Multidiszciplináris tudományok, 12. kötet, 2. sz. (2022), pp. 14–24 **https://doi.org/10.35925/j.multi.2022.2.2**

RENDSZERINTEGRÁCIÓS MEGOLDÁSOK ERP-KÖRNYEZETBEN

Hornyák Olivér
egyetemi docens, Miskolci Egyetem, Informatikai Intézet
3515 Miskolc-Egyetemváros, e-mail: oliver.hornyak@uni-miskolc.hu

Kiss Áron
tanszéki mérnök, Miskolci Egyetem, Informatikai Intézet
3515 Miskolc-Egyetemváros, e-mail: kiss.aron@uni-miskolc.hu

Nehéz Károly
egyetemi docens, Miskolci Egyetem, Informatikai Intézet
3515 Miskolc-Egyetemváros, e-mail: aitnehez@uni-miskolc.hu

Absztrakt

A cikkünkben arra keressük a választ, hogy létezik-e standardizált ERP integrációs modell törzsadatok szinkronizálására, másrészt a nyers számlaadatok standardizált, újra felhasználható struktúrára való átalakítására és egy idegen ERP-rendszer felé eljuttatására. A bemutatásra kerülő ERP-integrátor-komponens újszerű, mert nem tisztán adatbázis-replikáción alapul, nem is tisztán az üzleti logikát integrálja, hanem új megközelítésként az SQL-adatbázisoknál használatos eseménykezelő eljárásokat hív meg. A módszer fontos jellemzője, hogy az egységtesztek (unit tesztek) létrehozását természetes módon támogatja, így a működést leíró specifikációs feladatokat a tesztlépések sorozatának definiálásával is meg tudjuk határozni. A módszer kellőképpen általánosítható, azaz ilyen módon, tetszőleges (SQL-alapú) ERP-rendszer integrálható. Az integrátor alkalmazásával, az integrációs eljárás kidolgozásakor nem kell ismerni az idegen (third party) ERP belső adatbázis struktúráját, valamint az idegen ERP fejlesztőjének sem kell ismerni a komponens működését, hanem csak az integrátor által elvárt előre definiált nézeteket (DB view) és eseménykezelési mechanizmusokat kell alkalmazniuk.

Kulcsszavak: *ERP, informatikai rendszerintegráció*

Abstract

This article investigates whether there is a standardized ERP integration model for synchronizing master data and transforming raw account data into a standardized, reusable structure and transferring it to a third-party ERP system. The ERP integrator component presented in this paper is an innovative approach because it is not purely based on database replication, it does not purely integrate business logic, but it also utilizes the event handling procedures for SQL databases. An important feature of the method is that it naturally supports the creation of unit tests. Thus, the operational specification can also be expressed by defining a series of test steps. The method can be sufficiently generalized, so that using the method recommended in this paper every SQL based ERP system can be integrated. Using the proposed integrator, architectural design does not need to know the internal structure of the third-party ERP database when developing the integration process. Furthermore, the third-party ERP developers do not need to know how the component works internally, only some of the predefined database views and some event management mechanisms required by the integrator to be applied.

Keywords: *ERP, information systems integration*

1. Bevezetés

Az ERP-rendszerek használata során gyakran felmerülő igény a funkciók bővítése külső rendszerkomponensek, önálló rendszerek illesztésével, integrációjával. „Enterprise integráció" (Enterprise Application Integration, EAI) alatt definíció szerint több különböző platformon futó alkalmazás összekapcsolását értjük. A szoftvergyártók általában szolgáltatnak eszközöket a legtöbb platform és a rajtuk futó programozási technológiák összekapcsolásához valamint, a legelterjedtebb üzleti alkalmazások saját programból való eléréséhez szükséges interfészeket is nyújtanak. Ezek az eszközök azonban az integráció problémakörének csak egy kis részét fedik le, mivel a feladat esetenként messze túlmutat az üzleti és a műszaki kihívásokon. Egy vállalati szintű szoftverintegráció nagy változtatást kényszeríthet ki a vállalati működésben. Egy nagyvállalat informatikai rendszere az egyes részlegeken belül önállóan fejlődik és változik (például a számlázás részleg informatikai eszközei teljesen függetlenek lehetnek a felhasználói visszajelzésekkel foglalkozó részleg eszközeitől), ezért az integráció nagy megkötést jelent, hiszen az összekapcsolás után már nem lehet egyedileg módosítani az összetevőket. Minden alrendszer a nagy egész része, a módosítás befolyással van minden más alrendszerre is. Az integrációt végző csapat általában kis létszámú, és gyakran kevés módosítási lehetősége van az integrálandó szoftveren, mivel az más gyártótól származik vagy olyan „örökölt rendszer", melynek sokszor már a forráskódja sem elérhető, ezért senki nem ismeri a működését, és így szinte lehetetlen vagy irreálisan drága lenne változtatni rajta. Gyakran visszatérő probléma, hogy bizonyos programokat egyszerűbb lenne újraírni, hogy jobban illeszkedjenek a rendszerbe, de ez nem lehetséges technikai vagy cégpolitikai okokból: egyszerűen nem tudják vagy nem akarják megfizetni az új fejlesztés költségét egy olyan rendszernek (üzleti komponensnek), amely jelenleg kifogástalanul működik, csak éppen ősrégi, vagy csak ősrégi operációs rendszeren futtatható. A régi rendszer használatából eredő komoly üzleti kockázatokat pedig nem ismerik fel időben. A kutatás fő kérdéseit úgy fogalmazhatjuk meg, hogy a) megvalósítható-e egy standardizált ERP integrációs modell; b) ha megvalósítható, akkor milyen formában; c) milyen előnyökkel és hátrányokkal járnak az eddig ismert integrációs eljárások, és azokat hogyan kell továbbfejleszteni ahhoz, hogy a cégvezetés által megfogalmazott jelentős mértékű fejlesztés megvalósulhasson.

