Rendszerfejlesztés „polcról levehető” komponensekből

Mottó

- „Az étel annyira gusztusosan van elrendezve a tányéron, hogy nyilvánvaló, hogy valaki végigtapogatta minden részét.”

  (Julia Child, az újszerű konyhaművészetről)

„Polcról levehető” komponensek (COTS)

- Az összetettebb, nagy rendszerek fejlesztésénél egyre inkább előtérbe kerülnek a polcról levehető, kész komponensek.
- Ennek fő okai:
  - **Gazdaságosság** Kevesebb időt, munkaerőt, ezáltal pénzt emészt fel egy készre gyártott komponens megvásárlása.
  - **Szakértelem hiánya** Az egyes tudományterületeken olyan mértékű a specializáció, hogy nehézkes és költséges a megfelelő szakemberek alkalmazása.

Mit is takar ez a fogalom?

- Óriási funkcionalitással felvértezett, gyorsan hozzáférhető, készre csomagolt szoftvert jelent a COTS.
- A belsejéről semmit nem tudunk, kizárólag a funkciói és a kommunikációt bonyolító interfészei publikusak.
- Azzal azonban tisztában kell lennünk, hogy az eredeti fejlesztők nagy valószínűséggel nem a mi konkrét problémánkhoz fejlesztették ki.

Következmények

- A polcról levehető komponensek nem csupán a szoftvertervezés folyamatát változtatják meg, hanem architekturális (ezáltal minőségi) megszorításokat is megtestesítenek.
- Az „építésznek” (az architektúra tervezője) meg kell győződnie, hogy ezek a megszorítások elfogadhatók-e, illetve nem vezetnek-e kompatibilitási problémákhoz.
- A mai előadásban azt is megvizsgáljuk, hogy kész komponensek használata milyen hatással van a tervezett rendszer architektúrájára.

Egy konkrét példa

- A Quack.com egy olyan cég, amely kizárólag arra specializálódott, hogy tartalmi és kereskedelmi szolgáltatásokat nyújtson telefonon keresztül.
- A példa kiválóan illusztrálja a polcról levehető komponensek felhasználhatóságát és korlátait is.
- A fenti területen sok más cég is tényked ett, némelyikük biztosabb anyagi háttérrel, így egyetlen esélyük az volt, ha ők dobják elsőként piacra a terméküket.

Egy konkrét példa (folyt.)

- Felkutatták az összes elérhető (releváns) polcról levehető komponenst, így rengeteg időt spóroltak meg. Nem egy esetben több cég is kínált egy adott funkcióhoz készült komponenst.
- A stratégia bejött, hisz elsőként jelentek meg a termékükkel, és nemsokára az AOL is megvásárolta tőlük a rendszert. A nagyobb terheléshez (ügyfelek számának robbanásszerű növekedése) újraírták a komponenseket menet közben, ahol szükséges volt.
- Az architektúra rugalmassága lehetővé tette ezt, de kérdéses volt, hogy a komponensek hogyan reagálnak.

Egy konkrét példa (folyt.)

- Az egyik fejlesztő röviden ennyit mondott:
  - Ha egy függvény nem lényeges, COTS-ra bíztuk.
  - Ha van valamilyen szabvány, amelyre a rendszernek illeszkednie kell, szintén COTS-ra bíztuk (mondván úgyis lesz már kész megoldás a piacon).
  - Ha viszont bizonytalanok voltunk a komponens minőségét vagy megbízhatóságát illetően, akkor házon belül készítettük el.

Hátulütők

- Sajnos, a COTS használata nem teszi lehetővé a fejlesztők számára, hogy teljes ellenőrzést gyakoroljanak a rendszer minőségi tulajdonságait illetően.
- A Bertrand Meyer által bevezetett „contract” fogalma ezt hivatott kiküszöbölni.

A komponensek hatása az architektúrára

- Tegyük fel, hogy egy vegyipari gyár ellenőrzését végző szoftvert kell elkészítenünk. A gyárban folyó kémiai folyamatok aktuális állapotáról érzékelőkön keresztül jutnak el az adatok az operátor (speciális) képernyőjére. A szoftver egy nem elhanyagolható részének az a feladata, hogy a képernyőre rajzolja a műveletek modelljét. Egy szoftvercég kínál olyan komponenst, amely lehetővé teszi az ilyen spéci kijelzőkre történő rajzolást. Megvásároljuk a terméket, de csak ezután derül ki, hogy a modul kizárólag VisualBasic nyelven áll rendelkezésre.

Milyen hatással van eme döntésünk az architektúrára?

- Vagy az egész rendszert újra kell kódolnunk VisualBasic nyelven, mivel a megvásárolt komponens által produkált grafika is ezen a nyelven utasítja a kijelzőket, vagy a teljes operátori részt le kell választani a rendszer egyéb részeitől.
- Ez egy alapvető strukturális döntés, amelyet a rendszer egy szeletéhez kiválasztott apró komponens eredményezett.

Áthághatóság

- Még a legegyszerűbb komponens is hordoz magában valamiféle architektúrát, amelyet igencsak nehézkes áthágni.
- Például egy HTTP kiszolgáló feltételezi a kliens-szerver architektúrát, jól definiált interfészekkel és a háttérben meghúzódó funkcionalitást integráló mechanizmusokkal. Ha eltérünk attól, amit a HTTP kiszolgáló vár tőlünk, különösen nehéz integrációs folyamat elé nézünk...

Következmények

- A komponensek és a közöttük zajló interakció megértése alapvető fontosságú, **mielőtt** az architektúra véglegesítésére sor kerülne.

Architekturális eltérés I.

