KATEDRA INFORMATIKY
PŘÍRODOVĚDECKÁ FAKULTA
UNIVERZITA PALACKÉHO

INFORMAČNÍ SYSTÉMY

JIŘÍ HRONEK

VÝVOJ TOHOTO UČEBNÍHO TEXTU JE SPOLUFINANCOVÁN
EVROPSKÝM SOCIÁLNÍM FONDEM A STÁTNÍM ROZPOČTEM ČESKÉ REPUBLIKY

Olomouc 2007

Abstrakt

Tento učební text seznamuje čtenáře s vybranými partiemi z problematiky informačních systémů. Jednotlivé kapitoly popisují základní pojmy a kategorie obecných systémů, informačních podnikových systémů se zaměřením na využití databázové technologie – i nových oblastí, které nebyly uvedeny v předcházejícím kursu DBS a podporují ekonomické aplikace, například datové sklady a e-business. Přehledově jsou popsány vybrané partie z projektování, analýzy a implementace IS. Větší prostor je věnován vybraným implementačním technikám, webovým technologiím – například HTML, XML, dynamickým webovým stránkám s využitím databází. Text má usnadnit orientaci v množství používaných technologií a na vybraných příkladech naznačit praktické použití. Vzhledem k širokému spektru témat není možné popsat všechny detaily, jež jsou obsaženy v monografiích. Další studium spolu s praktickým ověřením by měly problematiku prohloubit a doplnit.

Cílová skupina

Text je primárně určen pro posluchače třetího ročníku bakalářského studijního programu Aplikovaná informatika na Přírodovědecké fakultě Univerzity Palackého v Olomouci.. Text předpokládá znalosti základních matematických pojmů, databázových systémů, základy softwarového inženýrství a programování.

Obsah

1 Vlastnosti systémů 9
1.1 Úvodní definice 9
1.1.1 Klasifikace systémů 12
1.1.2 Příklad matematického modelu 14
1.1.3 Experimentální zjištění parametrů modelu 18
1.1.4 Aspekty systémového přístupu 19
1.2 Informační systémy 19
1.2.1 Data 20
1.2.2 Informace 20
1.2.3 Znalosti, filozofie 24
2 Podnikové informační systémy 27
2.1 Vlastnosti podnikových IS 27
2.1.1 Vybrané pojmy z ekonomie 27
2.1.2 Modely podnikového IS 29
2.2 Architektura IS 32
2.2.1 Základní 3-vrstvá softwarová architektura 37
2.3 Integrace informačních systémů 37
2.4 Aplikační software 38
3 Tvorba a analýza IS 43
3.1 Tvorba systému 43
3.1.1 Metodika, metoda, technika, nástroj 45
3.1.2 Přehled vybraných metodik analýzy a návrhu IS 46
3.1.3 Objektově orientovaný přístup 47
3.1.4 Model jednání (Use Case) 47
3.1.5 Model spolupráce 48
3.1.6 Objektový model, diagram tříd 48
3.1.7 Funkční model 49
3.1.8 Dynamický model 50
3.2 Specifikace požadavků na systém 52
3.2.1 Návrh uživatelského vzhledu aplikace 52
3.3 Analýza systému 53
3.3.1 Strukturovaná analýza 54
3.3.2 Vybrané nástroje analýzy 54
3.3.3 ER-diagramy 54

3.3.4 Hierarchie funkcí 57
3.3.5 Diagramy datových toků 58
3.3.6 Stavový diagram 59
3.3.7 Datový slovník – popis metadat 59
3.3.8 Klasické metodologie analýzy 60
3.4 Strukturovaný návrh 63
4 Podpora podnikových IS 65
4.1 Datové sklady a OLAP 65
4.1.1 Datový sklad 67
4.1.2 Datový model 69
4.1.3 Komponenty DW 71
4.2 e-Business a e-Commerce v řízení firmy 71
5 Implementace IS s databází 76
5.1 Přístup k datům 76
5.1.1 Technologie firmy Microsoft pro přístup k datům 77
5.1.2 Postup při vytváření klientské aplikace v prostředí MS Visual Studio 2005 86
5.1.3 Technologie Java pro přístup k datům 89
5.2 Internetovské aplikace 91
5.2.1 Úvod – základní technologie 91
5.2.2 SGML, HTML 95
5.2.3 XML 101
5.2.4 DTD 114
5.2.5 XML Schema 119
5.2.6 Kaskádové styly 123
5.2.7 XSL 128
5.3 Dynamické stránky – klientské technologie 137
5.3.1 JavaScript 138
5.3.2 Java Applty a ActiveX 142
5.4 Serverové www technologie 143
5.4.1 PHP 144
5.4.2 ASP 148
5.4.3 ASP.NET 153
5.4.4 ASP.NET 2.0 156
5.4.5 Webové služby (web-service) 157
6 Závěr 162
7 Seznam literatury 163
8 Seznam obrázků 164

9 Rejstřík 165

1 Vlastnosti systémů

Studijní cíle: Student by měl po prostudování kapitoly rozumět základním pojmům z oblasti teorie systémů, popsat různé kategorie systémů, rozlišit jejich matematický popis a chování. Měl by umět definovat základní pojmy z informačních systémů (např. data, informace, znalosti, filozofie), jejich strukturu.

Klíčová slova: Systém, subsystém, prvek, hranice systému, okolí, stav, chování a stabilita systému, zpětná vazba, diferenciální a diferenční rovnice a transformace, data, informace, znalosti, filozofie

Potřebný čas: 4 hodiny

1.1 Úvodní definice

Vznik teorie systémů byl vyvolán praktickými problémy s růstem složitosti technických a ekonomických projektů. Systém je charakterizován s akcentem na určité pojmy a vlastnosti podle pohledu a praktického využití jednotlivých vědeckých disciplín. S využitím teorie systémů se můžeme setkat od filosofie, přes fyziku a technické aplikace až k informatice. Proto nalezneme různé definice pojmu :

Systém je

1. organizovaná množina myšlenek, principů, doktrín, seskupená za účelem vysvětlení vnitřního uspořádání nebo činnosti celku
2. soustava zvolených principů pro řešení určitých celospolečenských problémů (sociální systémy)
3. množina komponent (prvků), která interaguje, aby splnila nějaký cíl
4. pravidelně se ovlivňující nebo vzájemně závislá skupina položek, která je chápána jako celek.

Norbert Wiener (26. 11. 1894 - 18. 3. 1964)

Americký matematik, zakladatel kybernetiky. Zabýval se matematickou analýzou, teorií pravděpodobnosti, matematickou statistikou a výpočetní technikou.

Obr. 1 Norbert Wiener

Charakteristiky systému – komponenty (subsystémy, prvky), hranice, okolí, rozhraní(vstupy, výstupy), chování(procesy)

Teorie má do značné míry formální, logicko-matematickou a metodologickou povahu. Usnadňuje zejména studium různorodých spojení příčin a následků, které se vyskytují ve skupinách rozmanitých jevů nebo prvků, které na sebe vzájemně působí. Každý systém používá zjednodušující model, zahrnující jen vybranou část těchto projevů. Proto obvykle předpokládáme, že části, které jsme do systémového modelu nezahrnuli, neovlivňují vývoj modelovaného procesu do té míry, že se výsledné závěry stanou nepoužitelnými. Základní atributy systému:

- *struktura* – je definovaná množinou všech prvků a vazeb (vztahů, relací) mezi prvky, resp. dalšími různými podsystémy daného systému. Analýza chování systému vede k dekompozici zkoumaného systému na subsystémy, či prvky a vazby mezi nimi. Systém tedy můžeme graficky zobrazit jako reprezentaci

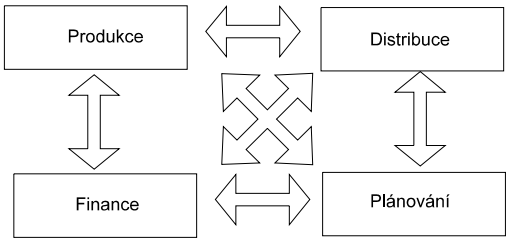
fyzické konfigurace reálných objektů, nebo jejich skupin s příslušnými vazbami nebo propojeními.

Prvky systému jsou obecně elementární, dále nedělitelné části systému, které představují jeho rozkladové (dekompoziční) části, tedy taková část systému, která tvoří na dané rozlišovací úrovni nedělitelný celek, jehož strukturu nechceme, nebo nemůžeme rozlišit. Prostřednictvím prvků a subsystémů popisujeme strukturální vlastnosti systému.

co bude prvkem záleží na rozlišovací úrovni (organizace, oddělení,zařízení ,součástka)

S*ubsystémy* – podmnožina prvků systému, kterou můžeme obvykle můžeme hodnotit jako samostatný systém se specifickou charakteristikou. Proces rozkladu systému na subsystémy je vhodný pro analýzu a vede k většímu počtu rozhraní (mezi subsystémy).

Systém i prvek jsou pojmy relativní.



Počet rozhraní je n(n-1) /2 , kde n je počet subsystémů

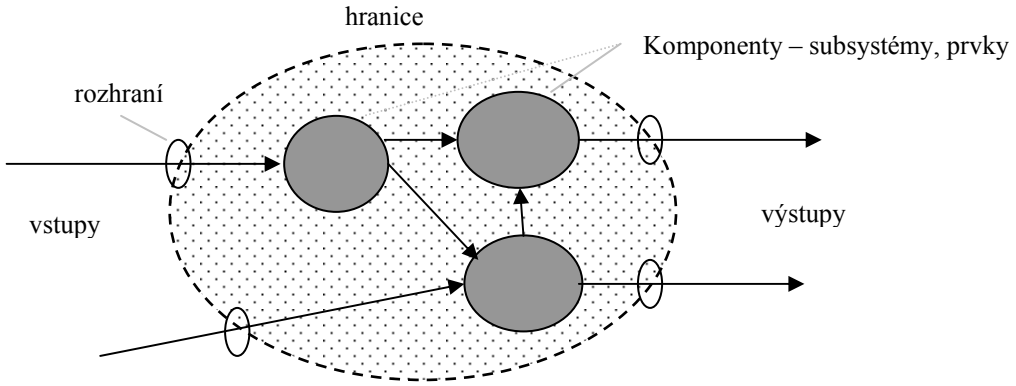
Separabilita systému,subsystému – systém je separabilní, jestliže jeho výstupy zpětně vlivem prostředí podstatně neovlivňují vstupy, případně ostatní subsystémy neovlivňují separabilní subsystém.

Okolí systému (prostředí) - pod pojmem okolí systému budeme rozumět takovou entitu, která je zdrojem podnětů působících na systém a která přijímá reakce systému na tyto podněty.

- všechny prvky, které neleží v systému, ale za jeho hranicí
- přes vstupy ovlivňují systém nebo přes výstupy jsou systémy ovlivňovány
- jejich změny stavu, chování nebo vlastností ovlivňují systém nebo jsou ovlivňovány jím

Hranice systému

- obvykle uzavírá systém nebo odděluje 2 nebo více systémů
- logická hranice - podsystém
- prostorová hranice - fyzická, nějak prostorově souvisí (otevřené okno)
- propojují se prostřednictvím výstupu jednoho a vstupu jiného



Ob. 2 Popis systému

Vstup systému – množina vazeb či proměnných, jejichž prostřednictvím se uskutečňuje působení okolí na systém.

Výstup systému – množina vazeb či proměnných, jejichž prostřednictvím se uskutečňuje působení systému na jeho okolí.

- *chování* – je projevem dynamiky systému. Dynamika je schopnost vyvolat změnu v systému, zejména jeho stavu. Chování prvků ve vzájemných interakcích, které vyplývají ze struktury systému, určuje chování celého systému. Popisujeme je pomocí vstupních, stavových a výstupních charakteristik. Podle vstupů a vnitřních stavů prvku popisujeme výstupy prvku. Domény uvedených charakteristik tvoří prostory. Prvky těchto prostorů někdy nazýváme vstupní, stavové a výstupní vektory prvku.

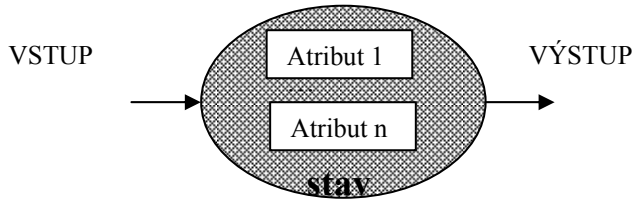
Dynamika je vlastností prvků systému, vazby jsou jejími iniciátory - vstupy, nebo nositeli důsledků -výstupy

- *stav sytému* - je souhrn hodnot jeho atributů, vlastností, které lze rozpoznat v daném časovém okamžiku za přesně definovaných podmínek. *Událost* je změna hodnoty některého atributu (prvku), nebo může být spojena se změnou konfigurace (propojení), se začleněním či vyčleněním prvku do (ze) systému. Přesněji pod pojmem událost budeme rozumět nejen změnu stavu prvku (hodnoty některé stavové proměnné prvku), ale také změnu hodnoty vstupní nebo výstupní proměnné prvku v určitém časovém okamžiku.
- *stabilita* je schopnost systému udržovat si při změně vstupů a stavů svých prvků nezměněné chování i přes působení procesů probíhajících uvnitř systému. Stabilitu chápeme jako vlastnost zaručující, že i po určité malé změně podmínek nastane v systému při nezměněných vstupech pohyb jen málo odlišný od původního.

Průvodce studiem

Komponenty, podsystémy, prvky:
- charakterizují vstupy, výstupy, vnitřní stavy , popis chování
- na základě chování částí a struktury (propojení subsystémů) mohu definovat chování celého systému

Nebudeme používat formální popis systému, protože v kontextu tématiky není důležitý, jen pro ilustraci :



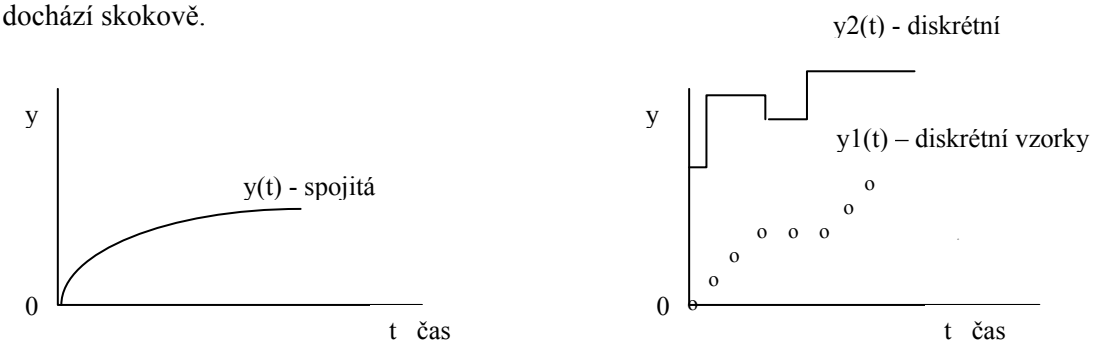
Obr. 3 Prvek systému

Zavedeme pojem atribut prvku, což je veličina, která charakterizuje vybranou vlastnost určitého prvku systému.

Chováním systému se rozumí závislost reakcí systému na podněty z okolí sledovaná v čase.

časová množina - podmnožina časů t_i (hodnoty jsou vztaženy k určitému okamžiku, nejčastěji začátek pozorování apod. a jsou z nějakého daného či zvoleného časového intervalu)

Stavu systému lze v libovolném časovém okamžiku t z časové množiny přiřadit hodnoty sledovaných atributů, které tvoří *stavový vektor*, jehož složky nazýváme *stavovými veličinami* (proměnnými) a prostor všech možných hodnot stavových veličin nazýváme *stavovým prostorem*. Podle charakteru a vývoje hodnot stavu systému lze systémy kategorizovat. Typicky rozeznáváme *spojitý průběh* - ke změnám dochází spojitě nebo *diskrétní průběh* - ke změnám dochází skokově.



Obr. 4 Spojitá a diskrétní funkce

1.1.1 Klasifikace systémů

1. uzavřené (konzervativní, autonomní) x otevřené
   - podle toho, zda nastává (otevřené) nebo nenastává (uzavřené) interakce s okolím,

a) U otevřeného dochází k energetické a informační výměně s okolím, zpracovávají neočekávané vstupní hodnoty, jsou adaptivní, reagují takovým způsobem, aby pokračovala jejich existence.

b) Uzavřený systém je zcela izolován, nemá se svým okolím žádné vazby, trpí entropií nebo poruchami.

2. deterministické x nedeterministické (stochastické)

Deterministické systémy operují a řídí se předurčenou množinou pravidel a zákonitostí, jejich následné chování je jednoznačně určeno aktuálním stavem , charakteristikami systému a vstupními veličinami.

Nedeterministické systémy jsou řízeny náhodnými událostmi, jejich chování je dáno více pravděpodobností, než jistotou, množina pravidel, ovlivňujících chování systému je neznámá, nebo příliš složitá a rozsáhlá.

Deterministické a nedeterministické prvky

Jestliže chování prvku (výstup), je určeno jednoznačně ze vstupů a jeho vnitřního stavu klasifikujeme jej jako prvkem s deterministickým chováním. Jestliže to neplatí prvek označíme jako prvek s nedeterministickým chováním.

Stochastické prvky

Prvek se stochastickým chováním můžeme popsat chování popsat stochastickými závislostmi, tedy vstupy , výstupy a stavy jsou dány náhodnými veličinami. Zobrazení, která popisují chování prvku se nazývají stochastické procesy. Podle tvaru časové množiny se rozlišují na diskrétní – s konečným nebo spočetným souborem časových okamžiků, nebo spojité - časová množina tvoří intervalem, stochastické prvky. Přechod z jednoho stavu prvku do druhého, případně výstup odpovídající danému stavu, je ohodnocen určitou pravděpodobností. podle toho, zda lze nebo nelze jednoznačně určit pravidla pro chování systému.

Stochastické prvky jsou zvláštním případem prvků nedeterminist ických.

- příliš rozsáhlé a složité systémy (nemůžeme nebo nechceme popsat deterministické chování)
- měřením, nebo statistickým zpracováním vstupů, vnitřních stavů a výstupů můžeme odhadnout stochastické charakteristiky.

3. dynamické x statické
   - výstup závisí pouze na vlastnosti vstupu (statické)
   - pamatuje si vnitřní stav (dynamické)
   - dynamika - diferenciální rovnice
   - statika - lineární rovnice
4. diskrétní x spojité x kombinované

Spojité prvky
Prvek označíme jako prvek se spojitým chováním, jestliže jeho časová množina je interval.
Diskrétní prvky
Prvek nazveme prvkem s diskrétním chováním, je-li časová množina prvku dána konečným nebo spočetným souborem časových okamžiků.

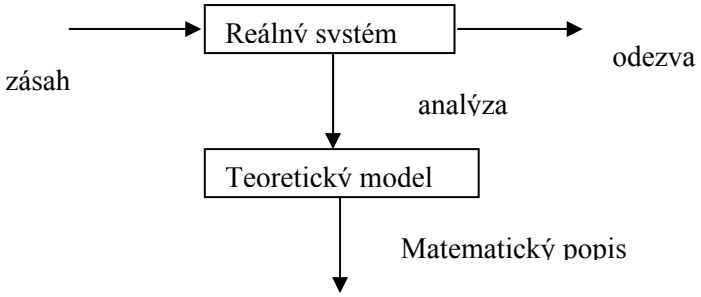
Spojité systémy
Systém se nazývá spojitým systémem (systémem se spojitým chováním), jestliže všechny jeho prvky mají spojité chování. U těchto systémů budou atributy vyjádřeny pomocí vektorů hodnot (vstupních, výstupních nebo stavových), přičemž zde může existovat i jejich nespojitost, která je podmíněna událostmi spojitými i nespojitými (dopředu známými i neznámými). Z matematického hlediska jsou spojité systémy popsané v kombinaci s diferenciálními rovnicemi obyčejnými i parciálními, diferenčními rovnicemi, algebraickými rovnicemi, apod. Jako příklady lze uvést fyzikální systémy, ale také ekonomické systémy.

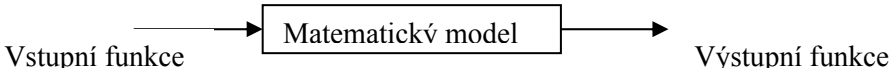
reálný objekt - část reality, která je součástí nebo tvoří systému. Může být přirozený, umělý, existující, plánovaný, atd.

Systémová analýza

Obvykle se zabývá systémy, které vytvořili lidé. Její úloha spočívá v identifikování informace požadované systémem pro vytvoření jistého rozhodnutí a také pro definici takového formačního systému, který toto umožňuje.

Definice systému představuje proces určení toho, co nás zajímá, výběr podstatných složek (prvků, vazeb) a vlastností. Obvykle vytvoříme *model*, který představuje abstraktní systém – zjednodušenou variantu reálného objektu, který obsahuje jeho abstraktní popis - soubor vztahů, případně instrukcí pro generování dat popisujících chování reálného objektu. Cílem je formální (matematický) popis, který používá pro vyjádření vztahů mezi prvky systému a hodnotami jejich proměnných matematický zápis (diferenciální či diferenční rovnice, logická pravidla, formalismy teorie automatů, …), umožňuje snadnou kontrolu úplnosti, konzistence a jednoznačnosti popisu systému.





Obr. 5 Vytváření matematického modelu

Matematické prostředky se různí, stejně jako při klasifikaci, podle:

1. typu časové základny (spojité, diskrétní, nezávislé na časovém měřítku)
2. charakteru proměnných (spojité, diskrétní, logické)
3. determinovanosti proměnných a parametrů (deterministické, nedeterministické, pravděpodobnostní, fuzzy,…)
4. vztahu k okolí (autonomní, neautonomní)
5. proměnnosti parametrů (lineární, nelineární, časově proměnné)
6. vztahu k minulosti (bez paměti, s pamětí)

Abychom mohli úspěšně řešit praktické problémy (analyzovat nebo vytvořit systém), potřebujeme reálné průběhy vstupních, stavových i výstupních veličin vyjádřit matematicky jejich abstraktními modely, při splnění dvou základních požadavků - výstižnosti, přesnosti a také jednoduchosti i snadná manipulace.

deterministické modely – průběh veličiny je jednoznačně popsán nějakou matematickou analytickou funkcí; známe-li tuto funkci, jsme schopni určit hodnotu v jakémkoliv čase

náhodné modely – neumíme jednoznačně popsat analytickou funkcí průběh veličiny, k popisu používáme parametry náhodných procesů

1.1.2 Příklad matematického modelu

Pro ilustraci popišme příklad matematického modelu pro většinu linearizovatelných fyzikálních a technických systémů. Obecně, spojitý systém n-tého řádu popisuje diferenciální rovnice n-tého řádu :

$$b_n y(n) + b_{n-1} y(n-1) + \ldots + b_0 y = a_m x(m) + {}_{am-1} x(m-1) + \ldots + a_0 x \ ,$$

která je, za předpokladu že parametry a_m, a_{m-1}, …, a_0, bn, bn-1, …, b0 jsou konstantní.

Technické aplikace většinou používají pro analýzu i syntézu vhodné lineární transformace. Zde se nabízí například Laplaceova transformace. Ta, podobně jako další podobné transformace, např. Fourierova spojitá i diskrétní, z – transformace, umožňuje převádět za určitých podmínek diferenciální nebo diferenční rovnice na algebraické v prostoru obrazů a tím usnadňuje jejich analytické řešení. Připomeňme definici a některé její vlastnosti. Laplaceova transformace je integrální transformace, která převádí funkci y(t) reálné proměnné t („předmět“) na funkci Y(p) obecně komplexního argumentu p („obraz“). Je definována vztahem

$$\mathcal{L}\{y(t)\} = \int_0^\infty y(t) e^{-pt} dt = Y(p)$$

a transformovaná funkce $y(t)$ musí splňovat tyto podmínky:

je po částech spojitá,

je exponenciálního řádu, tj. $|y(t)| \le M e^{ct}$ pro $t \ge 0$. $(M > 0, c \in R)$,

musí pro ni platit, že je $y(t) = 0$ pro $t<0$

Vybrané vlastnosti Laplaceovy transformace jsou:

linearita $\mathcal{L}\{A.f(t)+B.g(t)\} = A\mathcal{L}\{f(t)\}+B\mathcal{L}\{g(t)\}$ A, B ... konstanty

obraz derivace $\mathcal{L}\{y^{(n)}(t)\} = p^n Y(p) - p^{n-1}.y(0) - p^{n-2}y'(0) - \dots - y^{(n-1)}(0)$

věta o konečné hodnotě $\lim_{t\to\infty} y(t) = \lim_{p\to 0} pY(p)$

násobení e^{at} $\mathcal{L}\{e^{-at}.y(t)\} = Y(p+a)$

Protože určování obrazů integrací podle definiční rovnice je pracné, byl vypracován slovník Laplaceovy transformace, kde jsou k často se vyskytujícím předmětům uvedeny odpovídající obrazy. Několik základních položek slovníku uvádíme v následující tabulce:

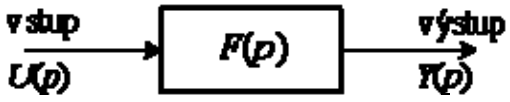
předmět	název funkce	obraz
$\delta(t)$	Diracův impuls	1
1(t)	jednotkový skok	$\frac{1}{p}$
t	čas (nezávisle proměnná)	$\frac{1}{p^2}$
t^n	n-tá mocnina času (nezávisle proměnné)	$\frac{n!}{p^{n+1}}$
$y'(t)$	1.derivace (nulové počáteční podmínky)	$pY(p)$
$y^{(n)}(t)$	n-tá derivace (nulové počáteční podmínky	$p^n Y(p)$
$\int_0^t y(t)\,dt$	integrál (podle nezávisle proměnné)	$\frac{Y(p)}{p}$

- Přenosová funkce, algebra blokových schémat

Chování systému můžeme vyjádřit pomocí tzv. přenosové funkce (obrazového přenosu), která je definována vztahem

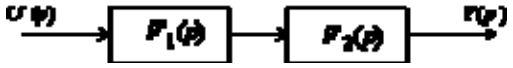
$$F(p) = \frac{\text{Laplaceův obraz výstupu } y(t)}{\text{Laplaceův obraz vstupu } u(t)} = \frac{Y(p)}{U(p)}$$

Schématicky se vyjadřuje systém jako blok charakterizovaný svou přenosovou funkcí:



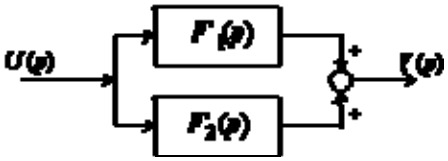
Takto definované bloky můžeme libovolně spojovat (sériově, paralelně, kombinovaně,...) a přenosová funkce výsledné kombinace se jednoduše určí pomocí algebraických operací. Pro přenosové funkce základních kombinací platí:

sériové zapojení bloků



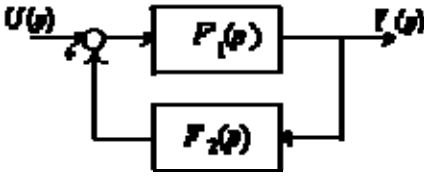
$$F(p)=\frac{Y(p)}{U(p)}=F_1(p)\,F_2(p)$$

paralelní zapojení bloků



$$F(p)=\frac{Y(p)}{U(p)}=F_1(p)+F_2(p)$$

antiparalelní zapojení bloků (ve zpětné vazbě)



$$F(p)=\frac{Y(p)}{U(p)}=\frac{F_1(p)}{1+F_1(p)\,F_2(p)}$$

Výhodou algebry blokových schémat je, že můžeme vytvářet v podstatě libovolné systémy vzájemným propojováním dílčích (stavebnicových) bloků a poměrně jednoduchým výpočtem určovat jejich výslednou přenosovou funkci.

Pro obecnou diferenciální rovnici n-tého řádu má přenosová funkce lineárního systému za předpokladu nulových počátečních podmínek tvar

$$\mathrm{F(p)}=\frac{\mathrm{Y(p)}}{\mathrm{X(p)}}=\frac{\mathrm{a_m p^m}+\mathrm{a_{m-1}}.\mathrm{p^{m-1}}+\mathrm{a_{m-2}}.\mathrm{p^{m-2}}+\ldots+\mathrm{a_1 p}+\mathrm{a_0}}{\mathrm{b_n p^n}+\mathrm{b_{n-1}}.\mathrm{p^{n-1}}+\mathrm{b_{n-2}}.\mathrm{p^{n-2}}+\ldots+\mathrm{b_1 p}+\mathrm{b_0}}$$

polynom ve jmenovateli přenosové funkce $b_n p^n + b_{n-1}.p^{n-1} + b_{n-2}.p^{n-2} + \ldots + b_1 p + b_0$

nazýváme charakteristickým polynomem systému a rovnici

$$b_n p^n + b_{n-1}.p^{n-1} + b_{n-2}.p^{n-2} + \ldots + b_1 p + b_0 = 0$$

charakteristickou rovnicí systému. Řešením charakteristické rovnice dostaneme n jejích kořenů p_i, i=1,…,n. Podobně můžeme určit i kořeny z_j, j=1,…,m rovnice, která vznikne položením polynomu v čitateli přenosové funkce rovno nule. Dekompozice čitatele i jmenovatele na součin kořenových činitelů (z jsou nulové body a p jsou póly) a parciální zlomky zjednoduší zpětnou transformaci. Jednotlivé členy součinu mohou mít podle stupně polynomu v čitateli a jmenovateli charakter proporcionálních, derivačních nebo integračních členů a můžeme si systém představit jako sériově seřazené prvky s jednoduchým chováním. Většinou jen několik málo členů součinu zásadně ovlivňuje chování, což dovoluje zjednodušování při tvorbě modelu. Proměnná p má obecně komplexní charakter a tedy nabývá tvaru

$$\mathrm{p} = \sigma + \mathrm{j}\omega\,,$$

kde σ je koeficient tlumení a ω = 2πf je kruhová frekvence. Předpokládejme, že koeficient tlumení σ = 0, pak po dosazení za p v operátorové přenosové funkci dostáváme

$$F(j\omega)=\frac{Y(j\omega)}{X(j\omega)}=\left|F(j\omega)\right|.e^{j\varphi(\omega)}$$

což představuje frekvenční přenosovou funkcí systému, která se využívá hlavně v systémech pro zpracování signálů (zvuk, obraz, video), v diskrétní podobě i v počítači nebo v systémech s regulací a řízením. Frekvenční charakteristika systému je grafické vyjádření frekvenční

přenosové funkce systému amplitudy přenosu pro frekvence, prakticky pouze v intervalu $0 \le \omega < \infty$. Z průběhu frekvenční charakteristiky se dá určit chování systému.

Další používaná transformace, zobecňující Fourierovu řadu pro periodické časové funkce, používaná pro určení frekvenčních charakteristik systému je Forierova transformace, která převádí neperiodickou funkci s(t) z časové domény na funkci S(ω) v kmitočtové oblasti. Je definována vztahem :

$$s(t) = \int_{-\infty}^{\infty} \frac{d\omega}{2\pi} \left(\int_{-\infty}^{\infty} s(t).e^{-j\omega t} dt \right).e^{j\omega t}$$

Označíme

$$S(\omega) = \int_{-\infty}^{\infty} s(t).e^{-j\omega t} dt$$

Funkce S(ω) se nazývá spektrální a její průběh opět charakterizuje chování systému. Pro časovou funkci můžeme psát vztah pro zpětnou Fourierovu transformaci

$$s(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} S(\omega).e^{j\omega t}.d\omega$$

Transformace má podobné vlastnosti, jako Laplaceova.

Platí translace funkce $s(t-\tau) \approx S(\omega).e^{-j\omega\tau}$

Transpozice spektra $S(\omega-\Omega) \approx s(t).e^{-j\Omega t}$

Konvoluce funkcí $$s_1(t) * s_2(t) = \int_{-\infty}^{t} s_1(x).s_2(t-x).dx \approx S_1(\omega).S_2(\omega)$$

Pro diskrétní systémy se používá diskrétní Fourierova transformace (DFD) nebo z-transformace. Souvislost s Laplaceovou transformací zjednodušeně naznačíme. Uvažujme spojitou funkci x(t). Vzorkujeme-li tuto funkci vzorkovačem s periodou T, dostaneme diskrétní funkci x(kT) a její Laplaceův obraz získáme stejně, ale musíme přejít od integrálu k sumě

$$L\{x(kT)\} = \sum_{k=0}^{\infty} x(kT)e^{-pkT}$$

Zavedeme-li novou proměnnou z vztahem $z = e^{-pT}$ dostaneme

$$X(z) = Z\{x(kT)\} = \sum_{k=0}^{\infty} x(kT)z^{-k}$$

Aby bylo možné počítat s frekvenčním spektrem na počítači, je třeba funkci X(z) (podobně je to u spektrální funkce) diskretizovat. DFT je potom definována vztahem

$$X(k\Omega) = \sum_{n=0}^{N-1} x(nT).e^{-jk\Omega nT} = \sum_{n=0}^{N-1} x(nT).e^{-jk\frac{2\pi}{NT}nT} = \sum_{n=0}^{N-1} x(nT).e^{-j2\pi kn/N}$$

$$X(k) = \sum_{n=0}^{N-1} x(n).e^{-j2\pi kn/N}$$

Vztah DFT a z-transformace :

$$\mathcal{Z}\{x(n)\} = \sum_{n=0}^{\infty} x(n).z^{-n}$$

$$\mathcal{DFT}\{x(n)\} = \sum_{n=0}^{N-1} x(n).e^{-jk\Omega nT} = \mathcal{Z}\{x(n)\}\Big|_{z=e^{jk\Omega T}}$$

Všechny uvedené i další obdobné transformace se používají v technicky orientovaných počítačových aplikacích, jako je simulace chování systémů, počítačem řízená regulace, ale i zpracování signálů (číslicová filtrace) a vyhledávání informací (z multimediálních dat), komprese dat.

Průvodce studiem

Laplaceova, Fourierova spojitá i diskrétní transformace, z- transformace jsou matematické prostředky pro analýzu i syntézu většinou technických lineárních spojitých i diskrétních systémů. Mají společnou vlastnost, že jsou schopny popsat chování systému z frekvenčních a dalších charakteristik (typicky reakce na standardní vstupní signály – harmonický, jednotkový skok, ...), které se dají experimentálně měřit. Mohou tím sloužit k určení návrhu i definici parametrů modelu systému.

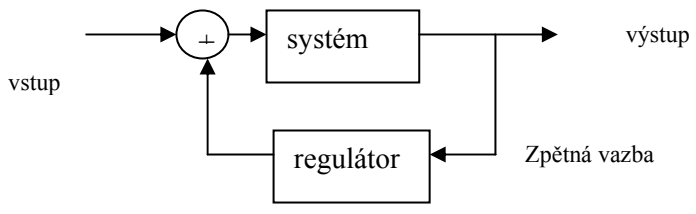
1.1.3 Experimentální zjištění parametrů modelu

Inspirativní na technických systémech je postup, kdy neznáme strukturu a charakteristiky komponent systému - modelovaný systém je „černá skříňka“ s definovanými vstupy a výstupy. Tento přístup je typický u elektrotechnických systémů, kdy testovací vstupní signály jsou harmonické s konstantní amplitudou a různou frekvencí signálu (amplituda výstupního signálu odpovídá jednomu bodu frekvenční charakteristiky) nebo skoková funkce z nuly na konstantní velikost (jednotkový skok). Model obecně vytváříme zjednodušeně následujícím postupem:

1. na základě předpokládaného chování systému zvolíme vhodný tvar matematického popisu, který obsahuje parametry neznámých hodnot
2. na vstup systému přivádíme proměnlivý signál (může být skutečný nebo uměle generovaný) a měříme hodnoty vstupního a výstupního signálu tak dlouho, až získáme dostatečně reprezentativní řady odpovídajících dvojic hodnot,
3. výsledky měření zpracujeme tak, abychom získali hodnoty parametrů zvoleného matematického popisu systému.

Poté, co můžeme popsat chování systému, snažíme se systém ovlivnit tak, aby optimálně fungoval. Tomuto procesu říkáme řízení systému. Řízení sytému je aktivita, kdy pomocí zjištěné odchylky od plánovaného cíle a z ní odvozené informace následuje korekce chování systému pro splnění cíle. V tomto procesu hraje důležitou roli pojem zpětná vazba. Zpětná vazba je vazba mezi výstupem a vstupem stejného prvku, subsystému nebo systému, která způsobuje to, že vstup je závislý na výstupu.

Řízení je cílené působení na řízený objekt tak, aby se dosáhlo určitého předepsaného cíle.



Prvky řídicího cyklu jsou:

-standard specifikující očekávaný cíl
-měření aktuálního stavu cíle
-porovnání aktuálního a požadovaného cíle
-zpětná vazba do řídicí jednotky
-akce řídicí jednotky, ovlivňující systém

Druh zpětné vazby v závislosti na typu reakce vstupu na výstup

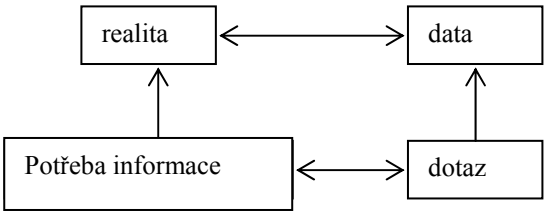
- Pozitivní (kladná) zpětná vazba - vstup se mění ve stejném směru jako výstup (s růstem výstupu vstup roste, s poklesem výstupu vstup klesá), vede k nestabilitě systému
- Negativní (záporná) zpětná vazba - vstup se mění v opačném směru než výstup (s růstem výstupu vstup klesá, s poklesem výstupu vstup roste), užití v regulaci a řízení.

1.1.4 Aspekty systémového přístupu

- synergie - spolupráce jednotlivých částí přináší větší efekt než pouhý součet jejich prací
- žádná část nemůže být změněna bez ohledu a bez vlivu na ostatní části (silná orientace na výsledný cíl)
- subsystémy systému by měly pracovat pro cíl systému, i kdyby to znamenalo, že je to proti jejich optimální dílčí činnosti („jedinci se obětují pro celek“)

1.2 Informační systémy

Na termín informační systém existuje mnoho různých pohledů a jejich kontextů. Základní informatický pohled na informační systém zobrazuje následující schéma :

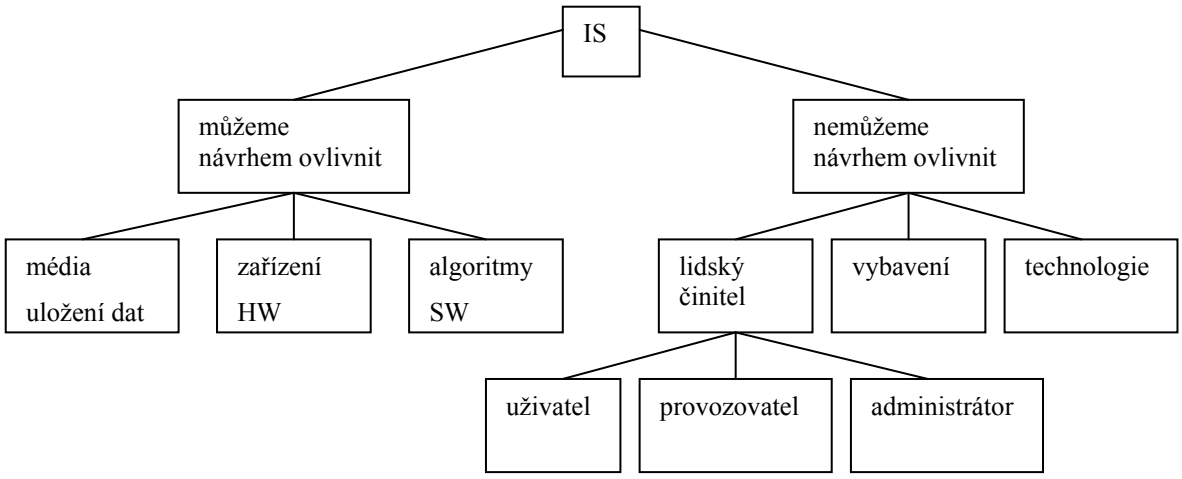


- první princip abstrakce - realita je reprezentována kolekcí dat vzniklých pozorováním nebo zkoumáním části reálného světa

realita - chápána jako nezávislá na individuálním náhledu

- Uživatel (nebo administrátor) používá informační systém dvěma způsoby:
  1. uloží informace (data), u kterých se předpokládá příští využití při dotazování ve vhodné formě – z pohledu účelné reprezentace reality i efektivity dotazování.
  2. nalézá informace (data) jako odezvu na požadavek uživatele

Definice: Informační systém je systém, umožňující účelné uspořádání sběru, uchování, zpracování a poskytování informací. V reálném IS rozeznáme dvě složky.
Ektosystém se skládá z uživatelů IS, investora IS a toho, kdo systém provozuje (user, funder, server). Ektosystém není pod kontrolou projektantů při návrhu IS.
Endosystém se skládá z použitého hardware (media, zařízení), a software (algoritmy, datové strutury) a je plně pod kontrolou designéra IS.
Informační systém = automatizovaná část (využívající počítače) + neautomatizovaná část informačního systému



Obrázek 6 Struktura IS

1.2.1 Data

Data představují fakta, měření, text, obraz, zvuk, video, nejčastěji v kontextu sledovaného procesu nebo situace. Vlastnosti dat:

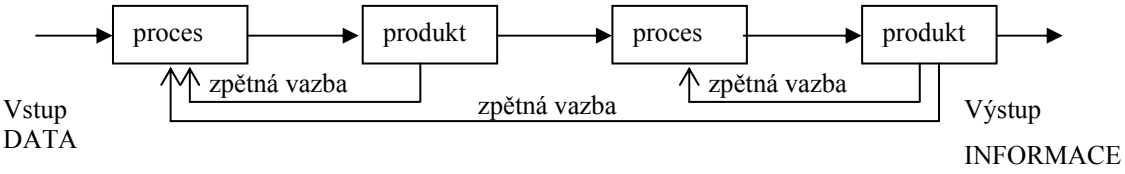
- nezávislá na uživateli, většinou odráží současný stav reality

- vždy zjednodušují komplexnost reality (jsou nekompletní). Při rozvoji IS to vede k procesu změn a inovací, v jejichž důsledku se buď dodávají nová data, nebo zjemňují, či zpřesňují (strukturálně) stará data.

- přesto představují velký objem, často mnoho detailů

- poměrně často a rychle se mění

- verifikovatelnost dat je možnost u většiny případů opakovaně objektivně ověřit správnost a přesnost dat (opakováním měření, pozorování).



Typický datový proces:

Informace je odpověď na otázku.

Informace je schopnost organizovat nebo v organizovaném stavu udržovat.

1.2.2 Informace

Pojem *informace* je možno chápat v různých základních významech:

1. „laický“ - sdělení, zpráva, jazykový projev, ve kterém se co nejobjektivněji věcně konstatují určitá fakta, znalost sdílená tím, že se komunikuje - sdělitelná znalost

2. význam „sémantický“(kvalitativní) - jedná se o absolutní zisk informace jak např. odpovídá významu jednotlivých slov
3. význam „pragmatický“ (předchozí ovlivňuje následné) - jedná se o relativní zisk informace - příjemce rozlišuje zda již sdělovanou informaci má, či ne - sdělení toho, co už vím, není podle tohoto pojetí ziskem informace
4. význam idealizovaný - zisk informace záleží na jeho zhodnocení příjemcem a to na základě jeho předchozích zkušeností, minulých i momentálních citů a emocí, logika přitom hraje zanedbatelnou roli - např. stejný obraz nebo stejná hudba je různými příjemci vnímána a hodnocena odlišně, dokonce i jedním příjemcem může být vnímána odlišně v různých časových okamžicích,
5. význam kybernetický - Zpráva o objektivní realitě, která funguje jako zpětná vazba, část poznání, která se používá k orientaci, k aktivní činnosti, k řízení - s cílem zachovat kvalitativní specifičnost systému a tento systém zdokonalovat a rozvíjet, proces, kdy určitý systém předává jinému systému pomocí signálů zprávu, která nějakým způsobem mění stav přijímacího systému
6. význam matematický, inženýrský (kvantitativní) - energetická veličina, jejíž hodnota je úměrná zmenšení entropie systému. Vychází z toho, že existuje otázka a k této otázce konečná množina možných odpovědí. To zakládá pravděpodobnost toho, že z množiny odpovědí bude právě jedna vybrána. Tato pravděpodobnost je vyčíslitelná. Předpokládá se tedy snížení neurčitosti vzniklé otázkou na základě odpovědi, která je možná jako výběr jedné odpovědi z množiny možných. Pravděpodobnost výběru odpovědi z dané množiny se stala základem stanovení velikosti informace pomocí informační entropie tak, jak ji stanovil Shannon v roce 1948, tj.

$$H_i = -\log p_i$$

dotazování na něco, co je obecně známé, zmenšuje množství informace a odvádí pozornost

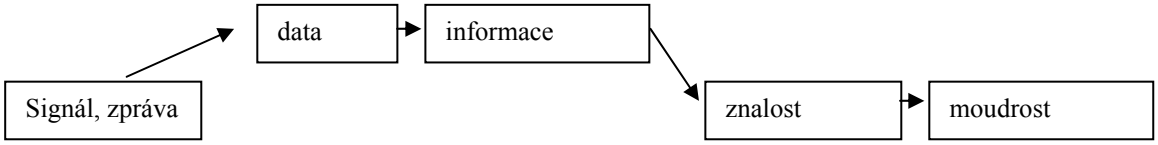
Pro nejmenší možnou informaci, tj. odpověď „ano - ne“ pak byla odvozena jednotka informace 1 bit = $-\log_2(1/2)$

informace = data + význam + struktura
Př. *Data : 1,141*
Informace : $\sqrt{2}$ *= 1,141*

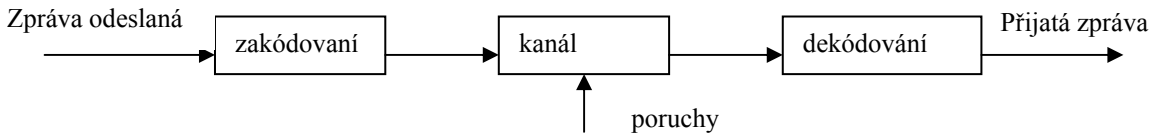
Signál je jev materiální povahy, nesoucí informaci o stavu systému, který jej generuje

Informace:

- strukturovaná, organizovaná, shrnutá a interpretovaná data, silně závislé na tom, kdo je požaduje, tedy závislé na uživateli (individuální hloubka znalostí, zkušeností...)
- množina dat jako odpověď na požadovanou informaci



- signál – nositel informace, při přenosu zprávy z jednoho místa na druhé. Pokusme se kvantifikovat množství informace dodávané zdrojem zpráv.



Obr. 7 Přenos zprávy – určení množství informace

Při uvažování poruch se může stát, že vstupní zpráva se nerovná výstupní, přijaté zprávě. Potom pro množství informace platí:

$$I = \log \frac{P(B)}{P(A)}$$

, kde P(B) je pravděpodobnost vzniku události (A) po příjmu zprávy (aposteriorní pravděpodobnost) a P(A) je pravděpodobnost vzniku události (A) před příjmem zprávy. Pokud poruchy nebudeme brát v úvahu, dostaneme :

$$I = \log \frac{1}{P(A)}$$

- Př.: abeceda obsahuje *m* možných prvků, zprávy obsahují *n* prvků, všechny prvky jsou stejně pravděpodobné. Počet různých zpráv je $N = m^n$, P(A) = 1/N. Po dosazení dostaneme výsledek, že každá zpráva nese informaci I = n log m.

Průvodce studiem

Informace je mírou redukce nejistoty, k níž dojde při přijetí zprávy. Pojem přesně definoval Claude Shannon z Bell Labs roku 1950. Její jednotkou je jeden bit, což je snížení nejistoty, k němuž dojde, pokud mohou nastat dva jevy s pravděpodobností 0.5 a jeden z nich nastane. Počet bitů informace se určí ze záporného logaritmu se základem 2 pravděpodobnosti jevu, zprávy -log2(p). Podle této definice zpráva o události s pravděpodobností 1/4 přináší dva bity informace, událost s pravděpodobností 1/8 tři bity atd.

Kvalitní informace je:

- přesná - neobsahuje chyby, je jasná a reflektuje význam dat, na kterých je založena
- včasná: potřebná informace je k dispozici ve vhodném čase
- relevantní: adekvátnost potřebě, odpovídá na otázky Co? Proč? Kde? Kdy? Kdo? Jak?
  - novost, využitelnost pro poznání a rozhodování
  - vysvětlení významu (např. "entropie = míra neuspořádanosti")
- srozumitelná
  - uspořádáním (např. abecední seznam, grafické schéma)
  - přiměřená (s jistou mírou redundance), více není lépe

Informace sama o sobě nemá hodnotu, té nabývá až v procesu využívání – rozhodující je kvalita a dostupnost, účel a kontext. Musí proto také splňovat komunikovatelnost (možnost přenášet v čase a prostoru) - šíření poznatků přenosovými cestami (kanály) a srozumitelnost - zakódování poznatku do jazyka (kódu), který je příjemci znám (kompatibilita).

Atributy (vlastnosti) informace:

1. neoddělitelnost informace od fyzikálního nosiče (relativní závislost komunikovaného obsahu na nosiči a autorovi) a současně diskrétnost (relativní nezávislost komunikovaného obsahu na nosiči a autorovi),
2. stárnutí informace - stárne nikoli s časem, ale s objevením se novější relevantnější informace
3. kumulativnost - vytvoření nové informace nezničí informaci starou (zpravidla se mění její hodnota). Někdy se dokonce hovoří o exponenciálním nárůstu, to se však týká spíše množství zdrojů než informací samotných
4. užitná hodnota - je relativní, nikoli absolutní

kvantifikovatelné atributy : - přesnost, pravdivost - množství chyb (šumy), chyby při ukládání dat (např. při přepisu), chyby při výpočtech a operacích s daty, záměrné dezinformace, přístupnost - snadnost a rychlost, s níž lze informaci obdržet (získat), rychlost, s níž lze informaci zpracovat a zprostředkovat, flexibilita - použitelnost pro více než jednoho uživatele, rozptyl - seskupení relevantních informací ve velkých souborech do zón podle určitého pravidla

nekvantifikovatelné atributy: - relevance, podrobnost, úplnost - zda obsahuje vše, co potřebujeme, přiměřenost - zda neobsahuje to, co nepotřebujeme, jasnost - stupeň nejasnosti a dvojznačnosti, ověřitelnost - stupeň konsensu, k němuž se dospěje mezi různými uživateli, kteří zkoumají tutéž informaci

informace jako ekonomický zdroj

Společné vlastnosti materiálních a informačních zdrojů - mohou být vyprodukovány, uchovány a distribuovány.

dočasnost hodnoty - obvykle ztrácí hodnotu po okamžiku, kdy byly spotřebovány.

specifika informačních zdrojů: expanzívnost (rozpínavost) - omezením je pouze čas a kapacita lidského poznání, jejich povahou je šíření (komunikace), používáním se spíše reprodukují než spotřebovávají.

informace jako komodita (zboží) - společné vlastnosti materiálního a "informačního" zboží- určení pro směnu, užitná a směnná hodnota.

Specifika informace jako zboží: méně propracované marketingové metody a techniky pro “obsah” - je snadnější propagovat nosič či technické zařízení (např. CD-ROM) než samotnou informaci.

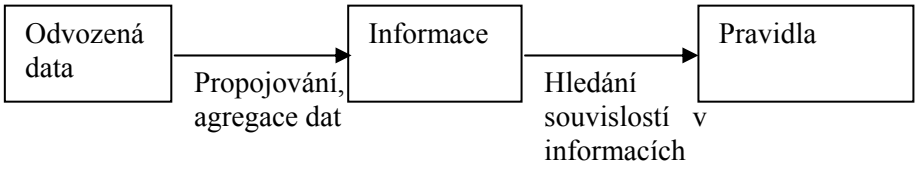
značná typová rozdílnost: např.

a) informace vysílané - informace kupované na nosiči jednotlivcem

b) informace placené uživatelem - informace placené reklamou - informace zdarma (nekomerční), placené z veřejných zdrojů, dotované např. daňovými a celními úlevami

uvedením do kontextu (souvislostí, vztahů k jiným skutečnostem)

Typický informační proces:



1.2.3 Znalosti, filozofie

Znalosti - to, co jednotlivec ví po osvojení dat a informací a po jejich začlenění do souvislostí. Jsou spoluvytvářeny individuálním vzděláním a zkušenostmi experta. Je to výsledek poznávacího procesu, předpoklad uvědomělé činnosti.

Vlastnosti znalostí:

- staví na informacích, integruje novou informaci s předešlými k zachycení komplexnějšího a abstraktnějšího pohledu na část reality
- zároveň informace jsou jen parciální - představují malou částí znalostí
- obvykle jsou méně přesné (proti datům), obsahují nejistotu, např. nová, detailnější data odporují aktuálním znalostem
- zahrnují proces abstrakce, generalizace a kategorizace dat a informací.
- představují obecné zákonitosti, které se nemění tak rychle, jako data.

znalost = informace + abstrakce + vztahy + zdůvodnění + aplikace

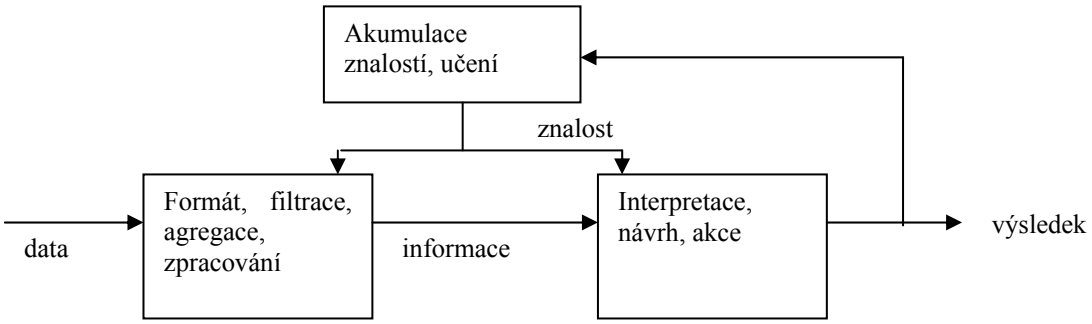
Rozdíl mezi informací a znalostí je analogický jako to, že znáte věc proti tomu, že o věci máte informaci.

Účel znalostí - umožňují porozumět skutečnosti.

- jsou využívány v procesech výběru, interpretace a rozhodování
- v procesu učení se mění, přetvářejí a rozvíjejí
- jsou základem pro práci s informacemi, vyhledávání datových zdrojů a jejich využívání.

Příklad uplatnění znalostí - formulace pravidel, popis procesu, vytvoření modelu

informace + rozhodnutí = akce

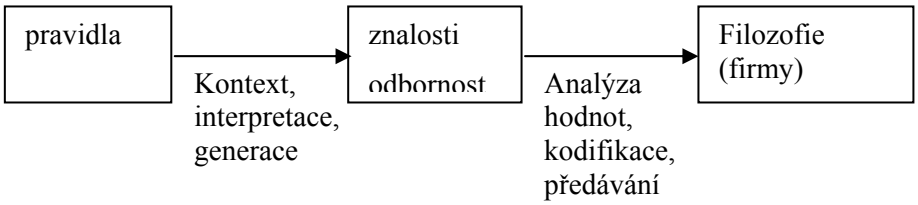


Odbornost – rychlá a přesná rada, vysvětlení a zdůvodnění, podporována s využitím subsystémů IS (základem umělá inteligence, znalostní báze, různá fakta, koncepty a pravidla (vytváří expert specialista), expertní systémy

Moudrost, filosofie

- poslední článek řetězce data - informace - znalost - porozumění - moudrost
- jasnější ucelený pohled zahrnující celou zkoumanou oblast se všemi vnějšími a obecnějšími souvislostmi
- ovládá a řídí zpracování informací
- typicky čerpá z více zdrojů, integruje znalosti
- pracuje i se zdroji, které nejsou přímo okamžitě dostupné
- představuje systematický přístup k zásadám fungování organizace, postavený na přínosech a hodnotách.

