Souhrnný snímek

- Orientace na služby a komponenty – klíčové paradigma současného softwaru
  - Pohled pragmatika
  - Větší důraz na použití v malých a středních podnicích a zčásti ve státní správě než na to co dodávají obří výrobci,
  - Založeno na praktických zkušenostech

Komponentová architektura

- Základní komponenty (aplikační)
  - ? Služby?
- Konektory pro jejich propojování,
  - DLL moduly,
  - API moduly, jejich účel je zajistit propojování
  - Samostatné komponenty, to je nejsilnější, zvláště, jsou-li komponenty komponované jako služby
- I malé změny konektoru a pravidel pro jeho používání obvykle silně ovlivňuje chování systému a to, co systém umožňuje

Orientace na služby – klíčové paradigma současného softwaru

Servisně orientovaný systém má specifickou servisně orientovanou architekturu (SOA), varianta komponentového pohledu,
asynchronnost + silná autonomie,

hrubozrnné rozhraní, konektory jen jako middleware

Dávno známé, přesto nové příležitosti (web)

Různé typy funkcí komponent

- Peers:
  - z hlediska technických vlastností rozhraní jsou komponenty rovnocenné
- Vrstvy komponent z hlediska uživatele
  - Klient, Logika systému, Datová vrstva
- Komponenty výkonné a ty, co zajišťují spolupráci.
  - Konektor jako služba(peer), konektor jako API, konektor jako dll modul

Komponentová orientace , především servisně orientovaná je klíčová filosofie

- Umožňuje agilitu i při vývoji velmi velkých systémů (vývoj pomocí změn konektorů a pravidel jejich využívání)
- Snížení pracnosti vývoje
  - Vyvinout N malých autonomních SW artefaktů délky *x* méně pracné než vývoj jednoho délky N*x, zrychlení je* $N^{-1/8}$, i když vše píši znovu,
    - Jsou-li komponenty autonomní lze je snadno koupitlze snadno znovu použít koupit nebo převzít, **to je hlavní přínos!!!!!**
  - *Je možný inkrementální vývoj (je dříve vidět, jak to bude fungovat, snadné prototypování,*
  - Snazší údržba a inkrementální výměna komponent

Servisně orientovaná architektura

- Virtuální p2p síť volně vázaných komponent
- Komponenty spolupracují asynchronním předáváním zpráv
  - Mohou se ale použít jiné způsoby komunikace
  - Základní styl komunikace je asynchronní, sekvenční je nadstavba
- SOA je pro některé výrobce značka pro jejich výrobky a jejich definice SOA a definice podle jimi inspirovaných SOA je proto užší

Pragmatický přístup

Snažíme se o co nejjednodušší integrací různorodých komponent

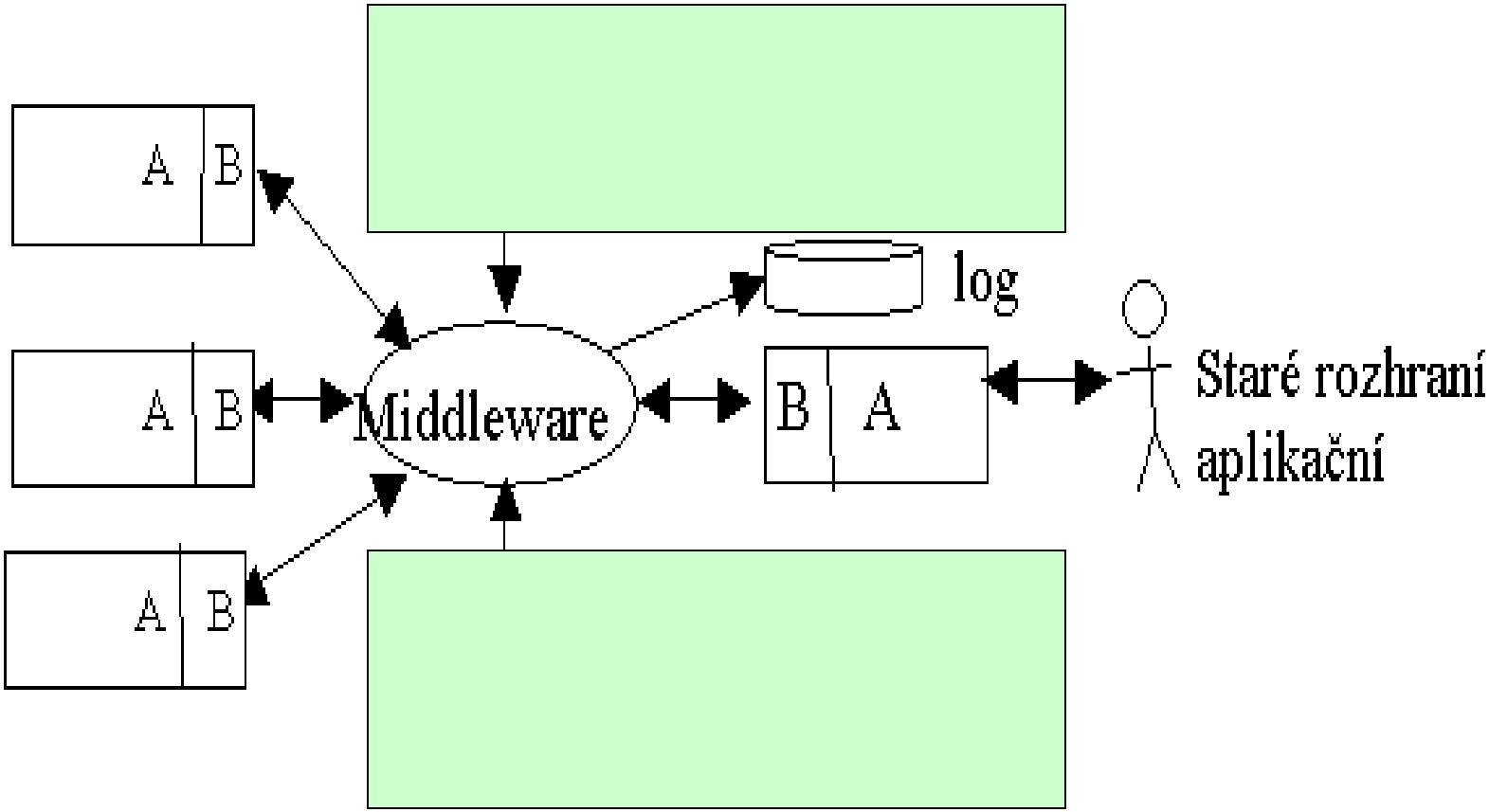
Jde tedy spíše o bazar či rychle rostoucí město než o katedrálu budovanou po dlouhou dobu podle striktně stanovených pravidel a plánů. V informatice se vše mění tak rychle, že je problematické neměnit plány a pravidla

V bazaru musí být možné nakupovat,

Někdy se se službou pojí řada vlastností

- SLA – service level agreement
  - jak se služba používá, popis rozhraní, na webu jazyk WSDL
  - Pravidla dostupnosti

Struktura jednoduché SOA



A – aplikační služba, B – její rozhraní (primární brána) UR je uživatelské rozhraní, např. portál

Staré aplikační rozhraní je u komplexnějších služeb nutností (viz IS úřadů), usnadňuje to podstatně i vývoj.

Preferujem vývoj zdola

- Pro malé firmy je vhodnější vývoj integrací
  - Lze snáze použít to, co už je
  - Lze snáze zvládnout mentálně a finančné
  - Inkrementálnost snazší
  - Open group a OSIMM

Varování

Budeme občas říkat zřejmé věci, kterých si však ke své škodě nevšímáme. Opomíjení samozřejmostí je hlavní problém IT všeobecně a softwaru zvláště.

Varování, obecně platná zkušenost

Čím samozřejmější věc zanedbáme tím horších následků se dočkáme

Platí to především pro informační systémy

Varování 2

Problémem je, že servisní orientace je zdánlivě samozřejmá. Paradoxně ji právě proto nevěnujeme dostatek pozornosti a dokonce se domníváme, že to není nic nového, a nejsme ji proto schopni-ochotni používat, ačkoliv se stává vedoucí filosofií současného softwaru. Ten se stává síťový z logického hlediska a reálnému světu podobný z uživatelského hlediska

Cloud computing a gridovské systémy zvyšují použitelnost a výhodnost SOA

SOA berou velké SW firmy jako značku svých výrobků, používají často normy OASIS skupiny

Protestují, když se SOA používá v poněkud jiném smyslu

Mnohé je známo po dlouhou dobu, celek a principy je však obtížné uplatnit, Je pro to třeba:

Změnit marketing (jak od předsudků osvobodit, jak změnit pravidla spolupráce), změnit management
Použít existující komponenty (legacy sys.)
Změnit metody specifikace požadavků
Zajistit dostupnost operací , jako je prototypování, decentralizace, sourcing,…
Umožnit nové postupy na straně uživatelů i výrobců SW (SaaS - SW jako služba, decentralizace, otevřenost), cloudy
Překročit hranice OO

Servisně orientovaná architektura v našem smyslu

- Virtuální síť p2p síť (velkých) volně vázaných komponent, spolupráce komponent je podobná spolupráci služeb v reálném světě,
  - Komponenty jsou zpravidla permanentně aktivní procesy, asynchronnost komunikace
  - Komponenty mohou být i dávkové systémy, např. Excel či OLAP
  - Existují i jiné komponentové přístupy, servisní je asi v informačních systémech nejužitečnější

Hlavní oblasti aplikace servisně orientované architektury

Techniky využívané v SOA se značně liší podle oblastí nasazení a způsobu dodávky, typické příklady:

1. Řízení technologií, hlavně varianta soft real-time
2. Systémy, kdy je nutná agilita při vývoji i používání, sourcing, a znovupoužívání systémů
   - malé a střední podniky, e-government, použití ve výzkumných projektech, obvykle velké komponenty a uživatelsky orientovaná komunikace
   - **pravděpodobně hlavní směr příštího rozvoje SOA, datové schránky**
3. Systémy pro velké podniky na klíč
   - Silně standardizovaná řešení, OASIS norma SOA reference architecture, v menších IS Open Group SOA reference architecture
   - Spousty (malých) služeb, komunikace pomocí vzdáleného volání procedur

Náš přístup – varianty implementace

- P2p síť autonomních komponent s rozhraním podobným „rozhraní“ služeb reálného světa
  - Základní je asynchronní komunikace a předávání dokumentů (srv. datové schránky)
  - Podobnost se zvyky reálného světa může být vzdálenější nebo užší
- Ne vždy se SOA takto chápe, požadují se další vlastnosti resp. dodržování norem

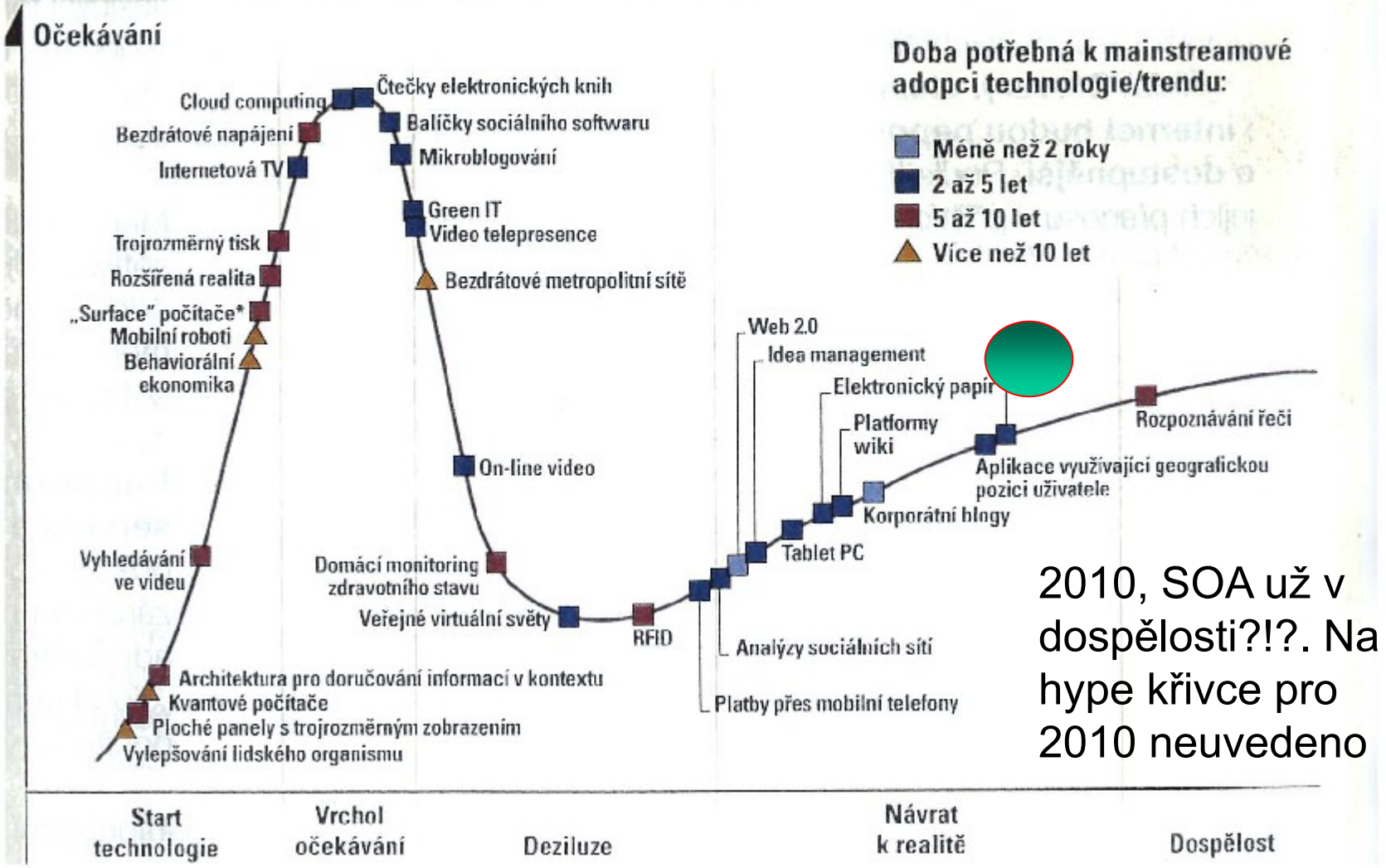
SOA se na považuje za klíčovou a již vyzrálou technologii

Viz polohu na SOA hype křivce v r. 2009 podle analytiků Gartner Group

Mnoho úspěšných použití (viz např. Progress Software, IBM, dokonce SAP)

Značné investice do SOA výrobců i uživatelů

Hype křivka rozvíjejících se trendů a technologií



2010, SOA už v dospělosti?!?. Na hype křivce pro 2010 neuvedeno

- Existence služeb není možná bez nějaké varianty (servisně orientované) architektury a middlewaru

Mnoho věcí je třeba doladit:

SOA pro malé podniky, tomu se budeme hlavně věnovat

vazby na cloud

využití výsledků výzkumu různých variant komponentových architektur

To, že je SOA zvládnuto, asi platí jen pro jistý typ SOA a pro jistá použití daného typu SOA

- SOA je v jistém smyslu bussword a výrobní značka
  - Zahrnuje dosti širokou třídu technologií a přístupů,
  - přivlastněno výrobci
- Typ SOA, který prosazují výrobci SW, je vhodný spíše pro velké a mohovité uživatele, usurpovali si i zkratku SOA
  - mají podobnou kulturu jako výrobci SOA (příklad SOA Reference Architecture by OASIS a Open Systém Integration Maturity Model by Open Group, ten je trochu blíže SME)
  - mají na to peníze i lidi
  - Výrobci prosazují standardy, které jsou vhodné pro jejich výrobky

To, že je SOA zvládnuto, asi platí jen pro jistý typ SOA a pro jistá použití daného typu SOA

- Existují samozřejmě rúzné důvody selhání SOA, např. ty, které jsou obecně platné pro každé SW projekty
  - Nevím-li pořádně k čemu a proč něco pořizuji, nemohu s tím být spokojen nezávisle na tom, jak je to uděláno
  - Klíčové byznys otázky za mne SOA nevyřeší,
    - může mi v tom ale významně pomoci
- Problém, že jde o specifické paradigma
  - Objektový pohled může být na závadu

Servisně orientovaná architektura v intuitivním smyslu

- Virtuální p2p síť (velkých) volně spolupracujících komponent,
  - Z hlediska uživatelů je výhodné, aby se komponenty v informačních systémech chovaly podobně jako služby reálného světa
    - Mnohdy je žádoucí uživatelsky orientované rozhraní služeb (rozhraní podobné jako u reálných služeb, - výměna dokumentů), je to obvykle výhodné i pro cloudy, musí se to ještě prozkoumat
    - Takové rozhraní je založeno na jazycích používaných ve znalostních doménách uživatele
  - Komponenty je proto výhodné implementovat jako permanentně aktivní procesy, chovají se jako služby, jsou SW služby

Servisně orientovaná architektura v intuitivním smyslu

– V tom není úplná shoda a to způsobuje problémy
  – Uživatelská orientace rozhraní služeb omezuje standardizaci a aplikaci obratů typických pro objektový vývoj
  – Přehnaná standardizace může být pro takový přístup pastí (antivzor standardizační paralýza)
  –Zesiluje chápání SOA jako procesu integrace (Open Group)

Sítě dávkových služeb jako předchůdce SOA

- Sítě dávkových služeb tak, jak je zobrazovaly diagramy toků dat, byly v lecčems předobrazem SOA.
  - Autonomie komponent, komunikace přes datová úložiště
- Známé výhody dávkových systémů
  - Modifikovatelnost, levný vývoj
  - Stabilita (viz zkušenosti s Y2K a systémy psanými v Cobolu)
  - Škálovatelnost
  - Znovupoužitelnost
- Je často nutné dávkové aplikace integrovat do SOA
  - K tomu stačí, aby dávkové aplikace se zbytkem systému komunikovaly přes datové úložiště a obvykle pomocí dávkového (bulk) přenosu dat

Řízení technologií

- Nejstarší oblast používání principů SOA, především ve variantě soft real-time; od sedmdesátých let
  - Soft real time (odpovědět včas, vyjímečné opoždění není v zásadě chyba, jen nepříjemnost, př. Počítač s terminály)
  - OS a inteligentní periferie
  - U hard real-time (odpověď vždy v termínu) jsou dodatečné nároky a použití SOA v nich nebývá vždy možné, pokud kritickou část neřeším v autonomních částech HW a SW
- Rozhraní služeb nemusí být v technologiíchuživatelsky orientováno
- Proprietální řešení komunikace komponent není výjimkou
- Obvykle celkem malý počet služeb
- Komponenty nejsou zpravidla rozsáhlé
- Agilita použití je omezena
  - Běžný uživatel nemá skoro žádnou možnost principy práce technologie měnit a ani tomu nerozumí
  - I ti co mají povolenu agilitu musí být odborníci a i ti pak mají zpravidla jen omezené možnosti měnit chování systému
  - Nelze aplikovat postup pokus-omyl, např. tak jak je obvyklé u agilních form vývoje

Systémy, kdy je nutná agilita, sourcing, a znovupoužití - konfederace

- Typické pro malé a střední podniky a některá řešení e-government, např. propojování úřadů
- Je nutné použít legacy a produkty třetích stran
  - bývají velké, jejich změny jsou drahé a nejsou na ně peníze,
  - existují legislativní omezení,
  - rozhraní má být srozumitelné pro uživatele (neškolené v IT),
  - požaduje se snadnost outsourcingu, …
- Rozhraní musí umožňovat agilní změny byznys procesů používajících služby v SOA (rychlá reakce na změny)
- Tlak na outsourcing resp. využívání cloudů
- Využívání existujícího know how je nutností
  - Realistické nároky na zaškolování koncových uživatelů
  - Snazší záběh a hladší provoz
  - Nízké náklady a možnost postupné i agilní implementace

Systémy, kdy je nutná agilita, sourcing, a znovupoužívání: vlastnosti řešení

- Služby, které jsou partnery v komunikaci, se nemusí obvykle vyhledávat, protože jsou známi a nemusí se měnit
  - Rozhraní služeb lze dohadovat ad hoc, partneři v komunikaci jsou „prověření“ , normy výhodou – ne nutností
  - Služby nemusí být webové služby ve smyslu norem
    - Pokud se používají webové služby, nemusí nutně vyhovovat všem normám, jak je prosazuje OASIS w3c a jiní. Webová služba je pak intuitivně spíše libovolná funkcionalita dosažitelná přes web.
    - Složitost norem může implikovat složitost používání a také nežádoucí tlak na změnu osvědčených postupů a použití jemnozrnných služeb což zhoršuje vlastnosti systému

Systémy, kdy je nutná agilita, sourcing, a znovupoužití: vlastnosti řešení 2

- Služby poskytující uživatelské funkce jsou většinou velké, často legacy, často je nutné omezené využívání norem, do chodu služeb mohou zasahovat uživatelé on-line,
  - Nejen klasická výměna zpráv a web,
    - Využívání datových úložišť a úložišť zpráv, probereme později
  - Zprávy a obvykle i služby mají být uživatelsky orientované a sémanticky bohaté a tedy hrubozrnné (coarse grained),
  - Někdy je nutné v komunikaci používat datová úložiště (pro integraci dávkových systémů, a pro implementaci sofistikovaných variant komunikace), dokonce i datové proudy
  - Lze u databázově orientovaných sytémů použít i uložené procedury

Systémy, kdy je nutná agilita, sourcing, a znovupoužívání (malé a střední podniky, e-government …)

- Je nutné použít legacy (jsou velké, změna je drahá a nejsou na ni peníze, je nutné vyhovět legislativě, rozhraní služeb má být srozumitelné pro neškolené uživatele, snadnost outsourcingu, použití toho, co funguje, …)
- Rozhraní musí umožňovat-usnadňovat agilní změny procesů
- Je žádoucí využívat existující know-how

Systémy pro velké podniky na klíč, aliance (partner se zpravidla vyhledává)

Hlavní výhoda pro uživatele

– Malá pravděpodobnost krachu projektu v důsledku volby osvědčeného dodavatele, je to i alibi pro management uživatele

Princip řešení a jeho výhody pro dodavatele

- **Mnoho malých služeb a použití standardů, aby se dosáhlo univerzality**

Výhody malých služeb z hlediska dodavatele

- Zákazník se obvykle stane závislý na dodavateli
  - antivzor *vendor lock in* a do jisté míry i *fine-grained services*
- Dají se využít (zlo)zvyky a dovednosti objektového vývoje
- V principu velmi silné, ale pracné na vývoj i údržbu (viz asembler)
  - Další důvod závislosti na dodavateli
- Velmi často založené na webových službách ve smyslu OASIS a w3c (antipattern *SOA=Webservices*) a standardech (*Standardization paralysis*

Systémy pro velké podniky na klíč

(aliance, partner se zpravidla vyhledává),

Hlavní nevýhody pro uživatele

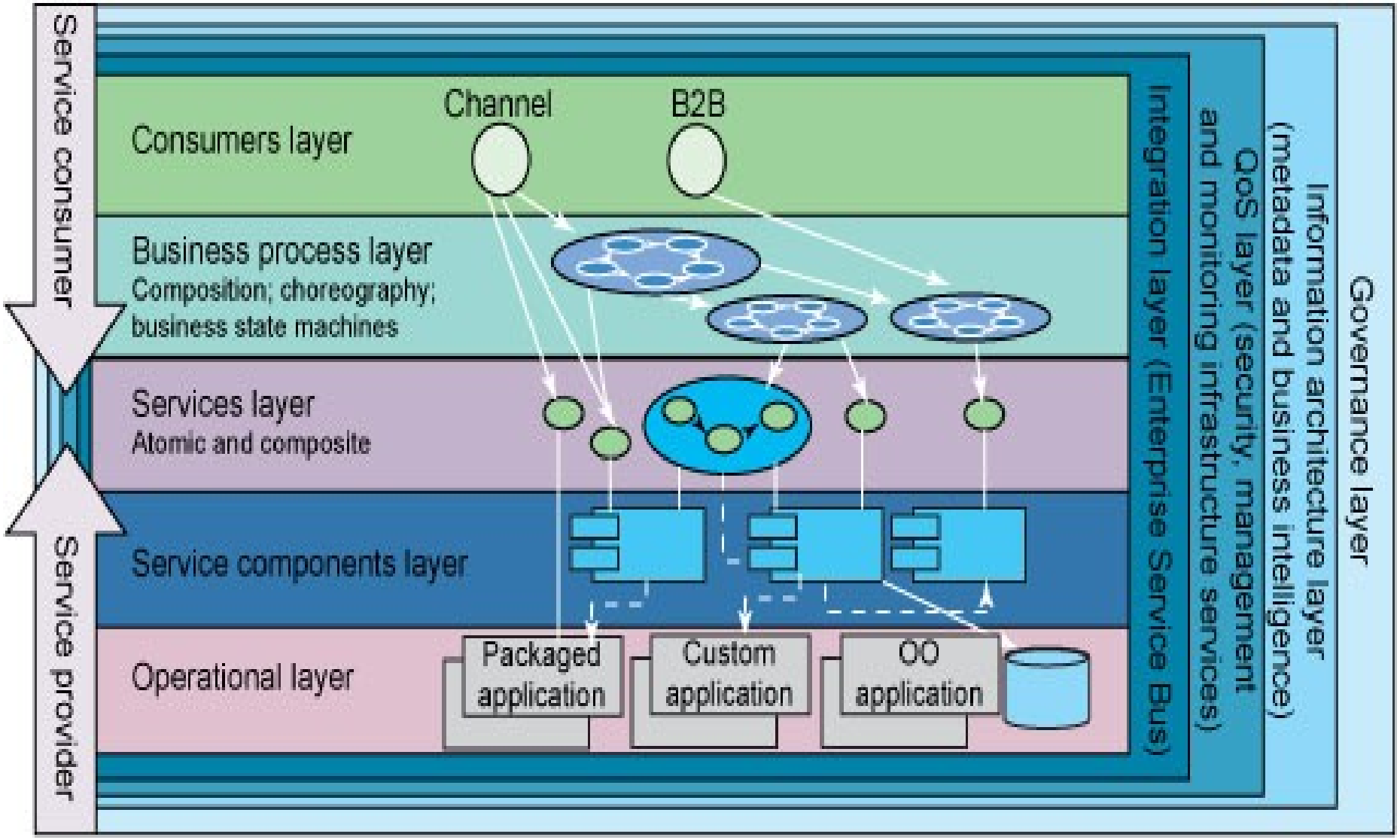
- Rozhraní služeb lze modifikovat jen za drahé peníze a za příliš dlouho a i pak bývá nepříliš uživatelsky příjemné
  - Problémy s agilitou byznys procesů
- Od velkých pro velké:
  - Pro malé podniky nevhodné nejen pro nákladnost a závislost na dodavateli, ale také proto, že neodpovídá jejich kultuře
  - V e-governmentu ten problém, že se zpravidla musí propojovat velké celky (IS celých úřadů)
    - Technicky lze provést pomocí datových schránek, ale pak je to konfederace
  - Prostor pro různé standardy, které jsou ale natolik komplikované, že je menší podniky nemohou samy použít, a drahé, rychle zastarávají
    - Musí se spolehnout na dodavatele a jejich knihovny a jiná jejich řešení
- Tento typ SOA se z marketingových důvodů ztotožňuje se SOA vůbec a jsou na něj časté stížnosti, hlavně od malých firem

SOA pro velké podniky podle standardů OASIS

- According to the OASIS SOA-RM specification, SOA is a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains. It provides a uniform means to offer, discover, interact with and use capabilities to produce desired effects consistent with measurable preconditions and expectations. The SOA-RM specification bases its definition of SOA around the concept of "needs and capabilities", where SOA provides a mechanism for matching needs of service consumers with capabilities provided by service providers.
- Pozor, SOA umožňuje vývoj velkých sytémů

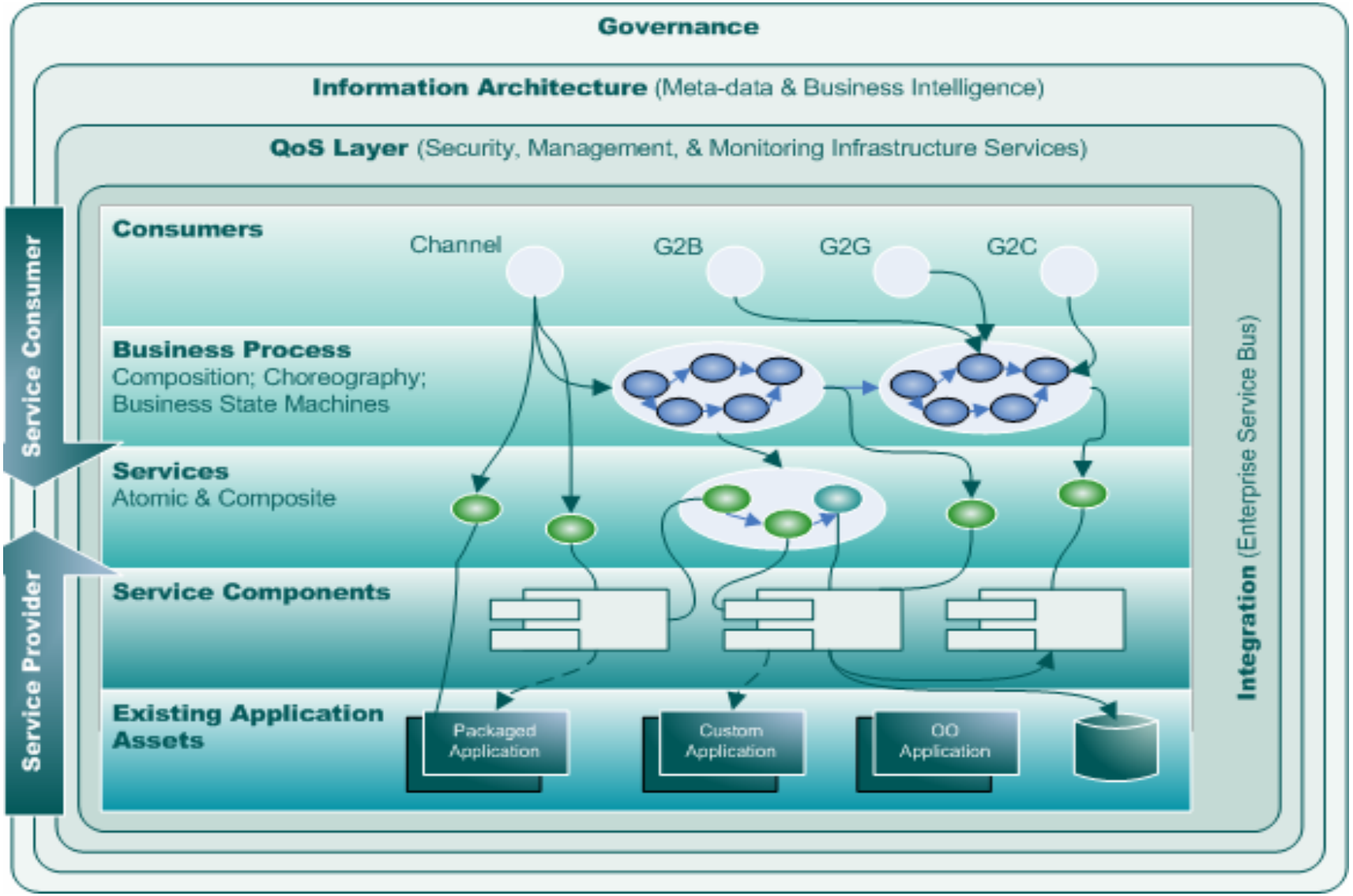
Řešení podle IBM

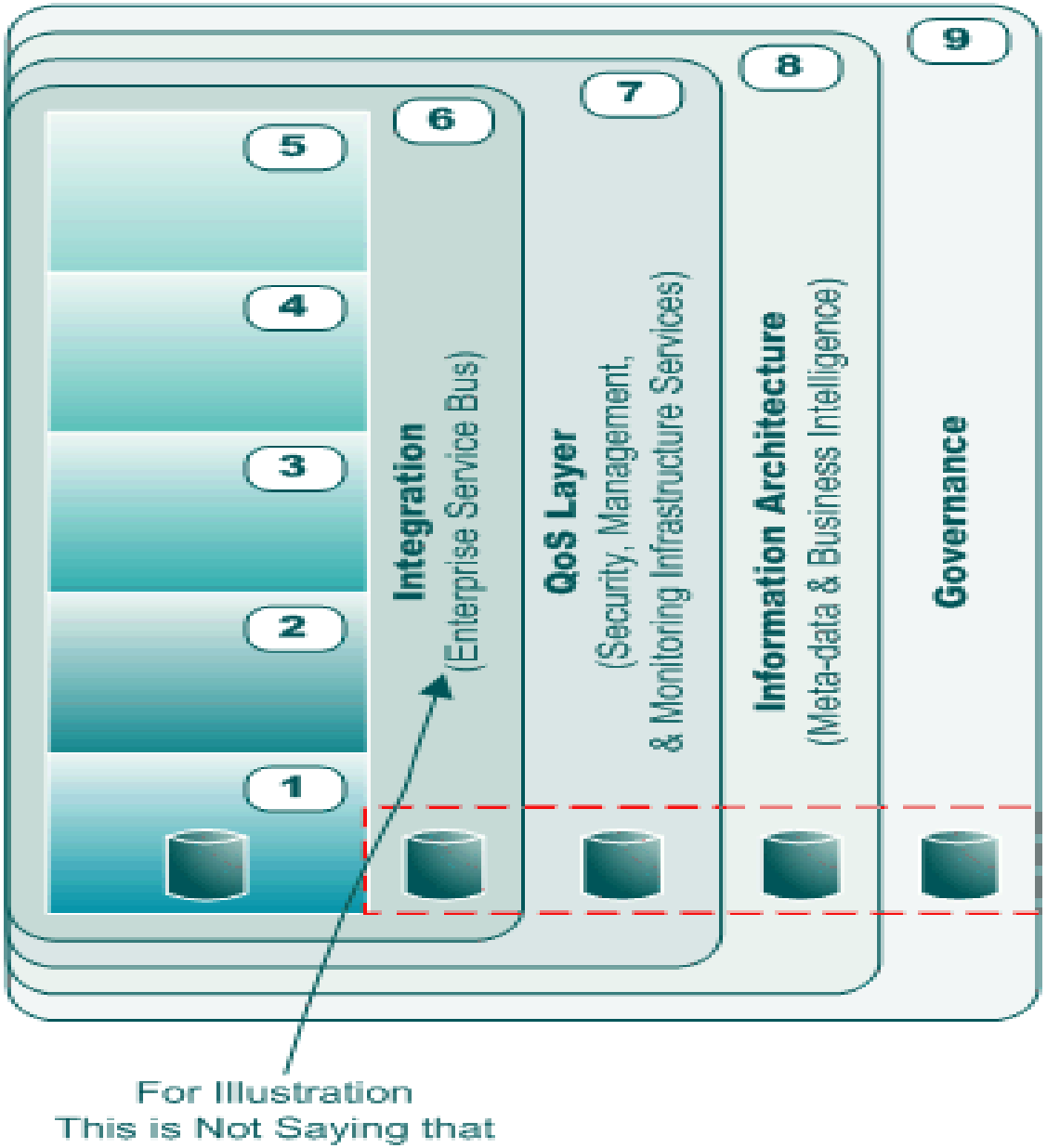
Figure 2. Layers of the SOA reference architecture: Solution stack view

