

VYSOKÁ ŠKOLA EKONOMICKÁ V PRAZE

---

Fakulta informatiky a statistiky

Katedra informačního a znalostního inženýrství



# Využití XML při grafické prezentaci ekonomických informací

Bakalářská práce

**Jakub Vojtíšek**

Vedoucí práce: Ing. Jiří Kosek

---

leden 2007

# Anotace

Práce se zabývá možnostmi popisu ekonomických a jiných dat pomocí XML a jejich zobrazováním ve vektorovém grafickém formátu SVG. V práci jsou navrženy tři typy XML schémat pro různé druhy grafů. Grafy jsou zde systematicky rozděleny nikoliv podle jejich vzhledu, ale podle typu dat respektive počtu a typu datových řad, kterými jsou tvořeny, na:

- graf s jednou měřitelnou datovou řadou a neměřitelnou řadou názvů (OSGR)
- graf tvořený více měřitelnými datovými řadami a neměřitelnou řadou názvů (MSGR)
- graf s několika dvojicemi měřitelných datových řad (XYGR)

V návrhu schémat se podařilo oddělit informace o vzhledu grafů od samotných dat a tím umožnit snadnou změnu vzhledu pomocí malých změn v XML souboru.

Hlavním přínosem práce je sada tří skriptů, pro každé navržené XML schéma jeden, které převádějí data a informace o způsobech jejich zobrazení do grafické podoby ve formátu SVG. Skripty využívají vylepšených programovacích možností nové verze 2.0 jazyka XSLT a v něm používaného dotazovacího jazyka XPath 2.0. Jsou navrženy tak, aby automaticky počítaly velikost rozměrů grafu, rozmístění jeho částí a meze datových os. Různým natavením atributů lze měnit výsledný vzhled grafu. Datové řady mohou být znázorněny jako několik druhů sloupců s volitelným 3D efektem, pomocí různých značek, či jako lomené nebo vyhlazené čáry mnoha typů s možností výplně ohraničených ploch. Řady lze barevně odlišit a samostatně nastavit i různý vzhled řad v jednom grafu. Tím je dosaženo velké variability zobrazovacích možností. Grafy s jednou datovou řadou (OSGR) mohou být navíc vykresleny pomocí výsečového grafu, víceřadové grafy přepočteny na skládaný nebo procentní skládaný graf. Grafy s dvojicemi měřitelných řad (XYGR) mohou být dobře použitelné při kreslení grafů funkcí, křivek nebo výsledků měření.

Skripty jsou spolu s ukázkami jejich použití a dokumentace volně k dispozici na adrese <http://code.google.com/p/graph2svg/> a tím umožněno jejich libovolné používání a další vývoj.

# Annotation

This thesis deals with possibilities of description of economic and other data in XML and their representation in the vector graphic format SVG. Three types of XML schemas are defined for different types of graphs or charts. The graphs are divided systematically into three types not according to their visual form but according to the number of data series which they are formed with:

- graphs with one measurable data series and one unmeasurable name series (OSGR)
- graphs with more measurable data series and one unmeasurable name series (MSGR)
- graphs with several pairs of measurable data series (XYGR)

The aim of proposed schemas is to separate information about the graph appearance from their data and so to enable the easy change of appearance by means of small changes in a XML file.

The main asset of the thesis is the set of three scripts each script for one XML schema which transform data and information about the ways of their representation into the graphic form in the SVG format. The scripts make use of improved programming possibilities of the new version of XSLT 2.0 language and XPath 2.0 query language which is used in it. They are proposed to calculate the size of graphs, the layout of graph parts and limits of data axes automatically. The final appearance of a graph can be changed by means of the setting of several attributes. The data series can be depicted as several types of columns with the optional 3D effect, as various marks or as pointed or smooth lines of many types with the possibility of the filling of surrounded areas. The series can be colour-coded and it is possible to set their individual appearance as well. It enables the great variability of depicting possibilities. Besides the graphs with one measurable data series (OSGR) can be depicted as a pie chart. Graphs with more series (MSGR) can be transformed into a stacked or 100% stacked graph. Graphs with pairs of measurable data series (XYGR) can be used efficiently for the drawing of function graphs, curves or measurement results.

The scripts, the examples of their use and the documentation are available at <http://code.google.com/p/graph2svg/> and allowed for any application and further development.

# **Poděkování**

Rád bych poděkoval všem, kteří mě podporovali ve studiu a při psaní této práce.

# **Prohlášení**

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a použil pouze literaturu uvedenou v příloženém seznamu. Nemám námitek proti půjčení práce se souhlasem katedry ani proti zveřejnění práce nebo její části.

V Praze dne 4. ledna 2007

Jakub Vojtíšek

# Obsah

Úvod .....	7
1. Typy grafů a jejich parametry .....	9
1.1. Rozdělení grafů podle typu dat .....	9
1.2. Graf typu OSGR .....	10
1.3. Grafy typu MSGR .....	14
1.4. Graf typu XYGR .....	17
2. Použití grafů generovaných z XML .....	19
2.1. Ukázka grafů typu OSGR .....	19
2.2. Příklady použití MSGR grafů .....	21
2.3. Tvorba grafů typu XYGR .....	25
2.4. Automatické generování grafů z HTML tabulek .....	28
3. Jak jsou styly naprogramovány .....	31
3.1. Instalace a základní použití stylů .....	31
3.2. Celková architektura stylů .....	31
3.3. Význam důležitých proměnných .....	33
3.4. Popis některých funkcí a pojmenovaných šablon .....	34
Závěr .....	36
Literatura .....	37

# Úvod

Pro fungování každé firmy nebo společnosti je nezbytné pracovat s daty a informacemi. Uchovávat je, zpracovávat a na základě nich se rozhodovat nebo je předávat dál. Pro všechny tyto činnosti je potřeba data nějakým způsobem ukládat. A to nejen samotná data, ale i informace o jejich významu.

K tomu je velmi vhodný jazyk XML (eXtensible Markup Language), viz [3] nebo [4]. Je to univerzální značkovací jazyk, tedy soubor pravidel, jak tvořit značky, v kterém si může každý vytvořit vlastní sadu značek, pomocí níž bude uchovávat data a přikládat k nim význam. V tom spočívá velká variabilita jazyka XML. Kromě toho jsou v XML definovány mnohé standardní sady značek, které jsou již široce používány. Například XHTML pro tvorbu webových stránek, MathML pro zapisování vzorců nebo DocBook pro tvorbu textů a dokumentace, v kterém je psána i tato práce. XML přitom není vázán na žádný komerční produkt, ale je spolu s dalšími jazyky založenými na XML, vyvíjen nezávislým W3C konsorciem a je každému volně přístupný. XML má spoustu výhod: je jednoduchý, rozšířený, snadno převoditelný, dobře podporován ve vývojových nástrojích, a proto je snadné implementovat jeho podporu v aplikacích a podnikových systémech.

Například firma zabývající se vývojem software může pro psaní dokumentace svých produktů používat DocBook, který je pro tyto účely navržený. Takto uchovaná a označovaná data pak lze podle potřeby převést do HTML pro online nápovědu nebo z nich generovat PDF pro tisk uživatelské příručky. Pro tyto běžné převody samozřejmě existuje spousta nástrojů. Jedním z nich je XSLT (eXtensible Stylesheet Language Transformations, viz kniha [5] nebo specifikace [6]), což je na XML založený jazyk popisující převod XML dat do jiných typů XML dat nebo souborů. Takovéto transformační styly jsou už pochopitelně pro běžně používané XML styly naprogramované a dokonce často volně přístupné.

Často potřebujeme zobrazit nebo zveřejnit nějaká data graficky, aby byla snáze interpretovatelná. Může to být kupříkladu tabulka tržeb v různých odvětvích v průběhu roku. V tabulce sice přesně vidíme jak velké jsou jednotlivé hodnoty, horší už to ale je s jejich porovnáním a nebo odhadnutím jejich vývoje. To bychom lépe viděli z grafu.

Nástrojů pro tvorbu grafů existuje celá řada. Ale pokud pracujeme s daty v podobě XML a pokud chceme vytvářet grafy z dat automaticky, bychom potřebovali mít nástroj, který umí převést data do grafické podoby, a to ve vhodném formátu. Takovým vhodným grafickým formátem může být SVG (Scalable Vector Graphics, viz [11]). SVG je škálovatelný vektorový formát navržený W3C konsorciem původně pro potřeby internetu. Absence nezávislého formátu vektorové grafiky slibuje možnost rozšíření SVG nejen ve světě internetu. Jeho obliba a podpora rychle narůstá (nové internetové prohlížeče už SVG umí zobrazit), stejně tak existuje mnoho aplikací s podporou SVG a vznikají další. Obrázek grafu popsany v SVG je oproti rastrovým formátům malý, lze jej přizpůsobit výslednému médii, může být vložen přímo do jiného XML souboru (například XHTML), vygenerován z něj PDF nebo PS soubor pro tisk a nebo převeden do rastrové podoby pro média, která vektorovou grafiku nepodporují.