2. Az integráció

Az integráció feladata, hogy véglegesnek gyártott eszközöket egyesítsünk olyan új célok és funkcionalitások elérésére, amelyek csak az egyes eszközök együttes használatával érhetőek el. Ebben a cikkben bemutatjuk az összes, a gyakorlatban használt integrációs eljárást, egy áttekintést adva arról, hogy milyen alapeszközöket tudunk felhasználni a problémák leküzdéséhez. Az alkalmazások kezdetben függetlenek, egymás zavarása nélkül változhatnak, fejlődhetnek, azonban az összekapcsolásuk egy nagy rendszerré komoly függőséget okoz: többé-kevésbé megszünteti az egyes alkalmazások önállóságát. Ez rugalmatlanná, sőt továbbfejleszthetetlenné teheti a teljes rendszert, ezért már az első lépések megtétele előtt mérlegelnünk kell a kialakuló függőségek nagyságát, jövőbeli hatásukat. Ez a gyakorlatban azt jelenti, hogy a programokat összekötő interfészeket a lehető legáltalánosabbá kell tervezni, teret hagyva az implementáció jövőbeni változásainak. A már meglévő eszközöket a lehető legkisebb módosítás elvégzésével kell alkalmassá tenni a rendszerben történő működésre, elkerülendő, hogy a jól működő eszközt elrontsuk. Ez a megközelítési mód a programozás hatékonyságát is növeli, azonban gyakran nehéz előre meghatározni azt az irányt, amely a fejlesztői munka folyamatában a leghatékonyabbnak bizonyul. A legkisebb behatással járó integrációs pontokat előzetes vizsgálatokkal kell felmérni. Az integrációhoz nagyon sok eszköz áll rendelkezésre. Mielőtt választanánk, mérlegelni kell ezek árát, a szükséges betanítási időt, az esetleges szállítótól való függést. Egyes esetekben egyszerűbbnek, olcsóbbnak

vagy hatékonyabbnak tűnhet bizonyos eszközöket „házon belül” előállítani, mert így nem kell megvásárolni, és megtanulni egy másik szoftvergyártó termékét, azonban ez gyakran a költségesebb út, mert a tapasztalatok szerint könnyen a probléma alulbecslésének hibájába eshetünk.

Mindenképpen szükséges egy egységes adatformátum kidolgozása a sikeres kommunikációhoz. A probléma ezzel az, hogy a legtöbb összetevő belső kommunikációs és adatfeldolgozási formátumai nem megfelelőek a mindenütt történő használatra, így általában egy teljesen új formátumra van szükség. Ez az új adatformátum adapterek segítségével könnyen lefordítható kell legyen az egyes összetevők belső adatformátumaira. További problémát jelenthet a közös adatformátum fejlődésében a későbbi átalakítás nehézsége, ezért már a tervezésnél figyelembe kell venni, hogy lehetőleg semmilyen irányba ne tegyünk a szükségesnél több megkötést.

Nagy adatmennyiségek megosztása időigényes feladat, amelynek végrehajtása során az egyes alkalmazások esetleg elévült adatokat használhatnak. Ha viszont a kommunikációt kis adatcsomagok cseréjére alapozzuk, akkor a hálózati kommunikáció miatt jelentkezhetnek veszteségek. A kommunikáció során cserélt adatok mennyiségét, és az adatcseréhez szükséges időt mindenképpen előre fel kell mérni, mert ha csak a rendszer kifejlesztése után észleljük hibás működésként azt, hogy a rendszer nagy terhelés esetén hibás, elévült adatok alapján dolgozik, akkor már nagyon költséges lehet átalakítani a teljes kommunikációs rendszert.

A legtöbb integrációs rendszer magában foglal olyan megoldásokat, amelyekkel nemcsak adatokat, de funkciókat is megoszthatunk. Ez növeli a rendszer absztrakciós szintjét, de mivel a távoli eljáráshívások megvalósítása nagyon hasonlít a helyi függvényhívásokra, így ez gyakran félrevezető lehet.

Lokális hálózaton való kommunikáció során nem sokat kell törődnünk a külső hatásokkal vagy az adatok titkosságának védelmével, viszont a távoli rendszerek között történő adatcsere lehallgatható. Problémaként jelentkezhet, hogy a tűzfalak is megakadályozhatják az átvitelt. A távoli kommunikáció mindenképpen hosszabb ideig tart, és sokkal gyakrabban sérülhet az adatátvitel, így a használata kiemelt figyelmet igényel. A távoli rendszerrel való kommunikációban csak az aszinkron kommunikáció működhet, mivel nem garantálhatjuk, hogy minden gép be van kapcsolva az üzenetküldés pillanatában. Az ilyen bonyolult rendszerek tervezése, fejlesztése és hibajavítása nehéz, és speciális szaktudást, nagy tapasztalatot igényel.