- Nem minden komponens működik együtt, még akkor sem, ha kereskedelmi termékről van szó, és a készítője azt állítja, hogy kompatibilis. A komponensek általában „majdnem kompatibilisek” egymással, ami gyakorlatilag azt jelenti, hogy nem kompatibilisek egymással.
- Ezek egészen apró hibákból eredhetnek, például párhuzamos rendszerek esetében az időzítéssel vagy az egymásutánisággal kapcsolatos rossz feltevésekből.

Architekturális eltérés II.

- Garlan, Allen és Ockerbloom architekturális eltérésnek nevezte a fenti problémát, amelyet tulajdonképpen az egymástól függetlenül fejlesztett komponensekben testet öltő **feltételezések** közti különbségnek tekinthetünk. Ez a jelenség általában csak a rendszerintegrációs folyamat során derül ki, esetenként még később.
- Mik ezek a feltételezések?

Architekturális eltérés III.

- A feltételezések kétféle formában jelenhetnek meg:
  - A **provides (biztosít)** feltételezések az adott komponensnek a felhasználók és az ügyfelek részére biztosított szolgáltatáskínálatának leírását tartalmazza.
  - A **requires (elvár)** feltételezések azokat a szolgáltatásokat részletezik, amelyek elengedhetetlenek az adott komponens helyes működéséhez.

2.23. ábra

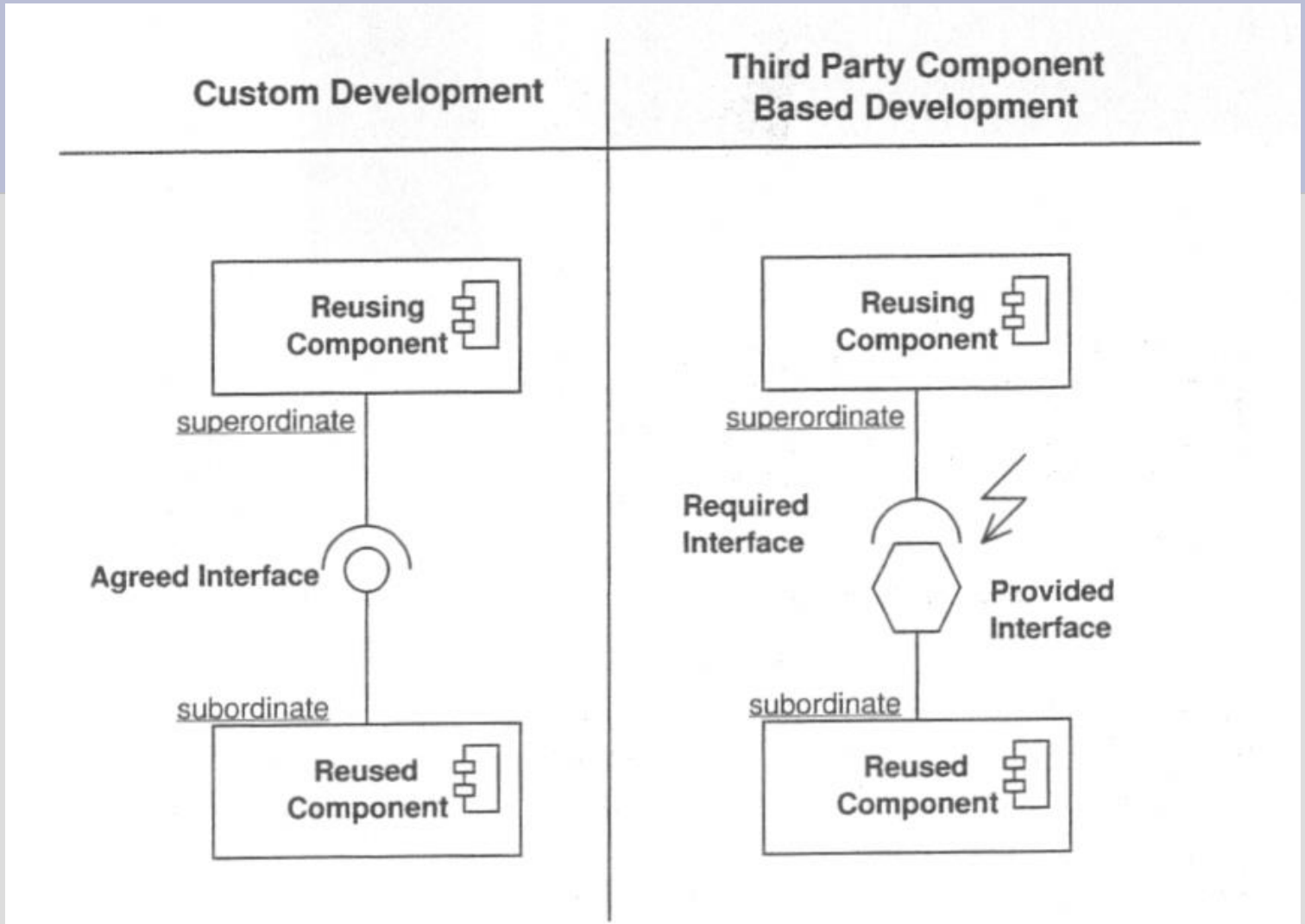


Fig. 2.23. Custom-designed component integration vs. third-party component integration

Mit tehetünk ilyen eltérések esetén?

- „A tegnapi hiba ma már beépített tulajdonsága az adott rendszernek.” Ez sok esetben életképes megoldás.
- Kerüljük el a komponensek gondos **specifikációjával és vizsgálatával**. Ez sajnos nem mindig lehetséges.
- Az elkerülhetetlen eseteket **vegyük észre** a komponensek gondos **válogatásával**.
- A felfedezett eseteket **javítsuk** a komponensek **adaptálásával**.