Jak jsme už uvedli, k transformaci XML souborů se široce používá jazyk XSLT. K samotnému převodu je ovšem potřeba kromě XSLT stylu i program zvaný XSLT procesor, který převod provede. Těchto programů existuje celá řada a to nejen profesionálních komerčních, ale i kvalitních volně přístupných. V jazyce XSLT ve své původní verzi 1.0 už od počátku chyběly mnohé funkce, přesto se dobře uchytil a stále se používá. Tvůrci jednotlivých procesorů chybějící funkce přesto implementovali pomocí různých rozšíření. Tato rozšíření má sjednotit až nově vycházející verze 2.0 jazyka XSLT (viz specifikace [7] nebo tutoriál [8]), která je ve fázi „Proposed Recommendations“, tedy těsně před definitivním schválením. Tato verze je už v některých procesorech implementována.

XSLT stylů pro převod XML dokumentů je vyvinuto a používáno mnoho. Horší je to už s převodem případných grafických prvků do výsledného dokumentu. V praxi se přitom často setkáváme s daty, která je lépe prezentovat v grafické podobě. Jedná se především o různé typy grafů.

A tento převod je právě cílem této práce. Chceme navrhnout schéma XML dokumentů pro zápis dat, která jsou zobrazitelná formou grafu a transformační styly, které je převedou do formátu SVG. K transformaci používáme jazyk XSLT 2.0 a procesor Saxon 8.8, který pracuje s XSLT 2.0 a v něm používaném dotazovacím jazyce XPath 2.0 (viz [9]).

Celá práce je rozdělena do tří kapitol. V první kapitole je nejprve provedena klasifikace datových řad a podle ní navrženy typy XML souborů reprezentující grafy. Každý typ je zde popsán, uvedeno jeho DTD, vysvětlen význam jeho elementů a atributů a specifikován jejich vliv na vzhled grafu. U některých atributů je na obrázku znázorněna podoba jejich možných hodnot.

Druhá kapitola obsahuje příklady použití stylů a ilustruje na nich význam různých hodnot atributů. Jsou zde uvedeny i komplexnější ukázky grafů. Na závěr je uveden příklad jednoduchého XSLT stylu, který transformuje vybranou tabulku z XHTML stránky do grafické podoby s použitím stylu pro MSGR graf.

V poslední kapitole se zabýváme tím, jak se styly používají a jak jsou naprogramovány. Cílem nebylo detailně popsat celý kód, ale spíše vytvořit základní představu o stylech, některých jejich funkcích a významu důležitých proměnných.

Při zkoumání a popisování typů grafů a jejich částí se často setkáváme s mnoha různými pojmy. Vycházíme při tom z obsáhlé a vyčerpávající knihy [1]. V rychle se rozvíjejících vědách, mezi které informatika bezesporu patří, bývá problémem používání, definování a správného přeložení užívaných pojmů do češtiny. Pro české pojmy používané v této práci jsme vesměs vycházeli z [2], která popisuje tvorbu grafů v kancelářském balíku MS Office. Závisí sice na konkrétním softwareovém produktu, ale je u nás značně rozšířen a proto jsou používané pojmy dostatečně známé.



# Kapitola 1

## Typy grafů a jejich parametry

### 1.1. Rozdělení grafů podle typu dat

Existuje nepřeberné množství typů grafů. Jak je ale třídit tak, aby bylo snadné vybrat pro daný účel nejvhodnější graf? V různých publikacích a programech jsou grafy tříděny převážně podle jejich vzhledu, častokrát i docela nelogicky. My konstruujeme klasifikaci grafů na základě typu dat, která graf zobrazuje.

Základním pojmem, který používáme je datová řada. V [1] by pojmu datová řada nejlépe odpovídal termín „data set“. „Date series“ je již chápáno jako spojení dvou nebo více „data sets“ stejné délky, které spolu souvisí. V této práci budeme datovou řadou rozumět uspořádanou množinu (posloupnost) dat nejrůznějšího typu (text, číslo, datum, ...). Podle charakteru dat lze rozlišit následující typy datových řad:

#### *neuspořádaná*

Jedná se o řadu dat, u kterých nemá smysl říkat, že je některý prvek větší než jiný. Například: Polsko, Česko, Slovensko, ...

#### *uspořádaná*

Řada dat, kterou lze rozumně uspořádat, ale nelze říct o kolik je jeden prvek větší než jiný. Například: leden, únor, březen, duben, ...

#### *měřitelná*

Typicky řada číselných (obecně reálných) hodnot. S těmito daty lze provádět matematické operace, tedy například odčítat a zjišťovat tak o kolik nebo kolikrát je jedna hodnota větší než druhá.

Jednotlivé typy grafů nyní rozlišíme podle počtu a typů datových řad, které graf zobrazuje. Pochopitelně vždy uvažujeme jen konečné datové řady. Všechny datové řady, které udávají jeden graf musí být stejné délky. Dále si všimněme, že neuspořádanou datovou řadu můžeme pro naše účely ztotožnit s uspořádanou. Neexistuje sice uspořádání dané povahou jejích prvků, ale přesto jsou jednotlivá data v určitém pořadí zadána. V praxi se můžeme setkat s následujícími typy grafů:

*OSGR* — jedna měřitelná datová řada, jedna uspořádaná datová řada

Tento typ grafů zobrazuje jednu datovou řadu měřitelných hodnot. Druhá datová řada uspořádaných hodnot popisuje názvy kategorií (category scale) jednotlivých údajů. Nezahr-

nujeme jej do následující obecnější skupiny grafů, protože má jistá specifika, která by obecnější případ nezachytil. Může být například zobrazen jako výsečový graf (pie chart), který v obecnějším případě ztrácí smysl. Označení OSGR, které budeme pro tento typ grafu používat je zkratkou anglických slov one series graph.

*MSGR* —  $N$  měřitelných datových řad ( $N > 1$ ), jedna uspořádaná datová řada

Tento typ grafů obsahuje několik stejně dlouhých datových řad, které lze vzájemně srovnávat v závislosti na jednotlivých kategoriích. Budeme jej označovat MSGR — multi-series graph

*XYGR* —  $N$  dvojic měřitelných datových řad

V tomto typu grafů je zadáno několik dvojic datových řad. Obě datové řady v každé dvojici musí být stejně dlouhé a představují souřadnice bodů roviny. Jednou dvojicí je zadána řada bodů, které lze spojit v křivku. Tímto způsobem lze zobrazit grafy funkcí a křivek v rovině, přičemž každá z křivek může být pojmenována. Ve shodě s [2] označujeme tento typ grafů XYGR.

Další speciální typy grafů

Pomocí datových řad můžeme klasifikovat i další typy grafů, které mohou mít různé speciální použití. Například u bublinového grafu (bubble graph) jsou zadány tři měřitelné řady (souřadnice bublin a jejich velikosti) a jedna uspořádaná řada (názvy). Některé grafy používané v statistice pro zobrazení vývoje cen akcií, rozptylů časových řad a podobně jsou tvořeny čtyřmi i více měřitelnými datovými řadami se speciálním významem. Generováním těchto typů grafů se ovšem kvůli jejich úzkému použití v této práci nezabýváme. V třetí části ukážeme, že některé z těchto typů grafů lze zobrazit jako graf MSGR s vhodnými parametry.

Je pochopitelné, že data určitého typu mohou být zobrazena mnoha způsoby. Pro každý typ grafu nyní musíme navrhnout samostatné schéma XML, které bude vhodně reprezentovat data i vzhled grafů. Schémata budeme v následujícím textu popisovat pomocí DTD a slovního popisu jednotlivých elementů a atributů.

## 1.2. Graf typu OSGR

Tento graf zobrazuje pouze jednu řadu hodnot. Hodnoty jsou zobrazeny buď jako sloupce různých výšek, body nebo lomená čára spojující jednotlivé hodnoty. Data mohou být rovněž po přepočtení na procenta zobrazena pomocí výsečového grafu. Struktura XML dokumentu popisujícího graf typu OSGR je dána následujícím DTD.

```
<!ELEMENT osgr (title?, names?, values+)>
<!ELEMENT names (name+)>
<!ELEMENT values (value+)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT value (#PCDATA)>
<!ATTLIST osgr
  graphType (pie|norm) "norm"
```

```

effect (2D|3D) "2D"
colType (none|block|cylinder|cone|pyramid) "block"
lineType (none|solid|dot|dash|longDash|dash-dot|longDash-dot|
dash-dot-dot|longDash-dash|dash-dash-dot-dot) "none"
pointType (none|point|cross|plus|star|square|circle|triangle|rhombl|
pyramid|squareF|circleF|triangleF|rhomblF|pyramidF) "none"
smooth (yes|no) "no"
yAxisType (auto|withZero|shifted|log) "auto"
yAxisDivision (none|1|2|4|5|10) "1"
xAxisDivision (none|major|minor|both) "major"
xAxisPos (bottom|origin) "origin"
xGrid (none|major|minor|both) "none"
yGrid (none|major|minor) "none"
colorScheme (color|warm|cold|grey|black) "color"
legend (none|left|right|top|bottom) "right"
labelIn (none|value|percent|name) "none"
labelOut (none|value|percent|name) "none">
<!--ATTLIST title color CDATA #IMPLIED-->
<!--ATTLIST value
color CDATA #IMPLIED
pointType (none|point|cross|plus|star|square|circle|triangle|rhombl|
pyramid|squareF|circleF|triangleF|rhomblF|pyramidF) #IMPLIED-->

```

Nyní ještě potřebujeme popsat možné hodnoty jednotlivých parametrů a jejich vliv na vzhled grafu. Všimněme si, že některé vlastnosti grafu lze definovat na více místech XML souboru. V takovém případě má přednost hodnota atributu, která je u speciálnějšího elementu (tedy v pomyslné stromové struktuře XML dokumentu blíže listům). U OSGR grafu tuto vlastnost vidíme na attributech `colorScheme` a `color`. Atribut `colorScheme` je zadán u kořenového elementu a určuje posloupnost barev, které se přiřazují jednotlivým hodnotám (sloupcům, bodům, výsečím). Pokud je ovšem u některé hodnoty (u elementu `value`) specifikován atribut `color`, je příslušná hodnota zobrazena zadanou barvou a ne barvou z vybraného barevného schématu. Podobný princip je použit i u atributu `pointType`.

