

**Debreceni Egyetem
Informatikai Kar**

**WEB ÉS GRID ALAPÚ
SZOLGÁLTATÁSORIENTÁLT
ARCHITEKTÚRÁK**

Témavezető:
Dr. Juhász István
egyetemi adjunktus

Készítette:
Hennel Zoltán
programtervező matematikus

*Debrecen
2008*

Tartalomjegyzék

1. BEVEZETÉS.....	3
2. A SOA ALAPJAI	4
2.1 A SOA kialakulásának háttere.....	4
2.2 SOA bevezető	5
2.3 A SOA elemei.....	6
2.3.1 Alkalmazás kliens.....	7
2.3.2 Szolgáltatás.....	8
2.3.3 Szolgáltatás tárház	9
2.3.4 Szolgáltatás busz	11
2.4 Összefoglalás	12
3. WEBSZOLGÁLTATÁS TECHNOLÓGIA.....	13
3.1 Webszolgáltatás architektúra	13
3.1.1 SOAP	14
3.1.2 WSDL	16
3.1.3 UDDI	18
3.1.4 A WSDL és az UDDI kapcsolata	28
3.2 ESB technológia	29
3.3 BPEL4WS	32
3.3.1 Alap aktivitások.....	33
3.3.2 Vezérlési szerkezet aktivitások.....	34
3.3.3 BPEL folyamatok publikálása UDDI regiszterekben.....	37
3.4 Összefoglalás.....	38

4. WEBSZOLGÁLTATÁSOK ÉS A GRID KAPCSOLATA.....	40
4.1 Grid rendszerek.....	40
4.2 Állapotőrző erőforrások szolgáltatásként megjelenítve	43
4.2.1 A GSH és GSR koncepció.....	44
4.2.2 Szolgáltatás gyárak	45
4.2.3 Szolgáltatás adat elemek (SDE)	47
4.3 Dinamikus hálózatok kezelése	48
4.3.1 Szolgáltatás élettartam menedzsment.....	49
4.3.2 SDE értesítés	50
4.3.3 HandleMap szolgáltatás.....	51
4.4 Az OGSZ és a WSRF	52
5. ÖSSZEFOGLALÁS	54
IRODALOMJEGYZÉK	56

1. BEVEZETÉS

A szolgáltatásorientált architektúrák, szolgáltatásorientált rendszerek, ezek technológiája az elmúlt évek egyik leggyakrabban emlegetett témakörévé vált az informatika és az üzleti élet területén. A szolgáltatásorientáltság nem új keletű dolog, hiszen már a 90-es évek vége felé megjelent ez a fogalom. Az elektronikus piac térnyerése és forgalmának növekedése új igényeket támasztott fel a meglévő rendszerekkel szemben. A folyamatosan változó üzleti igényeknek megfelelően a rendszer legyen könnyen testre szabható, legyen könnyen összekapcsolható a meglévő rendszerekkel, problémamentesen együttműködve azokkal. Az ezt megvalósító programozási modell támogassa az elosztott rendszereket, ügyelve a biztonságra és a nagymértékű terhelési igényekre.

A fenti igények megoldására irányuló kutató munka eredményeként megjelent a szolgáltatásorientált rendszer, a szolgáltatásorientáltság fogalma. A *szolgáltatásorientált architektúra* fogalom megtévesztő, hiszen a SOA nem egy konkrét megvalósítása a szolgáltatásorientált rendszereknek, hanem egy elméleti ajánlás. A szolgáltatásorientált rendszerek absztrakciós szintjét megnövelve próbálja definiálni a szolgáltatásorientáltság fogalmát. Több technológiai megvalósítás létezik, ezek közül is a legelterjedtebb a webszolgáltatás alapú rendszerek. De emellett a grid technológia is a szolgáltatásokra épülő irányt jelölte ki fejlődési útjaként.

A diplomamunkám célja, hogy bemutassam a SOA-ban rejlő lehetőségeket az alapfogalmak megismerésétől kezdve, annak webszolgáltatás alapú implementációjának ismertetésén, és a webszolgáltatás technológia grid rendszerekkel való kapcsolatán keresztül.

2. A SOA ALAPJAI

2.1 A SOA kialakulásának háttere

A web technológiáknak köszönhetően a 90-es évek végére az adatok, információk néhány kattintással szinte mindenki számára elérhető váltak. Egyre több nagy-, illetve kisvállalkozás jelent meg az Interneten. Ha a vállalatok versenyképesek akartak maradni az egyre növekvő és változó üzleti világban, akkor újabbnál újabb szolgáltatásokat és funkciókat kellett biztosítaniuk az ügyfelek és partnereik számára. Az új szolgáltatások megjelenése új szoftver, illetve hardver elemek integrálását követelte meg a már meglévő, működő rendszereikbe. Ennek hatására a vállalati infrastruktúra egyre komplexebbé, kezelhetetlen méretűvé vált, egyre nehezebb lett a fejlesztés, az új szolgáltatások integrálása, az üzemeltetés, karbantartás költségesebbé vált, lehetetlenné téve a változó piaci igényekhez való gyors alkalmazkodást. Ezen körülmények piacvesztéshez, akár bukáshoz is vezethettek.

A kiélezett piaci verseny új technológiák, módszerek kifejlődését vonta maga után. A fejlesztőknek be kellett látniuk, hogy az egyre összetettebb és bonyolultabb igényeket kielégítő rendszerek elkészítéséhez az absztrakt szint növelésére van szükség. Ez irányba történő elmozdulás első lépcsőfoka a strukturált programtervezés volt, amelyet az objektumorientált paradigma elterjedése követett. Az OO paradigma lehetővé tette az interfészek definiálásával az implementáció elrejtését, növelve ezzel az absztrakciós szintet. Megjelent az újrafelhasználhatóság fogalma kód szinten. A kód újrafelhasználhatósága mellett azonban felmerült az igény a funkcionalitás újrafelhasználhatóságára is. Ennek eredményeképp jött létre a komponens alapú szoftverfejlesztés. Az elképzelés az volt, hogy a rendszer függetlenül tervezett komponensek halmazaként álljon elő, amelyek legfontosabb jellemzője, hogy elrejtik az implementációjukat. A komponens készítője ezzel szemben megváltoztathatja a komponens implementációját, feltéve, hogy az interfész változatlan marad.

Ezen technológiák azonban nem voltak alkalmasak az elosztott rendszerek technológia független megvalósítására, mivel azok nem minden aspektusra kiterjedő szabványokon

alapultak, hanem sok, egymástól különböző technológiáktól függtek. Ezen körülmények hatására jelent meg a szolgáltatásorientált technológia fogalma.

2.2 SOA bevezető

Ahhoz, hogy megértsük a szolgáltatásorientált technológiák lényegét, növelnünk kell a rendszerről alkotott kép absztrakciós szintjén. A korábban megszokott erőforrás és szoftver komponens építőelemek helyett egy magasabb szinten mozgunk és szolgáltatásokról beszélünk.

Mi is ez a szolgáltatás? Szolgáltatásnak nevezünk bármely entitást, mely jól definiálható – számítógép program számára érthető – interfészen keresztül valamilyen funkcionalitást (szolgáltatást) nyújt az őt használó klienseknek. Egy szolgáltatásorientált rendszer tehát *szolgáltatásokból* (services) és *kliensekből* (clients) áll. Mi biztosítja azonban azt, hogy a kliens és a szolgáltatás egymásra találjon? Szükség van egy harmadik félre is a rendszer megbízható működéséhez. Ez a *közvetítő* (registry), amelynek szerepe, a kliens és a szolgáltatás egymásra találásának biztosítása. Miért is van erre szükség? A hagyományos elosztott rendszerek hátránya, a szerver ismertségének szükségessége volt. A kliens egy adott, ismert hálózati szerveren keresztül érte el a szolgáltatásokat, ami gyakran változtathatatlan volt, ráadásul szerver hiba esetén, a kliens tehetetlenné és működésképtelenné vált. A közvetítő beiktatása lazán csatolt kapcsolatot biztosít a kliens és a szolgáltatás között. A kliens a közvetítőtől kér szolgáltatást, hiba esetén kérhet egy másik, helyettesítő szolgáltatást. A szolgáltatásorientált rendszer, e szolgáltatásközvetítő kliens hármasa alkotja a szolgáltatásorientált architektúrát (Service Oriented Architecture, SOA).

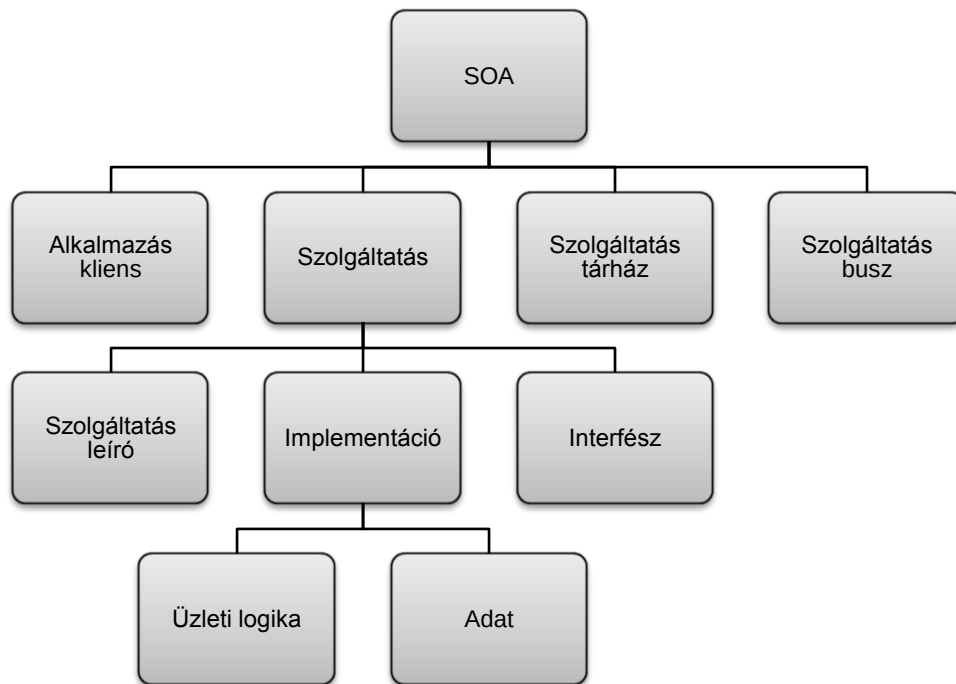
Nagyon fontos látni, hogy az alapelv a funkcionalitás interfészen keresztül történő megjelenítése. A szolgáltatásorientált rendszer nem szól az implementációról, kliens elől elrejtí azt, mivel egy kliens számára nem az a fontos, hogy hogyan és milyen programozási nyelv segítségével valósítja meg a szolgáltatás a felkínált funkciót, hanem hogy mit nyújt. Ez az implementáció független, interfész központú megközelítés teszi lehetővé, hogy különböző nyelven írt rendszereket kapcsolhassunk össze.

A helyettesítő szerver használata – mely a folyamatos rendelkezésre állás elérése miatt nagyon fontos elem – csak akkor oldható meg egyszerűen, ha a szolgáltatások állapotmentesek. Egy szolgáltatást *állapotmentesnek* (stateless) nevezünk akkor, ha a kliens tetszőleges sorrendben hajthat végre funkciókat a szolgáltatáson, az mindig ugyanúgy jelenik meg. Egy példa erre a statikus tartalommal rendelkező HTML oldalakat exportáló web szerver. A kliensek bármilyen sorrendben kérhetnek egy oldalt, bár hányszor, mindig ugyanazt kapják (eltekintve most a frissítésektől). Sok esetben azonban ez a működési mód nem biztosítható, pl. adatbázis módosítás vagy távoli szerveren feladat végrehajtás esetén. Ezek *állapotőrző* (stateful) implementációk (tipikusan a grid szolgáltatások), melyek bonyolultabb replikálási megoldásokat igényelnek.

2.3 A SOA elemei

A SOA egy specifikusabb megfogalmazása a következőképpen definiálja a szolgáltatásorientált architektúrákat: A SOA egy olyan szoftver architektúra, amely az *alkalmazás kliens* (application frontend), *szolgáltatás* (service), *szolgáltatás tárház* (service repository) és a *szolgáltatás busz* (service bus) fogalmakon alapszik. A szolgáltatás magába foglal egy *szolgáltatás leíró*t (contract), egy vagy több *interfészt* (interface), és egy *implementációt* (implementation).

A SOA fogalma teljes egészében az üzleti infrastruktúrára koncentrál. Amikor szolgáltatásokról beszélünk, igazából üzleti szolgáltatásokra gondolunk. Például repülőjegy foglalás, foglalás törlése, stb. Megkülönböztetjük az üzleti szolgáltatásokat a technikai szolgáltatásoktól, mint például egy tranzakció megnyitása, adat frissítés, stb. Annak ellenére, hogy ezek a technikai megoldások nagyon hasznosak az implementáció során, a SOA nem tekinti őket stratégiaileg fontos fogalmaknak. Még általánosabban fogalmazva, a technológia nem szabhat határt a magasabb szinten definiált alkalmazás struktúrának, az implementációt és technikai szolgáltatásokat el kell rejtetni az üzleti szolgáltatások elől.



1. ábra A SOA elemei

A következő részben megismerkedünk a fentebb említett SOA elemekkel egy kicsit részletesebben.

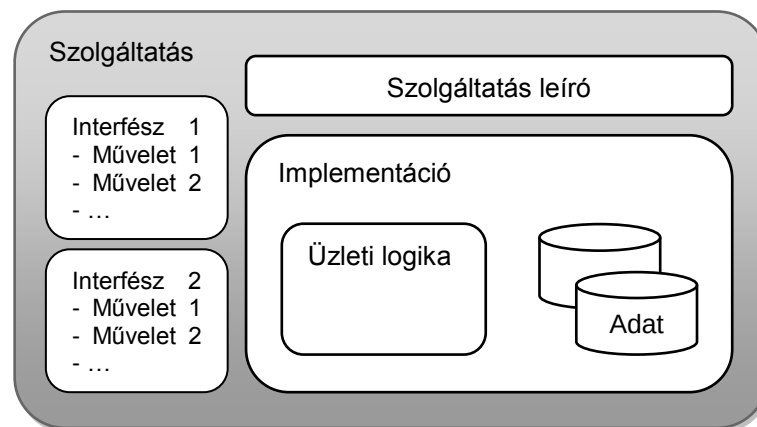
2.3.1 Alkalmazás kliens

Az alkalmazás kliensek a SOA aktív részei. A kliensek kezdeményeznek kéréseket, igényelnek szolgáltatásokat, vezérlik az összes aktivitást a vállalati rendszeren belül. Számos típusa létezik egy alkalmazás kliensnek. A legnyilvánvalóbbak ezek közül a grafikus interfésszel rendelkező alkalmazások, mint például webes alkalmazások, Windows alapú kliensek. Ezen típusú kliensek közvetlen kapcsolatban állnak a felhasználóval. Ám vannak olyan alkalmazás kliensek is, amelyek csak közvetett kapcsolatot tartanak fent a felhasználókkal. Ilyenek a kötegelt programok, hosszú távú folyamatok, amelyek periodikusan hívják meg a szolgáltatásokat, vagy akár speciális események is lehetnek alkalmazás kliensek.