3. Klasszikus integrációs minták

Az egyes alaptípusok történetileg egymástól függetlenül, folyamatosan fejlődtek. Alapvetően az alábbi négy mintát lehet megkülönböztetni:

Fájlátvitel: minden alkalmazás a megosztandó adatait fájlokba szervezve tárolja le, továbbá eléri a többiek megosztott fájljait, és felhasználja azok tartalmát.

Osztott adatbázis: egy közös, minden alkalmazás által elérhető közös adatbázisban vannak letárolva az adatok, amelyeket minden, a kommunikációban részt vevő összetevő (komponens) írhat és olvashat.

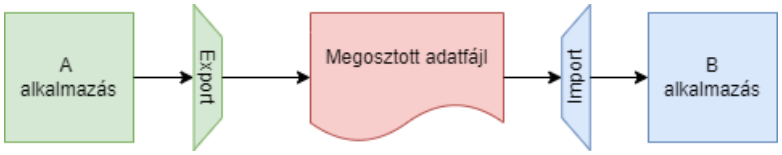
Távoli eljáráshívás: minden alkalmazás megosztja bizonyos függvényeit, hogy azokat más alkalmazások is elérhessék/alkalmazhassák. Ezt a megoldást nemcsak adatcserére, de a megosztott funkciók révén, bizonyos tevékenységek, például programrészletek megosztására is használhatjuk.

Üzenetváltás: minden alkalmazás egy közös üzenetküldő rendszerre csatlakozva üzeneteket küld és fogad. Az üzenetek tartalmazhatnak eljáráshívásokat, amelyek valamilyen feladat elvégzésére irányulnak. A fő előny az üzenetsor közbeékelése miatt kialakult aszinkron működésben rejlik.

Egy adott feladatnál, sokszor a fenti minták kombinálásával, vagy ezek egyidejű használatával érhetjük el a legjobb eredményt. Az alábbiakban röviden összegezzük a négy minta alapelveit és legfontosabb tulajdonságait.

3.1. Fájlátvitelen alapuló integráció

A fájlátvitel legfontosabb előnye az összes többi módszerrel szemben, a viszonylagos egyszerűsége. Örökölt rendszerek (legacy systems) esetén, sokszor ez az egyetlen megvalósítható megoldás. A fájlok hálózaton történő továbbítása minden platformon létezik, ez a megoldás nem igényel speciális szoftvert, legfeljebb elérési engedélyeket. A fájlátvitellel történő adatcsere folyamata rendkívül egyszerű, az adatokat a forrásalkalmazásból exportáljuk egy fájlba, amit aztán továbbküldhetünk vagy megosztott tárolóhelyen tárolhatunk, vagy bármilyen más úton eljuttathatjuk a célalkalmazásig, ahol már csak az adatok importálását kell megoldanunk.

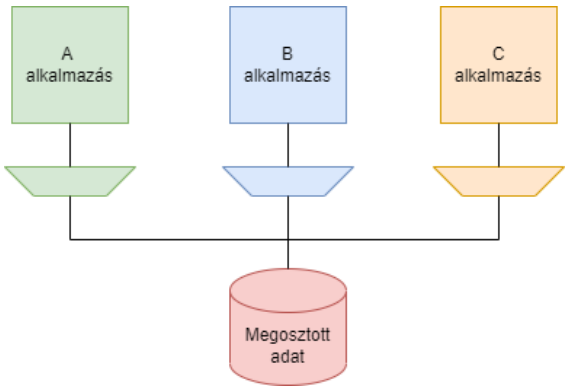


1. ábra. *Fájlalapú integráció*

A fájlátvitel alapú megoldás egyszerűsége miatt rendkívül kedvelt és talán a legszélesebb körben alkalmazható, azonban a hosszú távú felhasználására csak ritkán kerül sor, mert sajnos a legtöbb integrációs feladatnál a kommunikáció ideje kulcsfontosságú kérdés, a fájlátvitel-alapú integráció pedig egy lassú megoldás. Nem maga az input-output folyamat problémás, hanem a fájlrendszerbeli módosítások, egyes mappák periodikus megfigyelése. Továbbá a megoldás másik nagy kihívása, hogy ezeket az adatfájlokat mikor hozzuk létre és mikor dolgozzuk fel. Az esetleges párhuzamos működés inkonzisztens adatokat eredményezhet, vagy az operációs rendszer tiltja a hozzáférést, párhuzamos működés nélkül viszont a teljes alkalmazás várakozik a művelet elvégzéséig, vagyis nagy terhelésű rendszerekben a használata nem hatékony, nehezen felderíthető lassulásokat is eredményezhet. Tehát a fájlmegosztás egyszerű, de éppen az egyszerűsége miatt csak kis mértékben skálázható, rugalmatlan módszer.