Následuje seznam atributů a jejich popis. Výčet možných hodnot každého atributu uvádíme za názvem atributu ve složených závorkách, kde je implicitní hodnota označena hvězdičkou. Implicitní hodnota se použije tehdy, pokud není atribut zadán, nebo je-li jeho hodnota nesmyslná. Hodnoty atributů odlišujeme od ostatního textu anglickými uvozovkami.

```
graphType {"pie", *"norm"}
```





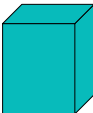
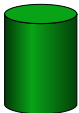


Pomocí tohoto atributu lze zvolit, jak bude celý graf vypadat. Typy jsou pouze dva: "pie" pro výsečový graf a "norm" pro ostatní grafy. Vzhled grafu typu "norm" je dán ostatními parametry, které se mohou dokonce „překrývat“. V extrémním případě mohou být tedy hodnoty zobrazeny pomocí sloupců v jejichž vrcholech jsou body spojené lomenou čarou.

```
effect {"*2D", "3D"}
```

Hodnotou atributu `effect` ovlivňujeme, jak budou vypadat sloupce, osy a mřížka grafu. Je-li zvolena hodnota "3D", jsou útvary zobrazeny prostorově (volným rovnoběžným promítáním). Prostorově lze zobrazit rovněž graf typu `pie`.

```
colType {"none", "*"block", "cylinder", "cone", "pyramid"}
```

Atribut ovlivňuje vzhled sloupců, případně mají-li být vůbec vykresleny. Nevykreslení značí volba "none". Ostatní hodnoty způsobí vykreslení sloupců příslušného tvaru. Přestože jsou názvy tvarů odvozeny od trojrozměrných útvarů, je potřeba si uvědomit, že každý typ sloupce má dvě možnosti vykreslení v závislosti na volbě atributu `effect`. Například volba "block" se při `effect = "3D"` zobrazí jako kvádr, zatímco v dvojrozměrné podobě grafu jako obdélník. Takto získáme velkou variabilitu vzhledu sloupcových grafů. Viz obrázek 1.1 – „Typy sloupců v závislosti na hodnotě atributu `effect`“.

colType	block	cylinder	cone	pyramid
effect = "2D"				
effect = "3D"				

**Obrázek 1.1. Typy sloupců v závislosti na hodnotě atributu `effect`**

```
lineType {"none", "solid", "dot", "dash", "longDash", "dash-dot", "longDash-dot", "dash-dot-dot", "longDash-dash", "dash-dash-dot-dot"}
```
























Není-li tento atribut nastaven na "none", jsou hodnoty dat vyneseny v grafu spojeny čarou daného typu. Typy čar vidíme na obrázku 1.2 – „Typy čar a bodů“.

```
smooth {"yes", "no"}
```

Atributem `smooth` můžeme nastavit vyhlazování čar. Datové body jsou proloženy hladkou (kubickou Bézierovou) křivkou. Řídící body této křivky jsou dopočteny tak, aby byla tečna ke křivce v daném bodě rovnoběžná s přímkou spojující sousední body.

```
pointType {"none", "point", "cross", "plus", "star", "square", "circle", "triangle", "rhomb", "pyramid", "squareF", "circleF", "triangleF", "rhombF", "pyramidF"}
```

Nastavení atributu `pointType` způsobí zobrazení zvoleného symbolu (viz obrázek 1.2 – „Typy čar a bodů“) v místě hodnoty dat. Atribut lze nastavit u elementu `osgr` pro všechny hodnoty grafu nebo jednotlivě u elementu `value`.

	solid		point		squareF
	dot		cross		circleF
	dash		plus		triangleF
	longDash		star		rhombF
	dash-dot		square		pyramidF
	longDash-dot		circle		
	dash-dot-dot		triangle		
	longDash-dash		rhomb		
	dash-dash-dot-dot		pyramid		

Obrázek 1.2. Typy čar a bodů

`yAxisType {"auto", "withZero", "shifted", "log"}`

Atribut určuje typ osy y, tedy svislé osy, na kterou se vynášejí hodnoty dat. Volba "withZero" znamená, že má osa vždy začínat od nuly. Pokud jsou všechny hodnoty dat velké, ale rozdíly mezi nimi malé, není zobrazení s nulou dostatečně názorné. V takovém případě je užitečné zvolit typ osy "shifted", který vhodně upraví začátek a konec osy. Implicitní volba "auto" provede automatický výběr mezi těmito hodnotami. Volba "log" je užitečná při řádově různých hodnotách dat. Data jsou přepočtena logaritmem o základu 10. Logaritmovat lze jen kladné hodnoty, a proto je ze záporných hodnot spočtena absolutní hodnota a nula je nahrazena jedničkou.

`yAxisDivision {"none", *"1", "2", "4", "5", "10"}`

Na ose mohou být hlavní značky, u kterých se zobrazují číselné hodnoty, a dále vedlejší značky, které pomáhají v přesnějším určení hodnoty a jsou zobrazeny méně výrazně. Atribut `yAxisDivision` udává, na kolik úseků bude úsek mezi sousedními hlavními značkami rozdělen pomocí vedlejších značek. Tedy například volba "4" rozdělí úsek mezi hlavními značkami třemi vedlejšími značkami na čtyři čtvrtiny. Pokud zůstane zvolena implicitní hodnota "1", nebudou vedlejší značky vykreslovány vůbec. Při volbě "none" nebudou vykreslovány ani hlavní značky. Je-li ovšem osa logaritmického typu (`yAxisType = "log"`), nemá dělení na stejné dílky smysl. Proto všechny hodnoty větší než jedna mají stejný efekt. Osa je rozdělena osmi vedlejšími značkami. Ty odpovídají násobkům menší hodnoty intervalu. Například v intervalu mezi hodnotami  $10^2$  a  $10^3$  jsou vedlejší značky v místě celých stovek.

`xAxisDivision {"none", *"major", "minor", "both"}`

Osa x v grafu typu OSGR slouží k vynášení obecně neměřitelných dat — názvů kategorií, proto je způsob jejího dělení jiný. Volba "major" zobrazuje značky mezi jednotlivými kategoriemi, naopak volba "minor" způsobí vykreslení menších značek v místě x-ové souřadnice vynášených hodnot. Volbami "none" nebo "both" můžeme buď potlačit zobrazování těchto značek nebo je zobrazit zároveň.

`xAxisPos {"bottom", *"origin"}`

Je-li dolní mez osy y menší než nula, připadají v úvahu dvě možnosti umístění osy x. Buď v místě, kde je na ose y zobrazena 0, nebo úplně dole v místě dolní meze osy y. Která možnost se použije závisí právě na hodnotě atributu `xAxisPos`.

`yGrid {"none", "major", "minor"}`

Umožňuje v poli grafu zobrazit mřížku, která usnadní odečítání hodnot. Volba "major" zobrazí horizontální čáry vycházející z hlavních značek osy y, volba "minor" k nim přidá i pomocné čáry vycházející z vedlejších značek. Jejich hustota je dána hodnotou atributu `yAxisDivision`.

`xGrid {"none", "major", "minor", "both"}`

Zobrazuje vertikální čáry mřížky, které vychází z odpovídajících značek osy x.

`legend {"none", "left", "right", "top", "bottom"}`

Způsobí vykreslení legendy na daném místě. V legendě se u OSGR grafů zobrazují názvy kategorií. V případě grafu typu "norm" ale nemá příliš význam legendu vykreslovat, protože se názvy kategorií zobrazují již na x-ové ose. K popisu výsečových grafů je ovšem legenda používána.

`labelIn {"none", "value", "percent", "name"}`

U grafů typu OSGR lze nastavit, aby se uvnitř sloupců (v jejich polovině) vypisovaly doplňující údaje a to buď přímo hodnoty dat, hodnoty dat přepočtené na procenta nebo názvy kategorií. Atribut má větší význam u výsečového grafu, kde se údaje vypisují uvnitř výsečí.

`labelOut {"none", "value", "percent", "name"}`

Atribut `labelOut` vypisuje totéž co `labelIn`, ale nad sloupce, v případě výsečového grafu vedle výsečí.

`colorScheme {"color", "warm", "cold", "grey", "black"}`

Pomocí tohoto atributu lze vybrat barevné schéma, podle kterého se obarví jednotlivé sloupce (případně výseče nebo značky). Je-li sloupců více než barev, začnou se barvy cyklicky opakovat. Přiřazené barvy mohou být změněny volbou atributu `color` u jednotlivých hodnot. Ve stylu je zabudováno pět barevných schémat: "color" pro výrazné barvy, "warm" pro barvy teplého charakteru, "cold" obsahující studené barvy, "grey" značící různé stupně šedi a "black" obsahující pouze černou barvu.