Interfész eltérés javításához használt technikák

- **Csomagoló** A csomagoló fogalma egyfajta befoglalásra utal, amelynek kertében egy megadott komponenst valamiféle alternatív absztrakció mögé rejtünk.
- **Híd** A híd az egyik komponens elvár feltevéseit transzformálja a másik biztosít feltevéseihez igazítva. A különbség a csomagolóhoz képest, hogy a híd független egy adott komponenstől.
- **Tárgyaló** A híd módszerét futásidőben alkalmazza.

Példa a híd és a tárgyaló közötti különbségre

- Tegyük fel, hogy egy komponens D^0 formában generálja az adatokat, miközben egy másik D^2 formában képes az adatokat elfogyasztani.
- Tegyük fel, hogy nincs közvetlen D^0 -> D^2 hidunk, de vannak D^0 -> D^1 és D^1 -> D^2 hídjaink, amelyek láncolhatók.
- A tárgyaló a fenti hidakból összeállíthatja a D^0 -> D^2 transzformátort.

2.24. ábra

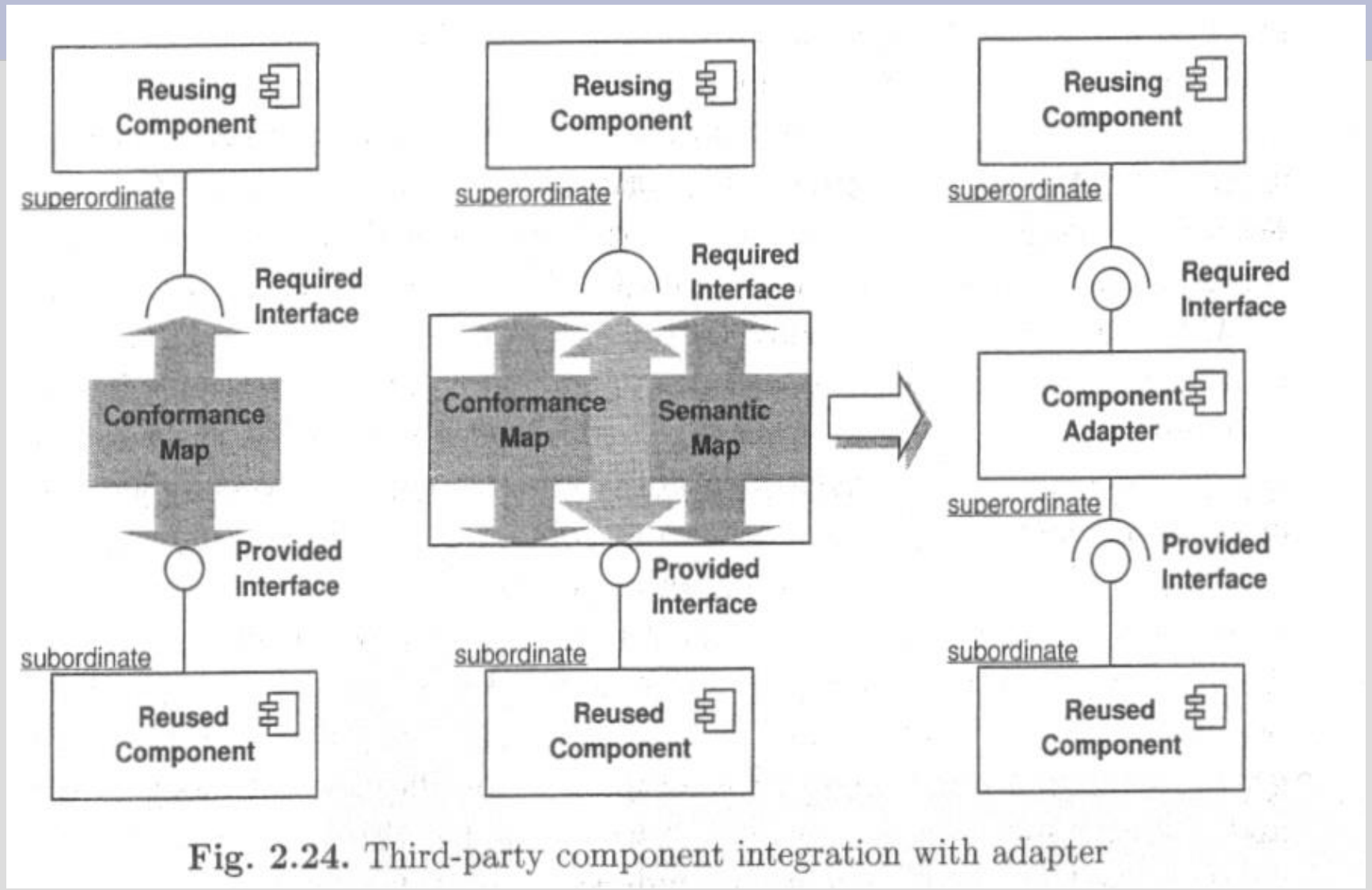


Fig. 2.24. Third-party component integration with adapter

Technikák az interfészek közötti eltérések kiderítésére

- Megfelelési térkép elkészítése
  - Leírja egy COTS komponens kívülről látható tulajdonságait egy olyan leképezés segítségével, amely megadja azt, hogy a COTS komponens kifejlesztésekor használt jelölés és az általunk használt között mi a kapcsolat.
- Szemantikus térkép elkészítése
  - Ha egy megfelelési térkép összeállítása után pozitív a döntésünk egy komponens újra felhasználása tekintetében, akkor a következő lépés a szemantikus térkép létrehozása.
  - A szemantikus térkép a két komponens (a megtervezett és a felhasználni szánt) specifikációjának hasonlóságaira és különbözőségeire koncentrál és megpróbálja modellezni a leképezést a két eltérő interfész között.

