# 7. Приложни комуникации

Васил Георгиев

 ci.fmi.uni-sofia.bg

 v.georgiev@fmi.uni-sofia.bg

# Съдържание

- Клиент-сървер: процедурен и обектен модел
- P2P модели
- Системи с обмен на съобщения

# Комуникации в разпределените системи

- осъществяват обмена между паралелните процеси свързвайки ги в единна система за разпределена обработка
- базират се на обмен на съобщения (message passing; PC не поддържат по-удобния за програмиране, но нескалируем модел на общата памет)
- ползват се наличните мрежови примитиви за message passing
- организацията на сложни високопаралелни изчисления (стотици, понякога хиляди паралелни процеси) е възможна само с приложението на message passing модели от високо ниво:
  - мидълуер за клиент-сървер модели:
    - RPC (remote procedure call) – специализаран модел на съобщения, подходящ за приложения от типа клиент-сървер
    - RMI (remote method invocation) – развитие на RPC за работа с разпределени обекти
    - MOM (message-oriented middleware) – управление на транзакциите, приложимо при асинхронни (не клиент-сървер) приложения, работещи на пренципа на e-mail: message-oriented комуникации (в IBM и MS Windows: message queuing)
    - поточни данни (streams) – предимно за мултимедийни приложения, данните се интерпретират като поток от аудио и видео информация с ограничения във времето за предаване – “sub real-time”
  - p2p модели

# TCP клиент-сървер приложения

- стандартен (разширен) и транзакционен (T/TCP) вариант - 7.4
- използва се последовотелно номериране на пакетите за подреждане на съставните съобщения
- състояния на TCP-процес (клиент или сървер)
  - LISTEN TCP-сървер: изчаква заявка за откриване на съединение (connection) от отдалечен TCP-порт
  - SYN-SENT TCP-клиент: изчаква отговор TCP-пакет с установени SYN и ACK флагове
  - SYN-RECEIVED TCP-сървер: изчаква потвърждение за издаденото потвърждение за откриване на съединение
  - ESTABLISHED TCP-сървер и клиент: преминава в състояние на приемане и придаване на на пакети от и към отдалечен порт след установено съединение
  - TIME-WAIT TCP-сървер и клиент: изчакване на времеинтервал (до 4 мин.) преди закриване на съединение
  - FIN-WAIT-1, FIN-WAIT-2, CLOSE-WAIT, CLOSING, LAST-ACK, CLOSED

# UDP-базирани клиент-сървер приложения

- при мрежи с надеждни комуникации
- за приложения, толерираващи загуби
- по-висока скорост и намален signaling
- ниска преносимост

# Клиент-сървер със сесийни и представителни протоколи

- сесийните протоколи са специализирани да поддържат обмена в режим на диалог – синхронизация на предаващия и приемащия възел
  - с възможност за промяна на ролите
  - възможност за контролни точки при дълги предавания (възстановяването след грешка с връщане в последната контролна точка)
- представителните протоколи са специализирани в представяне на съдържанието на съобщенията като структурирани записи

# Клиент-сървер с приложни протоколи

- Приложни протоколи (функциите на протокола се реализират от клиент с потребителски интерфейс) – стандартни мрежови приложения за
  - електронна поща,
  - файлов обмен,
  - терминална емулация,
  - достъп до документно съдържание (хипертекст и др.)
  - N.B.:
    - тези протоколи могат да се използват и с различни от първоначално зададените функции – напр. HTTP се ползва в Java RMI за обръщение към отдалечени обектни методи през защитни стени
    - различни клиентски приложения реализират в различна пълнота функциите на протокола

# Middleware услуги за клиент-сървер

- Middleware (междинни протоколи между транспортното и приложното ниво) практически е алтернатива на сесията и представянето (без приложението)
  - услуги от високо ниво – напр.:
    - проверка за автентичност (authentication)
    - [отдалечен] допуск (authorization) до разпределен ресурс (след успешна проверка за автентичност) – напр. RPC, RMI
    - разпределено резервиране на ресурси (locking)  при множествен достъп
    - обвързващи (commit) протоколи за едновременно изпълнение на дадена операция от група процеси – атомарност (atomicity) – напр. за отказоустойчивост
    - синхронизиране на поточни данни
    - обмен в реално време (за мултимедийни приложения)
- N.B.: middleware обикновено са настройваеми към изискванията на типа приложение (запазвайки единен интерфейс, но ползвайки услугите на различни транспортни протоколи) и затова те са необходима надстройка над мрежовите услуги

