Middleware - Vypracovane zkouskove otazky

1. Vysvetlete client-server architekturu

- architektura, která popisuje aplikaci, jenž je rozdělená na dvě části: klientskou a serverovou
- klient prezentuje data, server se stará o jejich uložení, výpočet nebo přípravu
- druhy klienta:
    - Thick client – většina práce probíhá u klienta, snižuje se režie na přenos po síti, vytíženost serveru, komplikovanější aktualizace SW na všech klientech (např. distribuovaný filesystem)
    - Thin client – většina práce probíhá na serveru, umožňuje to velmi zjednodušit klienta a jeho zařízení, snadná aktualizace SW (např. klienti jako terminály)

2. Vysvetlete two-tier architekturu

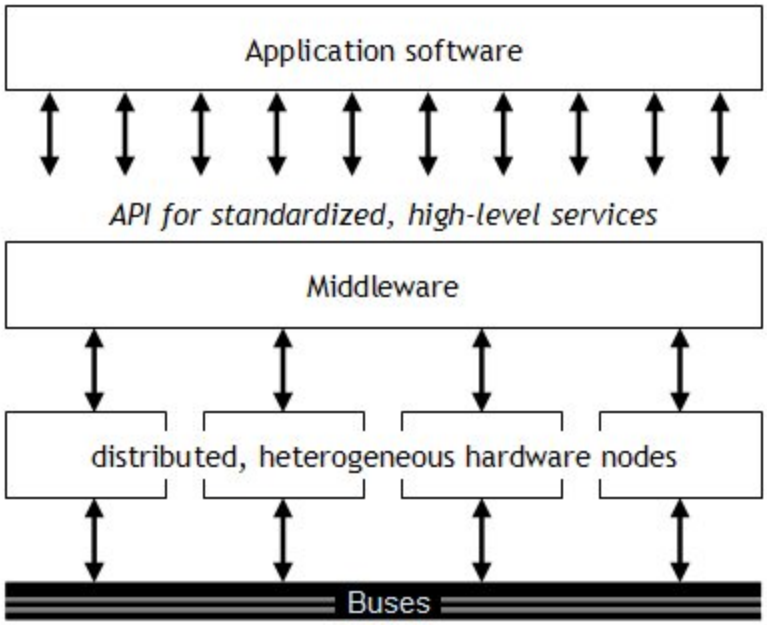
- viz client-server architektura
- 2 tiers - rozdeleni na prezentacni cast (klient) a na cast, kde probiha rizeni aplikace a zpracovani dat (server), tyto casti jsou fyzicky oddeleny

3. Vysvetlete three-tier architekturu

- client-server architektura s přidanou vrstvou – middle tier, implementuje aplikační logiku nebo takové vlastnosti aplikace, které stojí na pomezí mezi vývojářem aplikace a síťaři
- prezentační vrstva, aplikační vrstva, databáze (data management)

4. Vysvetlete termin middleware

- vrstva, ktera z lokalni aplikace vytvari distribuovanou
- vrstva, ktera propojuje klientskou a serverovou cast aplikace
- vrstva s vlastnostmi, ktere znacne zjednodusuji vyvoj aplikacni a serverove casti



5. Define the properties of an ideally reliable communication mechanism in terms of messages sent and received by the participants.

- kazda zprava (data) odeslana odesilatelem je prijata prijemcem prave jednou a neposkozena
- nejsou prijaty zadne zpravy, ktere nebyly odeslany
- tj exactly-once semantika

6. Describe the circumstances under which packet damage cannot be masked by the approaches used to provide reliable communication

- data duplication (=každá zpráva se posílá vícekrát) - nezjistí se chyba, pokud je poškozen bit ve zprávě a odpovídající bit v duplikované části
- checksum, parita – nezjistí se chyba vzniklá poškozením dvou bitů ve zprávě (poskodi se treba bit 1 na 0 a nekde jinde bit 0 na 1 → checksum i parita zustanou stejne)
- cross parity – nezjistí se chyba vzniklá poškozením tří bitů ve zprávě
- Hamminguv kod - i u rozsireneho se nezjisti chyba poskozenim tri bitu, napr. prvni tri
- CRC – vícebitové chyby vzdálené více než 16 bitů (u CRC-16)

7. Describe the circumstances under which packet loss and packet duplication cannot be masked by the approaches used to provide reliable communication.

- amnezie - chyba v cislovani zprav

8. The TCP flow control mechanism is based on the receiver informing the sender of the number of bytes that it can still accept. Explain why this approach is used instead of the receiver simply telling the sender whether it can accept any data or not.

- Příjemce může specifikovat přesně velikost dat, které může ještě zpracovat.
- Kdyby informoval pouze volno/plno, nepoznal by odesilatel, kolik toho muze jeste poslat a kdyby poslal vic, mohl by být jednoduše prijemce zahlcen a režie na nový přenos již odeslaných paketů by byla větší než info o oknu příjemce.

9. Navrhněte přenosový protokol, který bude zaručovat spolehlivé doručování zpráv od jednoho odesílatele jednomu příjemci. Váš návrh by měl definovat tyto funkce:

```
void ReliableSend (tMsg *pMessage, tAddr *pTarget);
 // Odeslání zprávy, blokuje do přijetí zprávy

void ReliableReceive (tMsg &*pMessage, tAddr &*pSource);
 // Příjem zprávy, blokuje do přijetí zprávy, zaručuje
 // právě jeden příjem nepoškozené odeslané zprávy
```

Váš návrh by měl používat tyto funkce:

```
void UnreliableSend (tMsg *pMessage, tAddr *pTarget);
 // Odeslání zprávy, neblokuje, nemusí odeslat

void UnreliableReceive (tMsg &*pMessage, tAddr &*pSource, int iTimeout);
 // Příjem zprávy, blokuje do timeoutu, může přijmout
 // poškozenou zprávu nebo tutéž zprávu vícekrát
```

Dále předpokládejte existenci rozumných funkcí pro manipulaci se zprávami jako je jejich vytváření a rušení, nastavování a dotazování atributů přenášených spolu s obsahem zprávy a podobně.

Pozor: při odesílání - pokud nesouhlasí hash, je třeba poslat NAK, a počkat na opětovný příjem zprávy. Při odesílání ACK je třeba přidat do zprávy id.

```
reliable_send(message, target)
{
  id = unique++; //unique je globalni sekvence pro daneho targeta

  m = pack_message(message, id, hash(message))
```

```
  unreliable_send(m);
  while(1);
  {
    unreliable_recieve(message, target, timeout);
    if (message.type == ACK && message.id == id) break;
    unreliable_send(m);
  }
}

reliable_receive(message, source)
{
  while(1)
  {
    unreliable_recieve(inmessage, source, timeout)
    if(hash(inmessage.zprava) == inmessage.hash)
    {
      if(inmessage.id < id_current)
      {
        unreliable_send(ACK, source);
      }
      if(inmessage.id == id_current)
      {
        unreliable_send(ACK, source);
        message = inmessage;
        id_current++;
        return 0;
      }
   }
}
```

10. The sender initiated and receiver initiated error recovery schemes differ in which of the two communicating sides is responsible for keeping track of delivered packets. Explain how this difference impacts the management of memory used to store packets.