`color`

Hodnoty atributu `color` mohou být v jakémkoliv formátu podporovaném v SVG (hodnota atributu je přímo vložena do SVG). Tedy buď hexadecimální číslo, které je složené ze zastoupení tří barevných složek RGB (například: #FE7312, #f50), nebo název příslušné barvy (například: "blue", "red"). Atribut nemá implicitní hodnotu, není-li totiž uveden, je použita barva z barevného schématu (atribut `colorScheme`). Pokud se jedná o atribut elementu `title`, je implicitně použita černá barva.

### 1.3. Grafy typu MSGR

Jak už víme, graf typu MSGR slouží k popisu a zobrazení několika datových řad. Struktura tohoto typu grafu je v podstatě podobná předchozímu typu jak je vidět v následujícím DTD. Hlavní rozdíl je v tom, že element `values`, v kterém jsou definovány hodnoty dat, může být použit vícekrát. Od toho se pak odvíjí jiný význam některých atributů a jsou přidány některé další atributy, které ovlivňují zobrazení hodnot v rámci jednotlivých kategorií a jejich zpraco-

vání. Může být rovněž zobrazena legenda, která narozdíl od OSGR grafů obsahuje názvy jednotlivých řad. Ty jsou pro jednotlivé datové řady zadány elementem `title` uvnitř elementu `values`. Graf typu MSGR neobsahuje atribut `graphType`, protože výsečový graf není pro více řad použitelný, chybí zde i atributy `labelIn` a `labelOut`. Různý vzhled grafů je dán volbou jednotlivých atributů, jejichž kombinování je omezeno pokud možno co nejméně.

```
<!ELEMENT<!ELEMENT msgr (title?, names?, values+)>
<!ELEMENT names (name+)>
<!ELEMENT values (title?, value+)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT value (#PCDATA)>
<!ATTLIST msgr
  stacked (none|sum|percentage) "none"
  shift CDATA "0"
  effect (2D|3D) "2D"
  colType (none|block|cylinder|cone|pyramid) "none"
  lineType (none|solid|dot|dash|longDash|dash-dot|longDash-dot|
    dash-dot-dot|longDash-dash|dash-dash-dot-dot) "solid"
  pointType (none|point|cross|plus|star|square|circle|triangle|rhom|
    pyramid|squareF|circleF|triangleF|rhom|pyramidF) "none"
  fillArea (yes|no) "no"
  yAxisType (auto|withZero|shifted|log) "auto"
  yAxisDivision (none|1|2|4|5|10) "1"
  xAxisDivision (none|major|minor|both) "minor"
  xAxisPos (bottom|origin) "origin"
  xGrid (none|major|minor|both) "none"
  yGrid (none|major|minor) "none"
  legend (none|left|right|top|bottom) "right"
  colorScheme (color|warm|cold|grey|black) "color">
<!ATTLIST title color CDATA #IMPLIED>
<!ATTLIST values
  color CDATA #IMPLIED
  colType (none|block|cylinder|cone|pyramid) #IMPLIED
  lineType (none|solid|dot|dash|longDash|dash-dot|longDash-dot|
    dash-dot-dot|longDash-dash|dash-dash-dot-dot) #IMPLIED
  pointType (none|point|cross|plus|star|square|circle|triangle|rhom|
    pyramid|squareF|circleF|triangleF|rhom|pyramidF) #IMPLIED
  smooth (yes|no) "no"
  fillArea (yes|no) #IMPLIED
  startFrom (axis|last) #IMPLIED>
```

Všechny atributy kořenového elementu `msgr` se vztahují k celému grafu a definují vlastnosti všech řad. Některé z nich mohou být změněny pomocí atributů elementu `values`, které mají větší prioritu. Okomentujme nyní ty atributy, které se nevyskytovaly v OSGR grafu, nebo mají jiný význam.

```
stacked {"none", "sum", "percentage"}
```

Tímto atributem můžeme ovlivnit, jak na hodnoty dat budeme nahlížet a jakým způsobem mají být zpracovány. Volba "none" znamená, že hodnoty zůstanou beze změny. Volbou "sum" říkáme, že ke každé hodnotě mají být přičteny hodnoty předchozích dat v dané kategorii. Pomocí této volby lze vytvářet takzvané skládané grafy (stacked). Volba "percentage" se chová podobně jako volba "sum" s tím rozdílem, že součty jsou navíc vyjádřeny jako procentní podíl ze součtu všech dat v dané kategorii. Využijeme ji tehdy, když nepotřebujeme znázorňovat absolutní velikost údajů, ale jejich podíl vůči ostatním datům. Na y-ovou osu pak nejsou zobrazeny absolutní velikosti hodnot, ale procenta. Grafy tohoto typu se nazývají procentní skládané (100% stacked). Poslední dvě možnosti rovněž způsobí, že sloupce (jsou-li vykreslovány) nezačínají od osy kategorií, ale od místa, kde končil předešlý sloupec. Podobně je tomu u plošného grafu (area chart). Toto chování lze ještě lokálně pro každou řadu změnit pomocí atributu `startFrom` elementu `values`.

```
shift {"0"}
```

Parametr udává posun datových sloupců uvnitř kategorií. Pokud je nastaven na 0 (implicitně) zobrazí se všechny sloupce nebo body v jedné kategorii ve stejném bodě osy x. U neskládaných sloupcových grafů je proto mnohdy vhodnější nastavit tuto hodnotu větší než 0, aby se sloupce nepřekrývaly. Je-li `shift` mezi 0 a 1 dochází k částečnému překrytí sloupců (overlap), pro `shift = "1"` jsou sloupce v každé kategorii těsně vedle sebe a je-li `shift` větší než 1, vykreslí se mezi sloupci mezera.

```
fillArea {"yes", "no"}
```

Atribut udává, jestli se má vyplňovat prostor mezi spojnicí bodů dané datové řady a x-ovou osou. Možnost "yes" je použitelná zejména s volbou `stacked = "sum"` nebo `stacked = "percentage"`. V tomto případě je vyplněn prostor mezi dvěma po sobě jdoucími řadami. Rovněž je-li nastaven atribut `startFrom` na hodnotu "last", je vyplňován prostor mezi spojnicemi dvou po sobě jdoucích řad.

```
legend {"none", "left", "right", "top", "bottom"}
```

Atributem `legend` můžeme určit, zda a kam se má zobrazit legenda grafu. V legendě jsou zobrazeny názvy datových řad a způsob jejich vykreslení. Datové řady, u nichž není zadáno jméno, se v legendě vůbec nezobrazí. Vykresluje-li se legenda vpravo ("right") nebo vlevo ("left"), jsou názvy řazené do sloupce. Zatímco u legendy zobrazované nahoře ("top") nebo dole ("bottom") jsou názvy zobrazovány do řádku.

```
startFrom {"axis", "last"}
```

Jak už bylo uvedeno výše, lze atributem `startFrom` ovlivnit, jak budou vykreslovány sloupce, a který prostor se má vyplnit, když je atribut `fillArea` nastaven na "yes". Toto chování lze nastavit pouze samostatně pro každou řadu. Atribut má svůj význam převážně v speciálních způsobech použití stylu MSGR, viz příklady.

Narozdíl od grafu OSGR, který se implicitně zobrazil jako sloupcový, je graf MSGR implicitně zobrazen jako spojnicový, čemuž odpovídá tato volba atributů elementu `msggr: colType = "none"`, `lineType = "solid"`, `pointType = "none"`.



Protože může být vzhled grafu vytvořen kombinací různých prvků, je důležité určit, v jakém pořadí se budou vykreslovat. Nejdříve budou vykresleny plochy u těch řad u kterých je `fillArea = "yes"`. Poté se vykreslí všechny sloupce, dále čáry a nakonec body.

## 1.4. Graf typu XYGR

Pro tento typ grafu je typické, že zobrazuje vždy dvě stejně dlouhé řady měřitelných hodnot. Hodnoty jedné řady vynášíme na x-ovou osu a druhé na y-ovou osu. Tímto vznikne posloupnost bodů v rovině, které mohou být spojeny lomenou čarou. Může být zadáno i více dvojic datových řad, což umožňuje do jednoho grafu zadat více křivek. Protože má tento graf poněkud jinou povahu než předchozí typy, zvolili jsme pro něj odlišný způsob uspořádání dat, viz následující DTD. Souřadnice bodů jsou zadávány jako atributy `x` a `y` elementu `point`. Všechny body jedné křivky jsou pak uzavřeny do elementu `curve`. Data tedy nejsou seskupena podle datových řad, ale podle jednotlivých bodů („kategorií“).