# Преходност и синхронност в комуникационните модели

- устойчивите (persistent) комуникации съхраняват съобщенията в транзит, дори и когато изпращащия и получаващия процес не са активни – т.е. комуникационната система ги буферира
- при преходните (transient) комуникации съобщението се съхранява и в крайна сметка предава само ако приемащия и предаващия процес са активни
  - междинния сървер изоставя съобщението, ако не може да го предаде веднага на следващия получател
- синхронните комуникации изискват блокиране на изпращащия процес до получаването на съобщението в приемащия буфер (или дори до потвърждение на приемането му)
- при асинхронни комуникации  предаващия процес продължава да се изпълнява независимо от резултата от комуникацията
- времедиаграми на преходност и синхронност в комуникационните модели – 7.9

# RPC

- Отдалечено обръщение към (+ отдалечено изпълнение на) процедура (Birell и Nelson – 1984) – не специализирано, но прилагано главно за клиент-сървер приложения
  - с локални параметри
  - с връщане на резултата
  - с отлагане на викащия процес (синхронност)
  - чрез прозрачен (скрит) обмен на съобщения
  - маскиране на разликата в локалните адресни пространства (N.B.: в C към локалните параметри има обръщение по стойност и обръщение по адрес – специално към масивите)
  - единно кодиране на викащите параметри (+ резултата) между двата отдалечени процеси

# RPC операции

- базират се на клиентски и сървърни стъбове, които емулират RPC като обръщение към локална процедура
- клиентския стъб се извиква като локална процедура, пакетира параметрите (marshaling) в съобщение и  с обръщение към локалната ОС ги препраща към отдалечения сървер – операция send, след което извиква операция receive и изчаква резултата от сървера
- сървърния стъб извлича данните от съобщението и извиква локалната процедура на сървера (обикновено сървърния стъб е в състояние на изчакване на съобщение с параметрите)
- когато сървърната процедура върне резултата, параметрите са разположени в локалното адресно пространство на сървера, откъдето сървърния стъб ги пакетира в съобщение и ги връща към клиентския стъб (след което стартира операция за изчакване на ново съобщение)
- за отдалечено обръщение към повече от една процедура се проектират стъбове, които кодират с номер локалните процедури и извършват съответната дешифрация при обръщение

# Последователни стъпки на RPC – 7.12

- локално обръщение към клиентския стъб
- клиентския стъб съставя съобщение (marshaling – представително ниво) и извиква ОС
- клиентската ОС изпраща съобщението на сървърната ОС
- сървърната ОС предава съобщението на сървърния стъб
- сървърния стъб разпакетира параметрите от съобщението и извиква локалната сървърна процедура
- сървърната процедура се изпълнява и връща резултата на стъба
- сървърния стъб съставя съобщение и извиква сървърната ОС
- сървърната ОС изпраща съобщението на клиентската ОС
- клиентската ОС предава съобщението на клиентския стъб
- стъбът разпакетира съобщението с резултата и го предава на клиентското приложение, което възстановява изпълнението си
  - локалната адресация на параметрите и резултата е маскирана по отношение на отдалечения процес

# RPC в хетерогенна среда

- различни клиентската и сървърната ОС
- отчита се разликата в кодирането на символните параметри (напр. EBCDIC, ASCII …)
- отчита се разликата в кодирането на цели, фиксирани и плаващи формати на параметрите (little- /big-endian) и се разграничават числовите от символните параметри
- масивите се извличат и предават като последователност от параметри (вместо по адреси)
- за по сложните структури се прилага спецификация на параметрите със специален RPC протокол, който трябва да се поддържа от сървера и клиента
- стъбът на група процедури и съответния RPC протокол се имплементира като единен интерфейс – обикновено на същия език, на който се разработва клиент-сървер приложението – макар това да не е задължително

# IDL спецификация на RPC стъб

- за спецификацията на интерфейса се използва IDL – Interface Definition Language – като получения код се компилира да сървърен и клиентски стъб
- всяка middleware базирана RPC система предлага версия на IDL
- 7.14