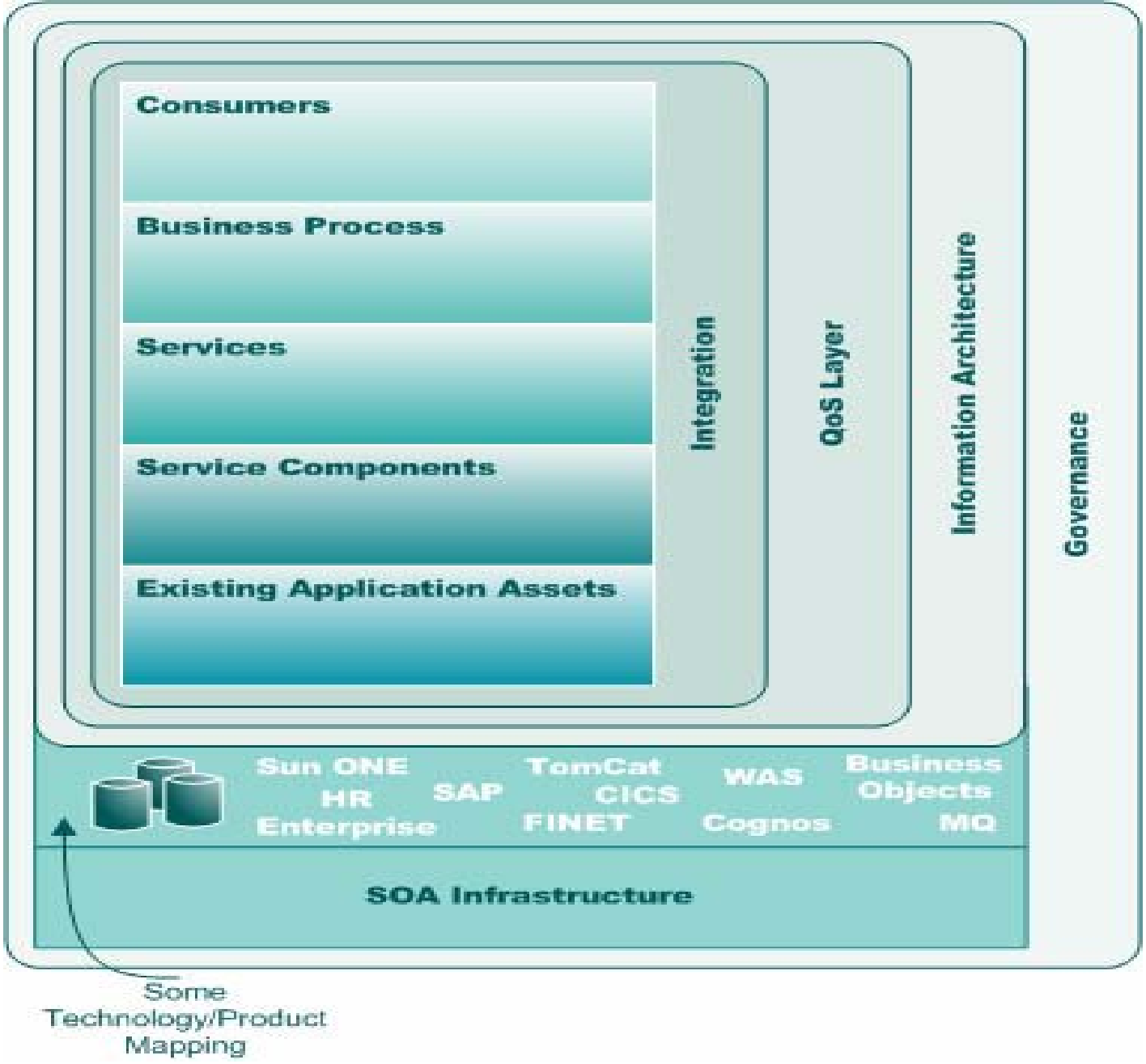


Návrh podle OASIS

http://dts.utah.gov/techresearch/techarchitecture/resources/soara081407.pdf




1
2
3
4
5
6
7
8
9
Integration
(Enterprise Service Bus)
QoS Layer
(Security, Management,
& Monitoring Infrastructure Services)
Information Architecture
(Meta-data & Business Intelligence)
Governance
For Illustration
This is Not Saying that


Consumers
Business Process
Services
Service Components
Existing Application Assets
Integration
QoS Layer
Information Architecture
Governance
Sun ONE
HR
Enterprise
SAP
TomCat
CICS
FINET
WAS
Cognos
Business
Objects
MQ
SOA Infrastructure
Some
Technology/Product
Mapping

Odkazy

Arsanjani, Ali, and Jorge Diaz, *SOA Reference Architecture: Concepts and Usage*, IBM: SOA Center of Excellence, July 25, 2007.

Bennett, Stephen, *Best Practices and Patterns from the field for SOA Governance*, Open Group Architecture Practitioners Conference, July 23-25, 2007, Austin Texas.

Bringing SOA Value Patterns to Life, Oracle White Paper, June 2006.

Dico, Awel, *Addressing Enterprise Business Needs through SOA and TOGAF*, Open Group Architecture Practitioners Conference, July 23-25, 2007, Austin Texas.

Gejnevall, Matt, *How EA and SOA Connect*, Open Group Architecture Practitioners Conference, July 23-25, 2007, Austin Texas.

High, Rob, Jr. et al, *IBM's SOA Foundation: An Architectural Introduction and Overview.* Version 1.0, November, 2005. http://download.boulder.ibm.com/ibmdl/pub/software/dw/webservices/ws-soa-whitepaper.pdf

Keen, Martin et al, *Patterns: SOA Foundation -Business Process Management Scenario,* Redbook, http://www.redbooks.ibm.com/redbooks/pdfs/sg247234.pdf

Odkazy

———————————, *Patterns: SOA with an Enterprise Service Bus,* Redbook, http://www.redbooks.ibm.com/redbooks/pdfs/sg246494.pdhttp://www-128.ibm.com/developerworks/library/ws-model7

Reference Model for Service Oriented Architecture 1.0, Committee Specification 1, 2 August 2006, Copyright © OASIS Open 2005-2006. All Rights Reserved.

Sides, Jim, *Applying Patterns, Platform Intelligence and Products to ESB Deployments*, IBM, Open Group Architecture Practitioners Conference, July 23-25, 2007, Austin Texas.

Sudarsan, Sridhar, *Adopting SOA - Aligning the business and IT*, IBM, Open Group Architecture Practitioners Conference, July 23-25, 2007, Austin Texas.

The Actionable SOA: Patterns in SOA, Applied Technology Solutions, Open Group Architecture Practitioners Conference, July 23-25, 2007, Austin Texas.

TOGAF 2006 Edition (Incorporating 8.1.1), The Open Group, August 2006.

Utah GovPay: The Official Payment Solution for Utah Government (Terchnical Manual), Utah Interactive: Salt Lake City, 2006.

Příklad uplatnění

Systém sběru recepisů a kontroly výdeje léků, primárně těch zneužitelných pro výrobu pervitinu

Předpokládaná úspora miliarda až dvě ročně a zabránění škodám na zdraví, později i zefektivnění léčby

ÚOOÚ fungující systém zakázal používat

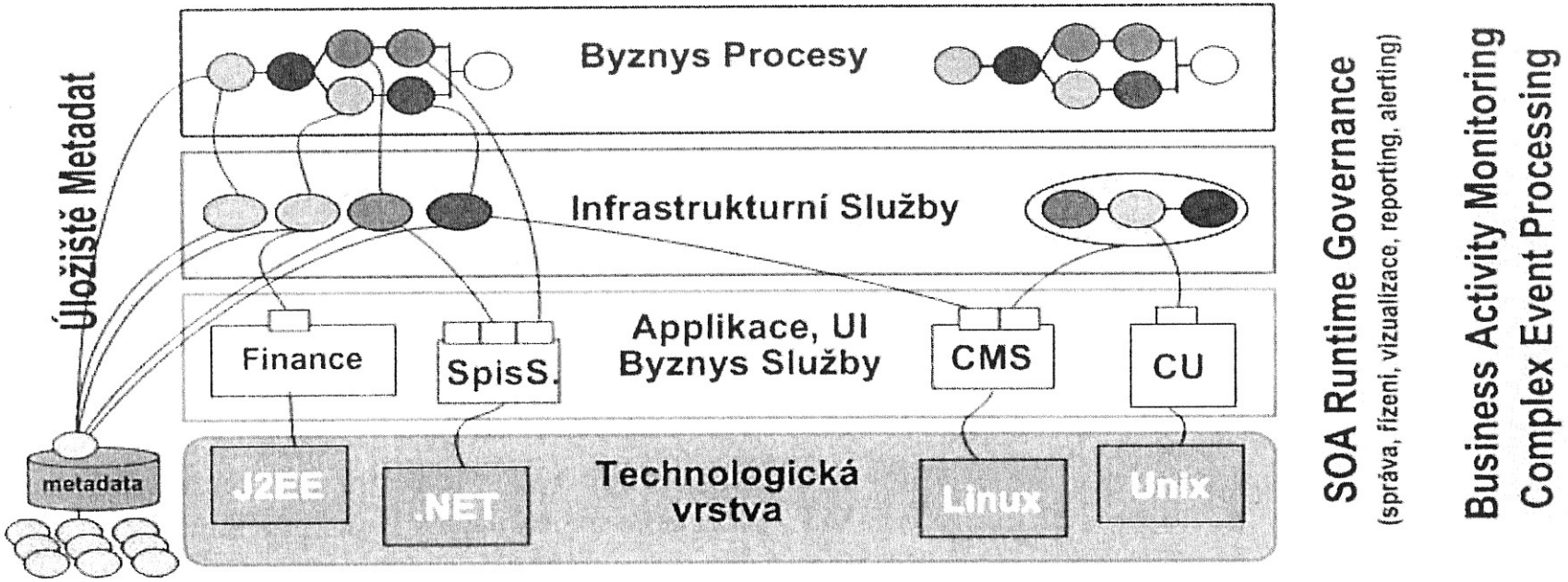
Principy řešení

- Cena projektu za čtvrt miliardy
- SOA se čtyřmi vrstvami podle OASIS se systémovou podporou
  1. Fyzická zařízení
  2. Oběh dat a základní služby
  3. Byznys služby a
  4. Byznys procesy
- Implementace třemi velkými dodavateli

Řešení podle Progres SW

Úspěšná realizace SOA u nás

Logický návrh



Obr. 2 Použitý referenční model SOA společnosti Progress Software.

Dodavatelé


O2 Business Solutions
PROGRESS
SOFTWARE
Aquasoft
PUREdata
PUREbusiness
netprosys
O2 Business Solutions

Pochybnosti

- Je otázka, zda bylo nutné použít takové SOA
  - Možná varianta DB a síť senzorů

Komplexní (těžká varianta) SOA

- V daném příkladě jde o uplatnění referenční architektury podle standardů OASIS
- Charakteristiky
  - Shora dolů + tendence nově vše (big bang neboli velký třesk)
  - Dosti univerzální
  - Dosti až velmi drahé
  - Spíše pro velké firmy a velké dodavatele
  - Mnoho malých služeb
  - Uplatňování principů objektového vývoje
  - V kombinaci velký pro velkého dosti úspěšné,
  - Hrozba antivzorů

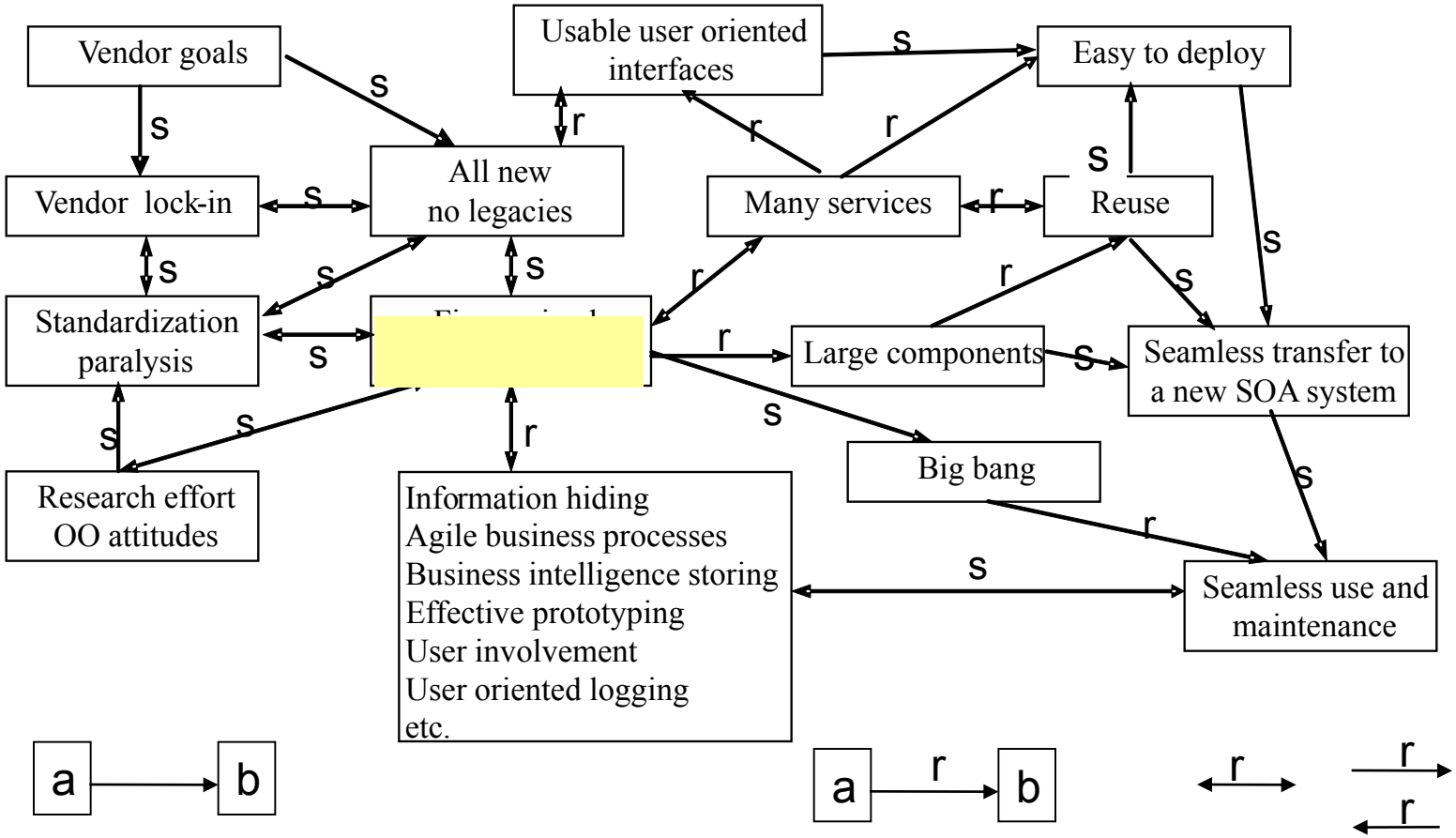
Problém fine grained services v SOA

- Jsou obvyklé u velkých dodavatelů a u webových služeb
  - Je jich mnoho, jsou spíše malé
- Do značné míry se vzájemně podmiňují s normami (normy je do jisté míry vyžadují, samy fine grained services jsou vhodné pro standardizaci)
- Z pohledu SME indukují mnoho nevhodných vlastností a praktik
- Mají tendenci používat RPC (i když to z norem neplyne, je to spíše díky nevhodnému používání praxe z objektově orientovaného vývoje)
  - To je pro uživatele nepříjemné
    - Zhoršuje to možnosti agility a spoluúčasti uživatelů při specifikaci požadavků a nadměrně zvyšuje počet služeb

Potvrzeno některými stížnostmi na SOA

- Kdo se má v spoustě služeb vyznat, cena je vysoká
- Velké náklady správu služeb (service government)
- Menší úspěch SOA u malých firem a někdy ve státní správě.
- Nevyhovuje daným potřebám
- Změny metodou velkého třesku

Undesirable consequences of object thinking (RPC + fine grained interfaces)



If a is getting stronger, b also does
if a is getting weaker, b also does

If a is getting weaker, b is getting stronger
if a is getting stronger, b is getting weaker

Příklady SOA ze života

- *e*-komerce
- Informační systém státní správy (e-government)
- Spolupráce zdravotnických zařízení
- Možná implementace CRM, SCM
- Informační systém globálního podniku
- Řízení procesů (soft real-time)
- Atd.

Termíny*:*

- ***Aliance*** (partner se musí vyhledat)
- ***konfederace*** (partneři jsou v podstatě známi),
  - rozhraní není být uživatelsky příjemné (real time)
  - rozhraní je uživatelsky orientované a hrubozrnné
    - tento typ konfederace nazýváme **unie**

Budeme se zabývat zpravidla konfederacemi. Mnohé lze použít i pro aliance

Konfederace a aliance

- ALIANCE
  - Na začátku komunikace se partner musí vyhledat. Typické pro e-komerci, reservační systémy atp. Nutnost standardů
- KONFEDERACE
  - partneři jsou zpravidla známi. Široká paleta aplikací, historicky nejstarší SOA systémy (soft RT systémy, e-government, řízení globálních společností, výrobní systémy, koalice podniků a zdrav. zařízení). Protokoly komunikace mohou být proprietární a uživatelsky orientované. Důraz na hrubozrnnost a uživatelskou orientovanost zpráv i byznys služeb
- Existují i mezistupně mezi konfederacemi a, nejsou ale časté. Často je jádro systému konfederace a periferie systému je koncipovna jako aliance. Konfederace může využívat služby a funkce vyvinuté pro aliance a naopak.
- Konfederace mohou mít specifické vlastnosti. Důležité jsou unie (platí pro jádra systémů), kdy jsou služby velké komponenty a vzájemně se znají. Pak lze snadno dohodnout uživatelsky příjemné rozhraní služeb i celého systému

Konfederace volné a úzce vázané (unie a vlastní konfederace)

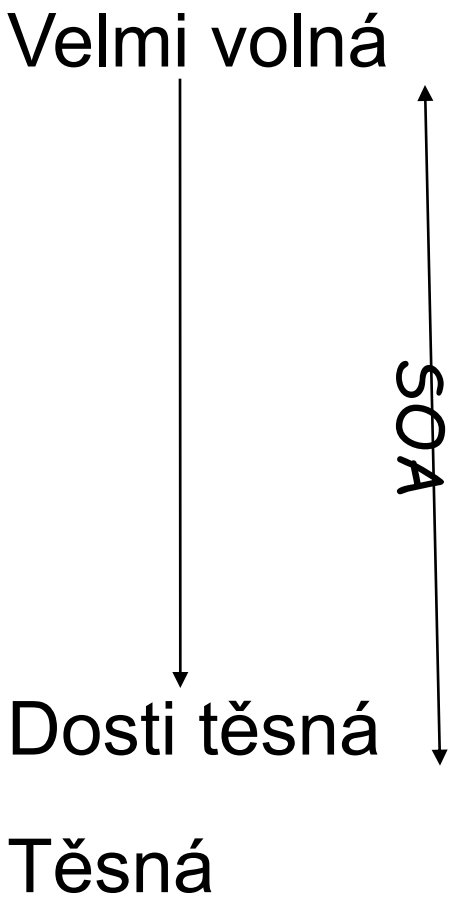
- Konfederace mohou být různého typu
  - IS podniku, části musí poslouchat centrum a systém není extrémně velký, typická oblast použití pro Enterprise Service Bus ESB
  - IS státní správy nebo globálního podniku, části jsou téměř nezávislé a není jich extrémně mnoho, takové systémy nazveme Unie
- Klíčová výhoda konfederací – rozhraní služeb může být uživatelsky orientováno

Konfederace otevřené a uzavřené

- Konfederace mohou být různého typu
  - IS podniku, části musí poslouchat centrum a systém není extrémně velký
  - IS státní správy nebo globálního podniku, části jsou téměř nezávislé - unie
  - Řídící systém menší technologie
    - Menší komponenty, programátorské rozhraní, uzavřenost
- Klíčová výhoda konfederací – rozhraní služeb může být uživatelsky orientováno, v uniích je obvykle uživatelsky orientováno

Těsnost vazeb

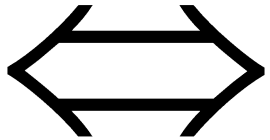
- e-komerce
- Informační systém státní správy
- Spolupráce zdravotnických zařízení
- Možná implementace CRM, SCM
- Informační systém globálního podniku
- ERP
- Řízení procesů (soft real-time)
- OO aplikace



Příklady SO ze života, kdy ESB

- e-komerce
- Informační systém státní správy
- Řízení procesů (soft real-time)
- Spolupráce zdravotnických zařízení
- Malý podnik
- Možná implementace CRM, SCM
- Informační systém globálního podniku
- Lokální divize globálního podniku
- Atd..

Málo vázané



Silně vázané

Příklady SO ze života, kdy ESB

- e-komerce
- Informační systém státní správy
- Řízení procesů (soft real-time)
- Spolupráce zdravotnických zařízení
- Malý podnik
- Možná implementace CRM, SCM
- Informační systém globálního podniku
- Lokální divize globálního podniku
- Atd..

Málo vázané

} Je vhodné/možné použít Enterprise Service Bus, pro převod na jednotný meziformát a jiné funkce

Silně vázané

Příklad SOA

- Volný systém e-komerce
  - Služby se vyhledávají
  - Celosvětový middleware
  - Velmi velká otevřenost
  - Nutnost používat normy

Příklad SOA – e-goverment

- Velmi velké velmi autonomní komponenty
- Jádro systému tvořeno relativně malým počtem komponent
- Uživatelské rozhraní komponent prakticky nutné
- Velmi mnoho i neautorizovaných uživatelů
- Podpora profesní byrokracie
- ESB zatím spíše ne
- **Datové schránky lze použít jako architekturní službu (prostředek integrace služeb do procesů)**

Příklad SOA – IS globálního podniku

- Relativně autonomní komponenty proměnné velikosti
- Často systém dodaný na klíč
- Střední úroveň otevřenosti
- Systém tvořen relativně velkým počtem komponent,
  - Tendence k jemnozrnosti
- Často napojení na e-komerci
- Uživatelské rozhraní komponent výhodné
- Různorodí uživatelé, není jich enormně mnoho
- Podpora strojové byrokracie
- ESB se používá často

Příklad SOA – IS menšího až středního podniku

- Relativně autonomní komponenty proměnné velikosti
- Často integrace komponent z různých zdrojů, nakoupené i vyvíjené, nutná integrace legacies
- Střední úroveň otevřenosti
- Jádro tvořeno relativně menším počtem větších komponent, často zapouzdřené legacy
- Uživatelské rozhraní komponent prakticky nutné
- Různorodí uživatelé, není jich mnoho, střední úroveň znalostí IT
- Podpora strojové byrokracie s prvky demokracie
- ESB spíše ne (cena)

Příklad SOA – soft real-time

- Relativně autonomní komponenty
- Často integrace komponent z různých zdrojů
- Nižší úroveň otevřenosti,
- Jádro tvořeno relativně menším počtem komponent, napojení na e-komerci obvykle chybí
- Uživatelské rozhraní komponent není prakticky nutné
- Různorodí uživatelé, není jich mnoho, nemají přístup k rozhraní komponent
- Fine grained rozhraní služeb založené na RPC
- ESB spíše ne
- Praktické použití – řízení procesů

Prvky SO se dosti běžně používají

- *Propojování aplikací interaktivní i dávkové*
- *IS a jiné programy spolupracují na základě p2p, je výhodné, když jsou zprávy coarse grained, user oriented*
  Enterprise Service Bus je vhodný tehdy, kdy je možné jednoduše nařídit standardy
- *Při pragmatickém řešení se kombinují dávky, datová úložiště, služby přístupné jako v aliancích i postupy z konfederací.*
- *Použitelné: web 2.0, ajax atp.*

The Reference Model

- Service
- Dynamics of Services
  - Visibility
  - Interacting with services
  - Real World Effect
- About services
  - Service description
  - Policies and Contracts
  - Execution context

- Pragmaticky: Volné asynchronní propojování
  - Kontrakty zahrnují hlavně dohodu o rozhraní, politika dohody jak používat
  - Změny nelze vrátit, lidi nelze algoritmizovat

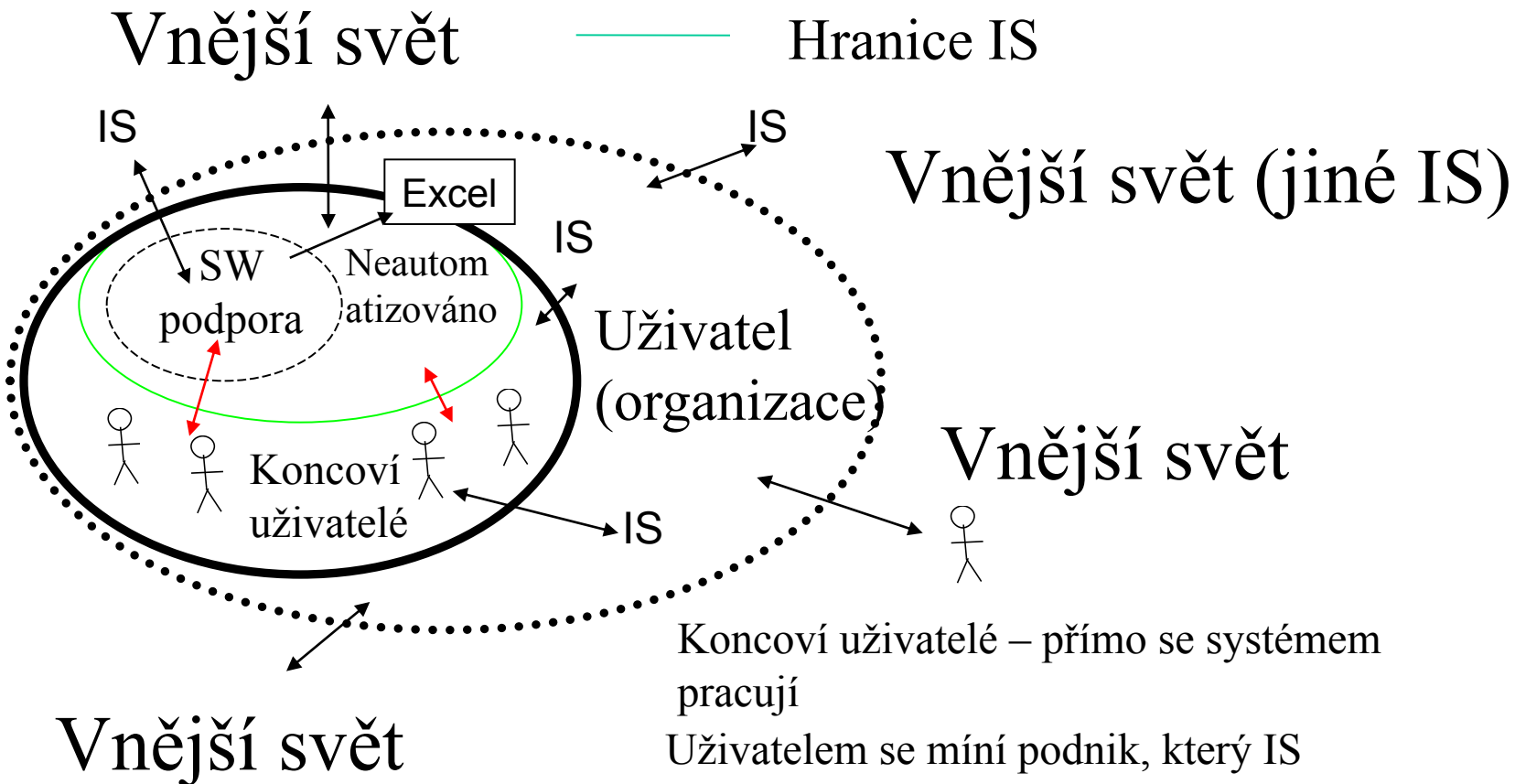
Služby typické pro SOA

- Služby se mohou se vzájemně znát (konfederace)
  - Pak odpadá problém viditelnosti, vyhledávání a zčásti i popisu
- Může existovat více nevyřízených požadavků, chová se jako stále aktivní proces $\Rightarrow$ komunikace je p2p charakteru a po stránce funkční implementuje asynchronní protokol
- Požadavky spíše deklarativní než procedurální
- Slabý vliv kontextu služby
- **Kritérium: Chová se podobně jako běžné služby hromadné obsluhy reálného světa**

Problém

- V opravdu rozsáhlých p2p sítích je problém s centrálními seznamy
  - Kdo za ně odpovídá
  - Co za to má, jak udržuje
  - Nebezpečí přílišného vlivu správce
  - Zbytečný tlak na standardy
- Nevíme, jaké je nejlepší řešení pro centrální službu, asi dekompozice na síť služeb
- Centrální direktorář UDDI se pro globální systémy neosvědčil
  - Částečně se svědčuje se v rámci podniků mj. jako databáze pro business intelligence

Informační systém je vždy součástí většího systému obvykle zahrnujícího i lidi a neautomatizované činnosti



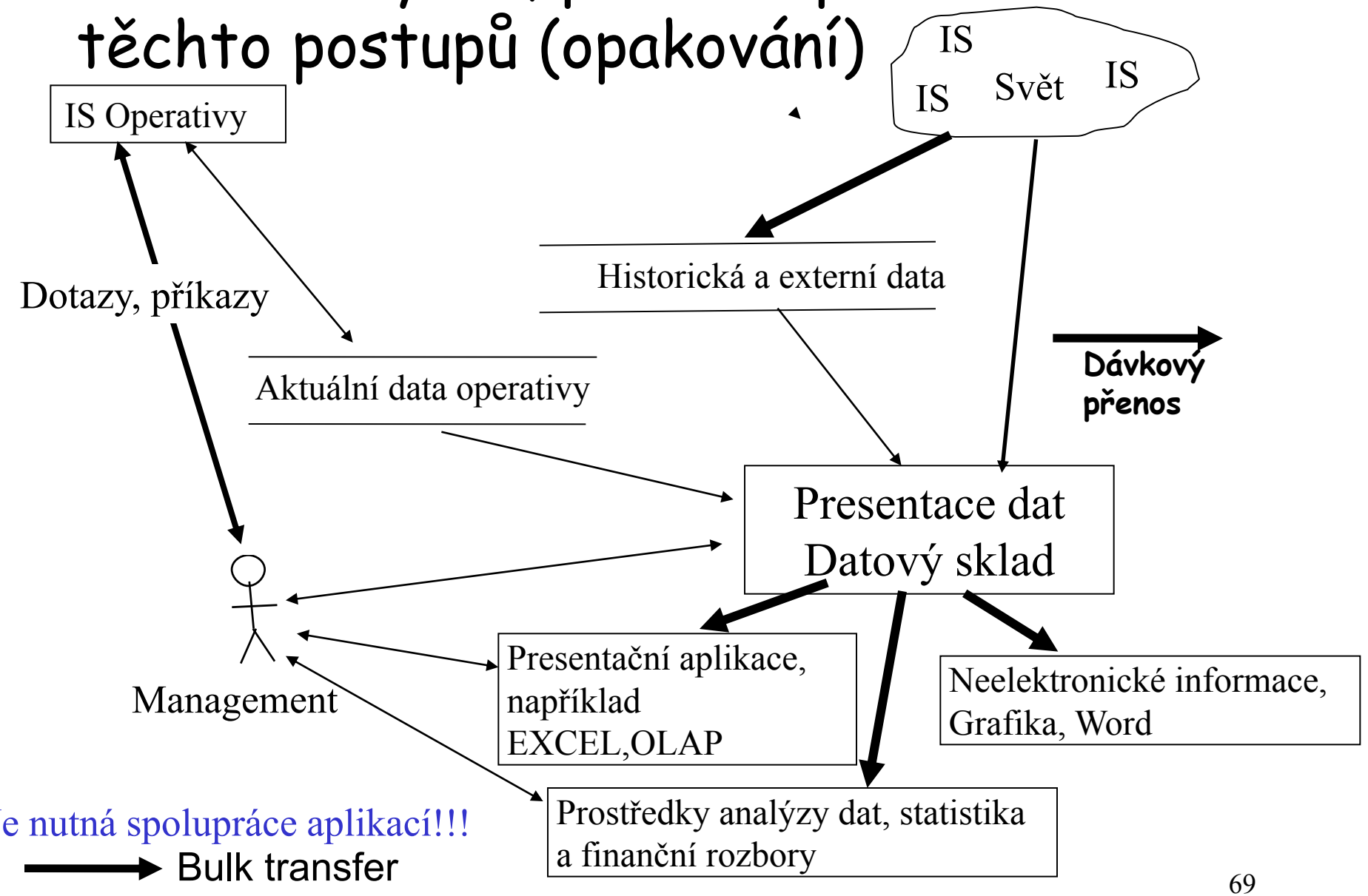
Koncoví uživatelé – přímo se systémem pracují

Uživatelem se míní podnik, který IS využívá

Je běžné, že informační systémy komunikují s jinými informačními systémy.