Typický znalostní proces:



Průvodce studiem

ČSN 369001:
Údaj (data): obraz vlastností objektu, vhodně formalizovaný pro přesnost, interpretaci nebo zpracování prostřednictvím lidí nebo automatů
Informace je význam, který příjemce přisuzuje údajům (datům)
Zpráva je nositelem informace, pokud přináší něco nového
Čím více zpráva snižuje nejistotu, tím silnější je korelace mezi vstupem a výstupem komunikačního kanálu.

Shrnutí Pojem systém definuje obecnou kategorii pro popis struktury (dat) a chování (zpracování dat) většiny počítačových aplikací, které zpracovávají informace. U sytémů využíváme společné vlastnosti a charakteristiky – například začlenění do kontextu – rozeznání nadsystému, dekompozice na funkční bloky a jejich vzájemné ovlivňování – subsystémy, zpětná vazba, synergie, řízení procesů. Vhodný matematický popis systému umožní odhady chování u různých kategorií systémů. Informační systémy operují s pojmy jako data, informace, znalosti. Vhodným dotazovacím aparátem poskytují uživatelům podklady pro analýzu, řízení a podporu rutinních činností většinou v ekonomických podnikových systémech.

Pojmy k zapamatování

- Systém, subsystém, prvek, hranice systému a okolí.
- Chování a stav systému, stabilita.
- Kategorie systémů: autonomní a otevřené, deterministické a stochastické, dynamické a statické, diskrétní a spojité (kombinované).
- Kladná a záporná zpětná vazba, řízení.
- Lineární transformace, řešení diferenčních a diferenciálních rovnic.

Kontrolní otázky

1. *Jaké základní pojmy popisují a charakterizují systém?*
2. *Jaké rozlišujeme kategorie systémů?*
3. *Jakým matematickým aparátem popisujeme chování systémů?*
4. *Co je a k čemu slouží zpětná vazba?*
5. *Proč je důležitá stabilita systému?*
6. *Jaká je struktura informačního systému ?*
7. *Jaké jsou rozdíly v charakteristice dat, informací a znalostí?*

Úkoly k textu

1. Vyberte vhodný jednoduchý fyzikální systém(například kyvadlo) a pokuste se matematicky a fraficky popsat jeho chování.
2. Popište příklad použití zpětné vazby.

2 Podnikové informační systémy

Studijní cíle: Studující by měl porozumět základním ekonomickým pojmům, rozdělení a cílům informačních systémů v podniku. Měl by charakterizovat jednotlivé úrovně IS a jejich typické reprezentanty na úrovni subsystémů, různé architektury IS, procesní, technologický a organizační model. Měl by rozumět procesu integrace IS, vysvětlit pojmy metodika, metoda, technila a nástroj, charakterizovat aplikační software.

Klíčová slova: Ekonomie, firma, management, informační technologie, strategické, taktické a operační informační systémy, architektura IS, procesní, technologický a organizační model, EIS, DSS, MIS, KWS, TPS, OIS, EDI, CAM, CAD, GIS, metodika, metoda, technika, nástroj, aplikační software

Potřebný čas: 3.hodiny

2.1 Vlastnosti podnikových IS

2.1.1 Vybrané pojmy z ekonomie

Ekonomie

- je společenská věda, která zkoumá, jak lidé a společnost rozhodují o využití vzácných zdrojů, které mají alternativní užití, za účelem výroby různých statku a služeb, a jak jsou tyto komodity rozdělovány pro současnou a budoucí spotřebu mezi jednotlivé osoby a skupiny ve společnosti. Tri základní otázky ekonomie zní:

Analytik

se snaží odlišit vliv jednoho faktoru na celek tím, že předpokládá,

že se ostatní faktory nemení.

- co ……. jaké statky se mají vyrábět a v jakých množstvích
- jak …… jak mají být vyráběny, tj. kdo je má vyrábět, s jakými zdroji a jakou technologií
- pro koho .. kdo je má dostat

Někdy nazývána vědou o rozhodování. Moderní ekonomie se rozšiřuje do tradičních odvětví jiných společenských věd (sociologie, politologie, historie, psychologie apod.) a používá svůj aparát k vysvětlení hraničních jevu z těchto věd.

- je založena na znalostech, informační technologii, mobilní a dálkové komunikaci a další špičkové technologii. Porovnání přístupů (vývoj ekonomiky) :

Oblast	**Ekonomika dříve**	**Ekonomika nyní**
Trhy	Relativně stabilní	Dynamické
Konkurence	Národní, místní	Globální(nadnárodní obchodní řetězce)
Forma organizace	Hierarchická	Síťová
Organ. výroby	Masová výroba	Flexibilní výroba, decentralizace, lokální nezávislost, spolupráce
Klíčové motory růstu	Kapitál/práce	Inovace/znalosti, produktivita, kvalita
Požadované vzdělání	Vyučení, maturita, VŠ	Celoživotní vzdělávání

Vztahy s ostatními firmami	Sám proti všem	Spolupráce, koordinace

Mikroekonomická analýza zkoumá postupy, jaké lidé volí v podmínkách vzácnosti. Při své analýze používají ekonomové modely. Model je zjednodušená reprezentace reality, která abstrahuje od nepotřebných detailu, čímž umožňuje zaměřit se na podstatné vlivy. Příklad modelu je např. mapa – nesmí být ani příliš moc, ani příliš málo podrobná, ale musí zachytit potřebné detaily a neobtěžovat zbytečnostmi. Pro různé účely se hodí různé (a různé podrobné) mapy. Totéž platí pro ekonomické modely. Při tvorbě modelu vychází ekonomové z pozorování a axiomů. Modely se snaží ověřovat tak, že zkoumají, nakolik se předpovědi modelu shodují s reálně pozorovanými fakty. Obvykle se spoléhají na srovnání s minulými pozorováními, protože provádění experimentu je v ekonomii obtížné.

Firma je produkční jednotka, která nespotřebovává veškerý svůj výstup a která je ekonomicky možná díky zvýšení produktivity dosaženého specializací. Jinak lze také říci, že firma je agent specializující se na výrobu, tj. přeměnu zdrojů ve statky. Příčiny existence firmy jsou v principu dva:

1) výhody týmové spolupráce a
2) snížení transakčních nákladu (nákladů na uzavírání a monitorování kontraktu apod.).
Pokud chce firma dosáhnout maximálního zisku, musí nejen vyrábět efektivně (tj. spojovat při výrobě optimální kombinace vstupu), ale také vyrábět to množství produktu, které bude maximalizovat zisk firmy. Zisk firmy je rozdíl mezi výnosy a náklady.

Management je proces plánování, organizace, řízení obchodních a podnikatelských aktivit.

Cíle podnikového IS :

- rozhodování

- koordinace a řízení subsystémů podniku

- plánování aktivit

Efektivita operací znamená provádět podobné aktivity lépe než konkurence. Strategická pozice předpokládá, že firma dělá jiné aktivity, než konkurence, nebo je dělá jinak.

Informační technologie (IT)

Souhrn všech rozličných technických (HW), telekomunikačních (sítě, mobilní zařízení), programových(SW), organizačních a jiných prostředků, technik a služeb, které můžeme využívat při jednotlivých operacích s informacemi.

Informační systém (IS)

Soubor lidí, metod a technických prostředků zajišťujících sběr, přenos, uchování, zpracování a prezentaci dat, jehož cílem je tvorba a poskytování informací podle potřeb jejich příjemců, činných v systémech řízení. Poskytuje informace potřebné pro rozhodování.

podnikový IS- ERP (Enterprice Resource Planning), plánování podnikových zdrojů , jádro podnikového informačního systému, soubor aplikací podporujících finance(majetek, pokladna, odběratelské a dodavatelské účty, náklady, …), lidské zdroje (mzdy, personalistika), výrobu a logistiku(plánování a řízení výroby, sklady, doprava, údržba, …).

Hlavní změny ve využití IS

	Hlavní technické prostředky	Oblasti využití prostředků IS
50. léta	Sálové počítače	Vědecko-technické výpočty, sběr dat
60. a 70. léta	Mainframy	Hromadné zpracování dat
80. léta	Osobní počítače a počítačové sítě	Kancelářský SW, podpora inženýrských prací (informace je strategický nástroj)

90. léta	Mobilní zařízení připojitelné k internetu	Komunikační nástroj, podpora rozhodování, (informace je zboží)

První informační systémy se zaměřovaly na zpracování evidenčních dat statistickým zpracováním získaných údajů. Později byly systémy využívány jako bilanční, které zajišťovaly provádění souhrnných bilančních výpočtů v různých variantách. Výsledky sloužily jako podklad k rozhodování. Od té doby IS můžeme začít využívat jako řídicí.

Manažéři mohou pomocí predikčních metod simulace dostat odpověď na otázku: "Co se stane, když...?" a tím prozkoumat následky možných rozhodnutí dříve, než jsou skutečně provedena. Moderní řídicí systémy jsou vybaveny schopností adaptability, aby byly schopny přizpůsobit se změněným podmínkám trhu nebo stavu firmy. To většinou znamená využít i metod umělé inteligence a koncipovat systémy tak, aby byly schopny v bázi znalostí shromažďovat sami určité poznatky a akceptovat zkušenosti, s využitím expertních systémů pro řízení – rozhodující nástup informační technologie.

Požadavky : Vytvořit pro firmu strategické výhody, rozhodovat o strategických změnách, konkrétně - zkrácení marketingových a inovačních cyklů, určování ceny, kvality, servisu, …, posílení odpovědnosti zaměstnanců, vytváření obchodních skupin, nové internetové techniky – elektronické obchodování, digitální firmy.

Průvodce studiem

Pod pojmem informační technologie (IT) rozumíme souhrn všech rozličných technických, programových, organizačních a jiných prostředků, technik a služeb, které můžeme využívat při jednotlivých operacích s informacemi (zpracování informace, ukládaní informace, přenášení informace, apod.).

Úrovně využití IS :

1. základní datová úroveň se soustřeďuje na ukládání, výběr, účinnost zpracování a rychlost předávání dat. Tuto činnost provádí transakční systémy.
2. informační úroveň představuje posun od technologického pohledu interpretací dat jako informací pomocí informačních procesů a analýz, podpora rozhodování, vzniká informační strategie firmy.
3. znalostní úroveň podporuje správu a sdílení znalostí – akumulace ve vlastní firmě, kooperace s jinými. Pomáhá s přípravou pracovníků na měnící se podmínky ve firmě i okolí (požadavky zákazníků)

2.1.2 Modely podnikového IS

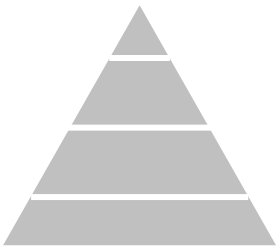
- Model z hlediska úrovní řízení –

Vrcholový management

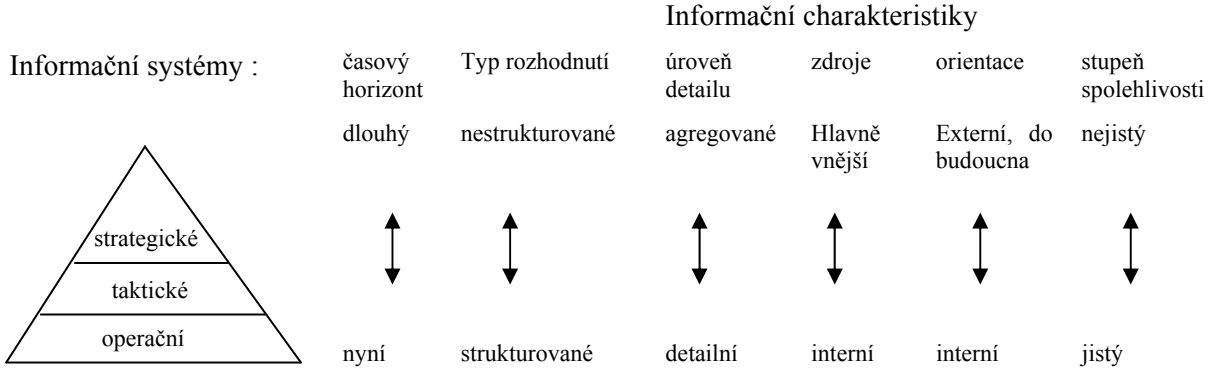
Střední management

Pracovníci pracují s daty a znalostmi

Pracovníci pořizují data a realizují výkonné činnosti



	Hlavní úkoly	informace	Nástroje IS
Vrcholový management	Vize a strategie podniku, informační strategie	Přehledné informace o okolí firmy	EIS, MIS,
Střední management	Zajištění a realizace zakázek	Plánování a řízení zakázek, stav zakázek	Integrované IS typu ERP
Pracovníci zpracovávají data a znalosti	Návrh výrobku, způsobu výroby, finanční analýzy	o materiálu, technologii, stavu zásob sledování nákladů	ERP, aplikace CAD, CAP
Výrobní a obslužní pracovníci	Realizace výrobku, sběr dat	Informace pro technologický a logistický proces	NC stroje,čárové kódy,terminály Zpracování faktů



- Procesní model

některé procesy jsou shodné u více podniků; např. zpracování nabídky, realizace zakázky, prodej produktu, finanční řízení, personalistika. Jen málo IS má zatím procesně orientovaný SW, kde je procesně řešen průběh zakázky

- Technologický model – jádrem je HW a na něm jsou vrstvy SW od OS až aplikačnímu SW; jsou to systémy vytvořené „na míru“

- Technické prostředky (hardware)

Počítače (Osobní počítače, servery, pracovní stanice, specializované řídicí počítače, apod.) a jejich bezprostřední komponenty (operační paměti, přídavné procesory, rozhraní, přídavné procesory) Periferní zařízení (tiskárny, souřadnicové zapisovače, digitizéry, snímače čárových kódů a čipových karet, speciální displeje, apod.) Síťové komunikační prostředky (kabeláž sítě, komunikační počítače, odbočovače v sítích, síťové karty do osobních počítačů, modemy, zesilovače signálu, apod.) Doplňková a podpůrná zařízení (zálohovací zdroje síťového napětí, filtry před obrazovky, specializovaný nábytek, klimatizační zařízení, měřící a testovací zařízení) Provozní materiál (papír do tiskáren, diskety, výměnné optické disky, náplně do tiskáren) Dokumentace technického vybavení

- Programové prostředky (software)

Operační systémy (základní operační systémy, síťové operační systémy, operační systémy řízení reálného času) a jejich přídavné části Systémové programy, které provádějí speciální řídicí a provozní funkce. Vývojové prostředky (programovací jazyky, knihovny programovacích jazyků, testovací prostředky, vývojová prostředí) Databázové systémy a jejich součásti Standardní aplikační programy (tabulkové procesory, textové editory, presentační programy, elektronická pošta, apod.) Speciální aplikační programy, zhotovené podle individuálních požadavků.

- Organizační prostředky (orgware) - nařízení a pravidla definující provozování a řízení IS, pokyny pro obsluhu a návody k obsluze, provozní pokyny IS, pravidla zajišťující dělbu a koordinaci prací kolem IS, stanovující zodpovědnost za správnost vkládaných dat, vymezující oprávněnost přístupu k datům a oprávněnost manipulace s daty. Pokyny pro archivaci dat a pořizování bezpečnostních kopií, k provádění antivirové ochrany a k zabezpečení dat před úmyslným zneužitím, zcizením nebo poškozením, pro vyhodnocování a sledování činnosti IS včetně evidence nákladů na IS, zásady pro údržbu a inovaci IS

- Lidská složka (peopleware) - adaptace a účinné fungování člověka v prostředí IS, znalosti, motivace, kompetence zaměstnanců.
- Reálný svět jako kontext IS - informační zdroje, legislativa, normy, …

Účelem automatizovaného informačního systému firmy je poskytovat informace pro podporu rozhodování a řízení firmy, aby mohla dobře prosperovat. Poskytované informace proto musí být: významné - přesné - včasné - dostatečné - úplné - srozumitelné. Aby IS poskytoval takové informace musí mít následující vlastnosti: - Poskytovat komplexně potřebné funkce, nutné pro zajištění všech činností, které se mají automatizovat. - Mít vhodné provozní parametry, aby jeho využívání bylo jednoduché, chod spolehlivý, požadované provozní náklady co nejnižší a rychlost poskytování informací přiměřená požadavkům. - Umožňovat relativně jednoduché a rychlé přizpůsobování svých funkcí a vlastností změněným požadavkům. - Musí mít dobře vyřešeno zabezpečení proti zneužití a proti poškození a to jak sebe sama, tak informací, které jsou v něm uloženy. - Vykazovat parametry vysoké jakosti a to jak v průběhu dodávky, tak návrhu a provozu. - Jeho vhodná cena musí být přijatelná pro uživatele. - Mělo by být pro něj charakteristické využívání dostupných progresivních informačních technologií, aby nebyl v krátké době zastaralý a překonaný. - Jeho koncepce i provedení by měly využívat racionální a objektivně zdůvodněné principy.

Obsahová náplň IS

- funkce, které je schopen poskytovat (oblasti - Prodej, Nákup, Sklady, Marketing, Finanční účetnictví, Controlling, Majetek, Lidské zdroje, Práce a mzdy, Technická příprava výroby, Plánování výroby, Operativní řízení a plánování výroby, Dílenské řízení výroby)

- hierarchicky uspořádaný souhrn všech operací s daty (založení informací, vytvoření nabídky, přehledy a analýzy)

- procesy, které podporuje (řídící, obchodní, výrobní, správní, plánovací) V co nejvyšší míře podporovat optimalizaci podnikových procesů
- data, která jsou předmětem zpracování (o zákaznících, výrobcích, dodavatelích, …)

Základní komponenty IS

- Vstupy: data, texty, zvukové a obrazové záznamy vstupující do IS a metody a prostředky, jimiž jsou vkládány a uchovávány
- Modely: kombinace procedurálních, logických a matematických modelů, které transformují vstupy na požadované výstupy
- Technologie: technici, hardware, software
- Databáze: soubor(y), v němž jsou uložena data potřebná pro všechny uživatele
- Správa: ochrana dat a dalších komponent systému
- Výstupy: kvalitní informace pro všechny úrovně řízení a ostatní uživatele uvnitř i vně organizace. Nemůže být lepší než umožňují vstupy a modely

2.2 Architektura IS

Původně je architektura chápána jako prostorové uspořádání hmotných prvků do celku tak, aby toto uspořádání splňovalo dané funkční, estetické i ekonomické požadavky.

Architektura informačního systému

- grafické a písemné vyjádření celkové koncepční představy IS, která v sobě bude zahrnovat základní představu o
  1. struktuře IS v návaznostech na organizační strukturu organizace
  2. funkcích, které bude IS zabezpečovat v návaznosti na procesy probíhající u organizace
  3. provozu a bezpečnosti celého IS v návaznosti a organizaci práce u organizace
  4. vazba IS na okolí

- celková architektura
  - jednotlivé bloky (moduly), každý blok je určen současně několika dimenzemi (rozsah požadovaných funkcí bloku, funkční rozsah)
  - datová - datové objekty ve správně bloku (udržuje, požívá)
  - organizační - organizační jednotky, které služby zajišťují či využívají
  - personální - personální nároky, které funkce bloku vyžadují
  - SW - zajištění bloku základním SW (současným standardním, navrhovaným), podklad pro plánování
  - technická - počítače, komunikační technika
  - metodická - orientace na projekční, orientační metody
  - ekonomická - předpokládané náklady, přínosy rozpočet

- vazby mezi bloky
  - hrany jsou pojmenované (určeny obsahem předávání dat, řídících nebo zdrojových)
  - formát dat
  - periodicita a časové rozložení aktivace vazby

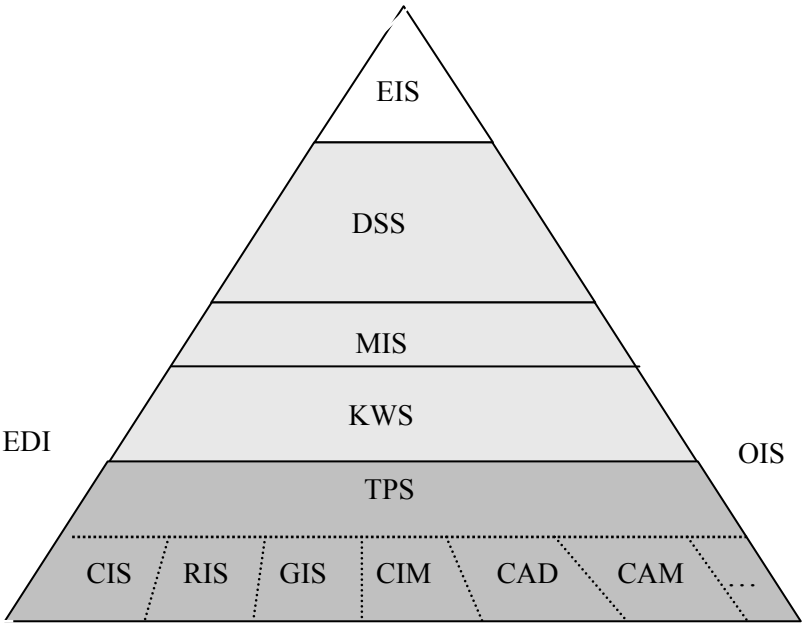
  Př.: MIS → EIS reprezentuje transport dat z MIS do EIS

  - Obsah vazby, struktura přenášených dat a jejich další charakteristiky
  - Periodicita vazby
  - Objem přenášených dat za jednotku času
  - Forma vazby (sdílení, replikace v dávkách, …)
  - Technologické zajištění (konvertory dat apod.)

- Organizační zajištění vazby (zodpovědnost za realizaci, kontrola správnosti, monitorování a analýza využití, zálohování, …)
- Řešení mimořádných stavů (výpadky, chyby, apod.)

vazba k okolí - externí subjekty

- Schéma, které bere v úvahu všechny podstatné dimenze návrhu IS
- Východisko pro dosažení potřebné úrovně konzistence, integrace a interoperability IS
- Vychází z celopodnikové strategie, z podnikových cílů, které se promítají do tří úrovní podnikových aktivit:
  - produkčních
  - řídících
  - řízení IS
- Principy tvorby architektury:
  - Perspektivní
  - Vyjadřuje celkovou vizi IS, je oproštěna od detailů
  - Je přiměřeně jednoduchá a srozumitelná, představuje kostru, do níž budou postupně zasazovány jednotlivé aplikace



EIS	Executive IS	Podpora vrcholového řízení, strategické řízení, marketing
DSS	Decision support system	podporují úroveň taktického řízení (znalostní báze), akce, optimalizační a simulační algoritmy, podpora metod rozhodovací a operační analýzy
MIS	Management IS	Podpora taktické úrovně řízení, sumarizace, agregace, generátory zpráv, grafická podpora. Cílem úloh je zajištění průběžné evidence procesů a zdrojů, zpracování dokumentů požadovaných legislativou i vnitřními předpisy, zpracování ekonomických aj. analýz
KWS	Knowledge work system	Expertní systémy
TPS	Transaction	Podpora operativní úrovně řízení spojené s typem

	Processing System	provozu dané organizace (rezervace lístků, bankovní systémy) na nejnižší úrovni
OIS	Office IS	Podpora rutinních kancelářských prací (zpracování a správa dokumentů, e-mail,..)
EDI	Electronic Data Interchange	Podpora elektronické výměny dat mezi obchodními partnery, bankami apod.
CIS	Customer IS	Bezprostřední styk se zákazníkem (odečty spotřeby, fakturace, …)
RIS	Reservation IS	doprava, cestovní ruch
CAM	Computer IS	konstrukční a návrhářské balíky, průmysl, návrh výrobků, jeho konstrukční řešení
CAD	Computer Aided Design	Konstrukční a návrhářské práce
GIS	Geography IS	Kreslení, digitalizace map, vytváření územních modelů
CIM	Computer Integrated Manufacure	Integrace výrobních procesů

AI – Artificial Intelligence

užití metod umělé inteligence, schopnost učit se z realizovaných akcí

ESS - Expert Suport Systems – expertní systémy

používají bázi znalostí vytvořenou experty a inferenční rámec formalizace implicitních znalostí. Program, který se chová jako expert v úzké problémové oblasti řeší problémy efektivně a účinně, používají se pro problémy, které lze symbolicky reprezentovat. Umožňuje uživateli přístup k expertním znalostem podobným způsobem, jako by konzultoval s expertem, s podobným výsledkem, nabídne inteligentní radu nebo provede inteligentní rozhodnutí, dokáže na vyžádání vysvětlit svůj způsob „uvažování“ způsobem, který je uživateli srozumitelný.

strukturální komponenty

- Techniky umělé inteligence, zvláště orientované na řešení problémů
- Znalostní komponenta: báze znalostí, které jsou sdílené, dostupné a na kterých se shodují experti v dané oblasti; formalizované reprezentace expertních znalostí
- Inferenční pravidla: hledání řešení za neurčitých podmínek, využití heuristik
- Modelování lidského experta: reprezentace problémové oblasti způsobem, jakým myslí experti, formalizace a využití informací získaných od expertů

Heuristiky jsou většinou individualizovaná pravidla dobrého usuzování, která jsou charakteristická pro expertní rozhodování v dané oblasti, vedou k nalezení řešení problému bez znalosti algoritmu

Znalostní inženýrství

- Získávání znalostí (experti, knihy, manuály, …) včetně vývoje znalostí potřebných pro řešení problémů
- Reprezentace znalostí
  - Formalizace znalostí (tacit - implicitní - explicitní)
  - Metody reprezentace znalostí (pravidla, rámce, případy)
- Vyvozování (a zdůvodňování) závěrů

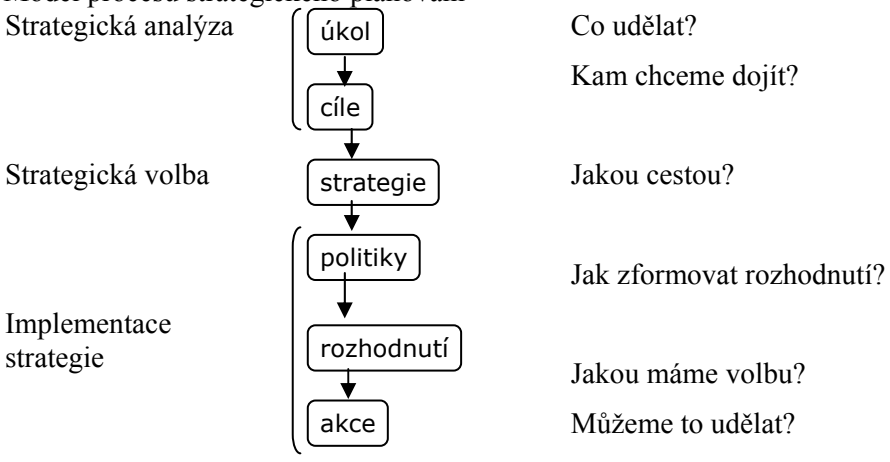
Výhody:	Nevýhody:
1. Permanence: systémy (na rozdíl od lidí) nezapomínají 2. Reprodukovatelnost 3. Efektivita - zvýšení průchodnosti, snížení osobních nákladů	1. Zdravý rozum (common sense) - mají lidé, stroje zatím ne 2. Kreativita - lidé umějí v neobvyklých situacích reagovat kreativně 3. Učení se - lidé se automaticky

4. Konsistence - podobné situace jsou zpracovány stejně; lidé jsou ovlivněny časovým faktorem (na rozhodnutí mají větší vliv nedávné nebo první informace) 5. Dokumentovanost 6. Úplnost (ES může na rozdíl od člověka zhodnotit všechny transakce) 7. Šíře záběru - lze kombinovat znalosti více expertů 8. Včasnost - informace pro rozhodování je dostupná rychleji 9. ES může pomoci vybudovat bariéry vstupu pro konkurenci nebo diferencovat produkt	přizpůsobují měnícímu se prostředí, ES musejí být upgradovány. Neuronové sítě a systémy založené na případech mohou zahrnovat učení se. 4. Smyslová zkušenost - ES jsou (zatím) závislé na symbolických vstupech 5. Degradace - ES obtížně rozeznávají případy, které nemají řešení nebo problém je mimo oblast jejich expertízy

S vyšší úrovní řízení roste neurčitost v požadavcích na informace, důraz se přesouvá z kvantity na kvalitu informací (agregace, selekce). Roste potřeba externích informací.

Typické funkce EIS : Komplexní analýzy, podklady pro rozhodování, strategické řízení (organizace nové výrobky, technologie, trhu, formuluje strategie) , strategické plánování (analytický proces, zaměřený na obchod, ekonomiku, technologii)

- generace strukturovaných výstupů s vysokou vypovídací hodnotou,
- rychlé a jednoduché vytváření nových pohledů na data, jejich agregace a řazení do nových souvislostí,
- analýzy trendů a odchylek, analýzy scénářů, práce s historickými daty, a předvídání budoucího vývoje, přístup k interním i externím datům (jednoduché ovládání, přehledná prezentace).
- Model procesu strategického plánování



Typické funkce MIS: Účetnictví, prodej a nákup, personalistika, PaM

Typické funkce OIS :

- Redukce nároků na administrativní operace (zpracování textů, kalkulace, …) a zvýšení pořádku v administrativě organizace (správa dokumentů, workflow)
- Zrychlení komunikace uvnitř i vně organizace (e-mail)
- Využití Internetu, intranetu
- Zvýšení formální úrovně kancelářských prací (formátování dopisů, zpráv, katalogů, …), jednotný styl organizace (šablony, …)
- Podpora skupinové práce (groupware)

Systém zpracování transakcí (transakční systémy) - je bezprostředně spojený s určitým typem procesu v rámci organizace.

Obsahuje subsystémy podporující:

- dílenské, skladovací a transportní operace výrobních podniků,
- konstrukční a technologické procesy,
- rezervační systémy dopravy, turistiky, hotelů,
- provoz na přepážkách bank, pošty apod.,
- poskytování zákaznických služeb distribučních společností

CIM, CAD, CAM : Slouží pro optimalizace řízení výrobních provozů, podpora řízení jakosti, automatizace výrobních linek, zajišťují vysoké nároky na bezpečnost a spolehlivost provozu, zabezpečení proti výpadkům, haváriím apod.

Význam architektury IS

- Vytváří relativně stabilní rámec, do něhož se v průběhu vývoje začleňují jednotlivé aplikace a prostředky
- Je významným komunikačním prostředkem mezi vedením organizace a projektanty IS. Zajišťuje souhlas a vzájemné porozumění různých složek organizace v tom, které aplikace, data a rozhraní budou v daném čase implementovány. Takto řízený růst minimalizuje duplicity, zaručuje kompatibilitu, vzájemnou propojitelnost a integraci systému
- Dostatečně otevřená architektura zajišťuje stabilitu vývoje IS i při rychlém rozvoji IT
- Umožňuje od počátku řešení uvážit hlavní požadavky na IS/IT a z nich odvíjet specifikace týkající se např. obsahu a rozsahu řešení, charakteru řízení, flexibility a spolehlivosti IS
- Umožňuje minimalizovat náklady vyplývající z chybného zadání
- Reaguje na trendy směřující k vytváření IS z hotových produktů a na jejich stále vyšší heterogenitu

Vrstvy architektury IS/IT

- Vrstva prostředí: ekonomické, legislativa, normy, organizační struktury, lidské zdroje, jejich kvalifikace, zkušenosti s IT, motivace. Obecněji kulturní prostudí, sociální struktura společnosti, politická stabilita.
- Vrstva aplikační: provozované i řešené projekty, jejich dokumentace, funkční a datová specifikace, organizační pravidla jejich řešení a provozu, aplikační software
- Vrstva technologická: návrh a provoz sítí, specifikace komponent IS/IT - základního SW, HW, vazby a vnitřní struktura

Dílčí architektury IS

- Funkční: hierarchický rozklad funkcí IS
- Procesní: návrh budoucích stavů procesů v organizaci
- Datová: návrh datové základny IS, který vychází z analýzy potřebných datových objektů a jejich vazeb
- Softwarová: určuje, z jakých SW komponent bude IS postaven a jaké mezi nimi budou vazby. SW systém je definován množinou modulů a vazbami mezi nimi
- Hardwarová: určuje typy, počty a vzájemné vazby HW komponent
- Technologická: rozhoduje o technologickém řešení aplikace

2.2.1 Základní 3-vrstvá softwarová architektura

Existuje mnoho variant architektury IS z pohledu architektury softwaru. Základem je 3-vrstvá architektura s vrstvami

1. prezentační
   - komunikace s uživatelem
   - HW, SW (koncové programy)
   - komunikace s další - vnitřní vrstvou IS
2. funkční
   - knihovny aplikačních podprogramů a aplikační servery
   - z nejnižší vrstvy získá informace a zpracuje je
   - vzájemná konektivita
   - identifikace, autentizace – zajištění bezpečnosti pro soubory, adresáře
   - síťová propojení, externí zdroje dat (distribuované aplikace)
   - synchronizace, aktualizace dat (transakční monitory)
   - řízení a kontrola verzí SW
3. datová
   - všechny systémy pracující s daty na nejnižší úrovni (zdroje informací - databáze, soubory (textové, tabulkové procesory, …), poštovní servery, datové sklady)
   - přístup funkční vrstvy (distribuce)

2.3 Integrace informačních systémů

Integrace systémů probíhá na třech úrovních : obchodní (mezi organizacemi), aplikační (sémantická), technologické(syntaktická).

Stupně integrace:

- Částečná podpora jednotlivých procesů
- Podpora procesů v jednotlivých útvarech
- Horizontální integrace - podpora procesů přesahujících hranice útvarů stejné úrovně řízení
- vertikální integrace - podpora procesů přesahujících hranice útvarů různé úrovně řízení
- Komplexní podpora podnikových procesů – CIB (Computer Integrated Business)
- Podpora procesů mezi podnikem a jeho externími partnery (externí integrace)

Úroveň integrace	Primární cíl IS	Sekundární cíl IS, důsledky zavedení	Příklady
Nízká, nezávislé procesy	Efektivnost rutinních prací	Informace pro řízení rutinních prací	Oddělené aplikace: fakturace, hlavní kniha,

			sklad, PAM,…
Částečná	Podpora opakujících se rozhodovacích procesů	Lepší pochopení podstaty procesů	MIS – IS dává strategické informace, ale není součástí strategie
Plná	Nabídka nových produktů, získání nových trhů	Pružná změna procesů a kritérií, více alternativ	IS je součástí strategie, produktu, služby (přímé propojení s partnery)

Předpokládá se:

- zkrácení doby odezvy na vstupy z okolí
- využití nových programovacích řídících metod (třeba vyšší dostupnosti a komplexnosti)
- efektivní působení na trhu díky dobrým a včasným informacím
- snížení chybovosti a nekonzistence informací
- snížení nákladů na duplicitní či multiplicitní informace
- rozvoj kvalifikace pracovníků (ukazuje se spíše negativně)

Nevýhody - zvýší se složitost systému (nároky na projektantskou činnost, úroveň informačních technologií, na uživatele (ztrácí komplexní náhled), pochopení všech relevantních vazeb

- větší a rychlejší dopad havárií (nároky na odolnost proti virům apod.)

Kategorie integrace

- vnitřní - jednotlivých elementů uvnitř podniků
- vnější - vnější vazeb (zákazníci, dodavatelé)
- integrace podle úrovní: vertikální (z úrovně do úrovně) nebo horizontální (v rámci úrovní)
- technologická - například SW, HW a kompatibilita, integrace na úrovni informačních a komunikačních vazeb : časové hledisko integrace - tj. synchronní či v reálném čase,
- systémová integrace - metoda vývoje a užití IS, jejímž cílem je komplexní IS organizace

Proces integrace aplikací

- integrace s navazujícími aplikacemi podniku kupujícího, které jsou typicky součástí ERP
- integrace s příslušnými IS dodavatele - vnější integrace (tzv. business networking).
- technologická integrace

2.4 Aplikační software

Aplikační software IS je komplex programů pro podporu specifických fukcí a procesů v podniku. Důležitým úkolem je metodická organizace projektu IS podniku. Mezi důležité úlohy pro odpovědné řešitele je volba softwaru pro realizaci a implementaci IS. Pro ilustraci si uvedeme v krátkosti, které atributy se berou v úvahu při porovnání dodávaného SW:

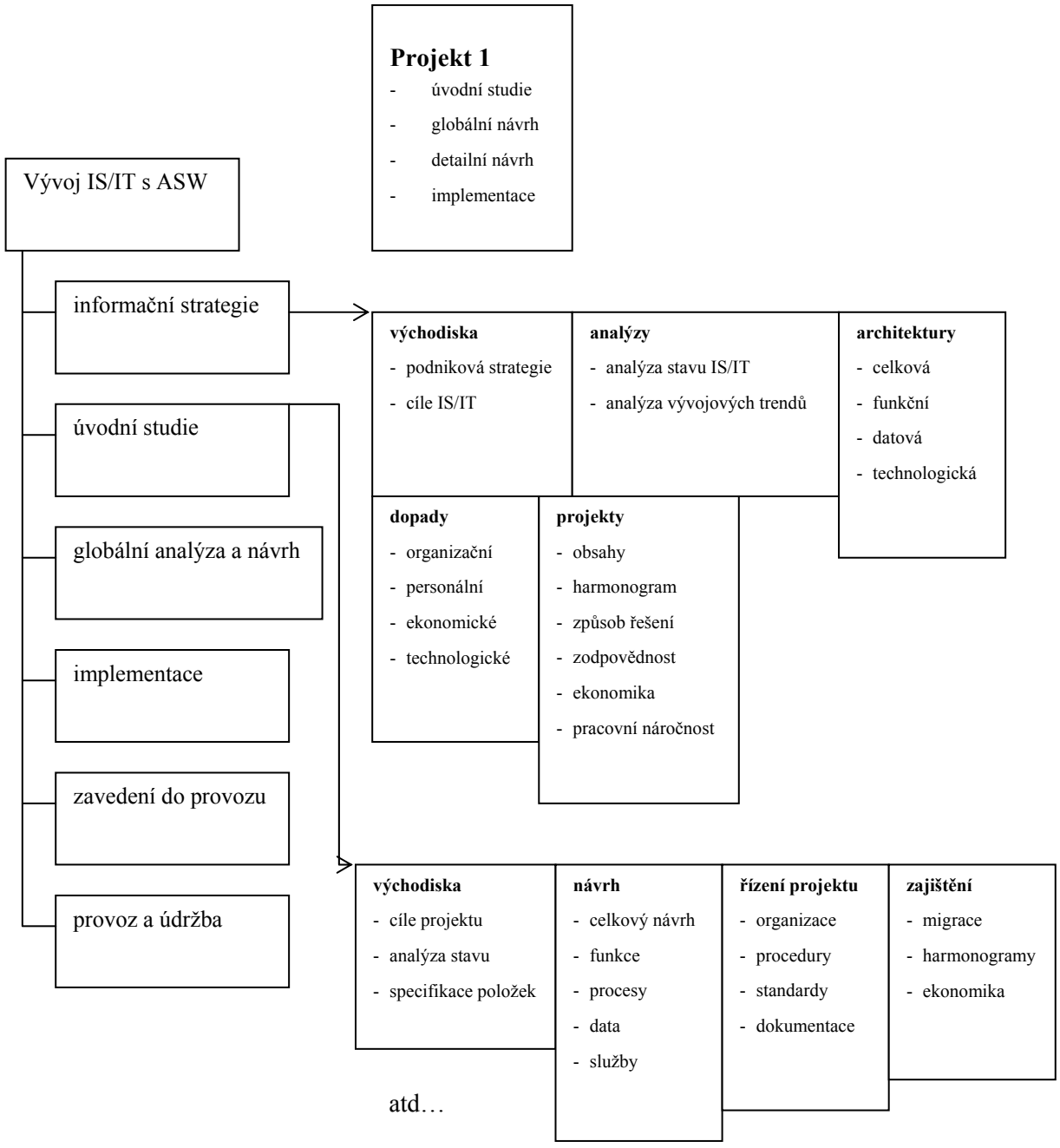
- dokumentace (od jednotlivých firem je různá kvalita – komplexnost, přehlednost, úplnost, grafická a jazyková úroveň)
- použitelné platformy (může běžet na různých systémech?)

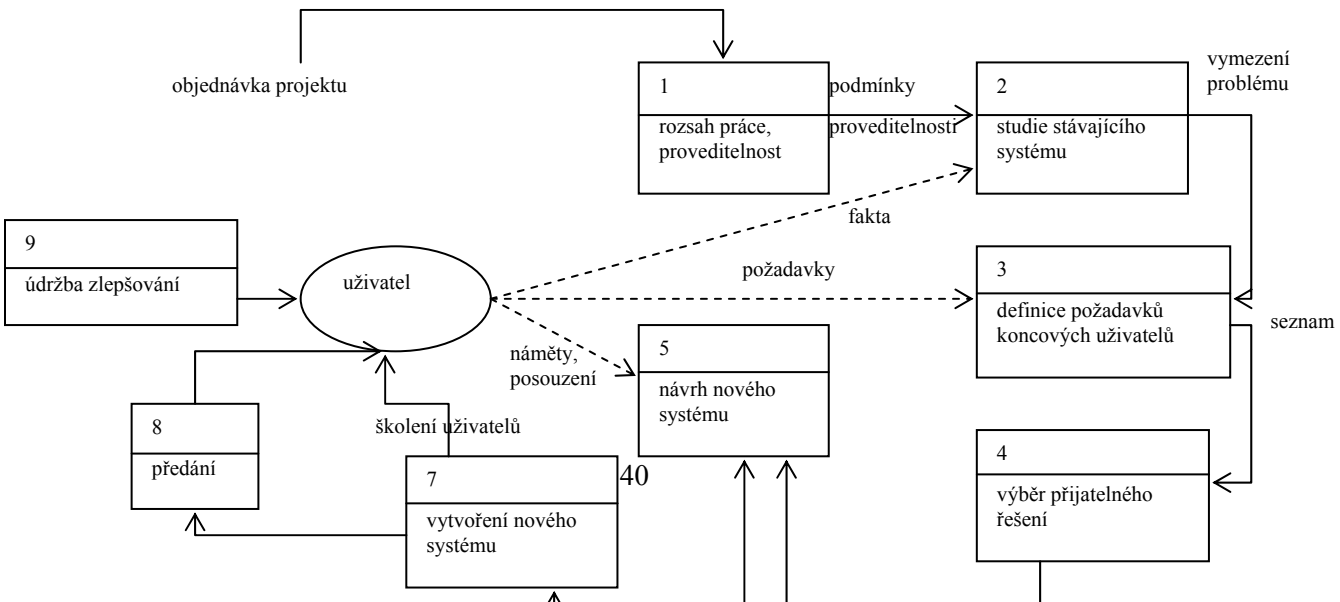
- terminologie (standardy a normy)
- cenová politika
- poskytované know-how (firmy nerady poskytují informace)

Pro porovnání můžeme porovnávat software podle10 základních skupin charakteristik

1. základní údaje
   - informace o tvůrci, distribuci
   - orientace vývojářské firmy
   - personální síla firmy, zastoupení, komunikace
   - distributor, kontakty, pracovníci, opravy pro instalaci
   - orientace SW - ekonomika (úroveň obecnosti, použití v různých sektorech, parametrizace)
   - cenové charakteristiky, celková cenová strategie firmy
   - úroveň lokalizace, přizpůsobení podmínkám národních parametrů
2. architektura, skladba modulu
   - jaké oblasti IS podporuje, do jakých logických modulů bude rozdělen, vymezení základních modulů
3. přehled instancí instalace
   - instalace v ČR, ve světě apod.
4. provozní prostředí
   - základní architektura - centralizovaná (terminály)
   - DB prostředí (vymezení, co už máme)
   - operační, síťový systém
5. vývojové prostředí
   - řízení a plánování projektu (systémy CASE)
   - uživatelské rozhraní
   - podpora hypertextové komunikace
   - unifikace prostředí (standardizace)
6. dokumentace, jazykové prostředí
   - funkce a nasazení v dalších světových jazycích, jazykové mutace, úplnost překladu
7. doplňující služby
   - údržba, upgrade (podle legislativních změn), periodicita verzí, cenová náročnost
   - služba horké linky (okamžitá pomoc při problémech u zákazníka)
   - projekční služby firmy (využití firemních specialistů z oboru – rychlost, komplexnost, kvalita)
   - školicí služby (proškolení SW, konzultace)
8. standardy, normy, specifikace, certifikace
   - je možné podrobit SW auditu
   - otestování nezávislou organizací
   - možnosti integrace (přenositelnost)
9. flexibilita
   - míra přizpůsobení uživatelským požadavkům (ale vzrůstají nároky)
   - úpravy podle potřeb zákazníka (zrušení nějakých voleb, nastavení pevných hodnot, nové funkce)
10. funkční struktura, specifikační funkce

Příklad klasické metodiky vývoje IS/IT s ASW ukazují následující schémata.


Projekt 1
- úvodní studie
- globální návrh
- detailní návrh
- implementace
Vývoj IS/IT s ASW
informační strategie
úvodní studie
globální analýza a návrh
implementace
zavedení do provozu
provoz a údržba
východiska
- podniková strategie
- cíle IS/IT
analýzy
- analýza stavu IS/IT
- analýza vývojových trendů
architektury
- celková
- funkční
- datová
- technologická
dopady
- organizační
- personální
- ekonomické
- technologické
projekty
- obsahy
- harmonogram
- způsob řešení
- zodpovědnost
- ekonomika
- pracovní náročnost
východiska
- cíle projektu
- analýza stavu
- specifikace položek
návrh
- celkový návrh
- funkce
- procesy
- data
- služby
řízení projektu
- organizace
- procedury
- standardy
- dokumentace
zajištění
- migrace
- harmonogramy
- ekonomika
atd…


objednávka projektu
1
rozsah práce, proveditelnost
podmínky proveditelnosti
2
studie stávajícího systému
vymezení problému
fakta
9
údržba zlepšování
uživatel
požadavky
3
definice požadavků koncových uživatelů
seznam
náměty, posouzení
5
návrh nového systému
8
předání
školení uživatelů
7
vytvoření nového systému
4
výběr přijatelného řešení

Obrázek 8 Metodiky vývoje IS/IT s ASW

Shrnutí

Mezi nejdůležitější typy IS patří ekonomické podnikové systémy. Historický vývoj ukazuje stále důležitější úlohu integrovaného, komplexního IS podniku pro efektivitu a konkurenceschopnost. Podle úrovně organizační, charakteristických vlastností, prostředků a cílů rozdělujeme podnikové systémy na operační, znalostní, taktické a strategické. Pro metodickou organizaci, analýzu a vývoje IS se vytváří různé modely (procesní, technologický), které popisují různé architektury, od datové, softwarové, funkční (procesní), přes softwarovou, technologickou (HW) až po komplexní systémovou. V procesu vývoje podnikových IS hraje důležitou roli integrace a systematické použití vhodných metodik, metod, technik a nástrojů pro zvýšení efektivity, komplexnosti, flexibility, přehlednosti a dokumentovatelnosti. Mezi důležité úlohy patří posouzení kvality použitelného aplikačního softwaru.

Pojmy k zapamatování

- strategické, znalostní, taktické a operační informační systémy,
- architektura IS, procesní, technologický a organizační model
- podnikové subsystémy - EIS, DSS, MIS, KWS, TPS, OIS, EDI, CAM, CAD, GIS, …
- metodika, metoda, technika, nástroj, aplikační software

Kontrolní otázky

8. *Jaké jsou charakteristiky jednotlivých úrovní podnikovýchIS?*
9. *Jaké jsou architektury podnikového IS?*
10. *Jak charakterizovat znalostní inženýrství?*
11. *Jaké znáte typické podnikové subsystémy?*
12. *Jaké jsou stupně a kategorie integrace?*
13. *Jak definovat pojmy metodika, metoda, technika, nástroj?*
14. *Podle jakých kritérií můžeme porovnávat aplikační software?*

Úkoly k textu

3. Pokuste se posat jednotlivé modely a architektury jednoduchého rodinného podniku (firmy) ve svém okolí – používaný HW, SW, programové subsystémy, … a analyzujte vlastnosti IS jako komplexnost, integraci, flexibilitu, efektivnost, spolehlivost, …

3 Tvorba a analýza IS

Studijní cíle: Student by měl po prostudování kapitoly porozumět obsahu, charakteristice a základním postupům z oblasti tvorby a analýzy IS. Kapitola poskytuje částečně i historický úvod do problematiky, která je systematičtěji zpracována v předmětech, které se věnují softwarovému inženýrství. Student by měl být schopný využít metody a metodiky pro pozdější vlastní návrh datové a funkční analýzy i implementaci jednoduchého IS.

Klíčová slova: Životní cyklus řídícího systému (ŘS) a projektu, analýza a návrh ŘS, studie, implementace, hierarchický rozklad, DFD, rozhodovací tabulka a strom, stavový diagram, Jacksonovy strukturogramy, ER diagram, datový slovník, prototypování, objektově orientovaný přístup, model jednání, spolupráce, objektový, funkční, dynamický, neformální a formální specifikace, uživatelský vzhled aplikace, strukturovaná analýza

Potřebný čas: 4 hodiny

3.1 Tvorba systému

Vývoj (hlavně analýza a návrh) informačních podnikových systémů je komplexní stimulační, inovační a organizační proces, prováděný specialisty převážně z oblasti systémového řízení a ekonomických aktivit (výroba, obchod, …) podniku. Základním cílem je zvýšení efektivity klíčových podnikových procesů pro pracovníky, kteří využívají počítačovou podporu softwarových řešení. Z historie návrhů a používání systémů ERP se dá vypozorovat, že rozvoj IS může probíhat třemi způsoby:

často se místo úpravy starého systému zbytečně vytváří nový, to má spíše marketingové důvody

1. rozvoj a inovace existujících řešení, stávajícího IS – přebudování a znovupoužití
2. vývoj nového, vlastního IS vlastními prostředky – návrh a realizace
3. nákup hotového IS

Pro úspěšné zvládnutí analýzy a návrhu IS musí být zvládnuty tři základní složky: - data, datové toky a procesní logika. Přístup může být orientován procesně – zaměřen na tok, transformaci a použití dat, nebo datově – zaměřen na optimalizaci stabilnější datové struktury ne na zpracování a užití dat. Při tvorbě systému se používá široké spektrum technik pro různé fáze životního cyklu vývoje – od čistě formálních (např. Petriho sítě), jejichž cílem je jednoznačnost a úplnost, přes poloformální techniky (např. jazyk UML - standardní jazyk pro zobrazení, specifikaci, konstrukci a dokumentaci prvků systémů nejčastěji se softwarovou charakteristikou), až po neformální techniky. Nejdůležitějším podpůrným prostředkem je v současnosti právě UML. Názorné grafické vyjádření na přiměřené úrovni složitosti napomáhá při pochopení a řešení problémů i při jejich dokumentaci. Jazyk pro popis konceptů na vysoké úrovni abstrakce použitý strukturovaným způsobem podporuje zvládnutí složitosti softwarových systémů obecně a IS pro jejich komplexnost zvláště. Používané modely prochází typicky podobným procesem, obsahujícím specifikace (definuje funkce a omezení systému), návrh a implementace (vytvoření systému, splňující specifikace), validace softwaru (testování, aby se dokázalo, že jsou splněny požadavky zadavatele), evoluce softwaru (vyvoj pokračuje, aby mohl uspokojit potřeby zákazníka při inovacích a změnách).

Při projektování podnikového řídicího systému formulujeme úlohy ŘS a vybranou metodikou realizujeme jednotlivé fáze životní cyklus řídicího systému nebo projektu, které se skládají z identifikace potřeby i priority, výběru, inicializace a plánování, analýzy (určení požadavků, vytvoření, porovnání a priorita alternativ), návrhu (logický se zaměřuje na aplikační logiku – obchodní, …, fyzický se týká technické specifikace), implementace a údržby. Životní cyklus zahrnuje i další činnosti, například plánování a řízení projektu, organizace a časový rozvrh,

kontrola kvality. Postup může například probíhat podle jednotlivých kroků scénáře, kdy se postupně vytvoří:

- Plánování (strategie řízení firmy) – formulace problému, cíle, rozsahu, termínů, rozpočtu, dosažitelnosti, časového rozvrhu, určí se řídicí, organizační a řešící pracovníci, …
- Úvodní studie zmapuje aktuální stav, požadované změny, postupy, zdroje, náklady, termíny, technická omezení, …
- Globální analýza a návrh, definice systémových požadavků a priorit, vytvoření prototypů, vyhodnocení alternativ, zohlednění připomínek vrcholového vedení podniku, návrh aplikační architektury, návrh databáze
- Detailní analýza a návrh (uživatelské rozhraní, …)
- Implementace, konstrukce softwarových komponent, verifikace a testování, konverze dat, ověřování s uživateli, dokumentace
- Instalace
- Provoz a údržba, rozšiřování systému, …

Proces typicky není přímo sekvenční, ale iterační, s návratem na předchozí kroky, kde se znovu řeší nově vzniklé problémy z aktuální úrovně. Příklady a alternativami klasického životního cyklu programu je prototypování (vhodný k pochopení a formování požadavků na systémy, které nejsou úplně a přesně specifikovány předem), spirálový model (kombinuje prototypování se systematickým sekvenčním přístupem a iterací na výším stupni zvládnutí problematiky s prioritou tvorby verze s vyšší rizikovostí), model vodopád (jednotlivé aktivity jsou zpracovány jako nezávislé procesy, které na sebe navazují), evoluční model (prokládá jednotlivé etapy realizace kontrolou se zákazníkem a jejich paralelním zpracováním tak, aby se postupné verze realizovaného systému co nejrychleji blížily požadavkům zákazníka), formální návrh (vytvoří formální matematický modelu specifikace systému, převáděný do programové podoby s verifikací odvozenou z matematického dokazování specifikací). Podstatný vliv na úspěch při řízení projektu má organizace řešitelů do efektivních týmů, s motivovací, koordinací a efektivní komunikací členů. Proces tvorby IS musí být přizpůsoben lidem a problému, vybrán vhodný model.

V současnosti se nejvíce prosazuje objektově orientovaná analýza a návrh, soustředěná na objekty jako abstrakce reálného světa se zapouzdřením vlastností i metod více, něž na samotná data a procesy. Detailnější průběh objektového návrhu systému obsahuje ve fázi analýzy model reality se zaměřením na důležité vlastnosti a chování, nezávislé na implementaci. Model je zpřesňován a adaptován na kontext prostředí. Fáze návrhu je rozdělěna na dvě fáze: - na systémový návrh (soustředěn na celkovou architekturu systému) a objektový návrh (k systémovému návrhu se přidají implementační detaily). Je charakteristické, že se podporuje schopnost zpracovat a pokrýt více problémových domén, zlepšuje se komunikace mezi uživateli, analytiky, návrháři a programátory a konzistence dílčích aktivit v průběhu tvorby, znovupoužitelnost řešení.