# Асинхронно, косвено и еднопосочно RPC

- асинхронно RPC се прилага когато отдалечената процедура не връща резултат – клиентския процес изчаква само потвърждение от сървърния стъб, че заявката е приета, 7.15
- косвено (deferred synchronous) RPC се прилага при бавно изпълняващи се заявки с отложено обръщение на клиентската процедура към резултата
- за връщане на резултата от косвено RPC сървърът ползва еднопосочно RPC

# Разпределени обекти

- отдалеченото обръщение към обекти (и методи) разширява възможностите на RPC за прозрачни разпределени обектно-ориентирани приложения
- разпределените (distributed, remote) обекти поддържат изпълним интерфейс на отдалечена машина т.е. достъпа до техните методи може да се реализира чрез отдалечения интерфейс – 7.16
- свързването на клиент с отдалечен обект става чрез зареждане на неговия интерфейс – proxy – в адресното пространдство на клиента (аналогично на клиентския стъб при RPC) с единствена задача да пакетира (маршалва) обръщението към отдалечения обектен метод
- сървърът, на който е разположен обекта, изпълнява сървърен стъб (skeleton), който разпекетира обръщението и активира викания метод; след изпълнение на заявката, сървърният стъб пакетира отговора в съобщение до клиента
- N.B.: контекста на разпределените обекти също може да бъде разпределен между няколко сървера, което също остава прозрачно за клиентските програми през отдалечения интерфейс

# Версии на РО

- статични (compile-time; language-level – Java, C++) и динамични (runtime) обекти
- устойчиви (persistent) и преходни (transient) обекти

# Статични разпределени обекти

- статични: compile-time; language-level – Java, C++
  - статичните обекти позволяват компилирането и свързването на интерфейсите в клиентския и сървърния стъб
  - интерфейсът[ите] към статичния отдалечен обект е публичен и може да се компилира в приложенията

```
public interface SomeInterface { ... // method signatures }
```

  - отдаленият обект имплементира този (и други) интерфейси
  - приложението се обръща към този интерфейс на отдалечен обект

```
SomeInterface si =
(SomeInterface)TransparentItemProxy.getItem(
 "//someHost:1198/someName", new Class[] { SomeInterface.class
} );
```

  - отдалечените обръщения са с еднакъв синтаксис и семантика като локалните, но методите се изпълняват от отдалеченият сървер

# Динмични разпределени обекти

- динамични (runtime) обекти:
  - динамичните обекти преодоляват езиковата зависимост като приложението може да бъде разработено от обекти на различни езици
  - интерфейсът към отдалечения обект се зарежда от приложението по време на обръщение:

```
// obtain reference
Object object = Remote.getItem("//someHost:1198/someName");

// typically obtained at runtime
String someMethod = "someMethod";

// also obtained at runtime
Object someArgs = new Object[] { someArgs, ... };

Object result = Remote.invoke(object, someMethod, someArgs);
```

  - често се използват се общи потоколи за обмен вкл. групов обмен (UDP/IP multicast), което позволява абониране на приложението за обяви на сървери (announcements) и се маскира адреса на сървера (прозрачна адресация)

# Устойчиви и преходни обекти

- устойчиви (persistent) и преходни (transient) обекти
  - текущия контекст на устойчивите (постоянните) обекти се съхранява във вторичната памет и изпълнението им може да бъде преустановявано и подновявано от различни сървърни процеси
  - преходните обекти терминират щом се терминира и процеса (приложението), който е създал обръщението към съответния обект
  - устойчивите обекти могат да бъдат реферирани [успешно] и след терминиране на породилия ги процес
  - Java Data Objects (http://db.apache.org/jdo/state_transition.html) е пакет за управление устойчивостта на обектите (съхраняване на контекста) с използване на методи като `makePersistent()`, `makeTransient()`, `deletePersistent()` на клас `JDOHelper`

# Явно и неявно свързване клиент-обект

- Извикването на обектен метод се извършва само след свързване на обекта
- свързването на обект представлява разполагане на обектния интерфейс под формата на негово прокси в адресното пространство на викащия процес; свързването може да е неявно (автоматично) стига да се локализира сървера, който изпълнява съответния обект
- неявното свързване е синтаксис за обръщение към методи по указател (reference) към обекта (C++):

```
Distr_obj* obj_ref;        //declare object reference systemwide
obj_ref = <distr. obj. reference>;
obj_ref□obj_method();      //implicit binding and invocation
```