- sender initated – Odesílatel si musí udržovat seznam těch, od nichž mu již přišlo potvrzení, což může být ve velkých sítích velmi vysoké číslo, špatně se škáluje. Po přijetí všech potvrzení může odesílatel paket smazat.
- reciever initiated – NAK při nepřijmutí paketu, musí ale vědět, že měl data přijímat. To lze například v souvislém proudu paketů, pak se používá timeout. Odesílatel neví, jak dlouho si u sebe nechat kopii paketu, protože negativní potvrzení může přijít kdykoliv. (jaktoze muze prijit “kdykoliv” - asi viz reseni 12.)

11. The use of positive acknowledgements in multicast can lead to network congestion problems due to excessive acknowledgement traffic, also termed as

ACK implosion. Outline scenarios in which this problem occurs and approaches used to remedy the problem while preserving positive acknowledgements.

- prilis mnoho uzlu potvrdi prijem paketu v jednom okamziku a tim se zahlti sit pobliz odesilatele
- resenim je udelat stromovu topologii site, ACK posilaji pouze potomci primym predchudcum pote, co dorazi ACK od vsech potomku

12. The use of negative acknowledgements in multicast can lead to network congestion problems due to excessive acknowledgement traffic, also termed as NAK implosion. Outline scenarios in which this problem occurs and approaches used to remedy the problem while preserving negative acknowledgements.

- Nastává při reciever-initiated strategii, když moc příjemců neobdrží v timeoutu paket (např. pokud se paket ztratí hned blízko odesilatele), odesílají NAK – těch se hodně sejde u odesílatele a síť se zahltí.
- Řešení: NAK odesílat multicastem po náhodné době. Pokud během této doby dostanu NAK od někoho jiného, nemusím již NAK posílat.

13. Define the source ordering guarantees in the context of multicast communication and explain in what situations and why this ordering is useful.

- Data od jednoho odesílatele budou přijata ve stejném pořadí, jako byla odeslána.
- Užitečné u serveru, který je stavový pro jednotlivé klienty, ale klienti jsou na sobě nezávislí.

14. Define the causal ordering guarantees in the context of multicast communication and explain in what situations and why this ordering is useful.

- Data jsou obdržena podle kauzální závislosti: jestliže existuje proces, ve kterém událost e1 předchází e2, potom e2 kauzálně závisí na e1, příjem zprávy kauzálně závisí na odeslání téže zprávy, funguje tranzitivita.
- Data o příčinách přijdou před daty o následcích.
- Užitečné u distribuovaného chatu – chceme, aby každému dorazila zpráva, na kterou odpovídám, dříve než moje odpověď.

15. Define the total ordering guarantees in the context of multicast communication and explain in what situations and why this ordering is useful.

- Totální kauzální doručování – data budou doručena všem příjemcům ve stejném pořadí a toto pořadí bude kauzální.

- Užitečné při duplikaci serverů – jeden server je hlavní a vykonává požadavky klientů, druhý server dělá to samé, aby měl stejný stav jako hlavní server a mohl ho v případě jeho výpadku nahradit. Je tedy potřeba, aby záložní server dělal přesně to samé, takže je potřeba aby požadavky dostával v naposto totožném pořadí jako server hlavní.

16. In a form of algorithms used by the senders and the receivers, sketch the approach used to achieve source ordering in multicast communication.

- Odesílatel provádí sekvenční číslování paketů a příjemce po obdržení zprávu přijme pouze v tom případě, že již dříve přijal zprávu s předcházejícím ID od toho samého odesílatele.

17. In a form of algorithms used by the senders and the receivers, sketch the approach used to achieve causal ordering in multicast communication. Use of a distributed algorithm is considered better than use of a centralized one.

- Kauzalní uspořádání zajistí vektorové hodiny, které nastavuje odesílatel. Příjemce po obdržení zprávu přijme pouze v tom případě, že nejvýše jedna složka hodin ve zprávě je právě o 1 větší než na hodinách příjemce a ostatní jsou menší. (tj. Příjemce po obdržení zprávu doručí pouze v tom případě, že již dříve přijal zprávu s předcházejícím ID ze stejné skupiny.)
- Pro překrývající se skupiny by se použily maticové hodiny.

18. In a form of algorithms used by the senders and the receivers, sketch the approach used to achieve total ordering in multicast communication. Use of a distributed algorithm is considered better than use of a centralized one.

- Sekvenční číslování zpráv zajištěné centrální autoritou, příjemce po obdržení zprávu přijme pouze v tom případě, že již dříve přijal zprávu s předcházejícím ID ze stejné skupiny.
- distribuovaný total-order protokol: skalární vektorové hodiny. Odesílatelé rozešlou zprávy, příjemce při příjmu vrací odesílateli potvrzení TSAi, odesílatel po přijmutí všech potvrzení odešle finalizační zpávu s TSF = max(TSAi), po příjmu finalizační zprávy doručí příjemce zprávy podle TSF.

19. The Lamport Clock is a logical clock used in some distributed algorithms. Outline the algorithm used to calculate the Lamport Clock timestamp and explain what are the useful properties of the timestamp.

- Každý proces ve skupině má svůj timestamp.
- Timestamp se zvedá při každé důležité události procesu.

- Při odesílání zprávy se k ní připojí aktuální timestamp (proces i vysílá v čase Ci(a) zprávu m, Tm = Ci(a).
- Při příjmu zprávy se případně zvedne timestamp nad přijatý (proces j přijme zprávu m v čase Cj(b), pak Cj = max(Cj(b), Tm+1).
- Pokud událost A kauzálně předchází události B, pak timestamp A je menší než B. Naopak to neplatí.

20. The Vector Clock is a logical clock used in some distributed algorithms. Outline the algorithm used to calculate the Vector Clock timestamp and explain what are the useful properties of the timestamp.

- Každý proces má vektor o délce n = počet procesů ve skupině, vektor obsahuje timestampy všech procesů ve skupině (VT(p)…časová značka procesu, VT(m)…časová značka zprávy).
- Proces i zvyšuje svůj timestamp při důležité události (při použití na kauzální řazení zpráv je to odeslání zprávy): VT(p)[i]++;
- Odeslání zprávy procesem pi: VT(pi)[i]++; VT(m) = VT(pi);
- Přijetí zprávy m procesem pj: proces pj přijme zprávu po obdržení až ve chvíli, kdy VT(m)[i] je právě o 1 větší než VT(pj)[i], a ostatní složky VT(pj) jsou větší nebo rovny odpovídajícím složkám ve zprávě.
- Událost A kauzálně předchází B <=> timestamp A je menší než B.
- Událost A nemá kauzální vztah k B <=> timestamp A není porovnatelný s B.
- Porovnává se po složkách a je větší pokud všechny složky jsou větší nebo rovno a jedna větší. Naopak analogicky.