Mindig az alkalmazás kliensek kezdeményezik az üzleti igényeiket, és mindig azok kapják vissza az eredményt. A szokásos többrétegű alkalmazásokkal szemben a szolgáltatások nem az alsóbb rétegekhez kapcsolódnak szorosan, hanem az alkalmazás klienshez.

2.3.2 Szolgáltatás

A szolgáltatás egy olyan szoftver komponens, amely valamilyen funkcionalitást, tipikusan egy magas szintű üzleti fogalmat valósít meg. A szolgáltatás számos összetevőből áll:



2. ábra Szolgáltatás felépítése

- **Szolgáltatás leíró:** A szolgáltatás leírók olyan, a szolgáltatáshoz szorosan nem kapcsolódó információkat tartalmaznak, amelyek a szolgáltatás céljáról, funkcionalitásáról, megszorításairól és használatáról tartalmaznak információkat. Ezen leírók specifikációja különböző lehet a szolgáltatás típusától függően. Ilyen információ lehet például a programozási nyelv, amelyen a szolgáltatás íródott, szolgáltatás által használt hálózati protokollok, futtató környezet, a szolgáltatás helye.
- **Interfész:** A kliensek az interfészen keresztül érik el a szolgáltatás funkcionalitását, amely elrejtí az implementációt a kliens elől. Az interfész leírását a szolgáltatás leíróban definiáljuk.

- **Implementáció:** A szolgáltatás implementáció valósítja meg konkrétan az üzleti logikát és a hozzákapcsolódó adatokat. Olyan technikai megvalósítás, amely teljesíti a szolgáltatás leíróban definiált információkat.
- **Üzleti logika:** A szolgáltatás által magába foglalt üzleti logika az implementáció része. Az üzleti logikát az interfészen keresztül éri el a kliensek.
- **Adat:** Az üzleti logikához kapcsolódó adatok, tipikusan adatbázisokban tárolva.

2.3.3 Szolgáltatás tárház

A szolgáltatás tárházak megkönnyítik a szolgáltatások felfedezését, és megadnak minden információt a szolgáltatás használatához, főképp akkor, ha a szolgáltatást az adott projekten kívül definiálták. Habár a szolgáltatás leírók számos információval rendelkeznek az adott szolgáltatásról, a tárházak további információkat képesek nyújtani. Ilyen információ lehet például a szolgáltatás fizikai helye, a szolgáltatás biztosítója, technikai megszorítások, esetleges biztonsági problémák, elérhető szolgáltatás szintek.

Annak ellenére, hogy készíthetünk SOA alkalmazást szolgáltatás tárházak nélkül is, hosszabb távon elengedhetetlen a használatuk. A tárházakat abban az esetben nélkülözhetjük, ha a szolgáltatás hatásköre csak egy projektre terjed ki, az alkalmazás kevés szolgáltatásból áll, minden projektet egy csapathoz tartozó tagok alkalmaznak. A valóságban egy vállalati rendszerben számos konkurens projekt létezik, különféle szolgáltatásokkal, amelyeket más és más csapatok használnak. Egy szolgáltatás tárháznak a következő elemeket kell tartalmaznia:

- **Szolgáltatás interfész:** A szolgáltatás milyen funkciókat nyújt, hogyan lehet azokat elérni, milyen argumentumok szükségesek a szolgáltatás eléréséhez.
- **Szolgáltatás tulajdonosa:** A szolgáltatás tulajdonos az, aki mind üzleti, fejlesztői és működési szinten is képes hozzáférni a szolgáltatáshoz.
- **Hozzáférési jogosultságok:** Olyan hozzáférés vezérlő listák, biztonsági mechanizmusok, amelyek meghatározzák, hogy ki, milyen jogosultsággal használhatja a szolgáltatást.

- **Teljesítmény és skálázhatóság:** A szolgáltatás teljesítményéről és skálázhatóságáról ad információt, mint az átlagos válaszidő, átmenő teljesítmény-korlátozások. Ezen információk megjelenhetnek a szolgáltatási szint szerződés (SLA) részeként.
- **Tranzakció tulajdonságok:** A szolgáltatás tranzakciós tulajdonságairól, a tranzakció egyedi műveleteiről tartalmaz információt.

Annak ellenére, hogy a szolgáltatás tárház rengeteg információt képes nyújtani a szolgáltatásról, a legfontosabb tulajdonsága mégis a szolgáltatás felderítés. A szolgáltatás felderítésnek különböző szintjeit definiáljuk attól függően, hogy a felderítést a szolgáltatásnak milyen tulajdonsága alapján végezzük:

- **Szolgáltatás neve alapján:** A legáltalánosabban használt felfedező eljárás. A szolgáltatás már fejlesztési időben ismert, a kliens tudja, hogy milyen szolgáltatást milyen néven ér el. A kliens számára lehetővé teszi, hogy egy adott nevű szolgáltatás különböző példányait használja (például, ha a szolgáltatásnak új verziója létezik).
- **Szolgáltatás tulajdonságai alapján:** Hasonlít az előző megoldásra, kivétel az, hogy a kliens nem ismeri a szolgáltatás nevét, de tudja, hogy milyen tulajdonságokkal kell rendelkeznie. A kliens az adott tulajdonságok alapján kér le szolgáltatást.
- **Szolgáltatás felderítés vizsgálattal:** Abban az esetben használatos, ha a kliens felfedezett egy adott tulajdonságokkal rendelkező szolgáltatást, de nem ismeri annak interfészét. Ez esetben egy vizsgáló mechanizmust kell beépíteni a kliensbe, amely képes dinamikusan felderíteni, hogy vajon az adott szolgáltatás interfésze, formája kielégíti-e a kliens elvárásait.

Érdemes törekedni arra, hogy a felfedezés a lehető legegyszerűbben történjen, mivel a szolgáltatás komplexitása és kockázata exponenciálisan növekszik a felderítő folyamat dinamikusságának szintjével.

2.3.4 Szolgáltatás busz

A szolgáltatás busz kapcsolja össze a SOA alkalmazásban résztvevő összes szolgáltatást az alkalmazás kliensekkel. Ha egy kliensnek szüksége van egy szolgáltatásra, akkor azt a szolgáltatás buszon keresztül érheti el.

A szolgáltatás busz a következő tulajdonságokkal rendelkezik:

- **Konnektivitás:** Az elsődleges célja a szolgáltatás busznak, hogy összekapcsolja a SOA alkalmazás résztvevőit. Biztosítja azt, hogy a kliensek könnyen hozzáférjenek az egyes szolgáltatásokhoz.
- **Technológiák közötti átjárás:** A szolgáltatás busznak több különböző technológiát kell támogatnia. A vállalati rendszerek többnyire különféle technológiák egyvelegeként állnak elő. Ebből adódóan a szolgáltatás busznak meg kell teremtenie az átjárhatóságot a különböző technológiák, programozási nyelvek, operációs rendszerek, futtató környezetek között.
- **Kommunikáció egységesítése:** Hasonló célt szolgált, mint az előző tulajdonság, csak itt a kommunikáció megkönnyítése a cél. Legalább a szinkron, aszinkron kommunikációt kell megkönnyítenie.
- **Technikai szolgáltatások:** Habár a szolgáltatás busz elsődleges célja a kommunikáció megkönnyítése, azon felül számos technikai szolgáltatást is nyújt. Ilyen szolgáltatások a belépéskezelés, auditálás, biztonság, üzenet transzformáció, tranzakció kezelés.

2.4 Összefoglalás

Láthatjuk, hogy a szolgáltatásorientált rendszerek a következő négy alapfogalomra épülnek: *kliens*, *szolgáltatás*, *szolgáltatás tárház*, *szolgáltatás busz*. Ezen elemek együttese hivatott kielégíteni a szolgáltatásorientált rendszerek igényeit:

- Változó üzleti igények gyors kielégítése.
- Gyors integráció a meglévő rendszerekbe.
- Elosztott rendszerek támogatása.
- Biztonsági és terhelési kérdések megoldása.
- Technológia és platform függetlenség.

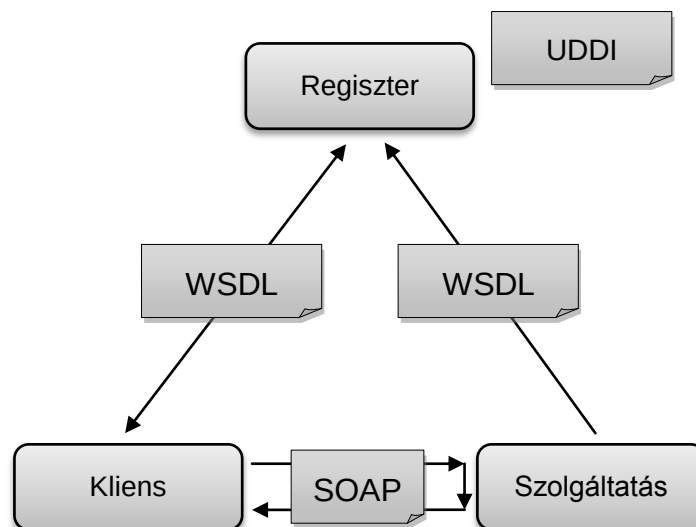
A kliensek veszik igénybe a szolgáltatások által nyújtott üzleti funkcionalitásokat. A kliensek számára a szolgáltatás tárházak teszik lehetővé, hogy képesek legyenek szolgáltatásokat keresni, felderíteni. A szolgáltatás tárházak számos információt képesek nyújtani a szolgáltatásokról, de a legfőbb feladata a szolgáltatások felderítése. A szolgáltatás busz a kliensek, szolgáltatások és tárházak összekapcsolására hivatott, amely technológia független kapcsolatot képes felépíteni a rendszer résztvevői között, megkönnyítve ezzel az integrációt. Ezen felül számos technológiai szolgáltatást hivatott implementálni, hogy megkönnyítse a szolgáltatások implementálását, és azok elérését.

3. WEBSZOLGÁLTATÁS TECHNOLÓGIA

A Weben megjelenő szolgáltatásokat egy ideig csak böngészőn keresztül érhettük el. A használat során a kliens oldali böngésző program a felhasználótól, vagy korábban látogatott weblapoktól kapott URL címek alapján távoli web-szerverek által nyújtott szolgáltatást vesz igénybe. Ez a szolgáltatás azonban kizárólag információ le- és feltöltést jelent. Később azonban az újabb szolgáltatások kezdtek HTTP alapú interfészeket biztosítani, hogy különböző információkhoz jussunk anélkül, hogy megnyitottuk volna a böngészőnket. Ezen információkat akár saját szolgáltatás létrehozásánál is felhasználhatjuk. Az így kialakult szolgáltatások hálózatát igyekeztek szabványosítani, létrehozva ezzel a webszolgáltatások technológiáját.

3.1 Webszolgáltatás architektúra

A webszolgáltatás egy olyan technológia, amely gépek közötti interakciót tesz lehetővé a számítógépes hálózaton keresztül.



3. ábra Webszolgáltatások architektúrája

A webszolgáltatások alapját három szabvány képi: SOAP, WSDL, UDDI. A SOAP a platform független üzenetváltást teszi lehetővé, a WSDL a webszolgáltatások interfészeinek szabványos leírását, az UDDI pedig a webszolgáltatások közzétételét, és a felderíthetőségét biztosítja központi, interneten elérhető regiszterek segítségével.

Ahhoz, hogy egy kliens le tudjon kérni egy szolgáltatást, ahhoz először a szolgáltatásnak regisztrálnia kell magát egy UDDI regiszterben, ahol a WSDL leírón kívül számos információ kerül tárolásra a szolgáltatással kapcsolatban. A kliens az UDDI regiszteren keresztül tudja a szolgáltatást lekérni, amely a szolgáltatás WSDL leírójával válaszol, ha a leíró gazdája a keresett szolgáltatás. A kliens ezután SOAP üzenetek váltásával képes igénybe venni a szolgáltatás funkcióit. A három fentebb említett szabvány specifikációját a következőkben részletezem.

3.1.1 SOAP

A SOAP (Simple Object Access Protocol) egy üzenetküldésre használt, XML alapú protokoll. A SOAP protokoll használatával teremtenek kapcsolatot a különböző webszolgáltatások egymás között. A SOAP üzenet alapjában véve egy olyan egyirányú protokoll, amely információ továbbítást valósít meg SOAP csomópontok között, platform- és nyelv független módon. A SOAP specifikációja alapján egy SOAP üzenet 4 fő részből áll:

- `<envelope>` – az üzenet gyökér eleme, mintegy bekeretezi az egész üzenetet. Az elem megadása kötelező.
- `<header>` – opcionális, az üzenet specifikus információit tartalmazza, mint például átviteli protokoll, karakterkódolás, stb.
- `<body>` – kötelező, az üzenet elsődleges információ tartalmát hordozza. A SOAP tervezésének egyik fő célja az volt, hogy adatokat, és azok pontos specifikációját lehessen megjeleníteni az üzenetben, míg a másik fő cél, hogy távoli eljárás hívásokat (Remote Procedure Call) lehessen vele becsomagolni. Az előbbi esetben a tetszőleges adatok megfelelő XML szintaktikával kerülhetnek bele a törzsbe. Az utóbbi esetben

azonban, egy RPC megvalósításához az üzenetnek tartalmaznia kell a következő információkat:

- a cél SOAP csomópont címét,
 - a metódus, vagy eljárás nevét,
 - a paraméterként átadott argumentumok azonosítóját és értékét, valamint a visszatérési értéket, és az output paramétereket.
 - az üzenetváltási mintát, amelyet az RPC végrehajtásánál alkalmazni kell.
- `<fault>` (hiba) – hiba esetén a `<body>` elem tartalmazhat egy `<fault>` tag-et, amely megadja a hiba kódját, valamint tartalmazhat egy olyan részt (`<reason>`-tag), amely emberi feldolgozás céljára szövegesen leírja a hiba lehetséges okait. A `<fault>` elem megjelenése és tartalma erősen függ attól, hogy a SOAP rendszer hogyan kötődik az alatta fekvő tényleges végrehajtásért felelős réteghez.

A W3C SOAP specifikáció egy ajánlás, melynek százas nagyságrendű implementációja van. Bár mindegyik implementáció a W3C ajánlást valósítja meg, az ajánlásban nyitva hagyott kérdések miatt a megvalósítások eltérőek, és ezért gyakran nem teljesen kompatibilisek.

Példa SOAP kérésre:

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
  <soap:Body xmlns:m="http://www.example.org/stockquote">
    <m:GetLastTradePrice xmlns:m="Some-URI">
      <m:tickerSymbol>IBM</m:tickerSymbol>
    </m:GetLastTradePrice>
  </soap:Body>
</soap:Envelope>
```

Példa SOAP válaszra:

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
  <soap:Body xmlns:m="http://www.example.org/stockquote">
```

```
<m:GetLastTradePriceResponse xmlns:m="Some-URI">
  <m:price>34.5</m:price>
</m:GetLastTradePriceResponse>
</soap:Body>
</soap:Envelope>
```

3.1.2 WSDL

A WSDL (Web Services Description Language), a webszolgáltatások leírására szolgáló dokumentum. A WSDL a webszolgáltatás nyilvános felületét írja le. Ez egy XML-alapú szolgáltatás-leírás a webszolgáltatással történő kommunikációról. A leírásban szerepelnek a protokollkötések, az adatok típusai és az üzenetek, amelyek az adott webszolgáltatások használatához szükségesek. Továbbá az üzenet továbbításához használt protokollok és a szolgáltatás elérhetősége. A leggyakrabban használt protokoll az üzenetek továbbítására a HTTP.