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT xygr (title?, curve+)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT curve (name?, point+)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT point EMPTY>
<!ATTLIST xygr
  lineType (none|solid|dot|dash|longDash|dash-dot|longDash-dot|
    dash-dot-dot|longDash-dash|dash-dash-dot-dot) "solid"
  pointType (none|point|cross|plus|star|square|circle|triangle|rhombl
    pyramid|squareF|circleF|triangleF|rhomblF|pyramidF) "none"
  axesPos (left-botom|origin) "origin"
  xAxisType (auto|withZero|shifted|log) "auto"
  yAxisType (auto|withZero|shifted|log) "auto"
  xAxisDivision (none|1|2|4|5|10) "1"
  yAxisDivision (none|1|2|4|5|10) "1"
  xGrid (none|major|minor) "none"
  yGrid (none|major|minor) "none"
  legend (none|left|right|top|botom) "none"
  colorScheme (color|warm|cold|grey|black) "black">
<!ATTLIST curve
  lineType (none|solid|dot|dash|longDash|dash-dot|longDash-dot|
    dash-dot-dot|longDash-dash|dash-dash-dot-dot) #IMPLIED
  pointType (none|point|cross|plus|star|square|circle|triangle|rhombl
    pyramid|squareF|circleF|triangleF|rhomblF|pyramidF) #IMPLIED
  smooth (yes|no) "no"
  color CDATA #IMPLIED>
<!ATTLIST name
  x CDATA #IMPLIED
  y CDATA #IMPLIED
  color CDATA #IMPLIED
```

```
visibility (none|legend|graph|both) "both">
<!ATTLIST point
  x CDATA #REQUIRED
  y CDATA #REQUIRED
  pointType (none|point|cross|plus|star|square|circle|triangle|rhombl|
    pyramid|squareF|circleF|triangleF|rhomblF|pyramidF) #IMPLIED
  color CDATA #IMPLIED>
```

Většinu atributů používaných v grafu typu XYGR známe z předchozích typů grafů. Nové jsou elementy `xAxisType`, `xAxisDivision`, `xGrid`, týkající se x-ové osy, na které nyní zobrazujeme rovněž měřitelné hodnoty. Jejich chování je proto stejné jako u již uvedených ekvivalentů pro osu y. Atribut `xAxisPos`, který v předešlých typech grafů ovlivňoval umístění x-ové osy je nahrazen atributem `axesPos`, který se týká obou os současně.

Všimněme si, že i zde je použito dědění některých vlastností a jejich případné překrytí vlastnostmi potomka. Například atributem `pointType` můžeme zvolit typ bodů pro všechny body v grafu, následně ho pro některou křivku změnit, nebo dokonce vykreslit jiným způsobem pouze jeden bod. To může být užitečné třeba tehdy, když potřebujeme v grafu zvýraznit nějaký důležitý bod.

Každá křivka může být pojmenována pomocí elementu `name`. Jeho atributy `x` a `y` určují, kde bude označení křivky umístěno. Nejsou-li zadány, je název umístěn vpravo od posledního bodu křivky. Název lze také vysázet v legendě, to určuje hodnota atributu `visibility`.

# Kapitola 2

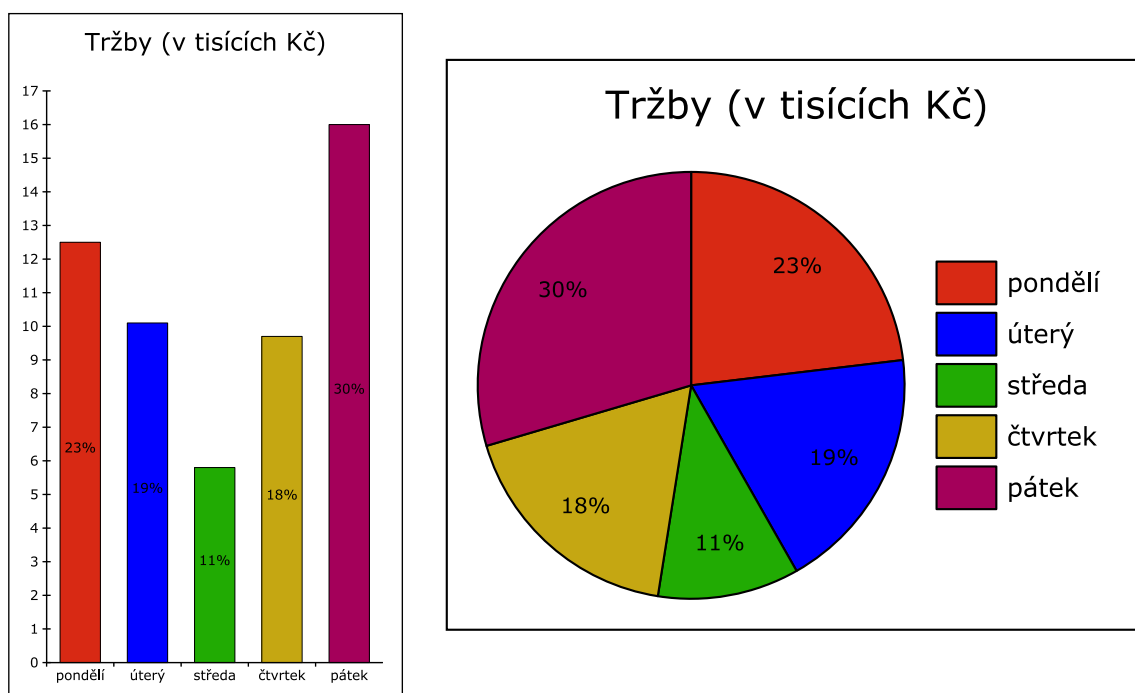
## Použití grafů generovaných z XML

### 2.1. Ukázka grafů typu OSGR

XML soubor, ze kterého lze generovat graf typu OSGR, může vypadat následovně.

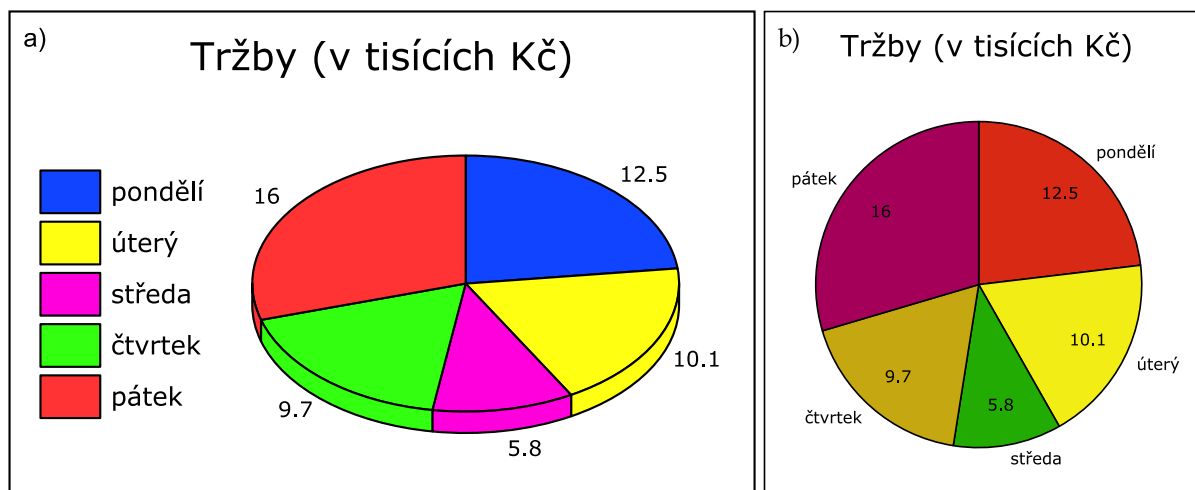
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE osgr SYSTEM "osgr.dtd">
<osgr colorScheme="warm" labelIn="percent">
  <title>Tržby (v tisících Kč)</title>
  <names>
    <name>pondělí</name>
    <name>úterý</name>
    <name>středa</name>
    <name>čtvrtek</name>
    <name>pátek</name>
  </names>
  <values>
    <value>12.5</value>
    <value color="blue">10.1</value>
    <value>5.8</value>
    <value>9.7</value>
    <value>16</value>
  </values>
</osgr>
```

Atributem `colorScheme` jsme nastavili sadu teplých barev ("warm"), která by úterní hodnotu zobrazila žlutě. Atributem `color` u elementu `value` jsme ji ovšem změnili na modrou. Výsledný graf je na obrázku 2.1 – „Normální a výsečový OSGR graf“ vlevo. Vpravo jsou stejná data zobrazena pomocí výsečového grafu.



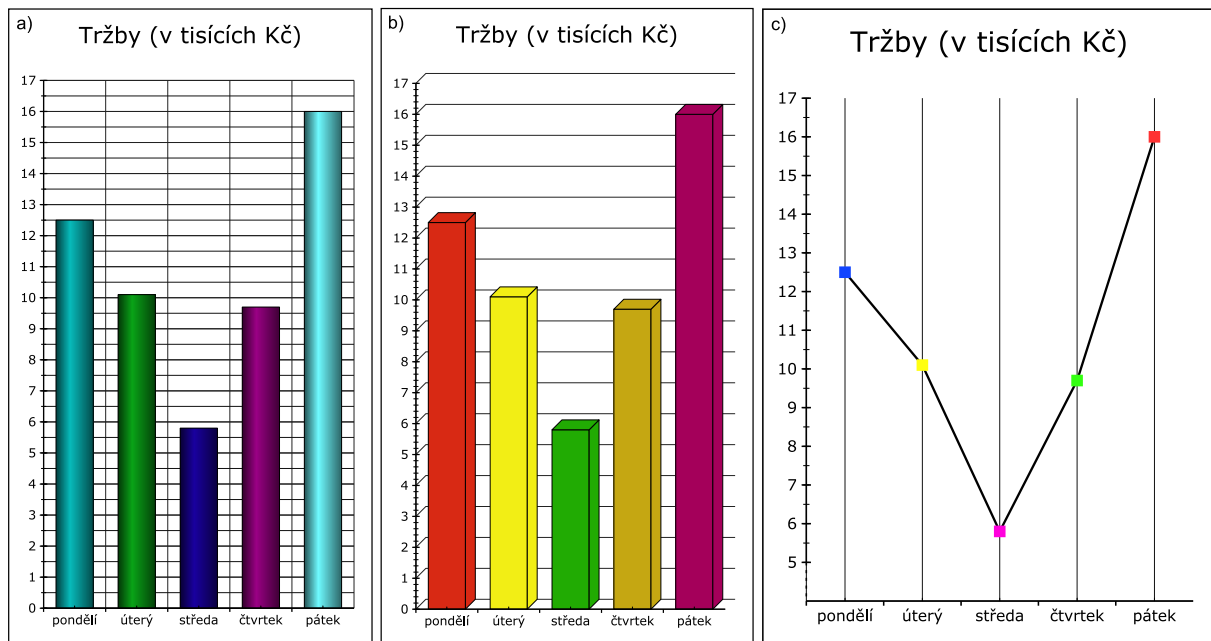
**Obrázek 2.1. Normální a výsečový (navíc: `graphType = "pie"` `legend = "right"`) OSGR graf.**

Na dalších obrázcích můžeme vidět různé jiné podoby OSGR grafů. Struktura vstupního XML dokumentu je podobná, proto jej už neuvádíme. Podstatné jsou atributy elementu `osgr`, ty uvádíme v popisu obrázků.