3.2. Adatbázison alapuló integráció

A fájlmegosztásnál is kézenfekvőbb adatmegosztási eljárás az adatbázis-kezelő rendszerek párhuzamos használata. A működése a következő: létrehozunk egy közös adatbázist, amiben a rendszert alkotó alkalmazások mindegyike adatokat kereshet, menthet vagy módosíthat. Bár a megoldás triviálisnak tűnik, mégis sok nem várt problémát eredményezhet. A legnagyobb előnye, de ezzel egyben a legnagyobb hátránya is – a fájlmegosztáshoz hasonlóan –, hogy közvetlen adatkapcsolatot teremt az alkalmazások között. Ebben az esetben is kihívást jelent a változások figyelése, nyomon követése, sok komponens esetén az egyes elemek folyamatosan lekérdezik az adatváltozásokat, a módszer nem tudja értesíteni az egyes komponenseket adatmódosításkor. Azon túl, hogy a módszer az alkalmazások egymástól való függőségét növeli, a teljes rendszert centralizálja. Így akár egy komponens elégtelen teljesítménye vagy hibás működése azonnal hatással van a teljes rendszerre, annak minden elemére. (Parent and Spaccapietra, 1998)

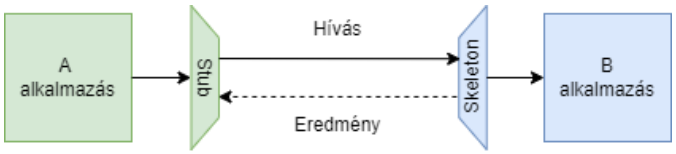


2. ábra. *Integráció osztott adatbázison keresztül*

3.3. Távoli eljáráshíváson alapuló integráció

A távoli eljáráshívás abból a szempontból jobb az egyszerű adatcserét megvalósító megoldásoknál, hogy itt lehetőség van az adatoktól és az adatszerkezetektől független feladatok elvégzésére is. Például egy felhasználó címének módosítása több regisztrációs automatizmust is beindíthat a hátérben. Az egységes, mindenki által elérhető adatok karbantartása rendkívül nehéz, mert a legapróbb adatszerkezet-változtatáshoz is több alkalmazást kell egyszerre módosítani. A távoli eljárás elrejti a belső adatszerkezetet a többi alkalmazás elől, ezért a karbantarthatósága ebből a szempontból lényegesen jobb.

Az adatok köré szervezett távoli eljárások leegyszerűsítik a szemantikai eltérések kezelését is. Több interfészt hozhatunk létre egy adat elérésére, így megoldható, hogy az egyes alkalmazások különböző módon lássák az adatot. Bár sok előnye van, a komponenseket nagyon szorosan kapcsolja egymáshoz, minden összetevőnek ismernie kell, hogy a többi összetevő hol van és hogyan működik. Ez az egyes alkalmazások egyedi fejlődését nagyban akadályozhatja.



3. ábra. *Távoli eljáráshívás*

3.4. Üzenetküldés-alapú integráció

Megvalósíthatóság szempontjából korábban egy üzenetváltásokon alapuló rendszer igényelte a legtöbb fejlesztői munkát, de mégis széles körben elterjedt, mert ez a megoldás biztosítja a legnagyobb rugalmasságot, és magában foglalhatja az összes többi integrációs technológiát. Így az integráció során nem sok kis, egyedi megoldást, hanem egy nagy, üzenetváltáson alapuló rendszert fejleszthetünk ki.

Az üzenetkezelő rendszerek (ÜKR) az alkalmazásokat lazán kötötté teszik azáltal, hogy a kommunikáció aszinkron. Ez a szinkron működés ellentétje, ami azt jelenti, hogy egy üzenet elküldése után egyáltalán nem várakozunk a válaszra. ÜKR alkalmazásával nő a rendszer stabilitása is, mivel a kommunikáló eszközöknek nem kell egyszerre rendelkezésre állnia (az üzenetküldő rendszer mindaddig új-

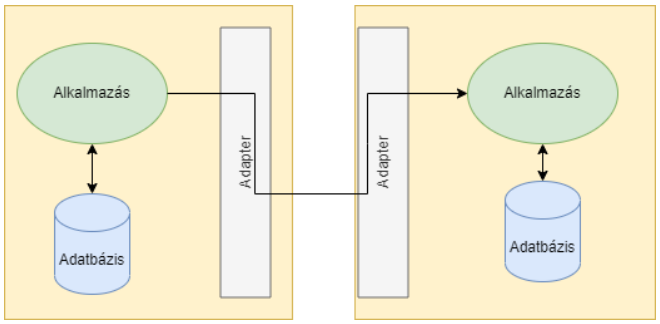
raküldi az üzenetet, amíg a címzett meg nem kapja azt). Mivel egy független üzenetküldő rendszer gondoskodik az adatok megérkezéséről, így az egyes alkalmazásoknak nem kell törődniük azzal, hogy hogyan jut el az üzenet a címzetthez, ami megkönnyíti az alkalmazások írását.