A WSDL leírás segítségével meg tudjuk találni a webszolgáltatást, kapcsolódni tudunk hozzá, illetve a leírásból tudjuk, hogy a szolgáltatás milyen formában fog válaszolni. A WSDL specifikációja hat főelemre bontja a WSDL-t:

- `<definitions>` - ez a gyökér elem, szerepelnie kell minden WSDL dokumentumban.
- `<message>` - egyirányú üzenetet ír le, ami lehet kérés, vagy válasz. Definiálja az üzenet nevét, valamint a paramétereit, vagy visszaadott értékeit.
- `<portType>` - ez az elem foglalja magába a szolgáltatás által nyújtott műveleteket, és a műveletek végrehajtásához szükséges üzeneteket.
- `<types>` - a kliens és a szerver között alkalmazott típusleírást tartalmazza. Hasonlóan a SOAP-hoz, az XML Schema specifikációt használja. Ha a szolgáltatás csak az XML Schema egyszerű típusait használja, nincs szükség a `<types>` elemre.
- `<binding>` - ezen elem alatti definíciók határozzák meg a protokoll (webszolgáltatásokról lévén szó tipikusan SOAP protokoll) és az adatátvitel tulajdonságát.

- `<service>` - a meghívandó szolgáltatás helyét határozza meg. Általában egy URL-t foglal magába a SOAP szolgáltatás eléréséhez.

További két elemet alkalmazhatunk egy WSDL leírásban:

- `<documentation>` - ember által olvasható magyarázó szöveget tartalmaz, bármely WSDL elem tartalmazhatja.
- `<import>` - más WSDL dokumentum, vagy XML Schema importálására alkalmazhatjuk.

Példa egy WSDL leíróra:

```
<?xml version="1.0" encoding="utf-8"?>
<definitions
  name="StockQuote"
  targetNamespace="http://example.com/stockquote/"
  xmlns:tns="http://example.com/stockquote/"
  xmlns:xsd1="http://example.com/stockquote/schema/"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns="http://schemas.xmlsoap.org/wsdl/">

  <types>
    <schema
      targetNamespace="http://example.com/stockquote/schema/"
      xmlns="http://www.w3.org/2001/XMLSchema">
      <element name="TradePriceRequest">
        <complexType>
          <all>
            <element name="tickerSymbol" type="string"/>
          </all>
        </complexType>
      </element>
      <element name="TradePrice">
        <complexType>
          <all>
            <element name="price" type="float"/>
          </all>
        </complexType>
      </element>
    </schema>
  </types>

  <message name="GetLastTradePriceInput">
    <part name="body" element="xsd1:TradePriceRequest"/>
  </message>
```

```

<message name="GetLastTradePriceOutput">
  <part name="body" element="xsd1:TradePrice"/>
</message>

<portType name="StockQuotePortType">
  <operation name="GetLastTradePrice">
    <input message="tns:GetLastTradePriceInput"/>
    <output message="tns:GetLastTradePriceOutput"/>
  </operation>
</portType>

<binding name="StockQuoteSoapBinding" type="tns:StockQuotePortType">
  <soap:binding style="document"
    transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="GetLastTradePrice">
    <soap:operation
      soapAction="http://example.com/GetLastTradePrice"/>
    <input> <soap:body use="literal"/></input>
    <output><soap:body use="literal"/></output>
  </operation>
</binding>

<service name="StockQuoteService">
  <port name="StockQuotePort" binding="tns:StockQuoteSoapBinding">
    <soap:address location="http://location/sample"/>
  </port>
</service>
</definitions>

```

3.1.3 UDDI

Az UDDI (Universal Description, Discovery and Integration) szolgáltatások regisztrálására, felkutatására és integrálására használt technikai specifikáció. Az UDDI a különböző ipari és üzleti ágazatokon átívelő, webszolgáltatásokat támogató specifikációk rendszere. Az UDDI regiszter nem más, mint a SOA elemeként definiált *szolgáltatás tárház* megvalósulása. Az UDDI specifikációja szerint az UDDI három fő részből áll:

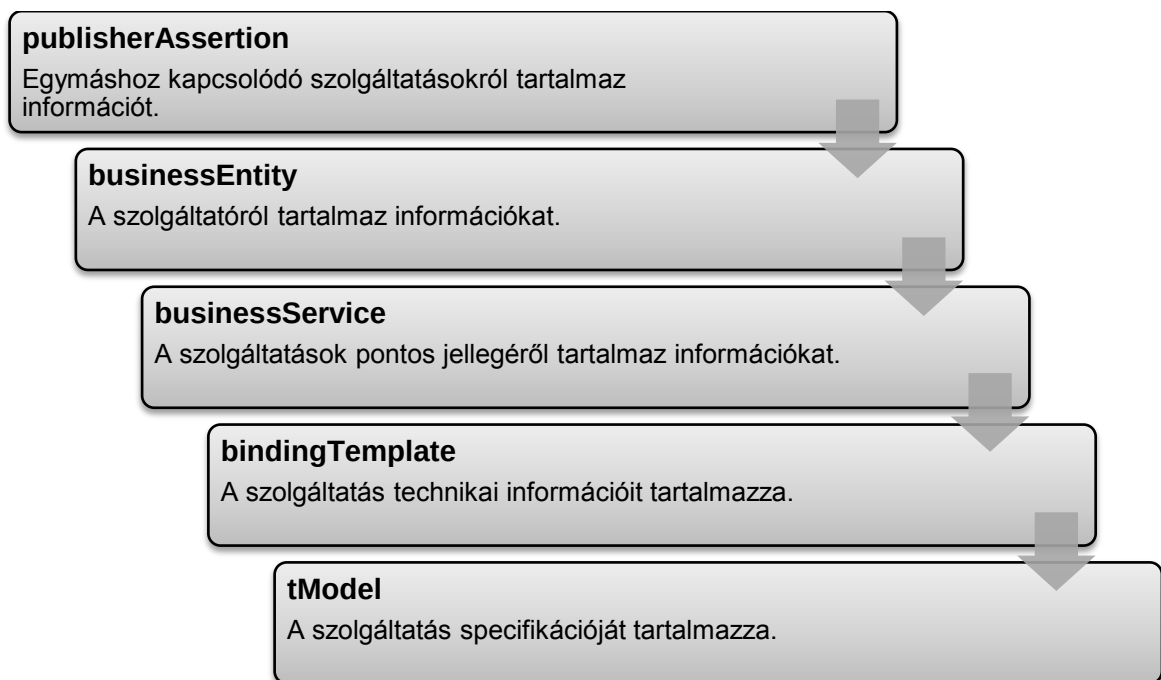
- **UDDI adatmodell** - ez egy XML Schema definíció üzleti vállalkozások, szolgáltatások, webszolgáltatások és azok kapcsolatainak leírására.
- **UDDI API** - SOAP alapú üzenetküldés API, az UDDI-ben adatok keresésére és közzétételére szolgál.

- **UDDI URB** - (UDDI Business Registry) Az UDDI specifikáció megvalósulásai, weboldalak, melyeken UDDI-hez kapcsolódó műveleteket lehet végezni.

Az UDDI-ben tárolt adatokat 3 csoportba soroljuk:

- **Fehér oldalak** - a vállalkozásokkal, és azok szolgáltatásaival kapcsolatban tartalmaznak információkat.
- **Sárga oldalak** – a vállalati szolgáltatások kategorizálva jelennek meg benne.
- **Zöld oldalak** - a szolgáltatások eléréséhez szükséges technikai információkat tartalmazzák.

3.1.3.1 Az UDDI adatmodell



4. ábra Az UDDI adatmodell

Az UDDI a következő alap adatszerkezeteket definiálja az adatok tárolására, amelyek segítségével adatokat helyezhetünk el egy UDDI regiszterben:

- `publisherAssertion`
- `businessEntity`
- `businessService`
- `bindingTemplate`
- `tModel`

A `businessEntity` adatszerkezet

A `businessEntity` adatszerkezet ábrázolja a szolgáltatóhoz tartozó üzleti entitást. Minden egyes `businessEntity` példányt egyértelműen azonosítja a `<businessKey>` attribútuma. Az attribútum megadása opcionális, ha nem adjuk meg, akkor a nyilvántartó generál egyet.

Az adatszerkezet elemei:

- `<discoveryURLs>` - olyan URL-ek listája, amelyek más file alapú, szolgáltatás kereső rendszerekre mutatnak.
- `<name>` - az üzleti entitás neve, szöveges formátumban. Több `<name>` elem is szerepelhet, melyek más-más nyelven tartalmazzák az információt.
- `<description>` - szöveges leírása az üzleti entitásnak, szintén több ilyen elem szerepelhet, különböző nyelven.
- `<contacts>` - csak egy ilyen elem lehet, mely az üzleti entitás elérhetőségét tartalmazza.
- `<businessServices>` - az üzleti entitás által nyújtott szolgáltatások listája.
- `<identifierBag>` - további azonosítókat tartalmaz, melyek más rendszerekben azonosítják az üzleti entitást.
- `<categoryBag>` - az üzleti entitás tevékenységi köreit adja meg.

A `businessService` adatszerkezet

A `businessService` adatszerkezet egy üzleti szolgáltatás specifikációját írja le. Minden egyes `businessService` példányt egyértelműen azonosítja a `<serviceKey>` attribútuma. Az attribútum megadása opcionális, ha nem adjuk meg, akkor a nyilvántartó generál egyet. Minden üzleti szolgáltatás pontosan egy üzleti entitáshoz tartozik, a `<businessKey>` attribútum ennek az üzleti entitásnak az azonosítóját tartalmazza.

Az adatszerkezet elemei:

- `<name>` - az üzleti szolgáltatás példány nevét adja meg, több is szerepelhet belőle.
- `<description>` - az üzleti szolgáltatás szöveges leírását adja, több ilyen elem is szerepelhet, különböző nyelven.
- `<categoryBag>` - azokat az üzleti ágazatokat azonosítja, amelyekhez az üzleti szolgáltatás tartozik (a sárga oldalak generálásánál van szerepe).
- `<bindingTemplates>` - elem a biztosított webszolgáltatások technikai leírása.

A `bindingTemplate` adatszerkezet

A `bindingTemplate` adatszerkezet egy üzleti szolgáltatás technikai információit hordozza magában. Minden egyes `bindingTemplate` példányt egyértelműen azonosítja a `<bindingKey>` attribútum. A `<serviceKey>` elem a szolgáltatást azonosítja, amelyhez a `bindingTemplate` tartozik.

Az adatszerkezet elemei:

- `<description>` - jelentése azonos a korábbiakkal, rövid leírást adhatunk a példányról, akár több nyelven is.
- `<accessPoint>` - a szolgáltatás használatához szükséges információt tartalmazó string. Nincs megkötés az információ mibenlétére. Általában egy hálózati cím, de lehet e-mail-cím vagy akár egy telefonszám is.
- `<hostingRedirector>` - elavult, funkcióját az `<accessPoint>` hordozza magában.

- `<tModelInstanceDetails>` - egy vagy több `tModelInstanceInfo` példányt tartalmaz. A `tModelInstanceInfo` példányokban lévő `<tModelKey>` attribútumok segítségével kereshetünk kompatibilis webszolgáltatásokat.
- `<categoryBag>` - hasonló a korábban leírt azonos nevű elemhez: annak a webszolgáltatásnak tartalmazza a felhasználási területeit, amelyet a `bindingTemplate` ír le.

A `tModel` adatszerkezet

A `tModel` adatszerkezet a szolgáltatás viselkedéséről ad információt, hogy az milyen konvenciókat, specifikációkat követ. Minden egyes `tModel` egyedet egyértelműen azonosít a `<tModelKey>` attribútum. A `tModel` példányok az UDDI nyilvántartóban olyan technikai leírások, specifikációk, amelyeket a kulcsukra hivatkozó webszolgáltatásoknak meg kell valósítaniuk.

Az adatszerkezet elemei:

- `<name>`, `<description>` - használatuk és céljuk megegyezik a korábban tárgyalt, azonos nevű elemekkel.
- `<overviewDoc>` - hivatkozásokat adhatunk meg további leírásokra, és előírásokra, a `tModel` példány használatát illetően.
- `<identifierBag>`, `<categoryBag>` - használatuk, és céljuk ugyanaz, mint ahogy azt korábban láttuk a `businessEntity` struktúrájánál.

A `publisherAssertion` adatszerkezet

Sok olyan vállalat és szervezet van, amelyet nem lehet leírni egyetlen `businessEntity` elemmel. Ennek általában az az oka, hogy a leírásaik, és ennek megfelelően a profiljaik szerteágazóak. A legegyszerűbb megoldás több üzleti entitást regisztrálni. Ebben az esetben viszont valamilyen módon jelölni kell, hogy ezek az entitások egymáshoz kapcsolódnak. Erre szolgálnak a `publisherAssertion` struktúra példányai.

Az adatszerkezet elemei:

- <fromKey>, <toKey> - azokat az üzleti entitásokat adja meg, amelyek egymással valamilyen kapcsolatban vannak.
- <keyedReference> - hivatkozás a kapcsolatot leíró adatszerkezetre.

A két businessEntity példányt, amelyek között fennáll a kapcsolat, egyértelműen azonosítják a <fromKey> és a <toKey> elemek. Magát a kapcsolatot a <keyedReference> tag-en belül írhatjuk le.