Průvodce studiem

Modelování – činnost, kdy se během jeho tvorby IS soustředíme jen na nejnutnější detaily a ostatní zanedbáváme. V průběhu tvorby je nutno vytvořit řadu různých modelů, které odpovídají jednotlivým pohledům na řešený projekt v příslušné fázi životního cyklu.Ty jsou použity k hledání akceptovatelného řešení. Při zpracování modelovaných znalostí je nutné postupovat logicky, racionálně, aby získané modely byly použitelné, přínosné, umožňovaly změny a pomohly při nových řešeních.

3.1.1 Metodika, metoda, technika, nástroj

Vysvětleme si některé pojmy z oblasti řízení projektu a vývoje IS:

Metodika (metodologie): - kolekce metod, které jsou vybrány na základě společné filosofie a společně podporují větší část (obvykle několik etap) životního cyklu programu, projektu. Doporučený souhrn etap, přístupů, zásad, postupů, pravidel, dokumentů, řízení, metod, technik a nástrojů, který pokrývá celý životní cyklus. Určuje kdo, kdy, co a proč se má dělat během etapy životního cyklu řídicího systému.

Metoda: - procedura nebo technika, pomocí níž se provede nějaká významná část životního cyklu programu, projektu, určuje, co je třeba dělat v určité fázi.

Technika: určuje, jak se dostat k určitému výsledku. Určuje přesný postup v jednotlivých činnostech. Je přesnější v závěrech a omezenější v okruhu použití.

Nástroj: je prostředkem k uskutečnění určité činnosti, pomáhá vytvářet model nebo jiné části projektu. Nástroje většinou bývají automatizovány (SW balíky, integrované vývojové prostředí, podpora nástrojů CASE), jsou to textové, grafické editory, formuláře apod.

Centrální kartotéka projektu (data repository) – strukturovaná paměť a zajišťuje vzájemnou vazbu mezi všemi částmi projektu.

Model: reprezentuje důležitý rys reálného světa ve formě diagramů a náčrtků.

Příklad: metodika návrhu řídicího systému (ŘS)

- Orientace na cíle a problémy
- Účast zadavatele i uživatele projektu na návrhu
- Klíčové dokumenty a jejich schvalování
- Modelování a abstrakce, tvorba architektury
- Ověřování a testování návrhu během celého vývoje
- Analýza a návrh v každé etapě
- Vývoj probíhá z hlediska všech úhlů pohledu na systém – je třeba analyzovat:

- data (jaká jsou třeba a jejich struktura),
- funkce (jaké funkce/procesy musí systém provádět),
- organizaci (jakou novou organizaci bude třeba zavést, nové odpovědnosti),
- technologii (typ a parametry HW a sítí),
- ekonomická hlediska (vztah nákladů a přínosů systému, provozní náklady).

3.1.2 Přehled vybraných metodik analýzy a návrhu IS

Možné rozdělení:

- *neškolený přístup*, implementace pomocí intuice a praktických zkušeností. Úspěch jen pro jednoduché úkoly, do určité úrovně složitosti a komplexnosti.
- Tradiční strukturovaný přístup, *hierarchický rozklad*

Klasické strukturované metody dekomponují projekt (strukturovaná analýza, návrh a programování) na menší, přesně definované struktury a určují jejich posloupnost a interakci. Mohou pomoci i při vytváření strukturované, úpnější a přesnější specifikace systému, užitečné pro uživatele, zadavatele i analytika a návrháře.

Dekompozice systému :

- funkční přístup – rozkládá systém na komponenty, odvozené z hierarchie subsystémů, modeluje systém jako množinu interagujících funkcí. Funkční transformace jsou realizovány v procesech, propojených datovými a řídicími toky. Příklad metod pro funkční rozklady:

  DFD (diagramy datových toků)

  JSP (Jacksonovo strukturované programování) nástroj k zachycení struktury dat a programů z této struktury odvozených, uzly stromu - představují funkce (procesy nebo akce), které musí být vykonány, kořen stromu - popisovaná funkce (proces, akce, jednotka). Stejný diagram je použit i pro popis dat (kořenem je popisovaná struktura, uzly na nižší úrovni představují složky), hrany - reprezentují komunikaci

*První zbraň proti složitosti – **hierarchické rozklady**, program se dekomponuje na části, které se vyvíjí odděleně a postupně skládají ve výsledný celek*

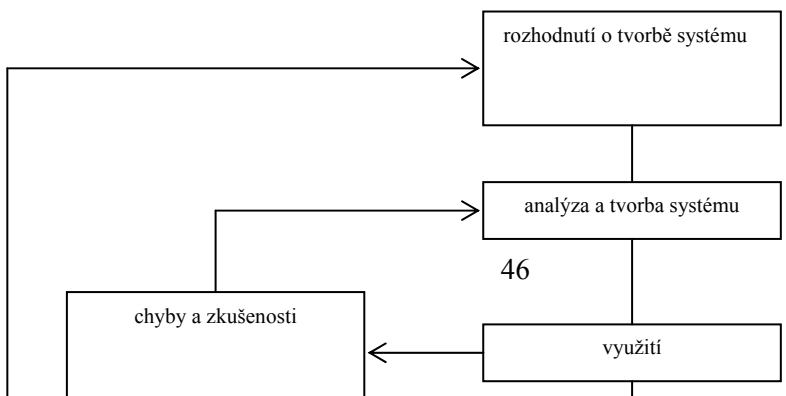
- objektově orientovaný přístup – modeluje systém jak množinu interagujících objektů, operace jsou zapouzdřeny v objektech
- datově orientovaný přístup – snaží se modelovat fundamentální datové struktury, které jsou vlastní dané aplikaci, funkční hledisko je potlačeno (podklad pro konceptuální model pro databázové systémy). Používá datový model (teorie rozkladu dat na entity a jejich vztahy - Chenův ER model) paralelně (iteračně) s funkčním hierarchickým rozkladem

*Druhá zbraň proti složitosti – **porovnávání modelů***

Strukturální metodiky poskytly možnost porovnání různých typů modelů, kombinují se rozklady podle funkčnosti a struktury dat, porovnává se úplnost a účelnost jednotlivých přístupů ve vzájemné konfrontaci. Používá se etapizace, kdy je proces návrhu rozdělen do několika etap (formulace zadání – na něm záleží smysluplnost systému, analýza – tvorba logického řešení, implementace – tvorba systému, testování, předání do provozu, provoz). Další typické charakteristiky procesu představuje návrat v postupu (je nutné do metod zařadit možnost vrátit se kdykoliv v postupu zpět do předchozích fází s aktuálně vzniklými požadavky) a prototypování (vytvoří se výchozí řešení - demo systému, na kterém se dohodnou požadavky, poté se postupně blíží ke správnému řešení).

Iteračníní cyklus projektu je znázorněn schematicky na obrázku:

*Třetí zbraň proti složitosti je **iterace***



Obrázek 9 Iteračníní cyklus projektu

- Metodiky pro *individuální*, často atypický, „originální“ přístup (spíše zdůvodnění, proč žádný systematický postup nebyl použit)

3.1.3 Objektově orientovaný přístup

Nahlíží na IS jako na kolekci komunikujících objektů, spolupracujících na splnění cílů. Nepracuje se přímo s pojmy jako proces, program, datová entita, soubor. Zpočátku se při analýze a návrhu vyšlo z vlastností objektů řízených událostí. OO analýza definuje všechny typy objektů a jejich interakce v informačním systému. OO návrh přidává typy objektů pro komunikaci s uživateli a zařízeními v systému. Hlavní přínosy objektově orientovaných metodik:

- členění do logicky definovaných etap
- výběr modelů pro porovnávání a z toho plynoucí přizpůsobení různým druhům úloh
- jednotnost návrhu a využití synergie objektů
- grafické jazyky (UML)
- vychází z praktických potřeb
- nabízí prostředky k sebekontrole
- lokalizace změn
- znovupoužitelnost

Charakteristika: návrh systému je komplexní problém, řešení má mnoho dimenzí (projekce, personální, organizační, finanční, kvalifikační, časovou atd.)

Objektová metodologie používá jednotlivé modely, přičemž model je chápán jako pohled na systém, který

- popisuje celý zpracovávaný systém (nebo jeho část)
- nebo zachycuje jen některé aspekty systému
- zobrazuje určitou abstraktní úroveň zobrazených aspektů

Používané modely se mohou rozdělit podle zaměření na

a/ Statickou strukturu systému (objektové diagramy, diagramy tříd a komponent a rozmístění)

b/ Dynamické chování systému (model jednání, diagramy aktivit a sekvencí, model spolupráce, funkční model)

c/ Dynamické chování jedné třídy (stavové diagramy, diagramy aktivit)

3.1.4 Model jednání (Use Case)

- cílem je popsat komunikaci vytvářeného systému jako černé skříňky a strukturalizovat okolí systému, které se systémem komunikuje, upřesnit zadání, definovat hranici, vymezit problém
- východisko dynamického modelu, jednotka funkcionality, která se projevuje posloupností zpráv, které si předávají aktoři (vnější entity) a systém,
- východisku pro tvorbu objektového modelu

Základní prvky : aktor - část okolí systému, která komunikuje s vytvářeným systémem (uživatel), primární aktor – užívá systém denně

typ jednání - kompaktní (logicky uzavřený) popis komunikace mezi aktorem a vytvářeným systémem. Popisuje se jako slovní scénář (často se objeví nepřesnosti z minula), třídní přístup (objektově orientovaný) – vše je třída.

scénář – podrobné rozepsání komunikace aktora se systémem

impuls – komunikace směrem od uživatele k vytvářeného systému (požadavek nebo odpověď na dotaz systému)

reakce – odpověď systému uživateli

faktorizace – relace mezi modely jednání, odstraňuje opakování společných postupů

extenze – relace mezi modely jednání, typ jednání, které doplňuje nebo rozšiřuje jiný typ jednání

Rozšířený model jednání - někdy je nutné vyjádřit vztah mezi typy jednání a konkrétními třídami, nedoporučuje se odvozovat třídy přímo z typů jednání.

3.1.5 Model spolupráce

Slouží k odvození objektového modelu z modelu jednání

- tvorba seznamu tříd a seznamu odpovědností (úkolů)
  - více méně intuitivní
  - podstatná jména – kandidáti na třídy
- přiřazení zodpovědnosti třídám
  - zodpovědnosti se beze zbytku a duplicit rozdělí mezi třídy
- tvorba vlastního modelu spolupráce
  - pro každou třídu se nakreslí diagram spolupráce
  - slouží pro návrh objektového modelu

3.1.6 Objektový model, diagram tříd

Snaha zachytit statické vztahy ve vytvářeném systému, slouží k reprezentaci statické struktury systému, odlišení základních prvků systému a specifikace jejich statických vlastností.

objekt a třída :

„dělník systému“, vlastník datových hodnot a nositel chování, každý objekt má svou identitu (jeho odlišnost je dána tím, že existuje). Třída je zobecnění objektu, ale při běhu systému pracují objekty (objekt je instancí třídy). Každý objekt má svou vlastní zodpovědnost

atributy: vnitřní data objektů z reálného světa (vlastnost reálného světa), vnitřní identifikace (z důvodů vnitřní organizace)

- linkové (vlastnost přísluší vztahu mezi třídami)
- odvozené (závislé, lze odvodit jeden z druhého)
- základní (neodvozené)

asociace

- obecná
- agregace – speciální případ, kdy objekt agregující třídy obsahuje jiné objekty
- odvozené – lze vydělit jako redundantní
- vlastnosti asociace
  - násobnost (kvantifikace)
  - pořadí
  - kvalifikace

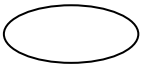
- metoda - procedura nebo funkce, která se provádí v rámci objektu
- operace
  - generalizace metody, určuje, co se dělá, metoda určuje, jak se to dělá
  - s vedlejším efektem – modifikují data objektu
  - dotazové – neovlivňují data
  - organizační – rozhodují o dalším postupu výpočtu
  - přístupové – import nebo export dat
  - implementační – implementují algoritmus nebo jeho část

- vztahy
  - link – spojení objektů nejrůznějších druhů
  - asociace – vztah mezi třídami
  - kvantifikace
- hierarchie dědění
  - dědění je uspořádání tříd do hierarchie tak, že níže umístěná třída (potomek) přebírá metody a/nebo atributy od výše umístěné třídy (předek)
  - generalizace – společné vlastnosti skupiny tříd se zobecní do třídy (CObject)
  - specializace – z obecné třídy se odvozují potomci
  - zastínění – dvě stejné metody se stejným jménem, předefinování
  - redukce – druh specializace, kdy potomek má méně možností než předek
  - abstraktní třídy – určují tvar operací, definují protokol styku mezi objekty

3.1.7 Funkční model

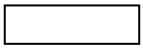
Snaha vytvořit kontrolní pohled na vytvářený systém, identifikovat a popsat algoritmy

- popis vztahů mezi procesy a daty
- proces



  - obsahuje výpočet (algoritmus nebo jeho část)
  - aktivní prvek systému (data jsou pasivní)
  - hlavní úlohou procesu je popsat transformaci dat
- datový tok

  - dynamický pojem, předávání datových záznamů

    - generalizace předávaných dat, předávaná data jsou instancí datového toku
- datový sklad
    - úložiště dat (datová paměť)
    - pasivní prvek systému
- terminátor (aktér)
    - část okolí vytvářeného systému, která s ním komunikuje prostřednictvím datových toků
    - synonimum aktéra 

- diagramy
    - kontextový diagram – terminátory a datové toky mezi nimi a vytvářeným systémem
    - systémový diagram – základní procesy, na které se rozkládá vytvářený systém
    - běžný diagram (DFD – data flow diagram) – zjemnění jednoho procesu na síť procesů a datových skladů komunikujících spolu pomocí datových toků
- listové procesy – nejsou zjemňovány, nepřísluší jim diagram
- agregované datové toky – označují více datových toků s tímtéž producentem a konzumentem
- řídící prvky (procesy) – přenos Booleovské hodnoty, která ovlivňuje proces

- textový popis
        - neoddělitelná součást grafického popisu modelu
        - popis datového skladu – musí obsahovat informace o struktuře dat do té míry, aby se dal porovnat s deklarací atributů odpovídající třídy
        - popis procesu
        - popis transformace
    - definuje účinek procesu (výstupní hodnoty jako funkce vstupních hodnot a vedlejší efekty na objekt)
    - lze použít následující prostředky
        - základní matematické funkce
        - tabulky vstupů a výstupů
        - rovnosti (funkce obecně)
        - pre- a post- podmínky
        - rozhodovací tabulka
        - pseudokód
        - přirozený jazyk
    - popis omezení
        - platnost transformace na určitý definiční obor nebo podmínky transformace
    - množnosti optimalizace
        - je vhodné poznamenat si možnosti optimalizace (zejména u složitých operací)

3.1.8 Dynamický model

- snaha zachytit dynamické vztahy ve vytvářeném systému
- tvorba stavových diagramů popisujících chování objektů jednotlivých tříd
- používá
    - slovní scénáře

    - meziobjektové interakce (grafické scénáře)
    - mapu událostí
    - stavové diagramy

- událost
    - podnět nebo impuls, kterým jeden objekt nebo aktor vyžaduje reakci jiného objektu nebo aktora
    - událost – komunikace jednoho prvku s druhým
    - externí událost – událost z okolí zkoumaného prvku
    - událost z vnitřku objektu
- impuls systému – událost směřující od uživatele k systému
- reakce systému – událost směřující od systému k uživateli
- stav – podmnožina hodnot atributů objektu

- přechod – změna hodnot atributů, že objekt přejde z jednoho stavu do druhého (obvykle důsledek události)
- aktivita – jakákoliv logicky konzistentní aktivita, kterou realizuje objekt své chování (váže se na událost nebo stav)
- akce – provedení operace objektu (doba trvání je nulová)

- slovní scénáře
    - snaha slovně popsat komunikaci mezi uživatelem a vytvářeným systémem
    - posloupnost vět, které popisují požadavek aktora, odpověď systému, nový požadavek aktora atd.
- grafické scénáře
    - identifikace událostí vyměňovaných mezi třídami
    - vychází ze slovních scénářů, snaží se odvodit chování jednotlivých tříd
    - viz sekvenční diagramy v CASE nástrojích
- mapa událostí
    - zobrazení všech událostí v jediném diagramu
- stavový diagram
    - stav – určité hodnoty proměnných
    - přechod – změna hodnot atributů
    - událost – vyvolá akci, která způsobí přechod mezi stavy (komunikace mezi objekty)
    - aktivita – nevětvící se posloupnost akcí
    - akce – činnost jednoho objektu
    - popisuje životní cyklus všech objektů jedné třídy
    - hierarchizace stavových diagramů
        - jeden zobecněný stav můžeme rozepsat jako samostatný stavový diagram
- stavová tabulka
    - každému řádku tabulky odpovídá jeden stav
    - každý sloupec odpovídá jedné události
    - v příslušných políčkách je popsána reakce objektu na událost, oddělený středníkem je zapsán stav, do kterého objekt přejde
    - odpovídá nekontextové gramatice
    - svádí ke špatně strukturovanému zápisu
- model událostí
    - snaha popsat všechny činnosti systému jako reakce na vstupní impulsy
    - identifikace potřebných aktivit
    - popisuje činnost systému mezi vstupním impulsem uživatele a reakcí systému jako serio-paralelní síť
    - skoro výhradně slouží k porovnání

3.2 Specifikace požadavků na systém

Ideálem je pevná, přesná, úplná a bezesporná specifikace, kdy lze produkci zcela oddělit od zadavatele. Není tomu tak vždycky, často je zapotřebí vytvořený systém dále upravovat a přizpůsobovat – zadavatel sám nemá přesnou představu o celém, komplexním zadání, nebo ji formuluje nesrozumitelně, nejednoznačně, částečně. V průběhu času se zadání může objektivně, nebo subjektivně změnit.

Neformální specifikace (odborný článek) – častý jednoduchý způsob, použití přirozeného jazyka neposkytuje dostatečně přesnou formu vyjádření.

Formální specifikace (podrobnější informace jdou za rámec textu) - formální aparát, jehož sémantika je jednoznačně a exaktně definovaná, vyjádření zadání je velice náročné (znalost formálního aparátu, podrobná analýza problému), ale pokud je úspěšně zvládnuté, může být vhodným vstupem pro automatickou podporu v dalších fázích vývoje IS (systémy CASE). Používá se matematická logika, algebraické a funkcionální specifikace

Nefunkční požadavky - definovány jako omezení kladené na systémovou službu (například cenové omezení). Jsou děleny do 3 základních tříd:

- požadavky kladené na výsledný program – rychlost, přenositelnost apod.
- požadavky kladené na řešení – použití standardů, programové prostředí apod.
- vnější požadavky – všechny ostatní (např. výše rozpočtu)

Přesná formulace nefunkčních požadavků chrání obě strany – zákazníka i řešitele. Prověření (validace) požadavků se skládá z validace potřeb uživatele, prověření konzistentnosti požadavků, prověření úplnosti a reálnosti požadavků.

Validace specifikace - specifikace musí být bezesporná, úplná, musí odpovídat skutečným požadavkům, musí být realizovatelná. Provádíme testování specifikace – pro vytvořenou specifikaci použijeme dostatečně velký soubor testovacích dat (musí obsahovat všechny typické případy), vytvoříme prototypové řešení přesně podle specifikace, prototyp se předloží zadavateli ke konfrontaci s jeho představami. Upřesňují se rozpory a nejasnosti.

3.2.1 Návrh uživatelského vzhledu aplikace

Tvoří nedílnou součást specifikace programového díla. Zásady:

- princip prvořadosti uživatele – program musí komunikovat podle zvyklostí uživatele
- princip jednotnosti – zobrazení, formát, selekce, vstupy ve shodném stylu
- princip vlídnosti – vzhled má pomáhat překlenout neznalost či chyby uživatele

Základní typy vzhledu

a/ příkazově orientované systémy - příkazy ve formě textového řádku. Výhody:

- minimální prostředky pro komunikaci
- výhodné pro zkušené uživatele
- jednoduché rozšíření

Nevýhody:

- uživatel musí umět příkazový jazyk
- musí se ošetřit chyby vstupu

b/programy řízené pomocí výběru z nabídky - systém nabídne uživateli několik možností (menu), uživatel volí postup. Výhody:

- snižuje se potřeba psaní klávesnicí a učení se jazyku
- znemožňuje zadání nepřípustné akce
- snadnější je realizace kontextově závislých nápověd

Nevýhody:

- obtížné jsou akce vyžadující kombinaci několika voleb
- pro zkušené uživatele může být zdlouhavé
- při rozsáhlých nabídkách je třeba se orientovat

c/uživatelský vzhled ve stylu přímého řízení - pohyb v modelu informačního prostoru pomocí řídícího panelu. Výhody“

- uživatel program ovládá, není ovládán programem
- chyby v činnosti jsou bezprostředně detekované
- čas učení je obvykle krátký

Nevýhody:

- musí existovat dostatečně ilustrativní a přehledný model
- u větších systémů může nastat problém navigace v modelu

d/ komunikace v přirozeném jazyce (spíše budoucnost). Výhody:

- uživatel se nemusí nic dalšího učit
- pokud zvukový přístup, odpadá psaní na klávesnici

Nevýhody:

- vyjádření v přirozeném jazyce je většinou rozsáhlé a nejednoznačné
- tyto programy jsou mnohem nákladnější než ostatní

3.3 Analýza systému

Analýza zahrnuje studium problému před tím, než podnikneme nějaké akce směřující k jeho řešení. Analyzujeme typicky existující systém (není ještě ujasněný, mění se podmínky, zadání, …), nebo nový systém (zákazník většinou ještě nemá přesnou představu). Produktem je specifikační dokument, který obsahuje cíl řešení, požadovaný výsledek, podrobně dokumentovaný cílový stav a důležité průvodní parametry spojené s řešením a provozem. Při náhradě stávajícího systému je přirozeně požadována kompatibilita – adekvátní reakce na všechny relevantní vnější podněty, které vnímal i dřívější systém - tedy nový systém zaručí stejné a nově požadované funkce a chování a bude mít shodnou nebo dokonalejší odezvu na podněty z okolí. Analytický přístup předpokládá komplexní zkoumání a porozumění problému, definování podmínek pro jeho řešení, vytvoření několika alternativních řešení, vyhodnocení přínosů a výběr optimálního řešení. Následuje fáze zpřesnění detailů a implementace při soustavném sledování, zda v procesu získáváme požadované výsledky. V podnikových IS se do analýzy promítá znalost obchodních funkcí, organizační struktury, managerských technik a pracovních procesů podniku. U lidských zdrojů se zohledňuje, jak lidé pracují, myslí, reagují na změnu, učí se a komunikují. Analytické modely obsahují

- seznam funkčních a nefunkčních požadavků – jednotlivé požadavky na systém
- seznam událostí a reakcí – model vnějšího chování systému
- požadované výstupy – obrazovky, tiskové sestavy, formuláře
- procesní model – logický sled a podstata transformací údajů
- datový model – data, která systém přijímá, se kterými pracuje, a která produkuje
- prezentační model – pomocí obrázků znázorňuje začlenění systému do okolního světa
- matematický model – definuje vlastnosti systému exaktním způsobem pomocí formálního matematického aparátu

3.3.1 Strukturovaná analýza

Slouží k řešení nových cílů s následujícími vlastnostmi:

- produkty analýzy musí být velmi dobře udržovatelné
- větší problémy musí být efektivně rozděleny na menší
- všude, kde je to možné, je třeba užít grafického vyjádření
- je třeba odlišit logické a fyzické aspekty a podle toho rozdělit odpovědnost
- logický model je nutné vytvořit proto, aby se uživatel seznámil se systémem ještě před jeho realizací

Charakteristiky nástrojů:

- musí pomoci při klasifikaci požadavků a musí ji dokumentovat
- musí poskytnout prostředky pro sledování rozhraní, bez popisu jeho fyzické podoby
- umožní popsat logiku a smysl jednotlivých částí systému

Přínos strukturované analýzy spočívá v podpoře formulace obecně srozumitelných požadavků na systém (základ pro návrh a implementaci), efektivnějšího využívání zkušených i méně zkušených pracovníků, plánování a řízení vývoje, zvyšování kvality systému.

3.3.2 Vybrané nástroje analýzy

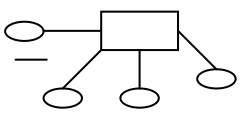
Pokud použijeme pouze jediný druh modelu, zdůrazníme jen ty vlastnosti systému, které model vyjadřuje

- funkční model
  - pokud není přesně známo, jaké údaje do systému vstupují, pak mohou chybět některé funkce (transformace dat)
- datový model

3.3.3 ER-diagramy

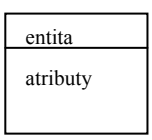
Popis datové struktury informačního systému reprezentují datové modely (nejčastěji v databázových systémech jsou to ER-diagramy). Základem pro výchozí návrh struktury jsou požadavky zadání s informacemi o tom, které údaje musí být uchovány v systému a v jakém jsou vztahu, tj. vztahy mezi entitami a jejich kardinalita, obecné asociace i agregace, povinné i nepovinné členství ve vztahu, rekurze, tj. vztahy mezi jednotlivými instancemi jednoho entitního typu, vzájemně výlučné vztahy (vztahy, které nemohou existovat současně), vztahy mezi nadtypy a podtypy entit (ISA hierarchie). Používají se různé notace. Klasická Chenova používá grafické prvky: entita – obdélník, atribut – elipsa (primární klíč – podtržen)

pokud nejsou zřejmé všechny transformace vedoucí ke vzniku některých údajů, mohou být v modelu nadbytečné či rozporné a chybné informace

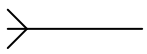


Úspornější notace (vše v obdélníku)

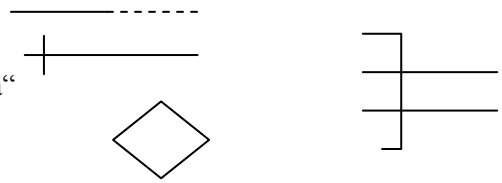
- primární klíč - uvozen #
- nepovinné členství – o
- povinné členství - *



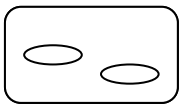
- vazby (čáry)



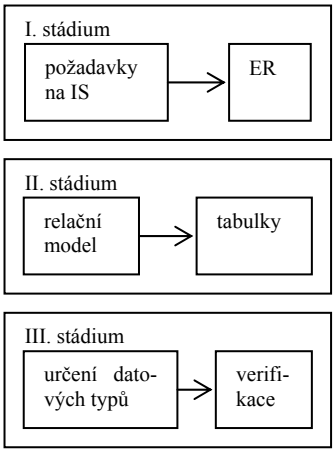
- povinné členství – plná (nebo „muří nohy“)
- nepovinné členství - tečkovaná
- slabá entita (nestačí k identifikaci) – kolmá čára
- výlučné vztahy (pokud se nepřekrývají) – „ohrádka“
- pojmenovaná vazba - kosodélník



- jiný pohled – množinový zápis



- tvorba návrhu



I. definice nezávislých (referenčních) entit, definice vztahů a jejich typů, definice závislých entit (slabých), hledání klíčových atributů

- nesrovnalosti (řeší se iteračně, dokud není v pořádku)

II. definice atributů, určení klíčů

- problémy s vícehodnotovými atributy, relacemi apod. (pokud s převodem počítáme od začátku, můžeme si práci usnadnit)

sémantický relativismus (atribut mohu realizovat jako entitu a opačně), existuje vždy více možností (rozhoduje zkušenost, funkčnost aplikace, …)

Doporučení pro tvorbu ER-diagramů:

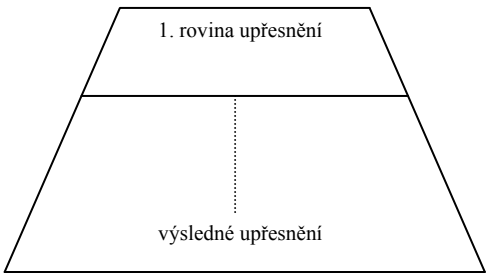
- preferovat horizontální kreslení vztahů
- kardinality 1:N – typ s N vlevo
- minimalizované šikmé čáry a křížení čar
- paralelní čáry ne těsně u sebe
- je možné používat strukturovaný obrázek podmnožin hierarchie

Doporučení pro entity (tabulky):

- smysluplné jméno v jednotném čísle
- výlučnost jména v kontextu projektu
- nejméně 2 atributy, ne více než 8 atributů
- vyrovnat se s homonymy/synonymy v názvech entit
- musí existovat alespoň jeden proces, který s entitou pracuje, mění se entita v čase? (historie)
- vyhovuje požadavkům na normalizaci?

podtypy entit

- v rámci entity výlučné
- vlastní identifikátor
- úplná množina podtypů



Doporučení pro atributy:

- název v jednotném čísle
- neměl by zahrnovat jméno entity
- jednoduchý

- známá definice formátu, délky, domény
- Otázky: Není lépe chápat jako entitu? Není již definován v rámci nadtypu? (zdvojení) Jak se mění v čase?

vazby

- kardinalita, typ členění (povinné / nepovinné)
- Není redundantní?

Zpřesňování návrhu datového modelu může probíhat podle různých strategií:

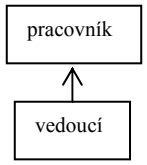
- princip shora dolů - nejdříve celek, postupně doplňujeme detaily
- princip zdola nahoru (zevnitř ven) - nejdříve jednotlivé entity, hledáme vztahy mezi nimi
- smíšená strategie - slévání dílčích podschémat

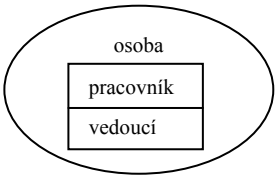
V praxi, při integraci datové struktury z více zdrojů (různých skupin analytiků) se nejčastěji setkáme s problémy:

- rozdíly ve struktuře
- konflikty jmen (dvě etity, jména mohou znamenat stejný reálný objekt, nebo naopak jedno jméno, entita může znamenat dva reálné objekty)
- pohledy (každý požaduje jiné vazby a musí se vyřešit)

Příklad analýzy konfliktů

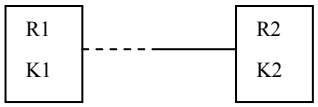
– Potřeba integrovat entity pracovník a vedoucí



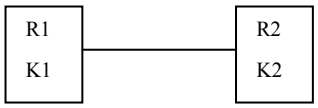


A – hierarchicky pomocí vazby B – vytvořit entitu (Osoba) s vlastnostmi pracovníka i vedoucího

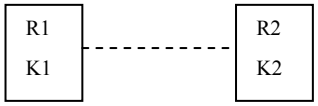
- Vzory pro realizace vazeb



R1 (K1,A)
R2 (K2, B)
K1 - unique not null
foreign key K1 references R1(K1)

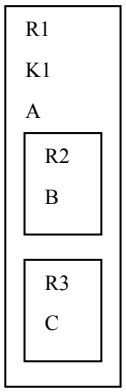


R (K1, K2, A, B)



R1 (K1, A)
R2 (K2, B)
K1 unique
foreign key K1 references R1(K1)

- realizace typů, podtypů



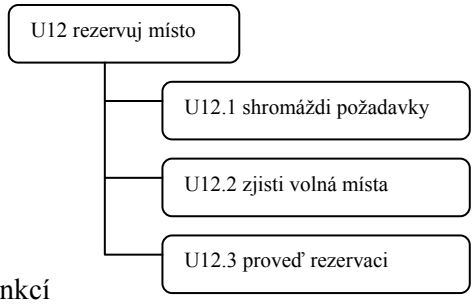
a) R1 (K1, T, A, B, C)
   - T rozlišuje podtypy
   - poměrně řídká tabulka

b) R1 (K1, A, B)
   R2 (K1, A, C)

c) R1 (K1, A)
   R2 (K2, B) foreign key K1 references R1(K1)
   R3 (K3, C) foreign key K1 references R1(K3)

- nejúspěšnější (ale musí se udělat vazby)

3.3.4 Hierarchie funkcí



Klasický prostředek funkční analýzy.

Vytváří se hierarchie funkcí (viz příklad na obrázku)

- Události, potřeby a jejich zpracování ve formě funkcí
- Hledají se funkční závislosti (porovnání závislostí mezi funkcemi organizace, vzájemné efekty mezi funkcemi)

Možná klasifikace funkcí:

- pracovní funkce – vztahují se na úrovni k činnosti organizace (kryje se s pojmem proces). Doporučení: slovní mnemotechnický popis – výstižné (nejlépe imperativ), dobré nezavádět do názvu popisy realizace („co“ uvádět, „jak, kdo, kde, kdy“ neuvádět)
- elementární funkce - má logický význam transakce, pokud nějaká substruktura selže, vše se vrátí, není to listová funkce (vznikají z ní další procesy)
- společná funkce - vyskytuje se na více místech (modul zvlášť)
- hlavní funkce - střed popisu funkčnosti, soustřeďuje detailní význam chodu systému, je v nich soustředěn smysl funkcionality

pokud není seznam událostí a odpovědí úplný, může systém reagovat nečekaným, nepřiměřeným a často i nepřípustným způsobem na události neuvedené v seznamu

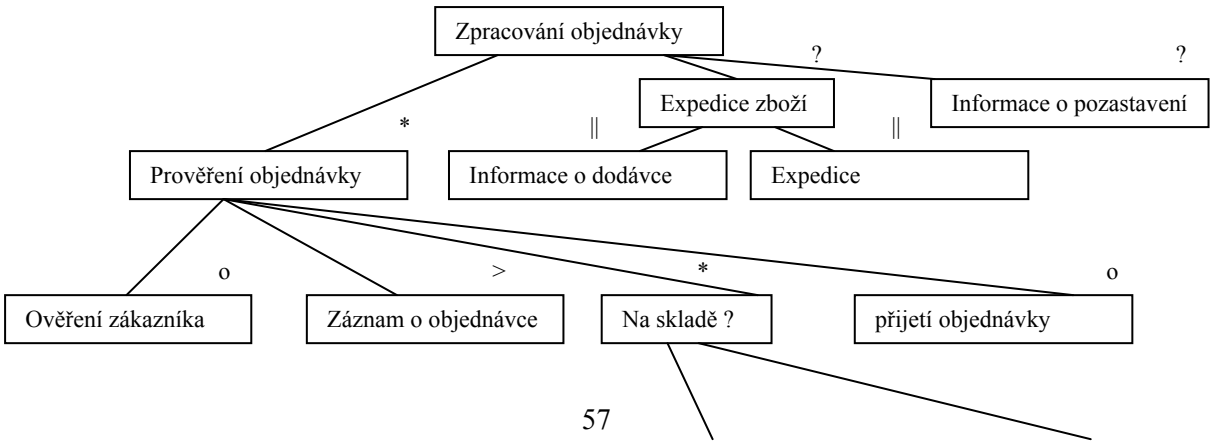
Les stromů s funkčností systému

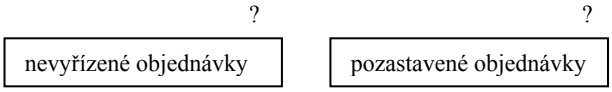
- získá se s odborníky na problematiku, křížové kontroly pracovníků, přesné schéma (na každé úrovni) napsáno srozumitelným způsobem, úplné, stručné
- vyvážené tak, že je snadné ho číst

datová stránka – musíme otestovat nebo zahrnout zmínku, co která funkce provádí s entitami (jak se mění, odstraňují apod.), popis logiky (algoritmus dané funkce)

Příkladem metody, vycházející ze stejných principů jsou Jacksonovy strukturogramy. Pro rychlejší rozlišení se označují uzly grafu (funkce) doplňujícími sémantickými značkami:

o	selekce, výběr	>	Povinná sekvence
*	iterace, opakování	||	Paralelní činnost
?	exkluzivní selekce	#	Předčasné ukončení
Bez označení	terminátor		





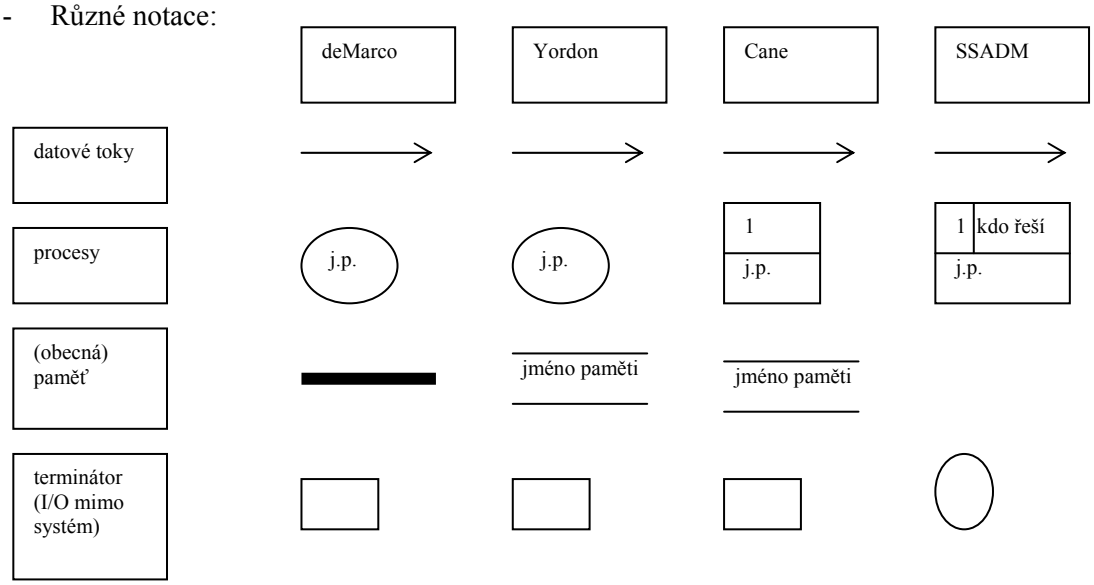
Obrázek 10 Příklad Jacksonova strukturogramu

3.3.5 Diagramy datových toků

Základní konstrukty DFD:

- datové toky – pohyb skupin údajů mezi částmi systému, procesy, či systémem a okolím, např. pojmenování výstupního toku by mělo vyjádřit podstatu transformace vstupu
- procesy – transformují vstupy na výstupy, jméno vyjadřuje činnost procesu
- paměti – zobecněná kolekce dat v klidu, pasivní – data se zapisují na explicitní příkaz
- terminátory – zdroje a příjemci dat v okolí systému – externí entity

- Různé notace:



Podle úrovně detailu tvoří jednotlivé DFD diagramy hierarchickou strukturu. Kořenem je **kontextový diagram**, kde celý IS je chápán jako jeden proces, zajímají nás terminátory a vnější vstupní a výstupní datové toky. Popisuje vnějšího chování systému, jednoznačné vymezení hranic systému.

Na další úrovni často klasifikujeme DFD jako systémový diagram, který může sloužit pro pochopení základních funkčních bloků systému. Doporučená dekompozice je 1 proces na 3-7 v další úrovni. Dekompozice končí v listech, kde už by měla být transformace vstupních dat (algoritmus) snadno implementovatelná. Na všech úrovních musí být zachovány spoje z vyšší úrovně DFD (zachování návaznosti). Konzistence DFD je kontrolována odstraněním procesů, které mají jenom vstupy, nebo naopak jenom výstupy. Kontroluje se také pojmenování všech konstruktů.

Listové procesy slouží jako podklad pro implementaci funkcí modulů, nebo metod objektů, musí to být jednoznačné a jasné algoritmy transformace vstupních dat na výstupní a označuje se jako minispecifikace. Může mít různé formy:

- s výhodou se používá **strukturovaná angličtina** (čeština), což je popis algoritmu procesu pseudoprogramovacím jazykem, jehož klíčová slova jsou anglická a s jejich pomocí se formulují základní programové konstrukty (například větvení a cykly). Záměrem je podpora

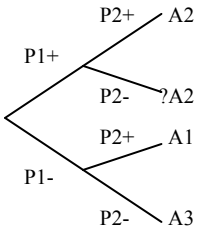
jednoduše realizovatelné implementace ve zvoleném programovacím jazyce. Slovník je složen z příkazů (sloves), výrazů z datového slovníku a klíčových slov pro formulaci logiky algoritmu.

- **rozhodovací tabulka** se skládá ze čtyř kvadrantů. V prvním kvadrantu jsou vyjádřeny logické podmínky, nebo parametry procesu, ve druhém všechny kombinace hodnot z podmínek v prvním kvadrantu. Ve třetím kvadrantu je seznam všech možných akcí a reakcí. Ve čtvrtém kvadrantu je nakonec názorné přiřazení jednotlivých kombinací hodnot ze druhého kvadrantu a jim odpovídajících akcí.

Př. Rozhodovací tabulka

P1 (podmínky)	0	0	1	1
P2	0	1	0	1
A1 (akce)		x		
A2			?	x
A3	x			

Rozhodovací strom



- **rozhodovací strom** je funkčně velice podobný. Od kořene se naviguje po vnitřních uzlech podle jednotlivých podmínek, nebo parametrů v příslušné úrovni stromu, dosažení listové úrovně potom definuje příslušnou akci, nebo posloupnost reakcí.

3.3.6 Stavový diagram

Stavový diagram (STD) umožňuje zachycení i časových souvislostí (událostí) zpracování dat v procesu - modeluje časově závislé chování systému.

- lze použít samostatně jako nezávislý model chování systému

  Vychází ze seznamu událostí (soupis podnětů, které působí na systém z jeho okolí). Kategorie událostí: F-událost – vstup dat do systému,

  T-událost – spojena s časem, nezávisí na příchodu dat nebo povelu do systému,

  C-událost – zvláštní případ T-události, jedná se o nepředvídatelnou událost

- formálně automat, jehož přechody jsou řízeny výskyty událostí. Systém přechází z jednoho stavu do následujícího stavu. Hrana STD je ohodnocena dvojicí událost / akce

Silnějším a obecnějším nástrojem pro časové souvislosti procesu jsou **Petriho sítě.**

3.3.7 Datový slovník – popis metadat

Datový slovník slouží k přehlednému a komplexnímu popisu nejrůznějších typů dat v informačním systému:

- elementární data – známé typy dat, nutno zadat přípustný obor hodnot, význam v kontextu
- strukturovaná, složená data – lze popsat odkazem na jednotlivé složky. Často používaná notace pro definici gramatiky, popisující strukturu, obsahuje konstrukty:

=	Skládá se z
+	A zároveň
( )	Nepovinná část
[|]	Výběr jedné z možností
{ }	Iterace (počet od - do) př. $_2\{\ \}_{10}$
* *	komentář
@	Identifikační klíč

Př.

objednávka = číslo objednávky + datum vystavení + id zákazníka + jméno zákazníka + (datum dodání) + { číslo výrobku + jméno výrobku + počet objednaných výrobků + (cena za kus) }

- datové struktury (data flow diagram – paměti, datové toky, listové procesy)
- popis proměnných, modulů, formulářů, atd.

Příklad doporučených položek pro vybrané typy dat

Datový Element -	Datová Struktura -	Proces z DFD -
jméno:	jméno:	jméno:
popis:	popis:	popis:
typ, doména:	Složení:	toky vstupních dat:
formát:	Hodnoty:	toky výstupních dat:
hodnoty (rozsah):	výskyt:	logika zpracování:
výskyt:	komentář:	výskyt:
komentář:		komentář:

Mezi další klasické metody patří využití dvojrozměrných tabulek k nejrůznějším účelům - mnoho aspektů analýzy lze reprezentovat pomocí matic. Příkladem je matice afinity - je výhodné vytvořit matici, která ukazuje, na kterých formulářích se vyskytují jednotlivé údaje, a jaké operace se provádí (řádky = údaje, operace, sloupce = obrazovky)

3.3.8 Klasické metodologie analýzy

Strukturovaná analýza – DeMarco

- připisováno prvenství použití diagramů toků dat pro analýzu informačních systémů určených pro běžné organizace. Charakteristika postupu:

1. studium stávajícího fyzického prostředí
2. odvození logického ekvivalentu fyzického systému
3. odvození nového logického systému
4. odvození fyzických charakteristik nového systému
5. kvantifikace cen a termínů
6. výběr jedné z možností
7. začlenění nových fyzických DFD do strukturované specifikace společně s datovým slovníkem a popisem procesů

Logické modelování – Gane/Sarson

Charakteristika postupu:

1. vytvoření systémového DFD, který postihne podstatu činnosti systému
2. návrh výchozího datového modelu
3. analýza entit a jejich vztahů – logický datový model
4. tvorba relačního datového modelu, normalizace
5. zpřesnění DFS tak, aby zachytil změny provedené v datovém modelu – logický procesní model
6. logický model procesů a dat se dekomponuje na procedurální jednotky
7. specifikace detailů každé procesní jednotky

Pohledová analýza – systém nemá přirozeně definovanou hierarchii dat (entit, atributů), charakteristika:

1. identifikace pozorovacích bodů – vytvoří se množina pohledů na systém (atributy)
2. sdružení pohledů do skupin (datové, servisní, nefunkční, uživatelské pohledy), které se nepřekrývají, funkce se řeší v rámci jednoho (funkčního) pohledu, popisuje zpracování informace
3. určení struktury pohledů (hraniční, definiční pohledy) – uspořádání do hierarchického diagramu
4. Vytvoříme tabulkové diagramy akcí (položky tabulky: zdroj dat, vstupní data, akce, výstupní data, příjemce dat) a akční diagram

Datově orientované přístupy (Warnierr/Orr – DSSD- Data Structure Systems Development)

- není formulováno jako striktně stanovená a popsaná metoda
- souhrn zkušeností, které vedou ve většině případů k úspěšnému dokončení projektu
- nejlepších výsledků se dosáhne v případě, kdy struktura programu odpovídá hierarchické struktuře datového modelu
- cílově orientovaná strategie – známe požadované výstupy systému a z nich odvodíme logické výstupy, logické vstupy a fyzické vstupy systému
- z požadovaných výstupů se nejdříve modeluje logická databáze

Yourdonova Moderní strukturovaná analýza

- soustřeďuje se na nalezení esenciálního modelu systému
- vyjadřuje podstatu systému, nezávisí na technických a jiných implementačních omezeních
- má dvě části, model okolí systému a model chování systému
- využívá dekompozici na základě událostí

Charakteristika postupu:

1. vytvoř seznam událostí
2. pro každou událost vytvoř jeden proces
3. proces pojmenuj podle odezvy, která je požadována
4. zakresli potřebné vstupní, výstupní toky, paměti údajů
5. výsledný prvotní DFD porovnej s kontextovým diagramem a se seznamem událostí, ověř úplnost a bezespornost prvotního DFD

- postup je přímočarý a jednoduchý, výsledný DFD je však nepřehledný – následuje vyvažování DFD na různých úrovních

- ukrývání informace – na vyšších úrovních skryjeme paměť, ke které přistupuje pouze daný proces
- pro vytvoření seznamu událostí můžeme využít identifikaci událostí pomocí
  - entit
  - kontextového diagramu

Porovnání klasických postupů- DeMarco kladl velký důraz na modelování pomocí DFD, vyskytla se spousta problémů, Wad a Mellor rozšířili postup o stavové diagramy, Yourdon použil kombinaci výše uvedených metod

Metodologie SSADM

Vychází z datového modelu, odděluje logický a fyzický návrh systému, klade důraz na podchycení nestandardních a chybových stavů systému (jsou odhaleny dříve, než se začne implementovat program)

- snaží se minimalizovat chyby v návrhu pomocí technik (Životní cyklus entity, Process Outlines - kostry procesů a logických dialogů)

Vytváří tři základní modely systému

- modely entit – datový model, který ukazuje informaci uchovávanou v systému a vzájemné vztahy mezi jednotlivými datovými komponentami, vychází z klasického Chenova relačního modelu
- diagramy datových toků – funkční model ukazující informační cesty a transformaci informace
- životní cyklus entity – jak entita vzniká, mění se a zaniká z pohledu modelovaného systému

Obsahuje velmi podrobný postup a popis jednotlivých kroků. Charakteristika postupu:

1. studium stávajícího systému
2. vytvoření specifikace požadovaného systému
3. zpracování, aby šlo formulovat technické řešení
4. a 5. ukončení detailního návrhu již na logické úrovni
6. převedení logického návrhu na fyzický návrh

Výstup jednotlivých etap -diagram datových toků, model entit, životní cyklus entity, kostra logického návrhu, kostra logiky dotazovací funkce, kostra logiky aktualizační funkce, složený logický datový návrh

3.4 Strukturovaný návrh

Cíle a problémy návrhu spočívají v přechodu od výsledků analýzy k podkladu pro programování Je nutné během etapy návrhu zkonstruovat implementační model systému. Základní pojmy: modul - programová jednotka, která je samostatná a rozlišitelná při překladu, při kombinaci s ostatními jednotkami a při sestavování, je to logicky separovatelná část programu

Modularita - vyjadřuje míru rozkladu programu na diskrétní komponenty takové, že změna v jedno komponentě má minimální vliv na ostatní komponenty

modulární programování - je metoda pro vývoj programu složeného z kolekce modulů

Další nástroje strukturovaného návrhu

- diagram struktury systému (STC nástroj k zachycení návrhu), uzly stromu - představují funkce (procesy nebo akce), které musí být vykonány, kořen stromu - popisovaná funkce, hrany - reprezentují komunikaci mezi elementy
- transakční analýza - spočívá ve vyhledávání podgrafů v DFD, které reprezentují samostatné transakce
- transformační analýza - spočívá ve volbě tzv. centrální transformace, která bude tvořit vrchol programové struktury, od ní se odvíjí ostatní transformace
- objektově orientovaný návrh

Systémy CASE(Computer Aided Software Engineering)

CASE- množina integrovaných nástrojů, které představují náhradu klasických nástrojů (tužka a papír) při analýze a návrhu informačních systémů počítačem. Šetří a zrychlují práci a umožňují automatizovat některé fáze životního cyklu IS s podrobnou dokumentací a návazností. Představují prostředek pro zmírnění softwarové krize.

Kategorizace systémů CASE :

PRE CASE – pro práce předcházejících vývoji software (analýza požadavků uživatele, stanovení cílů produktu, naplánování jeho realizace, studie vhodnosti projektu)

UPPER CASE – pro počáteční stádia vývoje software (specifikace požadavků, funkční a datová analýza, návrh modulové struktury produktu)

MIDDLE CASE - pro pokročilá stádia vývoje software – přechod z analýzy do fáze návrhu (funkční a datový návrh, specifikace programových funkcí)

LOWER CASE – pro závěrečná stádia vývoje software – implementace a testování (řešení na fyzické úrovni technického vybavení, generování popisů databázových schémat, programování v pro gramovacích jazycích a překlad, ladění programů)

POST CASE – pro údržbu a dokumentaci produktu. Další možná hlediska klasifikace jsou odvozena z architektury, podporovaných metodik, producenta CASE, role uživatele, …

Datová báze projektu bývá v systémech CASE nazývána například jako REPOSITORY, DATA DICTIONARY, ENCYCLOPEDIA, atd. Použité metodologie se mohou v detailech lišit, ale snaží se kopírovat moderní a osvědčené postupy (preferované dodavatelem softwaru). Úrovně integrace CASE nástrojů:

- integrace platformy
- integrace údajů (sdílení dat a jejich struktury, soubory i databáze)
- integrace prezentace, stejné uživatelské rozhraní
- integrace řízení, procesu

Nástroje CASE často svádí k neopodstatněnému optimismu, že vyřeší všechny problémy. Jsou náročné na zvládnutí a poměrně drahé.

Shrnutí

Komplexní popis problematiky vývoje softwaru je detailněji popsán ve speciálních předmětech (softwarové inženýrství). Tato kapitola slouží ke získání přehledu při potřebě zvládnou analýzu a návrh jednoduchého IS pro ilustraci implementačních technik a technologií. Pro popis zadání si zvolíme neformální prostý text s popisem dat a funkčnosti IS ze strany zadavatele. Analýzu a návrh můžeme provést strukturovanými, nebo objektovými metodologiemi. Pro popis vnějšího chování a funkčnosti použijeme model jednání, nebo kontextový diagram. Pro datovou analýzu při orientaci na databázové uložení dat zvolíme ER diagram. Ve strukturované analýze a návrhu použijeme pro modelování funkčnosti IS hierarchie funkcí, nebo diagram datových toků a stavový diagram pro vyjádření časových souvislostí. Datový i funkční návrh zpřesňujeme v iteracích, kdy větší detail funkce ovlivní datovou strukturu a její změna, nebo modifikace zpětně může ovlivnit funkčnost. Pro efektivní práci se současnosti používají podpůrné počítačové systémy CASE, založené na moderních metodologiích analýzy a návrhu s přívětivým grafickým rozhraním a komplexní dokumentací všech fází životního cyklu IS.

Pojmy k zapamatování

- Strukturovaný a objektový návrh IS, Životní cyklus ŘS a projektu
- neformální a formální specifikace
- hierarchický rozklad, DFD, rozhodovací tabulka a strom, stavový diagram, Jacksonovy strukturogramy
- ER diagram, datový slovník
- model jednání, spolupráce, objektový, funkční, dynamický model

Kontrolní otázky

15. *Jak probíhá tvorba IS?*
16. *Jaké znáte metody strukturované analýzy a návrhu IS?*
17. *Jaké znáte metody objektové analýzy a návrhu IS?*
18. *Jak vypadá životní cyklus ŘS, projektu?*
19. *Jaká je charakteristika a rozdělení systémů CASE?*

Úkoly k textu

4. Navrhněte zadání jednoduchého IS pro účely následné analýzy a návrhu formou neformální specifikace.
5. Ze zadaného IS si vyberte podklady pro vytvoření ukázkových příkladů použití strukturovaných, nebo objektových metod analýzy a návrhu – DFD, stavový diagram, ER diagram, hierarchie funkcí, nebo Model jednání, spolupráce, objektový, funkční, dynamický model.

4 Podpora podnikových IS

Studijní cíle: Student by měl po prostudování kapitoly rozumět problematice dvou vybraných rozšíření standardních databázových systémů – podsystémům pro podporu analýzy dat – datovým skladům a OLAP a dále porozumět problematice e-Business systémů – podpora obchodování, zásobování, atd..

Klíčová slova: OLTP , datový sklad, OLAP, ROLAP, MOLAP, agregace, analýza, dolování a čištění dat, multidimenzionální modelování, hvězdicové schéma, schéma vločka, vícerozměrné pole, bitmapový index, e-Business, e-Commerce, e-Marketplace, Systém EDI

Potřebný čas: 4 hodiny

4.1 Datové sklady a OLAP

Datový sklad můžeme charakterizovat jako „Komplexní data uložená ve struktuře, která umožňuje efektivní analýzu a dotazování pro procesy rozhodování a řízení (podniku). Celý systém lze rozdělit na dvě základní části. Na vstupní straně stojí klasické databázové systémy a další zdroje dat, které se označují jako OLTP, (on-line transaction processing), druhá součást systému je vlastní datový sklad s vlastnostmi, které umožňují efektivně uložit data pro složité analytické dotazy. Principy OLAP - Analytické informace: pracují s primárními daty vytvořenými v OLTP, data očištěna od nadbytečných detailů, obsahují různé agregace, zachycují historii. Proces plnění datového skladu je někdy označován jako proces ETL (extraction-transformation-load). Data je třeba nejprve extrahovat z primárních datových zdrojů. Vzhledem k tomu, že jednotlivé primární datové zdroje nepracují s týmž datovým modelem, mnohdy nepoužívají ani tytéž datové typy, některé údaje nejsou v datových zdrojích obsaženy pouze implicitně a je třeba je odvozovat z jiných údajů, následuje krok transformace, který převede data získaná z jednotlivých datových zdrojů do unifikovaného datového modelu, nad nímž je možné vytvářet agregace a získaná agregovaná data pak uložit do datového skladu (fáze load).