4. ábra. *Üzenetküldő rendszerek*

Az üzenetküldésen alapuló rendszerek komplex megközelítése a Message-Oriented Middleware (üzenetorientált köztes szoftver, MOM), amely mechanizmust biztosít az alkalmazások közötti tömeges, menedzselt, aszinkron üzenetek küldéséhez. Az üzenetek üzenetsorokba kerülnek, és amikor a célállomások elérhetők, akkor kerülnek elküldésre. A MOM az eseményeket leíró üzenetek sorba állítására szolgál, és elküldi ezeket az üzeneteket a távoli alkalmazásoknak, amelyek az események alapján műveleteket hajtanak végre. A MOM üzenetküldési lehetőségei jól általánosíthatók. (Albano et al., 2015)

3.5. Adapteralapú integráció

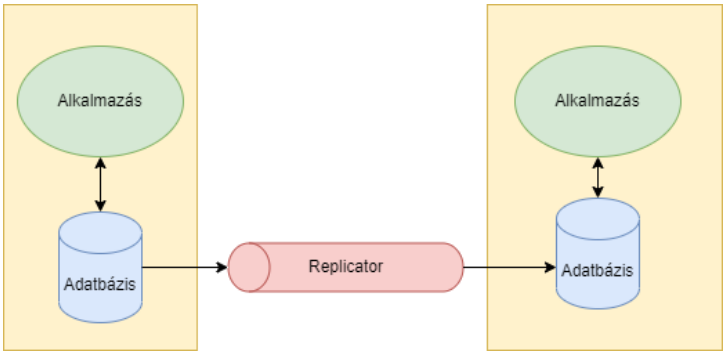
Az adapterek speciális interfészek a különféle alkalmazások között, és lehetővé teszik, hogy az alkalmazások közvetlenül kommunikáljanak egymással egyedi módon. Az alkalmazás nagyobb módosítására általában nincs szükség, feltételezve, hogy az alkalmazás vagy az adapter kidolgozója létrehozta/biztosította az adatmezők megfeleltetését (fordítását) és a felületet az adott alkalmazáshoz. Az adapterek elvégzik a szükséges fordítást egy közös kommunikációs protokollra, amelyet az adapter mindkét végén megért. (Burzlaff and Bartelt, 2021)



5. ábra. *Adapteralapú integráció* (Burzlaff and Bartelt, 2021)

3.6. Adatreplikációs motorok

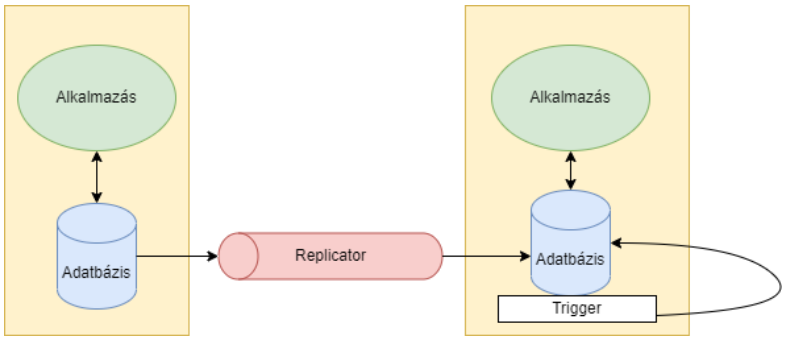
Az adatreplikációs motorok adatbázisszinten cserélnek adatokat a rendszerek között. A replikátor figyeli az adatváltozásokat és egyirányban (master-slave) szinkronizálja az adatbázisok tartalmát. Két különböző rendszeralkalmazás (ERP) esetén természetesen szűrni is lehet a szinkronizálandó táblákat, nem fog teljes egészében megvalósulni az adatmásolás. Ilyen megoldásokat a legismertebb adatbáziskezelők beépítve tartalmaznak, viszont általában nem várható el, hogy az integrálni kívánt rendszerek azonos adatbázist használjanak. Különböző rendszerek összekötése esetén mindkét alrendszerben jelentős forráskód-módosítást igényel egy ilyen típusú integráció. (Shute et al., 2013)



6. ábra. *Adatreplikáció* (Shute et al., 2013)

4. A javasolt integrációs eljárás bemutatása

Belátható, hogy vannak olyan integrációs feladatok, amelyeknél a megcélzott követelményeket egyik ismert módszer sem tudja teljesíteni. A kutatásunk alapján egy olyan módszert javaslunk, amely se nem tisztán replikációs módszer, se nem tisztán MOM vagy adapteralapú integráció. Olyan megoldást javaslunk alkalmazni, hogy a replikált adatok indítsanak el triggereket, amelyek akár az ERP-rendszer adatbázistábláiban már meglévő adatokat is felhasználják, létrehozzák az esetlegesen hiányzó adatszerkezeteket.