A komponens-alapú tervezés mint keresés I.

- Az eddigiek alapján megállapíthatjuk, hogy a komponens-alapú tervezés nem más, mint egy keresési folyamat, melyben a rendszer igényeinek leginkább megfelelő elemeket kutatjuk fel.
- Az építésznek minden egyes komponens esetén döntést kell hoznia, hogy a megadott elem integrálható-e a rendszerbe, valamint a rendszer követelményeinek megfelelően képes az architektúrába illeszkedni.

A komponens-alapú tervezés mint keresés II.

- Mindegyik elem hozzáadódik a felfedezendő (bejárandó) útvonalhoz. A felfedezés elsősorban a nem adaptálható architekturális eltérések felderítésére helyezi a hangsúlyt, de számításba veszi a javítás és a figyelmen kívül hagyás hatásait is.
- Többféle útvonal egyidejű bejárása túlságosan költséges, ezért a gyakorlat mindig egy elsődleges és néhány másodlagos útvonal bejárására korlátozódik.

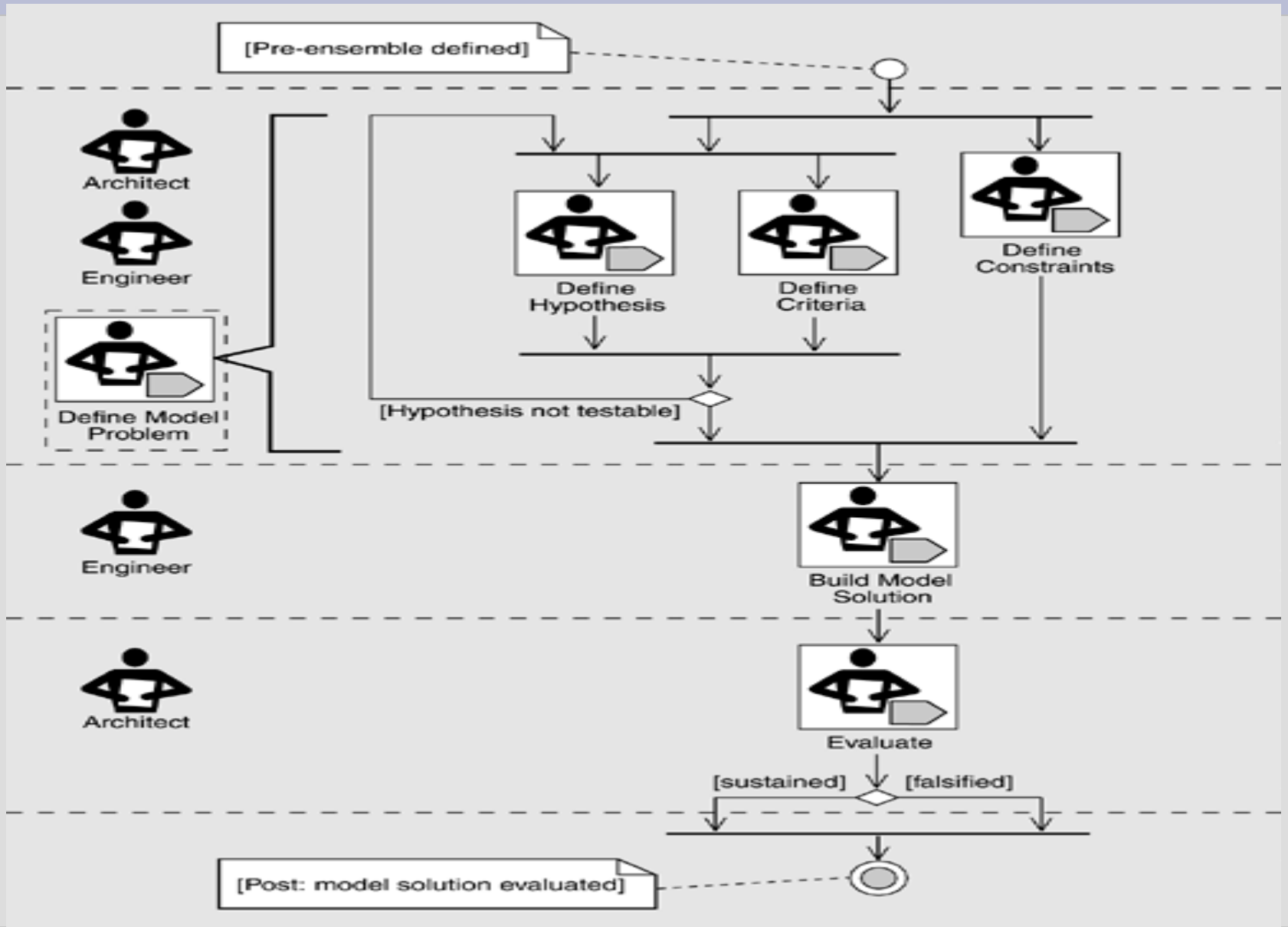
A komponens-alapú tervezés mint keresés III.

- A komponensek értékeléséhez és a minőségi garantálásához ún. modell problémát alkalmazunk. A modell probléma tulajdonképpen az implementációra vonatkozó megszorítások leírása (pl. A Web alapú interfészt mind a Netscape mind pedig az Explorer használhassa).
- **A tervezési mintának eleget tevő prototípust nevezzük modell megoldásnak**.

A modell probléma munkamenete

- 6 lépésből áll:
  - Az építész és a mérnökök azonosítják a tervezési felvetést, amiből kiindulunk.
  - Az építész és a mérnökök megadják a kiindulási kiértékelési feltételeket. Leírja, milyen a jó ill. rossz megoldás.
  - Az építész és a mérnökök megadják az implementációra vonatkozó megszorításokat.
  - A mérnökök készítenek egy probléma megoldást.
  - A mérnökök azonosítják a végső kiértékelési feltételeket (kezdeti + a közben felfedezett)
  - Az építész kiértékeli a modell megoldást, hogy a végső feltételeknek maradéktalanul eleget tesz-e.