21. Present a suitable example of a multicast communication in which sender ordering is violated. Explain why the ordering is violated. Use a notation in which S(A,X->B,C) denotes node A sending message X to nodes B and C and R(A,X) denotes node A receiving message X.

- S(A, X1 → B)
- S(A, X2 → B)
- R(B, X2)
- R(B, X1)
- B prijalo zpravy od A v opacnem poradi, nez v jakem je A odeslalo

22. Present a suitable example of a multicast communication in which causal ordering is violated, but less strict ordering guarantees are preserved. Explain why the ordering is violated. Use a notation in which S(A,X->B,C) denotes node A sending message X to nodes B and C and R(A,X) denotes node A receiving message X.

- S(A, X1 → B,C)
- R(B, X1)
- S(B, X2 → C)
- R(C, X2)
- R(C, X1)
- zprava X2 mohla zaviset na X1, ale C je dostalo v opacnem poradi

23. Present a suitable example of a multicast communication in which total ordering is violated, but less strict ordering guarantees are preserved. Explain why the ordering is violated. Use a notation in which S(A,X->B,C) denotes node A sending message X to nodes B and C and R(A,X) denotes node A receiving message X.

- S(A, X1 → B)
- S(B, X2 → A)
- R(B, X1)
- R(A, X2)
- A vidi realitu tak, ze se prvni stalo X1 a pak X2, ale B vidi realitu tak, ze prvni se stalo X2 a potom X1, takze nevidi realitu stejne, coz pri totalnim usporadani maji

24. Navrhněte přenosový protokol, který bude zaručovat spolehlivé doručování zpráv od jednoho odesílatele více příjemcům. Váš návrh by měl definovat tyto funkce:

```
void ReliableSend (tMsg *pMessage, tAddrList *pTargetList)
       // Odeslání zprávy, blokuje do přijetí zprávy všemi příjemci

void ReliableReceive (tMsg &*pMessage, tAddr &*pSource);
       // Příjem zprávy, blokuje do přijetí zprávy, zaručuje
       // právě jeden příjem nepoškozené odeslané zprávy

Váš návrh by měl používat tyto funkce:

void UnreliableSend (tMsg *pMessage, tAddr *pTarget);
       // Odeslání zprávy, neblokuje, nemusí odeslat

void UnreliableReceive (tMsg &*pMessage, tAddr &*pSource, int iTimeout);
       // Příjem zprávy, blokuje do timeoutu, může přijmout
       // poškozenou zprávu nebo tutéž zprávu vícekrát

Dále předpokládejte existenci rozumných funkcí pro manipulaci se zprávami jako je jejich
vytváření a rušení, nastavování a dotazování atributů přenášených spolu s obsahem zprávy a
```

podobně a existenci rozumných funkcí pro manipulaci se seznamem adres jako je jeho procházení.

Zadání vyžaduje návrh protokolu pro spolehlivý multicast nad nespolehlivým unicastem, na rozdíl od obvyklejšího návrhu spolehlivého multicastu nad nespolehlivým multicastem. Vysvětlete, jaký má toto omezení vliv na vlastnosti vašeho návrhu.

```
void ReliableSend (message, targetlist)
{
	for target in targetlist
	{
		id = ids[target]++; //ids je globalni proměnná pro počítání zpráv

		m = pack_message(message, id, hash(id, message))

		while(1)
		{
			unreliable_send(m, target);
			unreliable_recieve(msg, target, timeout);
			if msg != null && msg.hashIsOk &&
			   msg.type == ACK && message.id == id
			{
				break;
			}
		}
	}
}
```

Algoritmus pro odesílání by šlo vylepšit tak, že by nejprve odeslal všechny zprávy a potom čekal na potvrzení a případně odesílal zprávy znovu, ale tím by se kód výrazně zkomplikoval.

```
void ReliableReceive (message, source);
{
	while(1)
	{
		unreliable_recieve(msg, source, timeout)
		if msg != null && msg.hashIsOk && type == DATA
		{
			if msg.id < id_current
			{
				sendAck(msg.id, source); //Automaticky udělá heš
			}
			else if msg.id == id_current
			{
				sendAck(msg.id, source);
```

```
                        message = msg;
                        id_current++;
                        return;
                    }
                }
            }
}
```

vliv na navrh: vubec nesetri traffic

25. Explain how distributed hashing can be employed to implement a simple distributed file sharing service. Discuss the limitations of such an implementation in terms of the search query complexity.

- cesta, název = klíč; obsah souboru/adresáře = hodnota
- Uzlům sítě jsou přiřazeny identifikátory.
- Každý uzel má definovanou sadu operací: store(key, value), lookup(key, returnAddress), join(id), leave(id), všechny je možné provést přes libovolný z uzlů.
- uzel má na starost soubory, jejichž klíč je nejblíže ID uzlu
- O úspěšnosti sítě rozhoduje
    - graf překrytí uzlů (outgoing arity of each node),
    - mapování klíčů na identifikátory (ve kterém uzlu má být hodnota pro daný klíč uložena),
    - lookup (vyhledání uzlu, který se stará o daný klíč),
    - jak přidávat nové uzly do grafu a jak je zase odebírat (join, leave).
- U systému Chord je doba na lookup omezena shora M, ideálně je O(log2N).

26. Design an interface of a messaging middleware that is suitable for a highly efficient transport of messages between processes of a tightly coupled homogeneous multiprocessor cluster, to be used for scientific calculations. Explain your design choices and advocate the suitability of your design. (TODO: je to dostatecne?)

- Send(target, *message_ptr)
- Receive(source, *message_ptr)
    - Je možné použít předávání pomocí reference, jelikož ukazatelé odesílatele budou dávat u příjemce smysl.
    - Send zprávu nahraje do sdílené paměti všemi procesy a pošle ukazatel do této paměti.
    - Není potřeba se zabývat reprezentací dat – v rámci homogenního clusteru bude stejná.
    - Velikost zprávy může být uložená na prvním bloku paměti, kam míří ukazatel.
    - Multicast

    - Synchronizace
    - …

27. Design an interface of a messaging middleware that is suitable for an internet wide transport of messages between heterogeneous desktop computers, to be used for thick client information system running on multiple client platforms. Explain your design choices and advocate the suitability of your design. (TODO: je to dostatecne?)

- Send(target, message, representation_description, size)
- Receive(source, message, representation_description, size)
    - Je potřeba popsat reprezentaci dat – u příjemce se může lišit.
    - reprezentace zapsána ve zprávě – Zpráva je větší, ale konvertuje se jen jednou.
    - reprezentace je jednotná – Zpráva je malá, ale je možné, že se bude konvertovat na obou stranách.
    - Je vhodné uvést velikost zprávy, aby nemusely být všechny stejně velké.
    - předávání hodnotou – pochopitelně

28. When invoking a procedure that passes an argument by value, RPC middleware has to handle situations where binary representation of the values differs between the client and the server. Outline and discuss the approaches used to do this.

