International Journal of Information Technology & Decision Making.*

Kő, A. & Lovrics, L., 1997. Döntéstámogató rendszerek. In: A. Gábor, szerk. *Információmenedzsment.* hely nélk.:Aula kiadó, pp. 423-523.

Kő, A. & Lovrics, L., 2000. Vezetői információs rendszerek. In: *Válogatott fejezetek az információmenedzsment témaköréből.* Budapest: BKÁE Információrendszerek Tanszék, pp. 159-189.

Kő, A., Lovrics, L. & Sántáné-Tóth, E., 2008. Döntéshozatal és döntéstámogatás. In: *Sántáné-Tóth, Edit (szerk) Döntéstámogató rendszerek.* hely nélk.:Panem Kiadó, pp. 51-90, 40.

Leung, K. N., 2012. *Towards an Ontology-based Knowledge Management: An Ontology Mediation Framework to Reconcile Inter-organizational Knowledge.* Republic of Moldova, Chisinau-2068: LAP LAMBERT Academic Publishing.

Liang, X., 2018. [Online]
Available at: https://towardsdatascience.com/textrank-for-keyword-extraction-by-python-c0bae21bcec0

Liu, H. & Wang, P., 2014. Assessing Text Semantic Similarity Using Ontology. *Journal of Software,* 9(2), pp. 490-496.

Luqi, F. K. a., 2002. An Introduction to Rapid System Prototyping. *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING,* 28(9), pp. 817-820.

Motik, B., 2012. Representing and querying validity time in RDF and OWL: A logic-based approach. In: hely nélk.:J. Web Sem., pp. 3-21.

Nebot, V., Berlanga, R. & Pérez, J. M., 2009. *Multidumensional Integrated Ontologies: A Framework for Designing Semantic Data Warehouses.* [Online].

Neumayr, B., Anderlik, S. & Schrefl, M., 2012. Towards ontologybased OLAP: Datalog-based reasoning over multidimensional. In: hely nélk.:Proc. 15th Int. Workshop Data Warehousing OLAP, pp. 41-48.

Pâslaru-Bontaş, E., 2007. A Contextual Approach to Ontology Reuse: Methodology, Methods and Tools for the Semantic Web. In: hely nélk.:REFUBIUM - FREIE UNIVERSITÄT BERLIN, p. 6. fejezet.

Phoebe Wong, R. B., 2019. *Everything a Data Scientist Should Know About Data Management.* [Online]
Available at: https://www.kdnuggets.com/2019/10/data-scientist-data-management.html

Poggi, A., Lembo, D., Calvanese, D. & Giacomo, G. D., 2008. Linking data to ontologies. In: hely nélk.:J. Data Semantics, Vol 10, pp. 133-173.

Prasad, K. S. N., 2010. In: *Text Analytics to Data Warehousing.* hely nélk.:Kalli Srinivasa Nageswara Prasad: Text AInternational Journal on Computer Science and Engineering,.

Prat, N. & Cherfi, S., 2003. Multidimensional Schemas Quality Assessment. In: hely nélk.:ESSEC Research Center.

Revert, F., 2018. *An overview of topics extraction in Python with LDA.* [Online]
Available at: https://towardsdatascience.com/the-complete-guide-for-topics-extraction-in-python-a6aaa6cedbbc

Romero, O. & Abello, A., 2012. Ontology driven search of compound IDs. In: hely nélk.:Knowl. Inform. Syst., vol. 32, pp. 191-216.

Sharda, R., Delen, D. & Turban, E., 2018. *Business Intelligence, Analytics and Data Science: A Managerial Perspective.* 4th szerk. hely nélk.:ismeretlen szerző

Singh, A. & Anand, P., 2013. State of Art in Ontology Development Tools. *International Journal of Advances in Computer Science and Technology,* 2(7).

Skoutas, D. & Simitsis, A., 2007. Ontology-based Conceptual Design of ETL Processes for both Structured and Semi-structured Data. *International Journal on Semantic Web and Information Systems,* 3(4), pp. 1-24.

Slimani, T., 2015. Ontology Development: A Comparing Study on Tools, Languages and Formalisms. In: hely nélk.:Indian Journal of Science and Technology, Vol 8(24), ISSN (online): 0974-5645.

Sommerville, I., 2011. Software Engineering Ninth Edition. In: hely nélk.:Pearson, pp. 26, 47-52, 73-78.

Sowa, J. F., 2000. *Knowledge Representation: Logical, Philosophical, and Computational Foundations.* hely nélk.:Brooks Cole Publishing Co.

Stojanovic, L., 2004. *Methods and Tools for Ontology Evolution,* hely nélk.: ismeretlen szerző

Sundararajan, S., 2020. *Business Analytics -Overview, Curriculum, Opportunities and Skills,* hely nélk.: Researchgate.

Taye, M. M., 2010. Understanding Semantic Web and Ontologies: Theory and Applications. *Journal of Computing,* 2(6), pp. 6-8.

Vaishnavi, V., Kuechler, B. & Petter, S., 2004. DESIGN SCIENCE RESEARCH IN INFORMATION SYSTEMS. In: hely nélk.:Eds, pp. 1-10.

Wieringa, R. J., 2014. In: *Design Science Methodology for Information Systems and Software Engineering.* hely nélk.:Springer.