A modell probléma munkamenete



Egy példa: ASEILM

- A példánk témája egy Web alapú információs rendszer, amelyet egy Szoftverfejlesztő Intézet (SEI) fejlesztett ki abból a célból, hogy az intézet és a partnerei közötti adminisztratív kapcsolatokat automatizálják. Ebből a célból fejlesztették ki az ASEILM (Automated SEI Lincensee Management) rendszert.
- Az ASEILM rendszer feladatai
  - Támogassa a SEI termékeinek forgalmazását
  - Gyüjtsön össze statisztikai adatokat
  - Készítsen grafikonokat
    - a bevételről,
    - a honlapot felkeresők számáról és
    - a SEI termékekkel kapcsolatos egyéb információkról
  - Kövesse nyomon a SEI kurzusainak látogatottságát és a jogdíjakat.

ASEILM funkciói

- Többféle felhasználói kört változó jogosítványokkal szolgál ki.
- Funkciók:
  - A kurzusok szervezői számára szolgáltatja
    - A résztvevők listáit
    - Fenntartja a kontaktust a résztvevőkkel
    - Letölti a kurzusra vonatkozó anyagokat.
  - A vezető kiértékelők számára
    - Segíti a kiértékelést az input adatokkal és a kiértékelési űrlapok letöltésével
  - A SEI adminisztrátorok számára
    - Karbantartja a kurzusszervezésre jogosultak és a vezető kiértékelők listáját
    - Támogatja a rendszer karbantartását minden elvárható eszközzel.

1. táblázat

A minőségi követelmények listája

TABLE 18-1 Quality Attribute Requirements

Quality Attribute	Requirement
Functionality	Provide Web-based access to a geographically dispersed customer base
Performance	Provide adequate performance to users running overseas on low-bandwidth connections (i.e., download times in tens of minutes, not hours)
Compatibility	Support older versions of Web browsers including Netscape 3.0 and Internet Explorer 3.0
Security	Support multiple classes of users and provide an identification and authorization scheme to allow users to identify themselves
Security	Provide commercial-grade secure transfer of data over the Internet

Miva Empressa

- A fejlesztő csapatnak bizonyos előzetes ismeretei voltak a Microsoft Internetes Információs Serverének (IIS) kiterjesztéséről.
- A Miva Script alkalmazások a Miva Empressa vezérlése alatt futnak.
- A következő ábra a Miva Empressa egy összeállítását mutatja be, amelyet az ASEILM projektben használtak.
- Mint az ábrán látható az összeállítás a Miva Empressa alkalmazói szerver mellett további polcról levehető komponenseket is tartalmaz:
  - Microsoft Access - adatbázis kezelésre
  - Visual Mining's Charts Works – grafikonok készítésére
  - Microsoft IIS – HTTP szerverként
  - Windows NT 4.0 – operációs rendszer a szerveren
- A kliensek bármilyen platformon bármilyen keresőket használhatnak. A kezdeti összeállítás:
  - Netscape 3.0 kereső
  - Windows 98 operációs rendszert tartalmaz.