- standardizovaná reprezentace
    - krátká zpráva
    - nebezpečí zbytečných konverzí (až 2 místo 0)
- popis reprezentace uložen přímo ve zprávě
    - dlouhá zpráva
    - minimalizován počet konverzí
- reprezentace domluvena mezi komunikujícími (např. na začátku komunikace)
    - krátké zprávy
    - minimalizován počet konverzí
    - stavovost

29. When invoking a procedure that passes an argument by reference, RPC middleware has to handle situations where the reference has no meaning outside the address space of the client or the server. Outline and discuss the approaches used to do this.

- Obvykle se převádí na parametr určený hodnotou.

- Objekt, na který ukazatel ukazuje, je nejprve zkopírován (resp. přenesen) i na server. Vzdálená procedura pak při svém volání dostane ukazatel na tento zkopírovaný exemplář programového objektu, který pak může v rámci své činnosti příslušným způsobem modifikovat. Jakmile provádění vzdálené procedury skončí, je dotyčný objekt zrušen, změny se tak nevrací na klienta. (pro malé objekty)
- problémy: reference na funkce, reference mohou obsahovat další reference, jak přenést kód objektu
- Klient se může stát také serverem a poskytovat objekt předaný jako parametr vzdáleně. Ve volání se pak přenese např. serializovaná proxy. (pro velké objekty se složitými funkcemi)

30. Napište kód stubu na straně klienta a na straně serveru tak, aby zprostředkoval vzdálené volání funkce int read (int iFile, void *pBuffer, int iSize), která prečte data z otevřeného souboru. Váš návrh by měl používat tyto funkce:

```
void ReliableSend (tMsg *pMessage, tAddr *pTarget);
	// Odeslání zprávy, blokuje do přijetí zprávy

void ReliableReceive (tMsg &*pMessage, tAddr &*pSource);
	// Příjem zprávy, blokuje do přijetí zprávy, zaručuje
	// právě jeden příjem nepoškozené odeslané zprávy

Dále předpokládejte existenci rozumných funkcí pro manipulaci se zprávami jako je jejich
vytváření a rušení, přístup k obsahu zprávy a podobně.
```

```
int read (int iFile, void *pBuffer, int iSize)
{
	tMessage *pRequest = new tMessage;
	pRequest->PutInteger (SRV_FReadID);
	pRequest->PutInteger (iFile);
	pRequest->PutInteger (iSize);
	ReliableSend (pRequest, SRV_Address);
	delete (pRequest);
	tMessage *pReply = ReliableReceive (SRV_Address);
	int iResult = pReply->GetInteger ();
	if (iResult > 0)
		pReply->GetBlock (pBuffer, iResult);
	delete (pReply);
	return iResult;
}

void ServerLoop (void)
{
```

```
	while (TRUE) {
		tMessage *pRequest = ReliableReceive (CLT_Address);
		int iFunctionID = pRequest->GetInteger ();
		switch (iFunctionID) {
			case SRV_ReadID:
				int iFileID = pRequest->GetInteger ();
				int iLength = pRequest->GetInteger ();
				byte *pBuffer = malloc (iLength);
				int iResult = read (iFileID, pBuffer, iLength);
				tReply *pReply = new tMessage ();
				pReply->PutInteger (iResult);
				if (iResult > 0)
					pReply->PutBlock (pBuffer, iResult);
				ReliableSend (pReply, CLT_Address);
				delete (pReply);
				free (pBuffer);
				break;
		}
		delete (pRequest);
	}
}
```

31. To achieve maximum efficiency, the GM library is designed to minimize the need for copying data while communicating. Explain how the design minimizes data copying.

- Aplikace sama poskytne GM jeden buffer (lépe dva) pro každou prioritu a délku zprávy, kterou může přijmout – GM přímo tyto buffery plní a aplikace z nich čte, takže není potřeba nic kopírovat.
- Jsou-li poskytnuty dva buffery, jeden se plní, zatímco druhý se zpracovává.

32. Vysvetlete, jak standardy SOAP, WSDL a UDDI spolupracuji v prostredi web services.

- UDDI – Universal Description, Discovery, and Integration: registers and locates web services, obsahuje informace o providerovi služby a její WSDL popis
- WSDL – Web Service Description Language: data type, message format, encoding, protocol and network address
- SOAP – Simple Object Access Protocol, komunikační protokol, XML based, je možné jej přenášet přes HTTP nebo TCP.
- Poskytovatel služby se nejprve zaregistruje v UDDI a předá mu WSDL služeb, které poskytuje. Klient se potom dotáže UDDI, kdo poskytuje dané služby a UDDI mu vrátí informaci o umístění poskytovatele a WSDL jeho služeb. Klient potom komunikuje přímo s poskytovatelem a využívá jeho služeb. Veškerá komunikace probíhá přes SOAP.

33. Describe the reasons for choosing XML as the transport encoding in web services.

- Je to standardizovaný formát.
    - Již pro něj existují hotové knihovny pro parsování.
    - Mohou mu porozumět i uzly na cestě (firewall může propouštět jen některé zprávy).
- Dá se pomocí XSLT snadno (a deklarativně) přeložit do jiných formátů (např. adaptovat pro jiný formát stejné služby).
- Dá se snadno validovat.
- Je rozšiřitelné.

34. DCE relies on UUID (Universally Unique Identifier) as a unique identifier to distinguish interfaces. Explain how an UUID can be generated so that its uniqueness is guaranteed.

- Existují různé způsoby generování z hodnot
    - výrobní čísla HW,
    - MAC adresa,
    - (pseudo)náhodná čísla,
    - sekvenční čísla,
    - čas
    - kombinace sitove adresy, casu
    - ...
- Při generování se používá hešování, aby neunikla citlivá informace.

35. Using an argument with the pipe annotation, DCE allows a stream between the client and the server to be created within the context of a remote call. The argument with the pipe annotation is mapped to a pair of push and pull functions. Explain how these functions are used.

- Funkce implementuje klient.
- Pipe je složena z „chunks“, na začátku je v pipe specifikován počet elementů v chunku, elementem jsou typovaná data.
- Pipe může být
    - input parameter (klient->server kanál) – implementováno metodou pull,
    - output parameter (server->klient kanál) – implementováno metodou push,
    - nebo obojí.
- push – Pošle chunk ze serveru ke klientovi.
- pull – Požádá klienta o další chunk.
- Pipe není idempotentní – nemůže být obsahem jiné pipe.

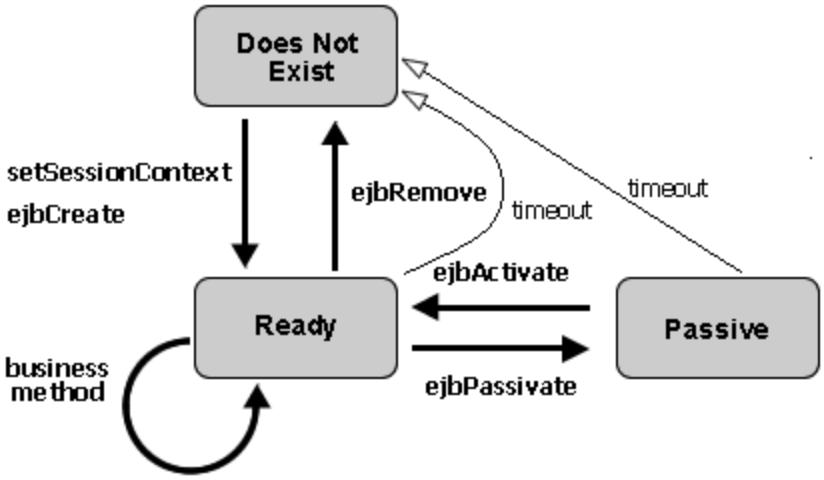
36. Explain what a bean and a container is in EJB.