Informační systémy mohou fungovat jako systémy automatického řízení (avionika letadla), dávat rozkazy lidem, být (dočasně) nahrazeny lidmi, řídit procesy reálného světa (tam je nutno řešit problém dostupnosti dat a problém včasnosti odezvy) a také nová formulace toho, co nazýváme transakce

Manažerský IS, příklad aplikace těchto postupů (opakování)



Pozorování

- Spolupráce SW komponent může být dávková
  - exportem a importem dat (bulk transfer)
  - pomocí společného (virtuálního) datového úložiště plněného dávkově, je-li třeba
- Spolupráce může být interaktivní, primárně asynchronní
  - Přímá výměna zpráv
  - Aktivní databáze, případně s uloženými zprávami
- Některé služby mohou být dávkové (Excel)

Systémy mezi sebou spolupracují
Aby mohly být používány rozumně, musí spolupracovat podobně jako služby reálného světa, být většinou stále k dispozici, (asynchronně) reagovat na požadavky z různých zdrojů a být používány jako černé skříňky. To reálně lze dosáhnout jen, tvoří-li virtuální p2p síť.

Jak zajistit, aby jedna služba čekala na dokončení jiné služby (synchronnost)

- Není systémová podpora, odpověď pozná volající z obsahu zprávy sloužící jako odpověď
- Volaná služba po skončení požadavku vrací odpověď obsahující identifikaci volání a informace, které umožní pokračovat v daném vláknu volající komponenty
- Je možné systémově a obecněji, proto byl vyvinut standard *enterprise service bus*. Jde o řešení typu middlewaru vhodné spíše pro podnikové systémy

Servisně orientovaná architektura, normy

Podrobný rozbor lze nalézt v *OASIS Reference Model for Service Oriented Architecture 1.0*

- Zahrnuje i méně používané modely
- Zdá se nehotový (málo pragmatický)

Jiné normy: W3C , workflow, byznys (podnikové) procesy atd., Open Group SOA Reference Model, Open System Integration Maturity Model

Open Group SOA Reference Architecture

Zaměřeno na integraci

Koncept ABB – Architecture Building Block

Také devět vrstev

Silný důraz na synchronní volání služeb

Nezdá se vhodné pro agilitu

Nové faktory v SW průmyslu

- Změna potřeb
  - Globalizace => nutnost a výhodnost konfederovaných distribuovaných systémů
  - Informační služby státu, dtto
  - Spojování produktů různých výrobců
  - Další SW inženýrské potřeby, např. inkrementální vývoj a modernizace, znovupoužitelnost, …..
- Změna možností technologie, konfederace je dostatečně efektivní
  - Dostupnost a kapacita komunikační infrastruktury
  - Existence programovatelného uživatelského rozhraní, existence vyhovujících norem

Aliance a konfederace a normy

Aliance:

- Partner se na začátku komunikace vyhledává, nelze předpokládat, že je na začátku komunikace znám. Je proto nutné použít celosvětové standardy a Internet

Konfederace:

- Partnerské komponenty se znají, lze dohodnout proprietální pravidla a protokoly, je-li to výhodné
- Komunikace může být i jiného typu, než výměna zpráv založená na internetu

Ukážeme, že servisní orientace je specifické paradigma.

Proto není snadné je zvládnout, je nutné najít a zvládnout nové přístupy a zbavit se některých existujících

Paradigma

- Základní filosofie, techniky, výsledky a postupy daného oboru
  - Newtonova mechanika * kvantová mechanika
- Nutné pro řešení nových problémů, nějak integruje starší paradigmata
- Obtížné přijmout lidmi zvyklými na starší paradigmata
  - Musí je často nahradit až nová generace
  - Přijetí a vycizelování nových postupů trvá roky
  - Spojeno se změnami obchodních a manažerských strategií a postupů a změnami struktury podniků

Problém přijetí paradigmatu Registr účtů a dluhů

Jak udělat registr účtů pro kontrolu oprávněnosti žádosti o sociální dávky a podpořit zjišťování okolností pro poskytování úvěrů, žádoucí je rozšíření na dluhy a perspektivně na majetky

Registr účtů a dluhů

Návrh v klasickém stylu:

Vytvořit centrální databázi účtů a úřad, který by ji spravoval. To chtějí úřady.

Pozorování:

Jde o centrální službu v konfederaci, p2p nemají „rády“ centrální služby (služby jsou IS bank, úřadů, …). Lze proto očekávat potíže, např.

- Náročnost replikace dat a aktualizace
- Malá flexibilita (a co jiný majetek, co dluhy)
- Problémy se zneužíváním (kdo použil, nebezpečí zcizení dat)
- Vysoké náklady na provoz

Registr účtů a dluhů

Alternativní řešení:

- Vytvořit relativně jednoduchou aplikaci, která se chová jako portál mající omezený přístup k ke všem IS bank pro dotazy tvaru: „Kolik má u vás pan X zrovna teď peněz/dluhů.“ Pro banky je to také jednoduché, většinou už mají téměř vše implementováno a mohou si ohlídat, kdo se to vlastně ptá.
- Problémy s efektivností jsou nepravděpodobné, dotaz musí odpovědný úředník naťukat, takže dotazy nebudou si příliš časté, a nepředpokládají se komplikovanější způsoby zpracování dat. Je nutná ochrana dat.
- Skrytý problém: Ochrana osobních dat

Nutnost SOA, IS globálního podniku

- Globální podnik je tvořen sítí autonomních divizí, které lze prodat nebo které byly nakoupeny
- Nutnost spolupráce s IS proměnné množiny obchodních partnerů pro B2B
- Potřeba neustálé modernizace IS
- Potřeba používat nové nástroje analýzy dat vyvíjené různými výrobci
- Velmi rozsáhlý soubor koncových uživatelů s různorodými a proměnnými potřebami a právy
- Sourcing

Výhody SOA při prodeji a nákupu org. jednotek

- Nakupované divize bývají kupovány se svými IS, které se musí nějaký čas používat
- Divize je při prodeji cennější, je-li prodávána se svým IS. Ten proto musí mít v rámci podniku jistou autonomii (lze ho oddělit, neprodává se s ním příliš mnoho znalostí o celofiremním IS)
- Je žádoucí/nutné, aby se při B2B (business to business) používaly IS partnerů pokud možno bez úprav

Nutnost SOA (konfederace), státní správa

- Jednotlivé úřady mají své IS
  - Vzniklo historicky
  - Je politický zájem zachovat samostatnost IS jednotlivých úřadů
  - Je to i zájem věcný
    - Ochrana dat
    - Jasné odpovědnosti
    - IS potřebují k práci –> budou se o něj starat
  - Spolupráce s IS podniků a daňových poplatníků
- Přístup občanů k IS úřadů => Různorodé skupiny koncových uživatelů, mnoho uživatelů

Kdy je nutné použít SO, příklad státní administrativy

1. Jednotlivé složky státní správy mají své systémy, které si chtějí pokud možno zachovat
   - Vložily do nich svoje znalosti, mnoho funkcí se nesmí měnit a musí se provádět na daném úřadě
   - Musí ručit za jejich práci a proto jim musí věřit, leccos je tajné
   - Lidé se brání se změnám, které jim bezprostředně nic nepřináší a mohou je i ohrozit, změny se ale musí dělat ve spolupráci s nimi

Kdy je nutné použít SO, příklad státní administrativy

1. Jednotlivé složky státní správy mají své systémy, které si chtějí pokud možno zachovat
   - Mocensky se lidé i úřady snaží zachovat svoji autonomii a své úkoly a zaměstnance
   - Autonomie je výhodná pro získání výhod (známosti mimo úřad, různé výhody, provize až korupce)
   - Je nereálné všechny systémy přepisovat znovu (cena, termíny, spolehlivost, rizika při přechodu) a i rekonstrukce nemůže mít za cíl monolitické řešení
   - Nelze připustit přílišnou závislost na jednom dodavateli

Kdy je nutné použít SO, příklad státní administrativy, výhody

1. Modifikace systému jsou snazší, mnohé SW inženýrské přednosti
2. Systém musí být otevřený a spolupracovat s obdobnými systémy (obce a města, armáda, IS podniků, zdravotní systémy, IS mezinárodních organizací). Rozsah spolupráce je pro různé úřady různý. Je proto žádoucí použít takovou architekturu, která umožní stejný způsob spolupráce mezi subsystémy uvnitř státní správy i mimo ni. *Možná řešení* ***Active MQ, Datové schránky***

Požadavky

- Je nutno zachovat autonomii IS úřadů
  - Nelze jinak z politických důvodů
  - Je to výhodné věcně (alespoň potenciálně)
  - Je to výhodné softwarově inženýrsky
- Je nutné umožnit spolupráci s autonomními IS podniků
- Je nutné umožnit přístup občanům i IS podniků.

To jsou prakticky identické požadavky jako u SOA pro malé a střední podniky !!!

Proč to nejde

- Zájmy firem nepřijít o zakázky (SOA může snížit rozsah zakázky)
- Nutnost změnit pravidla spolupráce
- Past standardů
- Hrozba účinné kontroly korupce a hospodaření
- Technické problémy

Příklad nákupního centra

Americké automobilky se před jistou dohodly, že zhromadní nákupy součástek do automobilů tím, že vytvoří společnou nákupní organizaci NO. Proces zhromadňování musí být založen na spolupráci IS v NO s IS klientských automobilek, které jsou i vzájemnými konkurenty. Odtud plyne, že IS v NO a IS klientů musí být nezávislé a komunikovat vhodným způsobem. IS klientů musí být integrovány jako černé skříňky. Je nejvýše vhodné, aby takový systém měl servisně orientovanou architekturu

Společné vlastnosti všech servisně orientovaných systémů, opakování

- Virtuální peer-to-peer spolupráce autonomních komponent (autonomie využití a také vývoje) spolupracujících jako služby masové obsluhy, p2p je nutnou podmínkou, aby se SW komponenty chovaly obdobně jako služby v reálném světě(v tom, co je služba není ještě mezi experty úplná shoda, pro nás autonomní komponenta, peer ve virtuální síti pracující asynchronně)

Společné vlastnosti všech servisně orientovaných systémů II

Dostupnost některých operací jinak obtížně realizovatelných (částečný outsourcing, decentralizace, podpora manažerských operací uživatelů obecně)

- Výhodné SW-inženýrské vlastnosti (otevřenost, modifikovatelnost, metody oživování, znovupoužitelnost, úspory při vývoji a údržbě,…)
- Použitelnost agilních forem vývoje,

Kdy je nutné použít SO

- Spolupráce existujících systémů, které musí být použity tak, jak jsou z následujících důvodů:
  - nelze je dost rychle přepsat (viz reorg cycle, vývoj trvá tak dlouho, že než systém dokončím, je zastaralý)
  - dosavadní uživatelé musí „svým“ službám věřit a musí mít příležitost specifikovat, co potřebují (feeling of ownership), neradi se učí něco, co jim nezlepšuje práci, jsou ochotni nést odpovědnost jen za to, čemu věří
  - uživatelé jsou sami velmi autonomní a většina funkčnosti jejich systémů zůstává lokální (srv. CRM, zdravotnictví, *e*-komerce), totéž chtějí od systému
  - politické a mocenské důvody
  - levoty (provize, úplatky, snahy o to, aby nebyl dohled)

Kdy je nutné použít SO případně komponentovou orientaci

- Velký systém musí být budován a modifikován po částech ze softwarově inženýrských důvodů
- Některé operace nelze rozumně impementovat jinak, než v SO (selektivní insourcing a outsourcing, decentralizace, integrace lidí do systému, agilní business procesy)
- Systém sám má charakter spolupráce služeb nebo aplikací (typické pro řízení procesů majících charakter služeb)
  - zčásti přítomno v systémech psaných v COBOLU
  - Soft RT, operační systémy – obsluha periferií, symetrický multiprocessing, komunikační sítě

Kdy použít SO

- Jako prostředek podpory distribuovanosti
- Jako prostředek dekompozice na daném místě, v daném počítači
- Jako prostředek sdružování kapacit (gridovské systémy, viz projekt CETI,cloud)
- Podpora inkrementálního vývoje a jiné SW inženýrské výhody, např. znovupoužitelnost, kombinace produktů různého původu
- Podpora otevřenosti, CRM a SCM
- Aby to vůbec šlo vyvinout v rozumných termínech(Neexistence SOA v sedmdesátých létech byla jedním z důvodů krachu projektu protiraketové obrany SALT)

Výhody servisně orientované architektury

- Chyby zůstanou lokalizované, modernizaci lze provádět po částech
- Stávající systémy mohou sloužit nadále bez podstatnějších změn, lze integrovat produkty třetích stran
- Flexibilita: Výše uvedený registr se dá snadno upravit na kontrolu dluhů a případně majetků
- Škálovatelnost: Lze snadno doplňovat nové služby a také je klonovat

Výhody komponetově (servisně) orientované architektury

- Autonomie částí zvětšuje odolnost systému proti výpadkům
- Dosavadní uživatelé existujících služeb jim mohou věřit, protože se v podstatě neměnní,. Nemusí se učit příliš nového a mohou důvěřovat i datům.
- Úspory nákladů z důvodů znovupoužitelnosti, použití produktů třetích stran a snadnějšího vývoje všeobecně

$$Prac = c * Delka^{1+a},\ a \cong 1/8$$

Dekompozice do n částí sníží pracnost přibližně n^{-a} krát

Proč až nyní

- Informační systémy musí být nyní *dekomponovány* do služeb pro zachování investic do existujícího SW,
  - dříve to nebylo ultimativním požadavkem, systémy bylo možné budovat znovu od počátku,
  - některé komponenty *musí* být integrovány jako černé skříňky (IS externích organizací a třetích stran)
- Nebyla *vhodná komunikační infrastruktura* pro všeobecné použití (internet) a výkon hardwaru včetně komunikací

Proč až nyní

- *Marketingové důvody*, neochota nabídnout zákazníkům větší samostatnost, využití stávajících customizovaných systémů, nutnost změny obchodní strategie výrobců softwaru
- Utajování metodologie SOA jako *konkurenční výhody nebylo možné do nekonečna, dnes se stává kontraproduktivní*
- Nutnost nových strategií vývoje, prodeje a používání SW

Proč až nyní takový poprask

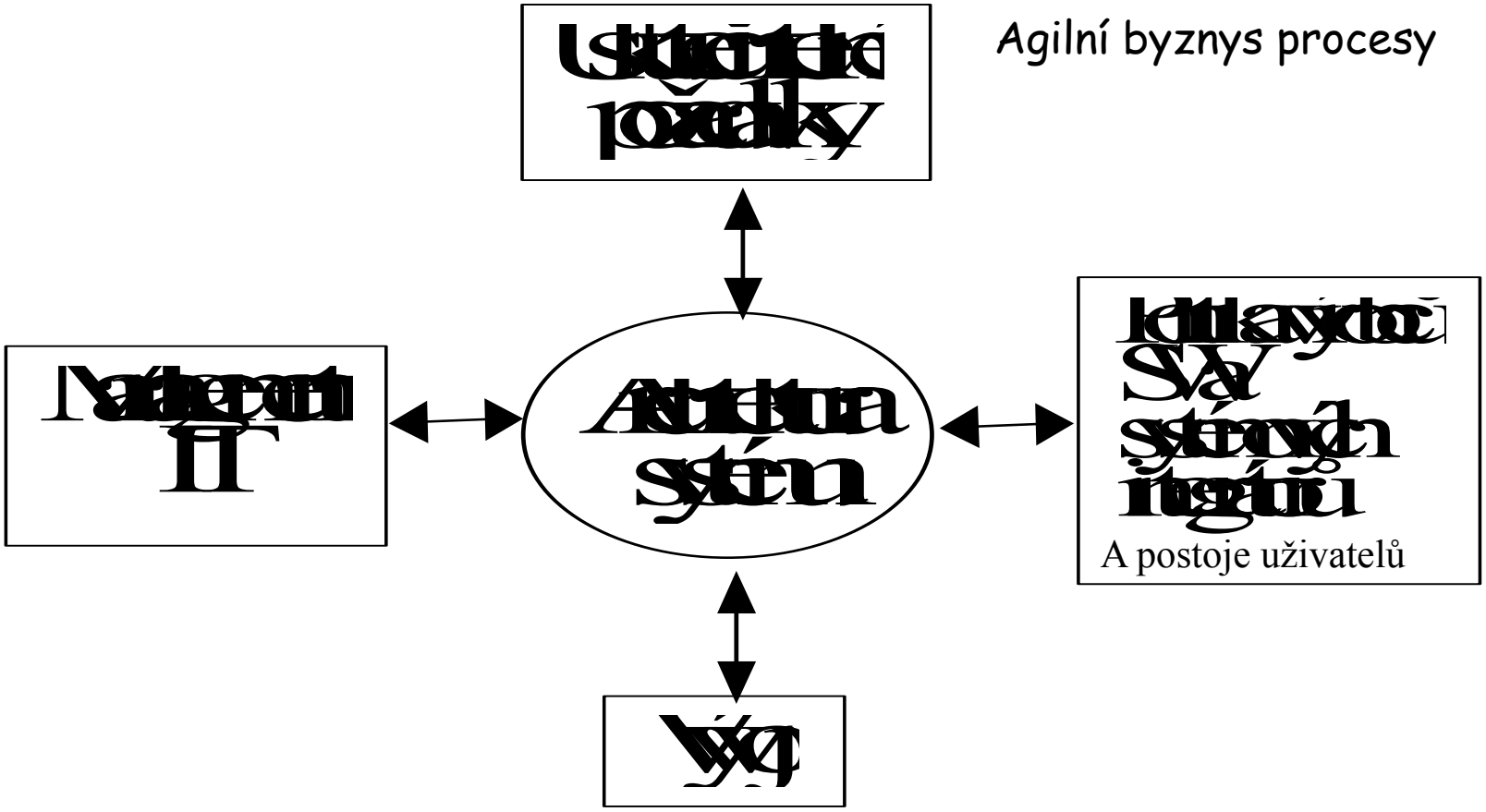
Nové paradigma – změna filosofie, technik, dovedností, nepříjemná nutnost používat existující zdánlivě zastaralé aplikace a kombinovat dávkové a interaktivní způsoby práce a navíc i datově a příkazově orientovaná rozhraní, *nové paradigma i pro managementy dodavatelů i zákazníků*!

- Potřeba uživatelsky orientovaného rozhraní => *nutnost hlubší spolupráce vývojářů s uživateli z různých stupňů organizační hierarchie*, uplatňování principů agilního vývoje, viz níže

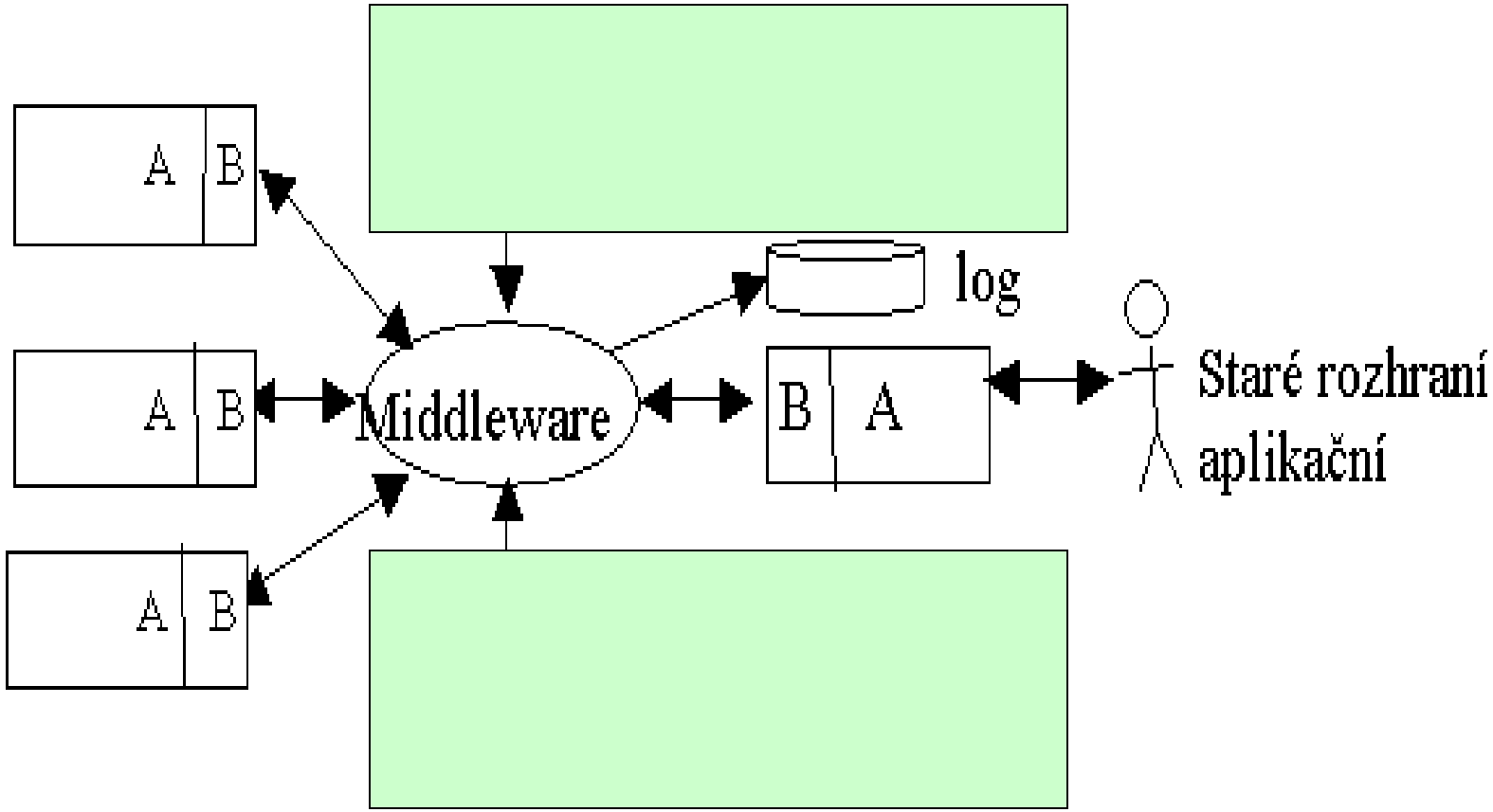
Hlavní překážka prosazení nového

- Pocit, že se o nic nového nejedná, že jde je o převlečené nové věci
- Lenost
- Snaha zachránit existující investice do SW
- Nevhodné použití aliancí, které jsou vhodné pro velké podniky, v malch podnicích a e-gov

Klíčová role architektury



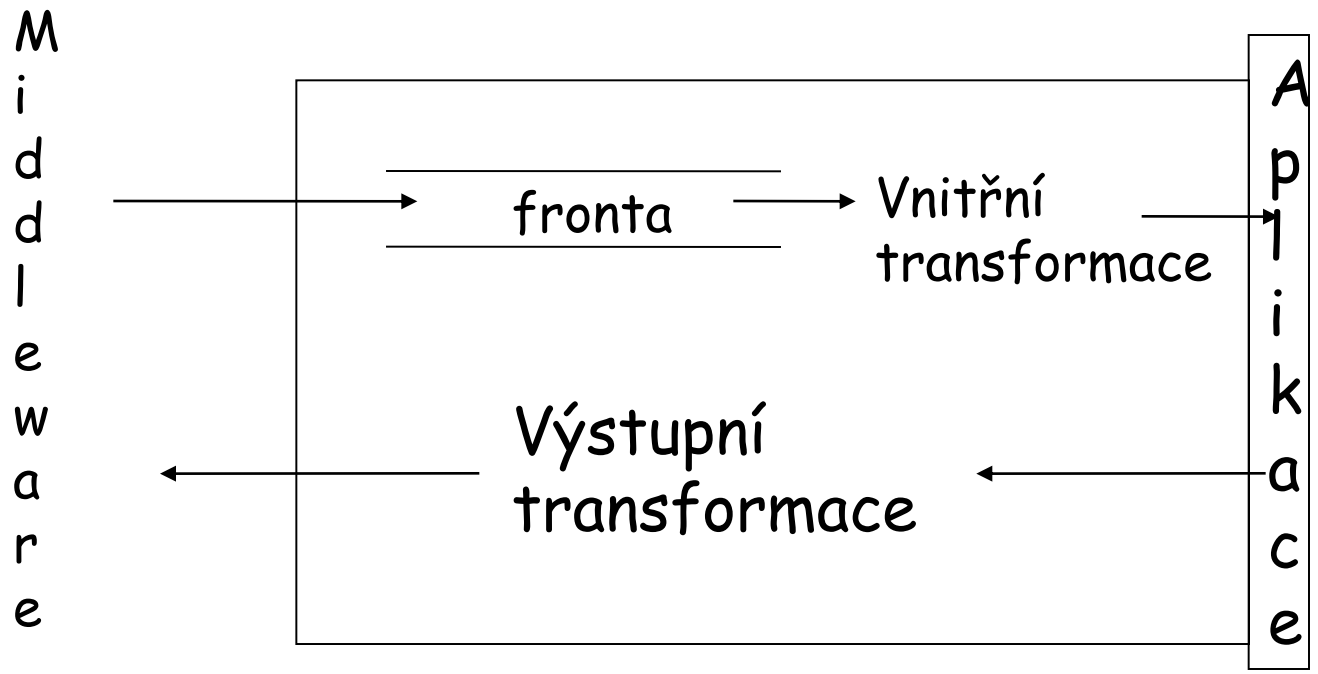
Architektura aliance a konfederace



A – aplikační služba, B – její rozhraní (primární brána) UR je uživatelské rozhraní, např. portál

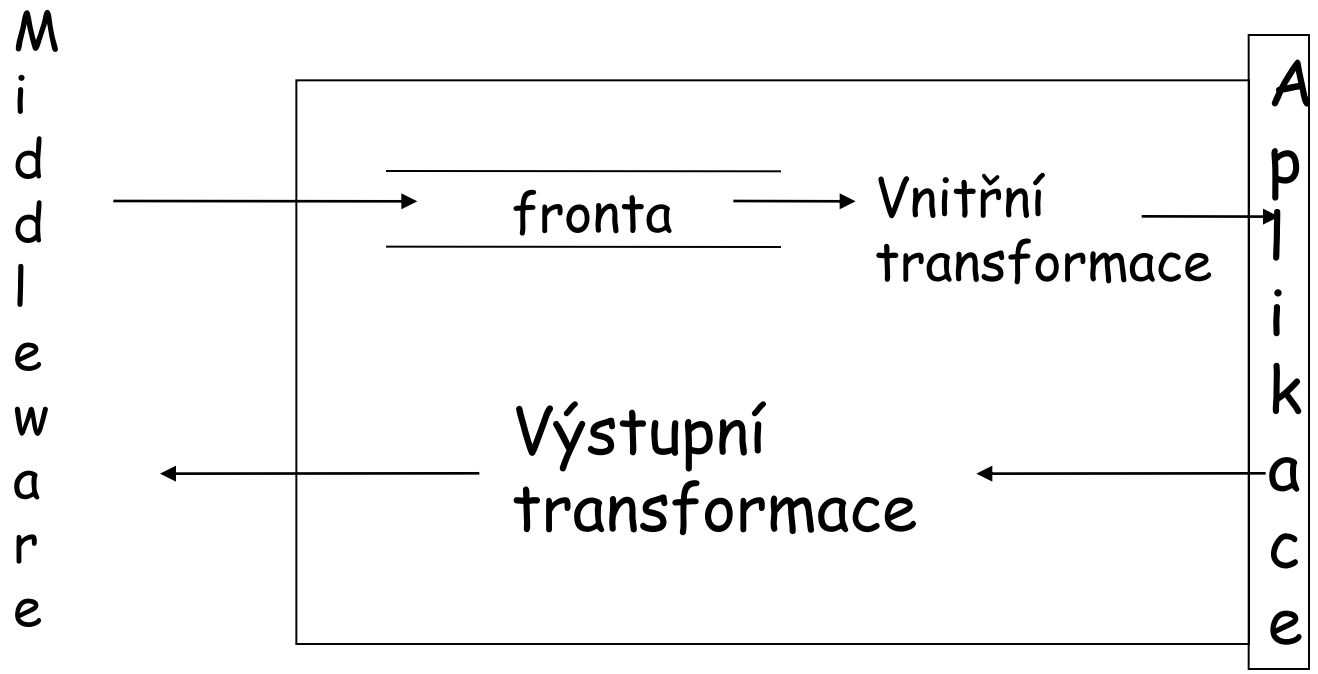
Staré aplikační rozhraní je u komplexnějších služeb nutností (viz IS úřadů), usnadňuje podstatně vývoj.

Struktura brány



Brána má interní skrytou datovou strukturu, u ESB je skryta, IBM to nabízí explicitně jako MQ – Message queue, otevřeně je dostuný systém Active MQ a mailboxy v Unixech

Struktura brány



Pokud middleware nabízí jen point-to-point, MQ to nezmění. Řešení je možné přes specializované (architekturní) služby typu data store. Je možné jako bránu použít datovou schránku

Struktura brány

Při implementaci lze použít dotNet,ASP, primitiva UNIXu, MQ firmy IBM, nebo použít nějaký ESB….

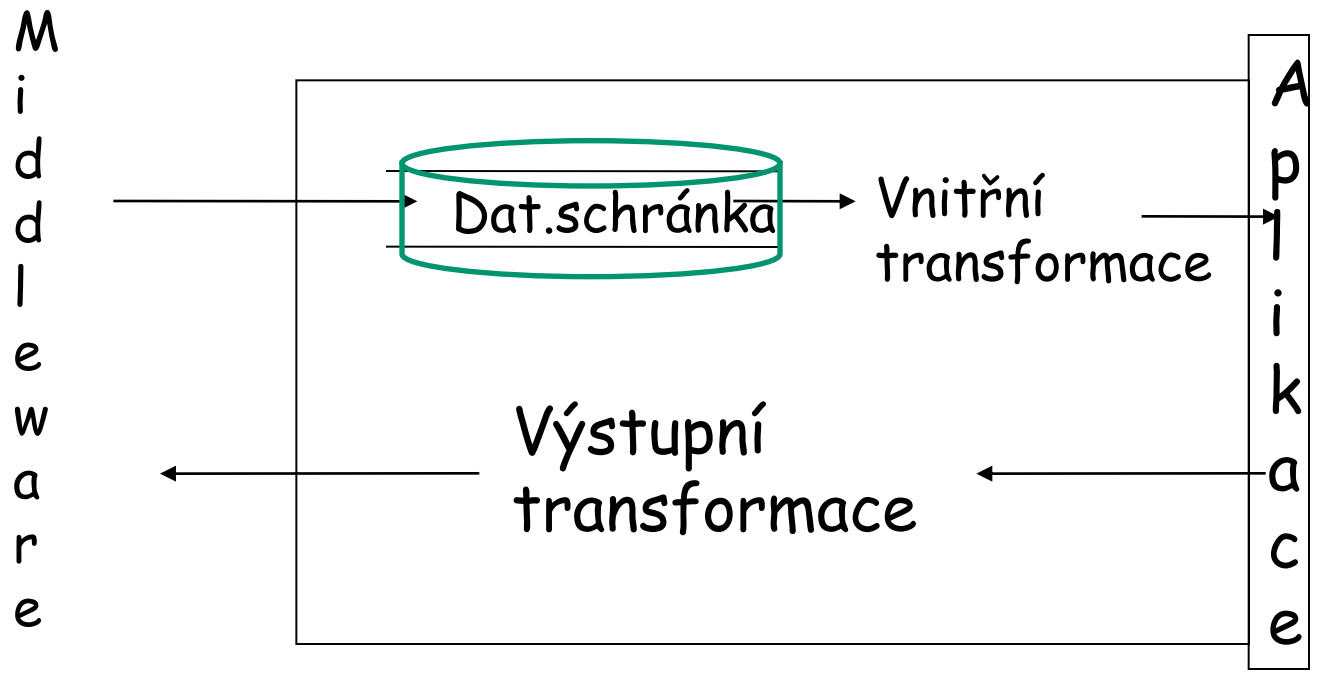
Je možné jako bránu použít datovou schránku

Je to příklad konektoru ve smyslu komponentových architektur.