Datový sklad je na rozdíl od OLTP databáze určen výhradně ke čtení dat pro potřeby nejrůznějších analýz

Typické charakteristiky a rozdíly mezi systémy:

OLTP (relační databáze)

- Operativně (každodenně) manipulují s aktuální agendou (INSERT, UPDATE), časté změny dat
- Struktura uložení je typicky normalizovaná (aspoň 3NF), dotazování se provádí nad specifickými logickými relacemi.

datový sklad, OLAP

- OLAP je součást systémů pro podporu rozhodování, informační technologie pro znalostní inženýrství
- vše v systému se soustředí na optimalizaci složitých dotazů,
- jiná struktura uložení dat (struktury ve formě hvězdice, vločky), nový způsob dotazování pomocí (multidimenzionální kostky)
- stálý objem dat v systému – občasná (periodická) aktualizace

Stručná srovnávací tabulka:

Znak	OLTP	OLAP,DWH

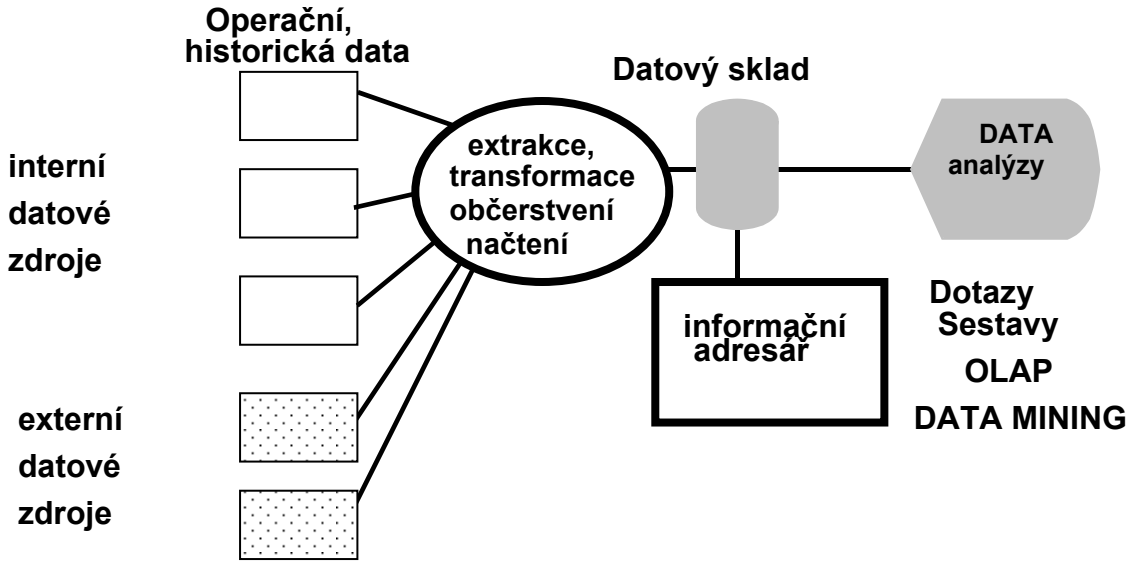
Charakteristika	Provozní zpracování, Podpora podnikových operací	Informační zpracování, Podpora rozhodování
Orientace	Transakční	Analytická
Uživatel	Úředník, databázový administrátor	Znalostní pracovník (manažer, analytik)
Funkce	Každodenní operace	Dlouhodobé informační požadavky, podpora rozhodování
Návrh databáze	Entitně-relační základ, aplikačně orientovaný	Hvězda/sněžná vločka, věcná orientace
Data	Současná, procesně orientovaná, detailní. Aktuální hodnoty	Historická, předmětně orientovaná, aktuální i historická, sumarizovaná
Sumarizace dat	Základní, vysoce detailní	Shrnutá, kompaktní
Náhled	Detailní (přesné na haléře)	Shrnutý, multidimensionální – (zaokrouhlené hodnoty)
Dotazy	standardizované dotazy, často opakované, generátory standardních zpráv	mnoho různých dotazů, přizpůsobení potřebám, sumaritace, agregace, trendy
Jednotky práce	Krátké, jednoduché transakce	Komplexní dotazy
Přístup	Číst a zapisovat	Většinou pouze číst
Zaměření	Vkládání dat	Získávání informací
Počet dostupných záznamů	Desítky	Miliony
Počet uživatelů	Tisíce	Stovky
Velikost databáze	100 MB až GB	100 GB až TB
Přednosti	Vysoký výkon, vysoká přístupnost	Vysoká flexibilita, nezávislost koncového uživatele
Míry hodnocení	Propustnost transakcí	Propustnost dotazů a doba odezvy
Užití	strukturované, opakované	ad-hoc, pouze částečně opakované
Příklady procesů	vstup dat, dávky	dotazy koncových uživatelů

Technologie datových skladů řeší klíčové problémy:

1. nedosažitelnost dat skrytých v databázích
   - historický problém (historie dat) – HW(čtecí zařízení) i SW(atypické formáty a hodnoty dat) problémy se staršími data
   - netriviální techniky (nestandardní protokoly) – třeba ODBC
2. dlouhé prodlevy, když se slabší systémy snažily vyhodnotit složité dotazy
3. složitá, uživatelsky nepříjemná rozhraní

4. cena v administrativě, složitosti a podpoře vzdálených klientů už byla vysoká (u starých systémů)
5. výsledek soutěžení mezi systémy transakčními a podporujícími rozhodování ovlivňuje i vývoj HW (klesající cena paměti, paralelní procesorové systémy)

4.1.1 Datový sklad



Obr. 11 Datový sklad (Data Warenhouse – DWH)

Označuje architekturu (obvykle založenou na SŘBD), která se používá pro údržbu historických dat získaných z databází operativních dat, která byla transformována, sjednocena (integrována) a zkontrolována před jejich zpracováním v DW databázi. Cíle:

- Shromažďování vybraných informací z různých databází do jednotného prostředí
- Zpracování analýz velkého rozsahu (i za dlouhá časová období)
- používá se pro prezentaci dat, testování hypotéz, objevování nových informací (zabezpečení, „data mining“ – dolování dat), ne přímo ale prostřednictvím dalšího SW (dává pouze datový základ)
- Monitorování stavu vstupních databází (MIS, CIS, …), zajištění zpřístupnění nebo konverzí dat do DWH

Jiná definice:

Strukturované rozšiřitelné prostředí navržené pro analýzu neměnících se dat, která byla logicky transformována z více zdrojových aplikací tak, aby byla uvedena do souladu se strukturou podniku. Data DW jsou aktualizována v delších časových intervalech, jsou vyjádřena v jednoduchých uživatelských pojmech a jsou sumarizována pro rychlou analýzu. Platí jistá analogie s výrobním procesem:

OLPT (výroba) → DW (sklad, distribuce) → OLAP (prodej)

DW je soubor dat pro podporu manažerského rozhodování, který je:

- předmětně orientovaný
- integrovaný
- časově rozlišený – dimenze času
- netěkavý (nonvolatile) - data jsou aktualizována v dávkách, ale po „snímku“ zamrznou
- Pro potřeby managementu

Úrovně dat:

- Stará detailní
- Běžná detailní
- Lehce agregovaná (sumarizovaná) data
- Těžce agregovaná (sumarizovaná) data
- Metadata

- optimalizace – rychlý výběr dat, sumarizace apod. (časově i zdrojově náročné úkoly)
- DW- nároky : dostatečná paměťová kapacita, kvalitní správa databází, obecně – použití mírně vylepšených stávajících databázových strojů

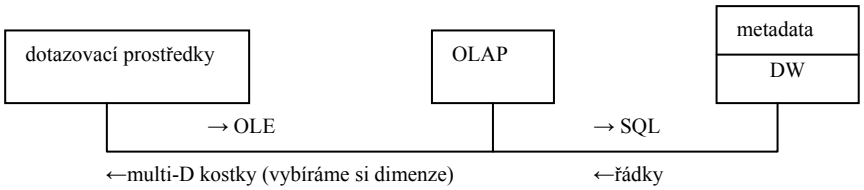
architektura DW,OLAP

- DW server je většinou relační SŘBD, někdy souborově orientovaný szstém
- OLAP server může mít být založený na relační technologii (ROLAP), což představuje rozšíření relačních SŘBD, ve kterém se operace s multidimenzionálními daty provádí prostědnictvím standardních relačních operací.
- Multidimenzionální OLAP (MOLAP) je speciálně navržený server, který přímo implementuje efektivní uložení multidimenzionálních dat a operací. Používá se i označení multidimenzionální databáze (MDD – MD SŘBD). Pro analytické operace jsou důležité multidimenzionální pohledy na data, kde relační databáze nejsou vhodné. Musí být podporováno nacházení souvislostí, které nejsou z primárních dat zřejmé.
- Na klientské straně je podpora analytických nástrojů, dolování dat (Data mining), tj. např. trendy, odhady, …

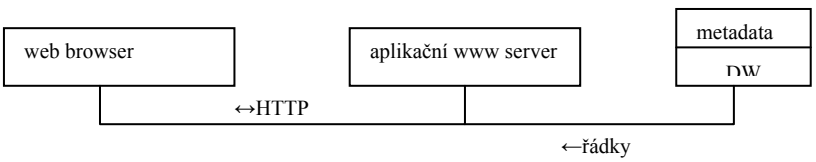
Struktura MDD:

Databáze je množina prvků, faktů –tj. bodů v multidimenzionálním prostoru. Prvek databáze obsahuje data nebo algoritmus pro jejich transformaci. Fakt (zisk, rozpočet, náklady, ceny, …) se vztahuje k vybrané skupine dimenzí v jistém rozsahu (measure dimension). Každá dimenze reprezentuje atributy (vlastnosti), často uspořádané ve formě hierarchie nebo svazu. Hierarchická struktura zajišťuje a umožňuje automatické agregace hodnot podle hierarchických úrovní (např. dílna – provoz – závod – podnik); drill-down (zpřístupnění detailů) proti drill-up. Umožňuje rychlé změny pohledů, ukládání do přehledných tabulek a grafů

Příklady používaných architektur: - tři vrstvy, klient- server



- použití Web serveru



V souvislosti s datovými sklady se můžeme setkat s pojmem datové tržiště (data mart). Tato architektura představuje dílčí, atomický DW s replikacemi dat pro ohraničené použití.

- Určena pro omezený okruh uživatelů
- Decentralizované datové sklady
- Možný mezistupeň při transformacích dat z produkčních databází

4.1.2 Datový model

- pro vyjádření schematu uložení se používá ER-model, data lze organizovat i klasicky pomocí speciálně navržené databáze, ale převážně se používá
- ***multidimenzionální modelování*** - uživatel přistupuje k datům jako k multidimenzionálním.
- Model připomíná postupy z tabulkových procesorů, ale pro více dimenzí.
- Dimenzionální model – Dimenzi si můžeme představit jako množinu dimenzionálních prveků, které představují reálné či abstraktní objekty podobného typu s jednotící charakteristikou – např. čas, poloha, …, odpovídají dimenzím podnikání organizace. Informace je typicky ve formě textových řetězců. Jedna dimenze (množina dimenzionálních prvků) je jednou osou informačního prostoru. Informační prostor může být n-dimenzionální. Prvkem informačního prostoru je fakt. Fakt je jeden údaj (jedna hodnota). Jedním údajem myslíme jeden atribut ve smyslu relační databáze, například množství, cena, zisk, typicky převažují číselné informace (jedním údajem může být i atribut typu BLOB obsahující složitá audiovizuální či grafická data). Fakt obsahuje také vazební informace, které jej váží ke konkrétním dimenzionálním prvkům z jedné či z několika dimenzí, může také obsahovat údaje o dvou časových intervalech, a to Validity Time a Transaction Time (logický interval platnosti údaje včetně vazeb na dimenzionální prvky a interval platnosti záznamu v databázi).
- Operace v multidimenzionálním datovém modelu:

  Agregace (roll up) – redukce dimenzí, sumarizace při průchodu agregovanou hierarchií (př. celkový zisk v lokalitě, regionu, ve státě)

  Selekce – vytvoření datové subkrychle

  Navigace k detailním datům (drill-down)

  Vizualizace dat – pomocí vhodných (dvourozměrných) průmětů do tabulek
- Při návrhu struktury DW nejsou klasické postupy s použitím ER-diagramu vhodné. Tabulky nemusí být v 3NF (je obecně proti optimalizaci rychlosti dotazu). Musí se respektovat multidimenzionální pohled. To vede ke konfiguraci, typicky tvořící hvězdici - hvězdicové (star) schéma: ve středu je vazební tabulka fakt a dimenzionální tabulky na okraji představují jakési číselníky dimenzí. Hierarchie dimenze není přímo explicitně ve struktuře vyjádřena. Ve faktech se nebojíme používat redundandně data, předzpracovat si agregační funkce a výpočty (i v různých rovinách řezu), připravit si vše, co by se muselo opakovaně počítat (hlavní snahou je zrychlení)
- Schéma sněhová vločka (snowflake) je podobné hvězdicovému, jen u dimenzí se připouští použití více tabulek pro optimálnější uložení dat jedné dimenze (normalizace skupiny tabulek, které reprezentují dimenzi), která představuje hierarchii dat (např. den- týden-měsíc-kvartál-rok).

Př. **schéma „hvězda „**

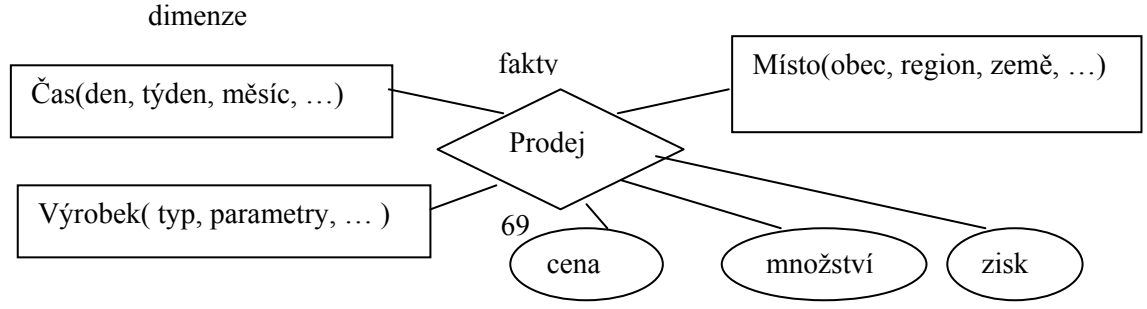
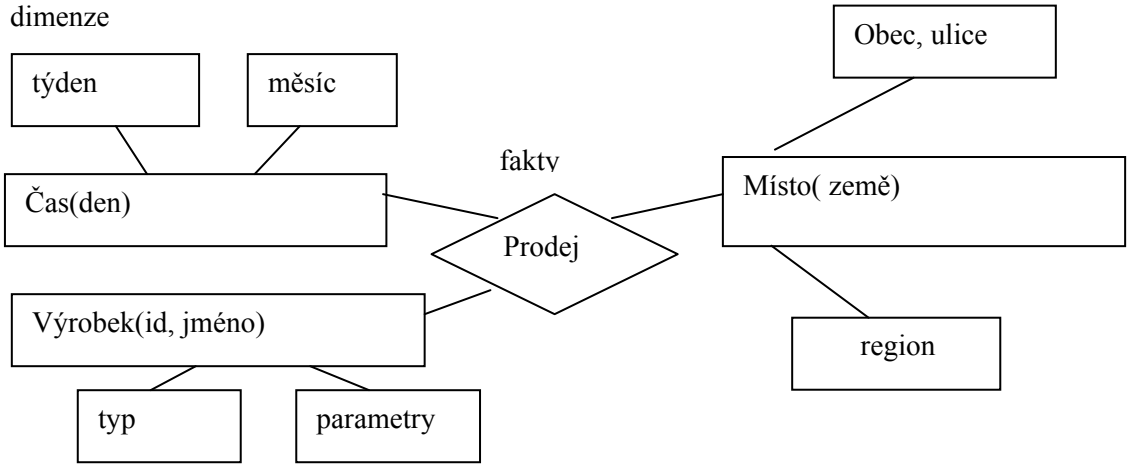
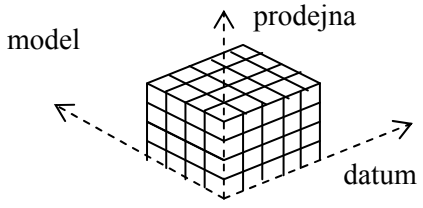


schéma „vločka „



- Multidimenzionální modelování můžeme na nižší úrovni abstrakce modelovat pomocí tabulek a vztahů mezi nimi (ROLAP). Jde o klasickou transformaci z ER-modelu, s použitím vazebních cizích klíčů v tabulce faktů a primárního klíče v tabulce dimenze. Častěji se implementuje multidimenzionální model (MOLAP) pomocí *vícerozměrných polí*, v terminologii OLAP se používá pojem multidimenzionální datová kostka. Pole nabízí proti tabulce informaci o počtu pozic v každé dimenzi, možnost seskupovat data formou řezů a toho využít při agregačních výpočtech. Podporuje dobře indexování, ale je neefektivní při využití paměťového prostoru při řídkých datech.

Průvodce studiem ***Schéma multidimenzionální databáze můžeme definovat jako množinu vícerozměrných polí.***



- Prvky kostky (krychle) bývají často bez hodnoty, protože nesouvisí všechno se vším

Podpora efektivity operací v multidimenzionální databázi je zajištěna pomocí nových indexovacích technik. Používají se ***bitmapové indexy***. Podstata přístupu spočívá ve vytvoření speciálních indexů pro jednotlivé vlastnosti s omezenou doménou přípustných hodnot, kde se mapuje primární klíč bázové tabulky do množiny možných hodnot vlastnosti binární 1 v příslušném sloupci, odpovídajícím konkrétní hodnotě atributu. Porovnávání, spojování i agregační operace se provádějí bitovou aritmetikou, což zrychluje tyto operace.

bitmapový indexy využívá rychlosti binárních bitových operací

Ilustrační příklad:

Bázová Tabulka

Cust	Region	Rating
C1	N	H
C2	S	M
C3	W	L
C4	W	H
C5	S	L
C6	W	L
C7	N	H

Index pro Region

Row ID	N	S	E	W
1	1	0	0	0
2	0	1	0	0
3	0	0	0	1
4	0	0	0	1
5	0	1	0	0
6	0	0	0	1
7	1	0	0	0

Index pro Rejting

H	M	L	
1	0	0	
0	1	0	
0	0	1	
1	0	0	
0	0	1	
0	0	1	
1	0	0	

Zákazníci (cust), pro které platí, že Region = W a Rejting = L

Při návrhu DW je výsledná architektura čtyřúrovňová, rozlišujeme schémata

konceptuální – multidimenzionální – databázové - fyzické

4.1.3 Komponenty DW

Komponenty souvisí s vytvářením datového skladu a obsahují prostředky pro:

- extrakci dat
- akvizici dat (generátory kódu, replikační nástroje, tvorba indexů, čištění dat - odstranění chybných dat kontrolou i nově formulovaných integritních omezení, vložení implicitních hodnot, eliminace - smazání duplicit, agregace, …)
- transformace z formátu vstupních (historických) dat do formátu DW
- řízení dat (archivace, autorizace, zálohování a zotavení z chyb, provoz, monitorování, ladění, řízení zdrojů)
- slovník informací – popis metadat, přístup k nim
- přístup k datům a dodání dat (OLAP, … řízené časem a událostmi)

4.2 e-Business a e-Commerce v řízení firmy

Globalizace ekonomiky se projevuje expanzí nových trhů a vytváří tlak na IS v oblasti podpory více jazyků, převodu kurzů různých měn, podchycení různých způsobů (kultur) obchodování - sjednocení mezinárodních dat, podpora ústní i psané komunikace v různých jazycích mezi uživateli, zákazníky a firmami. Prudký rozvoj internetových technologií – hlavně bezdrátové

komunikace a mobilních zařízení se prosazuje i v ekonomických aplikacích. Výhody internetové technologie v systémech pro podporu marketingových systémů:

- standardizovaná komunikační architektura.(technologie XML)
- Přímá komunikace mezi učastníky transakce (výrobce – dodavatel, obchodník – zákazník)
- Snižuje náklady a zrychluje transakce
- Flexibilita, distribuce procesů, znalostí, …

Proto většina moderních IS je navržena pro internetovou (intranetovou) architekturu, pro klieta je rozhodujícím a jediným programovým prostředkem internetovský prohlížeč.

e-Commerce – nákup a prodej výrobků a služeb po Internetu

Tři hlavní etapy vývoje:

1. užití interních zdrojů, snižování nákladů, zvyšování efektivity řídících a obchodních procesů - spojené s implementacemi ERP, …

2. interorganizační systémy založené na elektronické výměně dat mezi ekonomickými subjekty – EDI, typické jsou vztahy1:1

e-Business- využití Internetu pro provádění a podporu každodenních obchodních aktivit

3. vytváření společných elektronických obchodních komunit - typické jsou vztahy M:N

Aplikace

- elektronické podnikání (e-Business) v nejrůznějších formách, jako je elektronické zásobováni e- procurement),
- elektronická tržiště (e-Marketplace) a elektronický obchod (e-Commerce), řízení dodavatelských řetězců s vazbou na metody progresivního plánování a rozvrhování (SCM, APS)
- řízení vztahů se zákazníky (CRM - Customer relationship management) a jeho další modifikace. Zahrnuje rozšíření strategií, procesů a technplogií pro podporu vztahu podniku a zákazníka v celém obchodním cyklu – od marketingu, přes prodej až k dodavatelským a poštovním službám. Diky Internetu můžeme hovořit o elektronické verzi (eCRM).

Typy obchodních vztahů - B2x

B2B – business-to-business aplikace

Zkratka B2B v praxi realizuje výměnu informací mezi obchodními partnery v elektronickém formátu. Jako vhodný formát pro přenos dat se jeví právě jazyk XML, který je velice jednoduchý a podporovaný na mnoha počítačových platformách. Pomocí XML si firmy mohou vyměňovat objednávky, faktury a mnoho dalších údajů. To bylo podporováno již dříve v systémech EDI (Electronic Data Interchange). Datové formáty používané v EDI byly však dost složité a jejich implementace byla nákladná. Jednotlivé EDI systémy nebyly mezi sebou kompatibilní, a tak bylo často potřeba informační systém speciálně upravit pro každého dalšího obchodního partnera, se kterým jsme chtěli komunikovat elektronicky.

- výměna strukturovaných dat
  - Business-to-Business (B2B) - (objednávky, potvrzení, faktury,..),
  - Business-to-Reseller (B2R) -
  - Business-to-Government (B2G) - finanční úřady, pojištění, orgány místní správy, ...)
- webové aplikace
  - Business-to-Customer (B2C) -, virtuálními obchody na Internetu apod.
  - Consumer-to-consumer(C2C)electronic commerce- nabídka zboží a služeb mezi zákazníky

- Business-to-Employee (B2E) - interní webové aplikací, intranet

Mobile commerce (m-commerce) – využití mobilních bezdrátových zařízení – např. telefonů

Systém EDI - Electronic Data Interchange

Elektronická výměna dat

Cíle:

- Výměna dat s obchodními aj. partnery, zjednodušení a zrychlení postupů
- Strukturování dat na základě mezinárodních standardů

Nároky:

- Připojení IS do veřejných sítí a zajištění přístupů k jejich službám
- Sladění pravidel mezi partnery

Kritické faktory úspěchu:

- Výběr obchodních partnerů
- Tlak na potřebu těchto úloh (např. EDI jako nezbytná podmínka pro získání zajímavé dlouhodobé zakázky)

EDI - standardy a přínosy

- Standardy: UN/EDIFACT (mezinárodní), ANSI X12 (US, Kanada)
- XML/EDI (eXtensible Markup Language), XSL (eXtensible Stylesheet Language)

Přínosy:

- Eliminace chyb při vkládání dat
- Zrychlení přenosu dat, snížení nákladů na archivaci dokumentů
- Podpora procesů „Just-In-Time“
- Rozšíření okruhu zákazníků
- Posílení vztahů se stávajícími partnery

Základem technologie se stává jazyk XML (eXtensible Markup Language)

ebXML

- sada specifikací umožňující realizaci kompletních obchodních případů:
  - definice obchodních procesů a s nimi spojených zpráv,
  - registrace a vyvolávání sledu kroků obchodního procesu a s nimi spojenou výměnu zpráv
  - specifikace profilu obchodní společnosti,
  - specifikace dohod mezi obchodními partnery, jednotná transportní vrstva pro obchodní data a dokumenty.

e Commerce - služby

- inzerce a speciální nabídky zboží,
- elektronické katalogy nabízeného zboží a s nimi spojené vyhledávací služby (např. u knih podle autorů, názvů publikací, roku vydání apod.),
- presentace nabízeného zboží fotografiemi nebo různými vizualizačními prostředky, další specifikací všech potřebných parametrů a jejich variant (barva, úpravy povrchu,..), případně poskytování detailní dokumentace produktů,
- poskytování odpovědí na dotazy zákazníka a řešení jeho problémů cestou elektronické pošty, diskusních skupin nebo nabídkou odpovědí na tzv. časté dotazy (FAQ - frequent asked questions),
- výběr a uložení zboží podle nabídky do virtuálního nákupního košíku, případně rezervace různých služeb (ubytování, letenek apod.),
- platby elektronickou cestou pomocí kreditních karet, elektronických šeků, digitální hotovostí nebo tzv. mikroplatbami.

e Commerce - přínosy

- zvýšení výnosů nabídkou nových produktů a služeb - individualizované nabídky, automatická tvorba cen, elektronické služby dostupné 365 x 7 x 24,
- možnosti využití nových komunikačních kanálů a proniknutí na nové trhy,
- lepší a efektivnější kontakty se zákazníky,

- snížení nákladů - nákladů na prodej a marketing (distribučních a skladovacích nákladů, aplikacemi samoobslužného prodeje), režijních nákladů (snížení doby obchodních transakcí, vyhledávání dodavatelů),
- lepší využití zdrojů (vyšší obrátka zásob, snížení skladovacích a prodejních prostor),
- vyšší kvalitu obchodu - celkově kvalitnější služby zákazníkům, kvalitnější plánování poptávky, řízení objednávek, kvalitnější řízení celého dodavatelského řetězce, kvalitní podmínky pro působení v globálním, resp. celosvětovém měřítku,
- průzkumy spokojenosti zákazníků, shromažďování údajů pro další analytické operace, např. v rámci aplikací CRM, pro plánování výroby, pro řízení sortimentu apod.

e Commerce – příklady systémů:

- Prodej knih, CD, … www.amazon.com
- Prodej digitálních produktů – software, multimédia, …
- Nabídka služeb a podpory softwaru, hardwaru, …
- rezervace nejrůznějších služeb: ubytování v hotelích, rezervace a nákup jízdenek, vstupenek apod.
- kritická místa: provázání vlastního elektronického prodeje a navazujících logistických služeb

e Commerce - oblasti úspěšného nasazení systémů :

1. letenky, 2.cestovní služby, 3.PC software, 4. knihy, CD, videa, 5.vstupenky na různé kulturní, sportovní a společenské akce, 6. konfekce, 7.potraviny a nápoje, 8. investice, 9. automobily.

e-Procurement

- nalezení, zakoupení a dodání produktu od dodavatele do místa jeho využití u kupujícího.
- identifikaci a vyhodnocení dodavatelů,
- výběr specifických produktů
- podávání objednávek
- dodání produktu včetně platby
- řešení řady problémů, které mohou nastat po dodání produktu/služby, jako např. pozdní dodání, dodání nesprávného množství, nesprávného produktu
- monitorování průběhu zásobovací transakce, vytvoření a řízení vztahu s dodavateli.

e-tržiště

- V prostředí Internetu vytvářejí prostor pro uskutečňování mnohostranných elektronicky realizovaných obchodních transakcí ve vazbách M : N.
- Vytváří se tak virtuální obchodní komunita s vysoce optimalizovanými řídícími a obchodními procesy mezi širokou škálou obchodních partnerů.
- Pokrývají především oblasti prodeje, nákupu, skladů a marketingu
- Podporují realizaci obchodních transakcí aplikacemi a technologiemi Internetu.

Shrnutí

Mnohé technologie pro podporu ekonomických informačních systémů jsou součástí rozšíření objektově-relačních databázových systémů (norma SQL3) – např. XML databáze, modelování výrobních procesů (toky práce - work-flow systémy), dolování dat, … a také oblasti zpracované v této kapitole – datové sklady s rozhraním OLAP a systémy pro podporu e-Business. Datový sklad a systém OLAP je technologie pro analýzu dat. Po načtení vyčištěných a opravených, převážně statických historických dat do nového modelu uložení (schéma hvězda, vločka), optimalizovaného na efektivní dotazování (typicky velice složité analytické dotazy na vývoj ekonomických parametrů podniku, výroby, obchodu, … v čase) v nové logické formě dat – multidimenzionální kostky. Logické dimenze kostky představují popisné parametry- souřadnice jako čas, výrobek, lokalita, prodejce, … pro ekonomické informace ve formě faktů, jako např. zisk, cena, náklady, … . Rychlost zpracování je podporována strukturou uložení dat, redundací dat formou předzpracovaných mezivýsledků, uložených v databázi, na fyzické úrovni bohatou podporou indexových metod přístupu – bitmapové indexy. Sytémy pro podporu e-Business

aplikací využívají internetové technologie, databáze XML a nové protokoly na bázi XML pro rychlé, flexibilní aplikace pro obchodování, zásobování a podobně.

Pojmy k zapamatování

- OLTP , datový sklad, OLAP
- analýza, dolování a čištění dat, multidimenzionální databáze, modelování, kostka
- schéma hvězda, vločka, vícerozměrné pole, bitmapový index
- e-Business, e-Commerce, e-Marketplace, Systém EDI

Kontrolní otázky

20. Jaký je charakter dat vkládaných do datového skladu a jak se zpracovávají?
21. Jaká je architektura a model dat datového skladu a OLAP?
22. Co charakterizuje multidimenzionální modelování a dotazování?
23. Jaký je princip bitmapového indexu?
24. Jaká je charakteristika a kategorizace systémů pro podporu e-Business?

Úkoly k textu

6. Navrhněte jednoduché hvězdicové schéma ve formě ER diagramu, například v prostředí MS Access.
7. Nainstalujte si, pokud již nemáte, doplněk MS SQL serveru – Analysis services. Vytvořte a seznamte se s ukázkovými příklady z tutorialu. Vyzkoušejte MDX aplikaci a seznamte se s dotazovacím jazykem. Pokud máte podmínky pro jiné systémy (např. Oracle), seznamte se s tímto konkrétním prostředím pro analytické služby.

5 Implementace IS s databází

Studijní cíle: Student by měl po prostudování kapitoly zvládnout praktickou implementaci nejrůznějších technologií pro různé architektury – od klasické klientské až po webovské technologie ve kterých jsou data pro aplikaci uložena v databázových systémech.

Klíčová slova: ODBC, JDBC, ADO, ADO.NET, HTML, DHTML, JavaScript, http, XML, XPath, XLink, DTD, XML Schema, validace dokumentu, XMLDOM, kaskádové styly, XSL, XSLT, ASP, ASP.NET, WEB-service, WSDL, UDDI

Potřebný čas: 12 hodin

5.1 Přístup k datům

Implementace aplikací informačních systémů, podobně jako tvorba SW obecně, prošla vývojem, ve kterém hlavní trendy jsou odvozeny od změn architektury informačního systému, rozvojem technologií na různých platformách. Zároveň se prosazují požadavky na maximální otevřenost, flexibilitu, integrovanost, univerzálnost, unifikaci, interoperabilitu, distribuovanost a zároveň bezpečnost nových řešení. Problémy, související se zpřístupněním databázových dat:

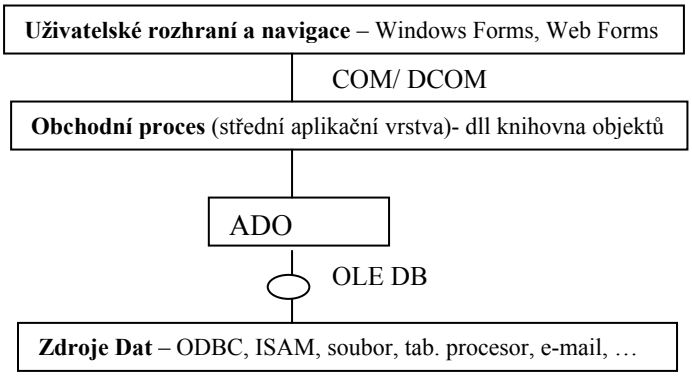
- Aplikace potřebují pracovat s několika databázovými produkty
- Uživatelský interface by měl zakrýt databázovou sémantiku
- Inicializace datových zdrojů je pomalá
- Podrobnosti přístupu k datům komplikují tvorbu a údržbu kódu
- Aplikace potřebují uchovávat (cachovat) data, ke kterým často přistupují
- Více uživatelů potřebuje přistupovat k datům souběžně
- Zajištění autentifikace uživatelů a bezpečnosti a integrity dat

Jedno kritérium kategorizace implementace IS určují použité technologie pro přístup k datům. Dalším hlediskem je případné využití počítačových sítí. Následující přístupy charakterizují důležité historické etapy vývoje typických aplikací:

- Desktop aplikace s použitím datových souborů (tabulkový procesor, textová data)
- Lokální databázová aplikace (FoxPro, Access, soubory dbf, aplikace jako kolekce formulářů, svázaných vazbou s daty v databázi)
- Síťová souborová aplikace (typicky v prostředí DOSu se síťovým přístupem k datovým souborům)
- Dvouvrstvá databázová architektura klient – server (databázoví klienti na síti sdílí data na databázovém serveru)
- Tří a vícevrstvá architektura (distribuované technologie, COM/DCOM, .NET Framework, Java,… slouží k vytváření střední vrstvy (objektů) business logiky, umístěné mezi prezentační a datovou vrstvu. Tato architektura je po rozšíření o vrstvu web-serveru hojně využívaná pro aplikace na Internetu). Vnitřní vrstvou jsou i aplikační servery, což je široká skupina produktů pro tvorbu rozsáhlejších aplikací. Typicky nabízí mimo jiné technologickou podporu sdílení přístupu k databázi (connection pooling), řízení přístupu k aplikaci a k vyrovnávací paměti, podporuje uživatelské relace (session state management)

a obecnější transakční zpracování. (využívané implementace jsou například J2EE a ASP.NET)

Příklad obvyklé architektury firmy Microsoft pro použití technologie COM, DCOM:



Obrázek 12 Třívrstvá architektura s technologií COM/DCOM

Protokoly pro komunikaci s db můžeme rozdělit na:

nativní – každá aplikace má svůj (atypický) protokol, lze plně využít všechny funkce databázového systému, ale přenos aplikace na jiný databázový server je komplikovaný

standardizovaná rozhraní – ODBC, JDBC, … je přidána vrstva navíc, která odstiňuje nativní

protokol, při změně databázového serveru stačí změnit ovladač, pro předávní příkazů se používá jazyk SQL.

5.1.1 Technologie firmy Microsoft pro přístup k datům

V průběhu vývoje softwaru, operačních systémů Windows a aplikačních technologií bylo vyvinuto mnoho technologií pro přístup k datovým zdrojům. Cílem je dosažení interoperability (Windows DNA - Distributed interNet Applications Architecture) v heterogenním prostředí různých datových zdrojů na různých platformách. Namátkou vyberme ty nejpoužívanější v pořadí, jak se v čase prosazovaly do implementací IS (nejprve využívající technologie COM, DCOM, později rozšířeno na .NET Framework) :

ODBC (Open DataBase Connectivity) - základní technologie pro přístup k datovým zdrojům. Původně navržena pro přístup k relačním databázím na bázi SQL, ale později byla rozšířena i o ISAM databáze (např. Access, DBF, …). Efektivně je řešen problém různých nativních přístustupů k implementaci databázových operací v různých databázových systémech. Systém rozhraní ve formě skupiny funkcí (na poměrně „nízké“ – fyzické úrovni) realizuje standardní způsob komunikace ze strany aplikace, různí výrobci relačních databázových systémů poskytnou ze strany datového zdroje vhodný driver.

DAO (Data Access objects) – jako součást MS Accessu umožňovala přístup i k jiným ISAM a SQL zdrojům za použití ODBC (s označením MS Jet).

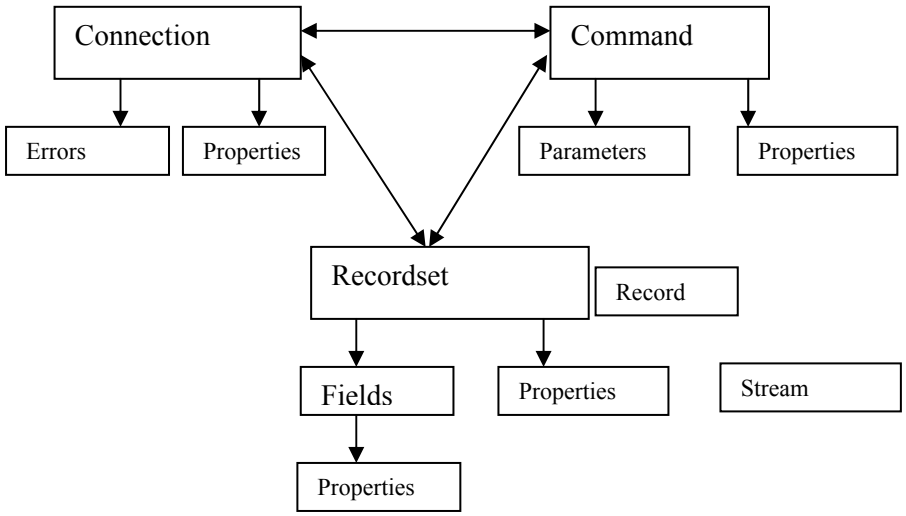
RDO (Remote Data Objects) – nahrazuje DAO . Představuje jen tenkou vrstvu nad ODBC. Součástí knihovny je přístup k datovým zdrojům přes Internet pomocí komponenty ADC (Advanced Data Connector).

OLE DB – Přechod na otevřenou univerzální specifikaci (skupina COM objektů s jednodušším a pohodlněji zvládnutelným rozhraním) s možností přístupu k relačním databázím i ostatním zdrojům, jako mainframe ISAM/VSAM, e-mail, souborový systém, text, …. Nové možnosti a rozšíření - **OLE DB provider** (implementuje kolekci rozhraní pro zajištění základní i rozšířené funkcionality standardním způsobem – ODBC provider, simple provider, Orecle provider, …). **OLE DB consumer** je různorodý software, přistupující k heterogenním datovým zdrojům – nejtypičtějším představitelem je ADO.

ADO (ActiveX Data Object) – základní rozhraní pro přístup k datům přes OLE DB. Technologie COM nabízí využití pro fron-end databázové klienty, vícevrstvé aplikace i na internetu, i ve vysokoúrovňových jazycích, jako jsou například různé skriptové jazyky.

MDAC (Microsoft Data Access Components) - soubor komponent zahrnující oblast UDA (Universal Data Access) – s cílem poskytnout objektově orientované rozhraní nad ODBC nebo OLE DB. Obsahuje sadu OLE DB data providerů pro databáze i pro ostatní podporované datové zdroje – např. Directory Services (ADSI), Index Server, Site Server Search a Internet Publishing (FrontPage Server Extensions a Distributed Authoring and Versioning – DAV), komponenty ADODB, ADOR (obsahuje samostatný recordset), RDA (Remote Data Services), ADOX – pro operace s jazykem DDL a zabezpečením, ADO MD (ADO Multidimensional – přístup k OLAP) a další.

Pro ilustraci vývoje, také pro možnost nutnosti úprav softwaru dřívějších verzí vývojových nástrojů, se seznamíme se základní logickou strukturou objektů v technologii ADO.



Obrázek 13 Zjednodušený objektový model ADO

Objektový model ADO tvoří tři hlavní objekty - Connection, Command a Recordset. Tyto základní objekty spolupracují s pomocnými objekty (nebo jejich kolekcemi) - Error, Field, Parameter a Property. Objekt Connection zodpovídá za spojení s datovým zdrojem a určuje základní parametry tohoto spojení. To se děje prostřednictvím property ConnectString. Součástí řetězce je variantně OLE DB provider , název databáze, uživatelské jméno, heslo, atd.. Tvar connect stringu má několik variant a ty je možno nalézt v dokumentaci. Jednou z možností je

použití – Data Link File. To je speciální soubor s příponou .udl, který obsahuje všechny požadované parametry, které je možné definovat až za běhu aplikace.

Objekt Command slouží pro provádění SQL příkazů a uložených procedur metodou Execute, ale není jedinou možností. Jednoduché SQL příkazy je možné provádět metodou Execute objektu Connection, nebo metodou Open objektu Recordset. Mezi důležité property objektu Command patří ActiveConnection - obsahuje odkaz na patřičný objekt Connection, CommandText – obsahuje text SQL příkazu nebo volání procedury a CommandType, který specifikuje typ příkazu. Může to být SQL příkaz, volání stored procedury nebo příkaz typu DirectTable, kde CommandText potom obsahuje jméno tabulky a výsledkem dotazu je obsah tabulky.

Objekt Recordset zachycuje data, která jsou výsledkem dotazu. Mezi nejdůležitější metody patří metoda Open, ktera otevře Recordset, s parametry obsahující text příkazu, odkaz na objekt Command nebo Connection a nastavení typu kurzoru a zámků v tabulce - ReadOnly, Pessimistic, Optimistic a BatchOptimistic. Další metodou je Move , která umožňuje pohyb mezi jednotlivými řádky recordsetu - Move, MoveFirst, MovePrev, MoveNext, MoveLast. Metoda Close zavře objekt Recordset a uvolní načtená data. Pro provádění operací s daty v databázi se dají využít SQL příkazy DELETE, INSERT, UPDATE, nebo je možné využít editačních možností objektu Recordset. K tomu slouží metodyDelete (zruší aktuální řádek v otevřeném Recordsetu), AddNew (přidá do otevřeného recordsetu prázdný řádek, nastaví jej jako aktuální. Nové hodnoty se definují pomocí objektů Field z kolekce Fields) a Update (zapíše změny do tabulky). Pro dotazování můžeme použít SQL příkaz SELECT, nebo metody objektu Recordset Seek a Find pro vyhledání řádku podle hodnot některých sloupců. Metoda Save umožňuje uložení Recordsetu na disk. Další property objektu jsou např. ActiveConnection (odkaz na případně použitý objekt Connection), ActiveCommand (odkaz na případně použitý objekt Command), Bookmark – pro vytváření záložek, umožňující návrat kurzoru na označený řádek. Počet řádek Recordsetu určuje RecordCount. Filter vybere požadovaná data z obsahu Recordsetu. Sort provede třídění dat v Recordsetu podle zadaného sloupce.

Objekty Field obsahuje kolekce Fields. Tato kolekce reprezentuje záhlaví dat, uložených v rekordsetu. Objekt Field dodává jednotlivá sloupcová data dotazu pro čtení i aktualizaci na aktuálním řádku. Jejich počet odpovídá počtu sloupců vrácených příkazem SELECT. V kolekci je možné použít jako klíč property Name objektu Field (logické jméno sloupce zdrojové tabulky nebo index. Pproperty Value obsahuje hodnotu příslušného sloupce v aktuálním řádku. S daty typu BLOB (Memo, long binary, ...) pracují metody GetChunk pro získání příslušné hodnoty z databáze a AppendChunk pro zápis binárních dat do databáze.

Ukázka vybraných částí kódu:

```
Dim Conn1 As New ADODB.Connection
Dim Cmd1 As New ADODB.Command
Dim Rs1 As New ADODB.Recordset
…
Conn1.Open "DRIVER={sql server};" & _
                      "server=csnt\kmi" & ";" & _
                      "Database=Northwind;" & _
                      "PWD=;UID=;"
'            jiný zdroj dat - databáze Access
' Conn1.Open "DRIVER={Microsoft Access Driver (*.mdb)};" & _
'                  "DBQ=databáze1.mdb;" & _
'                   "DefaultDir=.;" & _
'                "UID=admin;PWD=;"
Cmd1.ActiveConnection = Conn1
Cmd1.CommandText = "SELECT * FROM  Employees"
Set Rs1 = Cmd1.Execute()
If Rs1.State = 1 Then
   Dim ss As String
   Dim j As Integer
   While Rs1.EOF = False
       ss = ""
       For j = 0 To Rs1.Fields.Count - 1
```

```
            ss = ss & Rs1.Fields(j).Value & " | " & vbTab
        Next
        List1.AddItem ss
        Rs1.MoveNext
    Wend
    …
```

Struktura ADO.NET

ADO.NET je nový koncept Mikrosoftu pro univerzální přístup k datům v prostředí .NET Framework. V některých aspektech navazuje na předchozí technologie ADO, v jiných přináší výrazné inovace. Je to reakce na potřebu efektivně pracovat se semi-strukturovanými daty (XML) a dalšími heterogenními datovými zdroji v prostředí objektově orientovaných aplikací, operujících nad více jazyky. ADO.NET lépe podporuje:

- práci s daty i mimo vlastní datový zdroj (odpojitelné od datových zdrojů) formou pullingu, což výrazně podporuje škálovatelnost aplikací (hlavně na Internetu) – redukuje množství aktivních připojení.
- Stabilitu a výkon při rozsáhlém simultánním dotazování v připojené databázi
- Možnost prezentovat data v mnoha prvcích rozhraní (data binding)

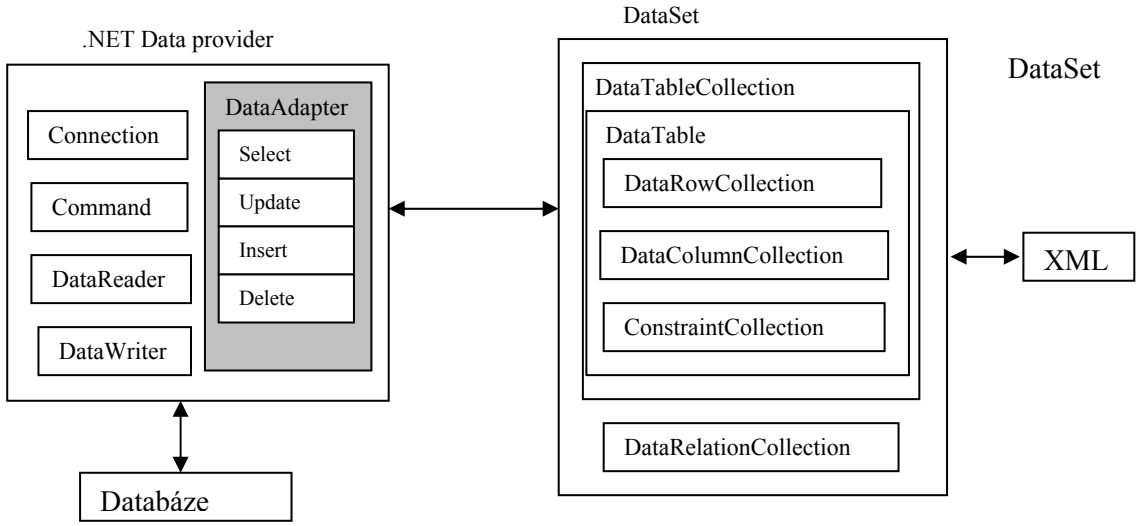
Je to jednoduchý objektový model pro ovládání připojení aplikace ke zdroji dat, provádění příkazů pro definici a manipulaci s daty. Pro připojení ze strany datového zdroje zůstává klíčová úloha na *data providerech*. Charakteristické je maximální využívání technologie XML, která slouží jak pro reprezentaci uložených a přenášených dat, tak pro popis struktury (metadata) a operace. Ve struktuře vychází ADO. NET částečně z předcházejícího modelu ADO.

Příklady data providerů:

MySQL - ODBC

Microsoft Access - OLE DB

SQL Server – ODBC, OLE DB



Obrázek 14 Zjednodušený model ADO. NET

ADO.NET obsahuje dva základní typy objektů:

1. Objekty založené a zaměřené na připojení k datovému zdroji. Logickou skupinu tvoří objekty jako Connection, Command, DataAdapter, DataReader. Connection zajišťuje připojení ke zdroji dat, Command umožňuje formulovat a vykonat SQL příkaz (např. metodami .ExecuteNonQuery() pro příkazy bez dotazů – bez odezvy ve formě požadovaných dat, nebo .ExecuteReader() pro skupiny příkazů, obsahující i příkaz SELECT, nebo provedení adekvátní uložené procedury). Adapter definuje a provádí operace manipulující s daty (SQL příkazy INSERT, UPDATE, DELETE a SELECT) pro jednu tabulku. Dotazy mohou být i parametrizované s pomocí kolekce Parameters a jejích členů objektů Parameter. Hlavním parametrem pro vytvoření spojení s datovým zdrojem je vlastnost obsahující Connection String. Ten může mít různou formu podle

Nastavení Connection Pooling:

druhu zdroje a verze knihovny. Obsahuje informace o lokalizaci datového zdroje, databázi, autentifikaci uživatele, použitelném driveru a podobně. Protože spravovat datové zdroje operačním systémem (otevírat a zavírat spojení) je relativně časově náročné, používá se řešení s technologií *connection pooling,* která umožňuje sdílet permanentní množinu databázových připojení několika běžícím aplikacím, nebo sezením (sessions). ADO.NET však přímo tuto technologii nepodporuje.

2. Objekty zaměřené na uchování obsahu a struktury dat. Logickou skupinu tvoří objekty jako DataSet, DataColumn, DataRow, DataRelation a další. DataSet reprezentuje obecně celé sub-schéma struktury dat. Obsahuje tabulky i vazby mezi nimi, často původně z rozdílných datových zdrojů. Data v těchto tabulkách jsou redukována proti zdrojovým na taková, která uživatel aktuálně zpracovává a využívá. Další objekty umožňují efektivní práci s řádky a sloupci relačních tabulek, případně reprezentují vztahy mezi tabulkami.

Pro praktické použití např. ve vývojových nástrojích Visual Studia je třeba znát jednotlivé jmenné prostory ADO.NET, které obsahují skupiny datových a databázových objektů:

System.Data – obsahuje třídy datových kontejnerů pro modelování a operace s tabulkami, pohledy, sloupci, řádky, vazbami a integritními omezeními.

System.Data.Common – obsahuje abstraktní třídy, implementující jádro funkcionality ADO.NET.

System.Data.OleDb - obsahuje třídy pro připojení k různým OLE DB providerům.

System.Data.SqlClient - obsahuje třídy pro práci s Microsoft SQL Server databází.

System.Data.OracleClient - obsahuje třídy pro práci s databází Oracle (OracleCommand, OracleConnection, OracleDataAdapter), používá optimalizovaný Oracle Call Interface (OCI).

System.Data.Odbc - obsahuje třídy pro připojení k různým ODBC driverům

System.Data.SqlTypes - obsahuje třídy pro práci s nativními datovými typy na SQL Serveru

Kromě databázových a obecnějších datových zdrojů (OleDb) je hlavním formátem pro definici struktury i obsahu dat v Datasetu formát XML. Přístup k požadovaným datovým položkám Datasetu je umožněn prostřednictvím kolekcí (Tables, Rows, …).
Např. :

```
…     dataSet1.Tables.Item(0). Rows(0).Item(0).ToString
```

Jinou variantou je použití objektu DataRow jako člena kolekce Rows.

Připojení ke zdroji dat není souvislé, nepřetržité. Objekt Data connection disponuje metodami .Open() a .Close() pro explicitní otevření a ukončení připojení. Při požívání objektů Command nebo Data adapter nemusíme explicitně otevřít, nebo zavřít připojení. Při provádění operace se testuje připojení a přídadně se aktuálně vytvoří a po operaci s databází ukončí. Původně otevřené spojení se však nezavře. Objekt Data connection podporuje metodou .BeginTransaction vytvoření transakčního objektu a s jeho pomocí umožňuje správu transakcí aplikace.

Data ve formě datasetů jsou udržována ve vyrovnávací paměti (cach). Jsou uložena a často přenášena (z komponenty do komponenty) a zpracovávaná ve formátu XML (soubor, proud) a tím se stávají nezávislá na datovém zdroji, případně softwaru, platformě.

Další variantu přístupu k datům (přímý přístup) podporuje objekt DataReader, který řeší situaci, kdy očekávaná data by v datasetu – lokální paměti zabrala mnoho prostoru. DataReader

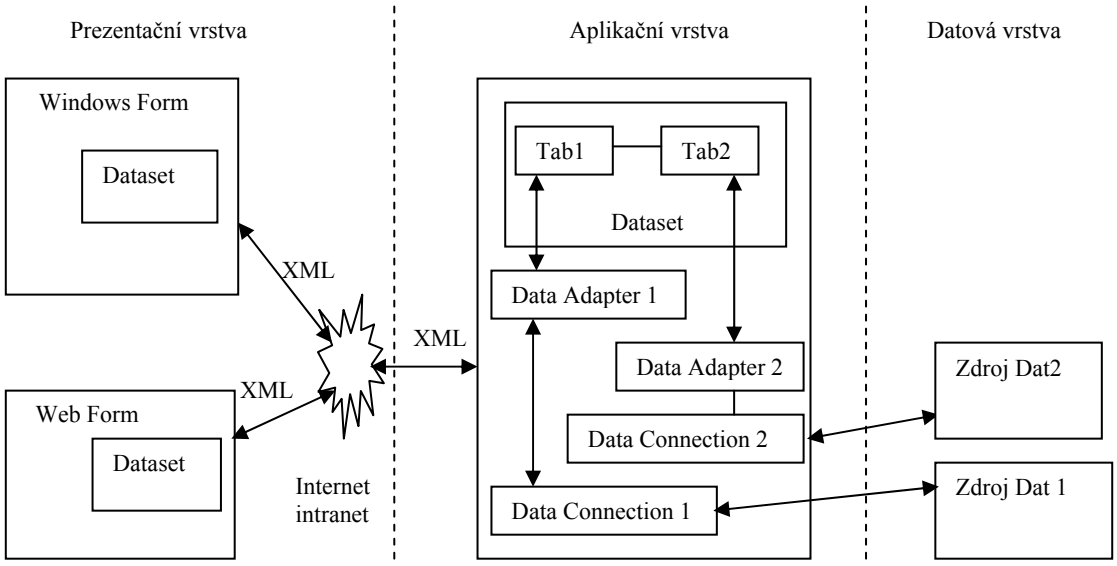
poskytuje aplikaci proud nemodifikovatelných záznamů, čtených ze zdroje jeden za druhým směrem jen od prvního k poslednímu. Funkčnost zajišťuje metoda .GetString(i), kde i = 0,1,…představuje index pole aktuálního záznamu.

Příklad části kódu přístupu k datům pomocí objektu Datareader:

```
Imports System.Data.OleDb
…
Dim connString As String = _
   "Provider= SQLOLEDB.1;Data Source=CSNT\KMI;" & _
   "uid=webstudent;pwd=webstudent;Initial Catalog=Northwind;"
Dim myConn As New oleDbConnection
myConn.ConnectionString = connString
myConn.Open()
Dim sqlStat As String = "SELECT * FROM Products where ProductName like
'%" _
      & Me.TextBox1.Text.Trim & "%' "
Dim myComm As OleDbCommand = New OleDbCommand(sqlStat, myConn)
Dim myReader As OleDbDataReader = Nothing
myReader = myComm.ExecuteReader()
Me.ListBox1.Items.Clear()
While myReader.Read
      Me.ListBox1.Items.Add(myReader.GetValue(0) & "   " _
            & myReader.GetString(1) & "   " & myReader.GetValue(5))
 End While
 myConn.Close()
…
```

Podobně se dá využít objekt DataWriter pro případ potřeby vykonat SQL příkaz (nevracející data jako INSERT, …), definovaný v aplikaci.