**Obrázek 2.2. Ukázka výsečových OSGR grafů:**

- a) `graphType = "pie"` `effect = "3D"` `legend = "left"` `labelOut = "value"`,  
b) `graphType = "pie"` `colorScheme = "warm"` `labelOut = "name"` `labelIn = "value"`



Obrázek 2.3. Různé podoby normálních OSGR grafů:

- a) colType = "cylinder" colorScheme = "cold" xAxisDivision = "both"  
yAxisDivision = "2" yGrid = "minor" xGrid = "major",  
b) effect = "3D" colorScheme = "warm" xAxisDivision = "both"  
yAxisDivision = "5" yGrid = "major",  
c) colType = "none" lineType = "solid" pointType = "squareF"  
xAxisDivision = "both" xGrid = "minor" yAxisType = "shifted" yAxisDivision = "2"

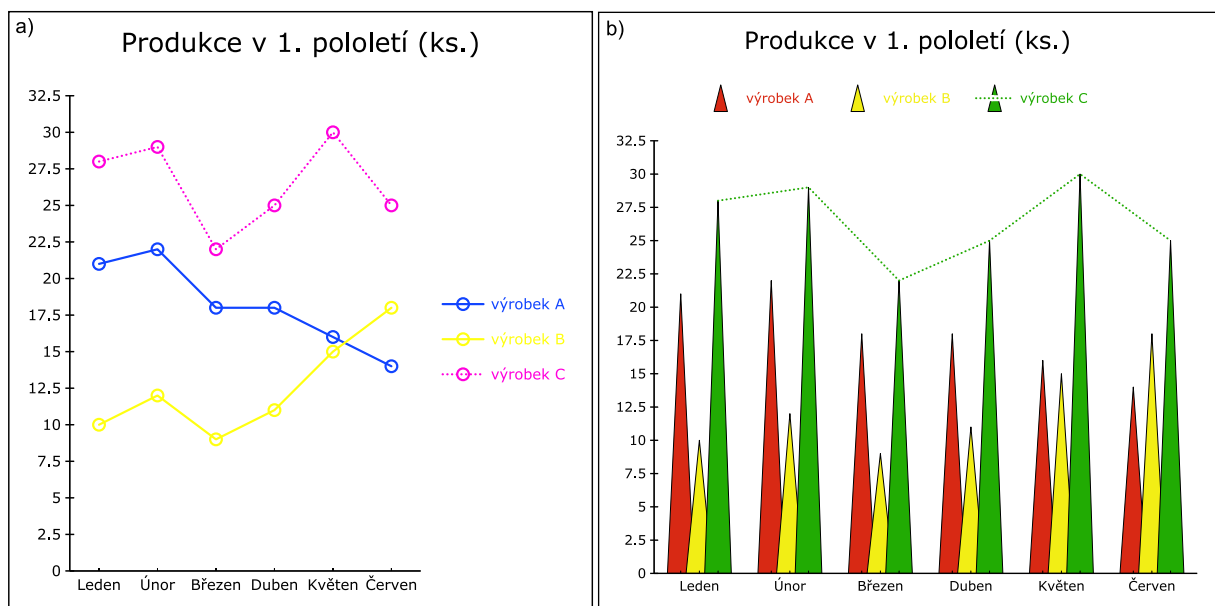
## 2.2. Příklady použití MSGR grafů

Grafy typu MSGR zobrazují více datových řad. Zdrojový XML soubor může vypadat například takto:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE msgr SYSTEM "msgr.dtd">
<msggr pointType="circle">
  <title>Produkce v 1. pololetí (ks.)</title>
  <names>
    <name>Leden</name>
    <name>Únor</name>
    <name>Březen</name>
    <name>Duben</name>
    <name>Květen</name>
    <name>Červen</name>
  </names>
  <values>
    <title>výrobek A</title>
    <value>21</value>
```