Moderní systémy nabízejí modernější řešení (publish-subscribe, …). To lze implemetovat pomocí služby data store ukládající zprávy

Jiná možnost je použít datové schránky ve státní správě jako architekturní služby

Brána s datovou schránkou



Brána může být značně komplikovaná, schránka je vlastně SW službou

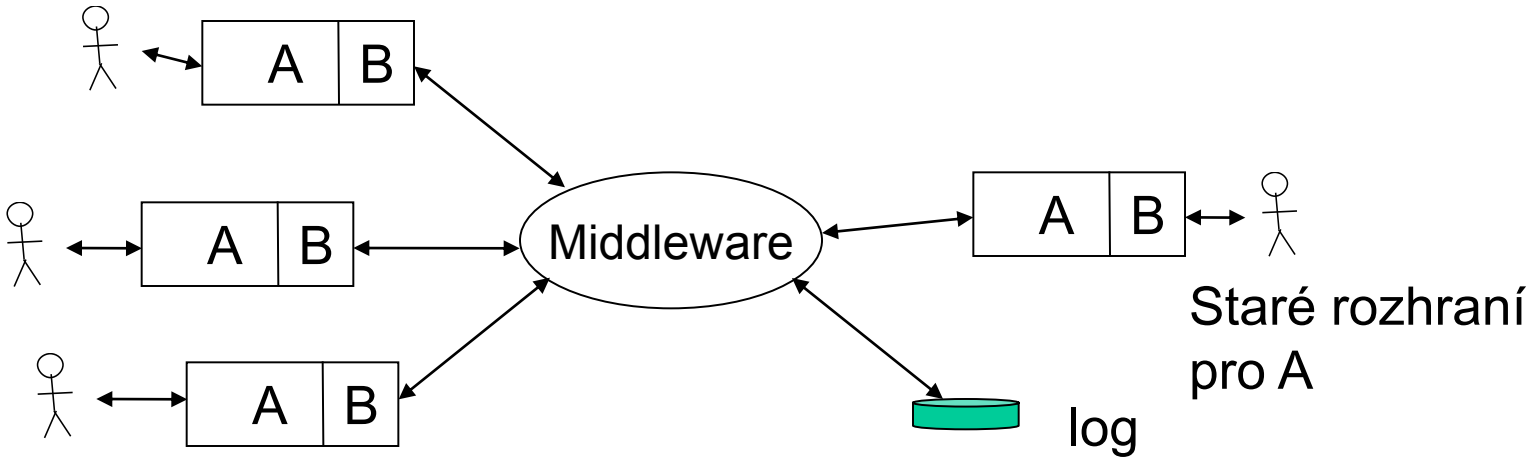
Důsledky p2p architektury

- Stabilita systému zásadním způsobem závisí na stabilitě (neměnnosti) rozhraní
- Rozhraní se bude málo měnit, pokud nebude záviset na implementačních detailech a změnách úrovně technologie a bude ověřeno používáním (je-li např. deklarativní a uživatelsky orientované, uživatelé v operativě používají léty ověřené postupy, viz účetnictví)

Důsledky p2p architektury

- P2p systémy se málo kamarádí s centralizovanými službami, i jen funkčně centralizovanými.
  - Takové služby lze sice někdy používat na úrovni podniku, lze však očekávat problémy na globální úrovni, viz zkušenosti s UDDI
- Přidání/ubrání služeb je relativně snadné, pokud mají vhodné rozhraní
- XML a www výhodou, ne však podmínkou, důležitá je hrubozrnnost rozhraní komponent

Obvyklá architektura aliance, *e*-komerce



Funkcionalita brány může být zčásti přesunuta do separátní služby (to je obecnější řešení) nebo zčásti do middlewaru

Podmínky spolupráce v alianci

Partner se musí nejdříve vyhledat ⇨

- Partneři dostupní přes veřejný celosvětový middleware
- Rozhraní služeb závisí na otevřených standardech
  - Je-li jazyk zpráv standardizován, je v jistém smyslu univerzální
  - Přístup přes standardní nástroje (prohlížeče)
  - Poskytované služby popsány standardním způsobem, popis dostupný i pro programy (WSDL)
  - Výhodný je telefonní seznam (UDDI), ale to je centrální služba a mohou s tím být potíže v p2p prostředí, to se již potvrzuje (údržba seznamu se ukázala problematická)
    - Kdo to zaplatí
    - Zvýhodnění udržovatele
    - Důvěryhodnost

Komunikace formou zpráv

  - žádná data v middlewaru, na internetu se datové úložiště obtížně implementuje

Podmínky spolupráce v alianci

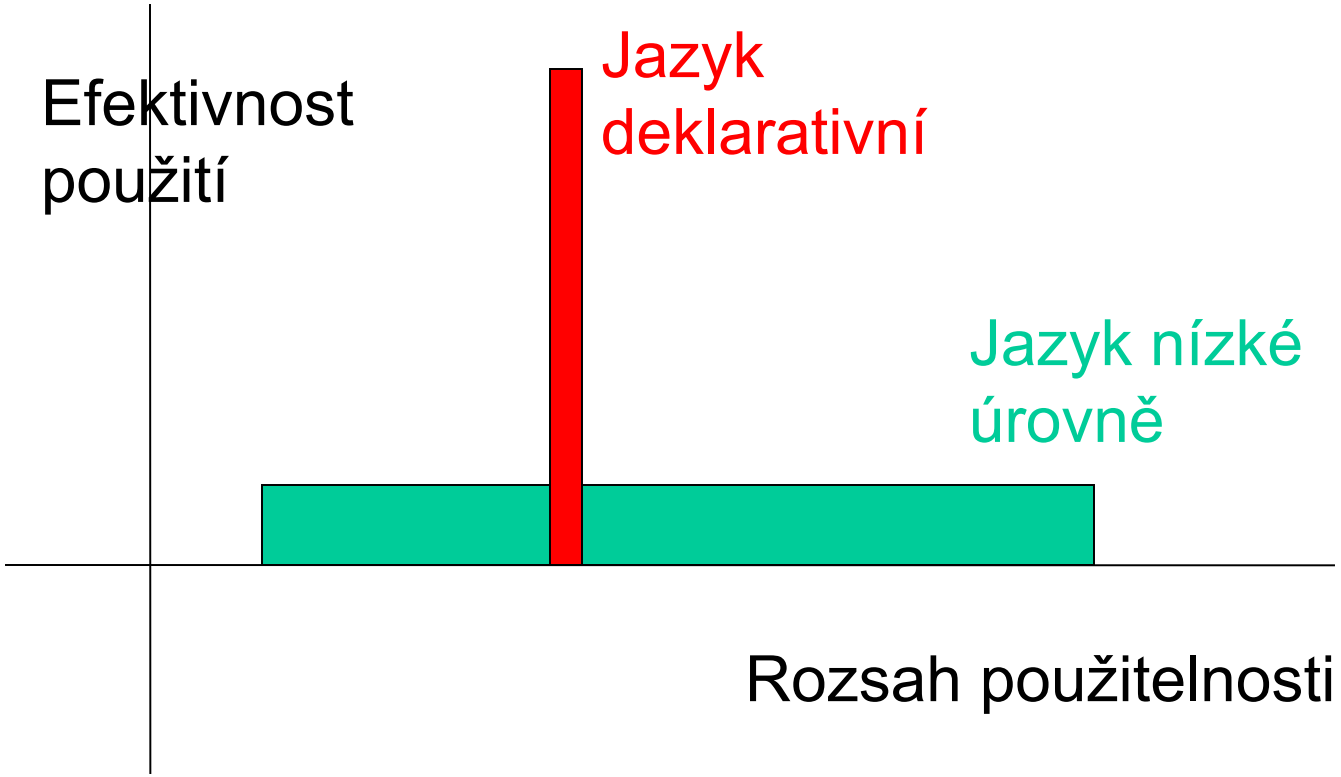
WSDL, SOAP a UDDI původně vycházely z nízkoúrovňového konceptu volání metod, čili **procedurálním rozhraní**

To ale znamená, že bylo obtížné dosáhnout uživatelské orientace rozhraní a hrubozrnnost rozhraní

To je podstatné omezení

V poslední době se přechází na SOAP – message literal nebo message encoded a tím na rozvinutější komunikaci, SOAP je obálkou formalizované věty uživatelského jazyka, má to i výhodu menšího zatížení komunikačních linek

Deklarativnost kontra procedurálnost



Podmínka deklarativního rozhraní

- Je nutné mít dobře vymezené komponenty
  - Mají totiž mít možnost mít uživatelsky orientované rozhraní pro nějaký ne zcela malý uživatelský obor
- Musí být zčásti součástí specifikací (pro integraci legacy systémů je to nutností)

Formáty zpráv v alianci, adresát se vyhledává

- Univerzální řešení formátu zpráv: univerzalita ⇒ nízká úroveň ⇒ procedurálnost ⇒ SOAP, ve starší versi (řešeno v rámci SOAP – RPC, nebo SOAP RPC literal)
  - Programátorsky orientováno ⇒ omezení role uživatelů při definování a řízení business procesů a zásazích do jejich chodu, není výhodné pro uživatelsky orientovaná rozhraní
  - Preferuje tvar specifikace služeb (WSDL) a užití telefonního seznamu (UDDI)
  - Vhodnější pro operativu a pro rozhraní bez vlastní inteligence

Nevýhody aliancí

- **Služby nemají uživatelsky orientované rozhraní –**
  - **obtíže při použití (např. u podnikových procesů)**
  - **při vývoji (je to problém pro agilní formy specifikací společně s uživateli)**
- Potíže při řešení sporů, odpovědností a ad hoc zásazích do procesů (nečekané události, nové informace), zprávám koncoví uživatelé a experti nerozumí
- Změny obchodních procesů je nutné řešit uvnitř služeb
  - Neflexibilní, u služeb s vlastnostmi černých skříněk (produkty třetích stran) téměř nereálné

Nevýhody aliancí

- Existence centrálních služeb je proti duchu p2p
- Rozhraní je pracné, není uživatelsky orientované a má tendenci zviditelňovat implementační detaily
- Potřebné normy se rychle mění a rychle rostou => nestabilita rozhraní a nadměrný vliv velkých SW firem (srv. Vendor lock in antipattern)
  - Tendence k drobným službám
  - Potíže s agilitou byznys procesů

Výhody aliancí

- V *e*-komerci jiných systémech v podstatě jinak nelze
- Služby lze programovat autonomně, podobně jako kdysi programy používající hloupé terminály
- Využívají se webovské služby a výsledky výzkumu sémantického webu
- V lidské společnosti fungují služby podobně, lze použít analogické postupy
- *Úloha pro další výzkum*

Souhrn hlavních rysů konfederativního SOA

- Síť autonomních komponent, které mají rysy nebo přímo jsou aplikacemi
- Při navazování komunikace se partner nemusí zpravidla vyhledávat (je znám předem)
- Formát zpráv lze dohodnout, je žádoucí a možné aby byl uživatelsky orientován
- Komponenty jsou používány jako černé skříňky
  - Komplexní funkcionalita, komplexní rozhraní deklarativního typu
  - Různé technologie, různí výrobci kódů
  - Je možná distribuovanost
- Výkonný middleware s podporou flexibilního uživatelského rozhraní a s dalšími funkcemi o které se může dělit s komponentami (kryptografie, směrování, optimalizace polohy komponent,…)

Vlastnosti konfederací

+ Lze dohodnout pravidla šitá na míru
  - formát zpráv může být deklarativní, uživatelsky orientovaný – *klíčová přednost*, snazší specifikace a snazší integrace uživatelů do procesů, snazší on-line změny obchodních procesů (agilita)
  - lze použít i jiný než webovský middleware, lze kombinovat různé technologie middlewaru i komunikace
  - služby mohou zahrnovat i služby reálného světa (řízení procesů, uživatele), ?doba odezvy?, na webu složitější

Vlastnosti konfederací

+ Lze dohodnout pravidla šitá na míru, uživatelsky orientovaná
  - lze snadno kombinovat dávkové a interaktivní zpracování
  - lze kombinovat strukturované/funkcionální a objektové techniky
  - snadná integrace produktů třetích stran a existujících aplikací
  - snadné modifikace a výhodné SW inženýrské vlastnosti
  - k dokumentaci často stačí popis rozhraní služeb
    - výhodné pro agilní procesy vývoje

Vlastnosti konfederací II

++ Služby mohou být i informační systémy jednotlivých organizací a organizačních jednotek nebo nakupované/vyvíjené aplikace a dokonce přímo technologie či jednotliví pracovníci (za vhodným rozhraním), výhody

- + snazší decentralizace, prodej/akvizice částí podniku
- + nástroje kontroly spolupráce částí
- + snazší spolupráce s obchodními partnery (CRM, SCM, nákupní koalice..) a se státní správou
- + podpora strategických záměrů managementu

Implementace podnikových systémů ve formě aliance je riskantní, neefektivní a pracná (proto raději např. enterprise service bus)

Přednosti a nevýhody SOA

- Neví se, je-li vůbec možná model driven architecture, chybějí příklady řešení a CASE nástroje pro SOA
- Problémy s proprietálními řešeními, poněvadž pro ně nelze vždy použít standardy
  - lze zmírnit správnou strategií (užití XML)
  - může urychlit vznik uživatelských XML jazyků a tím nevýhodu změnit na výhodu

+- Nutná úzká spolupráce mezi skoro všemi vývojáři a mnoha uživateli i z nejvyšších postů (CEO se musí účastnit specifikací toho, co bude používat)

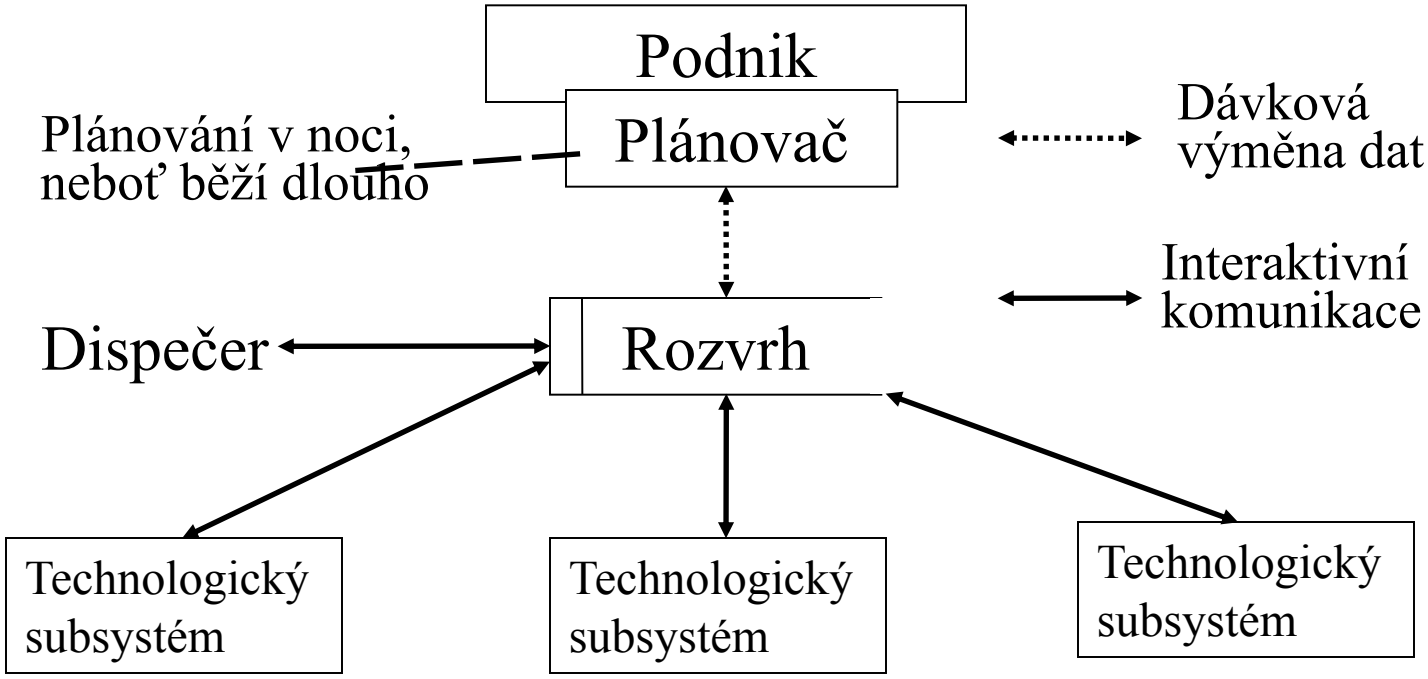
++ Velmi dobré zkušenosti s provozem (dvacet let bez údržby, viz též zkušenosti s Y2K)

Pozorování

- Některé akce musí být dávkové (trvají příliš dlouho), to platí pro některé algoritmy, pro lidi a pro procesy reálného světa
- Jsou třeba zásahy dispečera, tj. je třeba agilní provádění procesů
  - Nečekané nebo vzácné události –
    - nevyplatí se je zahrnovat do rozvrhování (Vonásek je lempl, Pepa se včera ztřískal)
    - Turbulence na trhu nebo u dodavatelů
  - Kvalita dat
    - Nedostupná, neznámá, nepřesná (mají velký rozptyl)
    - Zřídka potřebná (nevyplatí se sbírat)
  - Potřeba využít inteligenci lidí jako součásti procesů

Neinteraktivní komunikace

Spolupráce plánovacích algoritmů s provozem vyžaduje inteligenci při přenosu požadavků

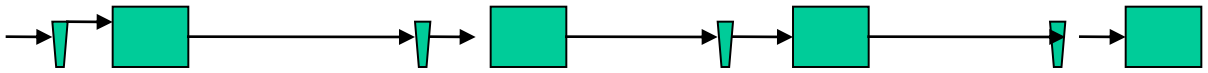


Další důvody, proč je třeba dispečer

- Data jsou nevčasná, neboť plánovací algoritmy jsou pomalé, změny se musí přesto udělat rychle, tj. musí je udělat člověk, aby výroba či obchodní proces nestály, když se objeví nečekaná překážka
- Data jsou nepřesná $\Rightarrow$plán je nedokonalý

Řízení na buffery

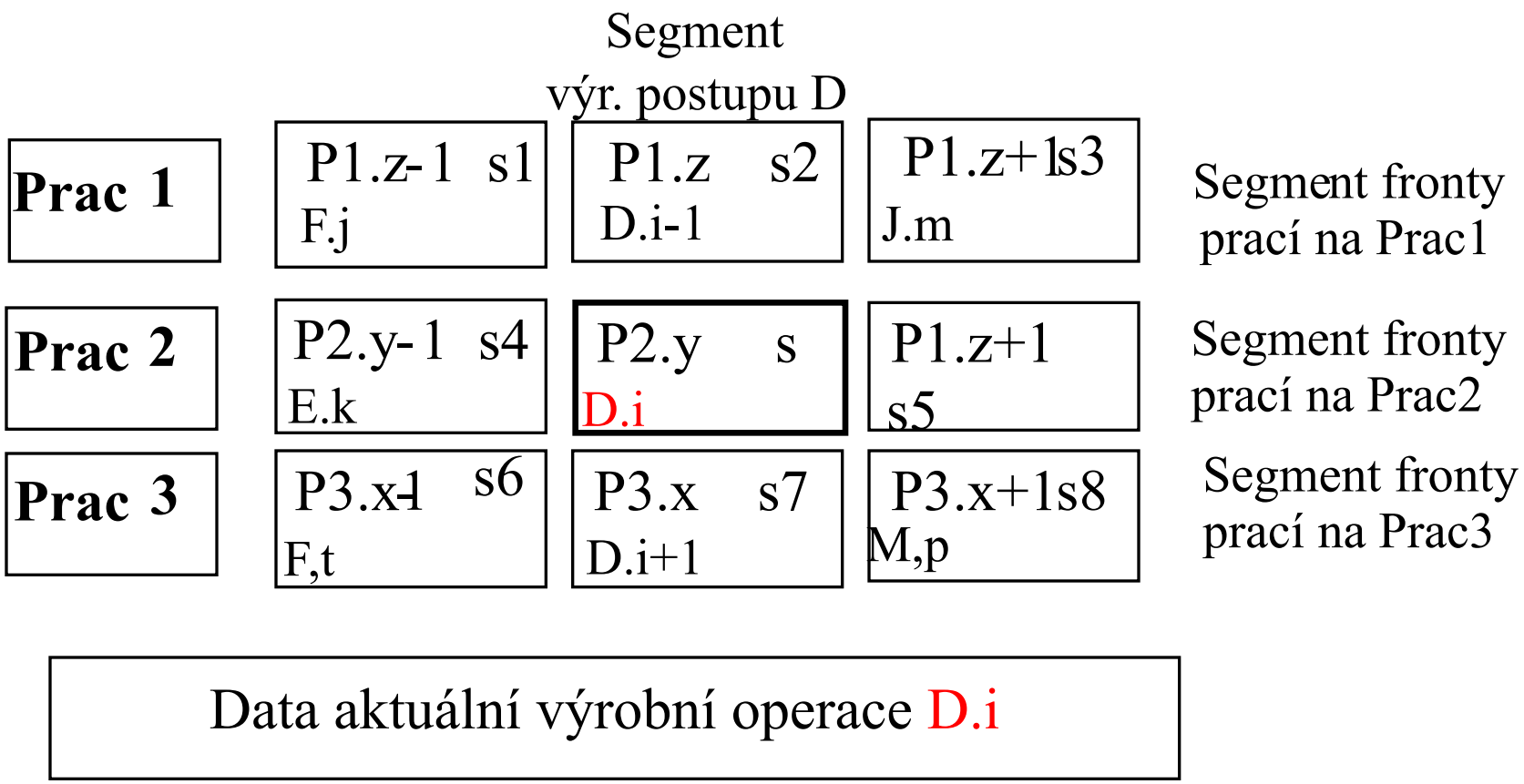
- Před každým pracovištěm fronta požadavků (buffer požadavků)



Mistr sleduje buffery a hledá pro dané pracoviště práci, když se jeho bufer nepříjemně zkracuje (řízení na průšvih, ale na průšvih, který nastane v budoucnosti). Mistr nemusí zasahovat, když je plán OK

Nutné pro výroby s krátkými sériemi a velkou variabilitou

Prostředek sledování postupu prací



Pozorování 2

- **V daném příkladu neběží jen o poskytování informací, předávají se i příkazy, služby mohou být služby reálného světa (raketové základny)**
- **Datově bohaté rozhraní je typické pro manažerskou úroveň řízení, ta se stává hlavním tématem současnosti**
- Snadné kombinování ručních a automatizovaných postupů je nejvýše žádoucí
- Vysoká autonomie služeb
- Částečný návrat DFD a datových úložišť (používání dříve zavrhované funkční dekompozice)
- Podle současných znalostí je implementace rychlého datově bohatého rozhraní v aliancích obtížná, i když možná

Pozorování 3

- Datové úložiště může být virtuální (bez replikace dat), data poskytovaná službami mohou být čtena z jejich databáze, vypočítávána, měřena, zadávána uživateli – *klasická DB nemusí být adekvátní koncepcí (problém indexace), viz sémantický web*
- Důležité je deklarativní hrubozrnné uživatelsky orientované rozhraní
- Úložiště může být plněno z více zdrojů současně (lze použít předřazené brány diskutované níže)
- Výhodné dávkové plnění dat
  - ušetří to mnoho práce při vývoji a údržbě, stabilita řešení

Další výhody SOA

- Paradigma typické pro služby v lidské společnosti, je proto srozumitelné uživatelům
- Jedná se o jiné paradigma než objektová orientace nebo vzdálené volání procedur (kombinace s těmito paradigmaty je možná, pokud se koncepce používají jako ortogonální nástroje, často ale vede k nadměrnému počtu malých služeb – antipattern fine grained services)
- Lze poměrně snadno vylaďovat spolehlivost (doručitelnost zpráv), zabezpečení, kontrolu průběhu, efektivnost, rozhraní a umožnit ruční zásahy do průběhu procesů
- Poměrně snadné je využívání aplikací, které se nakupují, existují nebo jsou téměř nezávisle vyvíjeny

Doporučení, uplatnit, pokud lze

- Použít SOA a XML, možná SOAP
  - To ale spíše ve formě SOAP message encoding resp. literal
- Raději komunikace přes datové úložiště a to raději neinteraktivní (bez trigerů) není-li na závadu (často nebývá)
- Používat i dávkové zpracování, šetří to náklady, vyšší stabilita
  - Jde použít častěji, než se má za to

Doporučení, uplatnit, pokud lze

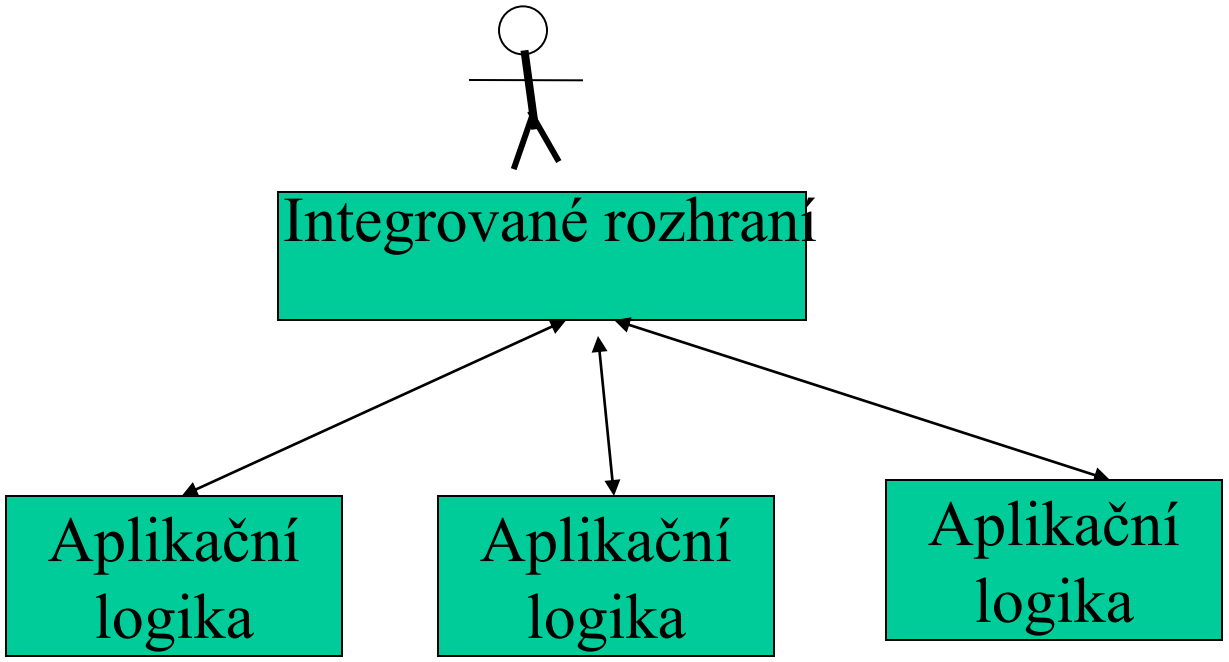
- Uživatelsky orientované rozhraní, hranice služeb obdobná hranicím služeb v reálném světě
- Uživatelé by měli být schopni snadno měnit obchodní procesy během provádění
- Umožnit simulaci služeb uživateli
- Uživatelé by měli být schopni analyzovat žurnál komunikace pro kontrolu a pro případné soudní řízení
- Maximálně využívat znalosti obsluhy

OLAP, Excel a snad i datové sklady používat jako služby

Klíčová role rozhraní

- Stabilita konfederace závisí především na stabilitě (neměnnosti) rozhraní
- Jazyk rozhraní by měl být vysoké úrovně (deklarativní)
  - Není to výhodné pro dodavatele, není vendor lock in
  - Problémy se standardy
- Rozhraní by se mělo chovat podobně jako rozhraní služeb v reálu (být uživatelsky orientované)
  - Být srozumitelné uživatelům, mělo by být podobné tomu, nač jsou zvyklí; je to nutné pro návrh a kontrolu průběhu business procesů a jejich modifikace on-line
  - Takové má šanci nebýt měněno, je vlastně ověřeno dlouhodobým používáním
  - To je snažší, je-li deklarativní

Hlavní problém uživatelského rozhraní: Potřeba skládat dokumenty



Řešení bylo nalezeno také v jazyce XML a nástroji XSLT

Textový

Dovoluje rozšiřování

Jsou dostupné prostředky pro skládání a transformaci dokumentů (XSLT)

Trochu problémy s efektivností a spolehlivostí

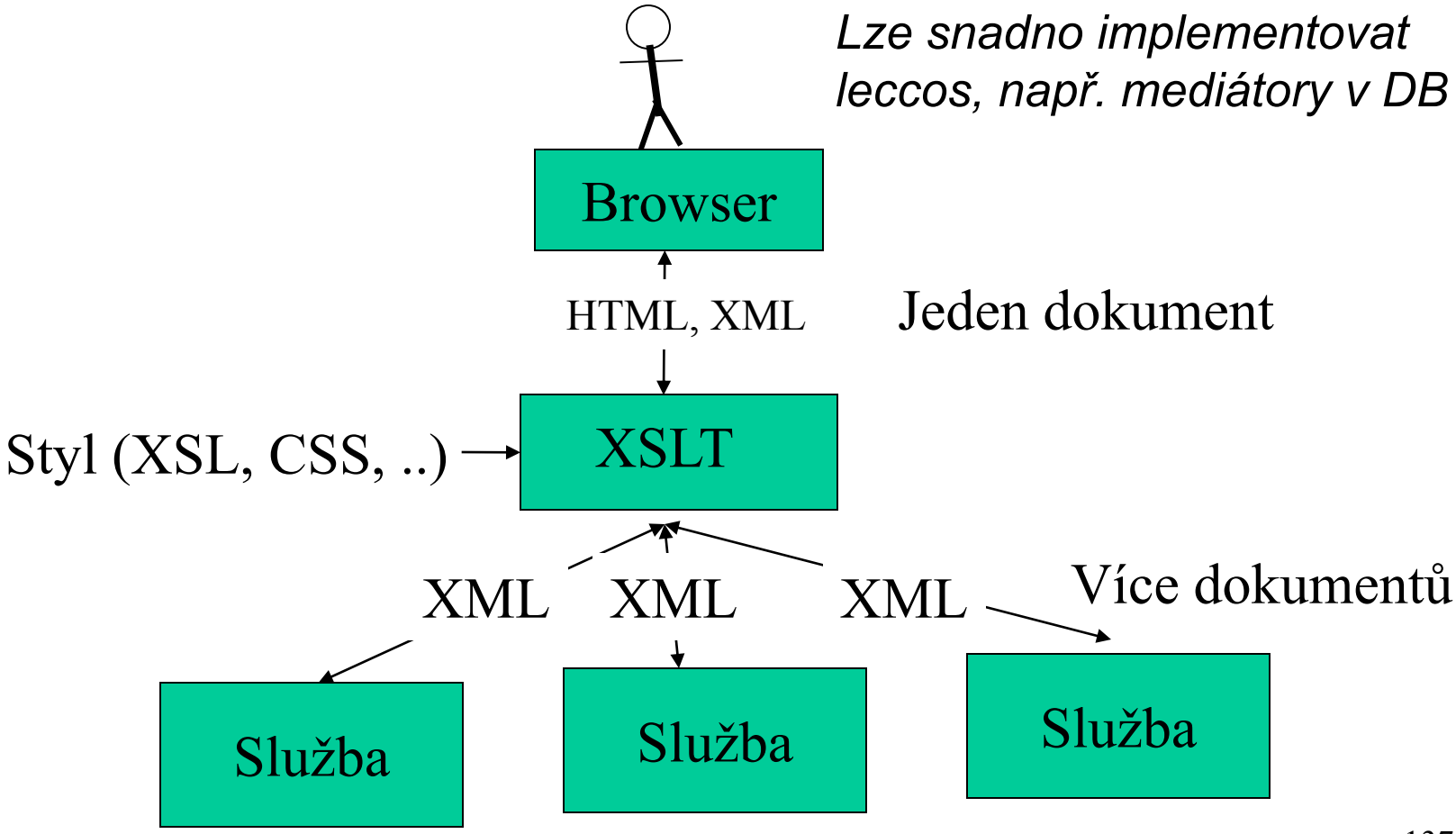
Dá se použít i při programování uživatelského rozhraní

Funguje to !!!!

Norma Portlet

Nebezpečí Babylonské věže!!