Typická vícevrstvá distribuovaná aplikace v prostředí Internetu je znázorněna na následujícím obrázku. Datovou a aplikační vrstvu implementuje webová služba. Prezentaci zajišťují klienti ve Windows nebo na webu.



Obrázek 15 Architektura aplikace s ADO. NET

Jednoduchý příklad zdrojového kódu webové stránky WebForm1.aspx.cs pro využití webové služby, jajíž zdrojový kód je v kapitole o webových službách:

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
```

```
using System.Drawing;
using System.Web;
using System.Web.SessionState;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.HtmlControls;

namespace AuthorsWebClient
{
	public class WebForm1 : System.Web.UI.Page
	{
		protected AuthorsWebClient.localhost.authors1 AuthorData;
		protected System.Web.UI.WebControls.DataGrid DataGrid1;

		private void Page_Load(object sender, System.EventArgs e)
		{
			// Put user code to initialize the page here
			AuthorsWebClient.localhost.AuthorsService ws =
				new AuthorsWebClient.localhost.AuthorsService();
			ws.Credentials =
System.Net.CredentialCache.DefaultCredentials;
			AuthorData.Merge(ws.GetAuthors());
			if (! Page.IsPostBack)
			{
				DataGrid1.DataBind();
			}

		}
		#region Web Form Designer generated code
		…
		#endregion
		private void DataGrid1_EditCommand(object source,
System.Web.UI.WebControls.DataGridCommandEventArgs e)
		{
			DataGrid1.EditItemIndex = e.Item.ItemIndex;
			DataGrid1.DataBind();
		}
		private void DataGrid1_CancelCommand(object source,
System.Web.UI.WebControls.DataGridCommandEventArgs e)
		{
			DataGrid1.EditItemIndex = -1;
			DataGrid1.DataBind();
		}
		private void DataGrid1_UpdateCommand(object source,
System.Web.UI.WebControls.DataGridCommandEventArgs e)
		{
			// Change the data in the dataset.
			for (int i=1; i <= AuthorData.authors.Columns.Count; i++)
			{
				TextBox t = (TextBox)(e.Item.Cells[i].Controls[0]);
				DataRow row =
AuthorData.authors[e.Item.DataSetIndex];
				row[AuthorData.authors.Columns[i-1].Caption] =
t.Text;
			}

			// Update the database.
			if (AuthorData.HasChanges())
			{
				AuthorsWebClient.localhost.AuthorsService ws =
					new
AuthorsWebClient.localhost.AuthorsService();
				ws.Credentials =
System.Net.CredentialCache.DefaultCredentials;
				AuthorsWebClient.localhost.authors1 diffAuthors =
					new AuthorsWebClient.localhost.authors1();
				diffAuthors.Merge(AuthorData.GetChanges());
				ws.UpdateAuthors(diffAuthors);
```

```
                        AuthorData.Merge(diffAuthors);
                    }
                    DataGrid1.EditItemIndex = -1;
                    DataGrid1.DataBind();

                }
        }
}
```

Verze pro klienta ve Windows, soubor Form1.cs :

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;

namespace AuthorsWinClient
{
        public class Form1 : System.Windows.Forms.Form
        {
                private System.Windows.Forms.DataGrid dataGrid1;
                private System.Windows.Forms.Button LoadData;
                private System.Windows.Forms.Button SaveData;
                private AuthorsWinClient.localhost.authors1 AuthorData;
                private System.ComponentModel.Container components = null;

                public Form1()
                {
                        InitializeComponent();
                }

                …

                [STAThread]
                static void Main()
                {
                        Application.Run(new Form1());
                }
                private void LoadData_Click(object sender, System.EventArgs e)
                {
                        AuthorsWinClient.localhost.AuthorsService ws =
                                new AuthorsWinClient.localhost.AuthorsService();
                        ws.Credentials =
System.Net.CredentialCache.DefaultCredentials;
                        AuthorData.Merge(ws.GetAuthors());
                }
                private void SaveData_Click(object sender, System.EventArgs e)
                {
                        if (AuthorData.HasChanges())
                        {
                                AuthorsWinClient.localhost.AuthorsService ws =
                                        new
AuthorsWinClient.localhost.AuthorsService();
                                ws.Credentials =
System.Net.CredentialCache.DefaultCredentials;
                                AuthorsWinClient.localhost.authors1 diffAuthors
                                        = new AuthorsWinClient.localhost.authors1();
                                diffAuthors.Merge(AuthorData.GetChanges());
                                diffAuthors = ws.UpdateAuthors(diffAuthors);
                                AuthorData.Merge(diffAuthors);
                        }

                }
        }
}
```

Jinou používanou variantou je použití knihovny .dll na místo webové služby, soubor Autors.cs:

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
namespace AuthorsClassLibrary
{
	public class AuthorsClass
	{
		private System.Data.SqlClient.SqlCommand sqlSelectCommand1;
		private System.Data.SqlClient.SqlCommand sqlInsertCommand1;
		private System.Data.SqlClient.SqlCommand sqlUpdateCommand1;
		private System.Data.SqlClient.SqlCommand sqlDeleteCommand1;
		private System.Data.SqlClient.SqlConnection sqlConnection1;
		private System.Data.SqlClient.SqlDataAdapter sqlDataAdapter1;
		public DataSet GetAuthors()
		{
			DataSet authors = new DataSet();
			sqlDataAdapter1.Fill(authors);
			return authors;
		}
			public DataSet UpdateAuthors(DataSet authorChanges)
		{
			if (authorChanges != null)
			{
				sqlDataAdapter1.Update(authorChanges);
				return authorChanges;
			}
			else
			{
				return null;
			}
		}
		public AuthorsClass()
		{
			this.sqlSelectCommand1 = new
System.Data.SqlClient.SqlCommand();

			…
	}
}
```

A odpovídající klient v prostředí windows:

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;

namespace AuthorsClassWinClient
{
	public class Form1 : System.Windows.Forms.Form
	{
		private System.Windows.Forms.Button button1;
		private System.Windows.Forms.DataGrid dataGrid1;
		private System.Windows.Forms.Button button2;
		private System.ComponentModel.Container components = null;
		private DataSet  AuthorData = new DataSet();
		public Form1()
		{
			InitializeComponent();
		}
		…
```

```
            [STAThread]
            static void Main()
            {
                  Application.Run(new Form1());
            }

            private void button1_Click(object sender, System.EventArgs e)
            {
                   AuthorsClassLibrary.AuthorsClass ws = new
AuthorsClassLibrary.AuthorsClass();
                  AuthorData.Merge(ws.GetAuthors());

                  this.dataGrid1.DataSource = AuthorData.Tables[0];

            }

            private void button2_Click(object sender, System.EventArgs e)
            {
                  if (AuthorData.HasChanges())
                  {
                        AuthorsClassLibrary.AuthorsClass ws = new
AuthorsClassLibrary.AuthorsClass();


                        DataSet diffAuthors = new DataSet();
                        diffAuthors.Merge(AuthorData.GetChanges());
                        diffAuthors = ws.UpdateAuthors(diffAuthors);
                        AuthorData.Merge(diffAuthors);
                  }
            }
      }
}
```

5.1.2 Postup při vytváření klientské aplikace v prostředí MS Visual Studio 2005

Novější verze MS Visual studio 2005 navazuje na předchozí a dále rozšiřuje možnosti při práci s databázovými daty. Podrobnější popis je např. v dokumentaci, přístupné v Microsoft Visual Studio help systém. Tam ve složce Class Library Reference (přes .Net Development - .Net Framework SDK -) jsou popsány podrobně objekty určené pro práci s databázemi v různém kontextu, tedy konkrétně knihovny Systém.Data. se specializací jmenných prostorů např. System.Data.SqlClient, System.Data.OleDb, System.Data.Odbc, System.Data.OracleClient a s nimi spolupracujícími prezentačními komponentami ve jmenných prostorech System.Windows.Forms pro klientské aplikace ve windows, nebo System.Web.UI.WebControls pro aplikace na webu. Pro ilustraci a snadnější úvod do praktického zvládnutí technologie nabízím relativně podrobný popis postupu při vytváření jednoduché prototypové aplikace. Záměrem je ukázat řešení vybraných základních kroků i s různými variantami. Postup není samozřejmě dogma, ale komentovaná inspirace.

Klientská aplikace v prostředí MS Visual Studio 2005

Prototypový formulář 1:

1. Otevřeme projekt typu WindowsApplication.
2. Pokud nemáme ve vývojovém prostředí, v panelu nástrojů všechny potřebné komponenty pro uvažovanou technologii, otevřeme z nabídky Tools – Chaose Toolbox Items, tedy zřejmě hlavně ty, které mají předponu Data-, OleDb-, Odbc-, Oracle-, Sql- jako např. SqlDataAdapter, OleDbConnection, OleDbDataAdapter, … . Připojím pouze informaci o tom že skupina objektů s předponou SQL je určena hladně pro produkty Microsoft, speciálně MS SQL Server a má zajišťovat nadstandardní výkon a služby. Předpona OleDb je potom univerzálně použitelná pro všechny zdroje (ne čistě databázové) podporující dané rozhraní.

3. Nejprve vytvoříme prototypovou stránku pro manipulaci s daty jedné databázové tabulky s pomocí objektů a technologických postupů, převážně známých již z verze VS 2003. Za zdroj databázových dat zvolíme například databázi MS Access. Pro úspěšnou realizaci spolupráce zdroje dat s aplikací je nutno dodržet běžný standard návrhu databázového schématu – definovat primární, případně cizí klíče, připravit logické pohledy – view (v Accessu jsou to pojmenované dotazy), nastavit oprávnění a podobně. Na KI v Olomouci je v adresáři …Vyuka\SQLPLUS\access prototypová databáze podnik1.mdb, kterou je možno použít. V nástrojích najdeme komponentu OleDbDataAdapter a přeneseme ji na formulář.

4. Zpustí se Configuration wizart. Pokud jsme s danou databází ještě nepracovali, zvolíme new connection a definujeme typ databázového zdroje – Access, cestu a jméno souboru .mdb. Dále ponecháme implicitní volby (use SQL statements), dále zvolíme Query Binder a vybereme databázový objekt, třeba tabulku pracovník. Definujeme selekční dotaz SQL a dokončíme návrh. Objeví se objekt OleDbConnection1, který zajišťuje připojení databáze do aplikace.

5. Dále vytvoříme objekt DataSet, např. stiskneme pravé tlačítko myši nad objektem OleDbDataAdapter1 a vybereme GenerateDataset. Ponecháme implicitní volby a vytvoříme nový objekt.

6. Na formulář dáme z nástrojů silnou komponentu pro prezentaci – DataGridView. Vybereme Data Source –(Other Data Sources- Project Data Source – DataSet..- Pracovník {tabulka})

7. Pro naplnění DataGridu je nutno, aby v kódu, např v události `Form1_Load` byl příkaz – v prostředí VB.NET:

```
Me.OleDbDataAdapter1.Fill(Me.DataSet11)
```

8. Přidáme podobně další adaptér – pro zajištění dat do následně doplněného ComboBoxu s dotazem

```
… SELECT DISTINCT [Číslo_místnosti] FROM místnost
```

9. Pokud chceme výběr dat podle parametru v textovém boxu, použijeme adaptéru a jeho vlastností pro definici SQL příkazů pro manipulace s daty, např.(každá další data v datasetu – tabulky, pohledy, dotazy mají další DataAdapter) – kód události tlačítka :

```
Me.OleDbDataAdapter1.SelectCommand.CommandText = "SELECT IDpra, Příjmení,
Jméno, [Číslo_místnosti] FROM pracovník wHERE (Příjmení LIKE '%" &
Me.TextBox1.Text.Trim & "%')"
        Me.DataSet11.Clear()
        Me.OleDbDataAdapter1.Fill(Me.DataSet11)
        Me.OleDbDataAdapter2.Fill(Me.DataSet11)
```

10. Po editaci dat se použije metoda update adapteru, např v události tlačítka:

```
Me.OleDbDataAdapter1.Update(Me.DataSet11)
```

11. DataGridView je možno modifikovat – změnit textové okno na combobox. Otevřeme vlastnost columns, najedeme na číslo_místnosti a změníme ColumnType

naDatGridComboColumn. Data Source nastavíme na vhodný zdroj (DataSet1-Místnost), Display Member na číslo_místnosti.

Prototypový formulář 2:

Jiným, jednoduchým prototypovým návrhem, je následující variantní postup:

1. Pokud již neexistuje vhodná nabídka ve složce Data Sources, vytvoříme novou položku ve formě DataSetu s požadovanou strukturou databázových tabulek. Z nabídky menu Data vybereme položku Add New Data Source. Necháme se vést průvodcem Data Source Configuration Wizard. Vytvoříme případně nové, nebo použijeme existující spojení s dostupným databázovým zdrojem (v našem případě podnik1.mdb). Vybereme uvažovanou množinu tabulek, nebo pohledů (dotazů).
2. Popíšeme opět jednoduchý příklad – vazbu 1: N mezi tabulkami místnost a pracovník. Ve vytvořeném datovám zdroji rozvineme položku s DataSetem - např. podnik1DataSet do úrovně místnost.
3. Cílem je procházet jednotlivé (později vybrané) místnosti a prezentovat množinu pracovníků, kteří v místnosti trvale pracují. Máme možnost vybrat ze dvou připravených návrhů prezentace. Volbu provedeme otevřením ComboBoxu, přidruženém ke jménu tabulky. Pro místnost (kardinalita 1) zvolíme položku Details (dvojice popisek – textové okno pro jednotlivé atributy tabulky). Po volbě přetáhneme položku místnost na připravený formulář. Vygenerují se potřebné objekty pro komunikaci s databází i prezentační komponenty – základní ovládací lišta pro ovládání datových operací, … .
4. Aby se uplatnila uvažovaná vazba, musíme použít položku pracovník, která je vnořena ve zdroji dat pod položkou místnost. Pokud bychom použili položku pracovník na stejné úrovni jako místnost, tabulka pracovník by byla nezávislá na tabulce místnost. Pro položku pracovník (kardinalita N) tentokrát zvolíme položku DataGridView (prezentace n – položek tabulkou). Po volbě přetáhneme položku pracovník opět na formulář.
5. Ověříme funkčnost zpuštěním aplikace. Projdeme jednotlivé místnosti (pomocí komponenty místnostBindingNavigator) a vidíme skupiny pracovníků z místností.
6. Přidáme ještě možnost výběru určitých místností. Pravým tlačítkem nad objektem příslušného adaptéru(místnostTableAdapter) vyvoláme menu a zvolíme položku Add Query. Pomocí Query Builderu zformulujeme vhodný SQL dotaz, např.

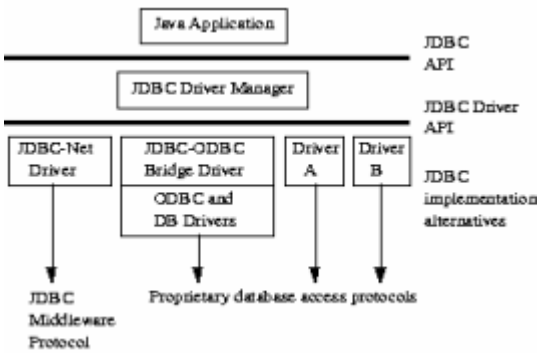
```
SELECT     [Číslo_místnosti], Plocha
FROM         místnost
WHERE     ([Číslo_místnosti] LIKE ?)
```

   , kde každý ? představuje jeden parametr dotazu. Můžeme dále definovat text v tlačítku, které provedení dotazu zpouští.
7. Generátor kódu vytvoří prototypovou komponentu ToolStrip, kde pro každý parametr dotazu je implicitně vytvořeno textové okno. Návrh je později možno upravit (např. nahradit TextBox za ComboBox a upravit příslušnou obslužnou proceduru tlačítka – konkrétně metodu patřičného objektu …Adapter.Fill…(…))

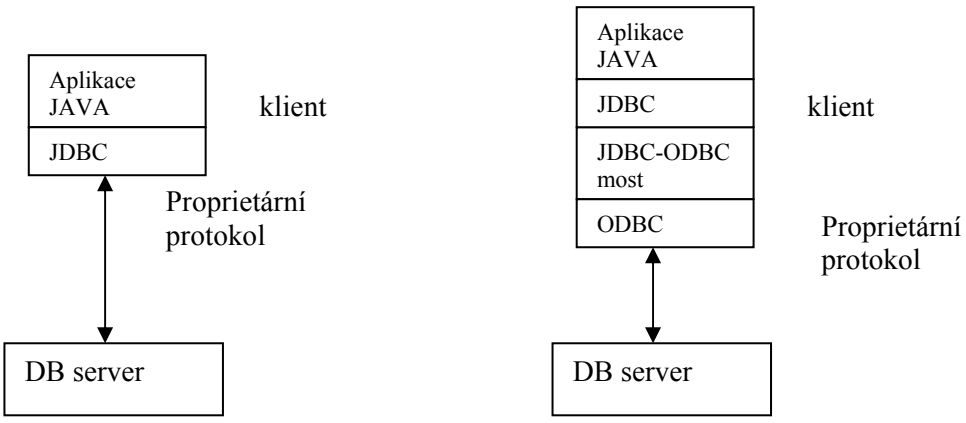
5.1.3 Technologie Java pro přístup k datům

Programovací jazyk Java patří k nejrozšířenějším a nejuniverzálnějším prostředkem pro psaní programů nezávislých na konkrétní platformě a typu klientské nebo webové aplikace. Proto je také dobře integrován i s webovými službami. Do HTML dokumentu je možno vložit Java applet, což je objekt vytvořený v jazyce Java, který webový prohlížeč zpracuje podobně jako třeba obrázek v HTML dokumentu. Applet ve formě byte-code sekvence je v prohlížeči interpretován pomocí JVM. Další technologie, vycházející z jazyka Java, jsou např. JSP(analogie ASP – skriptovací jazyk je založen na jazyku Java a komponentách - JavaBeans, Enterprise JavaBeans) a Servlet(Servlet je program napsaný v jazyce Java, který běží na serveru, a který dokáže zpracovávat požadavky zaslané pomocí HTTP protokolu a také na ně HTTP protokolem odpovídá) – viz další text.

Java applet nebo Java aplikace komunikují s databází pomocí JDBC API, což je čisté řešení v prostředí jazyka Java. Použití standardu Mikrosoftu - ODBC je možné za pomoci mostu JDBC-ODBC, ale není příliš výhodné, protože ODBC má rozhraní implementované v jazyce C, což přináší zpomalení (např. Java nepoužívá jako proměnné ukazatele, …). Funkčně je JDBC 2.0 API kompatibilní s MS UDA (Universal Data Access - OLE DB, ADO, RDS), tedy například podporuje SQL3. Následující obrázek, převzatý z dokumentace, ukazuje možnosti implementace JDBC.

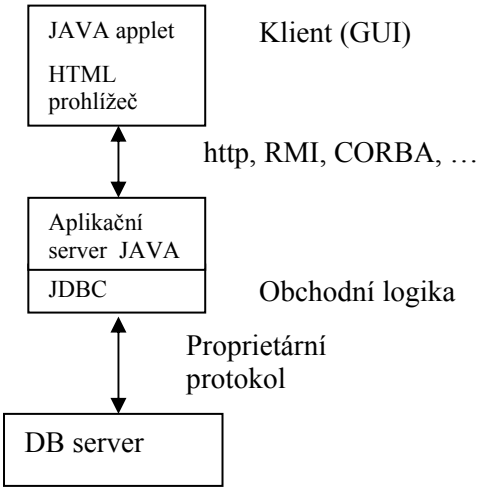


Dvouvrstvé architektury s podporou JDBC API, ve které Java applet nebo aplikace komunikuje přímo s databází, která může být i na jiném, po síti přístupném počítači.



Obrázek 16 dvouvrstvá architektura Java

Třívrstvá architektura s podporou JDBC API využívá střední vrstvu aplikačního serveru. Aplikační server může být implementován několika technologiemi, podporovanými Javou. Mezi nejčastější implementace patří komponentní technologie JavaBeans, umožňující sestavit aplikační vrstvu z nezávislých softwarových komponent, které mají své vlastnosti, metody a události. Pro uživatelské rozhraní se používají AWT JavaBeans, Swing JavaBeans. Pro webovské aplikace s vícevrstvou architekturou Java nabízí podporu J2EE (Java 2 Platform, Enterprise Edition) s moduly Java servlets, HTTP servlets a JavaServer Pages (JSP) pages.



Obrázek 17 Třívrstvá architektura Java

Příklad jednoduchého kódu pro přístup databázovým datům v jazyce Java:

```
import java.sql.*;
import oracle.jdbc.driver.*;
import java.io.*;
import java.util.*;

class JDBCVersion
{
   public static void main (String args[])
          throws SQLException
   {
      // Load the Oracle JDBC driver
      DriverManager.registerDriver
               (new oracle.jdbc.driver.OracleDriver());
      Connection conn = DriverManager.getConnection
             ("jdbc:oracle:thin:scott/tiger@localhost:1521:inf9");
      //        ("jdbc:oracle:thin:@hronek:1521:ora92h","scott","tiger");

      // The query we will execute
  String query = "select 'Hello JDBC: ' || sysdate from dual";
  // Create a statement
       Statement stmt = conn.createStatement ();

       // Execute the query
        System.out.println ("Executing query " + query + "\n");
       ResultSet rset = stmt.executeQuery (query);

       // Dump the result
       while (rset.next ())
         System.out.println (rset.getString (1) + "\n");

       // We're done
       System.out.println ("done.\n");
```

```
    }
}
```

Příklad jednoduchých zdrojových kódů pro ukázku Appletu

FirstApplet.htm:

```
<html>
<body>
<applet code=FirstApplet.class width=200 height=200>
</applet>
</body>
</html>
```

FirstApplet.java:

```
import java.awt.Graphics;

public class FirstApplet extends java.applet.Applet
{

    public void paint(Graphics g)
    {
        g.drawLine(0, 0, 200, 200);
    }
}
```

FirstApplet.java se přeloží do FirstApplet.class např. pomocí

FirstApplet.bat:

```
cd ..cesta..\pracovní_adresar
C:\cesta ...\jdk\bin\javac FirstApplet.java
```

5.2 Internetovské aplikace

5.2.1 Úvod – základní technologie

Webové aplikace mají svůj počátek v rozvoji technologií počítačových sítí a nových technologií elektronických dokumentů s např. multimediálními, databázovými informacemi dynamického charakteru.

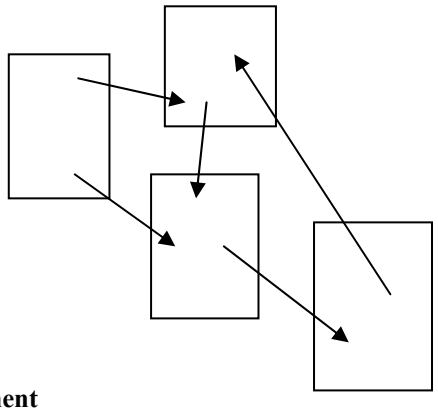
Historie Internetu začíná v roce 1969 vznikem grantové agentury ARPA – ARPANET v USA, která rozvíjela decentralizovanou univerzitní síť. V roce 1983 síť přechází na protokoly TCP/IP a opouští zastaralou architekturu, založenou na režimu Host-Terminál. Od roku 1990 se prosaruje nová struktura a topologie Internetu. Základ tvoří NSFNET – páteřní síť, na tu se připojují regionální sítě a na ně lokální sítě. Agentura NSF trvala na tom, aby síť byla jen pro vědeckovýzkumné - nekomerční účely. Financování provozu Internetu agenturou však bylo dále dlouhodobě neudržitelné, proto komercializace pronikla do struktury a to vedlo k oddělení NSFNETu. Vznikla hierarchická struktury poskytovatelů. Československá republika je připojena k internetu od roku 1991 prostřednictvím CESNET na zahraniční akademické sítě.

Identifikaci počítače v síti určuje IP adresa (Internet Protocol Address), což je 32 bitové binární číslo, zapisované po Bytech ve 4 oktetech (např. 127.0.0.1). Další důležitý pojem je DNS (Domain Name System), což je služba pro překlad doménového jména na IP adresu (např. www.inf.upol.cz odpovídá 158.194.80.80).

Trocha historie www (World Wide Web):

V roce 1945, Vannevar Bush napsal článek "As We May Think" popisující stroj Memex, obsahující lidské znalosti, organizované v podobě propojených materiálů, které se týkaly stejných témat. Tuto ideu využil Ted Nelson v projektu Xanadu a zavedl v roce 1965 termín hypertext. Hypertextový dokument není souvislý a jednolitý, ale skládá se z propojených částí dokumentu. Hyperlink je odkaz, propojující jednotlivé části dokumentu. Hypermedia představují multimediální hypertextový dokument.

Hypertext-
Hyperlink-
Hypermedia



Obrázek 18 Hypertextový dokument

Nástup WWW (World-Wide Web) služby

V roce 1990 Tim Berners-Lee vytvořil on-line systém pro podporu výzkumu ve švýcarském CERNu, založený na hypertextu. Umožňoval propojení různých počítačových sítí a vyměňování nebo sdílení informací. Vzniká první verze HTML (Hypertext Markup Language - značkovací jazyk pro popis struktury webových stránek, který je nezávislý na platformě), odvozená z obecnějšího a obsáhlejšího SGML a síťový přenosový protokol HTTP (Hypertext Transfer Protocol - jednoduchý aplikační bezstavový protokol modelu požadavek/odpověď, vytvořený nad protokolem TCP). Hlavní technologická inovace spočívá v adresování pomocí URL adres (Uniform Resource Locator - adresa, která jednoznačně identifikuje zdroj v Internetu). Uživatelsky přívětivé rozhraní zajišťuje internetovský prohlížeč, který zpřístupňuje webovské stránky pomocí zadání jejich URL.

Syntaxe URL (jednoho z typů URI - Universal Resource Identifier):

<protokol>://*<server_DNS_nebo_IP_adresa>*[:*port*]/[*cesta*][/*zdroj*]

Specifikuje typ spojení, např. *http*, *https* (secured site), *ftp*, *telnet*, atd.	Doménové jméno Web-serveru nebo IP adresa	Číslo portu, implicitně je 80, nebo 443 pro *https*	Adresář, kde je zdroj umístěn	Jméno zdroje

Případně s rozšířením:

*<protokol >***://***<uživatel>:<heslo>@<host>:<port>/<cesta>?<parametry>*

URI obecně sjednocuje dva druhy lokátorů – URL (Unified Resource Locator) a URN (Unified Resource Name). URN identifikuje zdroje na základě jejich jména. Používá se forma s prefixem *urn:*

urn:<nid>:<nss>

, kde < *nid* > je Namespace Identifier a < *nss* > je Namespace Specific String

Příklad:

urn:issn:0167-6423 - (časopis"Science of Computer Programming", identifikovaný sériovým číslem)

urn:mpeg:mpeg7:schema:2001- (jmenný prostor pravidel MPEG-7 video metadata)

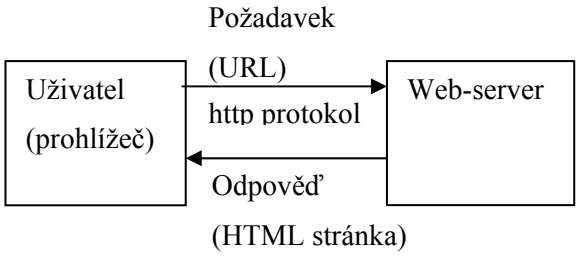
Mezi průkopníky patřili studenti university v Illinois, kde v roce 1993 v NCSA vznikl Mosaic, v roce 1995 Marc Andreessen navrhl Netscape. Internet se stal klíčovým fenoménem informatiky. Jeho nástroje a technologie můžeme klasifikovat do kategorií, zjednodušeně charakterizovaných:

s příkazovým řádkem – FTP (1971), *s menu* - gopher (1991), *pro vyhledávání* – WAIS(1991) a *hypermedia* – WWW(1991)

nebo jinak:

na komunikační služby – E-mail, telnet, IRC (internet relay chat), …, *na uložení a výměnu informací* – FTP, Gopher, Alex, …, *na vytváření informačních indexů* - Archie, Veronica, Wais, UCSTRI, Whois, … a *interaktivní práce s multimediálními informacemi* – WWW.

Základní princip WWW služby ukazuje následující obrázek:



Obrázek 19 Princip WWW služby

Při implementaci webových aplikací musíme vzít v úvahu, že protokol HTTP je bezstavový, web-server nemá trvalé nepřerušované spojení s klienty, nemůže je identifikovat. To komplikuje webové aplikace, které pracují se stavovou informací. Tato komplikace se řeší například přenášením údajů v příponě URL , ve skrytých polích formuláře, pomocí cookies nebo session proměnných.

- Cookies

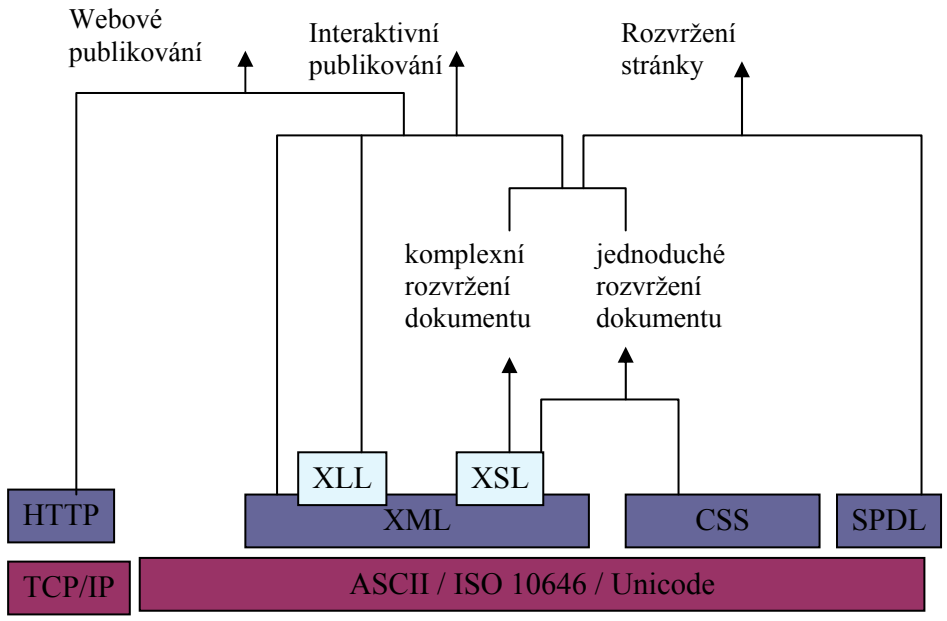
Cookie je svázána se serverem i klientským adresářem a proces zpracování informací z cookie funguje tak, že při opakovaných přístupech k témuž serveru je odpovídající cookie automaticky zaslána zpět na server. Platnost, délka uložení cookie se dá nastavit.
Předávání stavových informací pomocí cookies je nebezpečné – všechny stavové informace jsou v každém požadavku/odpovědi. Implementace je velice snadná, ale některé starší prohlížeče cookies nepodporují, novější zase umožňují cookies vypnout. Proto na cookies nelze spoléhat.

Cookies - krátká informace, kterou si web-server ukládá na straně klienta a využívá ji také k uchování stavu.

- Session proměnné

Každému novému uživateli se přiřadí unikátní identifikátor (tzv. session-id), které se předává s každým požadavkem pomocí cookie nebo parametrů v URL, resp. skrytých polí ve formuláři. Session-id je konstruováno tak, aby bylo těžko odhadnutelné (většinou náhodné číslo + hashovací funkce MD5 nebo SHA). Pro každé session-id má webový server vyhrazen prostor pro ukládání dat (proměnných) - sdílená paměť, soubory, databáze. Předávání stavových informací pomocí session proměnných je bezpečné – s každým požadavkem se přenáší jen malá část dat a session-id, šetří kapacitu sítě – data jsou ukládána přímo na web-serveru. Výhodou je snadná implementace (většina prostředí pracuje se session proměnnými téměř stejně jako s běžnými proměnnými) a podpora session proměnných ve skriptových prostředích.

Původně byl Web navržen pro sdílení statických dokumentů, nyní v něm lze používat různé aplikace a technologie (informační systémy, vyhledávače informací, elektronické obchody, …). Výhodou nové platformy je globální, celosvětový dosah, možnost nasazení levných tenkých klientů – malé náklady na instalaci a správu aplikace. Základem všech technologií je HTML.



Obrázek 20 Tecnologie publikování na webu

Internet přináší i mnohé problémy – například nebezpečí neoprávněného vniknutí do systému počítače (hacking), infikace viry, původní přenosové mechanizmy nechrání přenášená data. Problém nevhodných informací a dat (spam, autorské práva na multimediální data), věrohodnost informací, obtížné vyhledávání.

5.2.1.1 Webové servery

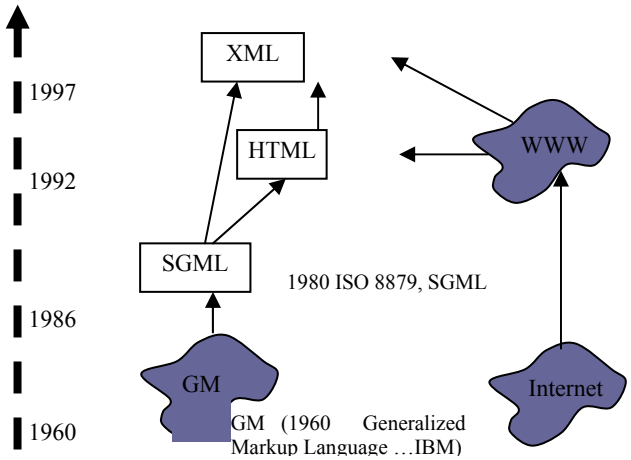
Pro praktické použití internetových technologií je nutné mít přístup k webovému serveru. Může to být server, který nabízí mnozí poskytovatelé (na KI UP je na katedrální síti k dispozici všem studentům webový server na adrese http://student.inf.upol.cz/, který je určen např. pro provoz webových aplikací a projektů), ale při experimentování a vývoji webových aplikací patrně dáme přednost instalaci a provozu lokálního personálního serveru. Záleží samozřejmě na platformě uvažovaného počítače, ale mezi uživateli si výsadní postavení udržuje webový server Appache (oficiální webové stránky , oficiální dokumentace , **přímý link na stažení** verze 2.0.59 win32 – x86 – no_ssl [4,3 MB] pro platformu Windows) – protože je nabízen pro různé platformy a je k dispozici volně ke stažení (Open Source). Daleko méně jsou používány další produkty, např. server SunOne.

Uživatelé operačních systémů Microsoft Windows patrně použijí server IIS, který je součástí dodávaného operačního systému. Problémy instalace a nastavení vlastností (zabezpečení – autentizace, …; rozšíření – použitelné skriptovací jazyky) jdou za rámec tohoto textu, ale alespoň několik poznámek k práci s IIS.

Služba IIS není součástí běžné instalace OS. Pokud potřebujeme IIS nainstalovat, zvolíme z ovládacích panelů položku Přidat nebo odebrat programy a na formuláři vybereme nabídku Přidat nebo odebrat součásti systému. Zaškrtneme nové požadované položky – Internetová informační služba. Administrace IIS probíhá z konzoly správy počítače (např. pravým tlačítkem na položce Tento počítač získáme menu, ve kterém zvolíme položku spravovat. Rozbalíme postupně Služby a aplikace, Internetová informační služba, Wbové servery. Pak je možno přes volbu Vlastnosti nastavovat jednotlivé objekty (server, aplikace – adresáře, stránky) i případně cele podstromy objektů, které jsou určeny kořenovým objektem.

5.2.2 SGML, HTML

HTML vychází z vybraných zjednodušených pravidel SGML (Standard Generalized Markup Language), což je ISO standard 8879 z roku 1986 a slouží k definici značkovacích jazyků. Je to metajazyk, prostředek pro definici nových jazyků, původně pro popis struktury a reprezentaci obsahu rozsáhlých dokumentů. SGML deklaruje možnou syntaxi pomocí tagů, které umožňují identifikaci nejenom samotných hodnot v dokumentu, ale také jejich sémantiku - popisuje „obsah + význam“ strukturovaných dat.



Obrázek 21 Vývoj značkových jazyků

Je flexibilní, slouží k popisu jednoduchých i komplexních dokumentů. Odděluje syntaktická pravidla od vlastního obsahu dokumentu, soustřeďuje se na strukturu a nabízí mnoho sofistikovaných možností, což způsobuje nepřiměřený nárůst rozsahu normy a komplikovanost - příliš velké možné zobecnění. To dále vede k náročnému zpracování (složité syntaktické analyzátory, aplikace …) . V jazyce SGML se může vytvářet definice typu dokumentu (DTD). DTD obsahuje seznam elementů, které se dají v dokumentu použít. U každého elementu jsou definovány jeho atributy a rovněž další elementy, které obsahuje.

První formální specifikace - HTML 2.0- byla schválena IETF v r. 1994. Obsahovala základní formátování a strukturování dokumentu, obrázky, formuláře. V roce 1995 ambiciózní návrh HTML 3.0 s rozšířením například o matematické vzorce, tabulky, obtékání obrázků, styly dokumentů, ale byla opuštěna, protože rozhodující firmy se nevyrovnaly se složitostí implementace prohlížeče a nedomluvily se na kompromisu. Rozšířeným standardem se stal v roce 1996 až HTML 3.2, ve kterém konsorcium W3C vybralo společnou podmnožinu rozšíření HTML různých výrobců prohlížečů. V roce 1997 další rozšíření HTML 4.0. Obsahuje podporu kaskádových stylů, skripty vložené do stránky, multimediální objekty, rozšířené možnosti tabulek, rámy. HTML 4.01 z prosince 1999 odstraňuje některé chyby ve specifikaci HTML 4.0. Další vývoj představuje XHTML (Extensible Hypertext Markup Language) – podobné HTML 4.01 se striktnější syntaxí XML. Obsahuje i podporu například vektorové grafiky, matematických vzorců, prezentace multimedií díky speciálním formátům XML: SVG, MathML a SMIL.

- Syntaxe HTML - Elementy a tagy

Jednotlivé části HTML stránky se označují pomocí elementů. Každý element se skládá z počátečního tagu, obsahu elementu a ukončovacího tagu. Tag je klíčové slovo HTML jazyka, uzavřené do lomených závorek. Elementy se hlavně používají pro vyznačení struktury dokumentu, pro vymezení parametrů elementu, sekce případně použijeme atributy. Atribut se zapisuje za počáteční tag. Pořadí atributů není určeno, může být v libovolném pořadí. Většina atributů je nepovinných a tedy nemusí být uvedeny a nabývají implicitní hodnoty. Explicitně uvedený atribut má vždy nějakou hodnotu, která se mu přiřazuje pomocí =. Pokud hodnota obsahuje pouze písmena, číslice, pomlčku a tečku, nemusí se hodnota atributu uzavírat do uvozovek. V opačném případě musíme hodnotu atributu uzavřít do uvozovek nebo do apostrofů. V klíčových slovech HTML se nerozlišují velká a malá písmena, v hodnotách se rozlišovat mohou. Tagy rozdělujeme na párové a nepárové. Párové uzavírají mezi sebe část dokumentu a přiřazují mu určité parametry:

<TAG parametr=hodnota > obsah (text) </TAG>

Některé elementy (například <br> a <hr>) nemusí mít ukončovací tag:

<TAG parametr= hodnota >

- Znakové entity

V HTML (i v XML) mají některé znaky speciální význam. Pokud je potřebujeme zapsat do stránky, použijeme znakovou entitu, která má tvar **&název_entity;**. Nejpoužívanější znakové entity jsou v následující tabulce:

Znak	Entita
<	<
>	>
&	&
"	"
©	©
nedělitelná mezera	

- Struktura HTML dokumentu

HTML dokument se skládá ze dvou částí, z hlavy (<HEAD>) a z těla (<BODY>). Obě části se nacházejí uvnitř HTML elementu, který označuje dokument jako HTML. Kostru dokumentu tedy tvoří:

```
<HTML>
        <HEAD>
        </HEAD>
        <BODY>
```

```
        </BODY>
</HTML>
```

<HEAD> obsahuje informace o dokumentu, které se nezobrazují s dokumentem, <BODY> obsahuje vlastní obsah dokumentu. Prvky, které jsou povoleny v <HEAD> nejsou dovoleny v <BODY> a naopak.

<HEAD> …… </HEAD> sekce může obsahovat následující elementy:

• <TITLE> …</TITLE> Titul stránky
• <BASE HREF="…"> bázová adresa pro relativní URL
• <SCRIPT …>…</SCRIPT> vnořený program ve skriptovacím jazyce
• <STYLE …>…</STYLE> specifikace stylu
• <LINK …> vytváří odkazy na další stránky
• <META…> obecný prvek pro meta-informace dokumentu, např. Name and Content :
<META Name="autor" Content="Novák">

<BODY …> …… </BODY> sekce, možné parametry elementu :
• BACKGROUND specifikuje URL obrázku na pozadí
• BGCOLOR specifikuje barvu pozadí
• TEXT specifikuje barvu textu
• LINK specifikuje barvu nenavštívených odkazů
• VLINK specifikuje barvu navštívených odkazů
• ALINK specifikuje barvu aktuálně vybraných odkazů
Vybrané elementy :

Nadpisy:

Nadpis je text, uzavřený do párového tagu <H*č* >, kde *č* tvoří číslo od 1 do 6. Čím je číslo *č* větší, tím méně důležitý nadpis vytvoříme.

Př.

```
<H1>Header 1</H1>
<H2>Header 2</H2>
<H3>Header 3</H3>
<H4>Header 4</H4>
<H5>Header 5</H5>
<H6>Header 6</H6>
<H? ALIGN=left|right|center>...</H?>
```

Horizontální čára:

K rozdělení textu se používá horizontální čára, se dvěma atributy, WIDTH – určuje délku čáry. Můžete ji zadat v pixelech nebo procentech a SIZE – udává šířku čáry.

Př.

```
<HR>
<HR SIZE=4>
<HR WIDTH=50%>
```

Řádkování a odstavce:

```
<BR>   nový řádek
 <P>odstavec</P>
<P ALIGN=left|right|center>...</P>
```

Také `<DIV>…</DIV>`

`<NOBR>…</NOBR> a <WBR>`

Písmo:

Písmo lze formátovat pomocí párového tagu <FONT>, atributy SIZE –udává velikost písma, ALIGN – určuje zarovnání textu, COLOR –

Text:

Pro změnu řezu písma se používají párové tagy:

```
<B>tučný</B> nebo <STRONG>…</STRONG>)
<U>podtržený</U>
<I>kurziva</I>

<TT>teletyp </TT>
<S>prošktnutý</S> (nebo <STRIKE>…</STRIKE>)
<BLINK>blikající</BLINK>
```

Pokud chceme, aby text vypadal stejné, jak ho píšeme v editoru (tedy i se všemi zalomeními řádku),

určuje barvu textu a FACE – udává druh písma

A také

```
<BIG>větší</BIG>
<SMALL>menší</SMALL>
<BASEFONT SIZE=3>
```

Odkaz:

Pro vložení hypertextového odkazu se používá párový tag **<A>**. Povinným atributem tohoto tagu je HREF a udává URL cíle odkazu. Obecný tvar zápisu odkazu vypadá takto:

```
<A HREF="URL"> text </A>
```

text se zobrazí v prohlížeči a kliknutím na něj se načte cílový dokument. Aby bylo možné odkázat na konkrétní místo v dokumentu (i stejném), musí se zde vytvořit záložka pomocí tagu <A> s parametrem NAME:

```
<A HREF="URL#zalozka">text</A>
…
<A NAME="zalozka">Toto je záložka</A>
```

Obrázek:

Obrázek vložíme do textu pomocí nepárového tagu <IMG>. Atributem SRC zadáváte adresu obrázku:

`<IMG SRC="URL">` , s atributy ALIGN – tento atribut udává polohu obrázku vzhledem k okolnímu textu (TOP| MIDDLE | BOTTOM | LEFT | RIGHT), WIDTH a HEIGHT – určují šířku respektive výšku obrázku v pixelech. Stačí uvést jen jeden atribut, druhý se automatický přizpůsobí, HSPACE a VSPACE – udává v pixelech horizontální nebo vertikální odsazení okolního textu od obrázku, BORDER –nastavuje šířku okraje obrázku. Při hodnotě 0 bude obrázek bez okraje.

použijeme

<PRE>Text se nemění

& 0<10</PRE>

<XMP>interpretace vypnuta 100>0</XMP>

Seznam:

Seznam se používá pro zdůraznění výčtu pojmů, může být - *řazený*:

```
<OL>
<LI>element 1
<LI>element 2
<LI>element 3
</OL>
<OL TYPE=A START=4 COMPACT>
<LI>element 1
<LI>element 2
<LI>element 3
```

</OL>, kde TYPE: 1 Arabská čísla | A abeceda (velká písmena) | a abeceda (malá písmena) | I římské číslice (velké) | i římské číslice (malé)

- *neřazený*

```
<UL>
<LI>element 1
<LI>element 2
<LI>element 3
</UL>
<UL TYPE=SQUARE>
<LI>element 1
<LI>element 2
<LI>element 3
```

`</UL>` kde TYPE: DISC •| CIRCLE o | SQUARE

- *definiční* termín, který chceme definovat zapíšeme do elementu <DT>, jeho popis do elementu <DD>.

```
<DL COMPACT>
<DT> termín1 </DT>
<DD>popis termín1 </DD>
<DT> termín2 </DT>
<DD> popis termín2 </DD>
</DL>
```

Příklad s grafickým objektem:

```
<DL>
<DT><IMG SRC="grafika.gif" ALT="*"> element
1
<DT><IMG SRC="grafika.gif" ALT="*"> element
2
</DL
```

Tabulka:

Tabulky se tvoří pomocí párového tagu <TABLE>, s parametry BORDER, ALIGN, WIDTH, CELLSPACING, CELLPADDING, BGCOLOR. Mezi tento tag se potom umísťují tyto párové tagy:

<CAPTION ALIGN=...>…</CAPTION> – slouží k zadání nadpisu tabulky,

<TR> – pomocí tohoto tagu se zadávají jednotlivé řádky tabulky (parametry ALIGN, VALIGN, BGCOLOR),

<TD> – tímto tagem se vloží do řádku buňka,

<TH> – slouží k vložení buňky se záhlavím buňky. Společné parametry jsou COLSPAN, ROWSPAN (

někdy je potřeba sloučit několik buněk v jednu. Pokud chceme sloučit několik buněk v jednom řádku za sebou, použijeme COLSPAN, hodnotou je počet buněk, které chcete sloučit, podobně ROWSPAN sloučí buňky pod sebou), ALIGN, VALIGN, BGCOLOR, WIDTH, HEIGHT, NOWRAP. Příklad:

```
<TABLE BORDER=1 WIDTH=100%>
<TR>
<TH>záhlaví1</TH><TH> záhlaví 2</TH>
</TR>
<TR>
<TD>data 11</TD><TD>data 12</TD>
<TR>
<TR>
<TD>data 21</TD><TD>data 22</TD>
</TR>
</TABLE>
```

Rámy:

Pomocí rámů máme možnost rozdělit okno v prohlížeči na několik částí. Do každé z těchto částí se načte jiný HTML dokument. Dělení okna může být jak vodorovné, tak svislé. Příklad:

```
<FRAMESET ROWS=50,*>
<FRAME SRC="DOC1.html">
<FRAMESET COLS="*,*">
<FRAME SRC="P1.html">
<FRAME SRC="P2.html">
</FRAMESET>
</FRAMESET>
<NOFRAME>
<BODY>Používej jiný prohlížeč</BODY>
</NOFRAME>
```

HTML obsahuje několik prvků, které umožňují vstup uživatele. Nejjednodušší je <ISINDEX>, který přijímá jednu řádku textu jako uživatelský vstup

Formulář:

Formulář použijeme v případě, kdy část nebo celý dokument funguje jako dialogové okno s běžně používanými prvky - vstupní pole, tlačítka, přepínače, popisky atd., které reprezentují vstupní data pro program, pracující na straně web-serveru. Pro přenos dat formuláře se využívají metody z http protokolu GET- data jsou přidána jako přípona URL (za otazníkem) v jednoduchém formátu jméno1=hodnota1& jméno2=hodnota2&…, nebo POST – data jsou přenesena „neviditelně" za hlavičkou http. Ke vložení formuláře do stránky slouží párový element FORM s nejpoužívanějšími atributy ACTION a METHOD. Atribut ACTION obsahuje typicky URL adresu skriptu, který bude zpracovávat data z formuláře (ASP, PHP, CGI, …) nebo e-mailovou adresu (mailto:e-mail@adresa). Atribut METHOD slouží k určení metody, kterou budou data zaslána na server (GET , POST) Pro větší formuláře bychom měli používat výhradně metodu POST. Formuláře nelze do sebe vnořovat. Dále ENCTYPE určuje zakódování dat. Obecně má tedy formulář následující strukturu:

```
<FORM ACTION="URL_script" METHOD="GET | POST" …>
```

…
Vstupní elementy formuláře
…

```
</FORM>
```

Vstupní elementy formuláře:

Textové pole

```
<INPUT TYPE="TEXT" NAME="t1" VALUE="Petr" SIZE="20">
```

Textové pole pro heslo, s atributy SIZE, VALUE, MAXLENGTH

```
<INPUT TYPE="PASSWORD" NAME="t2">
```

Textový blok

WRAP může být OFF| HARD | SOFT a určuje zobrazení původního řádku textu, případně rozdělení na konci řádku, podle šířky TEXTAREA .

```
<TEXTAREA NAME="t3" COLS="20" ROWS="3">
```

Řádky

textu

```
</TEXTAREA>
```

radiová tlačítka

```
<INPUT TYPE="RADIO" NAME="t4" VALUE="yes" CHECKED>
<INPUT TYPE="RADIO" NAME="t4" VALUE="no">
```

zaškrtávací tlačítka

```
<INPUT TYPE="CHECKLIST" name="t5" CHECKED>
<INPUT TYPE="CHECKLIST" name="t6">
```

Skryté proměnné

```
<INPUT TYPE="HIDDEN" NAME="t7" VALUE=5>
```

Obrazové mapy (na serveru)

```
<INPUT TYPE="IMAGE" NAME="t8" SRC="image.gif" ALIGN="RIGHT">
```

Kliknutí na obrázek předá na server dvě proměnné name.x a name.y (tady t8.x a t8.y)

Prvky bombo box a list box

```
<SELECT NAME="var9">
<OPTION VALUE="A"> option 1</OPTION>
<OPTION VALUE="B"> option 2</OPTION>
</SELECT>
<SELECT NAME="var10" SIZE="3" MULTIPLE>
<OPTION>Element 1</OPTION>
<OPTION SELECTED>Element 2</OPTION>
<OPTION>Element 3</OPTION>
<OPTION>Element 4</OPTION>
</SELECT>
```

Připojený soubor

```
<INPUT TYPE="FILE" NAME="t11"
VALUE="soubor.txt">
```

Tlačítko

```
<INPUT TYPE="BUTTON" NAME="t12"
VALUE="kontrola">
```

Odesílací tlačítko

```
<INPUT TYPE="SUBMIT" VALUE="Send">
```

Resetovaní tlačítko – nastaví proměnné na výchozí hodnoty

```
<INPUT TYPE="RESET" VALUE="resetuje
hodnoty">
```

Html je:
- jazyk pro rozvržení a propojení dokumentů
- definuje syntaxi a umístění elementů
- definuje hyperlinkové propojení dokumentů
HTML není:
- programovací jazyk
- nástroj pro zpracování textu

5.2.3 XML

Technologie XML se postupně prosadila nejenom do internetovských aplikací, ale hraje stále důležitější roli v databázových technologiích, standardizaci datových struktur, formátů a přenosových protokolů. Původně je XML (eXtensible Markup Language) určen pro výměnu a sdílení informací, pro dokumenty obsahující strukturované nebo semistrukturované informace. Je to metajazyk, čili souhrn pravidel pro tvorbu jednotlivých XML jazyků, standard World-Wide Web konsorcia. Aktuální verze je 1.0, z roku 1998. Porovnáme-li pro ilustraci HTML a XML, HTML má omezenou množinu přípustných tagů, syntaxe jazyka je dána omezeným výčtem tagů a jejich atributů, s příslušnou funkčností. XML dokument nemá množinu tagů limitovanou a význam a funkčnost tagů případně určuje autor. Další charakteristiky:

Zdroje podnikových IS obsahují jen

10 % srukturovaných informací

Semistrukturovaná data jsou neuspořádaná, nepravidelná nebo neúplná

.

- XML je založen na textovém formátu, což přináší čitelnost, jednoduché zpracování (i jednoduchým textovým editorem) a dokumentaci na nástrojích různých počítačových platforem, využitelných současně s HTML dokumenty. Součástí je silná mezinárodní podpora i jiných jazyků než je angličtina. Jako znaková sada se používá ISO 10646, což je 32bitová znaková sada, zcela shodná s Unicode 3.0 (kódování: UTF-16 a UTF-8). Dokáže rozlišit všechny dnes používané znaky světových jazyků (49 tisíc znaků). Mohou se vytvářet dokumenty, které obsahují texty v několika jazycích najednou. Pro jeden zvolený jazyk může být dokument v libovolném kódování (např. windows-1250, iso-8859-2).

  Př. <?xml version="1.0" encoding="iso-8859-2"?>

- Vysoký informační obsah - Pomocí XML značek vyznačujeme v dokumentu strukturu a význam jednotlivých částí textu, ne vzhled dokumentu. Vlastní zobrazení dokumentu XML je určeno stylem. Styl se dá definovat pomocí různých technologií, obecně se dá charakterizovat konverze XML do dalších formátů jako snadná. Existuje například několik stylových jazyků, které definují prezentaci jednotlivých elementů. Pro dokument můžeme vytvořit schéma (například DTD, XML Schema, RelaxNG), které definuje přípustnou strukturu dokumentu.

Soubor pravidel nebo příkazů, které převedou dokument do jiného formátu se říká styl.

XML umožňuje vytváření odkazů v rámci jednoho dokumentu i mezi dokumenty (XML Linking, tj. XLink nebo také XLL - the eXtensible Linking Language). Nabízí v tomto směru mnoho možností nad rámec odkazů v HTML. Připouští i vícesměrné odkazy, které spojují více dokumentů dohromady i možnost uložení odkazů mimo dokumenty, kterých se týkají a tak vytvářet různé anotace a komentáře k již existujícím stránkám. Dále nabízí dvousměrné odkazy, odkazy se speciálním chováním, odkazy do databází s možností filtrace, setřídění, analýzy a zpracování odkazovaných kolekcí. Tvorba odkazů je popsána ve třech standardech:

Odkazy v XML:

XLL (the eXtensible Linking Language)

XLink (XML Linking Language

XPath (XML Path Language)

XPointer (XML Pointer Language).