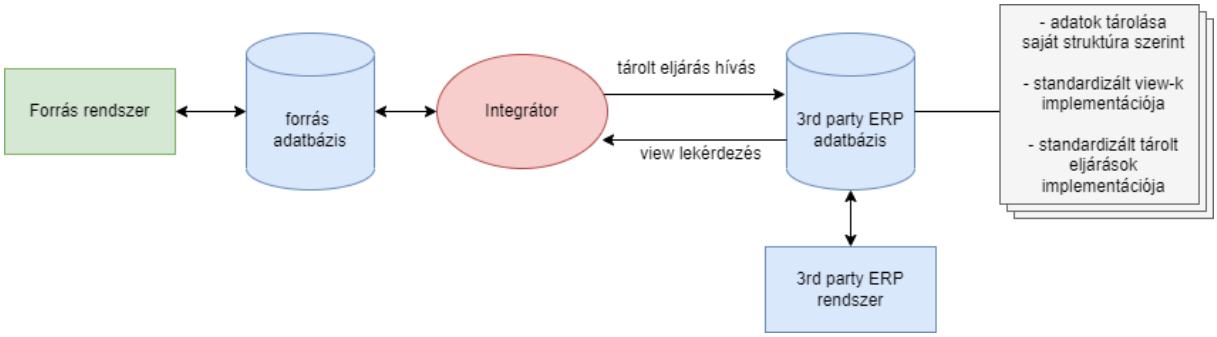


7. ábra. *A javasolt módszer elvi modellje*

A cikkünkben áttekintettük többek között a fájlalapú adatmegosztást, az adatbázis-alapú integráció különböző eseteit. Az áttekintett adatmegosztási módszerek közül a távoli eljáráshívás (RPC) alapú integráció elsősorban eljárások, és nem adatok megosztására szolgál, azonban a külső ERP-rendszerek integrációja során elsősorban az adatmegosztás (törzsadatok, érvényesített bizonylatok továbbítása) megoldása a cél. A fájlátvitelen alapuló adatmegosztást szintén nem tartjuk célszerűnek, hiszen felmerülnek a korábbi fejezetben már említett problémák: a fájlok elérésének módja, az adatok utólagos módosításának problémája, a külső rendszerek által alkalmazott eltérő kódolások kezelése. Az adatbázis-alapú integráció során az integrált alkalmazások közös adatbázis segítségével kommunikálnak egymással. Az új gondolat az, hogy adatbázis-alapú integráció megvalósulhat olyan módon, hogy a rendszerbe bekapcsolódó alkalmazások mindegyike a közös adatbázist különböző, igényeknek megfelelően testreszabott nézeteken keresztül éri el, ezek segítségével tudja írni és olvasni az adatbázist. A nézeteken keresztül történő kommunikáció lehetőséget biztosít arra is, hogy a nézetek egyes mezőit (vagy akár a

teljes nézetet) a külső rendszer meglévő üzleti logikája alapján hozzuk létre, így már standardizált nézet szinten is felhasználható a kapcsolódó rendszer know-how-ja. A nézetek működése az SQL specifikációjában rögzítve van, így a módszer használata szabványosnak tekinthető. Az adatbázis-alapú integrációs módszer előnyös lehet abból a szempontból is, hogy a külső ERP-rendszer fejlesztőjének nem kell külön integrációs alkalmazást implementálnia és üzemeltetnie, hanem az adatok cseréje a rendszer által használt adatbázisrendszerben történik, nézetek és tárolt eljárások megvalósításával, valamint egy megfelelő jogosultságokkal rendelkező felhasználó létrehozásával. Ezen megfontolások alapján a külső ERP-rendszerekkel történő adatcsere adatbázis-alapú kommunikációval történő megvalósítása mellett érvelünk.

A cikkben vázolt rendszer megvalósíthatóságát alátámasztó *ERP integrátor komponens*t egy Java-alapú alkalmazás formájában implementáltuk, mely a külső ERP rendszerből meghatározott szabványosított adatbázisnézetek (view-k) segítségével meghatározott időközönként (például perceként) adatokat emel ki, valamint érvényesített információkat juttat vissza az idegen rendszerbe, tárolt eljárások meghívásának segítségével, a szükséges információkat paraméterekként átadva. A standardizált view-k biztosítják azt, hogy a Java-alapú konnektor üzleti logikáját nem kell módosítani egy új ERP-integráció megvalósítása során, ugyanis a Java-konnektor mindig ugyanazt az ERP-adatstruktúrát kezeli. A tárolt eljárások biztosítják azt, hogy az *integrátor* az ERP-rendszer üzleti logikáját használja. Tehát például, ha az ERP rendszerben tárolt eljárás szintjén implementálásra került egy „pénzügyi bizonylat beillesztése" funkció, akkor egy „szállítói számla rögzítése" során az ERP gyártója által készített standard üzleti folyamatot az *integrátor* meg tudja hívni, és az adott ERP-rendszerben lévő ellenőrzött üzleti folyamat lesz az, ami a beillesztést elvégzi. A folyamatot a következő ábra szemlélteti.



8. ábra. *A javasolt módszer sematikus modellje*

A külső ERP-rendszer adatbázisa a megfelelő adatokat tetszőleges struktúrában tárolhatja. Ezekből az adatokból a megfelelő nézetek lekérdezésekor szabványos struktúrájú eredményhalmazt állít elő. Az integrátorkomponens az ERP-rendszer felé történő adattovábbításkor egy meghatározott elnevezésű és paraméterszignatúrájú tárolt eljárást hív meg, melynek paraméterként átadja a szükséges adatokat. Az integrátorkomponens integrációja az ERP-rendszer fejlesztője szempontjából a szabványos nézetek és tárolt eljárások implementálásával valósul meg. A rendszer üzemeltetőjének létre kell hoznia az integrátorkomponens számára egy megfelelő jogosultságokkal rendelkező adatbázis felhasználót, aki a szükséges nézeteket és tárolt eljárásokat eléri, illetve végre tudja hajtani.