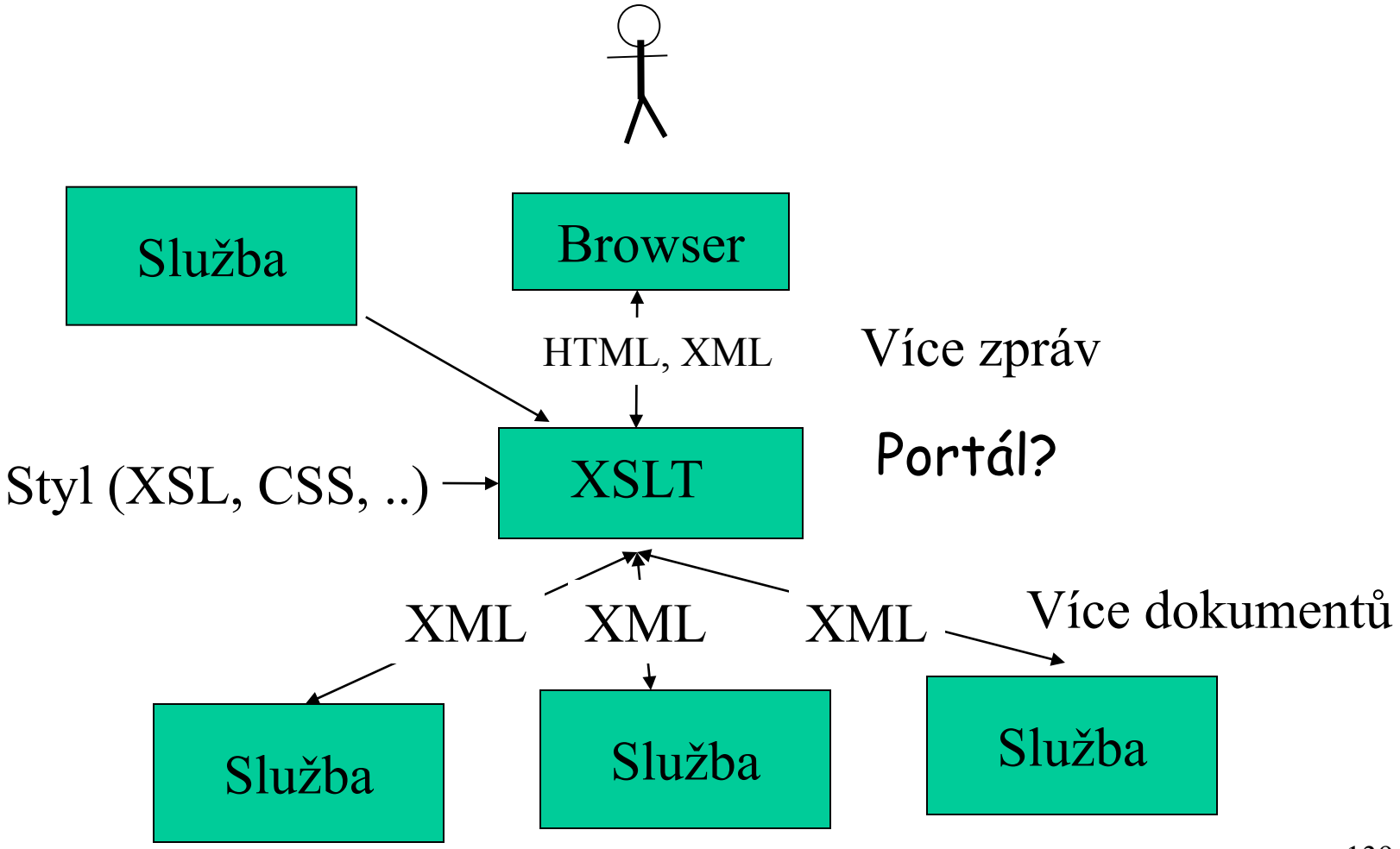
XML a uživatelské rozhraní srv též SAP



XSLT lze použít tak, že může transformovat v jednom kroku m vstupních zpráv na n výstupních zpráv

Takový aparát můžeme chápat jako zobecnění míst (places) v Petriho sítích s barevnými značkami
Problém: XSLT se moc neosvědčilo

XML a uživatelské rozhraní



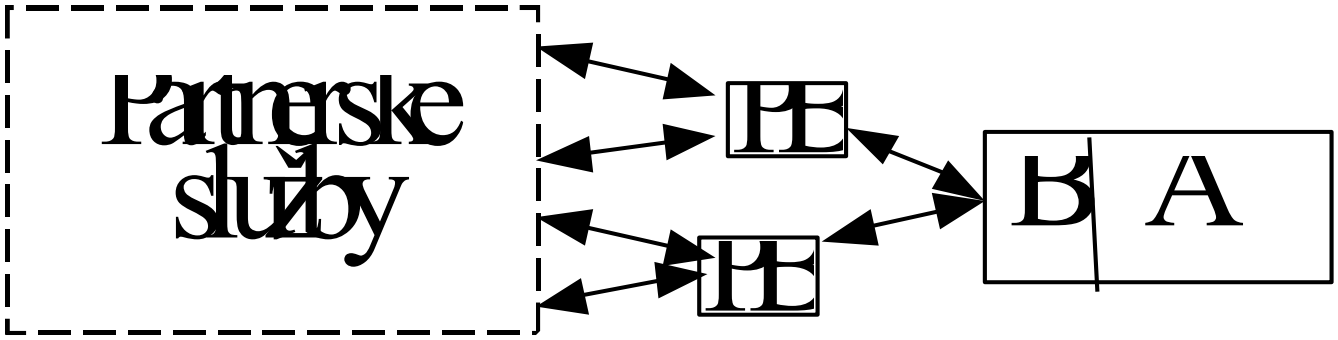
V čem je XML s podpůrnými nástroji důležitý (nutný)

1. Rozšiřitelnost, přijatý standard
2. Vhodný pro deklarativní rozhraní mezi aplikacemi (pravděpodobně nejdůležitější)
3. Schopnost transformovat a skládat dokumenty a schopnost definovat a využívat styly pro UI (nutné i pro práci s heterogenními databázemi)
4. Schopnost definovat metadata a schopnost tvořit dotazy a specifikovat semistrukturovaná data, schopné podporovat i *datový pohled*. Velmi slibné, nejméně významné (zatím)

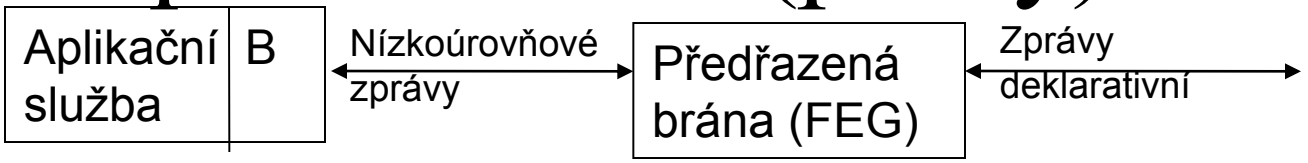
Techniky 1
Zlepšení stability rozhraní

- Existující aplikace nemusí mít vhodné rozhraní
  - Nutný odposlech uživatelského rozhraní aplikace, aplikace lepší bránu nemá
    - Snazší má-li třívrstvou architekturu
  - Rozhraní není uživatelsky orientované (je např. OO, RPC)
- Různí partneři požadují různé rozhraní -

Řešení: Infra služba jako přeřazená brána

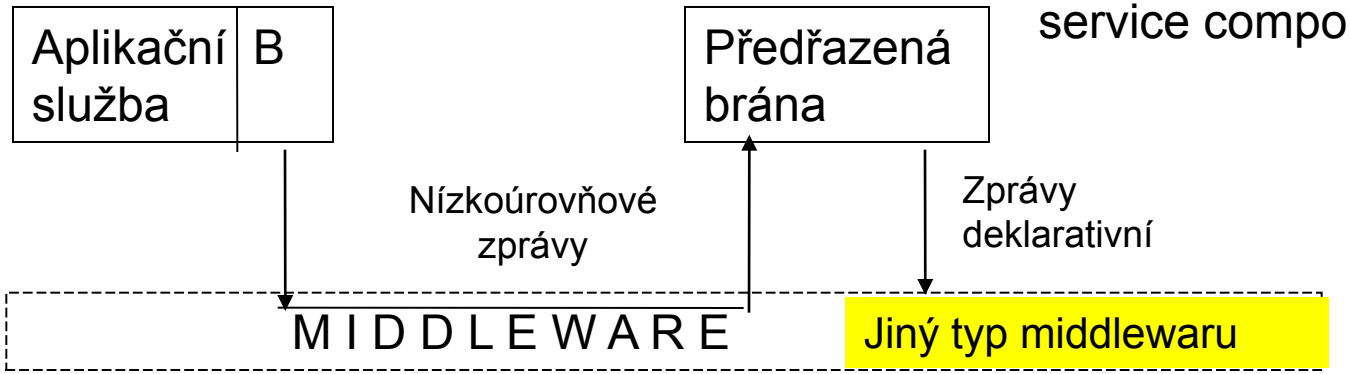


Logický a implementační pohled na předřazenou (proxy) bránu



a) Logický pohled

B se často implementuje jako API komponenta, zvaná service component

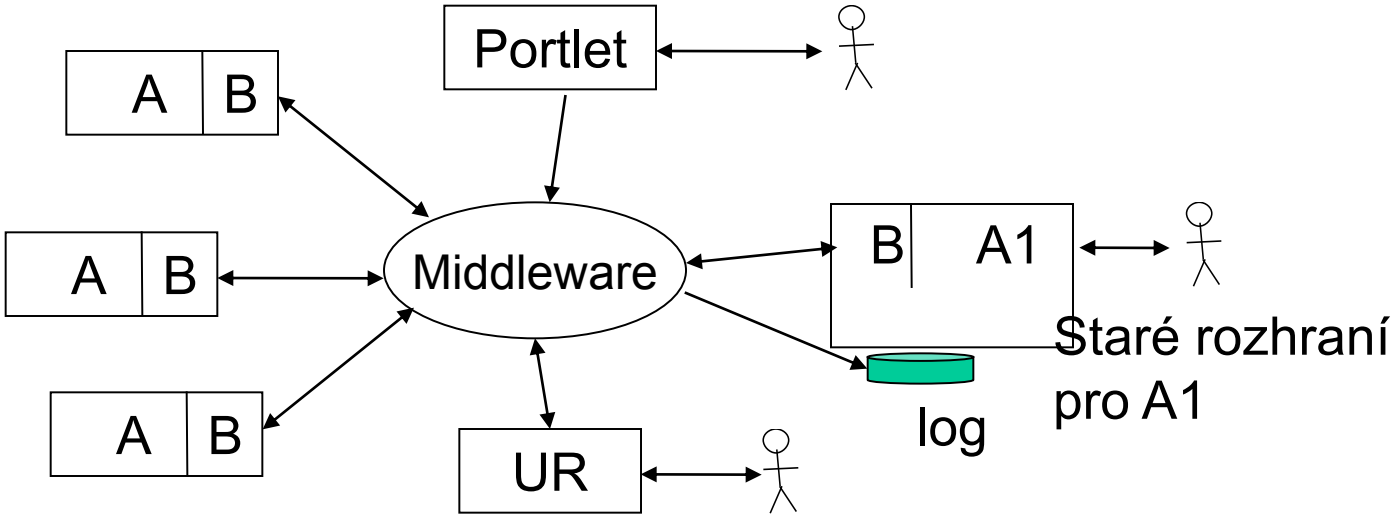


b) Implementace

Např. roura

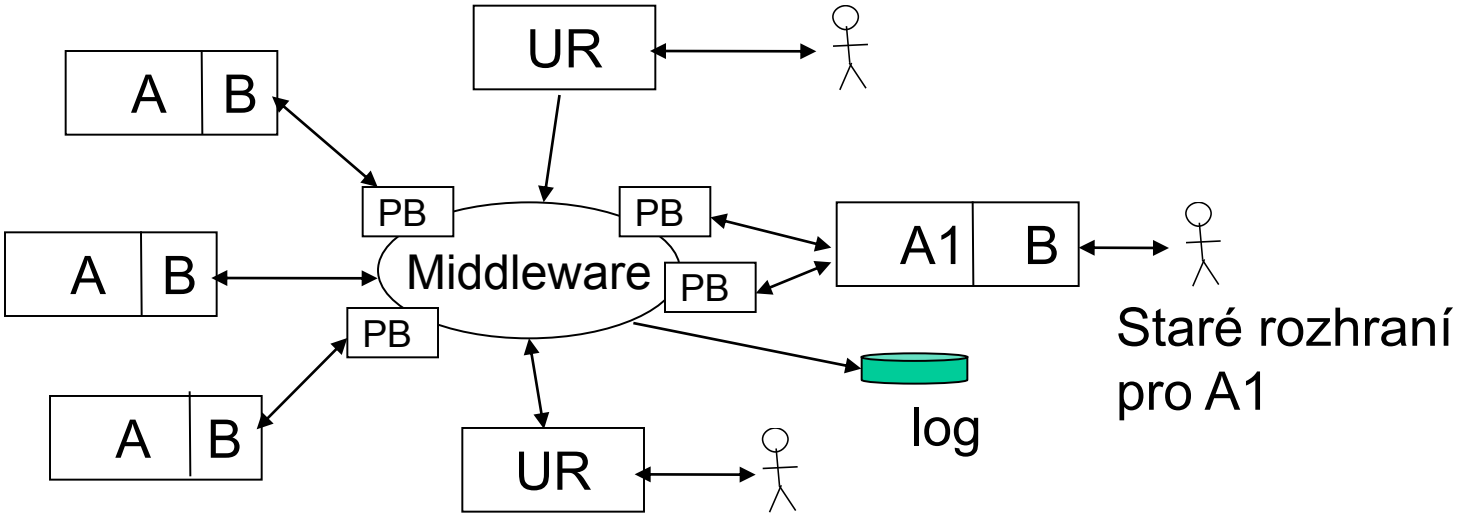
?? Simulace posílání zpráv pomocí rour a filtrů

SOA Konfederace (partneři se znají)



Nástroje na programování přeřazených bran jsou stejné jako u portálů/portletů

SOA s přeřazenými branami, konkrétní propojení



PB Předřazená brána, může jich být pro danou službu více, mohou i chybět. Pozor! B a PB mohou případně komunikovat přes internet, PB je příklad konektoru jako **služby.PB mohou zajistit, aby midleware komunikoval zprávy v jednotném formátu (viz ESB)**

Jak implementovat brány

1. Integrovat do komponent
2. Jako zcela plnoprávné služby
3. Integrovat do middlewaru
4. Kombinovat přístupy (dvoustupňové rozhraní, vnitřní brána - wrapper a předřazená brána PB)

Konfederace s přeřazenými branami

- Podobná řešení pro web services
  - Mediator (www.wsmo.org), není zamýšlen jako obecná služba mimo rámec web services a objektové technologie
  - Sessioners
  - Fasády v design patterns pro OO
- Nástroje XSLT
  - Ale má zatím chyby, neefektivita, obtíže s používáním

Předřazené brány (PB) jako místa v zobecněných Petriho sítích a barvenými značkami

Techniky vývoje PB jsou prakticky stejné jako u implementace uživatelského rozhraní. Lze použít XSLT a transformovat *m* vstupních a *n* výstupních zpráv. PB (a UR) lze proto chápat jako zobecněná místa v barvených Petriho sítích

Jiný pohled na SOA , když jsou známi partneři (konfederace)

Cooperative commerce (c-commerce)

Assembled applications

Composite applications

Lze služby chápat jako komponenty z OO, není to ale výhodné.

Nejasné jaká je nejsnazší implementace (možných je několik)

a uživatelsky nejsnazší řešeni

Konfederace pro strojovou byrokracii (podniky) jsou úžeji vázány, lze v tomto případě požadovat větší úpravy komponent a použít pak standardy. Pro takové systémy nemusí být přeřazené brány službami a mohou se implementovat jako rozšíření služeb, tj. splynou s branami uvnitř služeb kde se navíc integrují s doplňky

pro Middleware

Pro takové případy je výhodné použít prostředí Enterprise Service Bus řady výrobců

SAP používá konkurenční koncept NetVeawer

N^2 formátů a service bus

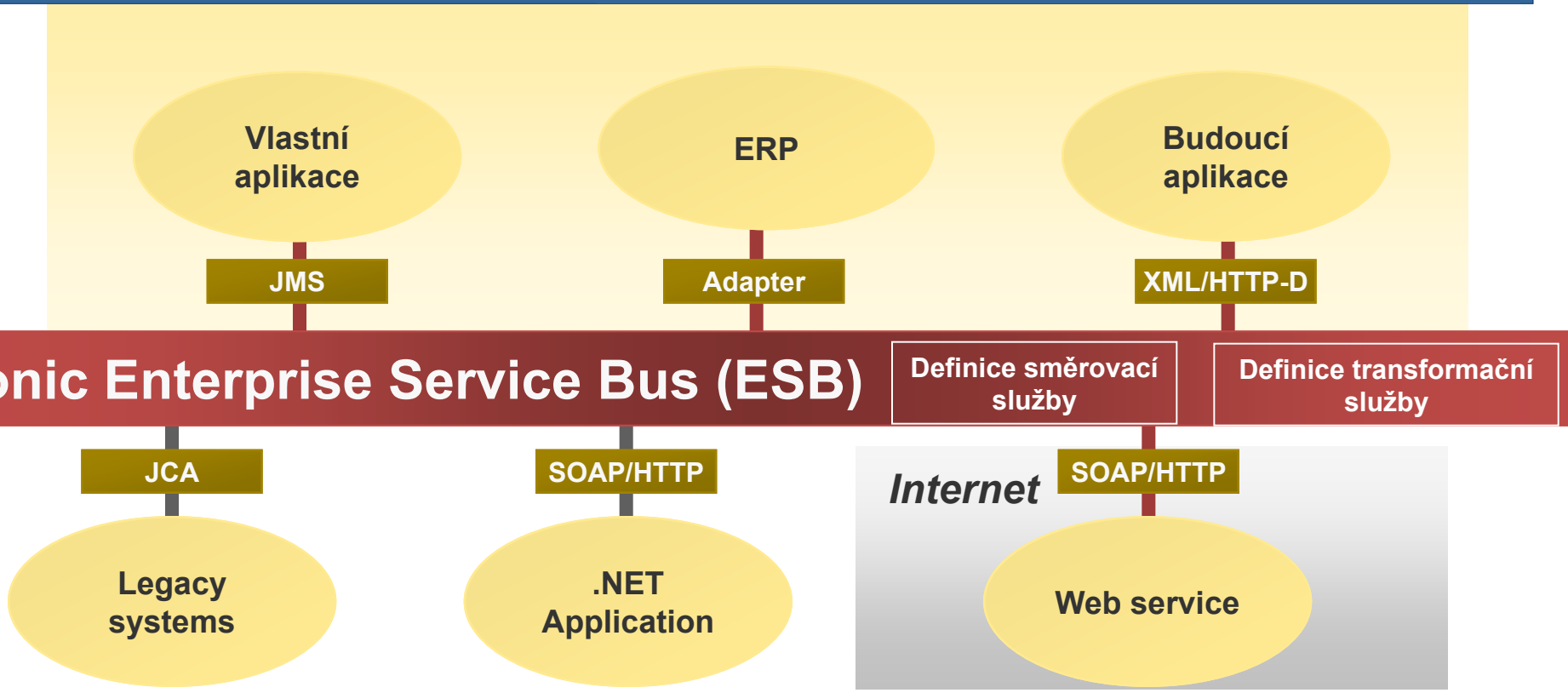
Komunikace může obecně pro N služeb vyžadovat až N*(N-1) transformací zpráv. Řešením může být jediný formát pro zprávy mezi všemi předřazenými branami. To je základní idea řešení Enterprise Service Bus od Sonic Software, nyní Progress Software.

Zprávy mohou být přenášeny mnoha způsoby (signály, mail, telefon, www, …)

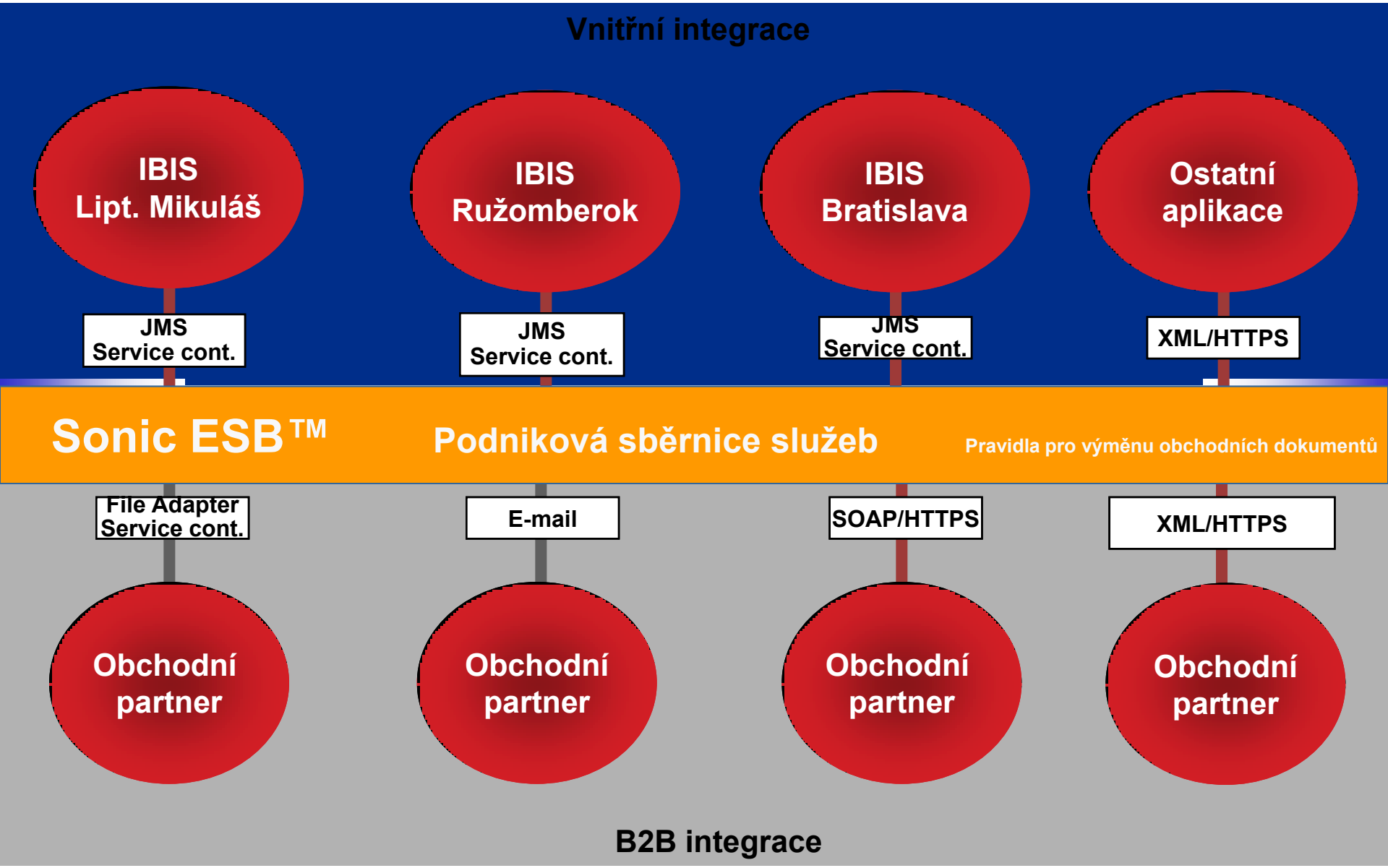
Stumpf Jindřich, Progress Software, Systémová integrace 2004, příklad

"Do roku 2005 bude *Podnikovou sběrnici služeb* (ESB) kombinující messaging, webové nebo i jiné služby, inteligentní směrování a transformaci dat, modelování a řízení business procesů, používat většina podniků."

Roy Schulte, VP and Research Fellow, Gartner Inc.

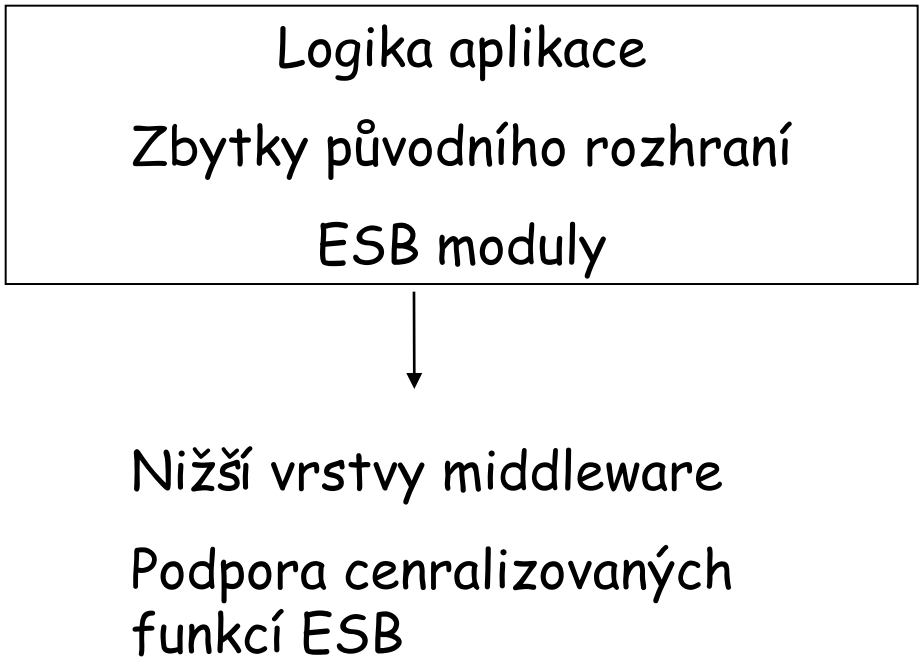


Stumpf Jindřich, Progress Software, SI 2004, příklad konkrétního projektu



Jak se použije ESB

- Nakoupí se knihovna zásuvných modulů
- Aplikace dostane s ESB tvar



Co v daném případě přinesl ESB podle Progress SW

- Skutečně se téměř nemuselo do existujících aplikací výrazně zasahovat
- Bylo to velmi rychle hotovo
- Plná spokojenost uživatele
- Dodavatel ale vlastně realizoval dodávky spíše menší velikosti. Dříve by taková dodávka byla podstatně větší
- Bylo méně práce i pro vývojáře (pro ty dramaticky) a tedy i menší výnos

Hrozby a příležitosti

- Snížení pracnosti desetkrát proti dřívějším zvykům → desetkrát menší příjem, desetkrát méně programátorů
- Je ale možné navrhnout nové služby a ty vyvinout → více programátorů
- *Není jasné, který trend převládne*

Co to ale stojí

V daném okamžiku není jasné, zda u ESB nehrozí podobné problémy, jako je tomu u aliancí, tj. převodníky mohou vyvíjet pouze programátoři a nikoliv koncoví uživatelé. Hrozí i problémy s porozuměním komunikaci a při implementaci nástrojů pro ruční zásahy.

Řešením jsou architekturní služby

Jak na SOA/ESB

Které middlewarové funkce nakoupit a které si vyvinout sám?

- Ne vše, co se nabízí, je vhodné a leccos úplně chybí (např. datová úložiště, uživatelsky orientovaná rozhraní)
- Co integrovat do aplikačních služeb,obvykle do bran
  - Adaptér- objekt, ne vždy rozumně lze, datové úložiště
- Co do služeb podporující integraci (dále architekturní služby)
  - Adaptér - služba
- Někdy je i možnost změn programů implemetujících část middlewaru

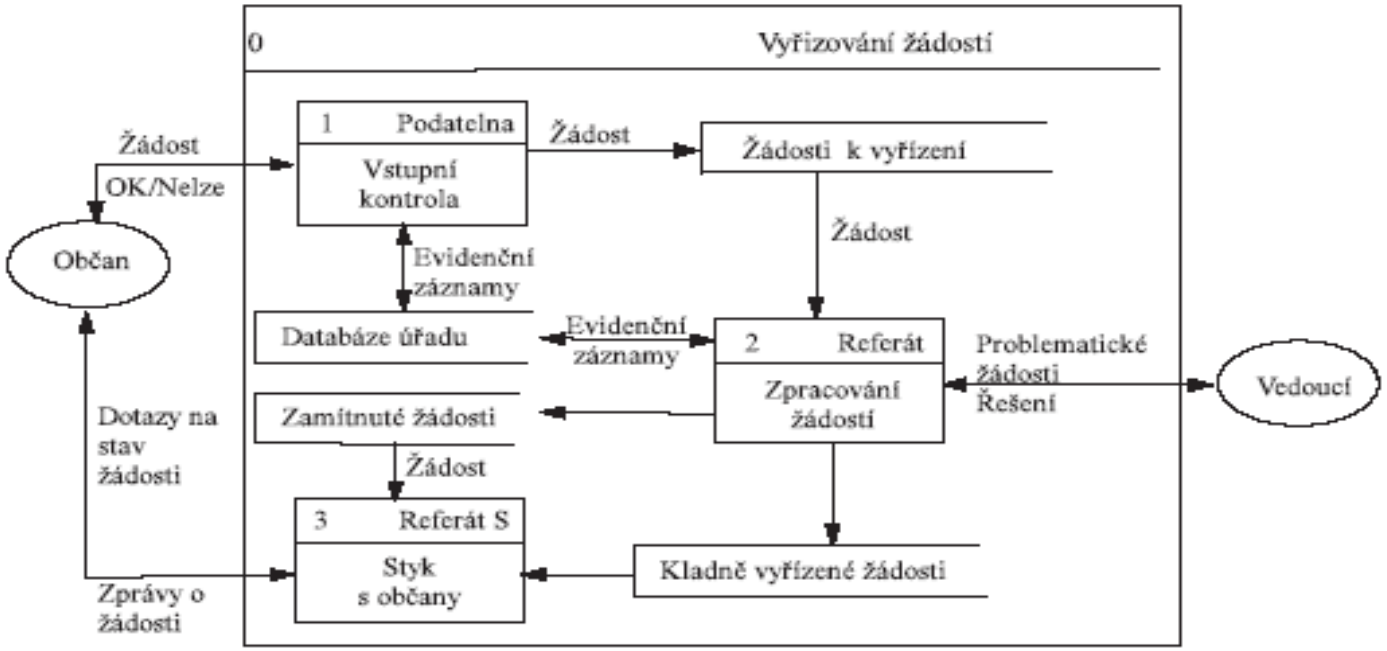
Architekturní služby

- Nejsou to API, jsou to služby (autonomní až nezávislé SW komponenty)
- Je to nezávislá dimense vývoje SOA
- Uplatní se hlavně při integraci legacy systémů integrovaných jako černé skříňky
- Architekturní služby se používají i v cloud řešeních
- Architekturní služby jsou konektory jako služby

Interaktivní a dávkové služby (opakování)

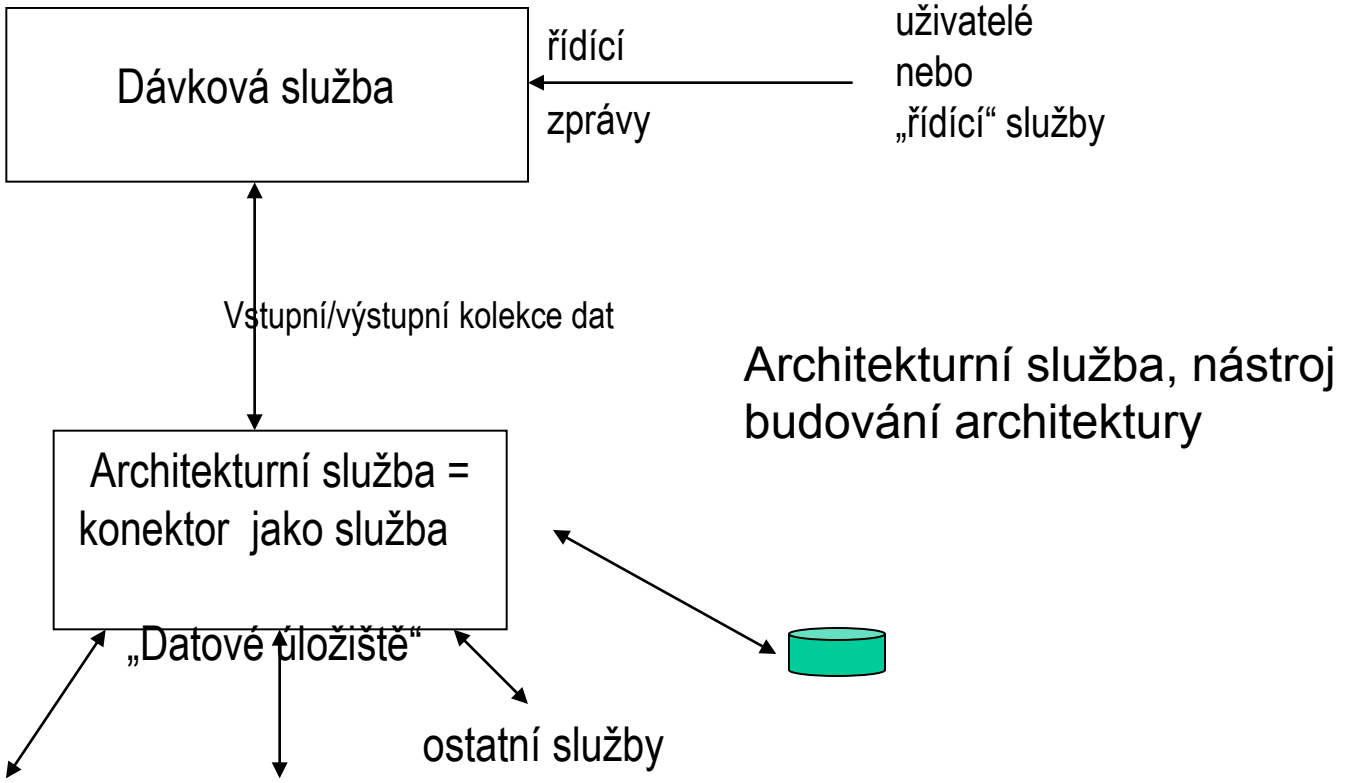
- Většina komponent v SOA komunikuje prostřednictvím ne extrémně dlouhých zpráv a doba provedení požadavků zpravidla není příliš dlouhá
- Někdy je ale vhodné/nutné použít nebo znovupoužít komponenty, pro které to neplatí a musí pracovat dávkově
  - Algoritmy rozvrhování, programy spolupracující s reálným světem, čištění dat, transformace finančních dat
- Někdy je vhodné použít osvědčené dávkové programy, nebo je dokonce znovu napsat včetně používání DFD
  - Snazší vývoj, úspory díky znovupoužití
  - Větší spolehlivost
  - Větší stabilita (zkušenost z Y2K)

Řešení – část SOA jsou dávkové programy komunikující tak, jak předpokládá digram toku dat



Obr. 12.9: Diagram toků dat systému Vyřizování žádostí.