A WSDL példához tartozó UDDI businessService és bindingTemplate adatszerkezetek:

```
<businessService
  serviceKey="102b114a-52e0-4af4-a292-02700da543d4"
  businessKey="1e65ea29-4e0f-4807-8098-d352d7b10368">
  <name>Stock Quote Service</name>
  <bindingTemplates>
    <bindingTemplate
      bindingKey="f793c521-0daf-434c-8700-0e32da232e74"
      serviceKey="102b114a-52e0-4af4-a292-02700da543d4">
      <accessPoint URLType="http">
        http://location/sample
      </accessPoint>
      <tModelInstanceDetails>
        <tModelInstanceInfo
          tModelKey="uuid:49662926-f4a5-4ba5-b8d0-32ab388dadda">
            <description xml:lang="en">
              The wsdl:binding that this wsdl:port implements. The instanceParms
specifies the port local name.
            </description>
            <instanceDetails>
              <instanceParms>StockQuotePort</instanceParms>
            </instanceDetails>
          </tModelInstanceInfo>
          <tModelInstanceInfo
            tModelKey="uuid:e8cf1163-8234-4b35-865f-94a7322e40c3">
              <description xml:lang="en">
                The wsdl:portType that this wsdl:port implements.
              </description>
            </tModelInstanceInfo>
          </tModelInstanceDetails>
        </bindingTemplate>
      </bindingTemplates>
      <categoryBag>
      <keyedReference
        tModelKey="uuid:6e090afa-33e5-36eb-81b7-1ca18373f457"
        keyName="WSDL type"
```

```

        keyValue="service" />
    <keyedReference
        tModelKey="uuid:d01987d1-ab2e-3013-9be2-2a66eb99d824"
        keyName="service namespace"
        keyValue="http://example.com/stockquote/" />
    <keyedReference
        tModelKey="uuid:2ec65201-9109-3919-9bec-c9dbefcaccf6"
        keyName="service local name"
        keyValue="StockQuoteService" />
</categoryBag>
</businessService>

```

A WSDL példához tartozó portType típusú UDDI tModell adatszerkezet:

```

<tModel tModelKey="uuid:e8cf1163-8234-4b35-865f-94a7322e40c3" >
    <name>
        StockQuotePortType
    </name>
    <overviewDoc>
        <overviewURL>
            http://location/sample.wsdl
        </overviewURL>
    </overviewDoc>
    <categoryBag>
        <keyedReference
            tModelKey="uuid:d01987d1-ab2e-3013-9be2-2a66eb99d824"
            keyName="portType namespace"
            keyValue="http://example.com/stockquote/" />
        <keyedReference
            tModelKey="uuid:6e090afa-33e5-36eb-81b7-1ca18373f457"
            keyName="WSDL type"
            keyValue="portType" />
    </categoryBag>
</tModel>

```

A WSDL példához tartozó binding típusú UDDI tModell adatszerkezet:

```

<tModel tModelKey="uuid:49662926-f4a5-4ba5-b8d0-32ab388dadda">
    <name>
        StockQuoteSoapBinding
    </name>
    <overviewDoc>
        <overviewURL>
            http://location/sample.wsdl
        </overviewURL>
    </overviewDoc>
    <categoryBag>
        <keyedReference
            tModelKey="uuid:d01987d1-ab2e-3013-9be2-2a66eb99d824"
            keyName="binding namespace"

```

```

        keyValue="http://example.com/stockquote/" />
<keyedReference
  tModelKey="uuid:6e090afa-33e5-36eb-81b7-1ca18373f457"
  keyName="WSDL type"
  keyValue="binding" />
<keyedReference
  tModelKey="uuid:082b0851-25d8-303c-b332-f24a6d53e38e"
  keyName="portType reference"
  keyValue="uuid:e8cf1163-8234-4b35-865f-94a7322e40c3" />
<keyedReference
  tModelKey="uuid:4dc74177-7806-34d9-aecd-33c57dc3a865"
  keyName="SOAP protocol"
  keyValue="uuid:aa254698-93de-3870-8df3-a5c075d64a0e" />
<keyedReference
  tModelKey="uuid:e5c43936-86e4-37bf-8196-1d04b35c0099"
  keyName="HTTP transport"
  keyValue="uuid:68DE9E80-AD09-469D-8A37-088422BFBC36" />
<keyedReference
  tModelKey="uuid:clacf26d-9672-4404-9d70-39b756e62ab4"
  keyName="uddi-org:types"
  keyValue="wsdlSpec" />
</categoryBag>
</tModel>

```

3.1.3.2 Az UDDI API-ja

Az UDDI API specifikáció SOAP alapú leírása olyan interfészeknek, melyeken keresztül az UDDI nyilvántartókban lehet műveleteket végezni: Szolgáltatásokat lehet benne publikálni, törölni, kereséseket végrehajtani, részletes információkat lekérdezni, hitelesíteni.

Publikáló műveletek

A publikáló műveletek segítségével szolgáltatásokat regisztrálhatunk, frissíthetünk az UDDI regiszterben.

- <save_business> - egy üzleti szolgáltató információit publikálja egy businessEntity adatszerkezet formájában.
- <save_service> - egy üzleti szolgáltatás specifikus információit publikálja egy businessService adatszerkezeten belül.

- `<save_binding>` - egy üzleti szolgáltatás technikai információit publikálhatjuk egy `bindingTemplate` adatszerkezeten belül.
- `<save_tModel>` - egy üzleti szolgáltatáshoz tartozó `tModel` adatszerkezet publikálását teszi lehetővé.
- `<set_publisherAssertions>` - az üzleti szolgáltatók közötti kapcsolatok kezelésére szolgál.

Törlő műveletek

A törlő műveletek segítségével szolgáltatásokat, ahhoz kapcsolódó információkat törölhetünk az UDDI regiszterből.

- `<delete_business>` - egy üzleti szolgáltatóhoz tartozó entitás törlését valósítja meg a regiszterből.
- `<delete_service>` - egy üzleti szolgáltatás specifikus információit törölhetjük segítségével.
- `<delete_binding>` - egy üzleti szolgáltatás technikai információinak törlését valósítja meg.
- `<delete_tModel>` - egy üzleti szolgáltatás `tModel` adatszerkezeteinek törlését teszi lehetővé.
- `<delete_publisherAssertions>` - az üzleti szolgáltatók között fennálló kapcsolatok törlését teszi lehetővé.

Kereső műveletek

A kereső műveletek segítségével szolgáltatásokat kereshetünk az UDDI regiszterben.

- `<find_business>` - egy üzleti szolgáltató keresését valósítja meg.
- `<find_service>` - egy üzleti szolgáltatás specifikus információinak lekérdezését teszi lehetővé.

- `<find_binding>` - egy üzleti szolgáltatás technikai információinak lekérdezését valósítja meg.
- `<find_tModel>` - egy üzleti szolgáltatás `tModel` példányainak keresését teszi lehetővé.

Részletes információk lekérdezése

A szolgáltatásokhoz tartozó részletes információkat kérdezhetünk le az UDDI regiszterből.

- `<get_businessDetail>` - egy üzleti szolgáltató `<businessEntity>` példányának lekérdezését teszi lehetővé.
- `<get_serviceDetail>` - egy üzleti szolgáltatás `<businessService>` példányának lekérdezését teszi lehetővé.
- `<get_bindingDetail>` - egy üzleti szolgáltatás `<bindingTemplate>` példányának lekérdezését teszi lehetővé.
- `<get_tModelDetail>` - egy üzleti szolgáltatás `<tModel>` példányainak lekérdezését teszi lehetővé.

Biztonsági műveletek

- `<get_authToken>` - hitelesítési token lekérdezését valósítja meg.
- `<discard_authToken>` - hitelesítési token eldobását teszi lehetővé.

Példa egy webszolgáltatás felderítésére:

```
<find_service generic="2.0" xmlns="urn:uddi-org:api_v2">
  <categoryBag>
    <keyedReference
      tModelKey="uuid:6e090afa-33e5-36eb-81b7-1ca18373f457"
      keyName="WSDL type"
      keyValue="service" />
    <keyedReference
      tModelKey="uuid:d01987d1-ab2e-3013-9be2-2a66eb99d824"
      keyName="service namespace"
    />
  </categoryBag>
</find_service>
```

```
        keyValue="http://example.com/stockquote/" />
    <keyedReference
        tModelKey="uuid:2ec65201-9109-3919-9bec-c9dbefcaccf6"
        keyName="service local name"
        keyValue="StockQuoteService" />
    </categoryBag>
</find_service>
```

Az UDDI API-nak több implementációja van, amely lehetővé teszi a nagyobb szolgáltatók UDDI nyilvántartóinak elérését.

3.1.3.3 Az UDDI UBR

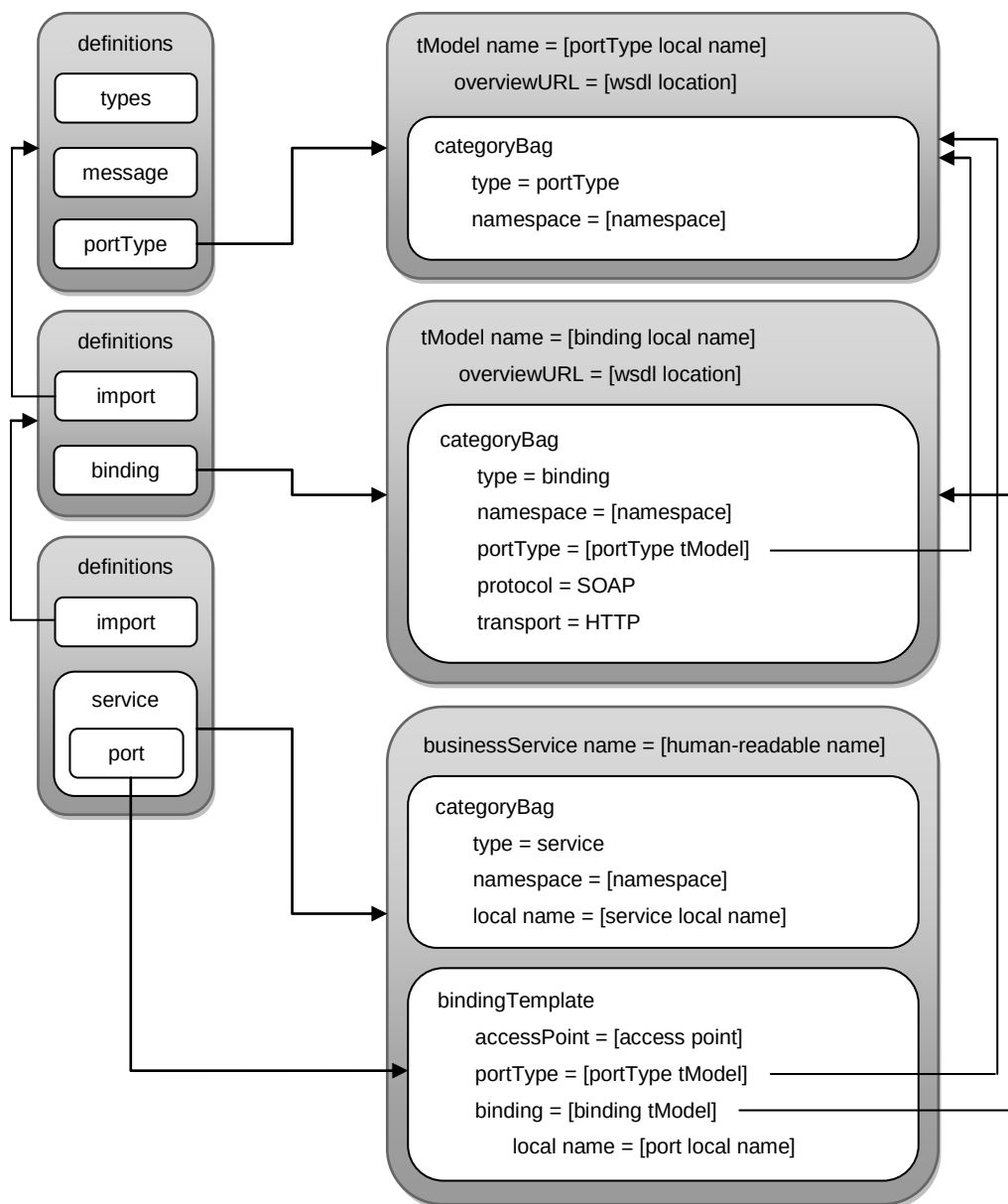
Az UDDI Business Registry kezdetben egy nyilvános weboldal volt, amelyet a legnagyobb cégek (Microsoft, SAP) együttesen működtettek 2006-ig, amikorra bebizonyosodott, hogy az UDDI 3.0 specifikáció elég robusztus, hogy a webszolgáltatások alapjául szolgáljon.

A nyilvános UBR üzemeltetése abbamaradt, azonban a különböző gyártók saját termékeikben továbbra is implementálják az UDDI lehetőségeket.

3.1.4 A WSDL és az UDDI kapcsolata

Webszolgáltatások UDDI regiszterekben való publikálása, keresése, az UDDI adatmodell és a webszolgáltatás leíró összerendelése alapján történik. Egy UDDI bejegyzés a következő összerendelés alapján készül egy WSDL leíróból:

- A `<wsdl:portType>` és `<wsdl:binding>` elemek az `<uddi:tModel>` adatszerkezetnek felelnek meg.
- A `<wsdl:service>` elem az `<uddi:businessService>` adatszerkezetnek felel meg.
- A `<wsdl:port>` elem az `<uddi:bindingTemplate>` adatszerkezetnek felel meg.

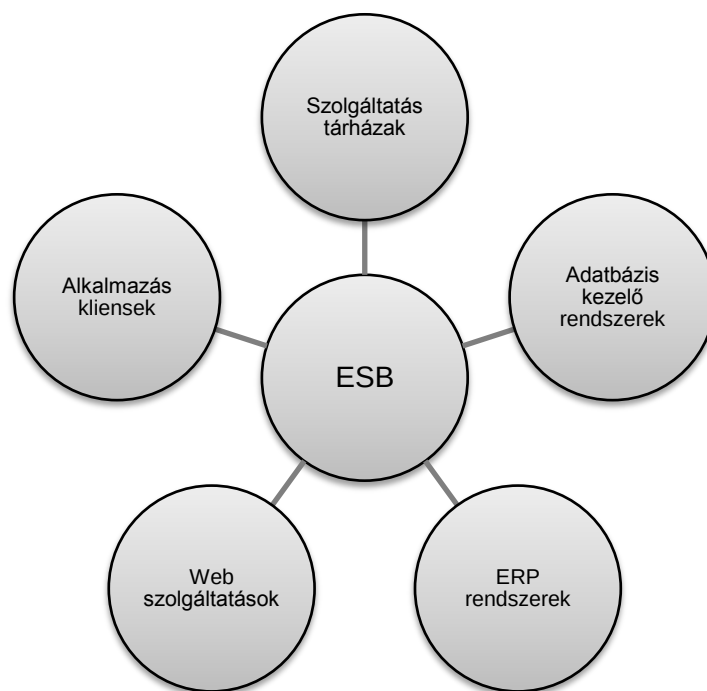


5. ábra A WSDL és az UDDI kapcsolata

3.2 ESB technológia

Az önálló webszolgáltatások még nem nevezhetők architektúrának, nem elégítik ki a SOA alapjait, céljait. Ahhoz, hogy architektúrát alkossanak, további funkciókat kell megvalósítani a szoftverrendszerekben. A mai üzleti alkalmazások elosztott architektúrájúak. Ahhoz, hogy egy tetszőleges platformon működő, és tetszőleges nyelven megírt alkalmazás hálózaton

keresztül egy másikkal optimálisan tudjon együttműködni, szolgáltatásorientált és eseményvezérelt architektúrákat kell létrehozni. Az Enterprise Service Bus (ESB) egy olyan köztes réteg, amely magában foglalja az integrációs technológiákat és a futásidejű szolgáltatásokat, lehetővé téve ezzel, hogy az üzleti szolgáltatás széles körben felhasználható legyen, és ezzel a legjobb megoldást nyújtsa az üzleti alkalmazások integrációjának problémájára. A gyakorlatban az ESB nem más, mint a SOA alapjait képező *szolgáltatás busz* technikai megvalósulása. Az ESB a technológiák közötti átjáráshoz különböző útvonalválasztó és transzformációs szolgáltatásokat valósít meg. A probléma az, hogy több óriásvállalat is készített ESB megoldásokat. Néhány a SOAP/HTTP protokollcsaládra szorítkozik, míg mások multi-protokollt támogató rendszerek. Az ESB implementációk listája a legtöbb lehetőséget nyújtó applikációs szerverektől a kisebb, specifikus ESB integrációt biztosító eszközökig terjed. Lássuk, hogy egy általános, mindenre kiterjedő ESB rendszer milyen igényeket valósít meg.