XPath (XML Path Language) je jazyk, který umožňuje adresovat jednotlivé části dokumentu. Jeho možnosti dále rozšiřuje jazyk -

XPointer (XML Pointer Language). Příklad výrazu:

```
http://Server1.com/xml/doc1#ROOT()CHILD(1,chap)STRING(7,"Havel",0)
```

nebo

```
http:// Server1.com/xml/doc1?XML-XPTR=ROOT()CHILD(1,chap)STRING(7,"Havel",0)
```

Absolutní umístění definujeme pomocí:

HERE() – identifikuje aktuální element

ID() – identifikuje element s atributem typu ID. Př. : `ID(sec1)`

```
<section target="sec1">…</section>'
```

HTML() – identifikuje HTML element <a> podle jména. Př. : `HTML(para1)`

```
<p><a name="para1">text...</a>…</p>
```

ROOT() – identifikuje celý dokument pro další navigaci
DITTO() – specifikuje výsledek prvního vyhledání jako výchozí bod pro použití v následujícím výhledávání.
Relativní umístění definuje:

CHILD() – identifikuje dětský uzel aktuálního elementu. Př.:

```
<para>…</para>          <1                          CHILD(3, .)
<list>…</list>          <2
<para>…</para>          <3
                                                    CHILD(3, *)
<number>13</number>       <1
High Str.,                <2
<town>NewTown</town>      <3
                                                    CHILD(1, *CDATA)
<number>13</number>
High Str.,                 <1
<town>NewTown</town>

                                                    CHILD(3, para)
<para>…</para>             <1
<list>…</list>
<para>…</para>             <2
<table>…</table>
<para>…</para>             <3
                                    CHILD(3, para, status, secret)
<para status="secret">…</para>      <1
<para status="SECRET">…</para>      <2
<para status="normal">…</para>
<para status="normal">…</para>
<para status="Secret">…</para>      <3

                                    CHILD(2, para, author, "D. Adams")
<para author="D.Adams">…</para>
<para author="Dikens">…</para>
<para author="D. Adams">…</para>    <1
<para author="d. adams">…</para>
<para author="D. Adams">…</para>    <2
```

ANCESTOR() – prohledává vnořené elementy
FSIBLING() - prohledává následné elementy na stejné úrovni

```
>       <para>…</para>
        <para>…</para>          <1
```

PSIBLING()- prohledává předcházející elementy na stejné úrovni

```
        <para>…</para>          <1
>       <para>…</para>
```

DESCENDANT() – indikuje element kdekoliv v podstromu elementu
FOLLOWING()– indikuje element kdekoliv v podstromu elementu a pokračuje až do konce dokumentu.
PRECEDING()– indikuje element kdekoliv v předcházejícím dokumentu
STRING()- indikuje písmeno, slovo, frázi z textu, první parametr je počet výskytů, třetí parametr definuje posunutí od počátku oblasti.

```
                                STRING(1, 'Havel je prezident', 3)
<n>Have</n> je <fu1> prezident</fu1>     <1
```

XLink (XML Linking Language) je specializovaný jazyk pro tvorbu odkazů. Jednotlivé dokumenty se určují pomocí jejich URL adresy, za kterou lze uvést ještě XPointer pro přesnější určení části dokumentu. Využívá se atributů *Href* a *xml-link*. Zdrojový text obsahuje element <title> pro pojmenování cíle, kterému odpovídá atribut elementu <link> - <!ATTLIST link … title CDATA #IMPLIED>

… <link href="#X1" title="Upřesnění ">termín1</link> … . Kategorizaci odkazů je možné provést pomocí atributu *role* :<!ATTLIST link …role CDATA #IMPLIED>

… <link href="#X1" role="popis">termín1</link> …

Př.1

```
<!ATTLIST odkaz1 xml-link CDATA #FIXED "odkaz1">


< odkaz1 href="http://Podklady.xml#kap1" xml-link="odkaz1">
Podrobnosti v kapitole 1
</odkaz1>
```

Př.2

```
<link target="Server1.com/xml/Podklady#X1">
Upřesnění </link>
…
</kapitola>
< kapitola ident="X1">
<title> Upřesnění </title>
<p> text …
</kapitola>
…
```

- Univerzální datový formát, definující datový model, jako základ pro databázi ve formě XML dokumentů. Rozlišujeme dva typy – buď založené na dokumentech, anebo na formátovaných datech.
- Následující tabulka porovnává typické vlastnosti dokumentů a databází.

	dokumenty	Dokumenty XML	databáze
obsah: **paradigma**: **metadata**:	- mnoho malých dokumentů - obvykle statický - implicitní struktura sekce, kapitola, odstavec, věta, … značky (elementy) přizpůsobený člověku formát/rozvržení na médiu, anotace "ulož jako", wysiwyg autor název, datum, abstrakt,	- několik dokumentů - statický i dynamický - implicitní struktura Uživatelem definovaná sromová struktura … značky (elementy) přizpůsobený člověku i stroji formát/rozvržení dat ulož jako data vlastní elementy, …	- několik rozsáhlých databází - obvykle dynamický - explicitní struktura (schéma) záznamy přizpůsobený stroji schéma, data, metody atomicita, souběžnost, izolace, trvanlivost popis schématu

Průvodce studiem

HTML je jazyk se specifickou množinou tagů. XML je metajazyk o pravidlech tvorby XML dokumentu, tagů, ... XML dokument nemá množinu tagů limitovanou a význam a funkčnost tagů případně určuje autor.

5.2.3.1 Základy syntaxe

Strukturu XML dokumentu můžeme rodělit na

1. fyzickou – složenou z částí, které nazýváme *etity* a
2. logickou – umožňující rozčlenit dokument na pojmenované části a podčásti, nazývané *elementy*.

Fyzický pohled- entity

Logický pohled – DTD, XML Schema

Rozlišujeme proto potom několik prvků XML dokumentu - Elementy s atributy, Instrukce pro řízení procesu zpracování (processing instruction), deklarace DOCTYPE, sekce CDATA, Reference na entity (entity reference) nebo na znaky (character reference) a komentáře (comments)

- Elementy a atributy

Základním prvkem XML dokumentu je element. Je tvořen pomocí počátečního a ukončovacího tagu a uvnitř mezi tagy je obsah elementu:

<význam_elementu > obsah_elementu </význam_elementu >

Prázdný element bez obsahu zapíšeme:

<význam_elementu ></význam_elementu > nebo zkráceně
<význam_elementu />

Přísná hierarchie elementů se vytváří vkládáním vnořených elementů do obsahu kořenového elementu

```
<význam_elementuA> obsah_elementuA
  <význam_elementuAA> obsah_elementuAA
  </význam_elementuAA>
</význam_elementuA>
```

Oproti HTML musí XML-dokument splňovat mnohem přísnější pravidla. Elementy se nesmí křížit, tagy musí být spárované nebo má element prázdný obsah. XML dokument povinně definuje jeden kořenový element, který obsahuje celý dokument.

Atributy se většinou používají pro přidání různých metainformací k elementům. Tvoří je dvojice název_atributu="hodnota", umístěné uvnitř startovacího tagu elementu. Hodnota atributu musí být uzavřena v uvozovkách:

<význam_elementu atribut="hodnota atributu"> obsah_elementu </význam_elementu >

Názvy elementů a atributů začínají písmenem, podtržítkem nebo dvojtečkou, další znaky jsou písmena, číslice, tečka, pomlčka, podtržítko, dvojtečka a některé další znaky. XML je case sensitive, rozlišuje malá a velká písmena. Přesná definice je ve specifikaci XML (http://www.w3.org/TR/REC-xml).

- Entity

Reference na entitu umožní vložit do dokumentu namísto reference určitá data. Základní důvody pro použití entity reference jsou při vkládání znaků, které nelze přímo typovat nebo

při použití odkazů na jiná XML data. Syntaxe pro zavedení entity reference zahajuje znakem **&** a ukončuje znakem **;** : **&refrence;**

Zápis rezervovaných znaků můžeme provést s pomocí entit:

&lt;	**<**
&amp;	**&**
&gt;	<
&apos;	'
&quot;	**“**, nebo tak, že znak s libovolným kódem z ISO 10646 zapíšeme pomocí entity:

&#kód; kde kód je číslo v desítkové soustavě, nebo

&#xkód; kde kód je číslo v šestnáctkové soustavě.

Jeden XML dokument může být uložen v několika souborech. Entity jsou části, ze kterých se dokument skládá. Rozlišujeme interní textové entity, externí textové entity, externí binární entity, parametrické entity (pouze v DTD). Deklarace entity se provádí buď v externím DTD nebo přímo uvnitř deklarace typu dokumentu:

```
<!DOCTYPE dokument [
<!ENTITY název definice>
]>
<dokument>
...
</dokument>
```

V dokumentu se entita aplikuje pomocí odkazu na entitu **&název;**

Interní textové entity se používají pro opakovaně často používané části textu s možností rychlé a konzistentní úpravy dokumentu, nebo pro znaky, které nejsou dostupné na klávesnici. Syntaxe deklarace: **<!ENTITY název "text">**

Př. :

```
<?xml version="1.0"?>
<!DOCTYPE historie [
<!ENTITY univerzita "Univerzita Palackého Olomouc">
]>
< historie >
<název>&univerzita;</název>
…
</ historie>
```

Externí textové entity umožňují rozdělení XML dokumentu do více souborů, což přináší lepší správu a možnost editace jednoho dokumentu více uživateli najednou. Pokud není entita v kódování UTF-8 nebo UTF-16 musí začínat deklarací kódování (obdoba deklarace XML, ale atribut version je nepovinný)

Syntaxe deklarace: **<!ENTITY název SYSTEM "zdroj URI">**

Zdrojem dat je *URI (Uniform Resource Identifier).* URI jsou definován jako jednoznačný identifikátor jakýchkoliv objektů na Internetu, který se skládá z URL (identifikuje zdroj dostupný na určitém místě určitým protokolem - FTP, HTTP, NNTP, Gopher, ...) a URN(Uniform Resource Name - identifikuje zdroj nezávisle na jeho umístění). Pro URN se nutné použití vhodné překladové služby, podobné DNS, která převede URN na URL. Při deklaraci externích entit můžeme také využít veřejný identifikátor, který identifikuje určitou entitu nezávisle na umístění (podpora přenositelnosti dokumentů mezi různými systémy) se syntaxí:

<!ENTITY název PUBLIC "veřejný identifikátor" "zdroj URI">

Př. :

```
<?xml version="1.0"?>
<!DOCTYPE historie [
<!ENTITY město SYSTEM "Olomouc.xml">
]>
< historie >
```

```
<obec>&město;</obec>
…
</ historie>
```

Další možností jsou katalogové soubory, s jejichž pomocí se mapují veřejné i systémové identifikátory na lokální kopie souborů. Systémový identifikátor obsahuje veřejně dostupnou URL adresu. Výhodné je využití v době, kdy není přístup k síti a použije se lokální kopie dat.
Př.:

```
<!DOCTYPE historie PUBLIC "-//něco//někde//historie" "historie.dtd">
```

Externí binární entity se nevkládají přímo do dokumentu, ale prostřednictvím jména se vytvoří odkaz na externí soubor. Do dokumentu se vkládá jako hodnotu atributu speciálního typu se syntaxí: **<!ENTITY název SYSTEM "zdroj URI" NDATA "notace typ dat">**
Př. :

```
<!ENTITY foto1 SYSTEM "/foto/foto1.tif" NDATA "TIFF">
```

Častěji se externí soubory vkládají uvedením jejich jména přímo ve zvoleném atributu.

Další možností práce s binárními daty je použití deklarace NOTATION, která slouží k identifikaci speciálních typů externích binárních souborů, se syntaxí :

<!NOTATION název_notation SYSTEM typ>

Aplikuje se podobně uvedením názvu zvoleného notation jako hodnoty některého z atributu typu NOTATION.

Parametrické entity umožňují sdílení deklarací. Syntaxe pro zavedení parametrické entity (předpona se znakem %):

Interní **<!ENTITY %nazev_param_entity definice>**

Externí

<!ENTITY % nazev_param_entity SYSTEM URI>

<!ENTITY % nazev_param_entity PUBLIC FPI URI>

Element deklarujeme pomocí parametrické entity:

<!ELEMENT nazev (%nazev_param_entity;)*>

Několik elementů s různými názvy může tak být deklarováno pomocí jedné parametrické entity.

Př.:

```
<!ELEMENT CUSTOMER (NAME, DATE, ORDERS)>
<!ELEMENT BUYER (NAME, DATE, ORDERS)>
<!ELEMENT DISCOUNTER (NAME, DATE, ORDERS)>

<!ENTITY % record "(NAME, DATE, ORDERS)">
<!ELEMENT CUSTOMER %record;>
<!ELEMENT BUYER %record;>
<!ELEMENT DISCOUNTER %record;>
```

- Komentář vložíme do dokumentu následovně:
  <!-- ukázkový komentář -->

Nesmí však obsahovat – a komentáře se nesmí být navzájem vnořovat. Pokud syntaktická pravidla XML dokument splňuje, je správně strukturovaný (well-formed). To lze ověřit pomocí vhodného parseru. Nejjednodušší kontrolou je bezproblémové zobrazení ve vhodném internetovém prohlížeči. Častou chybou je například nesouhlas tagů elementu, nesprávné překrytí, vnoření nebo neukončení tagů.

- Instrukce pro zpracování (processing instructions) nejsou součástí obsahu XML dokumentu. Umožňují standardním způsobe začleňovat nestandardní data pro instrukce

řízení XML procesoru a dalších preprocesorů. Interpretace závisí na aplikaci. Do dokumentu se vkládají ve tvaru:

<?aplikace instrukce?>

Příklady :

- první tag XML dokumentu <?xml version="1.0" encoding="Windows-1250"?>
- připojování stylu k dokumentu <?xml-stylesheet href="styl.css" type="text/css"?>
- zlom řádky/stránky <?myDTP page-break?>

- Sekce CDATA se používá pro části dokumentu XML, kde nechceme interpretovat vyhrazené znaky jako například '<', '&' a můžete tedy do této sekce vložit libovolný výraz:

  <![CDATA[řetězec]]>

Př.:

```
<firma><![CDATA[Hanák & syn]]></firma>
```

Sekce CDATA nesmí obsahovat]]>.

DTD má v XML podobný význam jeho schéma v relačních databázích

Možnost XML vytvářet uživatelské elementy, atributy a s nimi libovolné hierarchické struktury dat může být omezena v případech, kdy je třeba definovat specifický jazyk s relativně pevnou strukturou, kterou je třeba dodržet a proto zkontrolovat - provést validaci dokumentu (syntaktickou správnost dokumentu), tj. kontrolu struktury dokumentu porovnáním s dohodnutou definicí. Takto vymezenou strukturu můžeme definovat jeho typ dokumentu (např. webová stránka, dopis, faktura, …). Zároveň tím dostáváme prostředek pro definici syntaxe jednoho určitého XML jazyka. Historicky prvním prostředkem jednotného postupu při verifikaci je jazyk Data Type Definition – DTD. S jeho pomocí definujeme nové jazyky, které základní syntaxí vycházejí z XML. DTD definuje elementy a jejich vztahy, atributy i jejich typy, které můžeme v dokumentu daného typu použít, které elementy mohou být vynechány, které se mohou opakovat, definuje někdy implicitní hodnoty, atd.. Dokument vyhovující DTD hodnotíme jako validní dokument. Deklarace DTD se může vyskytovat buď uvnitř dokumentu XML, který má těmto definicím vyhovovat, nebo se do dokumentu XML uvede odkaz na definice DTD v externím souboru.Typ dokumentu určíme v deklaraci typu dokumentu (DOCTYPE), například:

<!DOCTYPE kořenový_element_dokumentu SYSTEM " název_souboru.dtd">

S připojením DTD k dokumentu XML souvisí funkce atributu standalone. Atribut se umisťuje do úvodního tagu : <?xml... standalone="yes" ...?>

Nastavení na hodnotu yes znamená, že dokument XML ke své interpretaci nepotřebuje DTD dokument, i když je na něj odkaz. V opačném případě obsahuje DTD informace, bez kterých dokument XML ztrácí smysl (implicitní nebo fixní hodnoty atributů).

```
<?xml version="1.0"?>
<!DOCTYPE dopis SYSTEM "dopis.dtd">
<dopis>
  <adresát>
    <jméno>Jan Novák</jméno>
    <adresa>...</adresa>
    ...
  </adresát>
  <předmět>Pozvánka na besedu o XML</předmět>
  <tělo>
    <para>Vážený pane,</para>
```

```
    <para>rád bych Vás jménem naší...</para>
    ...
  </tělo>
</dopis>
```

- *Jmenné prostory* slouží k rozlišení elementů (atributů) se stejnými jmény a umožňují tak kombinovat více sad značek v jednom dokumentu. Jmenné prostory se nejčastěji identifikují pomocí URI adresy, sloužící jako unikátní identifikátor. Při použití jmenných prostorů se jména elementů a atributů skládají ze dvou částí – ze jmenného prostoru a z lokálního názvu. Prakticky se pro zkrácení zápisu při deklaraci jmenného prostoru vytváří prefix, který jmenný prostor zastupuje:

  <prefix:element xmlns:prefix="http://nekde.com/neco"> .

  </prefix:element>

Často se v XML dokumentu používá implicitní jmenný prostor (prefix je prázdný)

Dokument se jmennými prostory

<kořenový_element xmlns: =“...URI1...“

xmlns:předponaA=“...URI2...“

xmlns:předponaB=“...URI3...“>

< element ...>...

<předponaA:element ...>...

<předponaB:element ...>...

...

</ kořenový_element >

Vzhled jednotlivých elementů a tím i celého dokumentu je potřeba definovat pomocí stylu. Pro definici vzhledu XML-dokumentů se používají stylové jazyky, např. CSS, XSL a DSSSL.

Struktura typického XML dokumentu:

prolog	Deklarace XML	`<?xml version="1.0" encoding="Windows-1250" standalone="yes"?>`
	Instrukce pro zpracování xml-stylesheet	`<?xml-stylesheet type="text/xsl" href="kniha.xsl"?>`
	Typ dokumentu	`<!DOCTYPE katalog SYSTEM "katalog.dtd">`
	Poznámka	`<!--katalog vytvořen 1.1.2005-->`
Elementy dokumentu	Kořenový element s deklarací jmenného prostoru	`<katalog xmlns="http://www.moje.com/katalog/">`

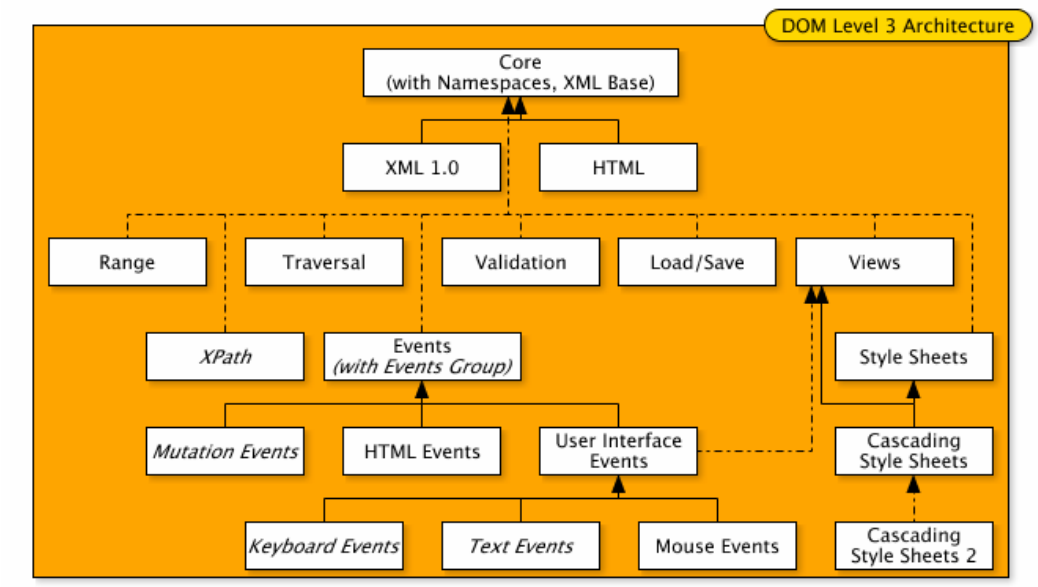
		`<list id =“l1“>`
		`<název> model &#47; K2 </název>`
		`<firma><![CDATA[Hanák & syn]]></firma>`
		`</list>`
		`</katalog>`

- XML Document Object Model - DOM

Zpracování XML dokumentu zahrnuje procesy pro uložení, čtení, zpracování událostí a manipulaci ve stromové struktuře dokumentu, transformace XML dokumentu. Za čtení dokumentu je zodpovědný XML procesor, detekuje mimo jiné chyby způsobené nevhodným formátem souboru, který aplikace není schopná zpracovat, kontroluje správnost odkazů. Obsahuje také jednotku etitního manažeru, který lokalizuje fragmenty dokumentu deklarované v entitách. Pro čtení obsahu XML dokumentu se používají dvě odlišné techniky:

1. událostmi řízené – dokument je zpracováván striktně definovaným postupem jako proud dat plynoucí do aplikace, přičemž každý element v datovém proudu vyvolá speciální událost, kterou aplikace může individuálně a specificky ošetřit.
2. průchod stromem, manipulace se stromem dokumentu – celý dokument je přístupný ve formě hierarchické struktury s možností získání a manipulace s daty v libovolném pořadí.

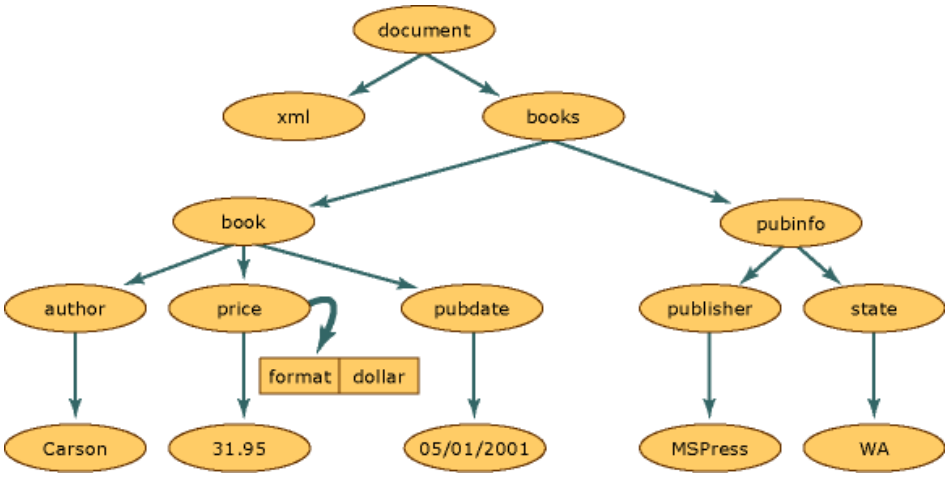
Nejčastěji se funkčnost zajišťuje pomocí DOM. Document object model pro HTML je objektovým modelem DHTML a například v technologiích Microsoftu se jmenuje DOM IE. Analogicky byl zaveden DOM pro XML dokumenty. Definuje na platformě a na programovacím jazyku nezávislé rozhraní pro programy a skripty a tím umožňuje dynamicky zpřístupňovat a upravovat obsah, strukturu nebo styl dokumentu. Prakticky se operace provádí metodami objektů, které se inicializují zpracovávanou HTML stránkou nebo XML dokumentem. V průběhu vývoje prošla specifikace několika úrovněmi. Architektura DOM level 3 je na následujícím obrázku, převzatém z http://www.w3.org.



Pro názornost příklad z dokumentace MS .NET - ze vstupního XML dokumentu:

```
<?xml version="1.0"?>
  <books>
    <book>
        <author>Carson</author>
        <price format="dollar">31.95</price>
        <pubdate>05/01/2001</pubdate>
    </book>
    <pubinfo>
        <publisher>MSPress</publisher>
        <state>WA</state>
    </pubinfo>
  </books>
```

Se vytvoří stromová struktura:



XML DOM je obecný objektový model XML dokumentu se specifickými vlastnostmi (procesní instrukce, CDATA, entity). DOM obsahuje také DOM Events – manipulační události, související s XML stromovou strukturou dokumentu i uživatelské události (myš, klávesnice, specifické HTML), DOM Cascading Style Sheets (CSS) – pro formátování dokumentu, DOM Validation, DOM XPath – umožňuje dotazování pomocí jazyka XPath. Dokument je zpracováván jako strom uzlů, kde například každému elementu odpovídá jeden uzel.Vnořené elementy a případný text tvoří příslušný podstrom. Existuje mnoho využívaných implementací DOM (COM, .NET, Java, …), pro ilustraci praktického použití se zaměříme na implementaci Microsoft. Základní objekty XML DOM jsou pro COM (.NET) implementace:

- XMLDOMDocument (XmlDocument) objekt
- XMLDOMNode (XMLNode) objekt
- XMLDOMNodeList (XMLNodeList) objekt
- XMLDOMNamedNodeMap (XmlNamedNodeMap) objekt

XMLDOMDocument objekt - provádí základní operace s dokumentem (načtení, procházení, dotazování, změny a uložení v XML dokumentu). Některé metody a vlastnosti:

- .load(string), .save(string) se vstupním parametrem, který identifikuje XML soubor cestou, URL, …
- .url URL posledního nataženého dokumentu (string)
- .loadXML(string) .saveXML(string) se vstupním parametrem ve tvaru XML sekvence, např. … .loadXML("<katalog>...</katalog>")
- .documentNode s pomocí této vlastnosti se dá přiřadit objekt typu XMLDOMNode a tím určit celou hierarchii dokumentu XML.
- .doctype informace o typu objektu (XMLDOMDocumentType). Ekvivalentní s <!DOCTYPE...> v DTD).
- .async - zda se má provést načtení dokumentu asynchronně.
- .readyState – průběžný stav analýzy (async = true)
- .parseError – uchovává poslední chybu analýzy s dalšími informacemi (XMLDOMParseError objekt)
- .validateOnParse – rozhoduje o provedení validace dokumentu (pokud ne, provede se kontrola syntaxe dokumenru)
- .resolveExternals - zda jsou přípustné externí dokumenty (DTD, …)
- .preserveWhiteSpace rozhoduje o zachování neviditelných znaků (implicitně jsou potlačeny).

Výběr uzlu v XMLDOMDocumentu:

- .documentElement vrací kořenový uzel XML dokumentu (objekt typu XMLDOCNode)
- .nodeFromId vrací uzel, element Node, podle Id definovaného v DTD nebo XML schéma.
- .getElementsByTagName vrací kolekci node podle požadovaného Tag-Name

Vytvoření nových prvků v XML dokumentu:

- .createNode se vstupními parametry jsou postupně typ prvku, název a URI pro jmenný prostor
- metody speciálně určené pro tvorbu prvku daného typu (createAttribute, createCDATASection , createComment, createDocumentFragment, createElement, createEntityReference, createProcessingInstruction, createTextNode). Pouhé vyvolání těchto metod (podobně createNode s parametrem typu) nezpůsobí vytvoření prvku v dokumentu. To se uskuteční až přidáním prvku do dokumentu metodou se vstupním parametrem tohoto prvku (např. .AppendChild).

XMLDOMNode objekt - obsahuje vlastnosti s informacemi o prvku XML:

- .attributes - vrací kolekci všech atributů typu XMLDOMNamedNodeMap
- .hasChildNodes - boolean, vrací true, pokud node má podřízené node
- .namespaceURI - string, read-only. Vrací namespace daného prvku
- .nodeName - název prvku
- .nodeType - typ prvku, jedna z možných hodnot výčtu číselných konstant
- .nodeTypeString - typ prvku jedna z možných hodnot výčtu string konstant
- .parsed - true, pokud jsou Node a všechny jeho podřízené Node rozparsovány (extense proti W3C)
- .specified - vrací false, pokud jeho hodnota byla dána z DTD, true pokud byla specifikována přímo v Node XML dokumentu (extense proti W3C)
- .xml (příklad 8), LoadXML(Node.xml) (extense proti W3C)
- .text, .nodevalue, .nodeTypedValue (spolu s tzv. XML Schema) zpřístupnění textu uzlu

Průchod hierarchickou strukturou XML dokumentu pomocí metod a vlastností:

(skupina metod, které vychází z daného aktuálního XMLDOMNode a míří k dalším prvkům XML dokumentu)

- .parentNode - vrací nadřízený prvek
- .childNodes - vrací kolekci podřízených prvků
- .firstChild - vrací první podřízený prvek
- .lastChild - vrací poslední podřízený prvek
- .previousSibling - vrací předešlého souseda na téže úrovni prvků
- .nextSibling - vrací následného souseda na téže úrovni prvků
- .ownerDocument - vrací root element celého dokumentu
- .selectNode, .selectNodes - vrací kolekci podřízených prvků vyhovujících dané

(skupina metod pro manipulaci s podřízenými prvky daného XMLDOM Node)

- .appendChild - přidá podřízený prvek
- .replaceChild - vymění podřízený prvek
- .removeChild - odstraní podřízený prvek
- .insertBefore - přidá „před“ daný prvek
- .transformNode - zpracuje daný prvek a podřízené prvky podle vstupního XSL stylesheet, vrací string
- .transformNodeToObject (pro XSL) - zpracuje daný prvek a podřízené prvky podle vstupního XSL stylesheet, vrací objekt

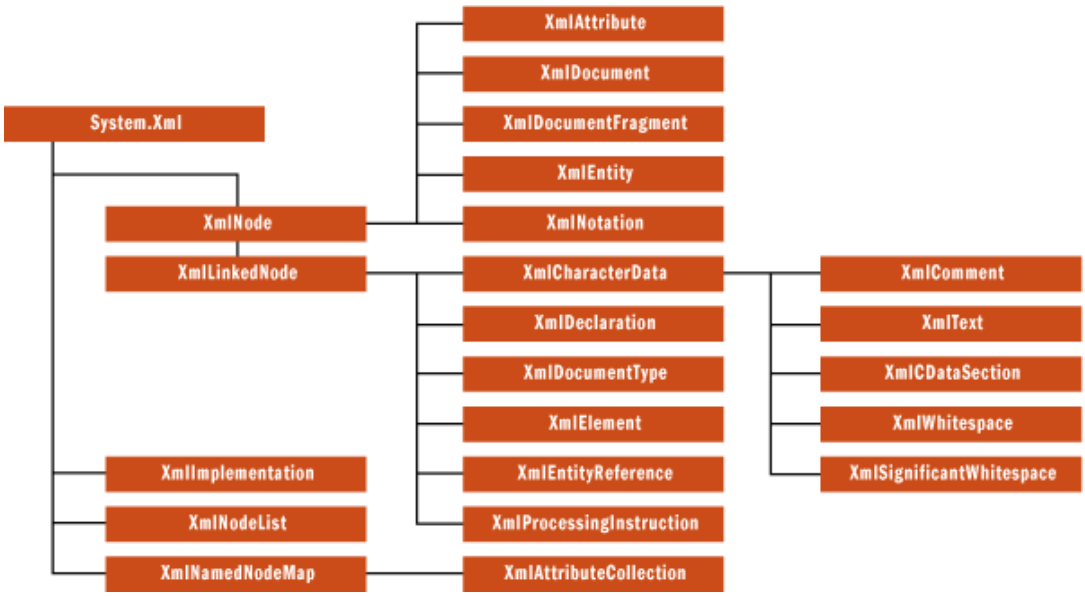
XMLDOMNodeList objekt určen pro práci s kolekcí prvků Node. Kolekce je výsledkem metod childNodes, selectNodes, getElementsByTagName.

- .length - vrací počet prvků
- .item(i) - (iterace začíná nulou) vrací aktuální i-tý prvek
- .nextNode - vrací další prvek (extense proti W3C)

XMLDOMNamedNodeMap objekt je kolekcí, která je navrácena jako vlastnosti prvku Node.

- .getnamedItem - vrací atribut specifického názvu
- .getQualifiedItem - vrací atribut specifického jmenného prostoru a názvu
- .item - vrací aktuální prvek
- .length - vrací počet prvků
- .nextNode - vrací následující prvek (extenze proti W3C)
- .removeNamedItem - odstraní prvek podle názvu
- .removeQualifiedItem - odstraní prvek podle jmenného prostoru a názvu
- .reset - vynuluje iterátor výčtu prvků (extenze proti W3C)
- .setNamedItem - přidá prvek do kolekce

Novější přístup nabízí platforma .NET Framework s implementací (System.Xml.XmlDocument), podporující jádro DOM level 1 a DOM level 2. se dvěma základními abstraktními XML třídami XmlReader and XmlWriter pro práci s XML (stream) a dalšími odvozenými třídami jako XmlTextReader, XmlValidatingReader.Net DOM API jsou velice podobné API v předchozích verzích MSXML knihoven.



Obrázek 22 Hierarchie tříd v .NET DOM

Ve stromové struktuře má každý uzel, kromě kořene, jediný rodičovský uzel. Uzly typu Dokument (kořen dokumentu s kompletní stromovou strukturou dokumentu), DocumentFragment (dočasná kolekce uzlů bez stromové struktury), EntityReference (reprezentuje odkazovaný text entity), Element(uzel elementu), Atribute mohou mít dětské uzly, na rozdíl od typů XmlDeclaration (reprezentuje uzel <?xml version="1.0"...>), Notation (reprezentuje notation z DTD), Entity (deklarace entity), CDATASection, Text (text elementu, atributu), Comment (komentář), ProcessingInstruction a DocumentType (reprezentuje uzel <!DOCTYPE...>)

Příklad použití knihovny ve VB.NET pro validaci XML dokumentu (z dokumentace MS .NET):

```
Imports System
Imports System.IO
Imports System.Xml
Imports System.Xml.Schema

public class ValidationSample

      public shared sub Main()
         Dim tr As XmlTextReader = new XmlTextReader("HeadCount.xml")
         Dim vr As XmlValidatingReader = new XmlValidatingReader(tr)

         vr.ValidationType = ValidationType.DTD
         AddHandler vr.ValidationEventHandler, AddressOf ValidationCallback
         while(vr.Read())
         end while
         Console.WriteLine("Validation finished")

      end sub

      public shared sub ValidationCallBack (sender As object, args As
ValidationEventArgs)

         Console.WriteLine("***Validation error")
         Console.WriteLine("Severity:{0}", args.Severity)
         Console.WriteLine("Message:{0}", args.Message)
     end sub
end class
```

se soubory HeadCount.dtd:

```
<!ELEMENT HeadCount (Name)*>
<!ELEMENT Name (Name)*>
<!ATTLIST Name First CDATA #REQUIRED>
<!ATTLIST Name Last CDATA #REQUIRED>
<!ATTLIST Name Relation (self | spouse | child) "self">
<!ENTITY MyFirst "Jeff">
<!ENTITY MyLast "Smith">
```

a HeadCount.xml:

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE HeadCount SYSTEM "HeadCount.dtd">
<HeadCount>
  <Name First="Waldo" Last="Pepper">
    <Name First="Salt" Last="Pepper" Relation="spouse"/>
    <Name First="Red" Last="Pepper" Relation="child"/>
  </Name>
  <Name First="&MyFirst;" Last="&MyLast;">
    <Name First="Sharon" Last="&MyLast;" Relation="spouse"/>
    <Name First="Morgan" Last="&MyLast;" Relation="child"/>
    <Name First="Shelby" Last="&MyLast;" Relation="child"/>
  </Name>
</HeadCount>
```

- XML v Internetovských technologích MS - databinding XML

Microsoft podporuje technologie databinding - provázání objektu s datovým zdrojem, kterým může být mimo jiné také XML dokument. V kontextu internetových technologií je používanou možností využití elementu <XML> a jeho atributů id (identifikátor) a src (zdrojový XML

dokument) ve spojení s vhodným prvkem pro prezentaci (např. tabulka) a jeho atributem *datasrc* a *datafld*. Příklad části HTML kódu:

```
...
<XML id=xml1 src="data.xml"></XML>
<TABLE datasrc ="#xml1" >   ...
    <TD><SPAN datafld ="položka1"></SPAN></TD>       ...
</TABLE> ...
```

V buňce tabulky je hodnota odpovídající atributu *datafld*. Možností je také s pomocí MS technologie DataIslands další využití elementu <XML> s atributem id - přímo do HTML kódu se vloží xml data. Identifikátor id se použije jako reference objektu HTML stránky.

5.2.4 DTD

DTD je syntaktická specifikace XML dokumentu, je to popis jeho typu. Používá se pro validaci XML dokumentu, tj. pro kontrolu správnosti a úplnosti dokumentu proti specifikaci. DTD se hodí pro popis formátů, které reprezentují především textové dokumenty. Neobsahuje ale nástroje na kontrolu jiných typů dat jako jsou čísla, měnové a časové údaje. Z hlediska programovacích jazyků jsou tedy DTD specifikace poměrně slabé, s jedním základním typem PCDATA, s netypovanými odkazy IDREF. Nepodporuje metody, integritní omezení, abstrakce.

Příklad validního XML dokumentu s implicitním DTD:

```
<?xml version="1.0"?>
<!DOCTYPE BOOK [
    <!ELEMENT BOOK (P*)>
    <!ELEMENT P (#PCDATA)>
]>

<BOOK>
    <P>chapter 1 - Intro</P>
    <P>chapter 2 - Conclusion</P>
    <P>Index</P>
</BOOK>
```

Deklarace DTD se dají vytvořit několikerým způsobem:

<!DOCTYPE kořenový_element_dokumentu [DTD]>

<!DOCTYPE kořenový_element_dokumentu SYSTEM URL>

Příklad pro souborové jméno:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE BOOK SYSTEM "book.dtd">
<BOOK>
......
</BOOK>
```

Příklad pro URL:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE BOOK SYSTEM "http://www.library.org/book.dtd">
<BOOK>
......
</BOOK>
```

<!DOCTYPE kořenový_element_dokumentu SYSTEM URL [DTD]>

Formální veřejný identifikátor (FPI) – skupiny se separují pomocí //

první skupina: - není standard, + je standard DTD

druhá skupina: jméno skupiny nebo člověka, zodpovědného za DTD

třetí skupina: typ dokumentu a verze

čtvrtá skupina: jazyk (dva znaky)

<!DOCTYPE kořenový_element_dokumentu PUBLIC FPI URL>

Příklad:

```
<!DOCTYPE BOOK PUBLIC "-//Joseph Smith//General Book Version 5.3//EN"
"http://www.library.org/book.dtd">
```

<!DOCTYPE kořenový_element_dokumentu PUBLIC FPI URL [DTD]>

Příklad XHTML Dokumentu:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
         "DTD/xhtml1-strict.dtd">

<html      xmlns="http://www.w3.org/1999/xhtml"      xml:lang="en"
lang="en">
<head>
  <title>Virtual Library</title>
</head>
<body>
   <p>Moved to <a href="http://vlib.org/">vlib.org</a>.</p>
</body>
</html>
```

DTD obsahuje deklarace čtyř typů:

- deklarace elementů
- deklarace atributů
- deklarace entit
- deklarace notací.

Deklarace typu elementu identifikuje povinný název elementu v XML dokumentu spolu s jeho obsahem (hierarchií podřízených elementů).

Syntaxe deklarace typu elementu: **<!ELEMENT název (obsah) .... >,**

kde název označuje název elementu v XML dokumentu. Název elementu musí začínat písmenem. Další znaky názvu mohou obsahovat písmena, číslice a některé speciální znaky jako ., -, _ ,: . Délka jména není omezená. Rozlišuje se velikost písmen.

Obsah označuje:

- seznam názvů následných podřízených prvků

- označení pro ukončení hierarchie (list stromu). Když element má již jen svůj text, použijeme výraz #PCDATA. Př.: <!ELEMENT jméno(#PCDATA)>

- prázdný element, který nemůže obsahovat žádné další elementy nebo text.

<!ELEMENT název EMPTY>

Opakem k EMPTY je ANY. Toto klíčové slovo zajistí, že element může obsahovat libovolné další elementy a text.

<!ELEMENT název ANY>

Pro strukturu vnořených elementů se používají regulární výrazy s pravidly:

, striktní pořadí elementů, sekvence `<!ELEMENT SUM (op1, op2)>`

?	nepovinný výskyt	`<!ELEMENT Table (plate)?>`
+	jeden nebo více	`<!ELEMENT BOOK (CHAPTER)+>`
*	žádný nebo více	`<!ELEMENT List (Object)*>`
|	jeden z několika, výběr	`<!ELEMENT POINT (COORDINATES | POLAR)>`
( )	skupina	`<!ELEMENT CHAPTER (INTRODUCTION, (P | QUOTE | NOTE)*, DIV*)>`

Pokud může element obsahovat jak text tak elementy, má smíšený obsah. Potom musí mít deklarace jeho obsahu speciální tvar - #PCDATA musí být uvedeno ve skupině jako první, skupina musí být spojena pomocí operátoru '|' a musí být volitelně opakovatelná (*), #PCDATA se nesmí vyskytovat v sekvenci.

Př.:

```
<!ELEMENT a  (#PCDATA | b)*>
<!ELEMENT b (#PCDATA)>
```

Deklarace atributu

V XML může mít každý element libovolné množství atributů. Deklarace atributu umožňuje identifikovat:

- které elementy mají atributy
- jaké atributy tyto elementy mají
- jaké hodnoty mohou tyto atributy mít
- jaké jsou implicitní hodnoty těchto atributů

Deklarace atributů jsou rovněž citlivé na velikost písmen a obsahují název elementu, název atributu, typ atributu a implicitní hodnotu. Syntaxe deklarace atributu:

<!ATTLIST název_element název_atribut typ implicit...>

Př.:

```
<!ATTLIST katalog katlog_id ID #REQUIRED>
```

Existuje několik možných typů atributů:

- CDATA – nejpoužívanější typ, může v XML dokumentu obsahovat libovolný string, text bez značek, tj. znaků (<) nebo (").
- ID - definuje jednoznačný XML identifikátor vůči všem ostatním ID atributům v dokumentu, bez mezer a podobných oddělovačů. Dá se využít k přístup k elementu metodami objektu XML dokumentu DOM. Implicitní hodnota je #REQUIRED.

Př.

```
<!ELEMENT student_name (#PCDATA)>
     <!ATTLIST student_name student_id ID #REQUIRED>
       …
     <student_name student_id="S9216735">Jo Smith</student_name>
```

IDREF nebo IDREFS - musí obsahovat jednu z hodnot atributů, který byl deklarován jako ID kdekoliv jinde. IDREF tedy zprostředkovává odkaz na jeden z elementů pomocí hodnoty atributu typu ID. Deklarace IDREFS může obsahovat několik odkazů oddělených mezerou.
Př.

```
<?xml version="1.0" standalone="yes"?>
<!DOCTYPE lab_group [
     <!ELEMENT lab_group (student_name)*>
     <!ELEMENT student_name (#PCDATA)>
     <!ATTLIST student_name student_id ID #REQUIRED
```

```
                                    tutor IDREF #IMPLIED>
            ]>
        <lab_group>
           <student_name student_id="S8904885">Alex Foo</student_name>
           <student_name student_id="S9011133">Sarah Bar</student_name>
           <student_name student_id="S9216735"
                                tutor="S8904885">Jo Smith</student_name>
        </lab_group>
```

- ENTITY nebo ENTITIES - jedna z deklarovaných ENTITY deklarací (např. v DTD), nebo několika ENTITIES , kde oddělovačem u několika deklarací je mezera.
- NMTOKEN nebo NMTOKENS - může obsahovat jedno slovo, které se skládá z písmen, číslic a dalších speciálních znaků('.', '-', '_', ':'). Při použití NMTOKENS atribut může obsahovat několik slov vyhovujících typu NMTOKEN, oddělených mezerou.

Př.

```
<!ELEMENT student_name (#PCDATA)>
     <!ATTLIST student_name student_no NMTOKEN #REQUIRED>
      …
     <student_name student_no="9216735">Jo Smith</student_name>
```

- NOTATION - slouží k určení typu dat, které obsahuje element, pokud se nejedná o XML-data. Hodnota je jméno, deklarované např. v DTD.

Př.

```
<!NOTATION Tex1 SYSTEM "..\TEXVIEW.EXE">
…

<!ENTITY Logo SYSTEM "LOGO.TEX" NDATA Tex1>
```

- Seznam možných hodnot - seznam hodnot se umisťuje za název atributu do závorky, oddělovačem je svislice |. Případná implicitní hodnota se umístí za závorku do uvozovek:

<!ATTLIST název_elementu název_atributu (hodnota | hodnota...) “implicitní hodnota”>

Př.

```
<!ELEMENT TITLE (#PCDATA)>
     <!ATTLIST TITLE
               ALIGN (LEFT | CENTER | RIGHT) "LEFT">
          …
     <TITLE>Programming XML in Java</TITLE>
          …
     <TITLE ALIGN="CENTER">Programming XML in Java</TITLE>
```

Průvodce studiem

DTD umožňuje v dokumentu zavádět vazby mezi elementy s kardinalitou 1:1 (ID a IDREF) a 1 : N (ID a IDREFS).Význam ID a IDREF je podobný jako je použití identifikačního klíče ve vazbách databázových relačních tabulek. IDREF odkazuje hodnotou na element, který má ID s hodnotou IDREF

Existují čtyři možné default nastavení pro vlastnosti atributu:

“hodnota” – definuje implicitní hodnotu atributu, atribut není vyžadován a pokud není uveden s jinou hodnotou, platí implicitní “hodnota”

Př.

```
<!ELEMENT ADDRESS (#PCDATA)>
     <!ATTLIST ADDRESS country CDATA "USA">
                …
       <ADDRESS >123 15th St. Troy NY 12180</ADDRESS>
       <ADDRESS country="ČR">Olomouc Zelená 55</ADDRESS>
```

#REQUIRED – bez implicitní hodnoty, atribut musí mít přiřazenu hodnotu v dokumentu

Př.

```
<!ELEMENT img EMPTY>
     <!ATTLIST img
               alt CDATA #REQUIRED
               src CDATA #REQUIRED>
          …
     <img src="xmlj.jpg" alt="XMLJ Image"/>
```

#IMPLIED - atribut není vyžadován a nemá implicitní hodnotu.

Př.

```
<!ELEMENT img EMPTY>
     <!ATTLIST img
                  alt CDATA #REQUIRED
            src CDATA #REQUIRED
            width CDATA #IMPLIED
            height CDATA #IMPLIED>
                  …
       <img src="xmlj.jpg" alt="XMLJ Image" width="300"/>
```

#FIXED “hodnota” - atribut není vyžadován, pokud je použit, musí mít konstantní hodnotu “hodnota”.

Př.

```
<!ELEMENT ADDRESS (#PCDATA)>
     <!ATTLIST ADDRESS
               country CDATA #FIXED "USA">
                  …
     <ADDRESS country="USA">123 15th St. Troy NY 12180</ADDRESS>
```

V předcházející kapitole byly popsány různé druhy používaných a deklarovaných entit, jako textové, interní, externí a parametrické entity, které také mohou být uvedeny v DTD.

Nakonec ještě jeden příklad pro ilustraci atributů id a idref, idrefs:

```
<?xml version="1.0" encoding="Windows-1250"?>

<!DOCTYPE rodina [
<!ELEMENT rodina (osoba)*>
<!ELEMENT osoba (jméno)>
<!ELEMENT jméno (#PCDATA)>
<!ATTLIST osoba id ID #REQUIRED
      matka IDREF #IMPLIED
      otec IDREF #IMPLIED
      děti IDREFS #IMPLIED>
]>

<rodina>
<osoba id="Marie" děti="Hana Petr">
      <jméno>Marie Nová</jméno>
</osoba>
```

```
<osoba id="Jan" děti="Petr Hana">
      <jméno>Jan Nový</jméno>
</osoba>
<osoba id="Hana" matka="Marie" otec="Jan">
      <jméno>Hana Nová</jméno>
</osoba>
<osoba id="Petr" matka="Marie" otec="Jan">
      <jméno>Petr Nový</jméno>
</osoba>
</rodina>
```

5.2.5 XML Schema

XML Schema je další variantou technologie umožňující validaci XML dokumentu. V průběhu vývoje vzniklo v Microsoft několik verzí (MSXML 2.0 - XML-Data Reduced (XDR) schemas, MSXML 4.0 - XML Schema definition language (XSD) schemas, MSXML 5.0 - inline XSD schemas, …) a také W3C XML schéma, což je podle W3C doporučení. XML Schemata specifikují strukturu dokumentu podobně jako DTD, ale využívají syntaxe XML a podporují více datových typů. Podrobný popis detailů sahá za rámec tohoto textu. Ukažme možnosti schémat na typických ilustračních příkladech.

- Jedním ze způsobů, jak se odkázat na soubor schémat je pomocí atributů xsi:schemaLocation nebo xsi:noNamespaceSchemaLocation.

Př.:

Soubor nn-valid.xml:

```
<?xml version="1.0"?>
<catalog>
   <book xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:noNamespaceSchemaLocation="nn.xsd"
         id="bk101">
      <author>Gambardella, Matthew</author>
      <title>XML Developer's Guide</title>
      <genre>Computer</genre>
      <price>44.95</price>
      <publish_date>2000-10-01</publish_date>
      <description>An in-depth look at creating applications with
      XML.</description>
   </book>
</catalog>
```

Soubor nn.xsd:

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="book" type="bookdata"/>
  <xsd:complexType name="bookdata">
    <xsd:sequence>
      <xsd:element name="author" type="xsd:string"/>
      <xsd:element name="title" type="xsd:string"/>
      <xsd:element name="genre" type="xsd:string"/>
      <xsd:element name="price" type="xsd:float"/>
      <xsd:element name="publish_date" type="xsd:date"/>
      <xsd:element name="description" type="xsd:string"/>
    </xsd:sequence>
    <xsd:attribute name="id" type="xsd:string"/>
  </xsd:complexType>
</xsd:schema>
```

- V XML dokument s obsahem XML Schéma je kořenový element <schema> (případně i se jmenným prostorem). XML schémata většinou nabízejí více možností, jak něco deklarovat. V XML schema jsou značky dvou druhů: deklarace (popis obsahu) a definice (vytváří nové typy). Deklarace elementu element1, který obsahuje pouze textový řetězec vypadá například takto:

```
<element name="element1" type="string"/>
```

Pokud má element obsahovat ještě nějaké další vnořené elementy, uvedou se jako součást definice typu elementu:

```
<element name="element1">
  <complexType>
    <element name="element11" type="string"/>
    <element name="element12" type="string"/>
    <element name="element13" type="string"/>
  </complexType>
</element>
```

Pokud by vnořené elementy byly složitějšího typu a obsahovaly další elementy, použil by se v definici elementu pouze odkaz na jejich definici:

```
<element name="element2">
  <complexType >
    <element ref="element1"/>
  </complexType >
</element> …
```

Počet opakování nějakého elementu určují atributy *minOccurs* a *maxOccurs*, které definují minimální a maximální počet výskytů. Použitelné jsou libovolné hodnoty, u maxOccurs můžeme uvést i hodnotu *, která zastupuje libovolný počet výskytů:

```
…     <element ref="element1" minOccurs="1" maxOccurs="*"/>   …
```

Element se smíšeným obsahem používá přídavný atribut u definice typu elementu:

```
<element name="element1">
  <complexType mixed="true">
    <element name="element11" type="string"/>
…
  </complexType>
</element>
```

Obvykle se mají elementy v dokumentu vyskytovat v pořadí, které specifikujeme ve schématu. Pokud chceme deklarovat výskyt elementů v libovolném pořadí, použijeme element <all>:

```
<element name="element1">
  <complexType>
    <all>
      <element name="element11" type="string"/>
      <element name="element12" type="string"/>
    </all>
  </complexType >
</element>
```

Při použití elementu <*sequence*> se musí elementy v dokumentu vyskytovat ve stejném pořadí jako ve schématu, při použití <*choice minOccurs="1" maxOccurs="1"*> *se* smí vyskytnout jen jeden z elementů definovaných ve skupině. Element <*group*> definuje elementy v odkazované skupině. Skupiny se do sebe vzájemně mohou zanořovat.

Syntaxe elementu:

<xsd:element name="jméno" **type="typ"** minOccurs="int" maxOccurs="int"/>, nebo

<xsd:element name="jméno" minOccurs="int" maxOccurs="int">

<xsd:complexType>

<xsd:sequence>
.../...
</xsd:sequence>

…

</xsd:complexType>

</xsd:element>

Definice atributů se provádí pomocí elementu *attribute*, který je součástí definice typu elementu. Pro ilustraci v MSXML 4.0 - XML Schemas je syntaxe:

<atribute default = string fixed = string form = (qualified | unqualified) id = ID
name = NCName ref = QName type = QName use = (optional | prohibited | required):
optional {any attributes with non-schema Namespace…}>
Content: (annotation?, (simpleType?))
</attribute>

Jednoduchý příklad:

```
<element name="element1">
  <complexType>
    <element ref=" element2"/>
    <attribute name="atribut1" type="string" />
    <attribute name="atribut2" type="date"/>
  </complexType>
</element>
```

- Datové typy

Využití datových typů jsou v XML schématech je velice široké. Kromě atributů můžeme typ určit i u elementů. Typy mohou být pojmenované i anonymní, jednoduché i komplexní. K dispozici jsou základní datové typy jako celá čísla, desetinná čísla, textové řetězce, datum, čas, logická hodnota nebo internetová adresa, atd. Od těchto typů je možné odvozovat vlastní typy. Mezi nejjednodušší způsob odvození typu patří určení minimální a maximální délky textových řetězců a určení intervalu, rozsahu a počtu desetinných míst pro čísla.

Př.:

```
<datatype name="odměna" source="decimal">
  <minInclusive value="10"/>
  <maxInclusive value="1000"/>
  <scale value="2"/>
</datatype>
```

U typů odvozených od textových řetězců můžeme pomocí regulárního výrazu definovat masku, které musí řetězec vyhovět. Můžeme také definovat přípustné hodnoty pomocí výčtu.

Příklad komplexního typu:

```
<complexType name="USAddress">
   <sequence>
      <element name="name" type="string"/>
      <element name="street" type="string"/>
      <element name="city" type="string"/>
      <element name="state" type="string"/>
      <element name="zip" type="decimal"/>
   </sequence>
   <attribute name="country" type="NMTOKEN" fixed="US"/>
</complexType>
```

Některé základní a odvozené datové typy (W3C):

string	"ahoj"
boolean	{true, false, 1, 0}
decimal	8.69
float	92.56E3, 15, 12560, 0, -0, INF, -INF, NAN
double	72.56E3, 12, 12560, 0, -0, INF, -INF, NAN
duration	P1Y2M3DT10H30M12.3S
dateTime	formát: *CCYY-MM-DDThh:mm:ss*
time	formát: *hh:mm:ss.sss*
date	formát: *CCYY-MM-DD*
gYearMonth	formát: *CCYY-MM*
gYear	formát: *CCYY*

gMonthDay	formát: *--MM-DD*
gDay	formát: *---DD*
gMonth	formát: *--MM--*
hexBinary	hex string
base64Binary	base64 string
URI	**http://www.ttt.com**
QName	jmenný prostor
NOTATION	NOTATION z XML specifikace
IDREFS	
ENTITIES	
NMTOKEN	
NMTOKENS	
Name	
NCName	
ID	
IDREF	
ENTITY	
integer	
nonPositiveInteger	
negativeInteger	
long	
int	
short	
byte	
nonNegativeInteger	
unsignedLong	
unsignedInt	
unsignedShort	
unsignedByte	
positiveInteger	
…	

- Pro další ilustraci následuje upravený příklad konverze DTD na XML schema z tutoriálu na www.w3.org:

DTD:

```
<!ELEMENT Knihovna (Kniha+)>
<!ELEMENT Kniha (Titul, Autor, Datum, ISBN, Nakladatel)>
<!ELEMENT Titul (#PCDATA)>
<!ELEMENT Autor (#PCDATA)>
<!ELEMENT Datum (#PCDATA)>
<!ELEMENT ISBN (#PCDATA)>
<!ELEMENT Nakladatel (#PCDATA)>
```

XML schema(`knihovna.xsd`**):**

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                    targetNamespace="http://www.knihy.org"
                    xmlns="http://www.knihy.org"
                    elementFormDefault="qualified">
    <xsd:element name="Knihovna">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element ref="Kniha" minOccurs="1" maxOccurs="unbounded"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="Kniha">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element ref="Titul" minOccurs="1" maxOccurs="1"/>
                <xsd:element ref="Autor" minOccurs="1" maxOccurs="1"/>
                <xsd:element ref="Datum" minOccurs="1" maxOccurs="1"/>
```

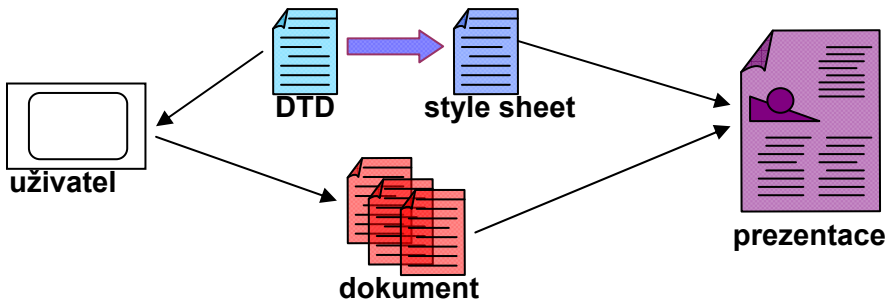
```
                <xsd:element ref="ISBN" minOccurs="1" maxOccurs="1"/>
                <xsd:element ref="Nakladatel" minOccurs="1" maxOccurs="1"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="Titul" type="xsd:string"/>
    <xsd:element name="Autor" type="xsd:string"/>
    <xsd:element name="Datum" type="xsd:string"/>
    <xsd:element name="ISBN" type="xsd:string"/>
    <xsd:element name="Nakladatel" type="xsd:string"/>
</xsd:schema>
```

XML dokument:

```
<?xml version="1.0"?>
<Knihovna xmlns ="http://www.knihy.org"
                    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                    xsi:schemaLocation=" … knihovna.xsd">
        <Kniha>
                <Titul>Babička</Titul>
                <Autor>Božena Němcová</Autor>
                <Datum>1958</Datum>
                <ISBN>1111-2222-3333</ISBN>
                <Nakladatel>Albatros</ Nakladatel >
        </Kniha>
        ...
</Knihovna >
```

5.2.6 Kaskádové styly

Použití kaskádových stylů je jednou z prvních možností, jak oddělit vzhled a obsah HTML stránky, později i u XML dokumentu. Vzhled klíčových elementů v HTML dokumentu je definován odděleně od kódu dokumentu. Jednotného a lehce modifikovaného vzhledu typické webové aplikace dosáhneme sdílením stylu více stránkami, dokumenty. Historicky v roce 1996 vznikla specifikace CSS1, od roku 1998 potom CSS2. Základní princip použití externích kaskádových stylů v kontextu například XML uživatelských dat pro prezentaci ukazuje následující obrázek.