Integrace dávkového programu do SOA



Změna protokolu komunikace

- Datové úložiště pro zprávy
- Politika jejich vybírání, zpracování a odesílání

Kdy jsou potřeba komunikační služby s datovými úložišti

1. Služba běží příliš dlouho nebo produkuje velké množství dat (př. plánování a rozvrh výrobních operací), nebo je prostě dávková
2. Je žádoucí použít existující dávkovou službu
3. Implementace složitých variant komunikace (obsluha částečně záměných pracovišť)
4. Snazší a stabilnější implementace nebo snazší sourcing
5. Decentralizace repositářů (globálních služeb)
6. Úložiště může být použito pro určitý typ uživatelů, kteří mají specifické principy na hodnocení souhrnné kvality souborů dat, obsahujících podmnožiny dat různé kvality

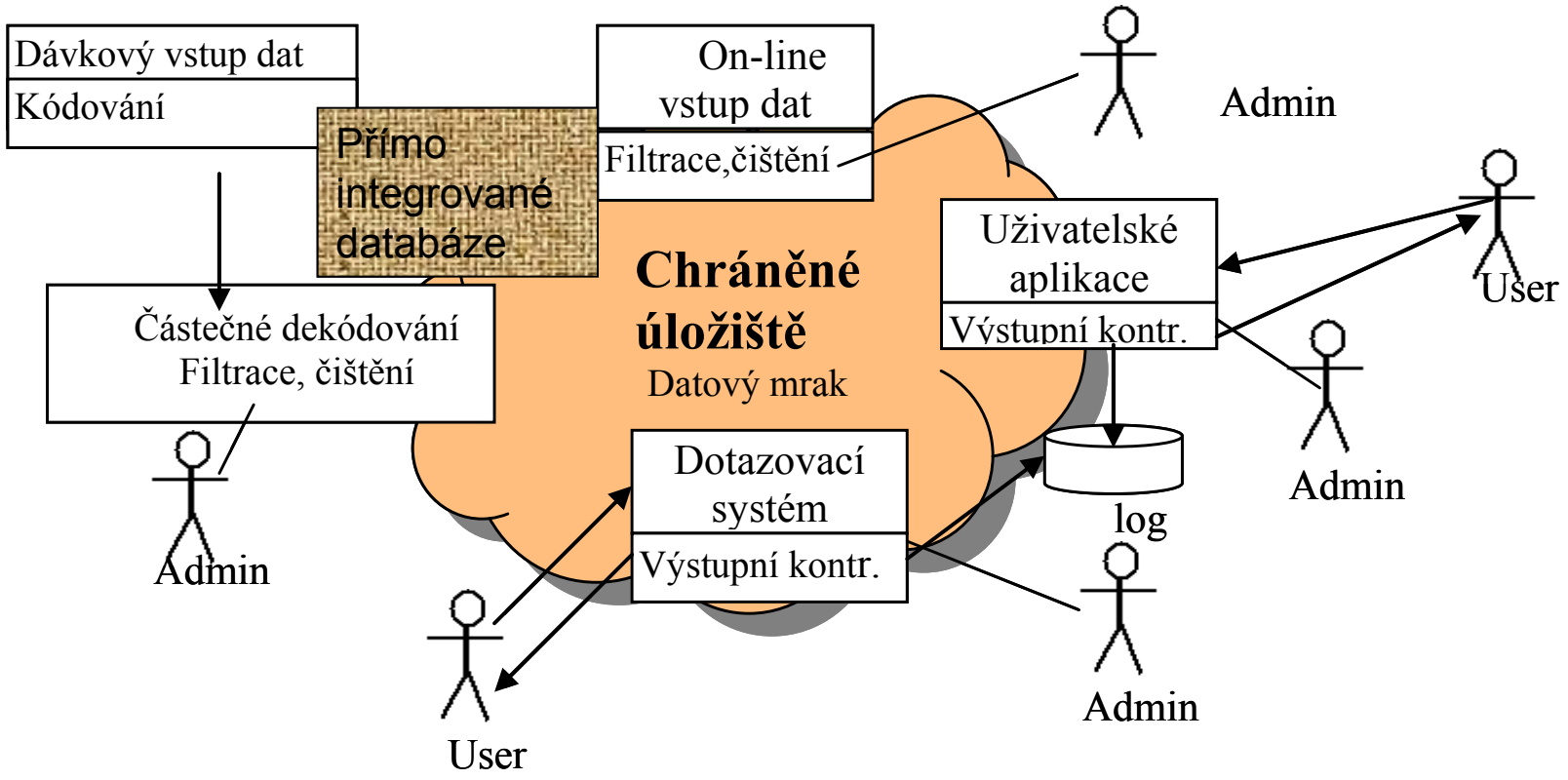
Možná struktura SOA v sémanticky orientovaném prohledávači

- Jádro systému realizovat jako konfederaci
- Klasický vyhledávač jako jedna ze služeb
  - problém je že to vyžaduje komunikaci jako proud dat
- Uživatelské rozhraní jako jedna ze služeb (slouží jako portál)
- Služby: inferenční stroj, databáze ontologií, dotazovací systém (jak zkoordinovat s vyhledávačem? Proudy dat?)
  - Služby mohou mít vlastní data a vlastní rozhraní, které může být přesměrováno na portál systému (ale nemusí)

Nedostupnost dat a řešení pomocí dávkové služby v SOA

- V e-gov ÚOOÚ de facto znepřístupňuje všechna (citlivá) data
- Zveřejnitelná informace je pro zjišťování ze strany občanů nedostupná, jeli k jejímu výpočtu potřebný citlivý údaj
- Přeceňují se případné škody z prozrazení dat a nedoceňuje ztráty ze ztráty informací
  - Je mnoho kanálů, kterými data unikají
    - Mobily, banky, sociální sítě, registrace na internetu, …
  - Současná praxe je cosi jako zákaz v podniku používat operativní data pro budování byznys inteligence

Zlepšení dostupnosti dat a SOA



Časté zákazy pořizování dat a omezení na jejich využívání, neví se jak dělat výstupní filtrace, nehodnotí se ztráty vyvolané znepřístupněním dat

Co s centrálními repositáři

- p2p se ve velkých systémech nesnáší s centrálními službami
  - Neúspěch UDDI jako celosvětových repositářů
- Co s repositáři, které např. potřebuje business intelligence
  - Repositář byznys procesů?
- Řešení: Repositář může být decentralizovaný a nemusí být v konkrétním případě použit
- V menších systémech centrální repozitář nevadí

Business proces v SOA

- Síť kroků
- Krok je provedení příkazu nějakou službou (i sítí služeb, kompositní službou)
- Je vhodné si pamatovat průběh provedených procesů, resp. modelovat proces
  - Neměl by být centrální repositář modelů procesů využívaný v průběhu celého procesu (důsledek p2p architektury), raději distribuovaná databáze
  - Potřeba využívání různých prostředků popisu procesů (volný text, BPEL, Aris, Petriho sítě,….)

Pragmatika

Propojování může být založeno na různých technologiích, některé může být určeno k propojování na daném stroji, jiné je možné použít globálně mezi různými stroji či sítěmi

Business procesy

- Mají stav (vlastní data, stav řešení).
- Jejich průběh musí být vlastníkem procesu měnitelný i „za běhu“, musí být agilita
  - Nereálné provádět uvnitř aplikací, které jsou dávno odladěné a používané jako černé skříňky
- Je nevhodné řídit proces v rámci centralizované služby (důsledek p2p architektury)

Kde mít řídící data

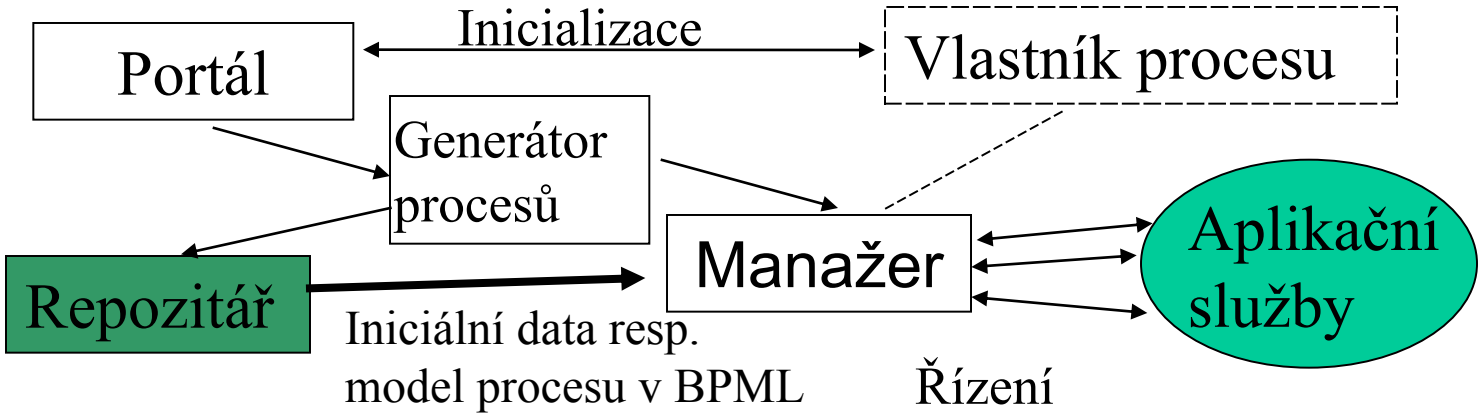
1. Nikde – služba ví s kým kdy co
   - Není možno sledovat průběh
   - Změna procesu $\Rightarrow$ zásah do komponent, ale to jsou obvykle černé skříňky
2. V předávaných zprávách (rozdělovníky)
   - Pružnější, explicitní řídící data
   - Obtíže při sledování
   - Jednotný formát, větší změny $\Rightarrow$ zásah do komponent

Kde mít řídící data pro BP

1. V centrální databázi
   - Implementační potíže chceme-li různorodá data a snadnou přístupnost
   - Neefektivní, obtížné sledování, centrální entita v p2p systému
2. Ve službě vytvořené pro každý proces znovu.
   - Lze snadno sledovat a implementovat různé modely. Není to levné

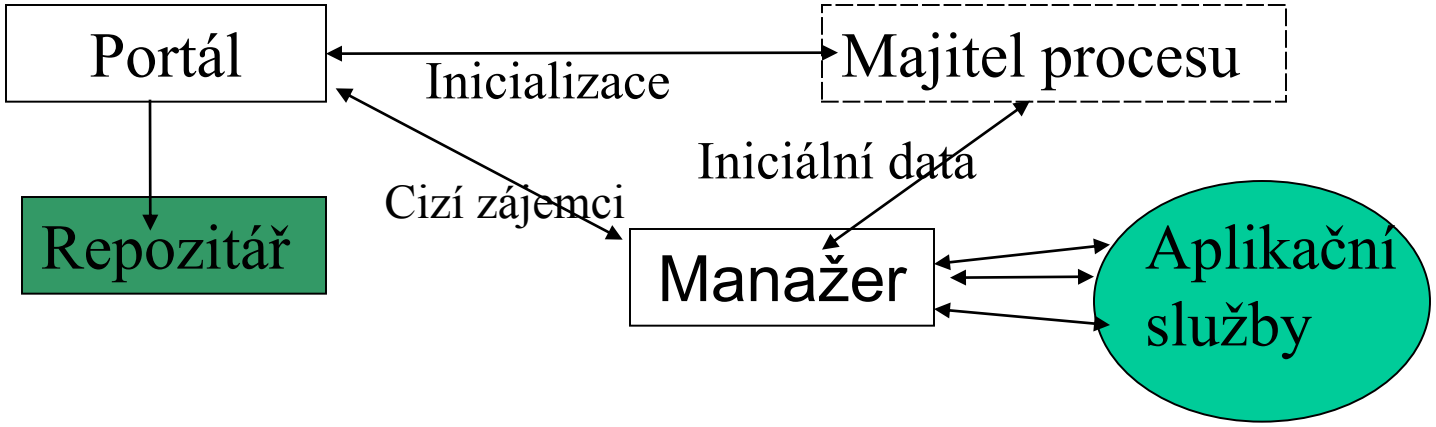
Business procesy

- Řešení:*Specializovaná služba **manažer** pro každý proces, simuluje prvky portálu, iniciální data z repozitáře*



Manažer se vytváří pomocí generátoru procesů s případným využitím modelů procesů uložených v repozitáři pro každý proces znovu, manažer je služba která se chová jako instance procesu

Business procesy



Řešení s využitím standardů popsaných v www.bpmi.org

- Modely procesů v BPML (dialekt XML)
- Data manažeru v BPML (XML), upravují se podle parametrů inicializace zadané vlastníkem procesu, tím vzniknou řídící data procesu, např. v BPEL

Manažer odpovídá instanci procesu podle BPML. BPML ale neuvažuje o implementaci jako o službě

Business procesy

Řešení s využitím standardů popsaných v www.bpmi.org není jediné možné. Další varianty:

- Popisy pracovních toků (workflow, BPMN)
- Použití activity diagrams z UML nebo diagramů s použitím IDEF 3.0
- Jednoduchý fulltextový popis se zaznamenáváním provedených akcí/kroků

! Často je nutné používat všechny čtyři varianty!!!!

- Obtížně se implementuje v centrální databázi u velmi rozsáhlých systémů(obecný problém centrálních DB v rozsáhlých p2p systémech)

Uživatelsky orientované rozhraní služeb

- Výše uvedené požadavky na podnikové procesy je možné splnit, jsou-li rozhraní služeb uživatelsky orientované - srozumitelné pro vlastníky procesů a pokud možné blízké zavedeným způsobům zadávání prací v reálném světě.
  - To je důležité i pro případné soudní spory a je to i podmínkou, aby bylo možné požadovat odpovědnost vlastníků procesů za případné škody
- Uživatelsky orientované rozhraní má tendenci být deklarativní (pak skrývá i implementační detaily a to je významná výhoda)

Inženýrské výhody uživatelsky orientovaných rozhraní, opakování

- Uživatelsky orientované rozhraní má tendenci být
  - Deklarativní (pak skrývá implementační detaily služeb což je známo jako významná výhoda pro modifikace)
  - Stabilní v čase (bývá založeno na prověřených znalostech a postupech)
  - Sémanticky bohaté (snižuje nároky na komunikační kanály)
  - Snadno použitelné při definici obrazovkových prototypů
- Nevýhodou je, že nebývá standardizováno
  - Může být i výhodou. Předčasná standardizace bývá nedokonalá a může proto představovat značný problém (norma se nehodí na daný úkol, často se mění, náročná na použití)

Inženýrské výhody uživatelsky orientovaných rozhraní

- Uživatelsky orientované rozhraní a autonomie komponent zvětšuje možnost aplikace principů agilního vývoje
  - Stačí dokumentovat jen rozhraní
  - Malé úkoly
  - Nezávislost úkolů
  - Spolupráce s uživateli
    - Viz pravidla extrémního programování a obecně agilních metod Scrum, atp

Uživatelsky orientované rozhraní (UOR) a agilní formy vývoje

- UOR se musí vyvíjet ve spolupráci s uživateli
- Dobré UOR silně omezuje potřebu dokumentace, kromě prostředků potřebných pro používání systému. Práce služby se totiž dá často odvodit z jeho rozhraní.
- Použití prototypů umožňuje rychlou odezvu uživatelů.
- Dekompozice do služeb usnadňuje inkrementální vývoj
- ALE: UOR je možné jen za podmínky správné dekompozice (přibližně hranice organizačních jednotek)

Důsledky

- Obchodní procesy mohou „programovat“ uživatelé a mohou za ně ručit
- Kromě aplikačních služeb jsme diskutovali služby, které bychom mohli nazvat službami architekturními
  - Předřazené brány (adaptéry služeb implementované jako služby)
  - Portály
  - Manažery procesů
  - Datová úložiště
    - Databáze
    - Zprávy
- Architekturní služby se většinou pro danou aplikaci vyvíjejí od počátku. Mohou existovat další typy infra služeb.
- Existuje pokus o obdobu manažerů na www

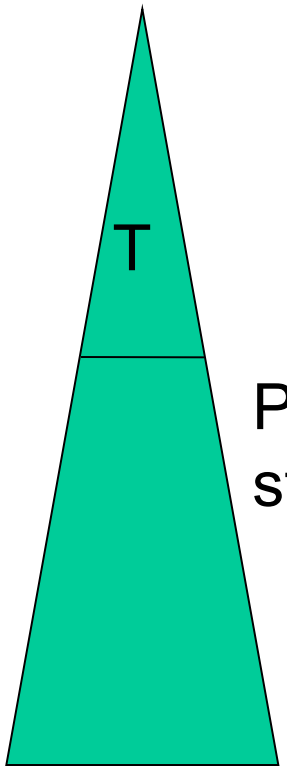
Pozorování

- Chování systému (business procesy) ovlivňuje do jisté míry sám koncový uživatel zadáváním parametrů instanciace procesu a úpravami jeho chování (to lze do jisté míry považovat za vývoj, který provádí koncový uživatel - enduser development)
  - Je snadné v konfederacích
- To je projev nových trendů ve vývoji a vlastnostech softwarových systémů, především pro malé a střední podniky a e-government

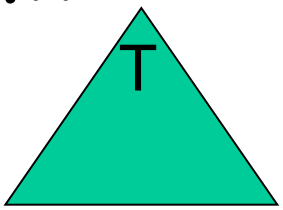
Kde hlavní přínosy

- Horizontální spolupráce přes hranice oddělení (příklad Otavan, prodejce si reservuje výrobní kapacity), ale to ohrožují některé manažery
- Dostupnost informací (PC+internet)
- Sociální kontakty, zlepšení ovzduší ve firmě
- Zefektivnění procesů
- Zpřesnění a vyšší dostupnost dat, zkvalitnění rozhodování
- Restrukturalizace obchodních procesů, větší agilita
- Zlepšení spolupráce s externími partnery

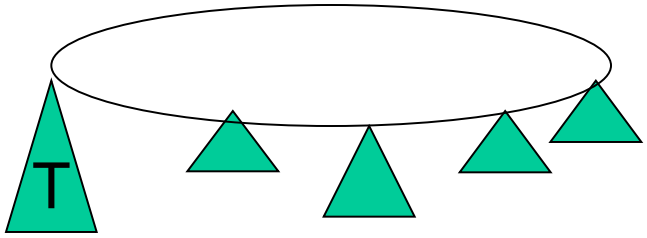
Dopad SOA na organizační hierarchii



Původní struktura



Čekalo se
(šéf uřídí více lidí)



Obvykle se stalo (decentralizace)

Vždy méně středního managementu i u něho změna profesní struktury (větší samostatnost šéfů divizí) při decentralizaci, příčina odporu a neúspěchů!!! Méně středního managementu ubylo při decentralizaci (další důvod výhiodnosti decentralizace)

Co se kupodivu v některých případech zjistilo

- Při pokusu o nízkou hierarchii se neušetřili úředníci
- Jsou náznaky, že se někdy zhoršila kvalita celkového vedení podniku (Jo Owen: Flat Organizations, Flat Results)
  - Zapomělo na to, že mnohé z inteligence, dovedností, zkušeností a znalostí, které jsou konkurenční výhodou firmy, jsou pouze v hlavách středníhomanagementu
  - Srv. špatné zkušenosti s business process reingeneering (BPR)

Co se kupodivu zjistilo

- Nastala mohutná centralizace a nárůst byrokracie. Místo odstraněných článků formálního hierarchického řízení zaujaly rozšiřující se štábní útvary, jejichž úkolem bylo koordinovat trénink, personální politiku, standardy, kvalitu, komunikaci, finanční kontrolu a celofiremní iniciativy.
- Rozhodnutí, která dřív vyžadovala 2 - 3 lidi náhle vyžadovala 10 - 20 lidí.
  - Snaha o alibi??

Co se kupodivu zjistilo

- Narůstalo politikaření a zmenšovala se zodpovědnost.
- Manažeři přestávali mluvit jeden s druhým. Tam, kde dřív jeden manažer zavolal druhému, aby s ním něco probral, nebo si dojednal schůzku, nyní nastupovaly sekretářky, které musely koordinovat velké množství diářů.
- Rozmnožily se porady a interní konference.
  - Často se nevědělo jak na věc když spousta znalostí a dovedností zmizela

Asi se nezměnila pravidla řízení

- Malý důraz, aby služby odpovídaly byznys akcím
- Asi nepoužívali
  - Video konference
  - Maily
  - Řízení přes IS
  - atd.
- Problém je ale přesto asi hlavně v koncepci

Asi se nezměnila pravidla řízení

- Administrativní režie se zavedením automatizace kancelářských prací a dalších informačních technologií zprvu nejen nezmenšila, ale naopak se v širokém spektru odvětví vytrvale zvětšuje. (P. Attewell: Information Technology and the Productivity Paradox).

Obrana: Service governance

- Sledování služeb a jejich vývoje
- Řízení projektů
- Stanovování priorit
- Bezpečnost
- .....

Musí být za účasti CEO

Možný model SOA (opakování)

Architekturní služby je možno chápat jako místa v zobecněné Petriho síti s barvenými značkami. Zobecněná místa mohou mít vlastní inteligenci.

Není jasné, zda je toto pozorování dostatečně nosné

Technické důvody pro SOA

- SW inženýrské přednosti
  - inkrementálnost, stabilita, láce, …
- Lze se vyhnout tzv. reorg. Cycle (než systém stihnu přeprogramovat, je nová verze již zastaralá)

Potřeba metadefinic

Konfederovaný systém může obsahovat velmi mnoho komponent komplikované funkcionality.

Komponenty je výhodné propojovat deklarativně (co je třeba udělat, nikoliv jak to udělat).

Pro různé skupiny komponent jsou třeba různé formáty.

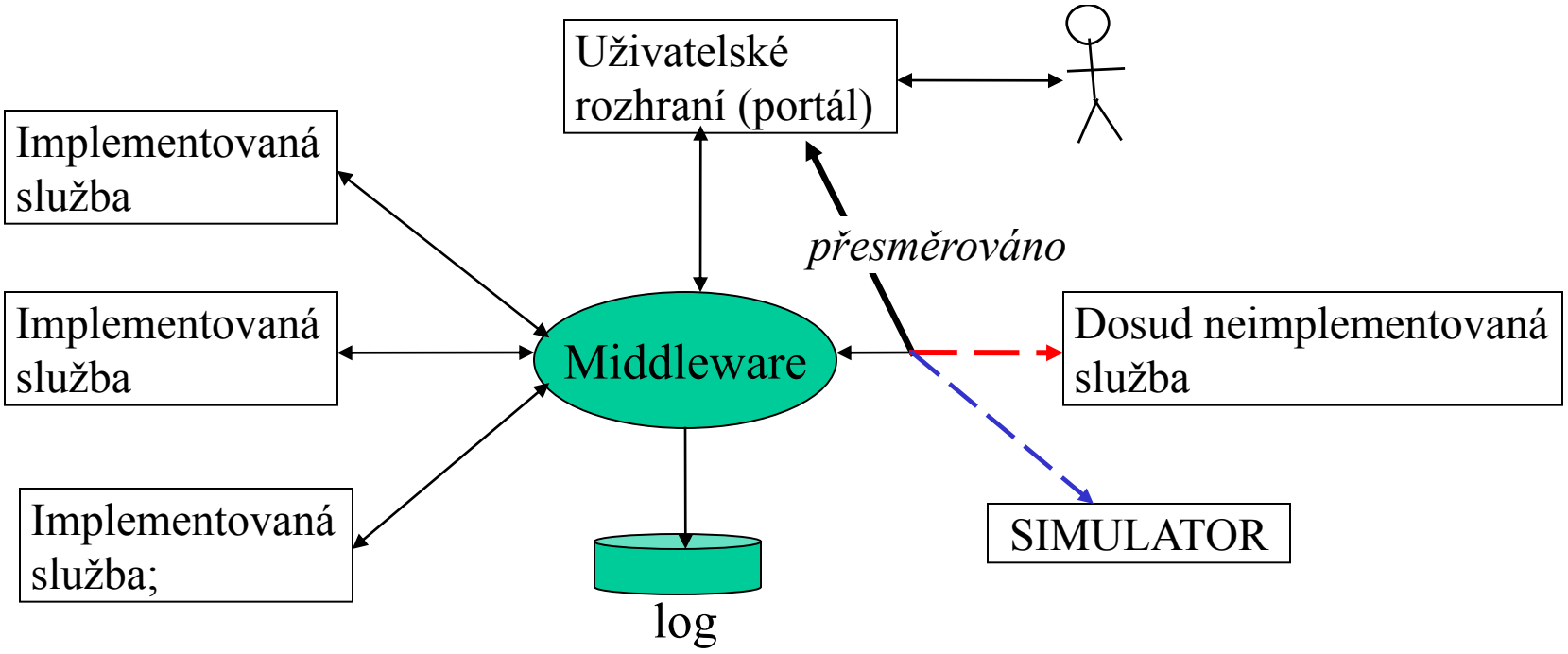
Je nereálné je definovat jednou provždy nebo centralizovaně => formáty dohadovat lokálně.

Jsou nutné dialekty!!!

Řešení: *Poskytnout nástroj pro rozšiřování syntaxe z jistého jádra*

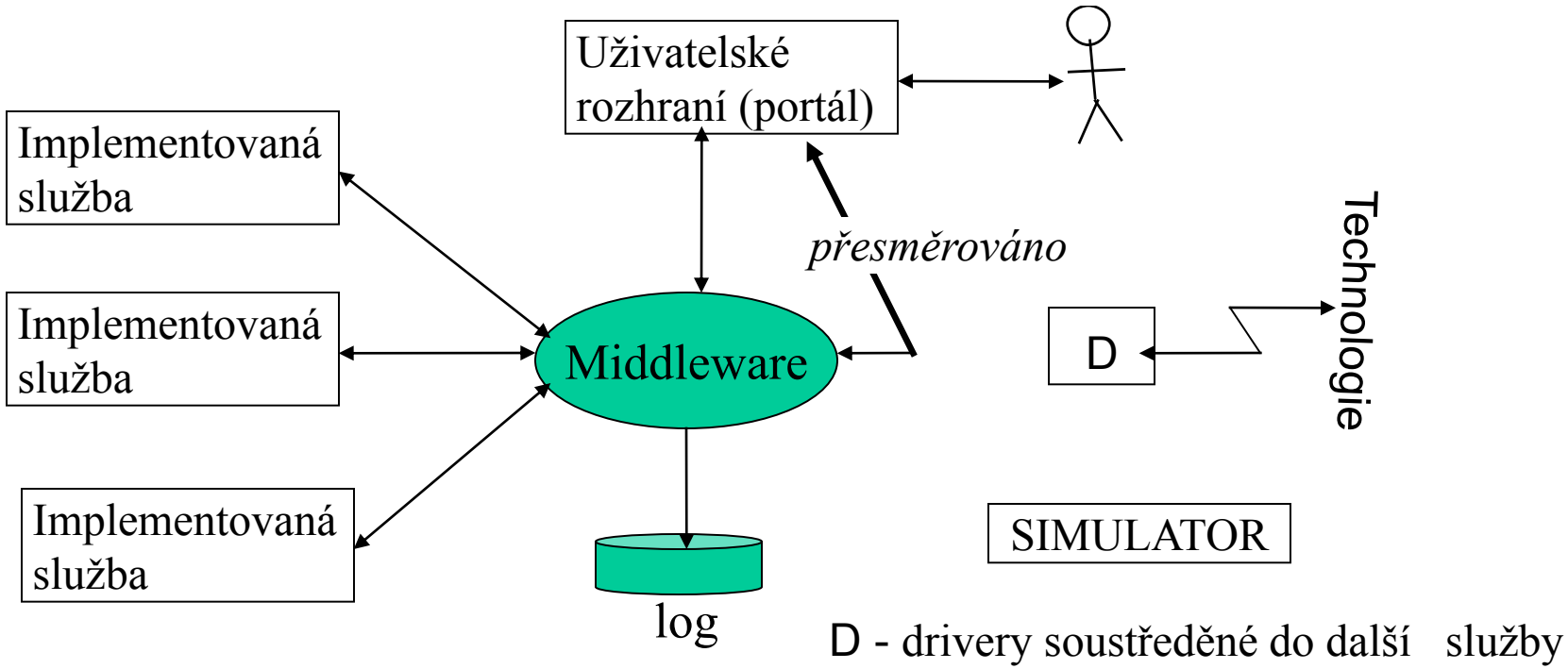
Řešení: XML,nebo koupě systému

Techniky 2
Prototypování, ladění RT systémů



Zprávy lze přesměrovat **beze změny implementovaných** služeb buď na uživatelské rozhraní (efektivní prototyp, použitelné i za běhu systému), nebo dokonce na simulátor (ladění RT systémů)

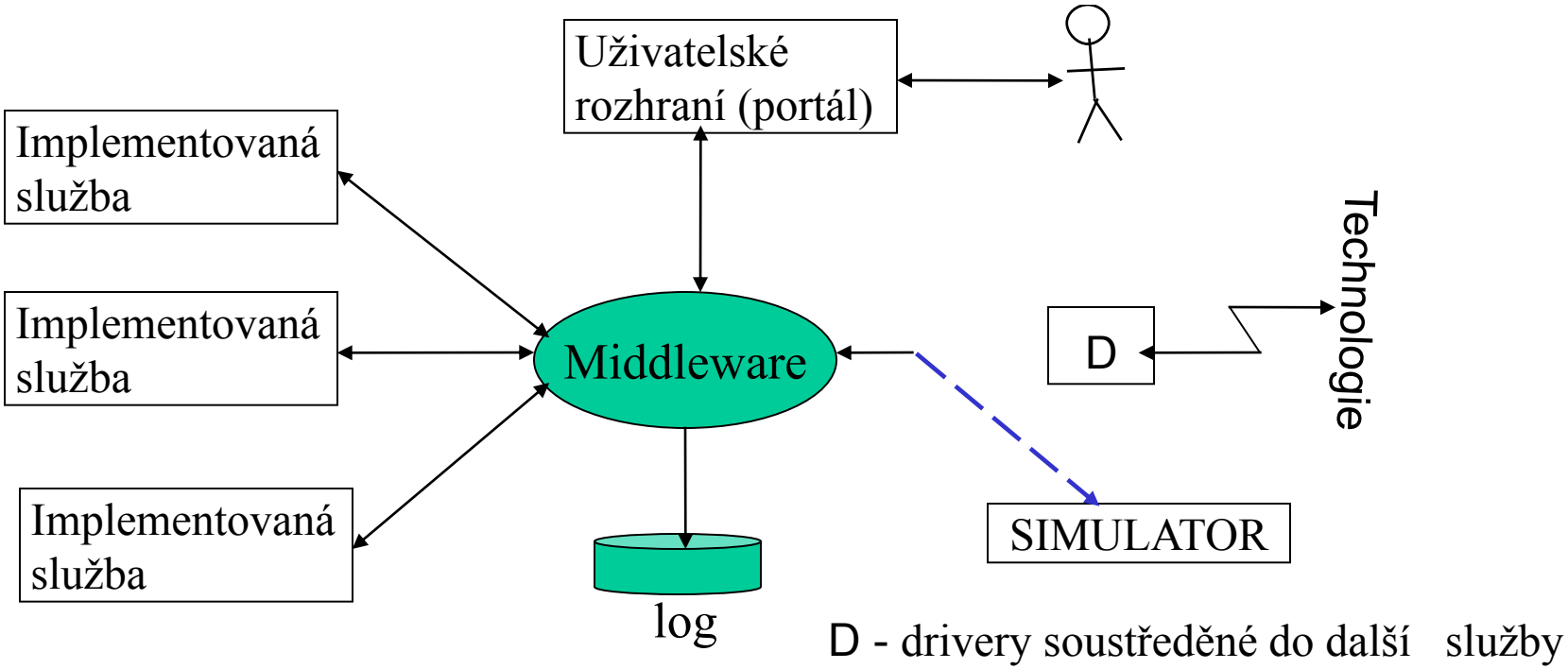
Techniky 2
Prototypování, ladění RT systémů



Zajímavý obrat – jak zjistit při dobu odezvy za nepřítomnosti řízené soustavy. Řešení

- Kalendář událostí v simulátoru,
- simulátor má nejvyšší prioritu

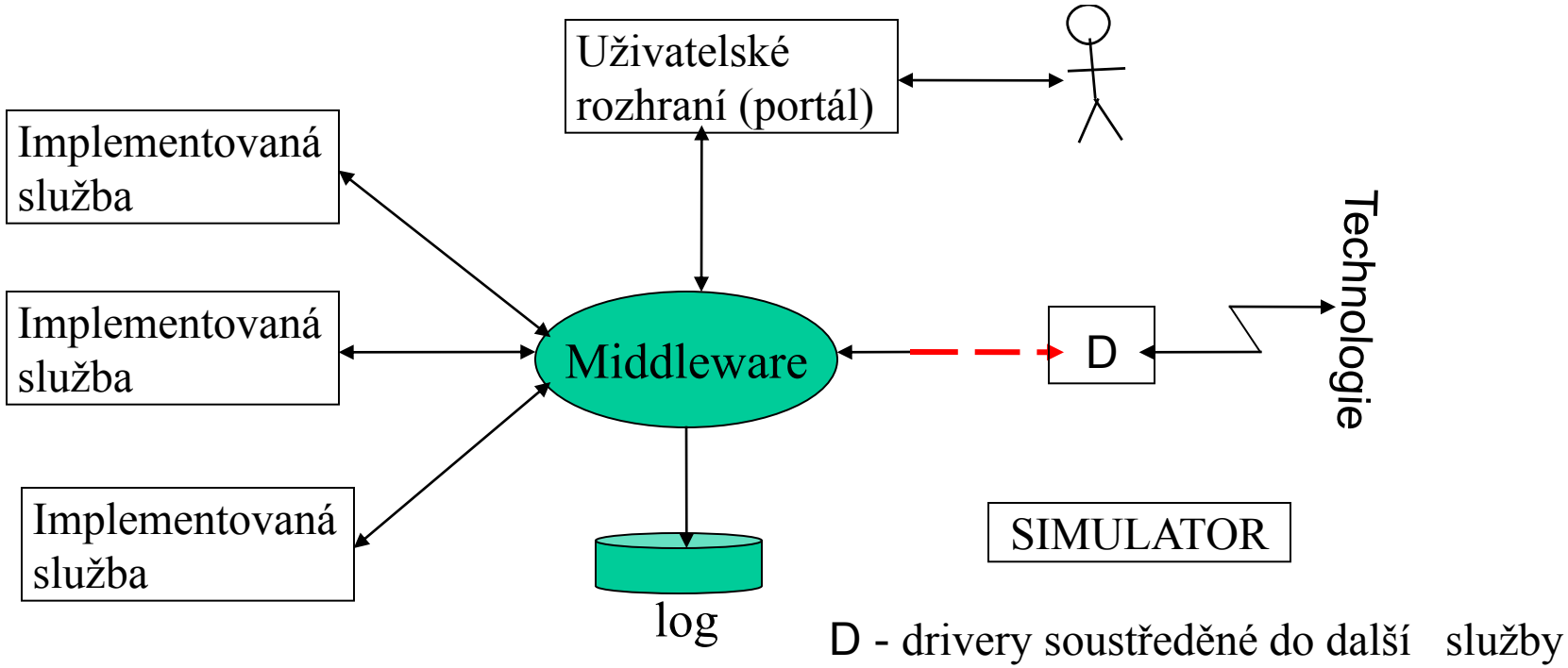
Techniky 2
Prototypování, ladění RT systémů



Zajímavý obrat – jak zjistit při dobu odezvy za nepřítomnosti řízené soustavy. Řešení

- Kalendář událostí v simulátoru,
- simulátor má nejvyšší prioritu

Techniky 2
Prototypování, ladění RT systémů



D - drivery soustředěné do další služby

Zajímavý obrat – jak zjistit při dobu odezvy za nepřítomnosti řízené soustavy. Řešení

- Kalendář událostí v simulátoru,
- simulátor má nejvyšší prioritu

Techniky 3

Nouzové základní brány

Aplikace nemusí být vybavena branou, tj. neposílá zprávy, jak doplnit?