6. ábra Az ESB elhelyezkedése az infrastruktúrában

Az ESB egyik legfontosabb feladata, hogy a szolgáltatók és a szolgáltatás igénybevevői között egy közvetett kapcsolatot hozzon létre. Az igénybevevő így a szolgáltatást nem

közvetlenül, hanem a buszon keresztül éri el. Az ESB biztosítja az üzenetek megérkezését, felmentve az alkalmazást ennek implementálása alól. Az ESB adat-transzformációt alkalmaz a különböző szolgáltatások által használt, egymással nem kompatibilis protokollok, adatok, üzenetek feloldására. Egyrészt ez teszi lehetővé a különböző technológiával készült (Java, .Net, stb.) szolgáltatások számára, hogy egymással kommunikáljanak, másrészt ennek segítségével, a már meglévő rendszereinkbe könnyebben tudjuk integrálni az új szolgáltatásokat, növelve az üzleti folyamatok szervezésének rugalmasságát.

A szolgáltatásorientált rendszerünk egészen komplex lehet, az alkalmazások menedzselése bonyolulttá válhat az alkalmazások számának növekedésével. Az ESB lehetővé teszi ezen szolgáltatásokhoz kapcsolódó konfigurációk, meta adatok megadását, amelyeket később tetszőlegesen megváltoztathatunk anélkül, hogy a szolgáltatásunkat újrafordítottuk, illetve telepítettük volna. Emellett az ESB számos futásidejű szolgáltatást valósít meg, amelyek segítik a rendszer felügyeletét, monitorozását és a hibakeresést.

Az ESB egyik feladata a szolgáltatások között terjedő üzenetek titkosítása. Ennek segítségével, a szolgáltatás létrehozásánál nem kell foglalkoznunk a titkosítási protokollok implementációjával.

Ahhoz, hogy az üzenetek eljussanak a megfelelő szolgáltatáshoz, az ESB-nek lehetővé kell tennie a szolgáltatások címzését, és az üzenetek irányítását, azaz a routing-ot. Léteznie kell egy szolgáltatások leírását és elérhetőségét tartalmazó nyilvántartásnak, ami nem más, mint a már korábban definiált szolgáltatás regiszter. Webszolgáltatások esetén ezt a nyilvántartást a már definiált UDDI protokoll valósítja meg.

Végül, de nem utolsó sorban az ESB lehetővé teszi különböző üzleti folyamatok definiálását az erre a célra alkotott XML alapú leírók segítségével. Webszolgáltatások esetén ilyen nyelv a BPEL4WS (Business Process Execution Language for Web Services).

Az ESB fenti jellemzői a webszolgáltatásokhoz kapcsolódó specifikációk alapján a következő módon valósulnak meg:

- URL címzést használva, a létező HTTP és DNS infrastruktúra biztosítja a routing-ot és a webszolgáltatások elhelyezkedésének átlátszóságát.
- SOAP/HTTP támogatja a kérdés/válasz üzenetküldési mintát. A HTTP transzport protokoll széles körben elérhető.
- A SOAP és a WSDL nyílt, implementáció független üzenetküldő és interfész leíró modellek.

3.3 BPEL4WS

A BPEL4WS egy XML alapú leírónyelv, amely WSDL specifikációra épül. Feladata a WSDL leírókkal definiált webszolgáltatások közötti üzleti folyamatok leírása. Az üzleti folyamatok definiálását egy magasabb, absztrakt szintre emeli, könnyebbé téve azok leírását. A konkrét aktivitások implementációját elrejtí a felhasználó elől. A BPEL4WS specifikációja alapján a BPEL4WS nyelv a következő fő részekből áll:

- `<process>` - az üzleti folyamat legkülsőbb eleme, gyökere.
- `<partnerLinks>` - ezen az elemen belül azon résztvevőket, szolgáltatásokat definiáljuk, amelyek részt vesznek az üzleti folyamatban.
- `<variables>` - ezen elemen belül definiáljuk azokat az adatváltozókat, amelyeket az üzleti folyamat során használunk, dolgozunk fel. A változók megtartják az értéküket két üzenetváltás között.
- `<faultHandler>` - ez a rész tartalmazza a hibakezelő részt, amely definiálja azon aktivitásokat, amelyeknek le kell futniuk adott hiba bekövetkeztekor. Minden hibának egyedi névvel kell rendelkeznie.
- `<sequence>` - ez a rész definiálja magát az üzleti folyamatot, amely 3 fő részből áll. A `<receive>` elemen belül definiáljuk azt, hogy hogyan fogadjuk a szolgáltatáskérést, a `<reply>` elemen belül pedig a választ. Ezek gyakorlatilag a szolgáltatás input és output műveleteinek képzelhetőek el. A két elem között történik a végrehajtás menetének megadása a `<flow>` elemen belül.
- `<scope>` - azon összefüggő aktivitások hatáskörét határozza meg, amelyek befolyásolják egymás működését. Az elem ugyanúgy tartalmaz `<partnerLinks>`,

`<variables>`, `<faultHandler>`, `<sequence>` elemeket, mint a gyökér `<process>` elem. A `<scope>` elemek tetszés szerint egymásba ágyazhatóak.

Az üzleti logikát az aktivitások összessége határozza meg. Ezek két csoportja az alap és a vezérlési szerkezet aktivitások. Az alap aktivitások azok, amelyek az üzleti folyamatok elemi lépéseit határozzák meg. A vezérlési szerkezet aktivitások pedig azok, amelyek különböző vezérlési szerkezeteket valósítanak meg.

3.3.1 Alap aktivitások

- `<invoke>` - a szolgáltatók által biztosított webszolgáltatások hívására szolgál. A hívások tipikusan a szolgáltatások műveletein operálnak. A műveletek lehetnek kérés-válasz típusúak, vagy egyirányúak. Az `<invoke>` egyirányú kérés típusú műveletet hajt végre a szolgáltatáson.
- `<receive>` - egyirányú válasz műveletet hajt végre a webszolgáltatáson.
- `<reply>` - kétirányú kérés-válasz típusú műveletet hajt végre a szolgáltatáson.
- `<assign>` - egy változó értékét másolja egy másik változóba. A másolás során, az értékeken kifejezéseket, egyszerű számításokat alkalmazhatunk új értékek előállítására.
- `<throw>` - kivételek explicit kiváltására szolgál. A kivételnek egyedi névvel kell rendelkeznie (lásd `<faultHandler>` elemnél).
- `<wait>` - késleltetés megadására szolgál, amely lehet egy időtartam, vagy egy adott időpont.
- `<empty>` - az üres utasítás megfelelője.
- `<extensionActivity>` - a leírónyelv lehetővé teszi, hogy saját aktivitásokat definiáljunk, amelyek nem részei a specifikációnak. Ezen aktivitásokat kiterjesztett aktivitásoknak nevezzük. Minden kiterjesztett aktivitásnak egyéni névtérrel kell rendelkeznie, amely nem eleme a BPEL4WS szabványnak.
- `<exit>` - ezen elem hatására az üzleti folyamat példánya azonnal leáll. Az üzleti folyamat minden aktivitásának azonnal véget kell érnie anélkül, hogy azok valamilyen más tevékenységet kiváltanának.

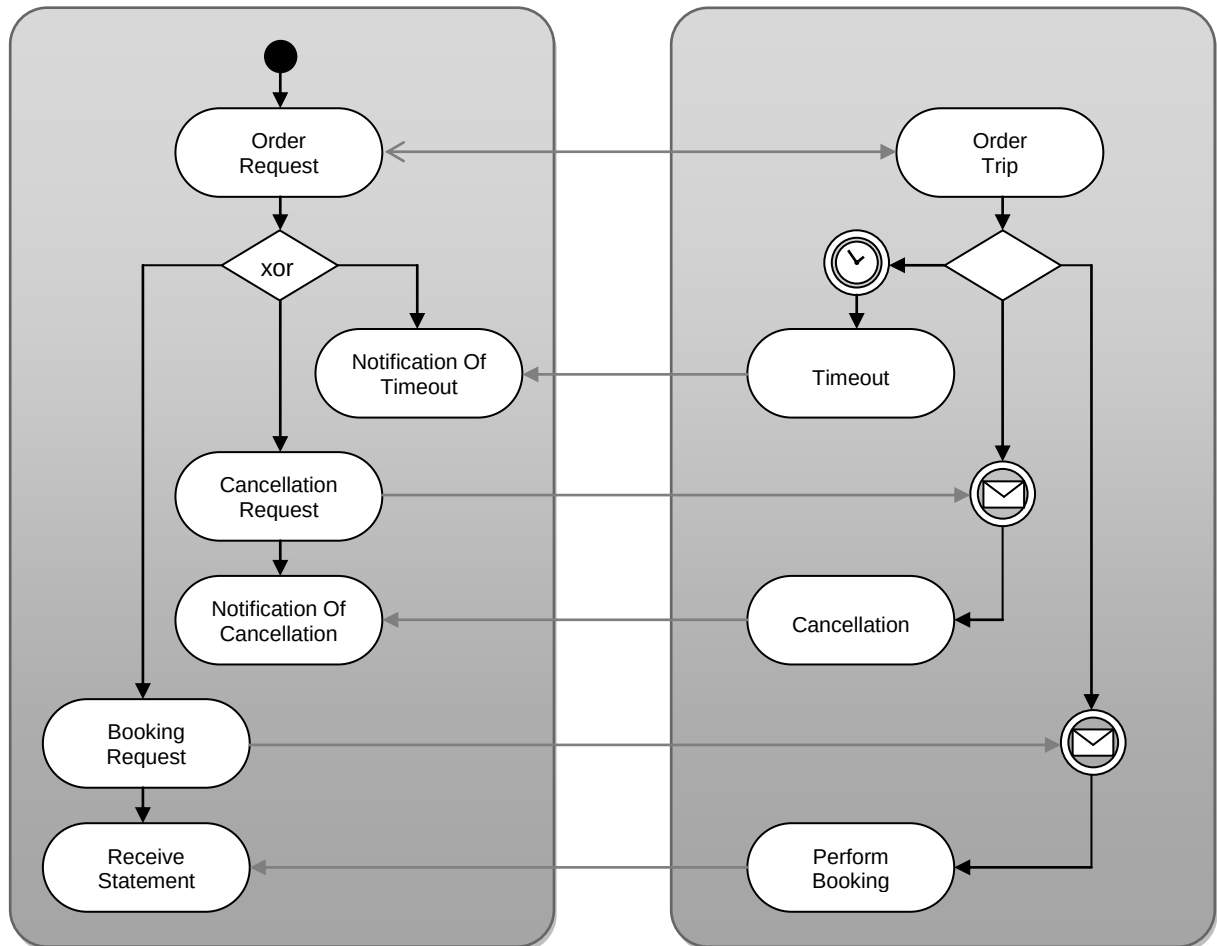
- `<rethrow>` - kivételkezelőben használatos. Már elkapott kivételek újbóli kiváltására szolgál.

3.3.2 Vezérlési szerkezet aktivitások

- `<sequence>` - a szekvencia elem egy vagy több aktivitást tartalmaz, amelyek egymás sorrendben, egymás után hajtódnak végre.
- `<if>` - feltételes működés leírására használatos. Működésük hasonló az imperatív nyelvekből ismert feltételes utasítás működéséhez.
- `<while>` - kezdőfeltételes ciklus megadására szolgál. A ciklusmag addig fut le, amíg a feltétel hamissá nem válik.
- `<repeatuntil>` - végfeltételes ciklus megadására szolgál. Először végrehajtódnak a `<repeatuntil>` elemben definiált aktivitások, majd kiértékelődik a feltétel. A ciklusmag addig fut le, míg a feltétel igazzá nem válik.
- `<pick>` - ezen aktivitás egy eseményhalmaz pontosan egy elemének bekövetkeztére vár, majd ezen eseményhez tartozó aktivitást futtatja le. Miután egyet kiválasztott, a többi esemény soha többé nem érhető el adott példányon belül.
- `<flow>` - konkurens és szinkron futást valósít meg az aktivitások között. Egy `<flow>` aktivitás befejeződik, ha az összes általa tartalmazott aktivitás befejeződik.
- `<foreach>` - a `<scope>` elemben definiált aktivitásokat futtatja le $N+1$ -szer, ahol N a `<finalCounterValue>` és a `<startCounterValue>` elemekben definiált értékek különbségeként áll elő.

A mai BPEL implementációk grafikus eszközöket biztosítanak az üzleti folyamatok lemodellezésére, megkönnyítve és meggyorsítva a fejlesztési folyamatot.

Egy általános jegyfoglalás (pl. repülőjegy) példa és a hozzá tartozó BPEL folyamat a következőképpen néz ki:



7. ábra Példa jegyfoglalásra

A fenti példához tartozó BPEL folyamat:

```

<process name="ReservationAndBookingTickets"
  targetNamespace="http://example.com/travelagent"
  xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
  xmlns:taw="http://example.com/travelagent/wsd1"
  abstractProcess="yes">

  <partnerLinks>
    <partnerLink name="TravelAgency"
      partnerLinkType="taw:TravelAgencyService"
      partnerRole="Customer"

```

```

        myRole="TravelAgent"/>
    </partnerLinks>

    <correlationSets>
        <correlationSet name="reservationCorrelation"
            properties="taw:reservationID"/>
    </correlationSets>

    <sequence>
        <receive partnerLink="TravelAgency"
            portType="taw:InterfaceOfTravelAgent"
            operation="OrderTrip"
            createInstance="yes">
            <correlations>
                <correlation set="reservationCorrelation"
                    initiate="yes"/>
            </correlations>
        </receive>
        <pick>
            <onAlarm duration="P0Y0M1D">
                <invoke partnerLink="TravelAgency"
                    portType="taw:InterfaceOfCustomer"
                    operation="NotificationOfTimeout">
                    <correlations>
                        <correlation set="reservationCorrelation"
                            pattern="out"/>
                    </correlations>
                </invoke>
            </onAlarm>
            <onMessage partnerLink="TravelAgency"
                portType="taw:InterfaceOfTravelAgent"
                operation="CancelReservation">
                <correlations>
                    <correlation set="reservationCorrelation"/>
                </correlations>
                <invoke partnerLink="TravelAgency"
                    portType="taw:InterfaceOfCustomer"
                    operation="NotificationOfCancellation">
                    <correlations>
                        <correlation set="reservationCorrelation"
                            pattern="out"/>
                    </correlations>
                </invoke>
            </onMessage>
            <onMessage partnerLink="TravelAgency"
                portType="taw:InterfaceOfTravelAgent"
                operation="PerformBooking">
                <correlations>
                    <correlation set="reservationCorrelation"/>
                </correlations>
                <invoke partnerLink="TravelAgency"
                    portType="taw:InterfaceOfCustomer"
                    operation="ReceiveStatement">
                    <correlations>
                        <correlation set="reservationCorrelation"
                            pattern="out"/>
                    </correlations>
                </invoke>
            </onMessage>
        </pick>
    </sequence>

```

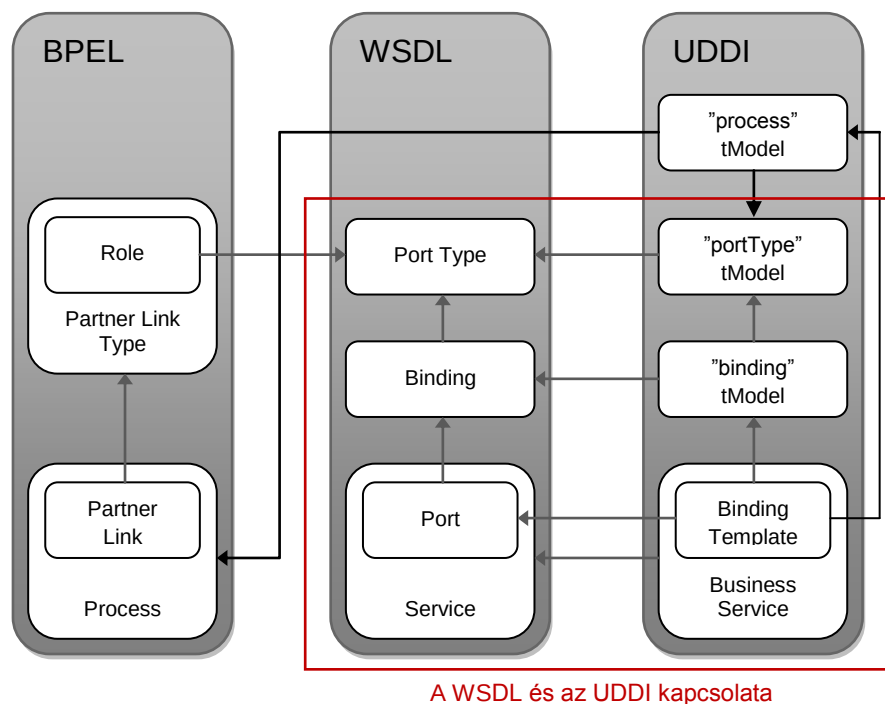
```
        </correlations>
    </invoke>
</onMessage>
</pick>
</sequence>
</process>
```

3.3.3 BPEL folyamatok publikálása UDDI regiszterekben

A BPEL4WS specifikáció lehetővé teszi, hogy BPEL folyamatokat kereshessünk, tegyünk közzé. Ahhoz, hogy ezen üzleti folyamatokat tárolhassuk az UDDI regiszterekben, a BPEL igénybe veszi a webszolgáltatásoknál megismert WSDL specifikációt, kiegészítve annak lehetőségeit. Ahhoz, hogy megértsük ezek publikálását, lássuk először a BPEL folyamatok és a WSDL kapcsolatát, majd megnézzük ezek milyen módon tárolódnak az UDDI regiszterekben.