- bean – Server-side komponenta zapouzdřující bussiness logiku aplikace.
- container – Poskytuje beany s určitými službami, např. lifecycle, transakce; umožňuje klientům přístup k beanům. Je to místo ve kterém jsou beany umístěny.

37. The EJB standard defines stateful session beans, stateless session beans, message driven beans and entities. Describe the basic properties and the intended application of these four types of beans.

- Session bean reprezentuje jednotlivého klienta uvnitř J2EE serveru.
- stateful session beans – Stav objektu přetrvává mezi voláním jednotlivých metod. = „conversation state“
- stateless session beans – Bezestavová entita, škálovatelná, může být použita více klienty (pozor na reentranci).
- message drive beans
    - klient nepřistupuje přes interface,
    - neuchovává si žádná data ani stav konverzace,
    - všechny její instance jsou ekvivalentní,
    - jedna instance může přijímat zprávy od více klientů
    - klient přistupuje přes JMS
- entities – pro databázové entity, „represents a business object in a persistent storage mechanism“, obvykle má každá entita příslušnou tabulku v databázi a každá instance entity odpovídá řádku v tabulce

38. Describe the lifecycle of a stateful session bean in EJB, that is, when instances of the bean are, or appear to be, created and destructed, from both the client and the server points of view.

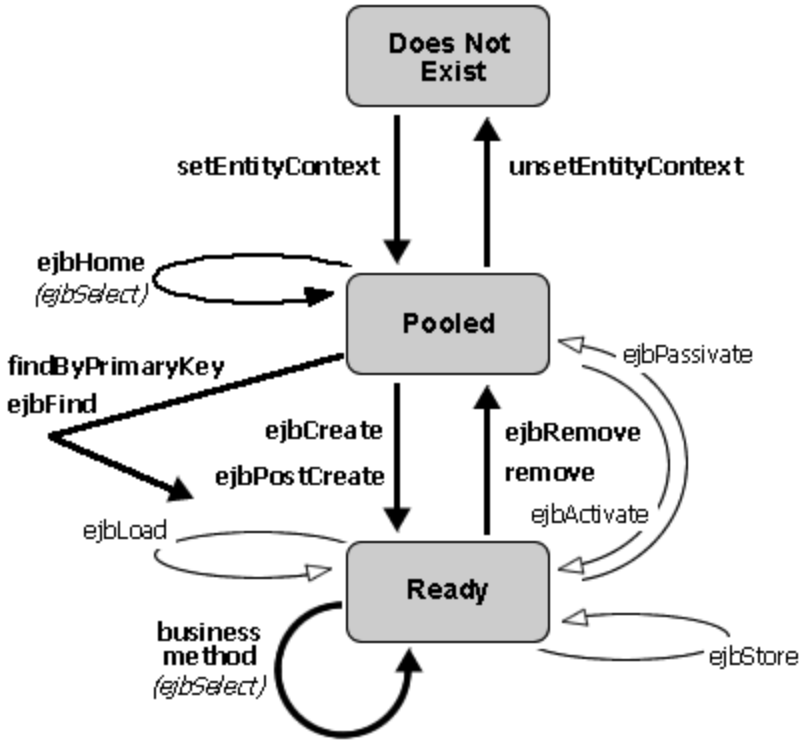


- client zavola metodu create → EJB kontejner vytvori novou instanci, vyvola se callback setSessionContext a pote callback ejbCreate (se stejnou signaturou jako klientsky create)
- stateful bean v ready stavu je svazana s prislusnym klientem v prubehu konverzace
- EJB kontejner muze uvest bean do pasivniho stavu (pro optimalizaci rizeni prostredku) → serializace na disk, vycisteni z pameti, po prichodu pozadavku od klienta (vyvolani nejake business metody) → opet aktivace
- klient muze zavolat remove → na EJB kontejneru se vyvola callback ejbRemove, pokud je treba implementovat nejakou logiku pro uklid beanu, implementuje se v tomto callbacku
- pri pouziti NRU cache se muze po urcitem timeoutu bean smazat - zavola se callback ejbRemove

39. Describe the lifecycle of a stateless session bean in EJB, that is, when instances of the bean are, or appear to be, created and destructed, both from the client and the server points of view.

- u klienta zde zadne systemove metody nejsou potreba
- pri vytvareni EJB kontejner vola callback setSessionContext, nasledne ejbCreate callback (podobne jako v minulem, zde muze byt impl. dodatecna logika pri vytvareni beanu)
- ejbCreate se vola jen jednou za zivotni cyklus, neni svazana s volanim create na klientovi (pri volani create se vzdy vrati jiz existujici instance v poolu, management je plne v kontrole EJB kontejneru)
- pokud je ready, muze obsluhovat klientske pozadavky
- EJB kontejner se muze rozhodnout omezit pocet instanci beanu a hodit bean k uklidu (vymazani), nacez se vola callback ejbRemove (volani remove z klienta invaliduje referenci na bean instanci, ale nepresunuje bean do “does not exist” stavu, spravu ma kompletne na starosti kontejner)

40. Describe the lifecycle of an entity bean in EJB, that is, when instances of the bean are, or appear to be, created and destructed, both from the client and the server points of view.



- pote co je nastartovan server, je vytvoreno nekolik bean instanci, ktere jsou ulozeny do poolu, tyto nejsou spjaty s zadnymi konkretnimi daty, pri vytvareni je zavolan callback setEntityContext
- u beans v poolu muzeme volat metody z home interface (*home interface specifies methods that allow the client to create, remove, and find objects of the same type. The home interface may also provide definitions for home business methods for entity beans. Home business methods are methods that are not specific to a particular bean instance.*), ktere mohou ponechat bean nespjatou s konkretnim objektem, ale mohou vyvolat metodu, ktera ji prevede do ready stavu (ejbCreate, findByPrimaryKey, find methods)
    - pokud klient vyvola create metodu na home interface daneho bean, vyvolaji se metody ejbCreate a ejbPostCreate, volajicimu je predana reference na bean a umoznuje mu tak volat na nem business metody
        - v ejbPostCreate ma uzivatel moznost nastavit reference na jine entity beans
    - findByPrimaryKey - nalezne existujici objekt podle klice a svaze jej s touto bean, vrati referenci uzivateli
    - find methods - nalezeni objektu podle daneho dotazu
- klient muze zavolat remove a zrusit tak svazani beanu s objektem, vola se callback ejbRemove a bean se vraci do pooled stavu

- prechod mezi pooled a ready stavem muze ovlivnit samotny EJB kontejner aktivaci a pasivaci beanu - pasivaci se uzivatel nedozvi, reference zustane validni a po aplikaci business metody se bean prevraci opet do ready stavu (pasivace je zde pouze ulozeni do DB, ne serializace na disk jak jinde)
    - callbacky ejbActivate, ejbPassivate
- EJB kontejner muze uplne smazat instanci beanu (pooled → does not exist), zavola se callback unsetEntityContext (napr pri vypnuti serveru)

41. Explain how the point-to-point message delivery model of JMS works.

- Existuje fronta, do které odesílatelé zprávy ukládají a příjemci je z ní vybírají.
- Pokud neexistuje příjemce, zprávy se ve frontě ukládají.