Obrázek 23 Použití kaskádových stylů

Některé aplikace požadují zachování věrnosti zobrazovaného dokumentu a originálu (přesné druhy písma, barvy, zarovnání a formátování). Jednou možností je specifikace stylu přímo u každého vhodného elementu. Připojí se atribut style a v něm se přímo deklaruje styl.

Př.: Odstavec bude červeně, zarovnaný vpravo.

Ke snadnému použití kaskádových stylů přispívá děděním vlastností ve vnořených elementech dokumentu. Pokud vlastnost u elementu definujeme a tento element obsahuje další elementy, automaticky tyto elementy dědí vlastnosti rodičovského elementu.

Př. .. p { color: red }…

červený odstavec taky červené

Pro některé elementy a vlastnosti v HTML dokumentu není dědění účelné, dědičnost je potlačena. K jednomu dokumentu můžeme připojit více stylů. Obecně platí, že pozdější deklarace mají vyšší prioritu. Styly se skládají celkem složitým způsobem, který je přesně popsán ve specifikaci CSS.

Další možností aplikace stylu je použití specifikací pomocí selektorů. Selektory umožňují efektivním způsobem vybrat elementy, na které bude aplikován styl. V dalších verzích CSS jsou možnosti selektorů postupně rozšiřovány, aby šlo výhody kaskádových stylů v plné šíři použít i s jazykem XML, případně SGML. Nejjednodušším selektorem je jméno elementu. Pokud potřebujeme, aby se pro více elementů použila stejná deklarace stylu, jména elementů v selektoru oddělíme čárkou. Pokud potřebujeme aplikovat styl na element, ale ne ve všech jeho výskytech, přidáme atribut class se společným identifikátorem. Ve stylu se pak na všechny elementy s touto třídou odvoláme pomocí selektoru .identifikátor_třídy.

Př. **.redp** { color: red }

běžný odstavec

<p **CLASS="redp"** >červený odstavec

Pokud chceme rozlišit výskyt třídy u určitého elementu, použijeme zápis element.třída. Následující pravidlo se tak aplikuje pouze na elementy div, u kterých je obsah atributu class nastaven na hodnotu **souhrn**

div.**souhrn** { color: yellow }

U každého elementu můžeme použít atribut id a přiřadit mu identifikátor. V selektoru se pak na tento element odvoláváme pomocí zápisu #id nebo podobně jako v minulém případě element#id.

Kontextové selektory umožňují definici pravidel, která se budou aplikovat pouze na elementy v určitém kontextu. Kontextem je příslušnost k určitému rodičovskému elementu. Pokud chceme, aby ve všech číslovaných seznamech, které jsou součástí dalšího seznamu (tj. jsou obsaženy v elementu li), bylo použito menší písmo, můžeme použít kontextový selektor li ol.

li ol { font-size: smaller }

Můžeme samozřejmě používat i složitější hierarchie, které obsahují více elementů. Kromě názvů elementů můžeme v jednotlivých částech kontextového selektoru používat jako selektory i třídy a identifikátory elementů. Všechny způsoby lze tedy navzájem velice dobře kombinovat. Pokud budeme chtít v souhrnu označeném pomocí ... mít citace kurzívou, můžeme použít pravidlo:

.souhrn blockquote { font-style: italic }

Nejjednodušší kaskádový styl je tvořen jednoduchými pravidly. Každé pravidlo má dvě části – selektor a deklaraci. Selektor určuje elementy, na které bude deklarace aplikována. Deklarace se skládá ze dvou částí (oddělených dvojtečkou) – z vlastnosti a její hodnoty. Deklarace můžeme sdružovat dohromady, oddělovačem je středníku.

Příklad pravidla:

h1 { color: blue; text-align: center }

Připojení stylu z externího souboru umožňuje využití jednoho stylu několika stránkami. Styl se k dokumentu připojuje pomocí elementu link, který je umístěn v záhlaví stránky.

```
…  <head> …
   <link href="styl.css" type="text/css" rel="StyleSheet">
… </head>
```

Styl pro celou stránku zapsaný přímo v HTML dokumentu se vkládá do záhlaví dokumentu do elementu style. Pomocí atributu type povinně určíme typ používaného stylového jazyka. Často se definice stylu uzavírá do HTML komentáře, aby nebyla chybně vyhodnocena prohlížeči bez korektní podpory kaskádových stylů.

Př.

```
… <head>…
   <style type="text/css">
     <!--     pravidla definice stylu ...
     -->
   </style>…
 </head> …
```

Při použití společně s XML nabízejí totiž selektory větší možnosti než ve spojení s jazykem HTML.

Tabulka 1. Příklady nejpoužívanějších selektorů

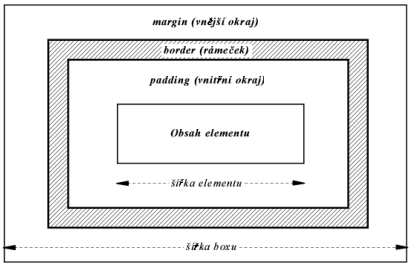
Selektor	Popis
*****	Vybere všechny elementy.
Element1	Vybere všechny elementy **Element1**.
Element1 item	Vybere všechny elementy **item**, které jsou potomkem elementu **Element1**.
Element1 > element1child	Vybere všechny elementy **element1child**, které jsou dětmi elementu **Element1**
Element1:*first-child*	Vybere element **Element1**, pokud je to první dítě svého rodiče.
Element1 + Element2	Vybere všechny elementy **Element2**, které bezprostředně následují za elementem **Element1**.
Element1 [atrib1]	Vybere všechny elementy **Element1**, které mají nastaven atribut **atrib1** na libovolnou hodnotu.
Element1 [atrib1="hod1"]	Vybere všechny elementy **Element1**, které mají atribut **atrib1** nastaven na hodnotu **hod1**.
Element1 [atrib1~="hod1"]	Vybere všechny elementy **Element1**, které mají v atributu **atrib1** seznam hodnot oddělených mezerami a jednou z těchto hodnot je **hod1**.
Element1 #idHod1	Vybere element **kapitola**, který má ID nastaveno na **idHod1**.

Pro každý element v dokumentu se pak navzájem zkombinují všechna pravidla, která mu podle selektorů vyhovují. Díky této vlastnosti lze styly psát velice rychle. Některé vybrané možnosti definice vlastností:

- Písmo - můžeme definovat styl písma (normální, kurzíva, skloněné), varianty písma (normální, kapitálky), velikost, … Př.:

```
BLOCKQUOTE { font-weigth: bold;
font-style: italic;
font-size: 14pt;
line-height: 16pt;
font-family: "Times New Roman", cursive
}
```

- barva textu, barva pozadí, obrázek na pozadí (opakování, souřadnice umístění, způsob rolování)
- Formátování textu - způsob zarovnání (doleva, doprava, centrování, do bloku), řádkování, vertikální zarovnání objektů na řádce, velikost odstavcové zarážky, …
- Formátovací model - u všech vnějších a vnitřních okrajů a u rámečku lze nastavit přesné rozměry, u rámečku i barvu a tvar. způsob zobrazení elementu je definován pomocí parametru display (block, inline, list-item, none)



Pro definici vzhledu XML dokumentů se používají různé stylové jazyky. Mezi tři nejznámější a nejpoužívanější patří jazyky CSS, XSL a DSSSL. CSS (Cascading Style Sheets) je stejný jako kaskádové styly používané v HTML. Kaskádové styly byly původně určeny pouze pro prezentování informací na obrazovce. Další parametry formátování, které je potřeba zadat pro další typy výstupu (tisk, mobilní zařízení, …) , byly přidány do nové verze kaskádových stylů CSS2, která je zpětně kompatibilní s CSS1. Pro potřeby XML byly do CSS2 přidána rozšíření, jako nové selektory se silnější podporou XML, nové možnosti pro přesné umístění elementů na stránce, stránkový výstup(nastavení okrajů na stránce zvlášť pro liché, sudé nebo první, řízení stránkového zlomu, …), formátovací model pro tabulky, možnost aplikovat konkrétní styl i na základě hodnoty atributu (CSS1 umožňovalo styl aplikovat pouze na určitý element bez návaznosti na obsah jeho atributů). Dále podpora několika výstupních médií parametrizací přímo ve stylu:

```
@media print {
  BODY { font-size: 8pt }
}
@media screen {
  BODY { font-size: 10pt }
}
@media screen, print {
  BODY { line-height: 1.2 },
}
```

nebo připojení několika různých stylů:

```
<link rel="stylesheet" type="text/css" href="s0.css" media="all">
<link rel="stylesheet" type="text/css" href="s1.css" media="screen">
<link rel="stylesheet" type="text/css" href="s2.css" media="print">
```

Protipólem k CSS je jazyk DSSSL (Document Style Semantics and Specification Language). Je to rozsáhlý jazyk, který byl původně vyvinut pro použití se SGML. Pro svou komplexnost jej zatím nepodporuje příliš mnoho aplikací.

Příklad dokumentu XML:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/css" href="booklist.css" media="screen"?>
<BOOKLIST>
   <ITEM>
      <TITLE>Instant JavaScript</TITLE>
      <DESCRIPTION>
         <CATEGORY>Category: </CATEGORY>
         <CODE>(16-048)</CODE>
      </DESCRIPTION>
      <DESCRIPTION>
         <RELEASE_DATE>Release date: 1998-04-21</RELEASE_DATE>
         <PRICE>Price: $49.34</PRICE>
      </DESCRIPTION>
   </ITEM>
   <ITEM>
      <TITLE>Instant HTML</TITLE>
      <DESCRIPTION>
         <CATEGORY>Category: HTML</CATEGORY>
         <CODE>(16-041)</CODE>
      </DESCRIPTION>
      <DESCRIPTION>
         <RELEASE_DATE>release date: 1998-03-07</RELEASE_DATE>
         <PRICE>Price: $34.23</PRICE>
      </DESCRIPTION>
   </ITEM>
</BOOKLIST>
```

A obsah stylového souboru booklist.css:

```
@media screen, print
{
   ITEM
   {
      display: block;
      margin-left: 40pt;
      font-family: Times New Roman;
      font-size: 12pt;
      font-weight: 500;
      margin-bottom: 15pt;
      text-align: left;
      line-height: 12pt;
      text-indent: 0pt;
   }

   TITLE
   {
      display: block;
      font-family: Arial;
      font-weight: 700;
      font-size: 14pt;
   }

   DESCRIPTION
   {
      display: block;
   }

   CATEGORY
   {
      display: inline;
   }
```

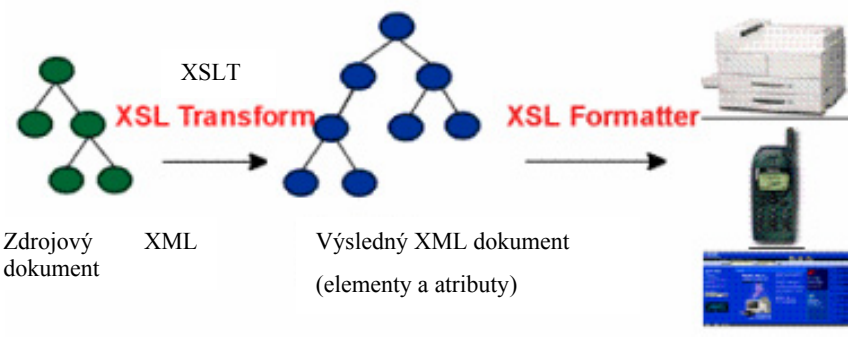
```
    CODE
    {
       display: inline;
    }

    RELEASE_DATE
    {
       display: inline;
    }

    PRICE
    {
       display: inline;
    }
}
```

5.2.7 XSL

V mnoha aplikacích je výhodné nebo nutné před prezentací (zobrazením, vytištěním) dokument modifikovat, například vyfiltrovat obsah, změnit strukturu apod. K tomuto účelu se hodí jazyk XSL (eXtensible Stylesheet Language), který vznikl speciálně pro potřeby jazyka XML. XSL má možnosti srovnatelné s DSSSL, jeho syntaxe je však mnohem jednodušší. Pro zápis stylu se využívá jazyk XML, takže lze použít běžné XML editory. XSL umožňuje dvě základní funkce - transformovat jeden XML dokument do jiného XML dokumentu a specifikovat formátovací vlastnosti jednotlivých částí dokumentu. Možnost transformovat jeden dokument do druhého se dnes využívá nejčastěji. Pro účel prezentace se jednotlivé XML elementy mapují do HTML kódu, který specifikuje výsledné zobrazení. Následující obrázek je převzat z popisu standardu Extensible Stylesheet Language (XSL) Version 1.1.



- Připojení stylu k dokumentu

Kaskádové styly mohly být zapisovány přímo do HTML dokumentu. V XML tato možnost není, styl musí být definován v externím souboru. Připojení stylu k dokumentu je nezávislé na použitém stylovém jazyce a provádí se pomocí speciální instrukce pro zpracování xml-stylesheet. Připojení stylu může vypadat například takto:

```
<?xml version="1.0" encoding="utf-8"?>
<?xml-stylesheet href="styl.xsl" type="text/xsl"?>
<dokument>
   ...
</dokument>
```

Pomocí atributu href je určena URL adresa (může být i relativní jako v našem příkladě), na které je uložen soubor se stylem. Atribut type určuje MIME typ použitého stylového jazyka. Pokud používáme XSL, použijeme type="text/xsl".

Standard typu MIME(Mult-ipurpose Internet Mail Extensions nebo Multi-purpose Internet Multimedia Extensions - podle kontextu) umožňuje přenášet prostřednictvím služeb sítě Internet zprávy, v jejichž těle jsou i jiné než čisté ASCII texty.

Např. text/html, text/xhtml, image/png

- Struktura

XSL bsahuje dvě samostatné části - XSLT a FO. Pracuje na jiném principu než kaskádové styly. Transformačním jazykem XSLT vytváříme pravidla obecně pro transformaci jedné skupiny XML dokumentů na výsledný XML dokument, který je obvykle použit pro následné zobrazení.

Dnes se stále nejčastěji XSL styl aplikuje tak, aby výsledkem transformace byl HTML dokument vytvořený ještě na web-serveru, a tak zpřístupněný libovolnému prohlížeči. Formátovací objekty (FO) vychází z jazyků CSS a DSSSL a popisují abstraktní vzhled dokumentu s využitím bohatého formátovacího slovníku. Interpretace FO slouží pro zobrazí na obrazovce, převedení do PS, PDF, atd., pomocí jmenného prostoru fo a jeho fo: typů elementů. Při formátování se generuje strom geometrických oblastí (area tree), které se mapují do stránek výstupního dokumentu. V první verzi se jedná o obdélníkové oblasti, nejčastěji se používají elementy jako *block,* s atributy *text-align,* …. Rozlišují se ale čtyři kategorie oblastí - container, block, line, inline. Container používá souřadný systém pro umístění vnořených objektů (s použitím top, botton, left, right). Block-oblast může obsahovat několik block-oblastí nebo line-oblastí. Block element se používá k ohraničení textu:

```
<fo:block>blok textu</fo:block>
<fo:block>Dalčí blok</fo:block>
```

Seznam je sekvence elementů <fo:list-item>:

```
<fo:list-block>
    <fo:list-item>…</fo:list-item>
    <fo:list-item>…</fo:list-item>
</fo:list-block>
```

Tabulka má syntaxi podobnou HTML:

```
<fo:table>
    <fo:table-header>…</fo:table-header>
    <fo:table-footer>…</fo:table-footer>
    <fo:table-body>…</fo:table-body>
</fo:table>
<fo:table-body>
    <fo:table-row>…</fo:table-row>
</fo:table-body>
<fo:table-row>
    <fo:table-cell>…</fo:table-cell>
</fo:table-row>
```

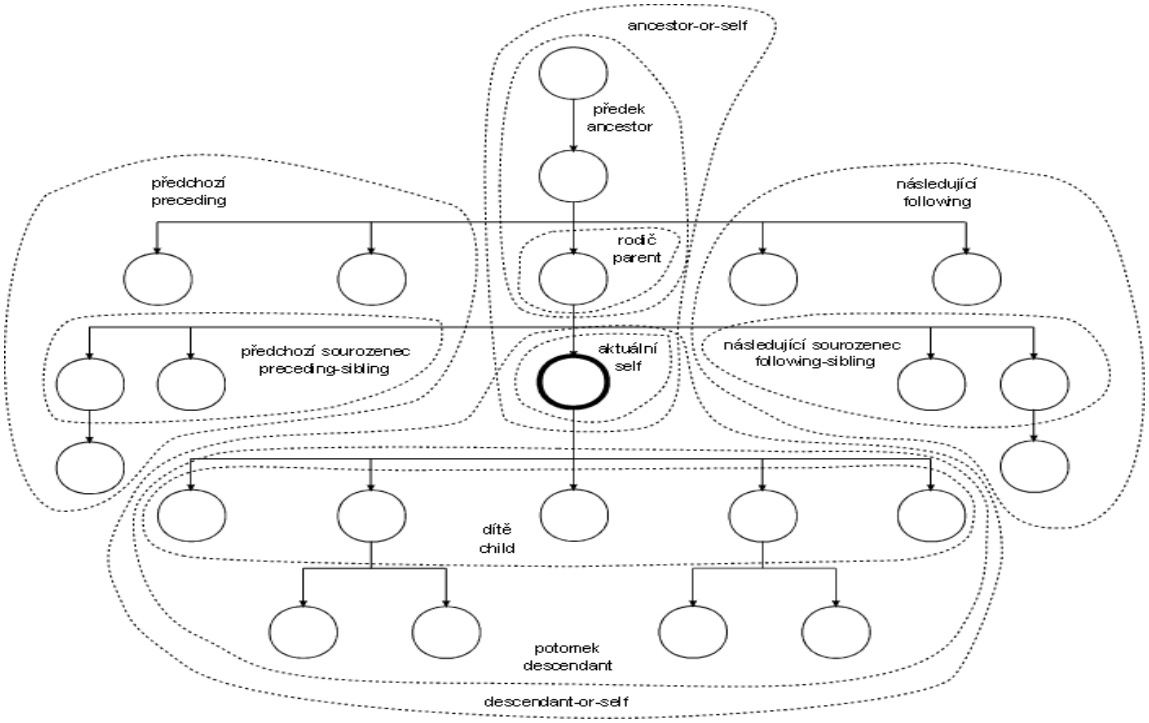
Line-oblast může obsahovat inline-oblasti.

XSLT je standardem W3C od roku 1999. Je to programovací jazyk pro zpracování XML dat, s několika datovými typy (boolean, number, string, externí objekty, …), operacemi (<xsl:template>, <xsl:apply-templates>, <xsl:sort>, <xsl:output>, …) a možností větvení programu(<xsl:if>, <xsl:for-each>, <xsl:choose>, …). Ve výrazech podporuje syntaxi jazyka XPath. Proces zpracování obsahuje mimo jiné načtení XML a XSL dokumentů, syntaktickou analýzu a vytvoření odpovídající stromové struktury XML i XSL dokumentů, aplikaci transformačních pravidel a vytvoření výsledného stromu s možností výstupu ve vhodném formátu.

XSL obsahuje převážně šablony, které určují, jak se budou jednotlivé části dokumentu zdrojového dokumentu transformovat. Části dokumentu jsou v šablonách lokalizovány pomocí atributu *match* s hodnotou, která je výrazem jazyka XPath. Dále je možné používat selekční podmínky, cykly, proměnné, funkce, třídění části XML dokumentu. XSLT procesor nepracuje přímo s fyzickou reprezentací XML dokumentu v souboru, ale operuje nad abstraktním modelem dokumentu, který si obvykle vytvoří v paměti. Model vychází se samostatné abstraktní specifikace informace v dokumentu - XML Infoset (W3C doporučení).

5.2.7.1 XPath

XPath je jednoduchý dotazovací jazyk nad stromovou reprezentací dokumentu se širokým využitím také například v XPointeru, XML schématech atd. Každý uzel ve stromové reprezentaci dokumentu je určitého typu a má obsah, který tvoří text, obsažený v uzlu. Pro nelistové elementy se zřetězí textové uzly jeho potomků. Uzly stromu mohou být sedmi typů – kořenové, uzly elementů, textové, atributové, uzly jmenných prostorů, uzly prováděcích instrukcí a komentářů. Kořenový uzel neodpovídá žádnému elementu v dokumentu. Jeho dětský následník je kořenový element dokumentu, variantně i další uzly jako komentáře, instrukce pro zpracování. Jako dětské uzly mohou být rekurzívně připojeny další uzly, reprezentující elementy, atributy, jmenné prostory, textové uzly, komentáře nebo instrukce pro zpracování. Element, ke kterému je uzel připojený, se chápe jako jeho rodič. Hodnotou uzlu je zřetězení všech následných textových uzlů, řazených podle výskytu v dokumentu. Atribut není považován za dětský uzel. Textový uzel je listový uzel bez názvu s obsahem, který odpovídá textovému obsahu, ve kterém se nahradí výskyty všech entit skutečným textem.



Výrazy se tvoří pomocí operátorů a speciálních znaků. Některé jsou podobné zápisu cest ve struktuře adresářů. Za výraz lze do hranatých závorek zapisovat podmínku, která výběr zúží. Pokud jako podmínku zapíšeme číslo, vybere se element s daným pořadím.

Výrazy pro výběr části dokumentu pomocí cesty se tvoří pomocí stanovených kritérií a vzájemné pozice uzlů – pomocí identifikátorů osy pohybu ve stromu dokumentu. Východiskem vyhledávání může být:

- aktuální uzel (relativní cesta).V šabloně je aktuálním uzlem ten, pro který byla šablona aktivována. Uvnitř cyklů je aktuálním uzlem právě zpracovávaný uzel.
- kořen (absolutní cesta). XPath umožňuje přesun se na určité místo ve stromu dokumentu prostřednictvím funkcí, jako id() …vybere uzel s daným ID. Každá část cesty se skládá z několika komponent: určení směru pohybu od aktuálního uzlu , testu uzlu na základě jejich typu i názvu a predikátů filtru.

Syntaxe identifikátorůOsy ::= 'ancestor' | 'ancestor-or-self' | 'attribute' | 'child' | 'descendant' | 'descendant-or-self' | 'following' | 'following-sibling' | 'namespace' | 'parent' | 'preceding' | 'preceding-sibling' | 'self'. Význam položek :
ancestor:: - rodič aktuálního uzlu a všichni jeho další předci až ke kořenu stromu dokumentu.
ancestor-or-self:: - aktuální uzel a všichni jeho další předci až ke kořenu stromu dokumentu.
attribute:: - uzly odpovídající všem připojeným atributům.
child:: - všechny dětské uzly aktuálního uzlu ve.
descendant:: - potomci aktuálního uzlu.
descendant-or-self:: - aktuální uzel a všichni jeho potomci.
following:: - uzly následující za koncem aktuálního uzlu.
following-sibling:: - následující uzly, které jsou sourozenci aktuálního uzlu.
namespace:: - uzly odpovídajících jmenných prostorů.
parent:: - rodič aktuálního uzlu.
preceding:: - uzly se svými potomky, které jsou před aktuálním uzlem.
preceding-sibling:: - předcházející uzly, sourozencem aktuálního uzlu.
self:: - aktuální uzel.
Microsoft ve své implementaci nabízí následující syntaxi pro tvorbu výrazů.

`/`	Operátor bezprostředního následníka(dítě), na začátku šablony se jedná o následníka kořenového uzlu.
`//`	Rekurzivně generovaný následník, hledaný v libovolné urovni podstromu.), na začátku šablony se jedná o rekurzivního následníka kořenového uzlu.
`.`	aktuální kontext.
`..`	Rodičovská úroveň aktuálního kontextu.
`*`	Výběr všech elementů, nezávisle na jménu.
`@`	Atribut; předpona pro jméno atributu.
`@*`	Výběr všech atributů, nezávisle na jménu.
`:`	Separátor jmenného prostoru elementu nebo atributu
`( )`	Sdruží operace pro explicitní vyjádření pořadí.
`[ ]`	aplikace filtračního vzoru.
`[ ]`	Popis indexů.
`+`	sčítání.
`-`	odčítání.
`div`	dělení.
`*`	násobení.
`mod`	Operace modulo.

priorita	**znak**	**účel**
1	`( )`	sdružení
2	`[ ]`	filtry
3	`/ //`	Zápis cest

Syntaxe pro volání funkcí ::= jménoFunkce '(' (Argument (',' Argument)*)? ')'

Argument ::= Výraz

Následuje stručný přehled vybraných funcí.

Funkce pro práci s uzly:

last() - pozice posledního uzlu mezi kontextovými uzly.
position() - pozice aktuálního uzlu mezi kontextovými uzly.

id() - uzel s požadovaným ID.
local-name() - lokální část názvu uzlu
name() - název uzlu.

Řetězcové funkce

string() - převod objektu na řetězec.
concat() - zřetězení řetězců.
starts-with - test řetězce, zda na začátku obsahuje hledaný řetězec.
Contains - test, zda řetězec obsahuje hledaný podřetězec.
substring-before - část řetězce, která se vyskytuje před zadaným řetězcem
substring-after - část řetězce, která se vyskytuje za zadaným řetězcem.
Substring - část řetězce.
string-length() - počet znaků řetězce.
normalize-space() - odstraní z řetězce přebytečné mezery.
translate() - jednoduchý překlad znaků v řetězci (znaky prvního řetězce, které jsou obsaženy v druhém parametru funkce, jsou nahrazeny znakem na stejné pozici ve třetím řetězci).

Logické funkce

boolean() - převod libovolné hodnoty na logickou hodnotu.
not() - negace výrazu.
true() - vrací hodnotu true.
false() - vrací hodnotu false.
lang() - nastavení jazyka pomocí atributu xml:lang.

Funkce pro práci s čísly

number() - převod objektu na číselnou hodnotu.Pokud je v řetězci za případnými mezerami číslo, vrátí se toto číslo, v opačném případě vrací funkce hodnotu NaN („not a number“), logická hodnota true je převedena na 1, false na 0.
sum() - součet hodnot uložených v množině uzlů.
floor() - zaokrouhlení čísla na celé číslo dolů.
ceiling() - zaokrouhlení čísla na celé číslo nahoru.
round() - zaokrouhlení čísla na nejbližší celé číslo.

Rozšiřující funkce definované v XSLT

document() - zpřístupnění uzlů z dalších dokumentů.
key() - nalezení uzlů s určitou hodnotou klíče.
current() - uzel, který byl aktuální v okamžiku začátku vyhodnocování výrazu.
generate-id() - generuje k uzlu jedinečný identifikátor.
system-property() - informace o právě používaném XSLT procesoru.

Ilustrace výrazů XPath na příkladech ukazuje následující tabulka.

`x/y[1]`	První dětský element `<y>` každého elementu `<x>`.
`x/y[position() = 1]`	První dětský element `<y>` každého elementu `<x>`.
`(x/y)[1]`	První dětský element `<y>` z celé množiny dětský elementů `<y>` elementů `<x>`.
`x[1]/y[2]`	Druhý dětský element `<y>` prvního elementu `<x>`.
`kniha[last()]`	Poslední element `<kniha>` aktuálního kontextu.
`kniha/autor[last()]`	Poslední element `<autor>` následník všech elementů `<kniha>` aktuálního kontextu.

`(kniha/autor)[last()]`	Poslední element <autor> celé množiny dětských elementů <autor>, dětských elementů <kniha> aktuálního kontextu
`kniha[výňatek]`	Všechny elementy <kniha> obsahující element < výňatek > jako dětský element.
`kniha[výňatek]/titul`	Všechny dětské elementy <titul> elementů <kniha> obsahující také dětský element < výňatek >.
`kniha[výňatek]/autor[titul]`	Všechny elementy <autor> obsahující dětský element <titul> a jsou dětským elementem <kniha> a také obsahují element < výňatek >.
`kniha[autor/titul]`	Všechny elementy <kniha> obsahující dětský element <autor> , který obsahuje dětský element <titul>.
`autor[titul][odměna]`	Všechny elementy <autor> obsahující dětský element <titul> a dětský element <odměna>.
`autor[(titul or odměna) and publikace]`	Všechny elementy <autor> obsahující element <titul> nebo <odměna> a alespoň jeden element <publikace> jako dětský element.
`autor[titul and not(publikace)]`	Všechny elementy <autor> obsahující dětský element <titul> a neobsahující dětský element <publikace> .
`autor[jméno = "Petr"]`	Všechny elementy <autor> obsahující dětský element <jméno> s hodnotou Petr.
`autor[jméno[1] = "Petr"]`	Všechny elementy <autor> kde první dětský element <jméno> má hodnotu Petr.
`autor[jméno [position()=1]= "Petr"]`	Všechny elementy <autor> kde první dětský element <jméno> má hodnotu Petr.
`titul[@od != "Harvard"]`	Všechny elementy <titul> jejichž atribut od se nerovná "Harvard".
`autor[. = "Petr Novák"]`	Všechny elementy <autor> jejichž hodnota je Petr Novák.
`autor[jméno = "Petr" and ../cena > 50]`	Všechny elementy <autor> obsahující dětský element <jméno> s hodnotou Petr a sourozenecký element <cena> s hodnotou větší než 50.
`kniha[position() <= 3]`	První tři elementy kniha (1, 2, 3).
`autor[not(jméno = "Petr")]`	Všechny elementy <autor> neobsahující dětský element <jméno> s hodnotou Petr.
`autor[* = "Petr"]`	Všechny elementy autor s libovolným dětským elementem, jehož hodnota je Petr.
`autor[jméno = "Petr" and příjmení = "Novák"]`	Všechny elementy <autor> obsahující dětský element <jméno> s hodnotou Petr a

	dětský element <příjmení> s hodnotou Novák.
cena[@měna = "Kč"]	Všechny elementy <cena> s atributem měna, jehož hodnota je "Kč".
titul[position() < 3]	První dva elementy <titul> které jsou dětskými elementy aktuálního kontextu
p/text()[2]	Druhý textový uzel elementu <p>.
ancestor::kniha[1]	Nejbližší předek <kniha>.
ancestor::kniha[autor][1]	Nejbližší předek <kniha> s dětským elementem <autor>.
ancestor::autor[parent::kniha][1]	Nejbližší předek <autor> který je dítětem elementu <kniha>.

5.2.7.2 Základy XSL

XSL je XML dokumentem, obsahuje instrukce pro XSLT procesor a značky výstupního formátu (HTML, FO, XML). K rozlišení značek se používají jmenné prostory. Před názvy všech elementů stylu, které souvisejí s XSL se uvede předpona xsl: . Použití jmenného prostoru je potřeba deklarovat u kořenového elementu stylu stylesheet:

```
<stylesheet xmlns="http://www.w3.org/XSL/Transform/1.0">
```

Kořenovým elementem může být variantně:

```
<transform xmlns="http://www.w3.org/XSLT/Transform/1.0">
```

Definice .xsl mohou být sdíleny i ve více souborech, např. :

```
<stylesheet …>
    <import href="tabulky.xsl">
    <import href="barvy.xsl">
    <template …>…</template> …
```

Nebo

```
<include href="…">…</include>
```

Nejdůležitější elementy:

-
```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">
... šablony ...
</xsl:stylesheet>
 <xsl:template match="XPath výraz">
...
</xsl:template>
```

- `<xsl:apply-templates/>` Element apply-templates XSL procesoru říká, aby obsah elementu postoupil dalším šablonám, které zpracují případně další elementy.

- `<xsl:value-of select="…"/>` vybere pouze text

- další konstrukty:

Jednoduchá podmínka	náhrada if-then-else	výběr z více variant
<xsl:if test="podmínka"> ... </xsl:if>	<xsl:choose> <xsl:when test="podmínka"> …	<xsl:choose> <xsl:when test="podmínka"> …

	</xsl:when> <xsl:otherwise> … </xsl:otherwise> </xsl:choose>	</xsl:when> <xsl:when test="podmínka"> … </xsl:when> <xsl:otherwise> … </xsl:otherwise> </xsl:choose>

Cykly iterace přes množinu uzlů	setřídění uzlů před zpracováním	číslování
<xsl:for-each select="XPath výraz"> … </xsl:for-each>	<xsl:for-each select="XPath výraz"> <xsl:sort select="výraz"/> … </xsl:for-each>	<xsl:number value="…" format="…"/>

Použití atributu select je velmi silný nástroj. Jeho vhodným použitím, můžeme části vstupního dokumentu vyřadit ze zobrazení nebo měnit pořadí elementů v transformovaném dokumentu. Následující příklad popisuje šablonu, ve které nejprve bude jméno autora, pak název článku bez ohledu na pořadí těchto atributů v sekvenci zdrojového XML dokumentu:

```
<xsl:template match="kniha">
  <xsl:apply-templates select="autor"/>
  <xsl:apply-templates select="nazev"/>
</xsl:template>
```

Příklad pro vyběr jen určitých elementů, které například mají určitou hodnotu svého atributu:

```
<template match="kniha">
    <apply-templates select="kniha[@typ='román'] " />
</template>
```

Na každý element dokumentu se aplikuje šablona, která má vyhovující vzor. Pokud je element možno zpracovat pomocí více šablon, má přednost ta poslední, nebo je pořadí určeno explicitně, pokud je definována priorita zpracování pomocí atributu *priority*.

Př.:

```
<template match="kniha//autor" priority = "2">
    …
</template>
```

Element <Sort> se používá v šabloně pro setřídění vybraných elementů:

```
<list>
    <item sortcode="1">xxx</item>
    <item sortcode="3">bbb</item>
    <item sortcode="2">kkk</item>
</list>

<template match="list">
   <apply-templates><sort/></apply-templates>
</template>

<sort select="@sortcode" />
```

Pro opakující se nebo podobný výstup šablon je možné odstranit redundanci použitím konstrukce s Variable, Value Of:

```
<variable name="barva">bílá</variable>

<html:h1>
    Barva byla <xsl:value-of select="$barva"/>.
<html:h1>
```

Nové elementy se vytvoří pomocí elementu <element> s atributem *name* a nepovinným *namespace*. Kopie zdrojového elementu se provede například:

```
<template match="Hlavička3">
    <element namespace="html" name="h3">
        <apply-templates/>
    </element>
</template>
```

Další možností je použití elementu <copy-of> s atributem select, který identifikuje elementy nebo části dokumentu, jež se zkopírují:

```
<template match="body">
    <body>
        <copy-of select="//h1 | //h2" />
        <apply-templates/>
    </body>
</template>
```

Komentáře a instrukce pro zpracování se vkládají do výstupního dokumentu pomocí zvláštních elementů:

```
<processing-instruction name="P1">INSERT_P1</processing-instruction>
```

a

```
<comment>Text poznámky</comment>
```

Podle specifikace XSL by měl mít každý XSL procesor zabudovány šablony, které způsobí vypsání celého dokumentu. To usnadní psaní stylů, protože pak stačí pomocí šablon ošetřit elementy, u kterých požadujeme speciální formátování. Na začátku každého XSL stylu bychom měli uvést definici následujících tří šablon:

```
<xsl:template match="/">
  <xsl:apply-templates/>
</xsl:template>

<xsl:template match="*">
  <xsl:apply-templates/>
</xsl:template>

<xsl:template match="text()">
  <xsl:value-of select="."/>
</xsl:template>
```

První šablona zajistí zpracování kořenového elementu dokumentu. Druhá šablona se postará o to, že budou zpracovány všechny elementy. Konečně poslední šablona do výsledného dokumentu vypíše textový obsah všech elementů na nejnižší úrovni. Díky tomu, že později uvedené šablony mají vyšší prioritu, neovlivní tyto tři šablony nijak další zpracování dokumentu

5.2.7.3 Šablony XSLT

Základem každého stylu jsou šablony. Jejich základní tvar je:

```
<xsl:template match="výraz">

   tělo šablony

</xsl:template>
```

Tělo šablony přitom přesně definuje, jak se části transformovaného dokumentu vyhovující výrazu budou zpracovávat. V těle šablony můžeme používat další konstrukce XSLT nebo přímo elementy z výsledného dokumentu.

Mezi dva nejpoužívanější příkazy, které se používají uvnitř šablony, patří *value-of* a *apply-templates*.

XSLT procesor na začátku své práce načte do paměti vstupní XML dokument a vytvoří si jeho stromovou reprezentaci. Tento strom je průchodem do hloubky procházen od kořene v pořadí v jakém jsou elementy obsaženy v dokumentu. V okamžiku, kdy je nalezena šablona odpovídající uzlu ve stromu, začne se její obsah zapisovat na výstup. Další potomci uzlu, pro který byla vybrána šablona, už nejsou dál automaticky zpracováváni. Na případné další vnořené výskyty (i stejného, jako je právě transformovaný element) elementu se šablona neaplikuje. Pokud tak chceme učinit, musíme uvnitř šablony použít instrukci <xsl:apply-templates/>. Ta říká, že se má daná větev stromu zpracovávat dále a mají se pro její uzly hledat odpovídající šablony. Např.

```
<template match="/|*">
    <apply-templates /> …
```

Pokud chceme v těle šablony použít jen textový obsah nějakého elementu a jeho podelementů, ale nechceme aplikovat další šablony, hodí se instrukce <xsl:value-of select="výraz"/>. Ta vybere pouze obsah textových uzlů, které jsou potomky elementu určeného pomocí výrazu (ten je opět zapsán pomocí syntaxe XPath).

V některých případech je jedno, kterou možnost použijeme. Budeme-li chtít, aby se název zobrazoval tučně (HTML tag <b>), máme dvě možnosti, jak toho dosáhnout.

```
<xsl:template match="nazev">
  <b><xsl:value-of select="."/></b>
</xsl:template>

<xsl:template match="nazev">
  <b><xsl:apply-templates/></b>
</xsl:template>
```

5.3 Dynamické stránky – klientské technologie

Standardní HTML stránky klasifikujeme jako statické, s neměnícím se obsahem. Při potřebě přizpůsobit obsah a formu stránky nějaké situaci – pro různé skupiny uživatelů, při výskytu nějaké události a reakci na ni, při zachycení nějakého procesu, akce a podobně využijeme technologií, které umožňují dynamicky modifikovat HTML stránky. Data pro takové změny často dodávají databázové zdroje. Změny mohou být vynuceny typem zobrazovacího zařízení (mobilní zařízení proti webovému prohlížeči), interakcí s uživatelem a podobně.

Jedna možných kategorizací technologií dynamických stránek je dělí na ty, které výkonnou část aplikace provádí přímo v prohlížeči a na takové, ve kterých výkonná část aplikace běží na web-serveru. Do první kategorie typicky patří technologie, jako DHTML (Dynamic HTML), využívající HTML typicky s klientskými skripty - často JavaScript ve spolupráci s objektovým modelem dokumentu DOM (Document Object Model), s kaskádními styly (CSS), variantně Java-applety, activeX komponenty a další technologie. Použití rozšiřujících klientských technologií urychluje odezvy a odlehčují zatížení web-serveru v případech, kdy je možné požadavky aplikace splnit na straně klienta – v prohlížeči. Nevýhodou je přetrvávající nekompatibilita jednotlivých prohlížečů (i k některým technologiím, závislým na použité platformě) a obecně nižší bezpečnost (neoprávněná manipulace s daty na počítači).

Princip použití klientských skriptů je jednoduchý. Přímo do HTML stránky lze zapsat nebo připojit jednoduché programy ve vhodném programovacím jazyce – uznávaným standardem a nejpodporovanějším v prohlížečích je JavaScript, ale některé prohlížeče podporují i VBScript, Perl, Tcl, Python, … . V prohlížeči je možno pomocí skriptu reagovat na načtení stránky, na události vyvolané uživatelem (většinou se pouze definují funkce, které jsou volány jako odezva na události), ale také manipulovat s dokumentem nebo prohlížečem.

Pro zapsání skriptu do dokumentu se využívá element <SCRIPT> s atributem LANGUAGE a obvykle se zapisuje do poznámky, aby nevznikly potíže v prohlížečích, které použitý jazyk nepodporují:

```
<SCRIPT LANGUAGE="JavaScript" TYPE="text/javascript">
<!--
program
// -->
</SCRIPT>
```

Příklad skriptu načteného do HTML dokumentu z externího souboru:

```
<SCRIPT LANGUAGE="JavaScript" TYPE="text/javascript" SRC="adresa souboru">
</SCRIPT>
```

U elementů v HTML dokumentu můžeme definovat několik atributů, které odpovídají

jednotlivám událostem a s hodnotou prováděcího kódu: <tag událost="obslužný kód">

Příklady dálostí, které lze obsloužit skriptem, jsou uvedeny v následující kapitole a JavaScriptu.

5.3.1 JavaScript

JavaScript byl vyvinutý firmou Netscape. Syntaxe je odvozena z Javy a C. Z hlediska IS JavaScript se často využívá při vstupní kontrole dat vkládaných do formulářů předtím, než jsou odeslány na web server. Existuje několik verzí jazyka - ECMAScript (standard), s rozšířením proti standardu potom JavaScript (Netscape) a JScript (implementace JavaScriptu od Microsoftu).

Proměnné mohou být typu number (integer nebo non integer), String, Boolean - (true, false) a Null. Jazyk není přísně typový – implicitní typ je String, ale typ proměnné se dá změnit, operace se provádí i s rozdílnými typy. Proměnné se nemusí deklarovat. Deklarace proměnné se uvozuje klíčovým slovem var. Identifikátor musí začínat písmenem nebo podtržítkem („_“), dále obsahuje i číslice. Rozlišují se velká a malá písmena, délka je podle potřeby.

var identifikátor = hodnota;

Dále je možné využít pole:

```
var barva = new Array(“červená”, “modrá”, “bílá”);        // barva[0]….červená,
                                                          //barva.length … 3
```

Komentář je definován znaky // .

JavaScript je jazyk založený na objektech, předdefinovaných i uživatelských. Používá se tečková notace pro přístup ke vnořeným objektům, atributům nebo metodám. Nejdůležitější skupina objektů patří do objektového modelu dokumentu a zpřístupňuje objekty stránky a prohlížeče. Základní a implicitní (při zápisu cesty se obvykle vynechává) objekt je window, který umožňuje přístup k prohlížeči.

Některé vlastnosti objektu Windows:

defaultStatus (implicitní zpráva na stavovém řádku)

frames (rámce zobrazené v prohlížeči)

length (počet rámců rodičovském okně)

name (jméno okna)

parent (rodičovské okno)

self (aktivní)

status (zpráva na stavovém řádku)

top (vrcholový objekt v hierarchii definované prohlížečem)

window (aktivní)

některé metody: alert() (modální) okno pro zobrazení zprávy.

confirm() (modální) okno pro výběr.

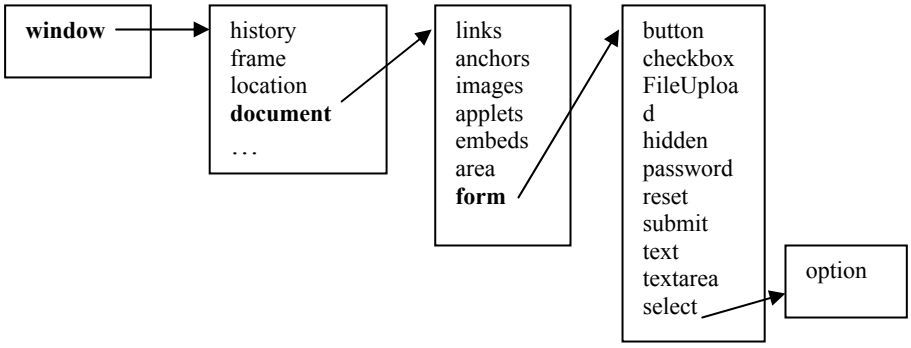
prompt() (modální) okno pro vstup dat.

clear() smazání obsahu.

open() otevření nového okna.

close() zavření aktuálního okna .

Objektový model dokumentu (DOM) uspořádává objekty reprezentující prohlížeč a zobrazené dokumenty, do hierarchie:



Objekt window obsahuje všechny ostatní objekty. Nejdůležitějším objektem je document, který zahrnuje všechny objekty a vlastnosti vztahující se k aktuálnímu dokumentu. Například history zachycuje historii dokumentů prohlížeče, frames popisuje rámy na stránce, location uchovává URL stránky zobrazené v okně, navigator informace o prohlížeči. Objekt dokument obsahuje další objekty jako images – obsahuje obrázky na stránce, forms – zpřístupňuje formuláře a jejich komponenty, pomocí frames je umožněn přístup k rámům, atd..

JavaScript však obsahuje i objekty, které s HTML nesouvisí. Jsou to objekty String, Date a Math, umožňují práci s řetezci, s údaji o datu a čase a s matematickými funkcemi.

Objekt String s atributem lenght a metodami: anchor(), big(), blink(), bold(), fontcolor(), fontsize(), italics(), small(), sub(), sup(), link() toUpperCase(), toLowerCase(), substring(), indexOf(), charAt().

Objekt Date s metodami: getDate(), getDay(), getHours(), getMinutes(), getMonth(), getSeconds(), getTime(), getTimezoneOffset(), getYear(), setDate(), setHours(), setMinutes(), setMonth(), getSeconds(), setTime(); setYear()

Př. var den= new date();

Objekt Math obsahuje konstanty - Math.PI, Math.E a metody abs(), acos(), cos(), asin(), sin(), atan(), tan(), exp(), log(), max(), min(), pow(), random(), round(), sqrt(), floor().

Mezi nejjednodušší příkazy patří přiřazovací příkaz ve tvaru: proměnná = výraz;

Výraz může obsahovat proměnné, konstanty a operátory, jako + (sčítání), - (odčítání), * (násobení) a / (dělení), ++ nebo – pro zvýšení nebo zvýšení proměnné o jedničku. Obsahuje i operátory +=, -=, *= a /=, využitelné analogicky jako v jazyce C.

Pro porovnání je v JavaScriptu operátor ==, pro nerovnost != a další relační operátory <, >, <= >=. Logické výrazy můžeme navzájem spojovat pomocí logických spojek && (a zároveň) a || (nebo). Negaci výrazu provedeme zapsáním vykřičníku '!' před výraz.

Podmíněný příkaz má tvar:

if (podmínka) {
příkazy1
}
else {
příkazy2
}
	if (podmínka 1)
	{ příkazy1;}
	else if (podmínka2)
	{ příkazy2;}
	else if (podmínka3)
	{ příkazy3;}
	else if (podmínka4)
	{ příkazy4;}
	….
	else { příkazy;}	

Část else je nepovinná. Pokud se má podmíněně provést jeden příkaz, nemusíme jej uzavírat do složených závorek. JavaScript nabízí několik typů cyklů. Například cyklus for má tvar:

```
for (počáteční inicializace; podmínka; inkrementace) {
   příkazy;
   }
```

Cyklus while má tvar:

```
while (podmínka) {
   příkazy;
   }
```

Definice vlastní funkce je uvedena klíčovým slovem function, následuje identifikátor a seznam parametrů v závorce, oddělených čárkou. Tělo definice je ve složených závorkách, návratová hodnota se definuje pomocí return (hodnota) :

```
function Faktorial(n) {
   if ((n==1) || (n==0))
      return 1
   else
      return (n * Faktorial(n-1));
   }
```

Funkce nemusí vracet žádnou hodnotu, pak ji používáme jako samostatný příkaz a ne jako část výrazu.

Funkce může mít i více parametrů, oddělují se pak čárkou.

Skripty mohou být na stránce umístěny v záhlaví (HEAD) i v těle dokumentu (BODY). Do záhlaví se umísťuje vše, co musí být vykonáno ještě dříve, než začne uživatel pracovat s dokumentem.

Událost	Popis
onLoad	(natažení dokumentu do okna prohlížeče nebo do všech rámů – používá se u BODY a FRAMESET)
onUnLoad	(odstranění dokumentu z okna nebo rámu - BODY a FRAMESET).
onClick	(kliknutí myší na element).
onDblClick	(dvojité kliknutí myší na element)
onMouseDown	(stisknutí tlačítka myši nad elementem)
onMouseUp	(uvolnění tlačítka myši nad elementem)
onMouseOver	(posunutí myši nad element)
onMouseMove	(při pohybu myši nad elementem)
onMouseOut	(po odsunutí myši z elementu)
onFocus	(při aktivování elementu myší nebo pomocí tabulátoru - LABEL, INPUT, SELECT, TEXTAREA a BUTTON)
onBlur	(element přestává být aktivní - LABEL, INPUT, SELECT, TEXTAREA a BUTTON)
onKeyPress	(po stisku a uvolnění tlačítka na klávesnici)
onKeyDown	(po stisku tlačítka na klávesnici)
onKeyUp	(uvolnění tlačítka na klávesnici)
onSubmit	(při odesílání formuláře - u elementu FORM)
onReset	(po vynulování formuláře - u elementu FORM)
onSelect	(po označení textu ve vstupním poli - u elementů INPUT a TEXTAREA)
onChange	(změna hodnoty vstupního pole formuláře - u elementů INPUT, SELECT a TEXTAREA)
onError, onAbort	(při přerušení)

Obsluhou události bývá nejčastěji dříve definované funkce.

U objektů, které odpovídají elementům stránky, máme k dispozici vlastnost i objekt style. Pomocí vlastnosti style můžeme přistupovat k definici stylu elementu stejně jako pomocí atributu STYLE.

Zajímavá je vlastnost innerText. Ta obsahuje text uvnitř elementu. Zápisem do této vlastnosti můžeme text změnit

V objektovém modelu nalezneme pod objektem document i kolekci styleSheets, která obsahuje jednotlivé styly připojené ke stránce. Pomocí její metody addRule(selektor, deklarace, index) můžeme do stylu přidávat nová pravidla.

Abychom mohli do kolekce stylů něco přidávat, musí kolekce existovat. Proto stránka obsahuje prázdný element STYLE, který vytvoří kolekci styleSheets s jedním prvkem.

Bohužel zde nemůžeme rozebrat celou objektovou hierarchii. Zmíníme se však ještě alespoň o několika zajímavostech. Adresa dokumentu, který je právě zobrazován, je přístupná přes objekt location. Pokud tedy chceme přímo skriptem změnit stránku, která se právě zobrazuje, můžeme použít příkaz location="http://někde.jinde.cz".

Pokud máme stránku rozdělenou na rámy, můžeme z jakéhokoliv rámu měnit stránku zobrazenou v ostatních rámech. Stačí novou adresu zapsat do parent.jméno_rámu.location. Tímto způsobem můžeme dosáhnout speciálních efektů, jako je změna obsahu více rámů najednou po zvolení jednoho odkazu. Vždy bychom však měli zvážit, zda je chováni stránky snadno pochopitelné pro její uživatele.

5.3.2 Java Applty a ActiveX

Java Applety jsou malé části kódu v jazyce Java, které se provádí přímo v prohlížeči a mají vymezený určitý prostor dokumentu a do něj mohou podle potřeby kreslit, číst vstupy z klávesnice nebo myši, adt.. Ze zdrojového souboru (.java) se Java Applet překládá do byte-kódu (.class), který je nezávislý na platformě. Jeho interpretaci provádí JVM (Java Virtual Machina). Bezpečnost aplikace zajišťuje security manager, který definuje a spravuje práva appletu. Rychlost a výkon Java appletům dodává JIT (just-in-time) kompilátor, který před spuštěním třídy appletu převede program z byte-code do nativního kódu.

Java applety se do stránky vkládají pomocí elementu applet, případně (od HTML 4.0) i pomocí elementu object. Minimálně definujeme tři atributy:

CODE = jméno appletu (.class souboru), který chceme spustit

WIDTH = šířka plochy vyhrazené pro applet v pixelech

HEIGHT = výška plochy vyhrazené pro applet v pixelech

Použitelné jsou i další atributy, například ALIGN, HSPACE a VSPACE.

Když je soubor s appletem v jiném adresáři než HTML stránka, definujeme adresář, kde je applet uložen, pomocí atributu CODEBASE. Pokud pracujeme na jedné stránce s několika vzájemně komunikujícími applety, identifikujeme je pomocí atributu name. Případné parametry můžeme appletům předávat vnořením elementu PARAM do elementu APPLET. Hodnoty parametrů se definují pomocí atributů NAME a VALUE. Dalšími využitelnými daty jsou soubory ze sítě, přístupné pomocí HTTP a dalších protokolů i databázové servery.

Příklad vložení appletu do stránky:

```
<html><head>
<title>Ukázka Java-appletu na stránce</title>
</head>
<body>
<applet code="Myapplet.class" width="…" height="…">
    <param name="P1" value="…">
    <param name="P2" value="…">
…
  </applet>
</body>
</html>
```

Binární komponenta z MS Windows (ActiveX) se vkládaná do HTML stránky podobně jako Java applet, ale problematická může být bezpečnost takového řešení. Komponenta není nijak

odstíněná od operačního systému počítače a může obsahovat nebezpečný kód, manipulující s citlivými daty. Do HTML stránky se ActiveX vkládá pomocí elementu object a informací v systém registry :