A BPEL4WS specifikáció webszolgáltatások között teszi lehetővé, hogy üzleti folyamatokat definiáljunk. Ezen webszolgáltatásokat a már ismertetett `<partnerLink>` elem, két szolgáltatás közötti kapcsolat jellemzőit pedig a `<partnerLinkType>` elem definiálja. Ezen jellemzőket a fentebb említett szabályokkal (`<role>`) definiáljuk, amelyek gyakorlatilag a WSDL-nél definiált `<portType>` elemnek felelnek meg, azaz a szabályok definiálják azt, hogy a kommunikációban résztvevő szolgáltatások milyen műveletekkel kommunikálnak egymás között. Ezen felül a WSDL és az UDDI a már definiált módon kapcsolódik össze.

Azt már tudjuk, hogy hogyan tudjuk a BPEL folyamatban résztvevő webszolgáltatásokat publikálni, ám még ezen felül definiálnunk kell, hogy maga a BPEL `<process>` folyamat hogyan fog megjelenni az UDDI adatmodellben. Minden BPEL `<process>` folyamat egy `<uddi:tModel>` adatszerkezetben tárolódik. Ezen `tModel` adatszerkezet azon `<portType>` típusú `<uddi:tModel>` adatszerkezetekhez kapcsolódik, amely `<portType>` elemek a BPEL folyamathoz tartozó WSDL leírókban vannak definiálva. A szolgáltatások technikai információit az `<uddi:bindingTemplate>` adatszerkezet tárolja, amelyhez a `portType` típusú `tModel` elemek tartoznak. Ezen `<uddi:bindingTemplate>` adatszerkezethez kapcsolódnak a folyamatot leíró `process` típusú `tModel` elemek is.



8. ábra A BPEL és az UDDI kapcsolata

Így már az üzleti folyamatot, és a folyamatban résztvevő webszolgáltatásokat is tudjuk publikálni UDDI regiszterekben, lehetővé téve annak rugalmasságát, hogy ne csak webszolgáltatásokat kereshessünk, hanem BPEL üzleti folyamatokat is.

3.4 Összefoglalás

A webszolgáltatás technológia hivatott a *szolgáltatások* és a *szolgáltatás tárházak* megvalósítására. A webszolgáltatás technológia három szabványon alapszik. A SOAP protokoll segítségével XML alapú üzenetek küldését valósíthatjuk meg a szolgáltatások között, illetve távoli metódushívások leírását teszi lehetővé. A WSDL leíró egy webszolgáltatás nyilvános interfészeit, meta adatait tartalmazza, amely segítségével egy kliens képes az adott webszolgáltatást elérni, és igénybe venni. Az UDDI regiszterek tartalmazzák a webszolgáltatások eléréséhez szükséges információkat, azok nyilvános interfészeit. Az UDDI API specifikáció teszi lehetővé a webszolgáltatások publikálását, keresését. Az ESB valósítja meg a *szolgáltatás busz* fogalmát. Az ESB lehetővé teszi, technológia független kapcsolatot létesítsünk a szolgáltatók és a szolgáltatásokat igénybe vevők között, megvalósítja a

routingot, és egyéb technológia megoldásokat kínál. A BPEL4WS szabvány lehetővé teszi, hogy üzleti folyamatokat definiáljunk szolgáltatások között, és hogy azokat közzé tegyünk, mint elérhető szolgáltatásokat. Az ESB és a BPEL azok az szabványok, amelyek az individuális webszolgáltatásokat üzleti folyamatokká kapcsolják össze, s ezzel létrehozzák a webszolgáltatások architektúráját.

4. WEBSZOLGÁLTATÁSOK ÉS A GRID KAPCSOLATA

Ahogy az az előző szakaszban olvasható volt, a webszolgáltatás technológia önmagában egy sor olyan problémát megold, amely a szolgáltatásorientált rendszerek kapcsán felmerült. A webszolgáltatás standardok segítségével a számítógépek megérthetik, és automatizált módon használhatják egymást, az UDDI specifikációra építve pedig az aktuális környezet függvényében a mindenkori igényeik szerint választhatják meg partnereiket, természetesen akár ezt is teljesen automatizált módon. Mielőtt még azt hinnénk, hogy a webszolgáltatás technológia önmagában elegendő a szolgáltatásorientált rendszerek megalkotására, emeljük ki két hiányosságát:

- A webszolgáltatás technológia erős webes háttere következtében csak *állapotmentes* (stateless) szolgáltatások létrehozására használható, míg a valóságban nagyon gyakran *állapotörző* (stateful) szolgáltatásokra van szükség.
- A webszolgáltatás technológia statikus, vagy kvázi statikus környezetekre lett kifejlesztve, míg a szolgáltatásorientált rendszereknek dinamikus környezetekben kell üzemelnie.

A hiányosságok leküzdése miatt a Global Grid Forum bevezette az OGSA (Open Grid Services Architecture) elveket, mint a webszolgáltatás technológia állapotörző és dinamikus változó rendszerekben való használhatóságát biztosító kiterjesztéseket. Az OGSA elvek definiálják azt, hogy a webszolgáltatás technológia kiterjesztésével hogyan lehetne szolgáltatásorientált számítási rendszereket, úgynevezett *grideket* létrehozni.

4.1 Grid rendszerek

A grid technika a szuperszámítógépek és klaszterek továbbfejlődésének természetes folytatásának tekinthető. A szuperszámítógépek létrehozásának legfőbb motivációja az volt, hogy hogyan lehet a kor (70-es évek) nagy kihívást jelentő tudományos alkalmazásait ésszerű időn belül megoldani. Ezek az alkalmazások az akkori egyprocesszoros számítógépeken hónapokig, sok esetben évekig futottak volna. Az akkori megoldás több processzor

alkalmazása és integrálása volt egyetlen sokprocesszoros architektúrán belül. Az ilyen sokprocesszoros számítógépeket nevezték *szuperszámítógépeknek*. Ezek a 90-es évek elejére teljesen általánosan elterjedté váltak mind az iparban, mind a kutatásban. Ugyanakkor túlságosan drágák voltak ahhoz, hogy tömeges elterjedésüket lehessen várni.

A 90-es évek közepén megjelentek a PC farmok – más néven *klaszterek* – amelyek lokális hálózaton keresztül összekapcsolt nagyszámú PC-vel megközelítették a szuperszámítógépek sebességét, lényegesen alacsonyabb ár mellett. Ezeknek a klasztereknek a programozása sok tekintetben megegyezett a szuperszámítógépek programozásával, azaz a szuperszámítógépek programozására kifejlesztett eszközök azonnal használhatók voltak a klaszterek esetén is. Ily módon a klaszterek megjelenése lehetővé tette a szuperszámítógép technológia tömeges elterjedését és alkalmazását. Ma már szinte minden kutatóintézetben és egyetemi kutatóhelyen használnak klasztereket, de gyakran igaz ez a vállalatok fejlesztési részlegeire is.

A szuperszámítógépek és klaszterek megjelenése és terjedése természetesen visszahatással volt a megoldandó tudományos problémák körére is. Annak ellenére, hogy olyan számítások váltak elvégezhetővé melyekről korábban csak álmodozni lehetett, felvetődtek olyan új igények melyek már az egyedi szuperszámítógépek és klaszterek teljesítményén is túlmutattak. Ilyen problémák például a klímamodellezés, csillagászati számítások, nagy energiájú fizika problémái, kémiai molekula-modellezések, stb. A kézenfekvő megoldás hasonló volt, mint a 70-es években, de most nem egyedi processzorokat kellett integrálni, hanem egymástól földrajzilag távol lévő szuperszámítógépeket és klasztereket. Ily módon a szuperszámítógép technológia, a hálózati technológia és az elosztott rendszerek technológiájának integrálásával létrejöttek az ún. *metaszámítógépek*, melyek elsődleges célja a nagy számításigényű feladatok megoldása volt. Ezeket, a 90-es évek első felére jellemző számítási rendszereket tekinthetjük a grid első generációjának. Ezekben tipikusan a számítógépek kihasználatlan ciklusait igyekeznek munkára fogni és az így összegyűjtött ciklusokkal nagy számításidejű feladatokat meglepően rövid idő alatt elvégezni. Tipikus példák az első generációs grid rendszerekre a SETI és a Condor.

Az első generációs grid rendszerek megjelenése a nagy számításigényű feladatok megoldása mellett új igényeket is generált, mint például nagy adatmennyiségek elosztott feldolgozása, a

kollaboratív munka támogatása, ill. az erőforrások dinamikusan változó munkacsoportokon – úgynevezett virtuális organizációkon – belüli megosztása. Az új problémákat a második generációs grid rendszerek a 90-es évek második felétől egyedi igényekre kifejlesztett protokollokkal próbálták megoldani. A 2. generációs grid rendszerek tipikusan a Globus Toolkit 2-re (GT-2) épülnek, amely de facto szabvány lett ebben az időszakban. A ma üzemszerűen működő grid infrastruktúrák túlnyomó többsége még mindig ennek a 2. generációs eszköznek az eredeti, vagy valamilyen kiegészített, módosított változatát használja, és tipikusan a tudományos kutatást szolgálja. Ilyen grid infrastruktúrák a CERN által kifejlesztett LHC Grid, az USA-ban létrehozott Grid 2003, a skandináv államok infrastruktúrája a NorduGrid, vagy az Egyesült Királyság Gridje, a UK National Grid Service. Magyarországon a tudományos és ipari kutatói közösség által ingyenesen használható HunGrid tartozik ebbe a körbe.

2003-ra világossá vált, hogy a mind az ipari-üzleti, mind a kutatói közösségek által széles körben alkalmazható grideknek valamilyen standard, szolgáltatásorientált technológiára alapozva kell létrejönniük. Csakis egy általánosan elfogadott szolgáltatásorientált technológia biztosíthatja a heterogén adatbázisok, számítógépek, laboratóriumi berendezések, mérőműszerek egymással való kompatibilitását és hibatűrő működését. Az így létrehozott rendszerben a szolgáltatásként megjelenített alapkomponeensek egymás funkcióját erősítve korábban sosem látott új lehetőségeket teremthetnek mind az üzleti-ipari, mind a tudományos szféra számára.

2002-ben a Grid kutatói közösség a webszolgáltatás technológiát nevezte meg a jövő gridjének technológiai platformjaként. Ezzel megkezdődött a 3. generációs, szolgáltatásorientált gridek kora. A munka kezdetben a webszolgáltatás technológia gridek létrehozásához szükséges kiterjesztésével, majd erre az új technológiai platformra épülő grid szolgáltatások létrehozásával telt. Ennek a webszolgáltatás kiterjesztési törekvésnek lett az eredménye az OGSA (Open Grid Service Architecture) koncepció és a hozzá kapcsolódó OGSi (Open Grid Service Interfaces), illetve WSRF (Web Services Resource Framework) szabványok.

Az OGSZ szabvány felhasználásával 2003 nyarára készült el a Globus Toolkit 3-as változata. A Toolkit ezen verziója a GT-2 által kínált eszközöket OGSZ alapú webszolgáltatásokként implementálta. A GT-3 tehát alapvetően nem funkcionalitásában, hanem protokolljaiban, ezáltal elfogadottságában múlta felül a GT-2-t. Míg a GT-2 egyedi, nem nyílt szabványokon alapuló zárt rendszereket eredményezett, addig a GT-3-mal lehetőség nyílt tetszőlegesen kiegészíthető, moduláris, nyílt rendszerek felállítására. A vállalatok számára azért jelent ez mérőöldkövet, mert ekkor nyílt először lehetőségük, hogy a korábban csak laboratóriumi, kutatói környezetekben használt nagykapacitású számítási rendszereket könnyedén a saját webszolgáltatás alapú infrastruktúrájukhoz kössék.

2004-ben végül megjelent a WSRF specifikációhalmaz 1.0-ás verziója. Erre természetesen a Globus fejlesztői közösség is reagált: 2005 első felére létrehozták a Globus Toolkit 4-es, WSRF alapú Webszolgáltatásokat tartalmazó változatát. A technológiaváltás a nyújtott funkcionalitásokat szintén nem befolyásolta túlságosan, megjelentek viszont az új szabvány szerinti kommunikációs protokollok és azokat támogató fejlesztői API-k.

4.2 Állapotörző erőforrások szolgáltatásként megjelenítve

Mint ahogy az a korábbiakban olvasható volt, a webszolgáltatás technológia a HTTP hálózati protokollt és az XML adatformátumot használja a kliens és szerver programok közötti kommunikációhoz. Mivel a HTTP protokoll állapotmentes, ezért következésképpen – a web szerverekhez hasonlóan – a webszolgáltatások is állapotmentes funkcionalitást eredményeznek. Az állapotmentesség a szolgáltatás esetén azt jelenti, hogy az interfészén keresztül beérkező klienskéréseket egymástól függetlennek tekinti, emiatt nem tárol információkat a kérésekről vagy azok eredményeiről. A valóságban azonban állapotörző funkciók – főként fizikai vagy logikai erőforrások (pl. adatbázisok) – szolgáltatásként való megjelenítésére van szükség.