```
<value>22</value>
<value>18</value>
<value>18</value>
<value>16</value>
<value>14</value>
</values>
<values>
  <title>výrobek B</title>
  <value>10</value>
  <value>12</value>
  <value>9</value>
  <value>11</value>
  <value>15</value>
  <value>18</value>
</values>
<values lineType="dot">
  <title>výrobek C</title>
  <value>28</value>
  <value>29</value>
  <value>22</value>
  <value>25</value>
  <value>30</value>
  <value>25</value>
</values>
</msgr>
```

Nejsou-li u MSGR grafů nastaveny žádné atributy, jsou všechny řady zobrazeny jako plné lomené čáry spojující jednotlivé hodnoty. V našem příkladu je navíc změněn typ čáry pro výrobek C na tečkovanou a jsou přidány značky v hodnotách datových řad. Výsledný graf je v levé části obrázku 2.4 – „Spojnicový a sloupcový MSGR graf“. Vpravo jsou zobrazena stejná data s jiným nastavením atributů elementu `msgr`. Nastavením `colType = "triangle"` jsme dosáhli vykreslení sloupců tvaru pyramidy. Zrušíme ještě implicitní vykreslování čar. Zůstane pouze tečkovaná spojnice u třetí řady nastavená lokálně. Dále musíme určit posunutí sloupců atributem `shift` tak, aby se sloupečky úplně nepřekrývaly. Protože jsme nastavili hodnotu atributu `shift` menší než 1, je překrytí sloupců jenom částečné. Pokud bychom chtěli sloupce úplně oddělit, nastavili bychom například hodnotu 1,2. Nakonec je ještě použita jiná paleta barev a legenda vykreslena nad grafem.

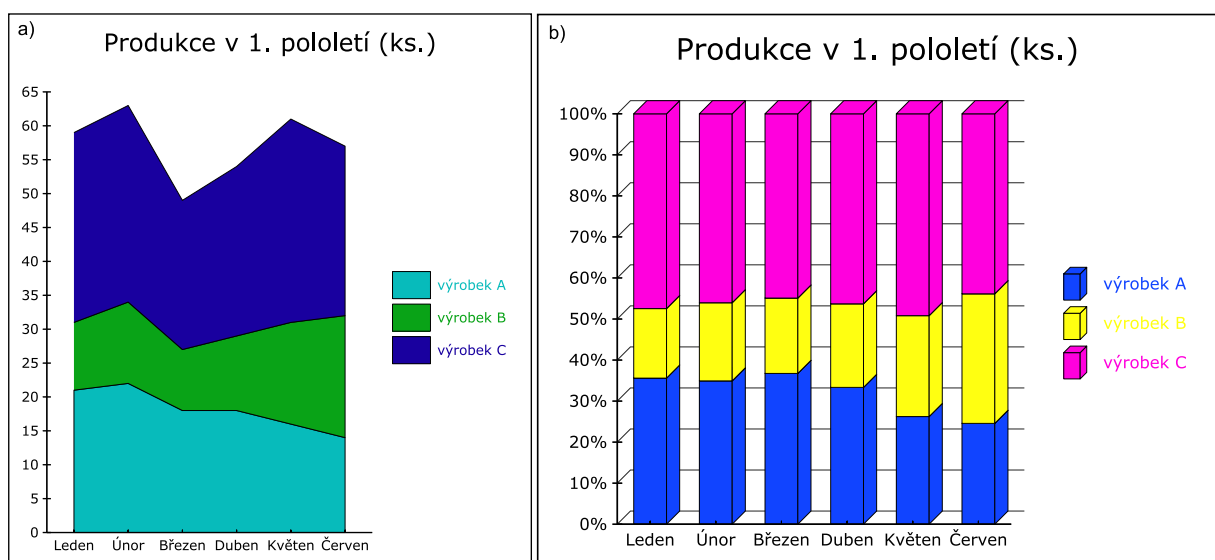


Obrázek 2.4. Spojnicový a sloupcový MSGR graf

a) pointType="circle",

b) lineType = "none" colType = "pyramid" shift = "0.7" colorScheme = "warm"  
legend = "top"

Ukázková data zaznamenávají, kolik výrobků jednotlivých druhů bylo v prvním pololetí vyprodukováno. Pokud by nás zajímalo i srovnání celkové produkce, případně vývoj procentního zastoupení jednotlivých výrobků, použijeme skládaný nebo procentní skládaný graf. Ty nastavujeme atributem `stacked`. Viz obrázek 2.5 – „Skládaný a procentní skládaný graf“.

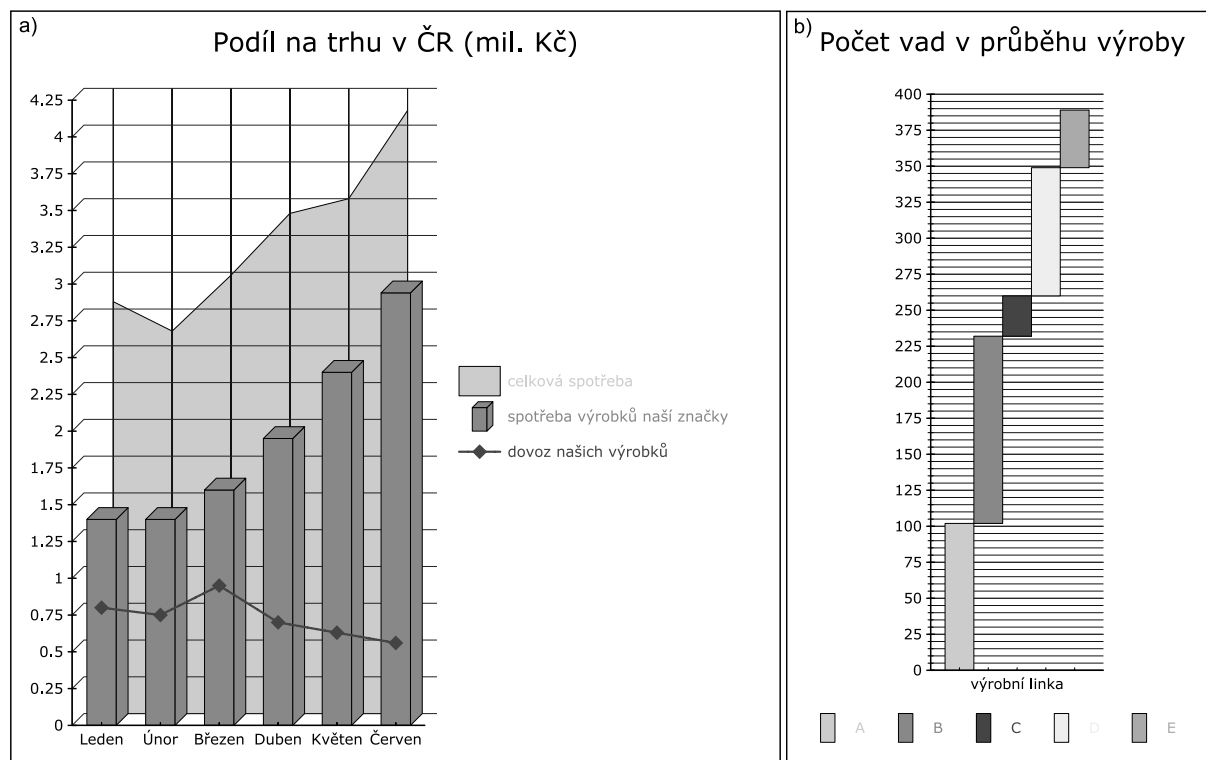


Obrázek 2.5. Skládaný a procentní skládaný graf

a) stacked = "sum" lineType = "none" fillArea = "yes" colorScheme = "cold",

b) stacked = "percentage" lineType = "none" colType = "block" effect = "3D"  
yGrid = "major"

Pomocí MSGR grafů lze zobrazit některé speciální grafy. V levé části obrázku 2.6 – „Kombinace různých atributů jednotlivých řad a Paretův diagram“ vidíme kombinaci různých nastavení pro jednotlivé řady. Vpravo je takzvaný Paretův graf použitý pro zobrazení počtu vad během výroby, který je vytvořen tak, že každá řada má jen jednu hodnotu.



**Obrázek 2.6. Kombinace různých atributů jednotlivých řad a Paretův diagram**

**a) lineType = "none" effect = "3D" xGrid = "minor" yGrid = "major"**

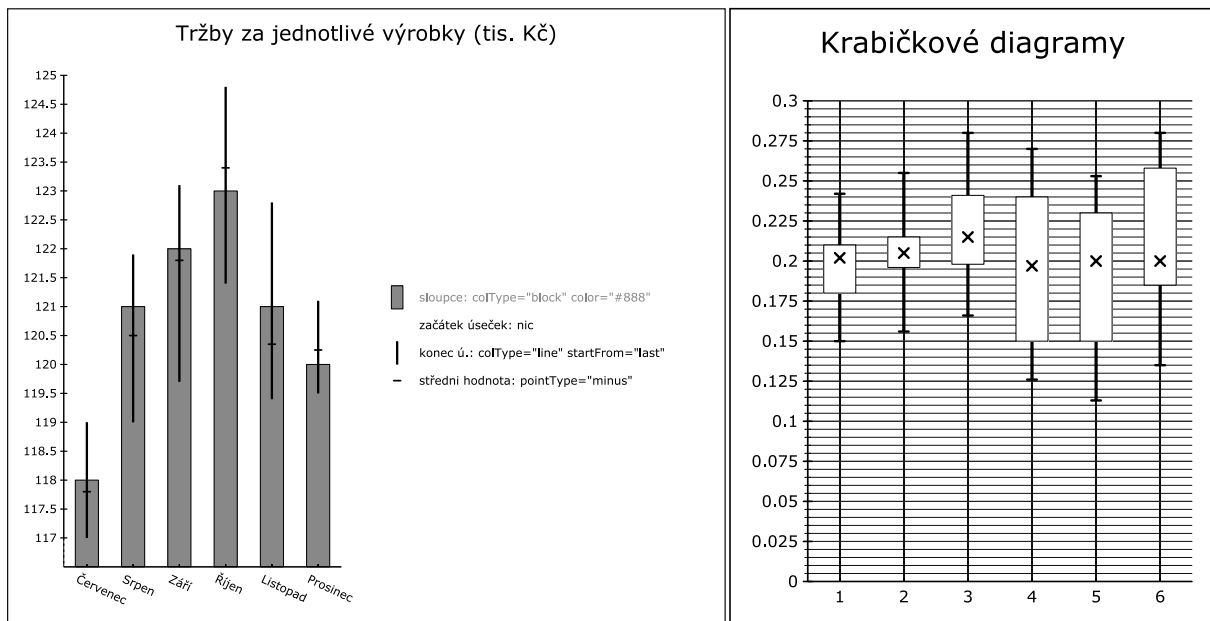
**colorScheme = "grey",**

**b) stacked = "sum" shift = "1" lineType = "none" colType = "block"**

**yGrid = "minor" yAxisDivision = "5" colorScheme = "grey" legend = "bottom"**

Pomocí speciálního typu sloupce "line" můžeme zobrazit například chybové úsečky nebo krabicové diagramy a další typy grafů používané v statistice, viz obrázek 2.7 – „Speciální grafy pomocí MSGR“. U levého grafu si všimněme, že se automaticky zvolil posunutý typ y-ové osy a příliš dlouhé názvy kategorií se píší šikmo.





Obrázek 2.7. Speciální grafy pomocí MSGR

## 2.3. Tvorba grafů typu XYGR

Pro znázornění výsledků měření, grafů funkcí nebo různých diagramů jsou vhodné grafy typu XYGR. Následující jednoduchý XSLT styl vygeneruje graf zadané funkce sinus. Ten je na obrázku 2.8 – „Graf funkce sinus bez vyhlazení“. Na dalším obrázku 2.9 – „Graf funkce sinus s vyhlazením“ vidíme, jak se graf změní, pokud použijeme vyhlazení (u elementu `curve` nastavíme atribut `smooth = "yes"`).

```
<?xml version="1.0" encoding="windows-1250"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:math="http://exslt.org/math"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  extension-element-prefixes="math"
  exclude-result-prefixes="math xs"
  version="2.0">
<xsl:include href="xygr2svg.xsl"/>
<xsl:output method="xml" encoding="utf-8"/>

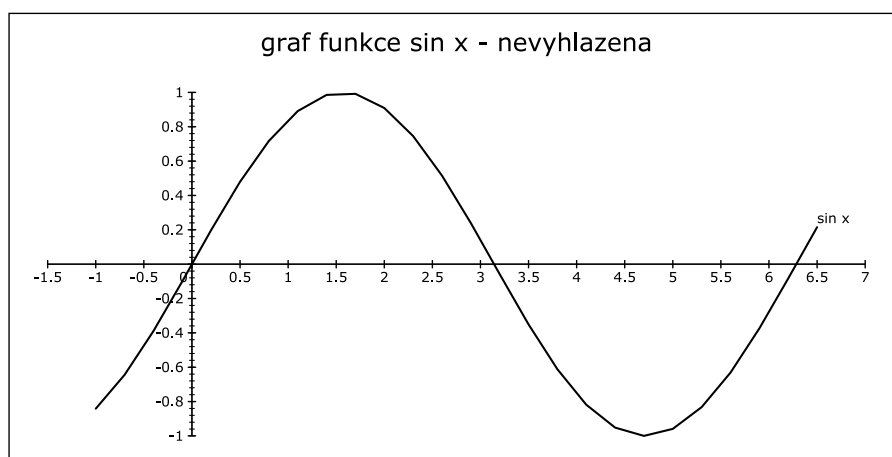
<xsl:param name="xMin" select="-1"/>    <!--minimalni x-->
<xsl:param name="xMax" select="6.5"/>   <!--maximalni x-->
<xsl:param name="xStep" select="0.3"/>  <!--krok-->