- явното свързване изпълнява същата задача чрез обръщение към специална функция bind, с която присвоява обектния указател на локален адрес – прокси:

```
Distr_obj obj_ref;         //declare object reference systemwide
Local_obj* obj_ptr;        //declare pointer to local object
obj_ref = <distr. obj. reference>;
obj_ptr = bind(obj_ref);// function call for binding
obj_ptr□obj_method();      // invocation
```

- указателят към разпределен обект съдържа информация за адреса на изпълняващия го сървер (машинен адрес + локален сърверен порт)
- понякога вместо “твърд” обектен адрес се използва предварително локализиране на обекта с помощта на location server (проблемна скалируемост, но прозрачност и преносимост на обектите по машини)

# Статично и динамично RMI

- след свързване на обекта обръщението към неговите методи – RMI – става през локалното обектно прокси:
  - обръщетието към обектните методи е прозрачно в рамките на системата
  - прилагат се специфични обектни стъбове за сървърната и клиентската страна (вместо стъбовете с общо предназначение при RPC)
- статичното обръщение използва готови интрефейсни дефиниции (изготвени напр. с IDL или с помощта на автоматичната стъб-генерация на Java)
  - при статичното обръщение интерфейсните дефиниции на разпределените обекти са известни на етапа кодиране на приложението и освен това промяната им изисква прекомпилиране на приложението
- динамичното обръщение генерира интерфейсните дефиниции по време на изпълнение на кода, което се задава със следния примерен синтаксис

```
invoke(object, method, input_params, output_params); // например
invoke(file_obj, id(append), int);//id(append) returns method's id
```

- динамично обръщение е по-удобно при програмиране напр. на браузери на разпределени обекти, при което обектните интерфейси трябва да се проверяват по време на изпълнение

# Предаване на обектни параметри при RMI

- предаването на переметри при RMI е по-прозрачно отколкото при RPC поради глобалната (т.е. systemwide, не повече!) идентификация на разпределените обекти – следователно ако параметъра е разпределен/отдалечен обект, то неговата стойност (която е указател!) може да се ползва в операцията свързване – явно или неявно
- тъй като се ползват не само отдалечени, а и локални [копия на] обекти (напр. за ефективност) – локалните обекти-параметри се предават към отдалечения обектен сървер не като указатели, а като стойността на самите локални обекти – 7.23

# peer-to-peer мрежи

- P2P мрежите ползват предимно разпределените общи ресурси на множество възли за протоколна обработка и за комуникации вместо централизираните ресурси на малък брой сървери и на опорните мрежи (backbone networks) – 7.24
- поради отсъствието на специализирани (dedicated) сървери p2p мрежите и приложенията за тях имат инцидентен (ad hoc) характер и имплементират специфичен не-йерархичен модел; всеки от процесите е равнопоставен – с клиентски и сървърни функции
- типични приложения:
  - споделяне на данни (файлове)
  - пренос на поточни данни (мултимедия - streaming, телефонни (VOIP)
  - текстови лични комуникации – форуми, чатове
- обикновено (но не винаги) p2p мрежите и приложенията разчитат на ограничена сървърна поддръжка – напр. IRC сървер за търсене и регистрация
- предимства
  - по-пълно използване на ресурсите
  - неограничена скалируемост
  - отказоустойчивост (проблематично поради ad hoc обслужването)
- проблеми
  - надеждност и качество на услугите
  - приложения в реално време
  - защита (нерегламентиран достъп; leeching – егоистична употреба; скрито зареждане на вредносен код; спам – нежелана информация)
  - правни проблеми – авторски права на съдържанието; нерегламентирана употреба на системни ресурси

# peer-to-peer (споделяне на файлове)

| протокол | приложения | описание |
|---|---|---|
| BitTorrent | споделяне$_1$ на файлове, BitTorrent и др., вграден | предимно Windows клиенти, ползва сървер директория – Tracker, допуска криптиране и проследяване на IP-адрес |
| eDonkey | споделяне$_2$, eMule, FlashGet и др., вграден | центр. сърверна директория ed2k и p2p съдържание, осн. Win клиенти, контрол на съотношение на споделяне |
| Gnutella | споделяне$_3$, за Mac, Win, JVM, Symbian | пълен p2p – не подлежи на закриване като Napster, статистика на възлите и на скоростта на обмен, набор протоколи: ping/pong, query/hit, push (за стени) |
| Kad Network | споделяне$_4$, eMule, eDonkey и др., вграден | разширение на ed2k – поддържа децентр. хеширан списък ключове на файловите имена, преодоляване на стени |