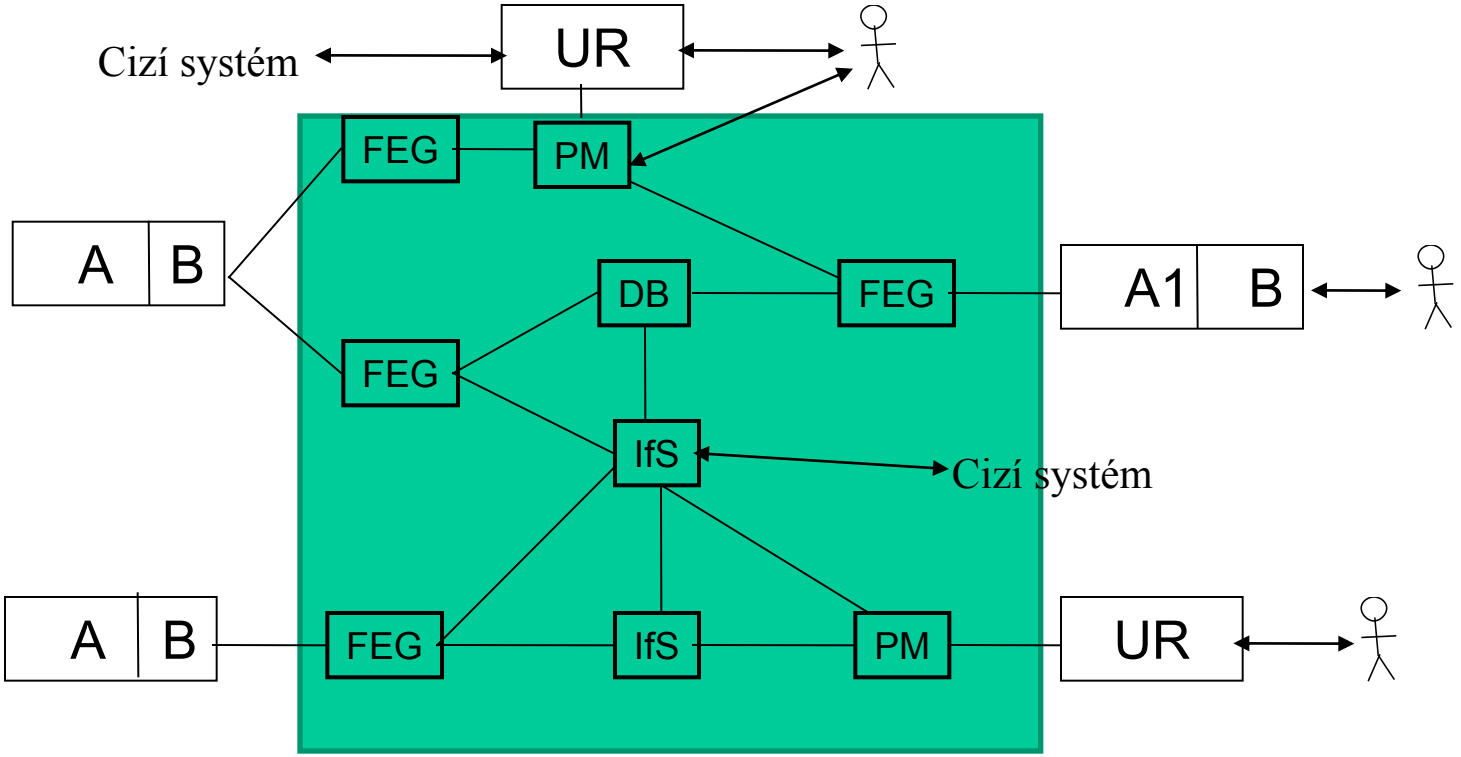
Jde to ztuha. Možnosti

- Mám kódy – upravím programy
- Odposlech na rozhraní UR * aplikační vrstva
- Odposlech terminálu

Architekturní služba

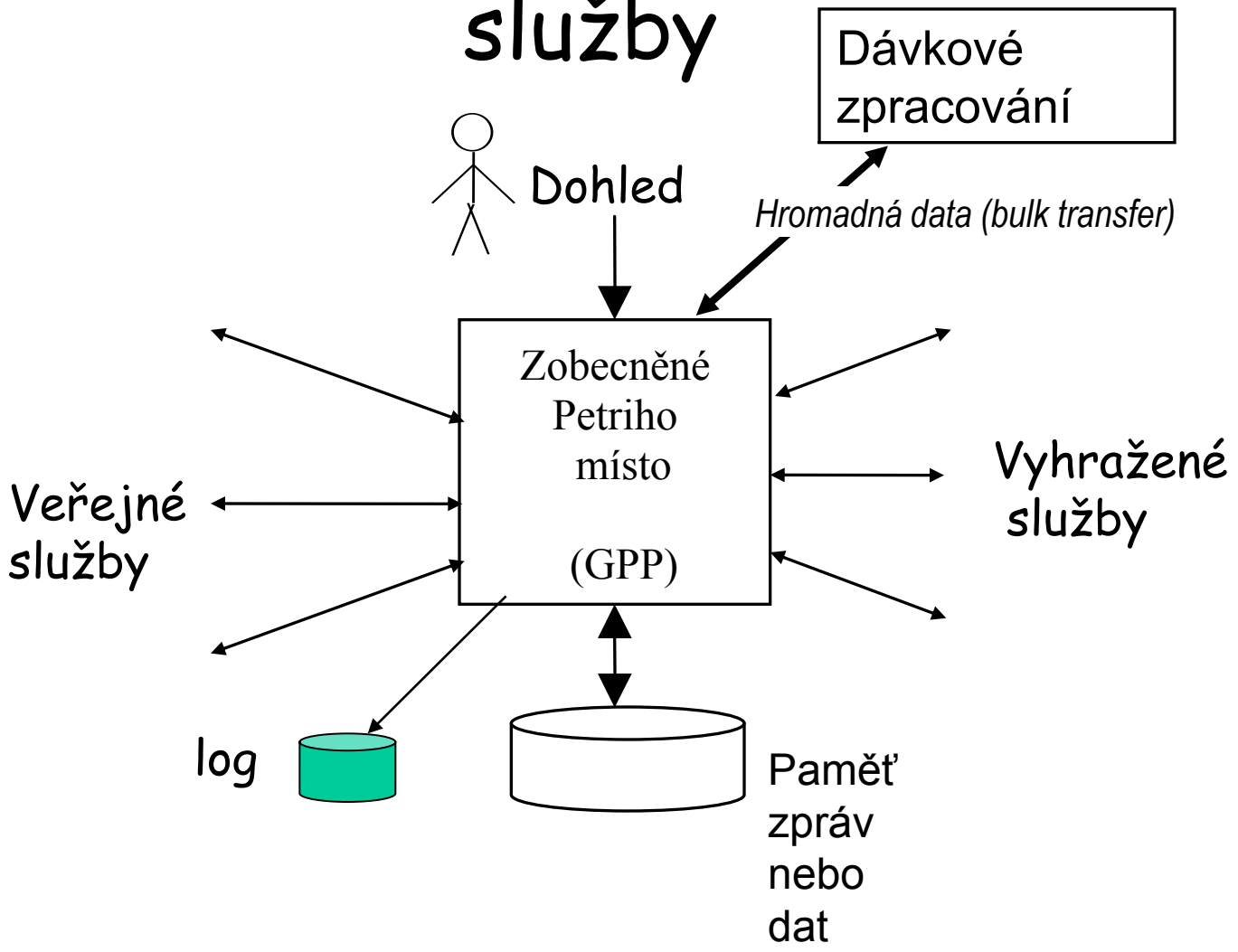
- Předřazenou bránu lze snadno modifikovat tak, aby
  - Přijímala zprávy od různých zdrojů
  - *m*-tice zpráv transformovala na *n*-tice zpráv
  - Výsledné zprávy směrovala na dynamicky volitelné adresáty
- Výsledek nazveme architekturní službou ArS
  - Jsou to konektory jako služby

Logický pohled na SOA s architekturními službami

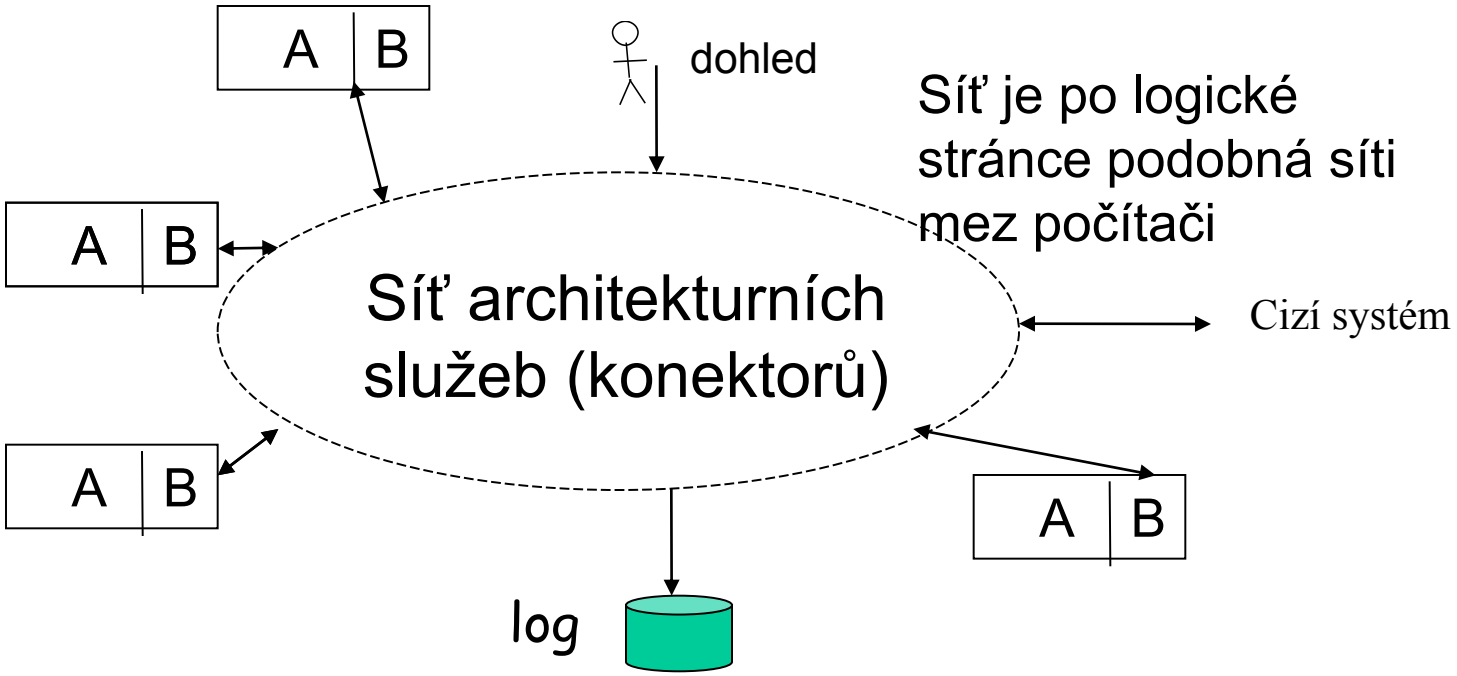


FEG – předřazená brána, UR – portál, PM - manažer procesu,
DB – datové úložiště IfS - infrastrukturní služba, obvykle záhlaví kompositní služby

Nejsilnější varianta architekturní služby



Jiný pohled



Dá se využít na gridech a v cloudech

Síť se chová jako služba poskytující architekturu, funkcemi silně připomíná HW poskitující konektivitu u počítačových sítí

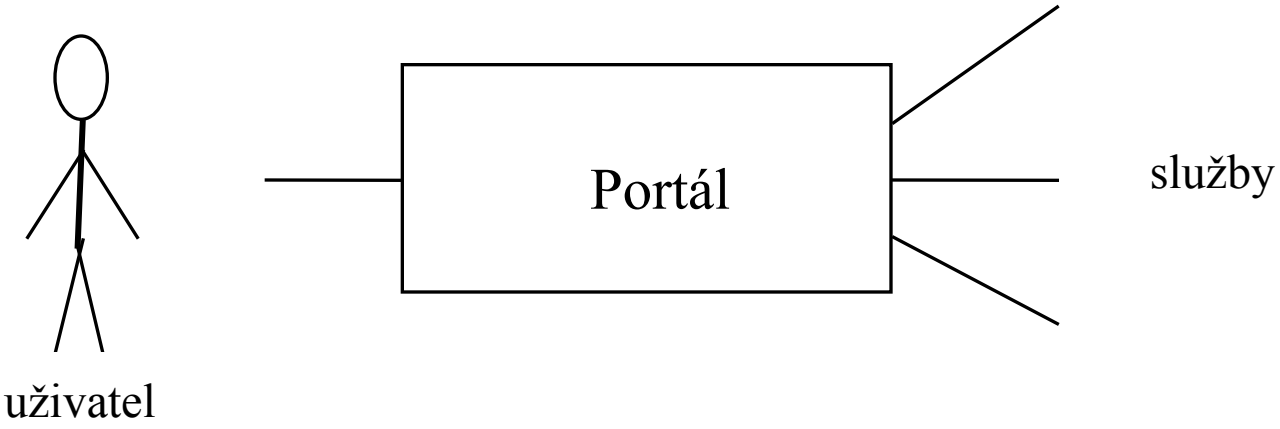
Architekturní služba funguje jako konektor implementovaný jako služba

Záhlaví kompositní služby

Varianta GPP (specializace), Předřazená brána je specializace záhlaví kompositní služby

Omezení (kontrakt resp. politika): libovolný požadavek na službu S_i od služeb z ostatních částí systému musí jít přes A.

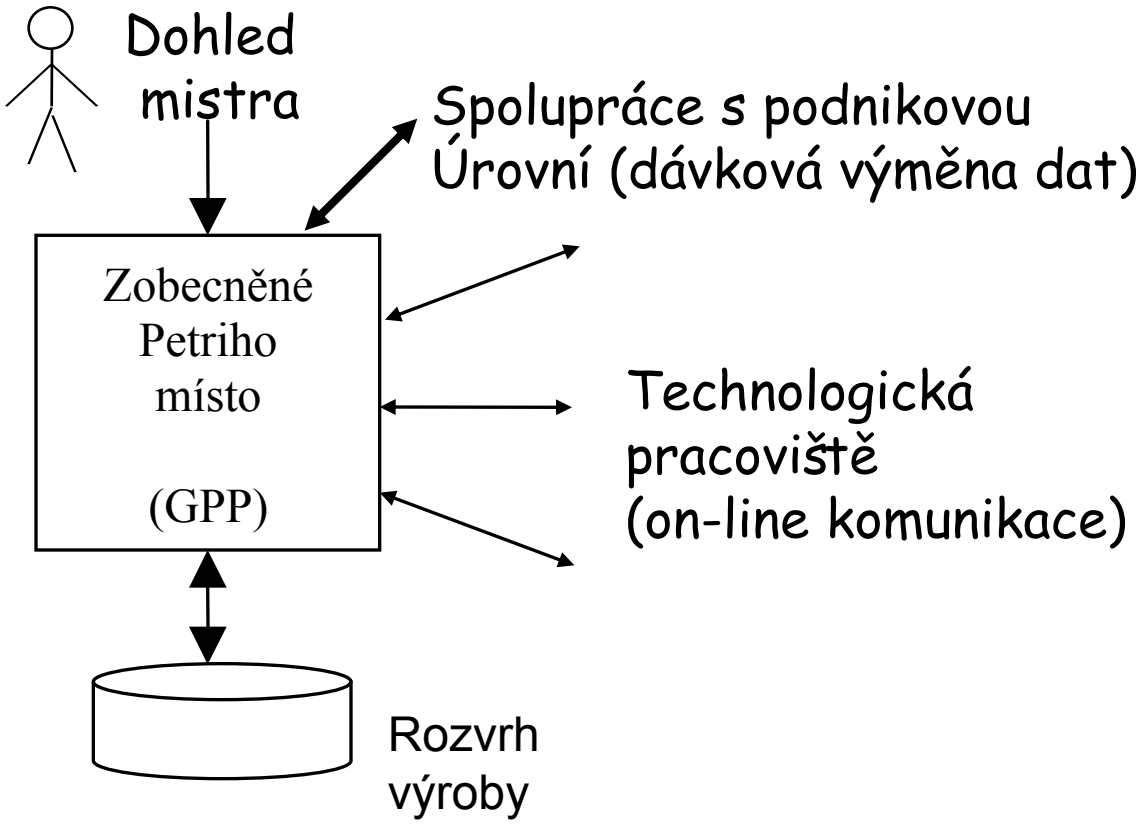
Varianta GPP (specializace), portál, vnější „služby“ jsou uživatelé



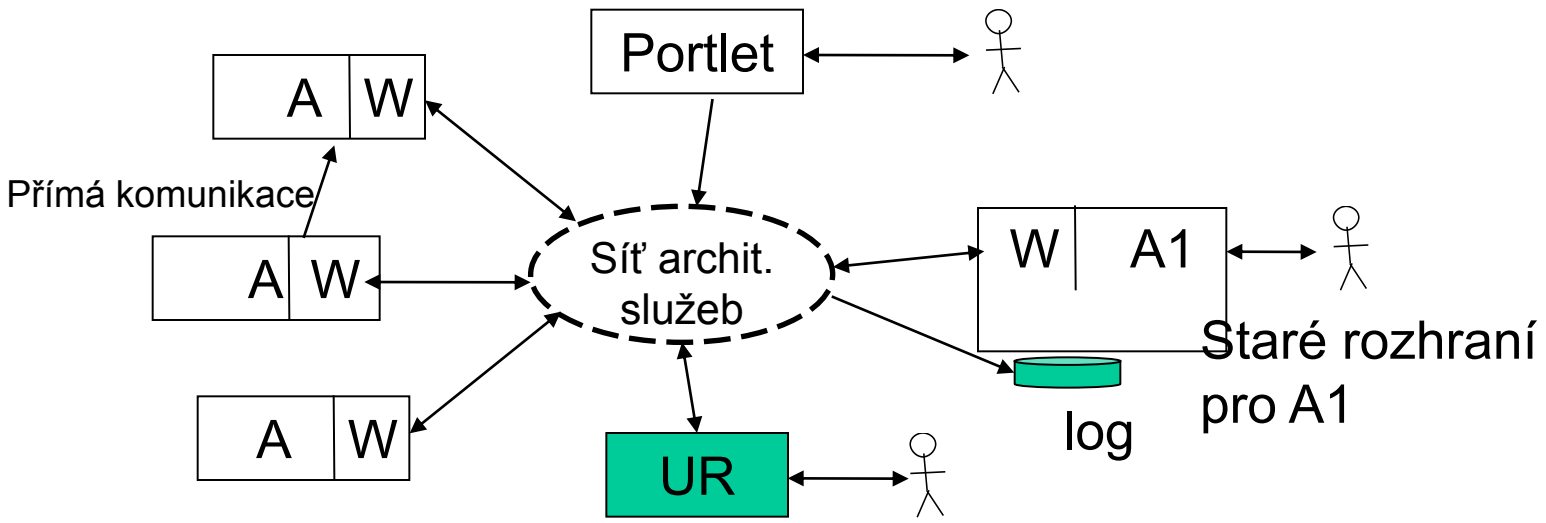
Omezení: zprávy mezi službami prochází prakticky vždy Portálem
Pro takové služby existuje skupina norem, výsledek se jmenuje

Portlet Service

Příklad prakticky použité varianty architekturní služby, řízení výroby



SOA s architekturními službami



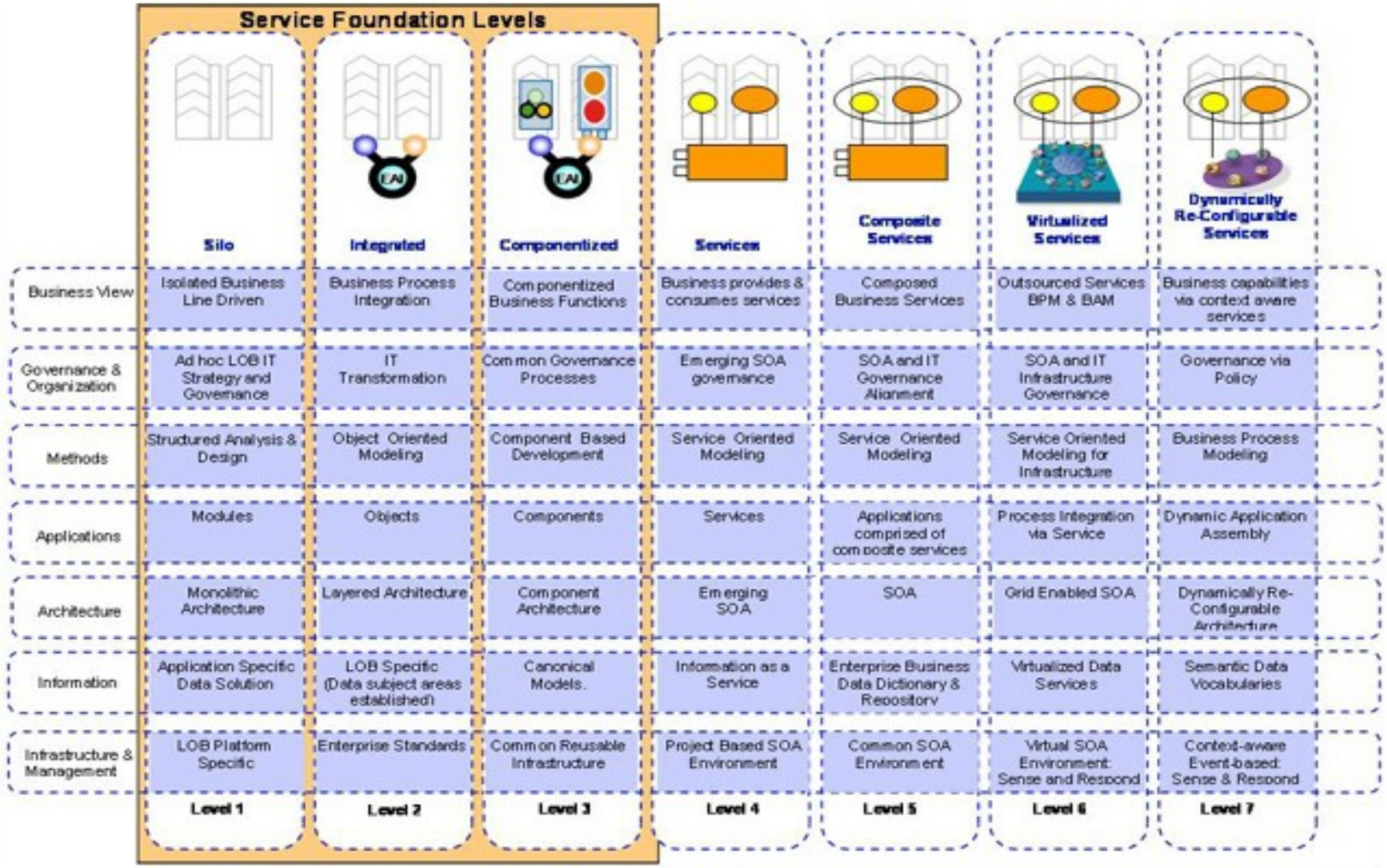
Zprávy mezi službami jdou zpravidla přes několik architekturních služeb. V tomto smyslu je architektura dvojvrstvá (nearchitekturní a architekturní služby podobně jako v cloudech). Lze vytvářet logické vrstvy podle variant funkcí arch. služeb). Síť architekturních služeb je vlastně také služba = jde tedy o implementaci architektury jako služby

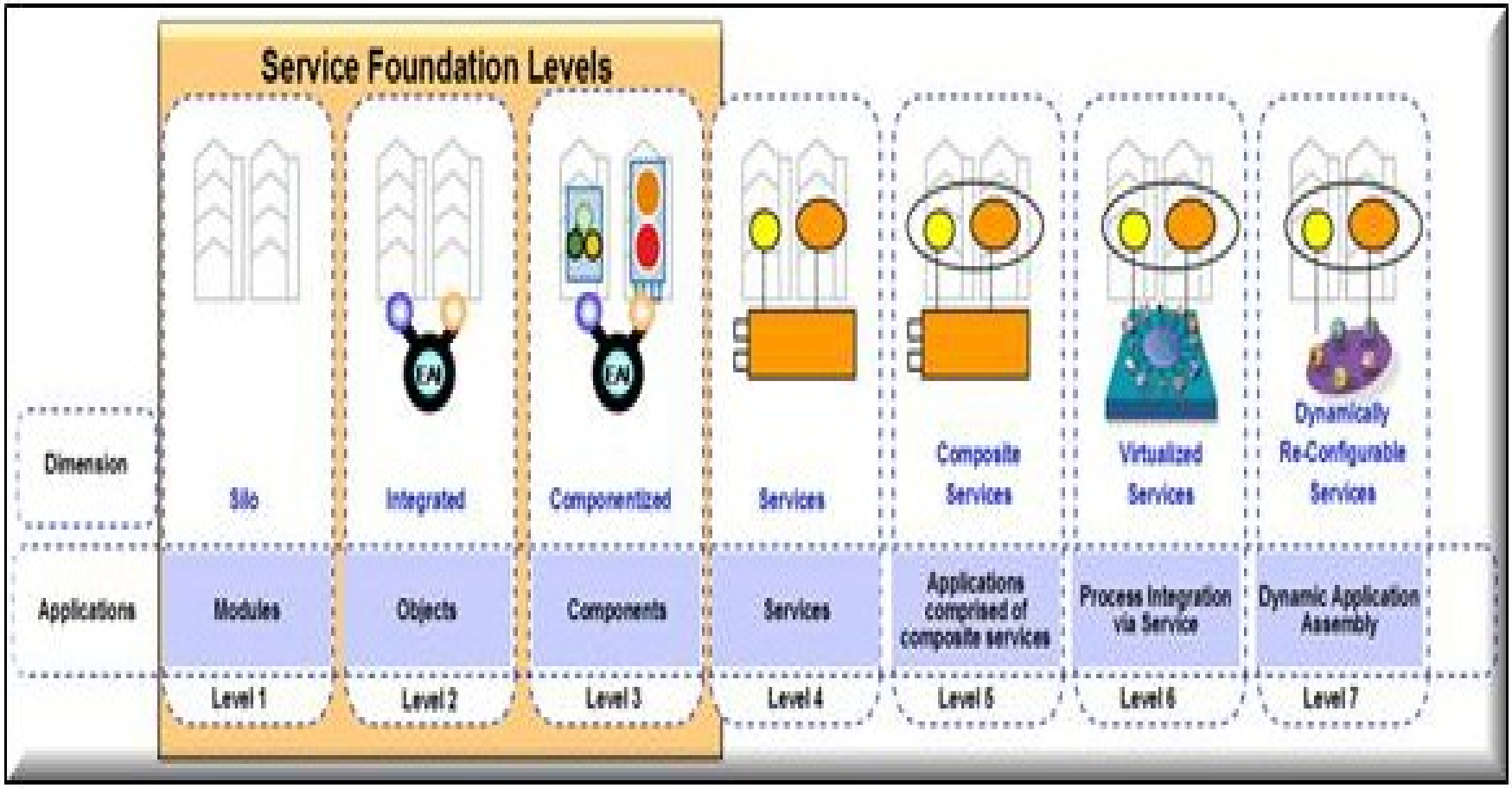
SOA s architekturními službami a agilní vývoj velkých systémů (ASOA)

- ASOA umožňují agilní vývoj o pro velké systémy
  - Agilita díky vývoji ArS a ty jsou jednoduché
  - Agilita díky kombinaci funkčností middlewarů
  - Agilita díky využívání úrovní granularity (wrapper, FEG, mezi ArS)
  - Agilita využíváním kapabilit ve wrapperech, ArS, middlewarech
  - Agilita díky tomu, že ASOA vytváří hierarchii služeb a služby mohou být různého typu

Použití v OSIMM,
Open Group Service Integration Maturity Model

- Silo (data integration)
- Integrated (application integration)
- Componentized (functional integration)
- Simple services (process integration)
- Composite services (supply-chain integration)
- Virtualized services (virtual infrastructure)
- Dynamically reconfigurable services (eco-system integration)


Service Foundation Levels
Silo
Integrated
Componentized
Services
Composite Services
Virtualized Services
Dynamically Re-Configurable Services
Business View
Isolated Business Line Driven
Business Process Integration
Componentized Business Functions
Business provides & consumes services
Composed Business Services
Outsourced Services BPM & BAM
Business capabilities via context aware services
Governance & Organization
Ad hoc LOB IT Strategy and Governance
IT Transformation
Common Governance Processes
Emerging SOA governance
SOA and IT Governance Alignment
SOA and IT Infrastructure Governance
Governance via Policy
Methods
Structured Analysis & Design
Object Oriented Modeling
Component Based Development
Service Oriented Modeling
Service Oriented Modeling
Service Oriented Modeling for Infrastructure
Business Process Modeling
Applications
Modules
Objects
Components
Services
Applications comprised of composite services
Process Integration via Service
Dynamic Application Assembly
Architecture
Monolithic Architecture
Layered Architecture
Component Architecture
Emerging SOA
SOA
Grid Enabled SOA
Dynamically Re-Configurable
Information
Application Specific Data Solution
LOB Specific (Data subject areas established)
Canonical Models.
Information as a Service
Enterprise Business Data Dictionary & Repository
Virtualized Data Services
Semantic Data Vocabularies
Infrastructure & Management
LOB Platform Specific
Enterprise Standards
Common Reusable Infrastructure
Project Based SOA Environment
Common SOA Environment
Virtual SOA Environment: Sense and Respond
Context-aware Event-based: Sense & Respond
Level 1
Level 2
Level 3
Level 4
Level 5
Level 6
Level 7