42. Explain how the publish-subscribe message delivery model of JMS works.

- Existuje kanál, který distribuuje zprávy připojeným příjemcům.
- Pokud není připojen žádný příjemce, zprávy se zahazují.
    - Vyjimka: pojmenovaná durable subscription, ktera zarucuje, ze zprávy se pro příjemce uchovávají.

43. Explain why all communication functions in MPI that deal with messages require a description of the message data type.

- Kvůli heterogennímu prostředí, ve kterém je možné MPI použít.
- MPI zaručuje přenositelnost zpráv.

44. Pick three of the functions provided by MPI for collective communication, listed below, and explain what they do:

```
int MPI_Barrier (MPI_Comm comm);
int MPI_Bcast (void *buffer, ..., int root, MPI_Comm comm);
int MPI_Gather (void *sendbuf, void *recvbuf, ..., int root, MPI_Comm comm);
int MPI_Scatter (void *sendbuf, void *recvbuf, ..., int root, MPI_Comm comm);
int MPI_Alltoall (void *sendbuf, void *recvbuf, ..., int root, MPI_Comm comm);
int MPI_Reduce (void *sendbuf, void *recvbuf, ..., MPI_Op op, int root, MPI_Comm comm);
int MPI_Scan (void *sendbuf, void *recvbuf, ..., MPI_Op op, int root, MPI_Comm comm);
```

```
int MPI_Barrier (MPI_Comm comm);
// Zablokuje se, dokud se všechny procesy nedostanou do této rutiny – „gruppen…
dostaveníčko“.
```

```
int MPI_Bcast (void *buffer, ..., int root, MPI_Comm comm);
// Zasílá zprávu od jednoho procesu s označením „root“ všem ostatním procesům ve skupině, včetně sebe sama.

int MPI_Gather (void *sendbuf, void *recvbuf, ..., int root, MPI_Comm comm);
// Sbírá (spojuje) dohromady data od více procesů.

int MPI_Scatter (void *sendbuf, void *recvbuf, ..., int root, MPI_Comm comm);
// Hromadu posílaných dat rozdělí mezi příjemce.

int MPI_Alltoall (void *sendbuf, void *recvbuf, ..., int root, MPI_Comm comm);
// Každý proces komunikuje s každým tak, že si od něj vyzobne tu část, která patří jemu.

int MPI_Reduce (void *sendbuf, void *recvbuf, ..., MPI_Op op, int root, MPI_Comm comm);
// Sesbírá data od více procesů a aplikuje na ně nějakou operaci a vyrobí tak jedny data.

int MPI_Scan (void *sendbuf, void *recvbuf, ..., MPI_Op op, int root, MPI_Comm comm);
// (Nějak) je určeno pořadí procesů a první dostane svoje data, druhý dostane data toho prvního, která jsou sloučena pomocí nějaké operace s daty druhého atd. až ten poslední dostane v podstatě výsledek operace Reduce.
```

45. The RMI technology uses standard Java interface definitions to describe the remotely accessible interfaces. Explain the limitations imposed on the interfaces by the RMI technology.

- Třída musí implementovat rozhraní Remote.
- Všechny vzdáleně přístupné metody musí mít povoleno vyvoláni výjimky RemoteException.
- Hodnotou mohou být předávány pouze serializovatelné typy.

46. Microsoft DCOM rozlišuje in process a out of process servery podle toho, zda běží v procesu klienta nebo ve vlastním procesu. Vysvětlete, proč Microsoft DCOM tyto druhy procesů nabízí.

- Komponenta jako DLL nemůže běžet sama o sobě, běží v procesu klienta = in-process, registrace probíhá v registrech počítače pod classID dané komponenty, registruje se zejména threading model.
- Komponenta jako EXE může běžet sama = out of process, místo registrace se volá CoInitializeEx s flagem threading modelu jako parametr.
- Kdyby se pro knihovny vytvářel speciální proces, zbytečně by se plýtvalo prostředky, např. nutnost přepínat kontext procesoru.
- Varianta mimo proces umožňuje snadno sdílet objekty mezi více procesy.

47. Microsoft DCOM uses a combination of keepalive pinging and reference counting to garbage collect unused objects. Evaluate this approach from the scalability point of view.

- Počítání referencí je dobře škálovatelné, dobrý a rychlý přístup k určení živých objektů, ale neřeší cyclic garbage (nejaka skupina objektu, ktera na sebe vzajemne ukazuje - uvnitr skupiny maji pozitivni reference counter, ale na skupinu se jiz neda dostat - najit takovou skupinu umi gc)
- Počítání referencí nemusí vždy také přesně sedět – klient např. nevolá release.
- Proto se pinguje jenou za dvě minuty server, po třech chybějících pingnutích je klient prohlášen za mrtvého a server tento objekt uvolní.
- Pokud by se jen pingovalo, vedlo by to k plýtvání pamětí, protože objekty by zůstávaly živé zbytečně dlouho.

48. The IUnknown interface in Microsoft DCOM has the following methods:

```
interface IUnknown
{
  HRESULT QueryInterface (REFIID iid, void **ppvObject);
  ULONG AddRef (void);
  ULONG Release (void);
};

Explain the semantics of the methods.
```

HRESULT QueryInterface (REFIID iid, void **ppvObject);
Metoda slouží k dotazu na existující rozhraní, které je identifikováno prvním parametrem, v němž je uloženo UUID požadovaného rozhraní. Pokud objekt rozhraní implementuje, je na něj vrácen ukazatel ve druhém parametru. Není-li implementováno, nabývá návratová hodnota funkce hodnotu E_NOINTERFACE a druhý parametr má neplatnou hodnotu.

ULONG AddRef (void);
Metoda inkrementuje počítadlo referencí, tedy počet odkazů ze všech připojených klientů. Metoda vrací aktuální počet referencí.

ULONG Release (void);
Metoda dekrementuje počítadlo referencí. Pokud počítadlo klesne touto operací na hodnotu nula, objekt se může sám odstranit z paměti. Návratová hodnota udává stav počítadla po dekrementaci.

49. The Chord middleware uses routing along a logical ring to deliver a message to a node with the address that is nearest to the message key. Describe the routing tables kept by the individual nodes and show the complexity of sending a message on a brief description of the routing algorithm.