```
<object
classid="clsid: …. číslo …"
codebase="http://..."
width="…"
height="…">
<param name="p1" value="…">
<param name="p2" value="…">
…
</object>
```

5.4 Serverové www technologie

Prakticky nejdůležitější je využívání technologií používaných na webových serverech, případně kombinace s klientskými skripty. Využívá se víceúrovňové architektury klient – server. Základní princip spočívá v komunikaci mezi uživatelem na klientské straně a serverem – prostřednictvím prohlížeče se předávají požadavky nebo události na webový server, který dynamicky vytvoří HTML stránky s požadovaným obsahem. V tomto textu se soustředíme na využívání databázových technologií a datových zdrojů. Přehled nejpoužívanějších technologií pro dynamické vytváření obsahu HTML stránky zahrnuje následující nástroje:

- Skripty CGI(Common Gateway Interface) a FastCGI – První případ použití aplikační dynamiky v HTML stránkách. Původně tato technologie umožňuje dynamicky vytvořit stránku pomocí C nebo Perlu. Rozhraní CGI definuje způsob komunikace. Pokud program dodržuje konvence rozhraní CGI, hovoříme o CGI skriptu. Předávání parametrů (protokol HTTP) umožňují dvě metody GET a POST. Standardně se používá metoda GET, pomocí které se k URL stránky připojí přípona obsahující data ve formátu název1=hodnota1&název2=hodnota2&... . Hodnoty jsou transformovány do textové podoby pomocí speciálních znaků ve tvaru %xx, kde xx je kód znaku v šestnáctkové soustavě i oddělovačů – mezeře odpovídá znak „+“, oddělovačem URL a datové přípony je znak „?“. Při použití metody GET zpracovává CGI skript data z proměnné prostředí QUERY_STRING, pro metodu POST platí, že CGI skript je načítá ze standardního vstupu. Využitelné jsou i další proměnné prostředí:

REQUEST_METHOD - určuje způsob předávání informací (GET nebo POST)
QUERY_STRING - data přenášená metodou GET
PATH_INFO - cesta zpracovávaná skriptem
PATH_TRANSLATED - cesta ke stejnému souboru jako PATH_INFO, ale mapovanému podle konfigurace serveru
CONTENT_TYPE MIME - typ dat zasílaných metodou POST
CONTENT_LENGTH - délka dat zasílaných metodou POST
SCRIPT_NAME - URL prováděného skriptu
SERVER_NAME - jméno serveru
SERVER_PORT - číslo portu

SERVER_SOFTWARE - verze programu vebového serveru
SERVER_PROTOCOL - verze protokolu požadaveku (HTTP/1.0 nebo HTTP/1.1)
GATEWAY_INTERFACE - verze použitého rozhraní pro spuštění skriptu (CGI/1.1)
REMOTE_HOST - doménová adresa počítače, ze kterého přišel požadavek
REMOTE_ADDR - IP-adresa počítače, ze kterého přišel požadavek
AUTH_TYPE - použutý způsob autentifikace uživatele
REMOTE_USER – jméno autentifikovaného uživatele

Výstupní informace CGI skriptu se předávají přes standardní výstup jako posloupnost HTTP hlavičky + prázdný řádek + informace, typicky ve formátu HTML. Web-server výstup skriptu zpracuje a ve formě HTML stránky dodá klientovi. Minimálně je nutné vygenerovat hlavičku Content-Type, určující druh odesílaných dat (text/html).

Nevýhodou technologie CGI je relativní pomalost odezvy a velké nároky na zdroje serveru, protože pro obsluhu každého požadavku musí být spouštěn nový proces. Vylepšená varianta rozhraní FastCGI proto snižuje zátěž serveru - skript se do paměti načítá jednou a postupně obsluhuje další požadavky.

- Přidané direktivy SSI (Server side Includes) – příkazy vnořené do HTML textu, prováděné na web serveru se syntaxí: <!--#příkaz parametry-->. Používané příkazy jsou:

  #config - nastavení formátu výstupu ostatních příkazů
  #echo - vypsání obsahu proměnné (DATE_GMT, DATE_LOCAL, DOCUMENT_NAME, DOCUMENT_URI, LAST_MODIFIED, QUERY_STRING_UNESCAPED)
  #exec - spuštění externího programu
  #flastmod – vrací čas poslední modifikace souboru
  #fsize - zjištění velikosti souboru
  #include - načtení externího souboru

- Rozhraní SAPI, ISAPI (Internet Server Application Programing Interface) moduly a filtry – další technologií, která vznikla jako alternativa k CGI. Přeložené a spustitelné aplikace napsané pro SAPI obvykle běží na web-serveru mimo hlavní proces v paměťovém prostoru odděleném od web-serverových aplikací – případný výpadek aplikace server neovlivní. Aplikace, které jsou implementovány ve formě DLL knihoven naopak běží ve stejném paměťovém prostoru, což zrychluje odezvu, ale při chybě může dojít ke zhroucení web-serveru.
- ASP(Active Server Pages) - ISAPI technologie je také základem technologie ASP, která běžící na MS Internet Information Serveru.
- PHP(Personál Home Page)
- Technologie Java - jazyk Java je základem několika používaných technologií např. JavaBeans - J2EE (Java 2 Platform, Enterprise Edition) s moduly Java servlets, JavaServer Pages (JSP). Java servlet je na platformě nezávislá, komponentní technologie, běžící na webovém serveru. Servlet se dá považovat za obdobu appletu na straně serveru. Servlety zpracovávají požadavek klienta a mohou odpovědět ve formě HTML, XML, obrázku, … . JSP je technologie srovnatelná s ASP, ale programovacím jazykem je Java.
- Technologie ASP.NET

5.4.1 PHP

Patří mezi oblíbené technologie pro svou dostupnost (open source), nezávislost na platformě

a na webovém serveru.

Důležitá data vývoje PHP

1994 - PHP/FI - Personal Home Page Tools/Forms Interpreter, základem se stal jazyk perl, implementován základní přístup k databázím

1997 - PHP/FI 2.0 - následná implementace v C

1998 - PHP 3.0 - nová verze s podporou objektů, databází

1999 - PHP 4.0 - rozšíření jazyka (HTTP session, buffering výstupu)

2004 - PHP 5.0 – další rozšíření jazyka, nový objektový model, podpora XML (DOM), SOAP

PHP podporuje širokou řadu souvisejících technologií, formátů a standardů , je otevřeným projektem s rozsáhlou podporou komunity. Je založen na vkládání skriptového kódu do HTML dokumentu. Je to interpretovaný, ne kompilovaný jazyk. Syntaxe PHP jazyka je založena na kombinaci rysů C, Javy a Perlu. Vybrané charakteristiky:

Ve skriptu je kód php uzavřen dvojicí značek , např. <?php … PHP kód … ?>, což je usporný ekvivalent klasického <SCRIPT LANGUAGE="php"> …PHP kód... </SCRIPT>

Jednotlivé instrukce se v PHP oddělují středníkem. Komentáře mohou být v PHP jednořádkové nebo víceřádkové. Používají se znaky "//" (dvě dopředná lomítka), "#" (mřížka) pro jednořádková a dvojice "/*" a "*/" pro víceřádková. Deklarace proměnné se provádí při prvním použití s možností použít několik typů proměnnnných (boolean, integer, float, string, array, object) s automatickou detekcí typu. Každá proměnná musí mít unikátní název, začínající znakem dolaru($). Příkaz echo zobrazí v dokumentu jeden nebo více řetězců. Spojování retězců se prování pomocí operátoru . (tečka). Další podrobnosti syntakse operátorů, příkazů, použitelných funkcí lze nalést v literatuře, v tutoriálech na internetu a podobně.

PHP určitě za svou popularitu vděčí i podpoře práce s databázemi, zvláště pak s MySQL. Pro ilustraci uvedeme některé používané funkce a postupy

- Připojení k databázovému serveru a nastavení aktuální aktivní databáze:

mysql_connect - otevře spojení s databázovým serverem MySQL, vrací identifikátor spojení

mysql_pconnect - otevře trvalé spojení s MySQL serverem (neukončuje se při ukončení skriptu ani funkcí mysql_close(), vrací identifikátor spojení.

mysql_select_db - nastaví aktuální aktivní databázi pro dané spojení s databázovým serverem

mysql_close - uzavře spojení s databázovým serverem MySQL

- Vytvoření a zrušení databáze:

mysql_create_db - vytvoří databázi spravovanou serverem MySQL

mysql_drop_db - zruší databázi spravovanou serverem MySQL

- Provádění příkazů SQL a práce s kurzorem (výsledek (result)):

mysql_query - pošle zadaný příkaz současné aktivní databázi serveru MySQL, pro INSERT, UPDATE a DELETE vrací příznak úspěšnosti, pro SELECT číslo kurzoru

mysql_db_query - vrátí číslo kurzoru pro zadaný dotaz pro danou databázi

mysql_affected_rows - vrátí počet řádků ovlivněných posledním příkazem INSERT, UPDATE nebo DELETE

mysql_fetch_array - vrátí řádek kurzoru jako asociativní pole (výběr podle jména sloupce)

mysql_fetch_row - vrátí řádek kurzoru jako pole s prvky zpřístupňovanými pořadovým číslem

mysql_fetch_object - vrátí řádek kurzoru jako objekt - přístup přes jména

mysql_result - vrátí hodnotu daného sloupce daného řádku daného kurzoru, neměla by být používána s jinými funkcemi zpřístupňujícími řádky kurzoru

mysql_data_seek - posune ukazatel kurzoru na řádek s daným pořadovým číslem

mysql_free_result - uvolní paměťový prostor kurzoru

mysql_num_rows- vrátí počet řádků kurzoru

mysql_num_fields - vrací počet sloupců kurzoru

mysql_fetch_lengths - vrátí délky polí posledně vybraného řádku kurzoru funkcí mysql_fetch_row

mysql_insert_id - vrátí identifikátor generovaný posledním příkazem INSERT pro sloupec typu AUTO_INCREMENTED

- Ošetření chyb:

mysql_errno - vrátí číslo chyby poslední operace MySQL

mysql_error- vrátí text chybového hlášení poslední operace MySQL

- Přístup k metadatům:

mysql_list_dbs - vrátí kurzor obsahující seznam dostupných databází, k procházení slouží funkce mysql_tablename

mysql_list_tables- vrátí kurzor obsahující seznam tabulek dané databáze, k procházení slouží funkce mysql_tablename

mysql_tablename - vrátí jméno tabulky s daným pořadovým číslem prostřednictvím kurzoru, vráceného funkcí mysql_list_tables

mysql_fetch_field - vrátí objekt, který nese informaci o daném sloupci daného kurzoru

mysql_field_seek - nastaví ukazatel kurzoru na daný sloupec

mysql_field_name - vrátí jméno daného sloupce kurzoru

mysql_field_table - vrátí jméno tabulky, které patří zadaný sloupec kurzoru

mysql_field_type - vrátí jméno tabulky, které patří zadaný sloupec výsledku

mysql_field_flags - vrátí příznaky ("not_null", "primary_key", …), spojené se zadaným sloupcem výsledku

mysql_field_len - vrátí délku specifikovaného sloupce daného kurzoru

mysql_list_fields - zpřístupní metadata pro danou tabulku, vrací číslo kurzoru, které lze použít ve funkcích pro práci s metadaty (mysql_field_flags(), mysql_field_len(), mysql_field_name() a mysql_field_type())

Problém přístupu do IS (HTTP je bezestavový = nutnost opakované autorizace – variantně historicky pomocí cookies nebo formulářových polí typu hidden) představuje nebezpečí odhalení loginu a hesla. Proto je použito nové schéma s procesem, kdy výchozí přihlášení (login a heslo) založí sezení, jehož sessionID se uloží do cookies a do souboru na serveru. Tak je možno provést mapování sessionID na uložené proměnné na serveru. Po odhlášení ze systému se sezení zruší a soubory s daty se smažou. Používají se funkce a proměnné jako

session_start(); session_register(); $_SESSION[ ],

SetCookie (,); $_COOKIE[ ];

Ilustrační příklad prezentující data (SQL dotaz), převzatý z jednoduchého tutoriálu :

Soubor `dbinfo.inc.php`

```
<?
$username="root";
$password="";
$database="contacts";
?>
```

Soubor index.php

```
<?
include("dbinfo.inc.php");
mysql_connect(localhost,$username,$password);
@mysql_select_db($database) or die( "Unable to select database");
$query="SELECT * FROM contacts";
$result=mysql_query($query);

$num=mysql_numrows($result);

mysql_close();

echo "<b><center>Database Output</center></b><br><br>";

?>
<table border="0" cellspacing="2" cellpadding="2">
<tr>
<th><font face="Arial, Helvetica, sans-serif">Name</font></th>
<th><font face="Arial, Helvetica, sans-serif">Phone</font></th>
<th><font face="Arial, Helvetica, sans-serif">Mobile</font></th>
<th><font face="Arial, Helvetica, sans-serif">Fax</font></th>
<th><font face="Arial, Helvetica, sans-serif">E-mail</font></th>
<th><font face="Arial, Helvetica, sans-serif">Website</font></th>
</tr>

<?
$i=0;
while ($i < $num) {
$first=mysql_result($result,$i,"first");
$last=mysql_result($result,$i,"last");
$phone=mysql_result($result,$i,"phone");
$mobile=mysql_result($result,$i,"mobile");
$fax=mysql_result($result,$i,"fax");
$email=mysql_result($result,$i,"email");
$web=mysql_result($result,$i,"web");
?>

<tr>
<td><font face="Arial, Helvetica, sans-serif"><? echo "$first $last";
?></font></td>
<td><font face="Arial, Helvetica, sans-serif"><? echo "$phone"; ?></font></td>
<td><font face="Arial, Helvetica, sans-serif"><? echo "$mobile";
?></font></td>
<td><font face="Arial, Helvetica, sans-serif"><? echo "$fax"; ?></font></td>
<td><font face="Arial, Helvetica, sans-serif"><a href="mailto:<? echo
"$email"; ?>">E-mail</a></font></td>
<td><font face="Arial, Helvetica, sans-serif"><a href="<? echo "$web";
?>">Website</a></font></td>
</tr>
<?
++$i;
}
echo "</table>";


?>
```

5.4.2 ASP

Technologie ASP(Active Server Pages) je koncept Mikrosoftu pro generování dynamických interaktivních HTML stránek pomocí skriptů na straně webového serveru. Základem je textový soubor s příponou .asp, který kromě základní technologie HTML elementů kombinuje text, příkazy skriptovacího jazyka, využívá COM komponenty. Spolu s PHP je to jedno z nejpoužívanějších aplikačních prostředí pro webové aplikace se skripty na straně serveru (nahrazováno novější MS technologií ASP.NET). Je standardní součástí webových serverů MS, podpora jiných serverů a platforem je minimální.

Princip zavedení dynamických změn obsahu stránky je jednoduchý. Přímo do HTML kódu, nejčastěji pomocí zápisu `<% ...kód skriptu na straně serveru … %>` se zapisují jednoduché příkazy v libovolném jazyku, podporujícím Aktive Scripting. Další možností je použití elementu `<SCRIPT LANGUAGE="VBScript" runat=server> ...kód skriptu na straně serveru …</SCRIPT >`. Standardně je to na straně serveru jazyk VBScript (na straně klienta je to standardně JScript). Další možnosti skriptovacích jazyků nabízí ostatní firmy, dodávající například Perl, REXX, Python.

Element `<% … %>` ohraničuje část kódu skriptu, zpracovávaného na straně serveru, v kontextu kódu klasické HTML stránky

Př. :

```
<HTML>
  <BODY>
  Poslední přístup na stránku na serveru <%= Now() %>.
  </BODY>
</HTML>
```

Syntaxe VBScriptu poměrně důsledně kopíruje syntaxi jazyka MS Visual Basic, její detaily jdou za rámec textu. Zápis výstupní direktivy `<%= …` je úspornou variantou pro zápis textu dokumentu z prostředí skriptu a odpovídá `<% Response.Write … .`

ASP prováděcí direktivy jsou umístěny na prvním řádku dokumentu a mají syntaxi

<%@ klíčové slovo %>. Dodávají informace pro zpracování ASP stránky, např. definuje jazyk skriptu:

```
<%@ LANGUAGE=VBScript %>
```

Kromě LANGUAGE jsou dalšími možnostmi:

ENABLESESSIONSTATE (využití stavu sezení – session),

CODEPAGE (kódování znaků),

LCID(lokální identifikátor souboru),

TRANSACTION (režim transakčního zpracování).

Dalšími direktivami jsou **SSI direktivy**, které nabízí možnosti:

#config(konfigurace parametrů, např. specifikace formátu pro datum a čas),

#echo(vkládá hodnotu proměnné prostředí http),

#exec (provede program nebo příkaz),

#flastmod (vloží datum a čas poslední modifikace),

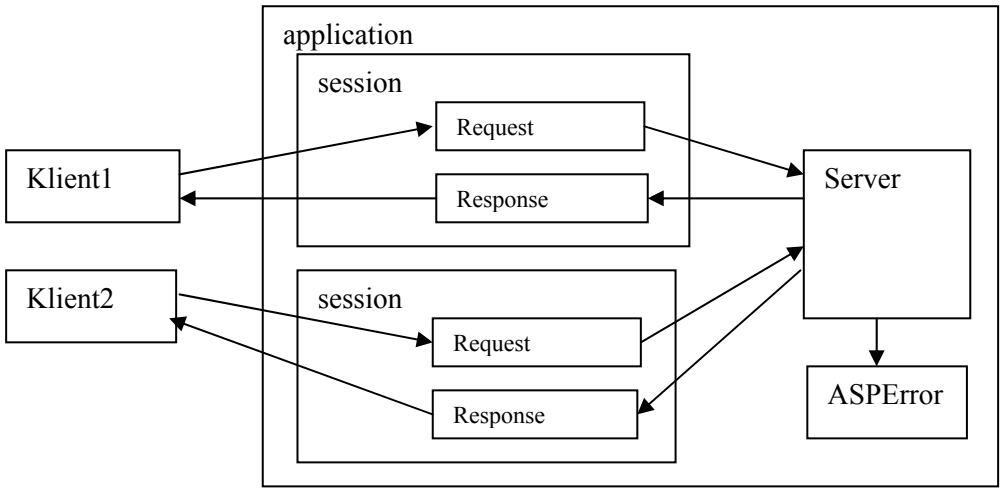
#fsize(vloží velikost souboru).

Příklad syntaxe direktivy pro vkládání souborů:

```
<!-- #include virtual | file ="… soubor" -->,
```

kde klíčové slovo virtual se používá pro virtuální adresář, file pro relativní adresu souboru.

Ve všech skriptovacích jazycích jsou dostupné základní vestavěné objekty, které svou funkcionalitou umožňují vytvoření plnohodnotné webové aplikace.



Obrázek 24 Základní vestavěné objekty ASP

Koncept ASP řeší komunikaci a přenos dat mezi klientským prohlížečem a webovým serverem v daném sezení (objekt session) pomocí objektů Request a Response na úrovni protokolu HTPP - je skryta a implementována pomocí metod a dat v kolekcích těchto objektů. Zpracování skriptu na webovém serveru je vyvoláno požadavkem na stránku (soubor) .asp, které je zpracováváno postupně od počátku do konce prováděním příkazů skriptu a vytvořením webové stránky, která odeslána zpět do prohlížeče.

Objekty Request a Response

Jednotlivé HTTP požadavky a informace ze strany klienta se do aplikace, stránky dostávají prostřednictvím objektu Request. Pokud prakticky nejčastěji na straně klienta definujeme parametry ve formuláři, podle použité metody jsou jejich hodnoty ve skriptu, který je zpracovává, dostupné v kolekcích Form (POST), nebo QueryString(GET). Kolekce ServerVariables obsahuje data z HTTP hlavičky klientského požadavku a data doplněná serverem – např. jméno serveru, port, metodu, adresy, uživatele, verze a jazyk prohlížeče, … .

Mezi důležité problémy i ASP technologie je uchování a sdílení stavu a identifikace uživatele aplikace. Web je z principu bezestavové anonymní prostředí (HTTP protokol je bezestavový), to znamená, že obecně nové načtení stránky nemá kontext předešlého načtení. Mezi klasická, ale neideální řešení tohoto problému patří použití Cookies. Zjednodušeně jsou Cookies krátké textové informace, které jsou prohlížečem ukládány na straně klienta a následně znovu zasílány na server, pokud je znovu zpracovávána stejná stránka. V ASP je možno k tomuto účelu využít kolekce Request.Cookies a Response.Cookies. Stručný přehled vlastností a metod objektů je v následujících tabulkách.

Objekt Request		popis
ClientCertificate	kolekce	Hodnoty klientského certifikátu
Cookies	kolekce	Hodnoty cookies
Form	kolekce	Hodnoty prvků formuláře (metada POST)
QueryString	kolekce	Hodnoty prvků dotazu – přípona URL po oddělovači „?“ (generováno automaticky např. metadou formuláře GET)

ServerVariables	kolekce	Hodnoty proměnných prostředí
TotalBytes	vlastnost	Celkový počet Bytů zaslaných klientem v požadavku
BinaryRead	metoda	Klientká data, zaslaná v části požadavku POST

Dynamické reakce v obsahu HTML stránky se generují pomocí objektu Response. Mezi důležité možnosti patří správa paměti cashe pomocí .*CacheControl*

Objekt Response		popis
Cookies	kolekce	Nastavují hodnotu, nebo vytváří položky Cookies
Buffer	vlastnost	Indikuje použití vyrovnávací paměti
CacheControl	vlastnost	Nastavuje hlavičku HTTP/1.1 Cache-Control pro vhodný mechanizmus paměti cache
Charset	vlastnost	Definuje kódovou stránku
CodePage	vlastnost	Definuje znakovou sadu
ContentType	vlastnost	Definuje http typ obsahu
Expires	vlastnost	Definuje dobu, po které se uvolní stránka z cache paměti
ExpiresAbsolute	vlastnost	Definuje datum a čas, kdy se uvolní stránka z cache paměti
IsClientConnected	vlastnost	Indikuje aktuální stav připojení klienta
PICS	vlastnost	Nastavuje formát datumu, času a měny
Status	vlastnost	Definuje status http specifikace
AddHeader	metoda	Definuje jméno HTML hlavičky
AppendToLog	metoda	Přidá řetězec do souboru log web- serveru
BinaryWrite	metoda	zapisuje bez konverze Bytů
Clear	metoda	Vymaže vyrovnávací paměť HTML
End	metoda	Ukončení zpracování asp stránky a odeslání aktuálního obsahu
Flush	metoda	Okamžité odeslání obsahu vyrovnávací paměti
Redirect	metoda	Pokus o přesměrování na URL adresu
Write	metoda	Zapisuje proměnné nebo text

Objekt Server

Přístup aplikace, stránky k datovým a databázovým zdrojům zajišťují standardně MS COM technologie, typicky prostřednictvím OLE DB. K důležitým vlastnostem a funkčnosti objektu *Server* patří zprostředkování přístupu ke COM komponentám na webovém serveru a s jejich pomocí například k datům na databázovém serveru. Komponenty mohou být implicitní, systémové – jako ADO, OleDB, …, nebo s jejich pomocí vytvořené specializované uživatelské komponenty, implementované ve formě .dll knihoven, které realizují vícevrstvou architekturu aplikace. Instanci komponenty ve skriptu vytvoří metoda `.CreateObject(…)` . Další vlastnosti a metody objektu jsou v následující tabulce:

Objekt Server		popis
ScriptTimeout	vlastnost	Definuje maximální časový interval zpracování skriptu
CreateObject	metoda	Vytvoření instance COM objektu
Execute	metoda	Zpuštění .asp souboru
GetLastError	metoda	Vrací objekt ASPError s popisem chyby
HTMLEncode	metoda	Aplikace HTML kódování na řetězec
MapPath	metoda	Mapování virtuální nebo relativní cesty na fyzickou
Transfer	metoda	Zaslání aktuálních stavových informací do jiného .asp souboru
URLEncode	metoda	Aplikace URL kódování na řetězec

Dalšími objekty v kontextu MS Visual Basic Object Model jsou:

- ASPError Object pro práci s chybovými stavy
- ObjectContext pro práci s vestavěnými objekty a jejich zapojením do transakčního zpracování

Objekty Session a Application

Problém uchování stavu (sdílení hodnot proměnných) se řeší v rámci buď celé aplikace – mezi všemi klienty webové aplikace, a nebo viditelnost v rámci jednoho sezení klienta. Také se nabízí možnost obsluhy vybraných událostí, které se vztahují k celé aplikaci (objekt Application), nebo k jedinému uživateli (objekt Session). Objekt Application má pouze jednu instanci v celé aplikaci, instance objektu Session se vyskytuje na serveru tolikrát, kolik je aktuálně přihlášených klientů - patří k danému přihlášenému klientovi. Protože je kód aplikace napsaný z pohledu jednoho klienta, klient nemá přístupné Session objekty jiných klientů. Deklarace a implementace událostí (a citlivé informace aplikace jako například login a heslo do databázových zdrojů) je umístěna v zabezpečeném souboru *Global.asa*, který je na serveru implicitně zpracováván. Krátký přehled objektů je v následujících tabulkách.

Objekt Session

Session objekt		popis
Contents	kolekce	Obsahuje položku, vloženou ze skriptu, platí v rámci sezení
StaticObjects	kolekce	Obsahuje objekty, vytvořené <OBJECT> tagem v kontextu session
CodePage	vlastnost	Nastavuje kódovou stránku pro data v objektech v rámci celého sezení (session)
LCID	vlastnost	Nastavuje formát datumu, času a měny
SessionID	vlastnost	Identifikátor sezení
Timeout	vlastnost	Definuje časový interval v minutách, po který je uchováván stav
Abandon	metoda	Zruší session objekt i všechny objekty v rámci session
Contents.Remove	metoda	Odstraní položku z kolekce Contents
Contents.RemoveAll	metoda	Odstraní všechny položky z kolekce Contents
Session _OnEnd	událost	Událost při zrušení sezení nebo uplynutí Timeout
Session _OnStart	událost	Událost při vzniku sezení

Objekt Application

Application objekt		popis
Contents	kolekce	Obsahuje položku, vloženou ze skriptu, platí v rámci celé aplikace
StaticObjects	kolekce	Obsahuje objekty, vytvořené <OBJECT> tagem v kontextu aplikace
Contents.Remove	metoda	Odstraní položku z kolekce Contents
Contents.RemoveAll	metoda	Odstraní všechny položky z kolekce Contents
Lock	metoda	Nasadí zámek, který znemožní ostatním klientům modifikovat vlastnosti objektu application
Unlock	metoda	uvolní zámek, který znemožňoval ostatním klientům modifikovat vlastnosti objektu application, dává možnost modifikace objektu
Application_OnEnd	událost	Událost při pozastavení aplikace
Application_OnStart	událost	Událost před výskytem prvního session v aplikaci

Jednoduchý příklad, ukazující prototypově použití objektů, jejich metod a kolekcí při přenosu informací z klienta na server a zpět. HTML stránka login.htm slouží pro určení hesla pro přístup uživatele webstudent do databáze na SQL serveru. Při použití metody post jsou data zpracována ve skriptu stránky SQL.asp v kolekci objektu Request.Form. Pro názornost jsou na

stránku dokumentu odpovědi vypsána data z této kilekce. Objekt Server vytvoří instanci komponenty ADOdb.conection. Pokud dojde k chybě – předpokladem je špatné heslo, uživateli je vrácena výchozí stránka. Jinak se do tabulky zapíší data z předdefinovaného SQL dotazu.

Soubor login.htm:

```
<HTML>
<HEAD>
<META NAME="GENERATOR" Content="Microsoft Visual Studio 6.0">
<TITLE></TITLE>
</HEAD>
<BODY>
<FORM id=FORM1 name=FORM1 action="sql72.asp" method=post>

<P>user:<INPUT id=text1 name=text1 ></P>
<P>password:<INPUT       type="password"       id=password1       name=password1
value=webstudent> </P>
<P><INPUT type="submit" value="Submit" id=submit1 name=submit1>
<INPUT id=reset1 type=reset value=Reset name=reset1> </P>
</FORM>
</BODY>
</HTML>
```

Soubor SQL72.asp:

```
<%@ Language=VBScript %>
<HTML>
<HEAD>
<META NAME="GENERATOR" Content="Microsoft FrontPage 4.0">
</HEAD>
<BODY>
<P>
<OBJECT id=OBJECT2 PROGID="ADODB.Connection" RUNAT="server"></OBJECT></P>
<P>
<OBJECT id=OBJECT1 RUNAT="server" PROGID="ADODB.Recordset"></OBJECT></P><%
on error resume next
for i=1 to  Request.Form.Count' - 1
s = Request.Form.Key(i) & "=" & Request.Form.Item(i)& "<br>"
Response.Write s
next

Set oConnection  = Server.CreateObject("ADOdb.connection")
'oConnection.Open("DSN=sql2000;UID=webstudent;PWD=webstudent;DATABASE=northwin
d")
oConnection.Open("DRIVER={sql  server};server=csnt\kmi;Database=Northwind;PWD="
& Request.Form.Item(2)& ";UID=webstudent;")
if Err.Number <> 0 then
'Response.Write "chyba <br><br>"
session("login_Error") = true
Response.Redirect ("login.htm")
end if
on error goto 0

Set     oRS     =     oConnection.Execute("Select     [Employees].[FirstName],
[Employees].[LastName] from Employees")
'Response.Write oRS(1)
'Response.Write oRS(2)

Response.Write "<TABLE BORDER=""2"" ALIGN=""center"">"
Response.Write "<CAPTION> Document </CAPTION><TR>"

             for each ff in ors.fields

      Response.Write "<TH>" & ff.name & "</TH>"
          next
```

```
   ors.Movefirst

   While ors.EOF = False
   Response.Write  "<TR>"
             for each ff in ors.fields
            Response.Write  "<TD>" & ff.value & "</TD>"
      next

      ors.MoveNext
     Response.Write  "</TR><TR>"
   Wend
Response.Write "</TR></TABLE>"
oconnection.Close

%>

</BODY>
</HTML>
```

5.4.3 ASP.NET

Asp.NET je součást .NET frameworku, která vytváří a provozuje webové aplikace a služby, tvoří programový rámec pracující na webovém serveru, generující a spravující dynamické webové stránky. Vývojovým prostředím a nejlepším nástrojem pro tvorbu je Visual Studio.NET, ale stačí i jednoduchý textový editor, např. Notepad. Syntaxí navazuje na ASP, ale je více, než další verze. V infrastruktuře ASP.NET se řeší i některé problematické infrastrukturní úlohy, které v ASP aplikacích a podobných technologiích nejsou systémově dostatečně vyřešeny. Například zachování stavu na webových farmách po výpadku aplikace bez použití cookie, ukládání konfigurace aplikace, autentizace uživatele při přístupu k databázi, potíže při aktualizaci knihoven DLL za chodu aplikace. ASP.NET mohou použít libovolný jazyk, kompatibilní s NET Framework, který se na rozdíl od technologie ASP překládá. NET framework je plně k dispozici a nabízí objektové rysy (dědičnost, …), typovou kontrolu atd.. Používá se nový programový model, infrastruktura, zajišťující větší škálovatelnost, bezpečnost a stabilitu.

Řešení spočívá v inovacích architektury pro scale-out. Aplikace není omezena na jediný server. Stav je uložen v Session i pro farmy webů, více serverů může sdílet stav uložený na jednom serveru. Je možné volit úložiště stavu aplikací – buď v procesu IIS aplikace (RAM, stejný proces), nebo ASP.NET state server (RAM, jiný proces), databázový SQL Server.

Mezi základní cíle a charakteristiky tedy patří striktně objektové programování, použití osvědčeného desktopového paradigmatu klasických aplikací (Visual Basic pro Web), úspora rozsahu kódu a proto levnější vývoj, ostré oddělení jednotlivých vrstev aplikace (prezentace, střední vrstva), provádění nativního kódu jazyků jako Visual Basic, C#, JScript® s lepším výkonem proti interpretovaným skriptům.

ASPX stránky: Princip

Jeden soubor .aspx může obsahovat kód i HTML značky nebo vzhled a kód lze oddělit - použijí se dva soubory, např: Form1.aspx – může obsahovat HTML značky, Form1.aspx.vb obsahuje kód. Zvolený způsob je určen syntaxí:

<%@Page CodeBehind=.. Inherits=.. - pro VS.NET se dvěma soubory

<%@Page Src=... - pro samotný framework

<%@Page > … -pro jeden soubor

Direktiva <%@Page ... může obsahovat i další atributy, např. AspCompat (umožňuje spustit stránku jako STA-single-threaded apartment- vlákno) , AutoEventWireup (idikace zpracování událostí), Buffer (režim spolupráce s vyrovnávací pamětí), ClassName (jméno třídy stránky), CodePage (znaková sada), ContentType, Debug, EnableSessionState, EnableViewState, EnableViewStateMac (indikace spuštění MAC - machine authentication check), ErrorPage (přesměrování při chybě), Language, LCID (lokální identifikátor), ResponseEncoding, Transaction (režim zpracování transakcí), ...

Na místo klasických HTML formulářových prvků se předpokládá použití serverových ovládacích prvků – prakticky stačí v klasickém prvku definovat atribut *runat="server"* .Objekt Page představuje stránku, která je vyžádaná klientem. Třída Page vytváří hierarchii ovládacích prvků, kde Page je v kořeni tohoto stromu a na rozdíl od dalších objektů má navíc dvě události – *Load* a *Unload*. Mezi nejdůležitější vlastnosti patří *IsPostBack*, která se podílí na způsobu zpracování událostí a má při prvním načtení stránky hodnotu *False*, ale je nastavena na *True*, když je stránka odeslána na server.
Statický text je reprezentován jako třída LiteralControl ve stromu ovládacích prvků. Na konci zpracování je vyvolána metoda Page.Render() – proběhne propagace hierarchií ovládacích prvků a generují se HTML stránky, odeslané klientovi (podpora více typů klientů - DHTML, HTML 3.2, WML, atd.)
Serverové ovládací prvky - zapouzdřují vizuální elementy, interakce s uživatelem. Generují události (změna hodnoty, stisknutí tlačítka, kliknutí na ploše, …) Provádění není lineární. V každé fázi zpracování vyvolána událost – např. Init, LoadViewState, LoadPostData, Load, Render, Dispose Unload. Kód reagující na události je vyvolán při zpracování nebo uživatelem. Může být součást ASPX, v separátním souboru nebo v DLL knihovně.

Mezi další objekty patří:

- **Response**, odvozený od HttpResponse – umožňuje přístup k výstupnímu proudu aktuální stránky – podpora vkládání textu ze skriptu, kookies a podobně
- **Request**, odvozený od Http Request - umožňuje přístup k požadavku aktuální stránky – podpora vstupních hodnot z kolekcí, kookies a podobně
- **Context**, odvozený od Http Kontext – uchovává celý kontext při sdílení informací mezi stránkami
- **Server**, odvozený od HttpServerUtility – obsahuje pomocné funkce, informace o chybách – např. metoda .Transfer(…) pro přesměrování stránky.
- **Application**, odvozený od HttpApplicationState – provádí metody a události společné všem uživatelům, např správa cache – paměti aplikace.
- **Session**, odvozený od HttpSessionState - provádí metody a události společné jednomu uživateli, např správa cache – paměti uživatele.
- **Trace**, odvozený od TraceContext – podpora diagnostiky sránky

Další vybrané charakteristiky ASP.NET:

- Integrovaná Cache:

  pro celou stránku, varianty – podle parametrů, jazyka, prohlížeče

  pro část stránky - umožňuje ušetřit čas pro výpočet částí aspx

  rozšiřitelný objekt Cache (Cache API) - je možné v ní uložit libovolný objekt, zvolit flexibilní možnosti pro expiraci, možnost invalidace cache na základě změny souboru

- ASP.NET autentizace: Basic, Digest, NTLM využívají IIS (Jméno/heslo ověřeno proti AD/SAM). Modul pro passport autentizaci (Přístupný Passport profil). Vlastní

formulářová autentizace - jednoduché použití, přihlašovací tiket ve formě šifrovaného cookie, vlastní přihlašovací stránka, ověření proti databázi, proti web.config, …

Global.asax – obsahuje direktivy aplikace, procedury pro události na úrovni objektů application a session.

Typické vlastnosti Asp.NET:

rozšiřitelnost, spolehlivost, dostupnost, škálovatelnost

- Konfigurace: Uložena jako XML soubor spolu se stránkami. Konfigurační soubory web.config mohou být umístěny v několika adresářích na ASP.NET Web serveru a obsahují všechny konfigurační nastavení ASP.NET(Autentizace, kompilace, trasování, ladění, chybové stránky, moduly, ovladače, …, dále libovolné nastavení v sekci <appSettings>, spojení do databází, cesty k adresářům.
- Nasazení ASP.NET aplikací: "XCOPY" nasazení, Odpadá nutnost registrace komponent, restartu služeb/serveru. Podporováno pro všechny součásti Web stránky, web služby, zkompilované komponenty (DLL), konfigurační soubory. Aktualizace aplikací za provozu nakopírujte nové DLL na místo původního. Aplikace použijí nové DLL počínaje příštím klientským požadavkem. Tento postup umožňuje snadnou spravu a udržbu aplikace (žádné restarty).
- Zpracování a prezentace databázových dat je postaveno na stejných základech, jako u klientských aplikacích, tedy na ADO.NET. Prototypová řešení pro prezentaci informací podporuje silná komponenta DataGrid. Další užitečné komponenty se mohou podílet na validaci dat, zadávaných do databáze s možností provádění kódu na straně klienta, v případě problémů na straně serveru. Existují různé typy kontroly(typů validátorů), zajišťující, že hodnota je definovaná, vyhovuje explicitnímu porovnání s definovanou, vypočtenou, nebo z databáze načtenou hodnotou, zajišťující, že hodnota leží v intervalu určených přípustných číselných, datumových i textových hodnot. Silnější prostředek nabízí možnost definovat regulární výraz, kterému musí zadávaná hodnota vyhovovat, případně je možné kontrolní mechanizmus programově uživatelsky naprogramovat.

- Zpracování události (**PostBack**): Události jsou generovány klientem, ale zpracovávány serverem. Události se dělí podle typu na ty co vyvolají okamžitě Postback (Click) a na ty, které se zaznamenají, ale nevyvolají - Non-Postback(Change). Zpracování výskytu všech událostí se zapíná nastavením vlastnosti AutoPostBack. S událostmi je třeba šetřit, obzvláště nejsme-li na LAN síti, zvyšují zátěž serveru a zpomalují práci klienta. Zda jde o první zobrazení stránky nebo o PostBack zjistíme pomocí vlastnosti Page.IsPostBack. Příklad minimálního kódu pro zobrazení dat z datasetu v DataGridu:

```
If Not IsPostback Then ' Evals true first time browser hits the page
Me.oleDbConnection1.Open()
Me.oleDbDataAdapter.Fill(dsProducts)
Me.oleDbConnection1.Close()
DataGrid1.DataBind()
End If
```

- **ViewState** je kolekce reprezentující stav stránky a všech obsažených ovládacích prvků. Stav prvků se neukládá na serveru, ale "ViewState" cestuje na klienta a zpět coby skryté pole formuláře. To umožňuje snadnou vizualizace pomocí trasování. Možnost volby – ViewState je:

  OFF – pro případ nastavení hodnoty, výpočty, databáze atd. při každém PostBacku

  ON – úspora výpočetního času, ale více kB putujících mezi klientem a serverem

  (Zakažte ViewState pokud není třeba, ušetříte kB mezi klientem a serverem – v případě, že stránka nepoužívá události, nenastavujete vlastnosti ovl.prvků dynamicky, nebo dynamické vlastnosti se nastavují při každém zobrazení stránky opakovaně). Vypnutí

stavu stránky se provede pro ovládací prvek: `<asp:DataGrid EnableViewState=false … />`

Pro celou stránku: `<%@ Page EnableViewState=false … %>`

Pro aplikaci, v souboru web.config: `<Pages enableViewState="false" />`

5.4.4 ASP.NET 2.0

Verze ASP.NET 2.0 přináší mnoho změn – zavádí silnou podporu deklarativnímu programování (např. objekty pro zpracování dat z databázových i obecnějších zdrojů) a nových technik, s možností například:

- vytvářet uživatelské třídy odvozené z předka *Page* s uživatelskými ovládacími prvky na bázi vzorů (templates), umístěnými i ve sdílených oblastech stránek (hlavičky, menu, …),
- koncept *master pages*, vzorových stránek, které tvoří podklad pro společný vzhled stránek webové aplikace,
- nové možnosti uživatelské modifikace stylu stránky pomocí *skins a themes*. Skin je skupina vzorů a vlastností, aplikovaná na ovládací prvky stránky, theme je kontejnerem pro skiny, obrázky, stylové a další soubory. Umístěn je standardně v presaři pod adresářem Themes.
- Individuální uživatelské nastavení struktury webu typu portál pomocí prvků WebPartZone a WebPart.
- Větší podpora zabezpečení aplikace prostřednictvím ovladacích prvků jako Login, LoginStatus, LoginName, …
- Nový způsob navigace ve webové aplikaci (Site Maps).
- Nová skupina prvků pro práci s daty - Data Source Controls, GridView, DetailsView, DropDownList, … s deklarativním způsobem určení parametrů (nemusí se používat klacický zdrojový kód ve zvoleném jazyce s použitím metod vhodných datových objektů).

Data Source Controls (implementovány ve jmenném prostoru System.Web.UI.WebControls) se dělí podle typu zpracovávaných dat na:

SqlDataSource (SQL Server provider ve jmenném prostoru `SqlClient`, databáze připojené přes OLE-DB nebo ODBC)

AccessDataSource připojení do Microsoft Access databáze

XmlDataSource umožňuje operace nad daty v hierarchických XML dokumentech

DataSetDataSource umožňuje operace nad daty v speciálních „nehierarchických“ XML dokumentech, které obsahují např. data z relačních tabulek (ADO.NET DataSet, XmlDataDocument)

ObjectDataSource umožňuje připojení vhodných uživatelských objektů, např. aplikační vrstvy.

5.4.5 Webové služby (web-service)

webové služby – souhrnné označení pro sadu technologií umožňujících komunikaci mezi aplikacemi. Hlavním účelem webové služby je vytvoření opakovaně využitelné komponenty, přístupné a využitelné na Internetu prostřednictvím protokolu HTTP. Webová služba je tedy jednoduchá komponenta, nabízející určitou službu – například nabídku zboží, převod měn, zjištění kurzu akcie, zpracování objednávky, překlad textu. Mezi základní vlastnosti patří schopnost nabídnout svou funkcionalitu a vlastnosti ve formě vlastního popisu, který je srozumitelný a využitelný v dalších aplikacích. Umožňují dalším aplikacím svou snadnou lokalizaci na Internetu, jsou typicky elektronicky registrovány v obdobě internetových „zlatých stránek“ (UDDI). Po nalezení mohou být využity s pomocí internetových standardních protokolů pří formulaci požadavků i při zpracování výsledků z odpovědí služby.

Technologie webových služeb je používaná pro distribuované vícevrstvé aplikace.

Porovnání a charakteristika webové služby proti technologiím Corba, DCOM, RMI:

- zcela nezávislé na použité platformě
- dostupné globálně, umožňují snadnou integraci různých prostředí
- postaveny na jednoduchých technologiích jako XML a HTTP
- méně efektivní než „nativní“ protokoly

XML webová služba je postavena na formátu XML se třemi základními složkami

- SOAP (Simple Object Access Protocol) – protokol na bázi XML pro přenos zpráv při přenášení strukturovaných informací, podporující různé způsoby výměny informací (včetně RPC - Remote Procedure Call), v prostředí distribuovaných a decentralizovaných aplikací. Zajišťuje nezávislost na protokolu, jazyku, operačním systému i platformě. Vzdálené volání služby (funkce) je definováno jako jednoduchá XML zpráva, která se obvykle přenáší pomocí HTTP protokolu. SOAP definuje základní strukturu XML obálky a způsob mapování datových typů do XML. Příklad vzoru zprávy ukazuje strukturu SOAP Envelope :

```
<soap:Envelope
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header> <!-- optional -->
    <!-- header blok... -->
  </soap:Header>
  <soap:Body>
    <!—elementy ... -->
  </soap:Body>
</soap:Envelope>
```

SOAP definuje zprávu jako element Envelope a uživatel formuluje pomocí XML specifické formáty sekce Header a Body.
Pro přenos dat je možné použít i standardní HTTP-GET, HTTP-POST metody pro jednoduché datové typy.

- WSDL(Web Services Description Language) – jazyk pro popis webové služby v XML formátu (popis rozšířené gramatiky webové služby dostupné přes SOAP), popisující zprávy i s převodem XSD pro získání obsahu zpráv – informace orientované na RPC i samotné zprávy, definice datových typů, publikované metody. Popisuje tedy, co služba dokáže, jak se dá připojit, zpustit a ovládat. Dokument obsahuje části typu:

  popis zprávy (XML Schema), operace – seznam zpráv v pruběhu zpracování jedné celé zprávy = operace (např. typu dotaz – odpověď: dvě zpravy), PortType, Port, Provázání(binding), Služba(Service). Z WSDL definice lze automaticky generovat

klientský kód (aplikace vytváří na straně klienta zástupce - proxy webové služby), který umí službu využívat, i skeleton serverového kódu.

- UDDI(Universal Description, Discovery, and Integration) – univerzální adresář obsahující seznam a popis dostupných webových služeb pro služby registrující webové služby (nová webová služba je uložená v příslušném adresáři a tak je zveřejněna), formulují nezbytné technické základy pro zveřejnění a nalezení implementace webové služby - umožňuje automatické nalezení požadované služby (komunikace, tj. požadavek o informacích z adresáře a odpověď je realizována formou SOAP protokolu).

  Rozlišují se dva typy registrů, veřejné a soukromé – používají se například v rámci jedné organizace.

Příklad požadavků ze SOAP zprávy:

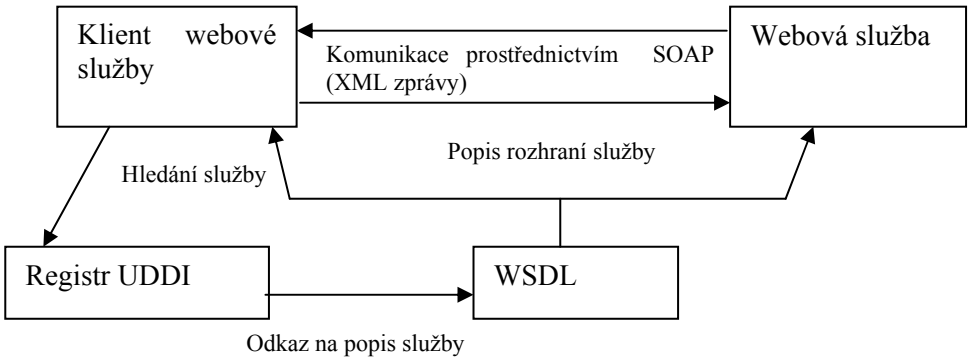
-find_Tmodel (definice WDSL)

-find_business (umístění služby)

-get_serviceList

-get_bindingDetail

Schéma práce s webovou službou:



Obrázek 25 Práce s webovou službou

Příklad implementace webové služby v prostředí .NET Framework:

Soubor AuthorsService.asmx.cs

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Diagnostics;
using System.Web;
using System.Web.Services;

namespace AuthorsWebService
{
	/// <summary>
	/// Summary description for Service1.
	/// </summary>
	public class AuthorsService : System.Web.Services.WebService
	{
		public AuthorsService()
		{
			//CODEGEN: This call is required by the ASP.NET Web
Services Designer
			InitializeComponent();
		}

		private System.Data.SqlClient.SqlCommand sqlSelectCommand1;
		private System.Data.SqlClient.SqlCommand sqlInsertCommand1;
		private System.Data.SqlClient.SqlCommand sqlUpdateCommand1;
		private System.Data.SqlClient.SqlCommand sqlDeleteCommand1;
		private System.Data.SqlClient.SqlConnection sqlConnection1;
```

```
            private System.Data.SqlClient.SqlDataAdapter sqlDataAdapter1;

            #region Component Designer generated code

            //Required by the Web Services Designer
            …

            #endregion

            [WebMethod]
            public authors1 GetAuthors()
            {
                  authors1 authors = new authors1();
                  sqlDataAdapter1.Fill(authors);
                  return authors;
            }
            [WebMethod]
            public authors1 UpdateAuthors(authors1 authorChanges)
            {
                  if (authorChanges != null)
                  {
                        sqlDataAdapter1.Update(authorChanges);
                        return authorChanges;
                  }
                  else
                  {
                        return null;
                  }
            }


      }
}
```

Soubor AuthorsService.disco:

```
<?xml version="1.0" encoding="utf-8"?>
<discovery xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://schemas.xmlsoap.org/disco/">
  <contractRef ref="http://localhost/AuthorsWebService/AuthorsService.asmx?wsdl"
docRef="http://localhost/AuthorsWebService/AuthorsService.asmx"
xmlns="http://schemas.xmlsoap.org/disco/scl/" />
  <soap address="http://localhost/AuthorsWebService/AuthorsService.asmx"
xmlns:q1="http://tempuri.org/" binding="q1:AuthorsServiceSoap"
xmlns="http://schemas.xmlsoap.org/disco/soap/" />
</discovery>
```

Soubor AuthorsService.wsdl:

```
<?xml version="1.0" encoding="utf-8"?>
<wsdl:definitions xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:s="http://www.w3.org/2001/XMLSchema"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:i0="http://www.tempuri.org/authors1.xsd" xmlns:tns="http://tempuri.org/"
xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/" targetNamespace="http://tempuri.org/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  <wsdl:import namespace="http://www.tempuri.org/authors1.xsd"
location="http://localhost/AuthorsWebService/AuthorsService.asmx?schema=authors1" />
  <wsdl:types>
    <s:schema elementFormDefault="qualified" targetNamespace="http://tempuri.org/">
      <s:import namespace="http://www.tempuri.org/authors1.xsd" />
      <s:element name="GetAuthors">
        <s:complexType />
      </s:element>
      <s:element name="GetAuthorsResponse">
        <s:complexType>
          <s:sequence>
```

```
          <s:element minOccurs="0" maxOccurs="1" name="GetAuthorsResult">
            <s:complexType>
              <s:sequence>
                <s:any namespace="http://www.tempuri.org/authors1.xsd" />
              </s:sequence>
            </s:complexType>
          </s:element>
        </s:sequence>
      </s:complexType>
    </s:element>
    <s:element name="UpdateAuthors">
      <s:complexType>
        <s:sequence>
          <s:element minOccurs="0" maxOccurs="1" name="authorChanges">
            <s:complexType>
              <s:sequence>
                <s:any namespace="http://www.tempuri.org/authors1.xsd" />
              </s:sequence>
            </s:complexType>
          </s:element>
        </s:sequence>
      </s:complexType>
    </s:element>
    <s:element name="UpdateAuthorsResponse">
      <s:complexType>
        <s:sequence>
          <s:element minOccurs="0" maxOccurs="1" name="UpdateAuthorsResult">
            <s:complexType>
              <s:sequence>
                <s:any namespace="http://www.tempuri.org/authors1.xsd" />
              </s:sequence>
            </s:complexType>
          </s:element>
        </s:sequence>
      </s:complexType>
    </s:element>
  </s:schema>
</wsdl:types>
<wsdl:message name="GetAuthorsSoapIn">
  <wsdl:part name="parameters" element="tns:GetAuthors" />
</wsdl:message>
<wsdl:message name="GetAuthorsSoapOut">
  <wsdl:part name="parameters" element="tns:GetAuthorsResponse" />
</wsdl:message>
<wsdl:message name="UpdateAuthorsSoapIn">
  <wsdl:part name="parameters" element="tns:UpdateAuthors" />
</wsdl:message>
<wsdl:message name="UpdateAuthorsSoapOut">
  <wsdl:part name="parameters" element="tns:UpdateAuthorsResponse" />
</wsdl:message>
<wsdl:portType name="AuthorsServiceSoap">
  <wsdl:operation name="GetAuthors">
    <wsdl:input message="tns:GetAuthorsSoapIn" />
    <wsdl:output message="tns:GetAuthorsSoapOut" />
  </wsdl:operation>
  <wsdl:operation name="UpdateAuthors">
    <wsdl:input message="tns:UpdateAuthorsSoapIn" />
    <wsdl:output message="tns:UpdateAuthorsSoapOut" />
  </wsdl:operation>
</wsdl:portType>
<wsdl:binding name="AuthorsServiceSoap" type="tns:AuthorsServiceSoap">
  <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document" />
  <wsdl:operation name="GetAuthors">
    <soap:operation soapAction="http://tempuri.org/GetAuthors" style="document" />
    <wsdl:input>
      <soap:body use="literal" />
    </wsdl:input>
    <wsdl:output>
      <soap:body use="literal" />
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="UpdateAuthors">
    <soap:operation soapAction="http://tempuri.org/UpdateAuthors" style="document" />
    <wsdl:input>
      <soap:body use="literal" />
    </wsdl:input>
    <wsdl:output>
```

```
        <soap:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="AuthorsService">
    <documentation xmlns="http://schemas.xmlsoap.org/wsdl/" />
    <wsdl:port name="AuthorsServiceSoap" binding="tns:AuthorsServiceSoap">
      <soap:address location="http://localhost/AuthorsWebService/AuthorsService.asmx" />
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

6 Závěr

7 Seznam literatury

[Lacko02] LACKO, B. *Projektování řídicích systémů*. Abstrakt k předmětu. Brno, říjen 2002

[Pokorný99] POKORNÝ, J. *Konstrukce databázových systémů*. 1. vyd. ČVUT, Zikova 4 Praha 6, Praha, 1999. ISBN 80-01-01935-8.

[Pokorný92] POKORNÝ, J. *Databázové systémy a jejich použití v informačních systémech* . 1. vyd. Akademia Praha, Praha, 1992. ISBN 80-200-0177-8.

[Pokorný99] POKORNÝ, J. *Konstrukce databázových systémů*. 1. vyd. ČVUT, Zikova 4 Praha 6, Praha, 1999. ISBN 80-01-01935-8.

[Thangarathinam06] Thangarathinam T. *Professional ASP.NET 2.0 XML*. Wiley Publishing, Inc., 2006. ISBN 0-7645-9677-2

[Homer03] Homer A., Sussman D., Howard R. *A First Look at ASP.NET v. 2.0* . Addison Wesley, 2003. ISBN 0-321-22896-0

8 Seznam obrázků

Obr. 1 Norbert Wiener 9
Ob. 2 Popis systému 10
Obr. 3 Prvek systému 11
Obr. 4 Spojitá a diskrétní funkce 12
Obr. 5 Vytváření matematického modelu 14
Obr. 6 Přenos zprávy – určení množství informace 22
Obrázek 7 Příklad Jacksonova strukturogramu 58
Obr. 8 Datový sklad (Data Warenhouse – DWH) 67
Obrázek 9 Třívrstvá architektura s technologií COM/DCOM 77
Obrázek 10 Zjednodušený objektový model ADO 78
Obrázek 11 Zjednodušený model ADO. NET 80
Obrázek 12 Architektura aplikace s ADO. NET 82
Obrázek 13 dvouvrstvá architektura Java 89
Obrázek 14 Třívrstvá architektura Java 90
Obrázek 15 Hypertextový dokument 92
Obrázek 16 Tecnologie publikování na webu 94
Obrázek 17 Vývoj značkových jazyků 95
Obrázek 18 Hierarchie tříd v .NET DOM 112
Obrázek 19 Použití kaskádových stylů 123
Obrázek 20 Základní vestavěné objekty ASP 149
Obrázek 21 Práce s webovou službou 158

9 Rejstřík

slovo, 2