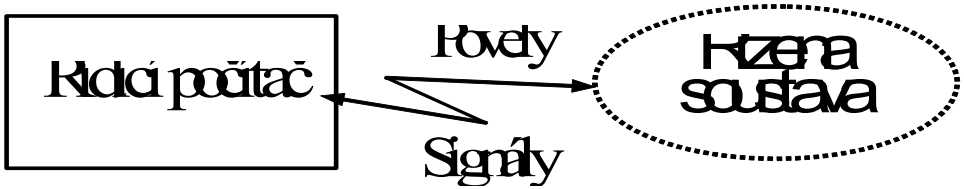

Service Foundation Levels
Dimension
Applications
Silo
Modules
Level 1
Integrated
Objects
Level 2
Componentized
Components
Level 3
Services
Services
Level 4
Composite Services
Applications comprised of composite services
Level 5
Virtualized Services
Process Integration via Service
Level 6
Dynamically Re-Configurable Services
Dynamic Application Assembly
Level 7

Maturity Level Cell name	***Maturity indicator***	***Maturity Attributes***
Silo *(Level 1)* Modules	*Application architectures are designed and implemented using SOA principles and development practices that utilize constructs such as loose-coupling, separation of concerns, and employ the use of service-enabled technologies such as XML, web services, service bus, service registries, and virtualization.*	*Low or nonexistent* *Application architectures and topologies are monolithic and lack integration between other systems across the enterprise.* *The use of web services or other SOA constructs are not present.*
Integrated *(Level 2)* Objects	*Application architectures are designed and implemented using SOA principles and development practices that utilize constructs such as loose-coupling, separation of concerns, and employ the use of service-enabled technologies such as XML, web services, service bus, service registries, and virtualization.*	*Limited* *Application architectures and topologies are monolithic with minimal separation of concerns between architectural layers or application tiers.* *Applications use minimal integration between other systems. Integration is usually implemented using point-to-point techniques*
Componentized *(Level 3)* Components	*Application architectures are designed and implemented using SOA principles and development practices that utilize constructs such as loose-coupling, separation of concerns, and employ the use of service-enabled technologies such as XML, web services, service bus, service registries, and virtualization.*	*Cross-organizational* *SOA development practices are applied inconsistently across the organization.* *Most application architecture topologies have a separation of concerns both physically and logically in presentation, business logic, and data tiers* *The use of SOA-enabling technologies – such as an ESB – is inconsistent across the enterprise.*

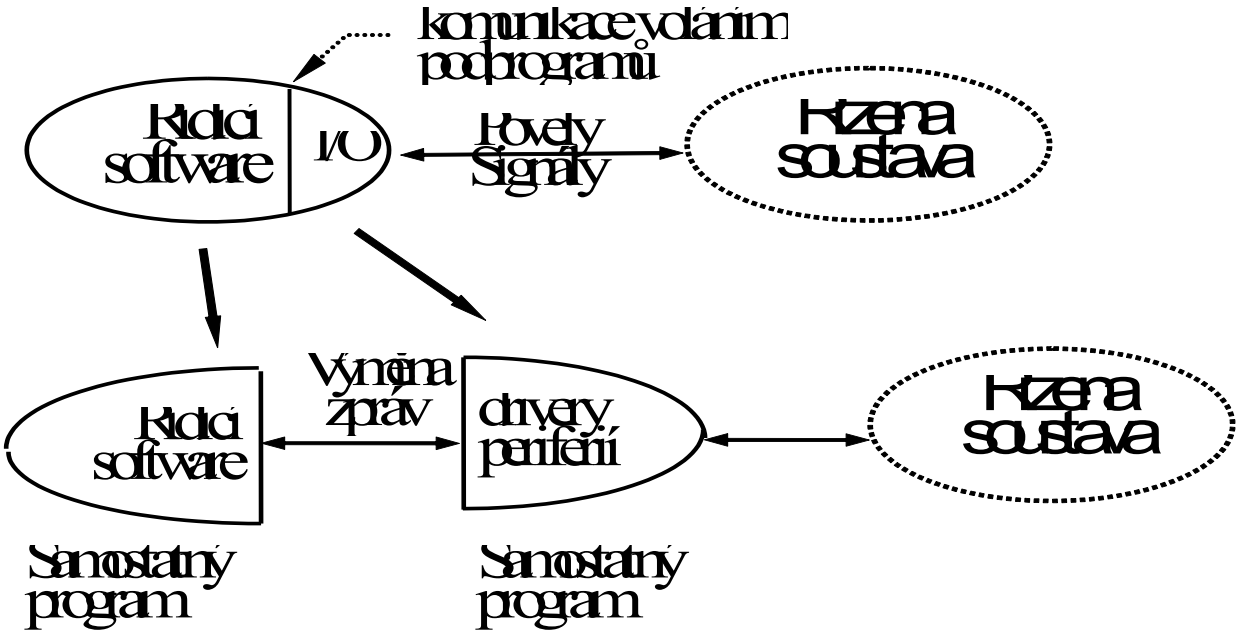
Maturity Level Cell name	***Maturity indicator***	***Maturity Attributes***
Services (level 4) services	*Application architectures are designed and implemented using SOA principles and development practices that utilize constructs such as loose-coupling, separation of concerns, and employ the use of service-enabled technologies such as XML, web services, service bus, service registries, and virtualization.*	*Cross-organizational* *SOA development practices are applied inconsistently across the organization.* *Most application architecture topologies have a separation of concerns both physically and logically in presentation, business logic, and data tiers.* *The use of SOA-enabling technologies – such as an ESB – is inconsistent across the enterprise*
Integrated (Level 5) Application are composed using composite services	*Application architectures are designed and implemented using SOA principles and development practices that utilize constructs such as loose-coupling, separation of concerns, and employ the use of service-enabled technologies such as XML, web services, service bus, service registries, and virtualization.*	*Integrated Enterprise-wide* *Application architectures are designed with a separation of concerns at the logical and physical layers.* *ESB integration patterns are used to support application and process integration to achieve sharing of services.*
Componentized (Level 6) Virtualized Services	*Application architectures are designed and implemented using SOA principles and development practices that utilize constructs such as loose-coupling, separation of concerns, and employ the use of service-enabled technologies such as XML, web services, service bus, service registries, and virtualization.*	*Integrated across the enterprise and externally between business partners.* *Application architecture is decoupled from infrastructure components.* *Extensive use of ESB architecture patterns to support Business Process Management (BPM).*

Maturity Level Cell name	***Maturity indicator***	***Maturity Attributes***
Dynamicaly reconfigrable services *(level 7)* *Services*	*Application architectures are designed and implemented using SOA principles and development practices that utilize constructs such as loose-coupling, separation of concerns, and employ the use of service-enabled technologies such as XML, web services, service bus, service registries, and virtualization.*	Adaptive Enterprise Application architecture supports dynamically reconfigurable business and infrastructure services and SOA solution for internal or external partner consumption.

K SOA

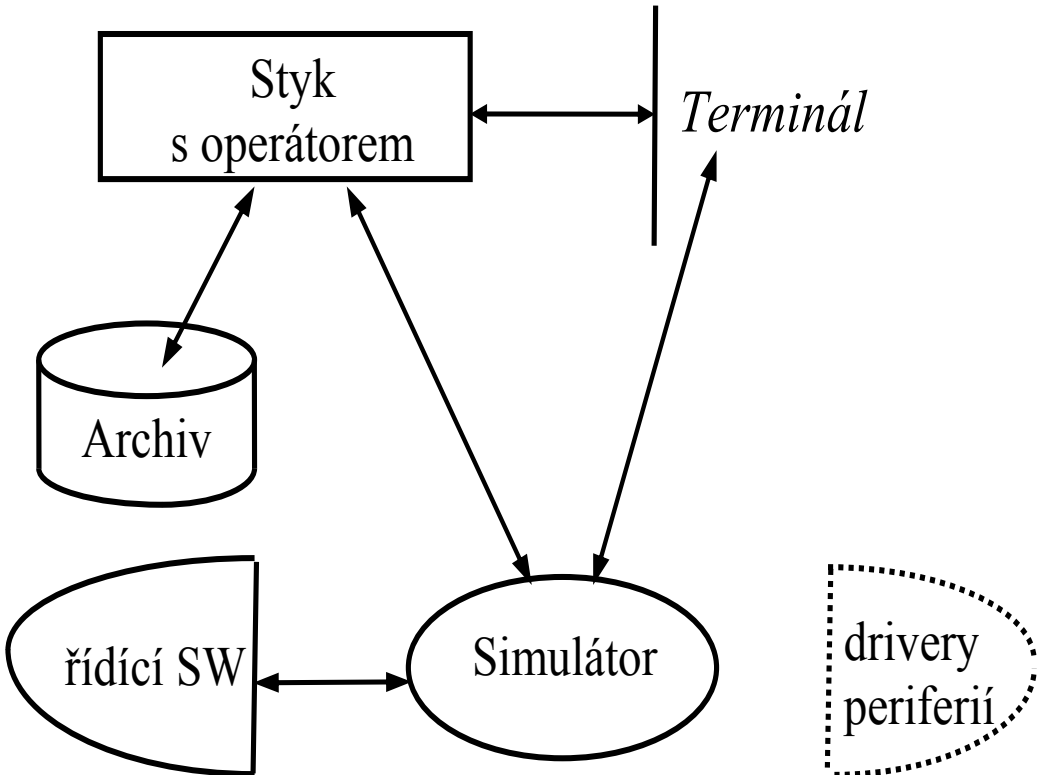


a) Soustava řízená počítačem



b) Oddělení ovladačů periferií realizujících styk s řízenou soustavou

K SOA, základní sestava



Problémy

- Je nutné uplatňovat nové marketingové, obchodní a manažerské přístupy
- Optimální využití architekturních služeb
- Využití výzkumu gridů a cloud computing
- Rovnováha kupovaného, starého a nově vyvíjeného softwaru
- Zvládnutí nových vzorů a optimální kombinace OO a SO

Pattern a antipattern

- Pattern – vzor - osvědčené řešení nějakého častého problému
- Antipattern – často používané nedostatečné až vysloveně špatné řešení nějakého úkolu
  - Jméno
  - Podstata řešení
  - Projevy, kde se vyskytne
  - Hlavní příčiny
  - Náprava

Techniky 4
Patterns and Antipatterns

Brown, W.J., et al: *Antipatterns. Refactoring Software, Architectures, and Projects in Crisis*, Wiley&Sons, 1998

Antipatterns v OO prostředí jsou takové prohřešky, které při vývoji OO systémů vedou s velkou pravděpodobností k selhání projektu

- Některé jsou fatální vždy (chybné specifikace),
- Jiné platí spíše pro monolitické systémy,
- Některé objektové antipatterns jsou v prostředí SOA předností.

Antiapatterns v OO, které jsou patterns v SO

Brown, W.J., et al: *Antipatterns. Refactoring Software, Architectures, and Projects in Crisis*, Wiley&Sons, 1998

- Použití existujících aplikací (legacy systems),
- Ostrovy automatizace (viz automatizace dílen, budování zdola integrací celých podsystémů),
- Funkční dekompozice, používání datových úložišť (DFD)

Patterns a antipatterns

Brown, W.J., et al: *Antipatterns. Refactoring Software, Architectures, and Projects in Crisis*, Wiley&Sons, 1998

Antipatterns, které mají v SO jen omezený vliv, zvláště u mneších systémů, nejsou tedy v SO v pravém smyslu antipatterns:

- Změna metodik vývoje během řešení
  - pokud se nedotýkají-li se změny infrastruktury systému
  - týkají se jen určitých komponent a jsou skryty za rozhraním,
- Vystřihovánky ve velkém
  - volné až dynamické kombinování služeb.

Patterns a antipatterns

Snadná prevence následujících OO antipatterns v SOA

- Plánování bez konce (lze prosadit rychlou realizaci jádra systému výběrem nejužitečnějších služeb, agilní používání a modifikace konektorů),
- Obtížný člen týmu (dáme mu vyvinout autonomní službu)
- Dlouhé termíny než je systém funkční (inkrementální vývoj),
- Návrh výborem který fakticky za nic neručí (neboť se nelze vymlouvat na fakt, že ještě nic nefunguje),
- Znovu vynalézat kolo (snadno používáme existující aplikace),
- Vendor lock-in - závislost na jednom dodavateli (lze doplňovat produkty třetích stran a nově vyvinuté komponenty)
    - Výrobci se brání prosazováním antivzoru „fine grained services“ tím, že prosazují pravidla správná pro objektovou orientaci

Servisně orientované vzory

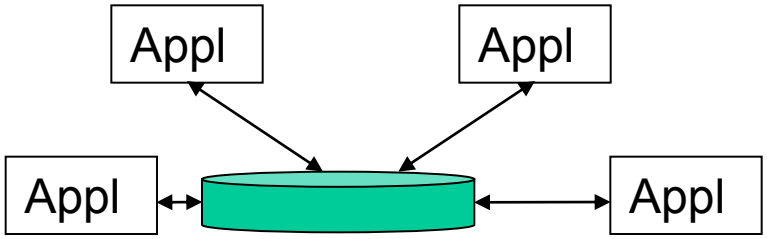
- Komponentová architektura
  - Konektory, zvláště architekturní služby
- |Prototypy
- Podpora agilních byznys procesů
- Uživatelsky orientované jazyky deklarativní na rozhraních komponent

Systémy podobné SOA

Autonomii a snadnou spolupráci částí lze u menších systémů dosáhnout i jinak
Podmínkou je

- •Výkonný HW a DB a
- •Nepříliš vysoké nároky na otevřenost a velikost systému
- • nebo web služby

• Častý případ: Samostatné aplikace spolupracující přes společnou databázi s využitím vestavěných procedur, Těch ale může být příliš mnoho



Specifikace požadavků

- Volba SOA může ovlivnit i vize (co je reálně možné požadovat), takže rozhodnutí použít SOA musí být učiněno většinou velmi časně
- Specifikace požadavků je pro SOA odlišná od klasických metod, neexistuje vhodné CASE, MDA nelze použít, snad jen zatím, ostatně se ani v OO pří a jeho použití může být kontraproduktivní
- Volba SOA musí být rozhodnutím i CEO i CIO uživatele, ti musí opustit své předsudky (vše naráz, vše od jednoho výrobce, najít cesty jako využít SOA v daném projektu)
- Mnohé se musí naučit i vývojáři

Objektovost a SOA

- Deklarativní rozhraní v XML je výhodně koncipovat neprocedurálně
  - Není tedy voláním metod, obvykle přenos dokumentů
  - Je spíše typu peer to peer
  - Stavební prvky jsou velké a uzavřené
- Používané technologie a praxe ukazuje, že i přístup k věci je v SO odlišný od objektové metodologie

Kdy je potřeba objektovost

- Od jisté úrovně hierarchie níže již nemohou mít komponenty konfederativní strukturu a nemohou být používány jako černé skříňky $\Rightarrow$ musí být spolehlivé $\Rightarrow$ měly by být OO
- ?? Od kdy nepoužívat SO
  - Př. Ve velmi dobře navržené a implementované konfederaci z USA nebyly kurzovní operace staženy do jednoho objektu. Výsledek – nedalo se to použít

Objekty předávány místo zpráv

- Př. Java Beans
- Předává se složitější struktura
- Je ale stále poměrně malá
- Je programátorsky orientovaná
- Závislost na platformě
- Problém logů
- Problém transformací objektů

Úvahy a otevřené problémy

- Není inženýrsky zvládnuto
  - Chybí zkušenost (success/failure stories)
    - Chybí metriky
    - Chybí osvědčené SW procesy pro SOA
      - ITIL je možná výjimka
    - Problémy s profesní skladbou týmů (méně klasických programátorů)
    - Nevíme, co je obtížné a co optimální
  - Co vše má dělat middleware (co kódovat na míru a co mít jako standard)
  - Jak na mobilní klienty a klonování služeb

Úvahy a otevřené problémy

- Nevíme jak optimálně využívat značkovací jazyky, především jejich metaúroveň
- Obchodní politika uživatelů a výrobců
- Souvislost s autonomními agenty, gridy, cloud computing a metapočítáním
- Optimalizace výkonu konfederace (mobilita, klonování)
- Kdy je konfederace výhodná a kdy ne

Úvahy a otevřené problémy

- Jak upravit studium počítačových expertů
  - Nutnost výuky ekonomie, induktivního uvažování a některých humanitních oborů. To je obecný problém inženýrského studia, pro SW zvláště urgentní
  - Nutnost otevřenosti a adaptibility
  - Syndrom o všem něco, nic pořádně
  - Syndrom hackera. S lidmi nerad jednám

Příčiny problémů v SOA

1) missing executive sponsorship,
2) problems to explain SOA business value,
3) lack of funding,
4) missing skills,
5) poor project management,
6) SOA as a project,
7) missing SOA governance,
8) underestimation of the complexity of SOA,
9) letting the vendors drive the architecture,
10) underestimation of the impact of organizational change

Diskuse

The issues 1) and 3) are common for the large software systems development.
The others are SOA specific.
6) is due to the fact that SOA project management must use specific managerial skills. Like the developers must develop and use specific development skills

Co lze očekávat v budoucnosti

- Zlepšení HW podpory
  - Dostupnost a kapacita linek
  - Poslední míle (WiMax, WiFi, Mobily, rozvod po elektrické síti)
  - Cloud computing, gridy, SaaS, ----
- Komerční nástroje
  - .NET?, Service Bus, Net Weaver, nové CASE
- Vyřešení věcných problémů
  - Hranice služeb, uživatelsky orientované standardy, komerční řešení, zvládnutí paradigmatu, best practices, Petriho sítě

Co lze čekat v budoucnosti

- Obchodně manažerské problémy
  - Změny obchodních praktik výrobců i odběratelů, změny SW procesů, změny organizace uživatelů, SOA se ČR zatím málo používá
- Subjektivní omezení
  - Změna postojů, přijetí paradigmatu, specifikace požadavků ve spolupráci „všech“, změny ve výchově informatiků
- Vedle podpory obchodních procesů průnik SOA do světa zábavy a snad i sociálního SW, což vynese gigadolary

Příbuzné problémy

- Systémy hromadné obsluhy, kdy klonovat služby?
- Systémy mající prvky p2p
  - Gridovské systémy (vztah k SOA, asi větší důraz na efektivitu), cloud computing
  - Softwaroví agenti (chytré služby)
- Není jasné, zda shody nejsou spíše povrchní a zda se mohou SOA systémy poučit od agentů a gridů

Důsledky pro SW profese

- Nejasný vliv SO na potřebu psaní nových komponent od začátku
  - Lze snáze uplatnit v existujícím balíku a snáze doplnit funkce do svého produktu
  - Komponentu lze získat kdekoliv na světě
- Větší důraz na nepočítačové znalosti a otevřenost myšlení
  - Nutnost rozumět funkcím komponent, tedy jiným oborům (např. statistika) – to je dnes problém většiny inženýrských oborů
  - Potřeba umět doporučit správné komponenty
- Umět vidět a analyzovat souvislosti (výhodné i pro změnu profese)
- Výhodné pro agilní formy vývoje i velkých systémů

Co chybí

- Optimální dělba funkcí mezi middleware a služby (tj. patterns)
- Realizace VPN (vyhrazené a zabezpečené spoje)
- Zpřesnění definice služby (omezení na opravdu nosné části, resp. vyjasnění variant)
- Doladění obchodních taktik
- Při jaké velikosti je SO vhodné

Open Group a OSIMM

Standard pro vývoj SOA jako
standard postupné integrace

ITIL and SOA: Can they Co-Exist?

Vijay Raghavan
TowerStrides Inc.

scaling new Frontiers …

Why SOA?

- Complexity of the Software
- Unrelenting user Requirements
- Forced to be Agile
- Integrate new mergers and Acquisitions
- Reuse existing Assets
- Reaching new threshold of connectivity and computing power.
- Get ready for Grid Computing and On Demand Computing

What is ITIL?

- ITIL (the IT Infrastructure Library) is the most widely accepted approach to IT service management in the world.
- ITIL® provides a cohesive set of best practice, drawn from the public and private sectors internationally.
- The best practice processes promoted in ITIL® support and are supported by, the British Standards Institution's standard for IT service Management (BS15000).
- A service-focused approach to delivering and supporting IT services.
- The objective of IT Service Management processes is to contribute to the quality of the IT services.
- Dnes již ITIL 3.0, dost se mění,
- Systematika pro service governance

Why ITIL

- Organize large, heterogeneous Chaotic IT Systems
- Instal best practices to avoid costly errors
- Gain Credibility, Cut Costs
- Improve Service
- Better Asset Utilization
- Helps you to focus on critical business processes

- Lze využít v SOA

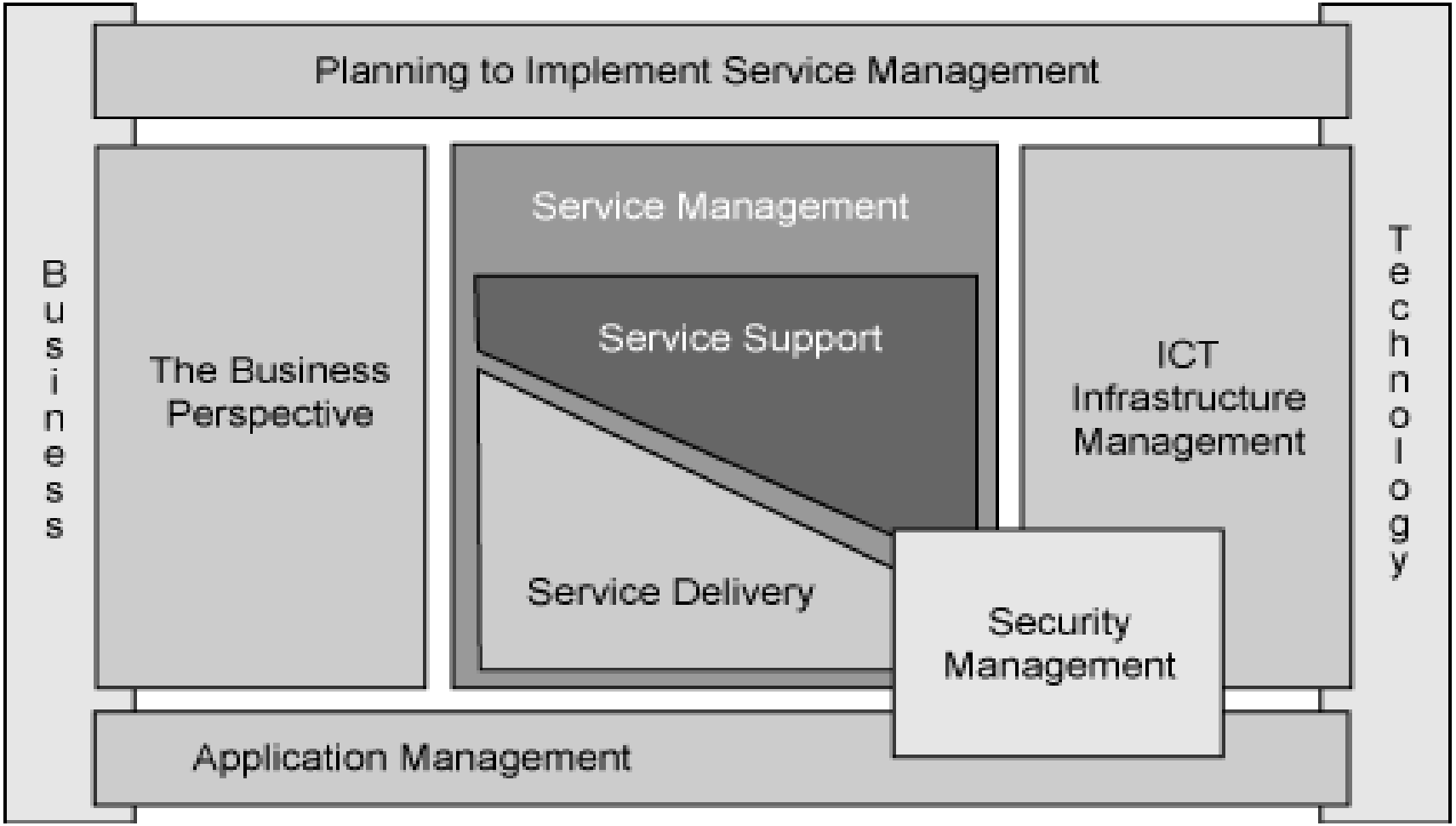
The common Ground

- SOA is not about Web services, it is about an approach that creates agility and responsiveness in both IT and business.
- That responsiveness to environmental conditions requires monitoring, reporting, and responding
- ITIL/ITSM focuses on just that.
- ITIL is all about governance, and SOA needs a good healthy dose of governance to move to the next level.

The ITIL v2 is defined in a collection of seven books,

- *The Business Perspective*
- *Planning to Implement Service Management*
- *Service Delivery*
- *Service Support*
- *Security Management*
- *Application Management*
- *ICT Infrastructure Management*

The seven ITIL books



http://www-128.ibm.com/developerworks/ibm/library/ar-soaitil/

The Business Perspective

This book is targeted to business managers, and it helps them to see the IT perspective on things and talk about concepts such as business continuity planning, outsourcing, business change management, and so on.

Planning to Implement Service Management

- This book is written by IT professionals who are highly experienced in implementing IT service management (ITSM) concepts. It provides practical advice, guidelines, checklists, and other project management tools to help you successfully implement ITIL and ITSM in your organization.

Service delivery

- This book is arguably the most important and explores how to deliver IT services. It covers several key concepts, such as:
  - Service-level management
  - Financial management for IT services
  - Capacity management
  - IT service continuity management
  - Security management
  - Availability management

Service support

- This book describes how customers can access the different IT services available. It covers concepts such as:
  - Service desk
  - Incident management
  - Problem management
  - Configuration management
  - Change management
  - Release management

Security management

- Data and information
- confidentiality,
- integrity,
- Availability

…..

Application management

- This book talks about how typical application support and management can be performed using a services perspective. It allows for applications to be managed within the context of the business objectives.

ICT Infrastructure Management

- This book is technology focused and talks about the people, process, and tools required to manage a stable IT infrastructure at an acceptable cost.

ITIL v3, core 5 volumes proti v2.0 silně upraveno

Přiblížení k SOA

Standardy ISO 20000, BS 15000

People and Process

- ITIL is all about people and process
- SOA cannot be successful if People and process are not together
- IT and Operations come together as part of SOA.

Book 1: Service Strategy

- Identification of market opportunities for which services could be developed in order to meet a requirement on the part of internal or external customers.
- The output is a strategy for the design, implementation, maintenance and continual improvement of the service as an organizational capability and a strategic asset.
- Key areas
  - Service Portfolio Management
  - Financial Management.

Book 2: Service Design

- Activities that take place in order to develop the strategy into a design document which addresses all aspects of the proposed service, as well as the processes intended to support it.
- Key areas
  - Availability Management,
  - Capacity Management,
  - Continuity Management and
  - Security Management.

.

Book 3: Service Transition

- Implementation of the output of the service design activities and the creation of a production service or modification of an existing service. There is an area of overlap between Service Transition and Service Operation.
- Key areas
  - Change Management,
  - Release Management,
  - Configuration Management and
  - Service Knowledge Management.

Book 4: Service Operation

- focuses on the activities required to operate the services and maintain their functionality as defined in the Service Level Agreements with the customers. Key areas of this volume are
  - Incident Management,
  - Problem Management and
  - Request Fulfillment.

Book 5:
Continual Service Improvement

- Ability to deliver continual improvement to the quality of the services that the IT organization delivers to the business. Key areas of this volume are
  - Service Reporting,
  - Service Measurement and
  - Service Level Management.

ITIL v 2.0

1. *The Business Perspective*
2. *Planning to Implement Service Management*
3. *Service Delivery*
4. *Service Support*
5. *Security Management*
6. *Application Management*
7. *ICT Infrastructure Management*

ITIL v 3.0

1. **Service Strategy**
2. **Service Design**
3. **Service Transition**
4. **Service Operation**
5. **Continual Service Improvement**

Nezohledňuje vliv architektury!!!!, nezná pojem konektoru

Příklad SOA technik

- Jak testovat RT SW bez přítomnosti řízeného systému

Odkazy

- ISO 20000 - Procesy
- ISO 250xx – Kvalita a metriky
- ISO 270xx - Zabezpečení

Principy agilního vývoje 1

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Business people and developers must work together daily throughout the project.

Principy agilního vývoje 2

1. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
2. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
3. Working software is the primary measure of progress.
4. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

Principy agilního vývoje 3

1. Continuous attention to technical excellence and good design enhances agility.
2. Simplicity--the art of maximizing the amount of work not done--is essential.
3. The best architectures, requirements, and designs emerge from self-organizing teams.
4. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Agile development prefers:

A. Individuals and interactions over processes and tools

B. Working software over comprehensive documentation

C. Customer collaboration over contract negotiation

D. Responding to change over following a plan

Terminologie a pojmy

- Nejsou ustálené
- Mění se
- Marketingové zájmy.
- Varianty architektur
  - Základní služby systému se nemusí vyhledávat
  - Základní služby musí navíc používat deklarativní i uživatelské rozhraní

Servisně orientovaná architektura v naší interpretaci

Soubor softwarových komponent spolupracujících obdobně jako služby reálného světa

- Všechny služby jsou stále aktivní
- Vyřizují požadavky asynchronně
- Konfederace – služby se znají - i aliance, kdy je službu nutno vyhledat

Služby a infrastruktura, zajišťující jejich spolupráci (middleware), včetně možnosti čekání na odpověď

V prakticky použitelných systémech se pravidla pro SOA vztahují pouze na **část** (**jádro)** systému. Jeho volnou součástí jsou dávkové části (export dat, hromadné výpočty, atd.)

Technické důvody pro SOA

- SW inženýrské přednosti
  - inkrementálnost, stabilita, láce, …
  - Agilními zásahy do konektorů lze dosáhnout možnost agilně vyvíjet velké systémy

- Lze se vyhnout tzv. Reorg. Cycle (než systém stihnu naprogramovat, je již zastaralý)

Potřeba metadefinic

Konfederovaný systém může obsahovat velmi mnoho komponent komplikované funkcionality.

Komponenty je výhodné propojovat deklarativně (co je třeba udělat, nikoliv jak to udělat).

Pro různé skupiny komponent jsou třeba různé formáty.

Je nereálné je definovat jednou provždy nebo centralizovaně => formáty dohadovat lokálně.

Jsou nutné dialekty!!!

Řešení: *Poskytnout nástroj pro rozšiřování syntaxe z jistého jádra*

Řešení: XML,nebo koupě systému

Net Weaver od SAP

- Technologie konkurenční k ESB
- Je méně otevřená
- Není jasné, jak dopadne
- Nevíme zda nejsou komerční systémy pro ESB i Netweaver příliš omezující.
- SAP ale otevírá prostor pro integraci třetích stran

Typy architekturních služeb

- Transformátory zpráv a směrovače (předřazené brány)
  - Dají se zobecnit na více vstupů a výstupů a chápat jako zobecněná místa ve smyslu Petriho sítí
- Konvertory dat a protokolů
- Portály
  - Chápat je jako služby (portlety)
- Řízení procesů
- Podpora nestandardních způsobů předávání zpráv
- Brány k virtuálním databázím

Použitelnost XSLT ?!

Komponentové architektury

- Komponenty a konektory (propojují komponenty)
- Obvyklé: Konektor se přilinkuje z knihoven jako dll modul, začíná se od „výkonných“ programů“ – komponent
- Varianta: Komponenty se hledají (včetně konektorů) ke zvolenému konektoru
- Varianta: Konektor je plnoprávná komponenta a je zároveň službou (na příklad FEG)
- To vede až ke struktuře, která získává architekturu jako službu, viz výše
  - Je to neobyčejně flexibilní systém se zajímavými možnostmi, viz RT systém

SOA pro velké podniky podle standardů OASIS

- **Service**
- The central concept of the Reference Model is that of *service*, which the Reference Model defines as follows: A mechanism to enable access to one or more capabilities, where the access is provided using a prescribed interface and is exercised consistent with constraints and policies as specified by the service description.
- Služba

Jiný pohled

- Srovnatelné co do složitosti s e-bankovnictvím
- Lze využít i mobilní spoje pokud nejsou ostré požadavky na rychlost odezvy (koupil si daný občan mnoho zneužitelných léků?) příliš přísné, zde evidentně nemusí být
- Pokud se nemusí dodržovat všechny možné standardy lze asi realizovat o mnoho levněji
- ***To je jedna z příčin rozčarování se SOA od velkých dodavatelů!!!!***
- ***SOA cestou integrace (Open Group)***
  - ***OSIMM*** **open group service integration maturity model**

Co chybí

- Obecná shoda o typech servisně orientovaných SW systémů (SOSS)
- Povědomí, jako optimálně kombinovat OO a SO,
- Příklady řešení (case studies, best practices)
- Dělba práce mezi middleware, centrální služby, infrastrukturní služby a aplikační služby
- Vhodné modelovací nástroje (model driven architecture?) a CASE systémy