Természetesen állapotmentes kommunikációs protokollt használó szolgáltatás is lehet állapotörző, de ehhez a szerverprogramnak valamilyen állapotörző back-end erőforrás segítségével kell a kérések között nyilvántartandó adatokat eltárolnia. Ezt a módszert

valósítják meg a web szerverek is, amikor CGI programok vagy Java servletek segítségével adatbázisokat írnak, olvasnak. Amit lényeges kiemelni az az, hogy az állapotörző működés megvalósítására sem a web, sem a webszolgáltatás technológia nem ad standard megoldást, a fejlesztőknek minden alkalommal az adott szituációnak leginkább megfelelő megoldást kell megtalálniuk. Mivel a webszolgáltatás technológia elterjedése során világossá vált, hogy nagyon gyakran kerülnek a fejlesztők ilyen szituációba, ezért a jövő általánosnak tekintett szolgáltatásorientált technológiájának standard megoldást kell adnia ezen problémára. Az ilyen standard megoldást használva a fejlesztők gyorsan és hatékonyan lehetnek képesek erőforrásokat szolgáltatásként megjeleníteni. Az OGSA a hiányosság leküzdésére a következő megoldásokat vezette be:

1. Grid Szolgáltatás Kezelő (Grid Service Handler – GSH) és Grid Szolgáltatás Referencia (Grid Service Reference – GSR)
2. Szolgáltatás Gyár (Service Factory)
3. Szolgáltatás Adat Elem (Service Data Element – SDE)

4.2.1 A GSH és GSR koncepció

A dolgok egyértelmű azonosítására az élet minden területén szükség van. Az informatikában az operációs rendszerek azonosítóval látják el a processzeket, az adatbázisok azonosítóval látják el a táblákat és a rekordokat. Szolgáltatásorientált rendszerekben sem lesz ez másképp: a szolgáltatásokat egyértelműen azonosítani kell. Az OGSA e célra vezette be a GSH-t. A GSH-k olyan globálisan egyedi értékek, amelyekkel minden OGSA szolgáltatásnak rendelkeznie kell. A szolgáltatók és azok kliensei ezeket a GSH-kat egyértelmű azonosítóként használhatják, amikor az szolgáltatásorientált hálózat résztvevőire hivatkoznak.

A GSR a GSH-hoz hasonlóan a szolgáltatásokhoz allokalta valamiféle speciális érték. A GSR azonban nem azonosításra, hanem a szolgáltatás aktuális viselkedésének megismerésére használható. A szolgáltató tehát a GSH-val megcímkézheti szolgáltatását, míg a GSR-rel leírhatja annak aktuális viselkedését. Amíg a GSH a szolgáltatás működésének ideje alatt állandó, addig a GSR az aktuális viselkedésének függvényében változhat. A GSH és GSR

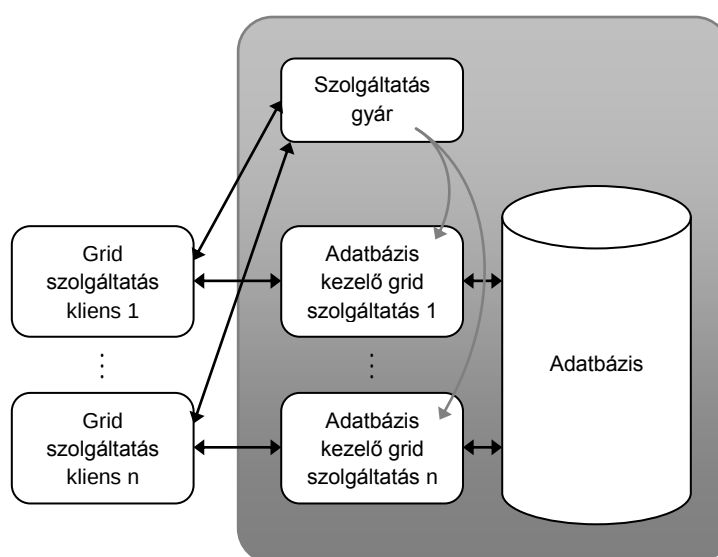
szétválasztásával az OGSA a viselkedésüket változtatni képes szolgáltatások fejlesztését kívánja támogatni. A koncepció segítségével olyan szolgáltatások készíthetők, melyek működésük közben megváltoztatják funkcionalitásukat – például úgy hogy új, eddig nem támogatott műveletek végrehajtására is lehetőséget adnak – anélkül, hogy ezzel a klienseik összeomlását okoznák. Mivel egy ilyen változás a szolgáltatás GSR-jének, de nem egy új GSH azonosítónak a bevezetését jelenti, a régi azonosító alapján a kliensek megszerezhetik és adaptálhatják magukat az új GSR-ekhez. Annak ellenére, hogy a működés közben viselkedésüket változtatni képes programok komoly implementációs kérdéseket vetnek fel, a jövő szolgáltatásorientált rendszereiben valószínűleg nem lesz ritka az ilyen magas szintű intelligencia.

4.2.2 Szolgáltatás gyáarak

Láttuk, hogy webszolgáltatások esetén a kliensek az előre, igényük felmerülésénél korábban az UDDI szolgáltatókhoz már beregisztrált szolgáltatásokat használhatják. OGSA alapú gridekben ezzel szemben az állapotörző szolgáltatások nem a kliensek megérkezése előtt, hanem kifejezetten a kliensek kérésére „jönnek létre”. A szolgáltatás gyáarak feladata éppen az állapotörző szolgáltatások létrehozása. OGSA alapú grid esetén tehát a klienseknek nem a kívánt állapotörző grid szolgáltatást kell az információs rendszer segítségével megkeresniük, hanem egy olyan szolgáltatás gyárat, amellyel a keresett szolgáltatás megfelelő tulajdonságokkal rendelkező példánya létrehozható. Ebből az is látható, hogy a szolgáltatás gyáarak ugyanúgy szolgáltatások, ahogyan az „információs rendszer” vagy a „csavarrendelés”, csak ők „állapotörző szolgáltatás létrehozás” funkciót nyújtanak.

Az új módszer valójában nem jelent nagy különbséget a webszolgáltatás technológiához képest, hiszen most nem a tényleges webszolgáltatásokat, hanem az azokat létrehozó szolgáltatás gyáarakat kell a szolgáltatóknak az információs rendszerbe regisztrálni. A grid szolgáltató programok és a kliens programok közé tehát az információs rendszer rétege mellett egy újabb réteg, a szolgáltatás gyáarak rétege ékelődik.

A szolgáltatás gyárak által létrehozott szolgáltatáspéldányok általában egy olyan új programnak vagy futási szálnak az elindítását jelentik, melyek közös erőforrást használnak más, korábban létrehozott, hasonló funkciójú példányokkal. Például egy adatbázis-kezelésre használható grid szolgáltatás példányosításakor valószínűleg nem másolódik át az eredeti adattartalom egy kihasználatlan, más programok által egyáltalán nem használt gépre, hanem a meglévő – és minden bizonnyal más kliensek által is igénybe vett – adatbázishoz indul egy újabb lekérdezőprogram. A 9. ábra ezen a konkrét példán keresztül mutatja be a szolgáltatás gyárak használatát:



9. ábra Állapotörző grid szolgáltatások létrehozása szolgáltatás gyár segítségével

Látható, hogy az adatbázis manipulálására használható grid szolgáltatás példányok a szolgáltatás gyáron keresztül hozhatóak létre. Az egyes példányokon keresztül pedig ezután a tényleges funkcionalitás – adat beírás vagy kiolvasás – elvégezhető. Mint az látható, a lekérdezőprogram példányok mind ugyanazt a közös fizikai gépet és adatbázist használják, a kliensek mégis úgy érezhetik, hogy kizárólagos használói az erőforrásoknak. Az OGSA célja a szolgáltatás gyárakkal éppen az „erőforrás virtualizáció”.

Annak ellenére, hogy az OGSA ezt a sémát írja elő állapotörző grid szolgáltatások létrehozásához, az implementációs részletekről semmit sem mond. A kívánt szolgáltatás gyár

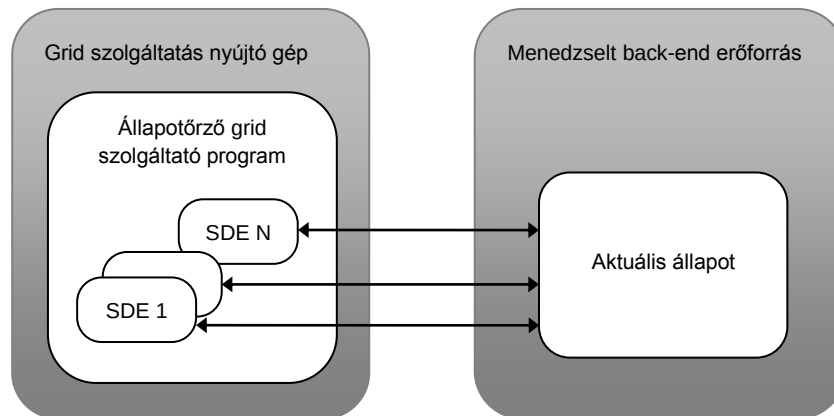
funkciót adhatja a bérbeadásra kerülő erőforráson futó külön program (mint ahogyan a 9. ábrán), a létrehozott példányokat magába záró konténer, de akár egy másik erőforrást használó, teljesen különálló alkalmazás is. Ebből következően a létrehozott példányok is több félék lehetnek. Elképzelhető, hogy a tényleges grid szolgáltatások önálló operációs rendszerbeli processzek, de akár az is előfordulhat, hogy a példányok csak a szolgáltatás gyár programon belüli futási szálak takarnak. Az OGSA nem határoz meg erre vonatkozóan semmit, a szolgáltatás gyáron keresztüli példányosítás részleteit a fejlesztőkre hagyja, akik így az eléjük kerülő konkrét szituációban mindig a legcélravezetőbb megoldást alkalmazhatják.

Ehhez kapcsolódóan annak az eldöntése is a fejlesztők dolga, hogy egy kliens példányosítási kérésének hatására tényleg új szolgáltató program példány induljon, vagy egy már korábban, más kliens számára létrehozott példányra mutató referencia adódjon csak vissza. Ez utóbbi esetben a szolgáltatás gyáraknak futási időben kell eldönteniük, hogy mely klienseknek hozzanak létre új példányt, és kik azok, akik között egyazon példány megosztható.

4.2.3 Szolgáltatás adat elemek (SDE)

Mivel a szolgáltatások célja az, hogy szerverprogramokhoz kapcsolt back-end eszközön távoli kliensek operációkat kezdeményezhessenek, ezért a szervereknek pontosan ismernie kell az általuk menedzselt fizikai eszköz, vagy gép, illetve logikai erőforrás mindenkori állapotát. Ha az aktuális állapot nem enged meg egy kliens által kezdeményezett operációt, akkor a szervernek meg kell tagadnia a kérést. A SDE koncepció éppen az ilyen – back-end erőforrásuk napra kész állapotát ismerő – szerverprogramok létrehozásához ad standardizált megoldást.

Egy SDE egy állapotörző szolgáltatáson belüli olyan adatelemet jelöl, amely a szolgáltatás által menedzselt erőforrás valamilyen attribútumát reprezentálja. A szolgáltatásoknak tehát nincs más feladatuk, mint az erőforrásaik leírásának megfelelő naprakész SDE-k tárolása. Ez a séma látható a következő ábrán:



10. ábra Szolgáltatás adat elemek és erőforrás paraméterek

Arról, hogy a szerverprogramok milyen SDE-ket tároljanak, és hogyan valósítsák meg azok naprakész nyilvántartását az OGSA semmit sem mond. Amit előír az csupán annyi, hogy a szerverprogramokon belül legyenek a back-end erőforrások mindenkori állapotának megfelelő adatmezők.

A szerveren belül nyilvántartandó SDE-k száma és típusa nyilván attól függ, hogy pontosan mi a menedzselt erőforrás és milyen adatokkal írható le annak állapottere. Az adatelemek naprakésztsége az erőforrás állapotának rendszeres lekérdezésével általában biztosítható. Jó esetben viszont a szervernek olyan erőforrást kell a gridbe kapcsolnia, amely képes őt minden, az állapotterében végbement változásról értesíteni. Az ilyen erőforrások grides szerverprogramjának csupán az erőforrástól beérkező értesítéseknek megfelelően kell a tárolt SDE-it beállítania.

4.3 Dinamikus hálózatok kezelése

A webszolgáltatás technológia alapvetően üzleti folyamatok integrálására használható. Mivel az üzleti környezetek a gridhez képest statikusak (egy vállalat nem tűnik el a piacról, egyik pillanatról a másikra, míg ugyanez egy erőforrással megtörténhet), a webszolgáltatás technológia kifejlesztésekor nem volt cél a magas dinamizmussal rendelkező hálózatok támogatása. A webszolgáltatás szemlélet szerint a vállalatok ugyanúgy fogják web szolgáltató gépeiket üzemeltetni, ahogyan azt mai IT infrastruktúrájukkal is teszik. Statikusan

webszolgáltatások üzemeltetésére allokált gépek lesznek, melyeken keresztül a kliensek a vállalatot elektronikus üzletkötés céljából a nap 24 órájában elérhetik.

A grid esetén ez a feltevés nem igaz. Mivel a gridben bármilyen erőforrás, jármű, szerszám, háztartási gép funkcionalitása megjelenhet, ezer és ezer szolgáltatás fog fel- és eltűnni minden egyes pillanatban. Egy PC, melynek tulajdonosa kávézni ment megjelenhet program végrehajtóként, egy taxi, amelyből épp kiszállt az eddigi utasa megjelenhet személyszállító szolgáltatóként, egy esztergapad a griden keresztül jelezheti a kiszolgáló robotjának, hogy befejezte a megmunkálást és jöhet a következő munkadarab. Ezen a néhány példán keresztül is látni, hogy az erőforrások szolgáltatásba „csomagolása” mennyire dinamikus hálózatot eredményez. Az OGSA a következő eszközöket vezette be a webszolgáltatás technológia ilyen irányú hiányosságainak pótlására:

1. Szolgáltatás élettartam menedzsment (Service lifetime management)
2. SDE értesítés (SDE notification)
3. HandleMap szolgáltatás (HandleMap service)

4.3.1 Szolgáltatás élettartam menedzsment

A gridek nem csak a kliensek, de a szolgáltatók szemszögéből is egy olyan dinamikusan változó halmazt jelentenek, melyekben partnereik bármikor fel-, vagy eltűnhetnek. Mivel az állapotörző erőforrások használata esetén az egyes szolgáltatás példányok a back-end erőforrásuk kisebb-nagyobb részét a klienseik számára allokálják, az előzetes bejelentés nélkül „távozó” kliensek feleslegesen lefoglalt erőforrás-darabjai plusz leterhelést, plusz költséget jelentenek. A kliensek hálózathálóból való eltűnésének hibamentes lekezelésére ad lehetőséget az szolgáltatás élettartam menedzsment. Használatával az állapotörző grid szolgáltató-programok „észrevehetik” hogy kliens nélkül maradtak, és erőforrásaik felszabadítása után, káros következmények nélkül tűnhetnek el a gridből.

Az élettartam menedzsment által alkalmazott alapötlet az, hogy a szolgáltatás gyáron keresztül létrejövő szolgáltatás példányok ne örökké, hanem csak egy adott ideig

„élhessenek”, az idő letelte után pedig automatikusan „semmisítsék meg” magukat. Ellenben ha egy szolgáltatás kliensének a példány létrehozásakor a szolgáltatás gyár által megállapított élettartam nem elég, akkor annak letelte előtt legyen lehetősége ezt jelezni, és a példány élettartamát meghosszabbítani. Ha az élettartam letelte előtt nem küld meghosszabbító kérést, akkor a példány be fogja szüntetni a nyújtott szolgáltatást, felszabadítva minden számára allokált erőforrást. Ezzel a módszerrel mind a hibás, mind az önszántukból a gridet elhagyó klienseknek lefoglalt erőforrások a lehető legrövidebb időn belül felszabadulhatnak.