2. ábra

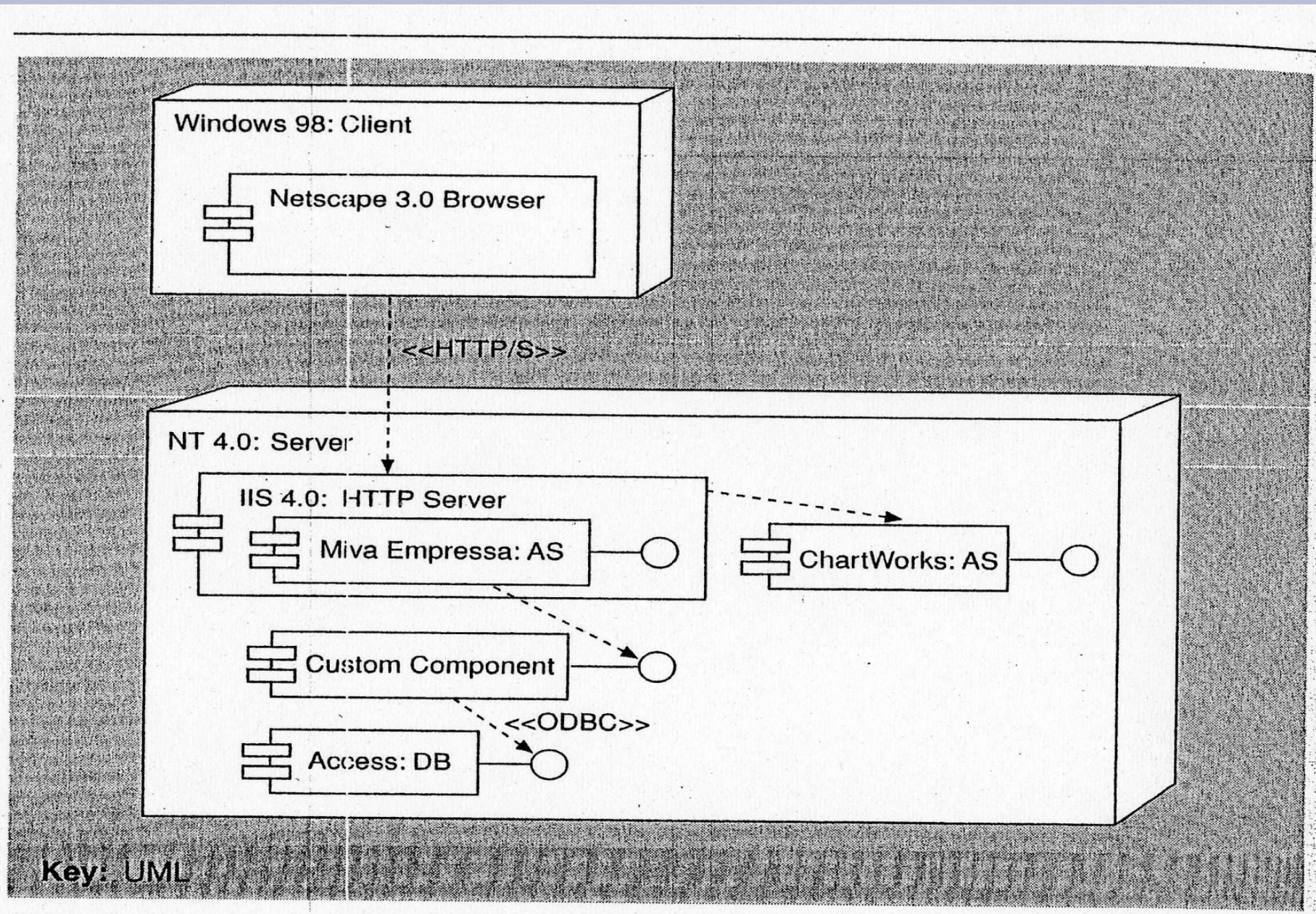


FIGURE 18.2 Miva Empressa ensemble

A rendszer modelljének elkészítése

- **1. lépés:** Identify a Design Question
  - A rendszerrel (összeállítás, komponens) kapcsolatos hipotéziseinket használati esetek formájában fogalmazzuk meg.
    - *1. hipotézis.* A komponens Web-en keresztüli elérést biztosít az adatok karbantartására az Access Adatbázisban. Megjeleníti azokat a felhasználó által kért grafikonok formájában.
    - *2. hipotézis.* A kommunikáció a Web browser és a HTTP szerver között titkosított módon történik a HTTPS segítségével.

- **2. lépés.** Define Starting Evaluation Critéria. Az elkészült modellt abból a szempontból kell értékelni, hogy az előzőekben megfogalmazott hipotéziseket betartja-e.
  - *1. kritérium.* A modell valóban az Access Adatbázisban tárolt adatokat jeleníti meg a kereső használata során.
  - *2. kritérium*. Valóban biztos (secure) adatokat kap a Web broser a HTTPS kapcsolaton keresztül.

- **3. lépés.** Identify Implementation Constraints. Ezek a megszorítások a tervezés szempontjából merev megszorításokat tartalmazzák.
  - Esetünkben ilyen megszorítások nincsenek.
- **4. lépés.** Produce Model Solution. Miután a megoldandó problémát teljesen definiáltuk a fejlesztő csapat elkezdte a modellt létrehozni. Eközben két újabb megoldandó probléma merült fel. A grafikonok megjelenítéséhez az adatokat el kell juttatni a ChartWorks komponensnek. Hogyan történjen ez meg biztonságos módon. Azaz
  - Hogyan kapcsoljuk össze az adatokat és a gráfokat
  - Hogyan tartsuk fenn a biztonságos kapcsolatot a két komponens között
    - Ez utóbbi problémát a fejlesztő csapat egy perl proxy szerver közbeiktatásával oldotta meg, ahogy az a következő ábrán látható.