# peer-to-peer (общи приложения)

| протокол | приложения | описание |
|---|---|---|
| Skype protocol | VoIP, Mobile VoIP, файлове и собщ., видеоконференция | VoIP функции но в p2p мрежа с разпределен регистър и йерархия възли – supernodes (в host cache на клиента )+ login server |
| Domain Name System | разпределен списък форматни имена и IP-адреси | поддържа йерархичен списък с адреси на възли и сървери – напр. мейл-сървери, опресняване (time-to-live) |
| JXTA | p2p Java/C/C++ приложения с XML обмен | настолни и вградени (клетъчни телефони, PDA) приложения, 2 нива на възлите (superpeers със сърверни функции) |
| P2PTV – 6.26 | поточна MM, глобализирано пре-предаване на TV прогр. | негарантирано QoS – вкл. при пре-предаващите възли, възможно предаване на собствено съдържание |
| Windows Peer-to-Peer | разработване на p2p приложения с общо предназначение | SW-компонент в XPSP2/Vista върху IPv6 – мета-мрежа (групови списъци на включени възли; publish/subscribe модел) |

# BitTorrent протокол

- портокол, технология и среда за споделяне на файлове в ад-хок p2p мрежи
- 1/3 от Интернет трафика и 1/5 от висикоскоростния трафик (асиметрията е понеже се поддържа основно от настолни компютри и потребителски мрежи, а не от висикоскоростните опорни мрежи)
- базира се на споделяне на дискови, компютърни, комуникационни ресурси с минимална сърверна поддръжка
- BT-клиента:
  - поддържа списък на локални работни копия на споделените файлове и прозрачно обслужва заявки (seeder)
  - генерира последователност от заявки за зареждане на файл
    - заявките са TCP-формат – към множество възли поддържащи отделни части от файла (контраст: при web-браузърите единична HTTP GET заявка към един сървер – дори и при реплики на файла) – по-сложно управление (signaling), но потенциално уплътнен трафик
    - негарантирана скорост и ред на зареждане – неподходящ за изпреварващо зареждане на поточни MM данни (progressive streaming) – текущи разработки за преодоляване
    - планът на заявките е rarest-first
  - зарежда метаданни за заявен файл – torrent – списък с текущите клиенти, поддържащи копия на части от файла и адрес на сървера, координиращ процеса (tracker)
- BT-сървер – tracker: директория с мета-данни за съдържанието (торенти – на файлово ниво); контролира коефициента на споделяне на клиентите (за избягване не leeching)

# Комуникации, базирани на съобщения

- MOM (Message Oriented Middleware), messaging – комуникационен модел за асинхронни разпределени приложения, базиран на обмен на съобщения
  - допуска приложения и със синхронни комуникации – вж. MPI
  - обменът е прозрачен и индиректен за комуникиращите процеси на приложението (вкл. “леки”, мобилни и преносими приложения)
  - съобщенията се съхраняват в опашки на сървърната инфраструктура
  - “развързването” на процесите с MOM е подходящо за CS и обектните модели
  - ограничена приложимаст за online и PB
  - подобрява планирането и обслужването в комуникационната инфраструктура – load balancing и т.н.
- моделите на обмен на съобщения са устойчиви и преходни
  - устойчивите съобщения (обикновено пакети данни) се записват във вторична памет
  - неустойчивите съобщения не се съхраняват при рестартиране на системата – напр. мониторинг
- най-разпространени технологии:
  - MPI
  - IBM's MQSeries,
  - Microsoft Message Queuing
  - DECmessageQ
  - прилагат [несъвместими] API – но съществуват протоколни конвертори (gateways, шлюзове) зе пренос на съобщенията в няколко среди – напр. между IBM и MS