- Logické adresy jsou přiděleny z určitého rozsahu, ve kterém se počítá modulo velikost rozsahu.
- Každý uzel lze adresovat jeho vlastní logickou adresou a také všemi logickými adresami, které jsou mezi jeho logickou adresou a nejbližší nižší přidělenou logickou adresou.
- Každý uzel si udržuje fyzické adresy (myšleno např. IP adresy) na uzly, které lze adresovat logickými adresami o 1, 2, 4, 8, 16… většími než je jeho vlastní logická adresa. (Tyto adresy se mohou samozřejmě překrývat.)
- Má-li uzel něco doručit na nějakou logickou adresu, tak to pošle na uzel, jehož fyzickou adresu zná a jehož logická adresa je nejbližší nižší cílové logické adrese.
- Při posílání se vlastně postupně „skládá“ binární reprezentace rozdílu logických adres uzlů ⇒ složitost log2N, kde N je maximální rozdíl logických adres.
- Příklad: Uzel 2 chce poslat zprávu na adresu 9. Vzdálenost je 9 − 2 = 7. Nejbližší známý vrchol je vzdálen 4, tedy má logickou adresu 2 + 4 = 6, což je vrchol 7. Zprávu dostane vrchol 7 a postupuje stejně: 9 − 7 = 2 ⇒ 7 + 2 = 9 ⇒ 11. Všiměte si, že vrchol 2 ví adresu vrcholu 11, ale nepoužil ji, protože pro něj platí pro adresy vzdálené 8 a více, tedy od adresy 10.

50. The CORBA middleware relies on a specialized interface definition language to describe interfaces of remotely accessible objects. Explain why this choice was made rather than relying on the interface definitions in an implementation language.

- Rozhraní může být zároveň implementováno/používáno ve více jazycích.
- IDL umožňuje i věci, které implementační jazyky nemají. Např. bezznaménkovost × Java

51. Even though CORBA IDL interfaces are mapped into classes both in C++ and in Java, the CORBA IDL interface attributes are not mapped into class attributes in C++ nor in Java. Explain why and outline how the attributes are mapped.

- Atributy jsou mapovány na dvojici funkcí, které umožňují čtení a psaní atributu.
- Pokud by byly mapovány na atributy, tak by proxy nevěděla, že se atribut změnil a nemohla by tak změnu přenést na server a na další klienty. (takto je to zabaleno do funkci a muze se to v nich zpracovat)

```
// IDL
attribute long accountNumber;
```

```
// C++
virtual CORBA::Long accountNumber(CORBA::Environment&); // getter
virtual void accountNumber(Long accountNumber, CORBA::Environment&); // setter
```

52. When mapping the CORBA IDL integer types into C++, the C++ integer types cannot be used directly because C++ does not define the precision of the integer types in as much detail as CORBA IDL. Explain how this problem is solved.

- Jsou mapovány na speciální typy, např. CORBA::Short, které jsou definovány v konkrétní implementaci systému CORBA pomocí typedef.

53. When mapping the CORBA IDL integer types into Java, not all the integer types can be mapped directly because Java does not provide all the precisions of the integer types available in CORBA IDL. Explain how this problem is solved.

- Neznaménkové typy jsou namapovány na odpovídající znaménkové varianty.
- V případě, že dojde k pokusu o přiřazení příliš velké hodnoty, je nahlášen „type mismatch“ a vyvolána vyjímka CORBA::DATA_CONVERSION.

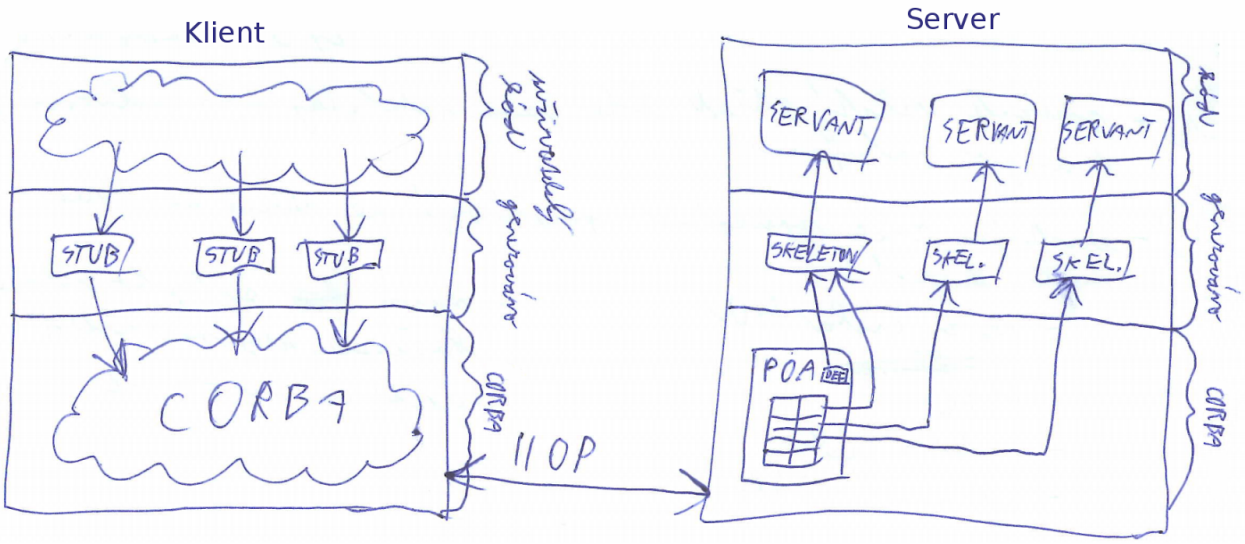
54. The CORBA IDL sequence type, which describes a variable length vector, has no direct counterpart in the basic C++ types. Explain how and why the type is mapped into C++.

- Mapování na speciální třídu v implementaci systému CORBA.
- Sekvence mohou mít maximální velikost, to vektor nemá, implementace je tedy příjemnější pro programátora.

55. The CORBA IDL union type, which describes a container that can hold any single item of a choice of items and preserves information on which item is held at a given time, has no direct counterpart in the basic C++ nor Java types. Choose either C++ or Java and explain how and why CORBA IDL union is mapped into that language.

- Mapují se na C++ třídy s funkcemi, které zpřístupňují union members (accessor functions), a s funkcemi, které mění union members (modifier functions).
- Používá se i diskriminator – funkce jenž určuje, která z položek unionu je ta aktivní.

56. Nakreslete strukturu CORBA aplikace. Vyznačte, kde jsou umístěné nebo se používají IIOP, POA, proxy, servant, skeleton a stub a vysvětlete funkci těchto prvků.



Tento obrázek byl součástí odpovědi za kterou byl udělen plný počet bodů.

- IIOP (Internet Inter-ORB protocol) – protokol pro přenos zpráv v systému CORBA
- POA – asociuje server s objekty, spravuje aktivní objekty, demultiplexuje příchozí požadavky na server a spolupracuje s IDL skeletonem na zavolání provedení správné operace volané od klienta na server.
- servant – server object implementation, instance objektu na serveru
- stub = proxy – provádí (un)marshalling na straně klienta, reprezentuje vzdálený objekt.
- skeleton – provádní (un)marshalling na straně serveru, je bránou mezi systémem CORBA a uživatelskými objekty.