3. ábra

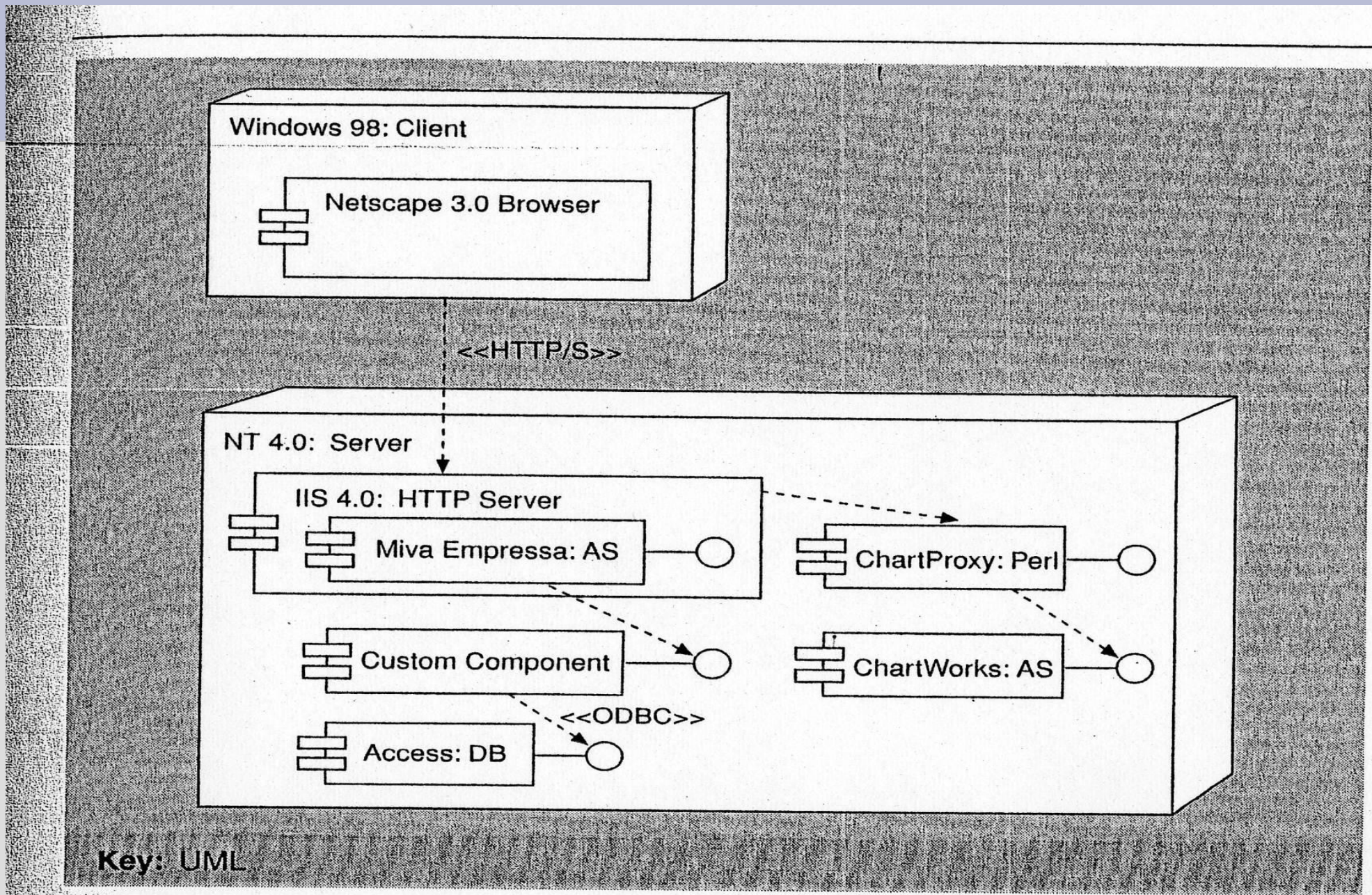


FIGURE 18.3 Introduction of Proxy server

- **5. lépés.** Identify Ending Evaluation Crtéria
  - *3. kritérium.* A prezentációs logika karbantartását függetlenül el kell tudni végezni a háttér üzleti logika és a adatbázis logika karbantartása nélkül jól definiált interfészen keresztül.
  - *4. kritérium.* Az adatbázist biztos helyen, tűzfal mögött kell tárolni.

- **6. lépés.** Evaluate the Model Solution. Miután a modell megoldást implementálták és az újabb kiértékelési kritériumokat meghatározták az építész elvégezheti a megoldás ellenőrzését a kritériumok teljesülése szempontjából.

- Az építész vizsgálatának eredménye a Miva Empressa összeállítást jól ismerők számára nem meglepő: ez az összeállítás **alkalmatlan** a kitűzött feladatot megoldó komponens megalkotására, mivel a két újabb kritérium teljesítésére alkalmatlannak bizonyult.

- A jó megoldás egy JAVA Servlet összeállítással tudták létrehozni a fejlesztők. Ezt az összeállítást mutatja be a következő ábra.