# MOM – модел с преходни комуникации (Message Passing)

- MOM (както и приложенията с обмен на съобщения) се базират директно върху примитивите на транспортното ниво посредством съединения (sockets)
- съединенията са крайни точки на комуникационните канали
- достъпни за {P|M}ОС, които формират системен комуникационен интерфейс и се използват за достъп до специфична примитивна услуга *на транспортния протокол*
- технически съединенията представляват буфери, в които се съхраняват изпращаните или приеманите съобщения докато се извърши съответната операция
  - съединителни примитиви в TCP (7.29)
    - socket откриване на ново съединение
    - bind присвояване на локален адрес на съединението
    - listen деклариране на готовност за откриване на комуникационни канали (connections)
    - connect заявка за откриване на комуникационен канал (обикн. на клиента)
    - accept приемане на заявка за откриване на комуникационен канал с блокиране на заявителя докато се създаде ново съединение (обикн. на сървера)
    - send изпращане на съобщение по канала
    - receive приемане на съобщение по канала
    - close закриване на комуникационен канал
  - чрез транспортните съединения се реализира ***синхронен модел*** с гаратирана доставка на дългите съобщения като проверена и подредена последователност от пакети
- буферирането на съобщенията – за асинхронни комуникации – се извършва от message passing платформата

# Message passing с MPI

- MPI (Message-Passing Interface) е стандартизирана платформа за MOM с преходни комуникации – обмен на съобщения в група от процеси
- комуникиращите процеси се идентифицират уникално в рамките на групата;
- обикновено една група съответства на едно разпределено приложение;
- може да се активни едновременно няколко групи процеси (с възможно припокриване), така че комуникационните адреси се състоят от двойката <groupID; processID>
- основни MPI примитиви:

| | |
|---|---|
| `MPI_send` | изпращане и изчакване докато съобщението се запише в *локален или отд. буфер* |
| `MPI_bsend` | изпращане и изчакване докато съобщението се запише в *локален изходен буфер* |
| `MPI_ssend` | изпращане и изчакване докато се върне *потвърждение (receipt)* |
| `MPI_isend` | предаване на указател към съобщение за изпращане на резидентния MPI процес |
| `MPI_issend` | като `MPI_isend` с изчакване на *потвърждение* |
| `MPI_sendrecv` | изпращане и изчакване на *отговор* |
| `MPI_recv` | блокиращо получаване на съобщение (с изчакване докато няма съобщение) |
| `MPI_irecv` | проверка и евентуално получаване на пристигащо съобщение (без блокиране) |

# MOM - модел с устойчиви комуникации (Message Queuing)

- подходящ за асинхронни приложения с буфериране на съобщенията в транзит и с време на обмен от порядъка на минути (вместо ~$10^{-1}$S при MPI/TCP съединенията) – поради обема и/или дистанцията
- едно или повече приложения от един възел (адрес) имат приемен буфер за съобщения (message queue); адресирането на съобщенията се извършва именно по уникалното име на буфера
- при MOM буферите винаги са локални с обслужваните от тях процеси или поне в същата локална мрежа, но наборът от възможни буфери се разпределя между възлите
- буферните имена се транслират до мрежови адреси, за което се поддържа транслираща таблица – като при DNS и e-mail адресите – 7.31

# Основни MQ примитиви

- `put` – добавяне на съобщение в буфер без блокировка
- `get` – извличане на първото съобщение от [FIFO] буфер с блокиране ако е празна
- `poll` – извличане на първото съобщение от [FIFO] буфер без блокиране ако е празна
- `notify` – стартиране на резидентен процес, следящ за появата на съобщение в буфера
- IBM MQSeries – вж. http://www-306.ibm.com/software/integration/wmq/index.html )

# MQ инфраструктура

- Буферни мениджъри (queue managers) – 2 типа (7.33.1):
  - за поддържане на интерфейс към комуникиращите приложения
  - маршрутизатори (щафетни постове, relays):
    - маршрутизацията се базира на таблици с двойките маршрутизатор-мрежов адрес;
    - мрежовия адрес се извлича от идентификатора на получаващия процес
    - поддържат регистър на предадените съобщения (log) за защита и отказоустойчивост
    - обслужват заявките за групово предаване (multicast) като изпращат репликират изпратеното съобщение
- Брокери на съобщения (message brokers)
  - форматът на обменяните съобщения в разпределените асинхронни приложения не е стандартизиран (както в мрежовите протоколи);
  - брокерите тук са в ролята на форматни конвертори или шлюзове (gateways, но на приложно ниво!), всъщност са част от разпределеното приложение, а не мрежов компонент – 7.33.2
  - пример: преформатиране на разделителите в таблица от база данни
  - функционират с помощта на база данни с конвертиращи правила между форматите на съобщенията (подобно на конверсията на електронната поща между X-400 и Интернет)
    - безопасно е да се приеме, че конверсията не гарантира 100% коректност на съобщенията