4.3.2 SDE értesítés

Az utolsó vizsgálat alá kerülő OGSA által bevezetett webszolgáltatás kiterjesztés az SDE értesítés keretrendszer. Ezen ajánlás célja, hogy ne csak a kliensek kezdeményezhessenek aszinkron hívásokat a szolgáltatóknál, de a szolgáltatók is kezdeményezhessenek aszinkron hívásokat a klienseknél. Ilyen hívásokra a gridekben nagyon gyakran szükség lehet, ugyanis ha valami döntő jelentőségű változás történik egy szolgáltatásként megjelenített erőforrás, gép vagy eszköz állapotában (például kifogy az üzemanyag a fedélzeti komputere által a gridbe kapcsolt vezető nélküli targoncából), akkor erről az őt távolról használó klienst értesíteni kell. Mivel a webszolgáltatás szabvány nem teszi lehetővé az aszinkron szolgáltató-kliens kommunikációt, ezért az OGSA-nak kellett az ehhez szükséges szabályrendszert bevezetnie.

Az OGSA ajánlása az SDE alapú állapotátroláson keresztül oldja meg a problémát. Mivel a grid szolgáltatásokat üzemeltető számítógépek SDE-k segítségével tartják nyilván a menedzselte erőforrásaik mindenkor állapotát, ezért kijelenthető, hogy minden esetben, amikor egy kliens szempontjából fontos változás történik egy gridbe kapcsolt erőforrás állapotterében, olyankor az őt menedzselő számítógép is megváltoztatja legalább egy SDE-jének az értékét. E kijelentés következtében a szolgáltatóktól a kliensek felé menő kommunikáció az új SDE-k értékének a kliensek felé való elküldésével egyenlő. Mivel a webszolgáltatás szabvány minden kliens és szolgáltató-program közötti kommunikációt HTTP protokollal továbbított XML alapú SOAP üzenetként ír elő, és mivel az OGSA épp ezen platformfüggetlen kommunikációs megoldások miatt támaszkodik a webszolgáltatás technológiára, ezért a szerverektől a kliensek felé történő üzenetküldéshez is ezen szabványok használatát ajánlják.

4.3.3 HandleMap szolgáltatás

Látható volt, hogy az OGSA alapú grid szolgáltatások használatához a klienseknek megfelelő szolgáltatás gyárak felfedezésére van szüksége. Ez a felfedezés a szolgáltatás gyár GSH-jának megszerzését jelenti. A GSH ismeretében megkapható a szolgáltatás gyár funkcionalitását leíró GSR, amely alapján a kliens a tényleges funkcionalitást nyújtó szolgáltatást példányosíthatja.

A szolgáltatás gyár használat eredménye egy újabb – immár a tényleges – grid szolgáltatás, melyet egy másik GSH-GSR pár jellemez. Amíg a GSH a szolgáltatást annak egész életében identifikálni fogja, addig a GSR csak az adott időszakban jellemző rá, a későbbiekben tartalma esetleg változhat. A futás közbeni GSR változásra felkészülve vezette be az OGSA a HandleMap szolgáltatást. Az ilyen szolgáltatást nyújtó hostok képesek a klienseiktől bemenő adatként kapott GSH-ra a GSH-val reprezentált szolgáltatás aktuális GSR-jével válaszolni. Funkcionalitásuk megoldja tehát a futás közbeni viselkedésváltás miatt használhatatlanná váló GSR-ek problémáját, hiszen amikor a kliens észreveszi, hogy szolgáltatásának funkcionalitása megváltozott, (ezt például onnan láthatja, hogy az eddig használt stub nem funkcionál többé, de akár értesítést is kaphat a változásról az SDE infrastruktúrán keresztül), akkor egy megfelelő HandleMap szolgáltatást használva megszerezheti az új funkcionalitást leíró GSR-t. Az új GSR alapján ezután létrehozhat egy új stubot, majd rajta keresztül folytathatja a szolgáltatás használatát.

Egy probléma van csak ezzel az elgondolással: honnan tudhatja meg a kliens, hogy melyik HandleMap szolgáltató tudja az ő szolgáltatáspéldányának az új GSR-jét? Erre a kérdésre az OGSA megoldásként azt ajánlja, hogy a szolgáltatás gyárak a szolgáltatáspéldányok létrehozásakor írják bele azok GSH-jába azon HandleMap szolgáltató GSH-ját, amelynél a szolgáltatás GSR váltás esetén új GSR-jét regisztrálni fogja. A kliens a szolgáltatáspéldányára állandóan érvényes GSH-ból kinyerheti a HandleMap szolgáltató címét, az információs rendszert használva megtalálhatja azt, majd tőle megkaphatja az új GSR-t.

4.4 Az OGSi és a WSRF

A grid szolgáltatások technikai leírásából látható volt, hogy az OGSA nem tett mást, csak a webszolgáltatások technológia alapján definiált olyan szituációkat és szerepköröket, melyekben a jövő grid rendszereit alkotó eszközök nap, mint nap részt vesznek majd. Ezzel a megoldással az OGSA megtartotta a webszolgáltatások által is alkalmazott nyílt rendszer elvet, vagyis hogy olyan protokollok és technológiák használatával kell a rendszerek közötti együttműködést megvalósítani, melyet bármilyen gyártó bármilyen számítógépe, tetszőleges platformot használva végre tud hajtani.

Annak ellenére, hogy az OGSA betöltötte a webszolgáltatás technológiából hiányzó, viszont a grid számára nélkülözhetetlen részeket, valójában közel sem tekinthető grid specifikációnak. Ha két fejlesztő egymástól függetlenül elkezdene OGSA gridet készíteni, akkor ezek a rendszerek szinte biztosan nem lennének egymással kompatibilisek, az egyik hálózat szolgáltatóit a másik hálózat kliensei képtelenek lennének megérteni és használni. Ahhoz, hogy létrejöjjön egy globális rendszerek létrehozására képes technológia, az OGSA elvekkel összhangban lévő, de annál sokkal részletesebb specifikációt kellett készíteni. Ennek a specifikációnak a neve Open Grid Services Infrastructure (OGSI).

Az OGSI célja az OGSA által bevezetett mintákra a webszolgáltatás technológiával összeegyeztethető grid rendszer specifikációt adni. Amíg az egymástól függetlenül fejlesztett OGSA gridek nem, addig az OGSI specifikáció szerint létrehozott gridek képesek lehetnek az együttműködésre. Mivel webszolgáltatásokból (és így OGSA alapú grid szolgáltatásokból is) az XML alapú interfészekon és az interfészekon keresztül küldött szintén XML alapú üzenetekon kívül semmi más nem látszik, ezért az OGSI specifikáció XML alapú dokumentumok segítségével standardizálja a grid szolgáltatásokat. A 11. ábra egy OGSI-specifikus GSH-t mutat. Ha egy szolgáltatás fejlesztő ilyen GSH-val látja el a szolgáltatását, akkor azt minden szabványos OGSI kliens megtalálhatja.

```
<xsd:element name="handle" type="ogsi:HandleType"/>
<xsd:simpleType name="HandleType">
  <xsd:restriction base="xsd:anyURI"/>
</xsd:simpleType>
```

11. ábra OGSi által specifikált GSH

Az elmúlt néhány évben a webszolgáltatás közösség felismerte, hogy a dinamikusan változó, állapotörző erőforrásokat is tartalmazó hálózatokat kezelni képes OGSi specifikáció hasznos lehet a webszolgáltatás világ számára is. Az OGSi specifikációt megvizsgálva, azt továbbfejlesztve néhány nagy IT cég (Globus Alliance, IBM, HP) kidolgozta a Web Services Resource Framework (WSRF) szabványt. A szabvány célja a webszolgáltatás közösség számára egy olyan általános elveket követő specifikációt adni, amellyel állapotörző és dinamikus erőforrásokat megjeleníteni képes webszolgáltatások készíthetők. A standard segítségével a webszolgáltatás fejlesztők gyorsabban és korábbi munkáikat újra felhasználva lehetnek képesek hibátűrőbb és sokrétűbb elosztott alkalmazások kifejlesztésére. A részösszegzéseként elmondható, hogy az OGSA a nyílt szabványokat követő webszolgáltatás technológián keresztül kíván szolgáltatás alapú rendszereket létrehozni. A HTTP kommunikációs protokoll segítségével továbbított, standardizált formátumú XML üzenetek biztosíthatják, hogy az Internetre, vagy lokális hálózatba kapcsolt intelligens programok képesek legyenek egymással emberi segítség nélkül együttműködni, egymásnak különféle szolgáltatásokat nyújtani. A webszolgáltatás technológia UDDI szolgáltatását használva ezek az eszközök képesek lehetnek igény szerint partnereket találni, így rétegezett, dinamikusan felépülő szolgáltatás-hierarchiákat alkotni.

Mivel a webszolgáltatás technológia nem törődik sem a hálózat komponenseinek dinamizmusával, sem az állapotörző funkciók szolgáltatásként való megjelenítésével, ezért az OGSA – és az arra épülő OGSi szabvány – megtette ezen kiegészítéseket. Eközben a webszolgáltatás közösség felismerte, hogy az OGSi szabvány által bevezetett eszközök a webszolgáltatások fejlesztőinek is sok esetben hasznos mintákat adhatnak, ezért kidolgozta az OGSi továbbfejlesztésének tekinthető WSRF standardot. A munka még jelenleg is folyik, nem csak a technológia kidolgozásának, de más, a webszolgáltatás piacot meghatározó egyéb cégek meggyőzésének irányába is.

5. ÖSSZEFOGLALÁS

A szolgáltatásorientált rendszerek kialakulásához leginkább a gyorsan változó üzleti világ jelenléte adott egyfajta kezdőlökést. A különböző programozási módszertanok fejlődése során világossá vált, hogy a szolgáltatás központú megoldások felé kell venni az irányt. Ezen üzleti igények kielégítését valósítja meg a szolgáltatásorientált architektúra, amely egy elméleti ajánlás, melynek implementációi eltérőek lehetnek. Az architektúra központjában a szolgáltatás áll, amely valamely üzleti folyamat leírására szolgál. Ezen szolgáltatások által nyújtott üzleti funkciókhoz férnek hozzá a kliensek. A szolgáltatás funkcióit az interfészen keresztül éri el a kliensek, elrejtve az implementációt előlük. Fontos összetevője az architektúrának, hogy ezen szolgáltatások hogyan fedezhetőek fel, illetve ezen szolgáltatásorientált rendszereket hogyan kapcsolhatjuk össze nyelv- és platformfüggetlen módon. Ezen kérdésekre ad választ a szolgáltatás tárház, illetve a szolgáltatás busz fogalma.

Az első, igazán széles körben elterjedt technológia a webszolgáltatások technológiája, amelyek nagy, akár üzleti szintű komponensek felhasználását teszi lehetővé akár az interneten keresztül, s így függetlenül attól, hogy az egyes szoftverkomponensek hol találhatók. Mindezt az XML-alapú szabványainak köszönheti, melyek szöveges megjelenésük miatt platform független módon ugyanazt jelentik bármilyen architektúrán. A webszolgáltatás három fő szabványon alapszik, az UDDI a szolgáltatások felderítését teszi lehetővé, a WSDL a webszolgáltatások azon meta adatait tartalmazza, amelyek segítségével egy kliens képes adott szolgáltatás igénybevételére, a SOAP pedig nyelv- és platformfüggetlen üzenetváltást tesz lehetővé. A webszolgáltatások kiegészítéseként beszélünk még ESB buszokról, amelyek a szolgáltatás busz fogalmát hivatott implementálni, illetve a BPEL4WS leírónyelvről, amely üzleti folyamatok XML alapú leírására szolgál.

A webszolgáltatások, ESB és BPEL4WS technológiák együttese hivatott a szolgáltatásorientált architektúrák megvalósítására. Ám ezen technológiák csak állapot nélküli szolgáltatásokat képesek implementálni, a web és az általa biztosított állapot nélküli HTTP protokollnak köszönhetően. Ezen probléma kiküszöbölését a grid technológia hivatott megoldani. Ezen technológia durván két irányban fejlődött, az egyik önálló grid rendszerek

megvalósítását tűzte ki célul, míg a másik a webszolgáltatások kiegészítéseként alakult ki. A webszolgáltatások kiegészítéseként definiált OGSA elvek hivatottak a webszolgáltatások hiányos részeinek betöltésére. Ezen elvek biztosítják azt, hogy állapotörző szolgáltatásokat tudjunk definiálni webszolgáltatások segítségével. Annak ellenére, hogy az OGSA betöltötte a webszolgáltatás technológiából hiányzó, viszont a grid számára nélkülözhetetlen részeket, még nem tette lehetővé, hogy egymástól függetlenül fejlesztett OGSA grideket hogyan kapcsolhatnánk össze. Ezen kérdéskör megoldásaként fejlesztették ki az OGSF és WSRF szabványokat, amelyek fejlesztése mind a mai napig folyamatban van.

A diplomamunkámban igyekeztem az informatika területén egyre jobban elterjedő szolgáltatásorientált architektúrákról egy áttekintést nyújtani, azok legelterjedtebb implementációjára, a webszolgáltatások és azok kiegészítéseként megjelenő grid rendszerekre fókuszálva. Remélem a munkám elnyerte tetszését, és segítségére vált a szolgáltatásorientált szemléletmód alapvető megismerésében.

IRODALOMJEGYZÉK

- [1] Dave Chappell: Enterprise Service Bus, O'Reilly, 2004
ISBN 059 600 675 6
- [2] Dirk Krafzig, Karl Banke, Dirk Slama, Enterprise SOA: Service-Oriented Architecture Best Practices, Prentice Hall Prentice Hall, 2004
ISBN 013 146 575 9
- [3] Gottdank Tibor: Webszolgáltatások, ComputerBooks, 2005.
ISBN 963 618 305 8
- [4] Juhász Zoltán, Kuntner Krisztián, Pásztory Ákos, Póta Szabolcs, Sipos Gergely:
Szolgáltatásorientált Grid architektúrák és technológiák,
http://pds.irt.vein.hu/files/publications/SOA_Tanulmany.pdf
- [5] IBM, Implementing an ESB using IBM WebSphere,
<http://www.redbooks.ibm.com/redbooks/pdfs/sg247335.pdf>
- [6] OASIS, UDDI Specifications,
<http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm>
- [7] OASIS, Using BPEL4WS in a UDDI registry,
<http://www.oasis-open.org/committees/uddi-spec/doc/tn/uddi-spec-tc-tn-bpel-20040725.htm>
- [8] OASIS, Using WSDL in a UDDI Registry,
<http://www.oasis-open.org/committees/uddi-spec/doc/tn/uddi-spec-tc-tn-wsdl-v2.htm>
- [9] OASIS, Web Services Business Process Execution Language,
<http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html>
- [10] W3C, SOAP specifications, <http://www.w3.org/TR/soap/>
- [11] W3C, Web Services Description Language, <http://www.w3.org/TR/wsdl>