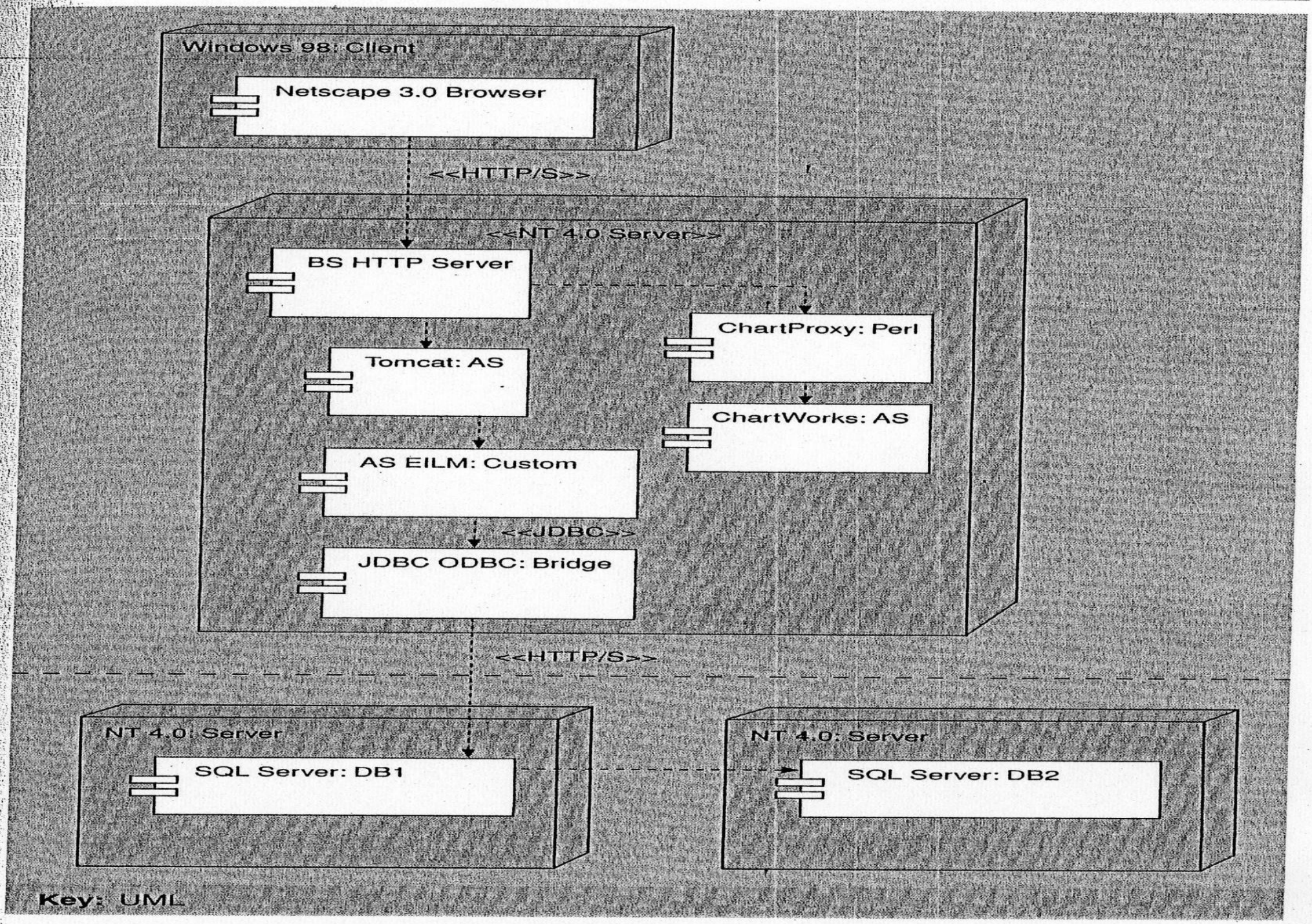


FIGURE 18.4 JavaServer Pages ensemble

- A sikerhez kellett egy új réteg, a custom komponens réteg bevetése, amelyet a következő ábrán láthatunk.

5. ábra

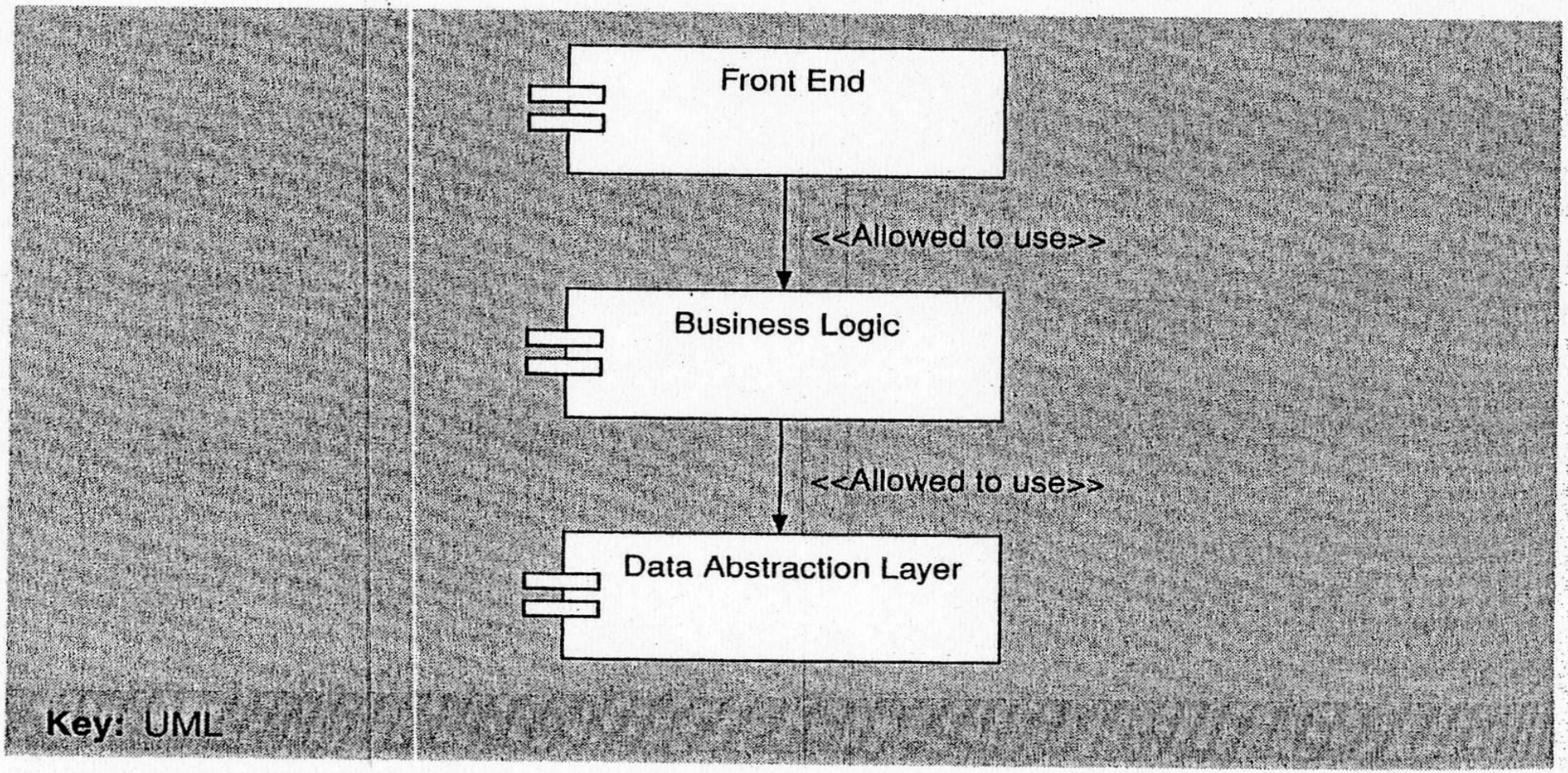


FIGURE 18.5 Layers of custom component

Összefoglalás

- A mai előadás tanulságaként megállapíthatjuk, hogy kész komponensekből összerakni egy működő rendszert más jellegű gyakorlást is igényel mint amikor saját magunk készítjük el a komponenseinket.
- A kész komponensek interfészei és a közöttük lévő kapcsolatok erősen befolyásolják a készítendő rendszer architektúráját.
- Ha nincs kellő gyakorlatunk az ilyen rendszerek létrehozásában előfordulhat, hogy a legkörültekintőbb eljárás esetén is csak viszonylag későn derül ki, hogy a készülő rendszer bizonyos követelményeknek mégsem tud megfelelni.