A külső rendszerben minden átadni kívánt törzsadattípusra létre kell hozni egy szabványos struktúrájú tranzakciókat jelképező nézetet, mely azokat a rekordokat kell, hogy visszaadja, amelyeken az *integrátor* legutolsó adatszinkronizációja óta valamilyen írási művelet [létrehozás (C), módosítás (U), törlés (D)] lett alkalmazva. A szabványos nézetek általános felépítése a következő:

Tranzakcióazonosító	Törzsazonosító	Művelettípus (C/U/D)	… Törzsadatmezők

A törzsadatmezők a módosult rekord adatmezőiben található új értékek.

Bemutatás példán keresztül

Vegyük például egy ERP-rendszer könyvelői moduljait. A rendszer például főkönyvi időszakokat, főkönyvi számlákat, fizetési módokat, minősítés osztályokat, minősítéseket, áfatípusokat, áfatípusok kontírozási bekínálásait, pénzügyi mozgásnemeket, pénzügyi mozgásnemek kontírozási bekínálásait, áfakulcsokat, iktatókönyveket, iktatókönyvek kontírozási bekínálásait, partnertörzseket, partnerek cím- és bankadatait, valamint kontírozási bekínálásait fogadhatja a külső rendszerből.

Az *1. táblázat*ban szereplő példa egy főkönyvi számla adatait tartalmazó nézet struktúráját írja le. Az *integrátor*komponens ebben a formátumban olvassa ki az adatokat a külső ERP rendszerből. Az 1–3. sorok a tranzakciót azonosító adatokat tartalmazzák, míg a 4–10. sorokban a külső rendszerben tárolt törzsadatok szerepelnek. A tranzakciórekordot a külső ERP-rendszer hozza létre (célszerűen triggerek segítségével), amikor a főkönyvi számlákat tároló táblában írási művelet történik.

1. táblázat. *Főkönyvi számlaadatok nézetének struktúrája.*

Sorszám	Oszlopnév	Típus	Oszlopleírás
1	ID	azonosító	Tranzakcióazonosító
2	DATA_ID	azonosító	Törzsazonosító
3	CUD	szöveg	Művelettípus
4	GENERAL_LEDGER_PERIOD_ID	azonosító	Főkönyvi időszakra hivatkozás
5	ACCOUNT_NUMBER	szöveg	Számlaszám
6	ACCOUNT_NAME	szöveg	Számla elnevezése
7	VALID_FROM	dátum	Érvényesség kezdete
8	VALID_TIL	dátum	Érvényesség vége
9	DESCRIPTION	szöveg	Leírás
10	VAT_ACCOUNT_NUMBER	szöveg	Adószám

Az integrátorkomponens érvényesített adatok átadására is képes a külső ERP-rendszer számára, szabványos elnevezésű és szignatúrájú tárolt eljárások meghívásával. Az integrátor a tárolt eljárások visszatérési értékeit nem veszi figyelembe. A következő, példaként bemutatott tárolt eljárás elnevezése INTEGRATOR_INVOICE_ITEM, és a fogadott, majd érvényesített pénzügyi bizonylati tételek beillesztésére szolgál. Paraméter szignatúrája a *2. táblázat*ban látható.

2. táblázat. *Érvényesített tételadatok struktúrája.*

Sorszám	Paraméter neve	Típus	Leírás
1	LINE_ID	szöveg	Tétel azonosító
2	ID	szöveg	Számlafej azonosító
3	ROWNUMBER	szám	Sorszám
4	VAT_RATE_ID	szám	Áfakulcs
5	NET_AMOUNT	szám	Alap (devizában)
6	VAT_AMOUNT	szám	Adó (devizában)
7	NET_AMOUNT_HUF	szám	Alap (forintban)
8	VAT_AMOUNT_HUF	szám	Adó (forintban)

9	ACCOUNTING_NET_AMOUNT_HUF	szám	Könyvelési alap
10	ACCOUNTING_NET_AMOUNT_HUF	szám	Könyvelési adó
11	UNIT_PRICE	szám	Egységár (devizában)
12	UNIT_PRICE_HUF	szám	Egységár (forintban)
13	LINE_TEXT	szöveg	Tétel szövege
14	FINANCIAL_MOVEMENT_TYPE_ID	szám	Mozgásnem azonosító
15	LINE_DELIVERY_DATE	dátum	Gyűjtőszámla esetén tételsor teljesítési dátuma
16	LINE_EXCHANGE_RATE	szám	Devizás gyűjtőszámla esetén tételsor árfolyama