57. Při podpoře paralelního zpracování více požadavků na serveru se může použít model thread pool, ve kterém má server množinu vláken připravených k obsluze požadavků. Vysvětlete přednosti tohoto modelu a doplňte zde uvedený popis modelu o podrobnosti, o které se tyto přednosti opírají.

- Množiny vláken mohou být připravené podle priority v tzv. lanes.
- Množina vláken si bere požadavky z fronty a obsluhuje je. Pokud nejsou požadavky, uspí se nebo skončí.
- Vhodné na real-time aplikace – je ušetřen overhead na vytváření a rušení vláken.

58. Portable Object Adapter ve standardu CORBA dovoluje definovat default servant, kterému jsou doručeny všechny požadavky, pro které nebyl nalezen jiný servant. Vysvětlete, v jakých situacích je vhodné default servant použít.

- Když více objektů má stejné rozhraní a to tak může být implementováno jen jedním servantem.

59. Portable Object Adapter ve standardu CORBA dovoluje definovat servant activator, který je požádán o vytvoření servantu v situaci, kdy pro příchozí požadavek nebyl žádný servant nalezen. Vysvětlete, v jakých situacích je vhodné servant activator použít.

- implementace zpožděného načítání objektů
- možnosti serveru jsou rozsáhlé, ale nejsou potřeba všechny najednou
- urychlení startu

60. Portable Object Adapter ve standardu CORBA dovoluje nastavit pomocí servant retention policy zda se mají v tabulce active object map evidovat všechny aktivní servanty. Vysvětlete, v jakých situacích je vhodné upustit od evidence aktivních servantů v active object map.

- spousta serveru, ktere musi poskytovat pristup k velkemu mnozstvi CORBA objektu zaroven maji nastavene servant retention policy na NON_RETAIN (active object map is absent), aby omezily pocet servantu, ktere musi byt najednou v pameti - aktivovani objektu pouze na pozadavek

61. Protokol GIOP ve standardu CORBA umožňuje pomocí zprávy LOCATION FORWARD informovat klienta o tom, že server se nachází jinde než se klient domníval. Vysvětlete, jak je tento mechanismus možné použít k implementaci persistentních objektů, jejichž reference zůstávají v platnosti i při ukončení serveru.

- existuje POA mediator (centralni autorita), ktera spravuje pozadavky v pripade padu serveru, umoznuje migraci na jiny server, sleduje aktivni persistentni objekty apod.
- pri padu serveru posle mediator klientovi informaci (transparentni pro uzivatelsky kod) o preposlani jeho pozadavku, ten pozada o novou lokaci serveru a nasledujici pozadavky jsou posilany jiz primo na tento server, dokud nedojde opet k presmerovani/padu serveru

62. Standard CORBA definuje službu Naming sloužící k registraci a vyhledávání serverů. Základní operace této služby jsou bind a resolve:

```
void bind (in Name n, in Object obj) raises (AlreadyBound ...);
Object resolve (in Name n) raises (NotFound ...);
```

Popište sémantiku těchto operací.

- bind - zaregistruje objekt pod zadanym jmenem, muze vyhodit vyjimku, ze je na dane jmeno nabindovany jiz jiny objekt
- resolve - vrati objekt, ktery byl drive zaregistrovan pod predanym jmenem, muze vyhodit vyjimku, ze dany binding neexistuje nebo ze typ bindingu je nespravny pro vykonavanou operaci

63. Standard CORBA definuje službu Trading sloužící k registraci a vyhledávání serverů. Základní operace této služby jsou export a query:

```
OfferId export (
       in Object reference,
       in ServiceTypeName type,
       in PropertySeq properties);

void query (
       in ServiceTypeName type,
       in Constraint constr,
       in Preference pref,
       ...
       out OfferSeq offers,
       out OfferIterator offer_itr,
       ...);
```

Popište sémantiku těchto operací.

- export vytvori nabidku sluzby - inzeruje objekt - argumenty: reference na dany objekt, typ nabidky a vlastnosti objektu
- query slouzi k nacteni nabidek/inzeratu z Trading sluzby, ktere splnuji prilozena kriteria

64. Standard CORBA nabízí možnost neblokujícího volání pomocí callbacku. Následující fragment CORBA IDL uvádí příklad blokující operace a k ní vygenerované rozhraní pro neblokující volání pomocí callbacku:

```
// Příklad blokující operace
void example_operation (in string anInArgument, out double anOutArgument);

// Rozhraní pro neblokující volání
void sendc_example_operation (in AMI_ExampleHandler ami_handler, in string anInArgument);

interface AMI_ExampleHandler
{
       void example_operation (in double anOutArgument);
}
```

Vysvětlete na tomto příkladu fungování neblokujícího volání pomocí callbacku.

- Klient zavolá sendc_example_operation, které předá odkaz na objekt, který implementuje rozhraní AMI_ExampleHandler.
    - Tato funkce jen předá odkaz na objekt a parametry a okamžitě se vrátí.
- Odpovědi už zpracovává objekt, který byl předán jako handler.

65. Standard CORBA nabízí možnost neblokujícího volání pomocí pollingu. Následující fragment CORBA IDL uvádí příklad blokující operace a k ní vygenerované rozhraní pro neblokující volání pomocí pollingu:

```
// Příklad blokující operace
void example_operation (in string anInArgument, out double anOutArgument);

// Rozhraní pro neblokující volání
AMI_ExamplePoller sendp_example_operation (in string anInArgument);

valuetype AMI_StockManagerPoller
{
	void example_operation (in unsigned long timeout, out double anOutArgument);
}
```

Vysvětlete na tomto příkladu fungování neblokujícího volání pomocí pollingu.

- sendp_example_operation vráti objekt AMI_ExamplePoller. Tento objekt slouží k aktivnímu čekání na výsledek.
- Klient může na objektu zavolat metodu, které předá informaci o tom, jak dlouho chce čekat a buď je do té doby vrácen výsledek, nebo se funkce vrátí s chybou.

66. The ProActive middleware uses futures to pass results of method invocations on active objects. Explain why.

- Volání serveru se okamžitě vrátí a vrátí future.
- future je speciální objekt, do kterého bude asynchronně uložen výsledek. Pokud se program na výsledek uložený ve future zeptá dříve než je k dispozici, volání se zablokuje a čeká se, než přijde výsledek.
- Výhoda tohoto postupu je zřejmá: Po odeslání dotazu na server lze provádět operace, které výsledek nepotřebují a to při zachování nízké složitosti kódu.

67. The basic operations of JavaSpace are write, read and take:

```
interface JavaSpace
```

```
{
	Lease write (Entry e, Transaction tx, long lease);
	Entry read (Entry template, Transaction tx, long timeout);
	Entry take (Entry template, Transaction tx, long timeout);
	...
}
```

Describe what these operations do.

- write – Places a new Entry in the space.
- read – Returns a copy of an Entry matching a particular template. Blocking, until one exists.
- take – Removes an Entry matching a particular template from the space and returns it to the caller. Blocks until one exists.