Az adatok fogadását, esetleges validációját, majd tárolását a külső ERP-rendszer fejlesztői implementálják, így az üzleti folyamat teljes mértékben ellenőrzött. A következőkben egy referenciaimplementáció pszeudokódját mutatjuk be:

```
1.   if LINE_ID is NULL then:
2.       exception('A külső sor azonosítót meg kell adni!')
3.   else if ID is NULL then:
4.       exception('A külső fej azonosítót meg kell adni!')
5.   else if VAT_RATE_ID is NULL then:
6.       exception('Az ÁFA kulcsot meg kell adni!')
7.   else if ACCOUNTING_NET_AMOUNT_HUF is NULL then:
8.       exception('Az adóalap összeget meg kell adni!')
9.   else if ACCOUNTING_VAT_AMOUNT_HUF is NULL then:
10.      exception('Az adó összeget meg kell adni!')
11.  end if
12.
13.  select biz_id INTO biz_id, devizas INTO xdevizas FROM bizonylatok WHERE kulso_id=ID
14.  if biz_id is NULL then:
15.      exception('Nem lehet meghatározni a bizonylat fejet az azonosító alapján!')
16.  else if xdevizas='I' and (NET_AMOUNT is null or VAT is null) then:
17.      exception('Nem adta meg a devizás értékek alap és adó összegét!')
18.  end if
19.
20.  select kulcs INTO xkulcs from afakulcsok where id=VAT_RATE_ID
21.  if xkulcs is NULL then:
22.      exception('Nem létező ÁFA kulcs!')
23.  end if
24.
25.  insert into bizonylat_tetelek (id, lsor, pbiz_id, alap, afa_kulcs, ado, szoveg,
     btet_mozg_id, tip, egyseg_ar, deviza, deviza_egyseg_ar, afa_alap, afa_ado)
26.  values (biz_id, sor, xpbiz_id, accounting_net_amount_huf, xafa_kulcs,
     accounting_vat_amount_huf, substr(line_text,1,40), financial_movement_type_id, 'A',
          unit_price_huf, net_amount, vat_amount, net_amount_huf, vat_amount_huf);
27.  insert into naplo (tabla, sajat_id, kulso_id, kulso_program) values ('
     BIZONYLAT_TETELEK', biz_id, ID, 'Integrátor');
```

A kód 1–11. sorai ellenőrzik azt, hogy az integrátortól minden szükséges adat megérkezett-e, amennyiben nem, kivétel keletkezik. A 13–18. sorok azt ellenőrzik, hogy a bizonylat feje és a sor pénzügyi adatai szerepelnek-e a rendszerben. A 20–23. sorok pedig azt vizsgálják, hogy a megkapott áfakulcs

létezik-e a helyi adatbázisban. Amennyiben az ellenőrzések nem találtak hibát, a kapott adatok beillesztésre kerülnek az ERP-rendszer megfelelő táblájába, illetve naplóbejegyzés készül a műveletről egy másik táblába.

Az integrátorkomponens alapjaként célszerű egy olyan könyvtár vagy API használata, amely többféle adatbázishoz történő kapcsolódást és kommunikációt is támogat. A Java programozási nyelv esetében a JDBC API segítségével valósítható meg a feladat, és az alábbi legfontosabb adatbázisrendszerekkel történő kommunikációt natív módon támogatja: MySQL, Oracle, PostgreSQL, MSSQL, DB2, Firebird.

5. Összefoglalás

A cikkben egy konkrét rendszerintegrációs feladat kapcsán fogalmaztunk meg egy új, általánosítható eljárást. A javasolt integrációs módszer, alapelvét tekintve adatbázisintegráció, ahol új táblák, nézetek és tárolt eljárások testesítik meg magát az integrációt és az interfészeit. A tervezett megoldás érdekessége, hogy a bemutatott interfésznézetek kialakíthatóak más, SQL-adatbázismotort használó ERP-rendszerek táblái és adatai alapján is, amely a gyakorlatban kiváló általánosítási lehetőséget biztosíthat a bemutatott módszer termékesítésében.

Irodalom

[1] Kasun, I., Siriwardena, P. (2018). *Microservices for the Enterprise*. Apress, Berkeley, ISBN-10: 1484238575.

[2] Roshen, W. (2009). *SOA-based enterprise integration: A step-by-step guide to services-based application.* McGraw-Hill Education, ISBN-10: 0071605525.

[3] Enterprise Integration Patterns (n.d.). Retrieved February 21, 2022, https://www.enterpriseintegrationpatterns.com/.

[4] Juhász, S. (2011). *Vállalati információs rendszerek műszaki alapjai*. Szak Kiadó, Budapest.

[5] Shute, J. et al. (2013). F1: *A Distributed SQL Database That Scales VLDB.* **https://doi.org/10.14778/2536222.2536232**

[6] Burzlaff, F., Bartelt, C. (2021). Knowledge-Driven Architecture Composition: Assisting the System Integrator to Reuse Integration Knowledge. In: Brambilla, M., Chbeir, R., Frasincar, F., Manolescu, I. (eds.). *Web Engineering*. ICWE 2021. Lecture Notes in Computer Science, vol 12706. Springer, Cham, **https://doi.org/10.1007/978-3-030-74296-6_23**

[7] Albano, M., Ferreira, L., Pinho, L., Alkhawaja, A. (2015). Message-oriented middleware for smart grids. *Computer Standards & Interfaces*, 38, pp. 133–143. **https://doi.org/10.1016/j.csi.2014.08.002**

[8] Parent, C., Spaccapietra, S. (1998). Issues and approaches of database integration. *Communications of the ACM*, 41 (5), pp. 166–178, **https://doi.org/10.1145/276404.276408**