<xsl:template match="/">
  <xsl:variable name="gr">
    <xygr yAxisDivision="5">
      <title>graf funkce sin x - nevyhlazena</title>
      <curve>
```

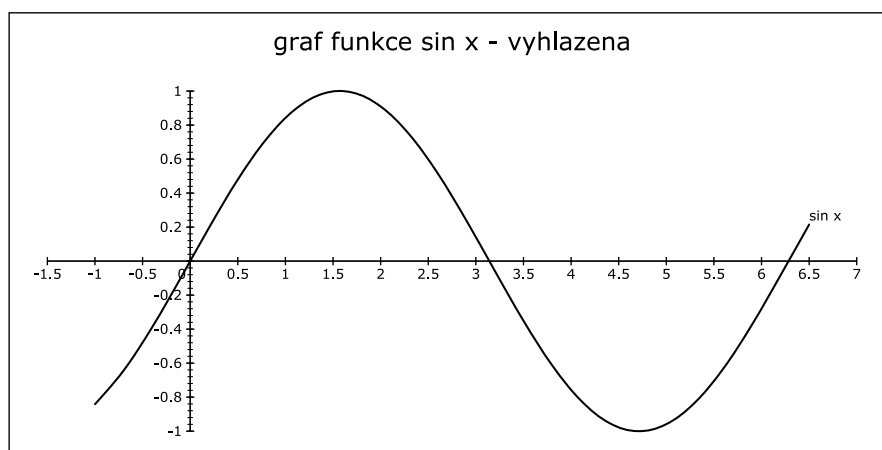
```

<name>sin x</name>
<xsl:for-each select="0 to (floor(($xMax -$xMin) div $xStep)
      cast as xs:integer)">
  <xsl:variable name="x" select="$xMin + (.)*$xStep"/> <!--x-->
  <point x="{ $x}" y="{math:sin($x)}"/> <!--funkcni predpis-->
</xsl:for-each>
</curve>
</xygr>
</xsl:variable>
<xsl:call-template name="xygr2svg">
  <xsl:with-param name="graph" select="$gr/xygr"/>
</xsl:call-template>
</xsl:template>
</xsl:stylesheet>

```

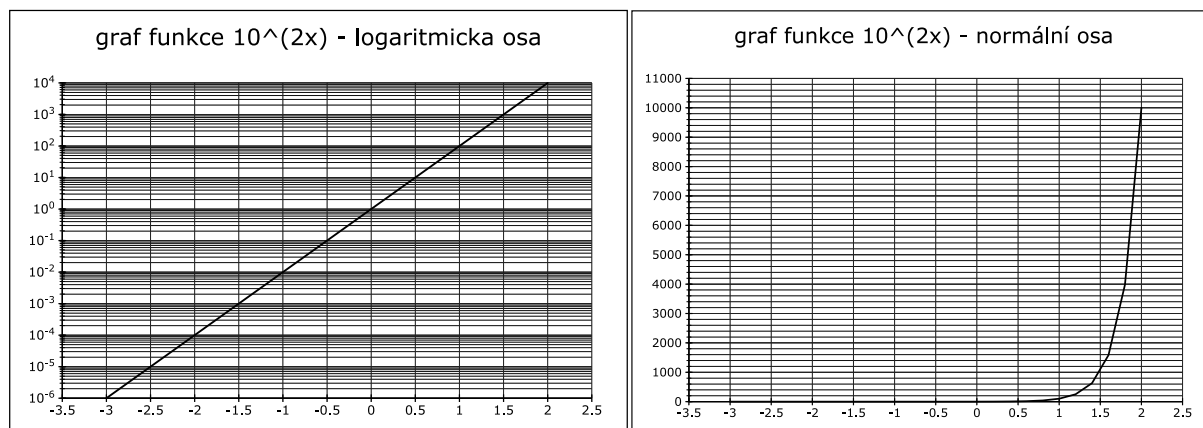


Obrázek 2.8. Graf funkce sinus bez vyhlazení



Obrázek 2.9. Graf funkce sinus s vyhlazením

K vytvoření grafů na obrázku 2.10 – „Graf funkce  $10^{2x}$  s a bez použití logaritmické osy“ jsme použili podobný skript. Můžeme na nich srovnat vykreslené grafy funkce  $10^{2x}$  s použitím logaritmické osy (`yAxisType = "log"`) nebo bez ní. Další parametry jsou u obou grafů nastaveny stejně: `yGrid = "minor"` `yAxisDivision = "5"` `axesPos = "left-bottom"` `xGrid = "major"`.



**Obrázek 2.10. Graf funkce  $10^{2x}$  s a bez použití logaritmické osy**

Grafy typu XYGR můžeme rovněž použít pro tvorbu různých schématických obrázků. Příkladem může být obrázek 2.11 – „Náčrtek ekonomických vztahů“, který byl vygenerován z následujícího XML souboru.

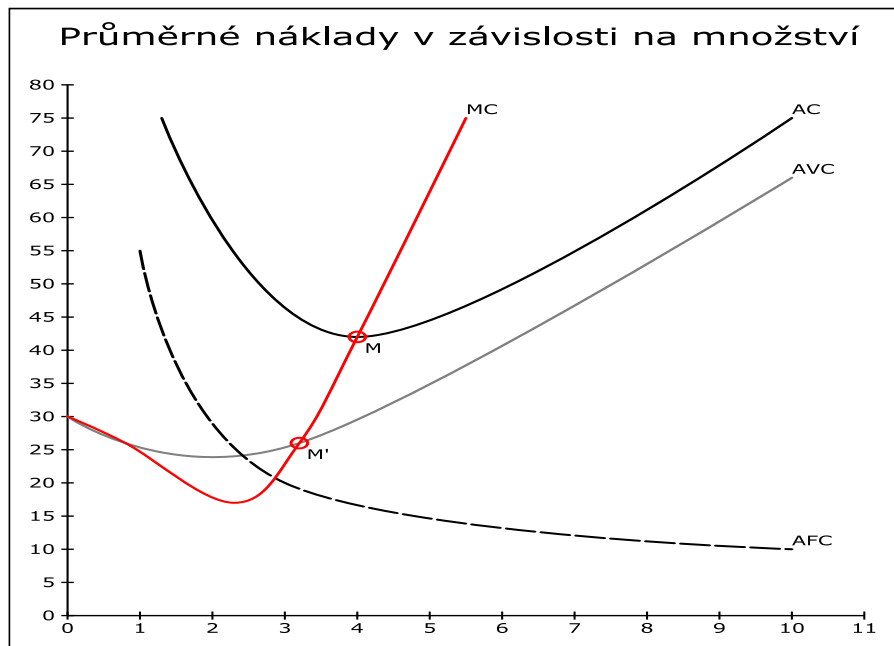
```
<?xml version="1.0" encoding="utf-8"?>
<xygr lineType="solid">
<title>Průměrné náklady v závislosti na množství</title>
<curve smooth="yes">
  <name>AC</name>
  <point x="1.3" y="75"/> <point x="4" y="42"/> <point x="10" y="75"/>
</curve>
<curve smooth="yes" color="grey">
  <name>AVC</name>
  <point x="0" y="30"/> <point x="3.2" y="26"/> <point x="10" y="66"/>
</curve>
<curve smooth="yes" lineType="longDash">
  <name>AFC</name>
  <point x="1" y="55"/> <point x="3" y="20"/> <point x="10" y="10"/>
</curve>
<curve smooth="yes" color="red">
  <name>MC</name>
  <point x="0" y="30"/> <point x="0.8" y="26"/> <point x="2.3" y="17"/>
  <point x="3.2" y="26"/> <point x="4" y="42"/> <point x="5.5" y="75"/>
</curve>
<curve color="red" pointType="circle">
  <name x="4.1" y="39" >M</name>
  <point x="4" y="42"/>

```

```

</curve>
<curve color="red" pointType="circle">
  <name x="3.3" y="23">M'</name>
  <point x="3.2" y="26"/>
</curve>
</xygr>

```



Obrázek 2.11. Náčrtek ekonomických vztahů

## 2.4. Automatické generování grafů z HTML tabulek

V této podkapitole uvádíme příklad, jak lze použít vytvořené styly pro tvorbu grafů z XHTML tabulky. Slouží k tomu následující skript.

```

<?xml version="1.0" encoding="windows-1250"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xh="http://www.w3.org/1999/xhtml" xmlns:g="x-vojt/gr2svg"
  exclude-result-prefixes="g xh" version="2.0">
<xsl:include href="msgr2svg.xsl"/>
<xsl:output method="xml" encoding="utf-8"/>

  <!--vyber tabulky a umisteni dat v tabulce-->
<xsl:param name="tabNum" select="1"/>
<xsl:param name="dataCols" select="2 to 12"/>
<xsl:param name="dataRows" select="3 to 6"/>
<xsl:param name="titlesCol" select="1"/>
<xsl:param name="namesRow" select="2"/>
<xsl:param name="grTitleRow" select="1"/>

```

```

<xsl:param name="grTitleCol" select="1"/>

<xsl:template match="/">
  <!xsl:variable name="graph">
    <xsl:apply-templates select="(//xh:table) [$tabNum]"/>
  </xsl:variable>
  <xsl:call-template name="msgr2svg">
    <xsl:with-param name="graph" select="$graph/msgr"/>
  </xsl:call-template>
</xsl:template>

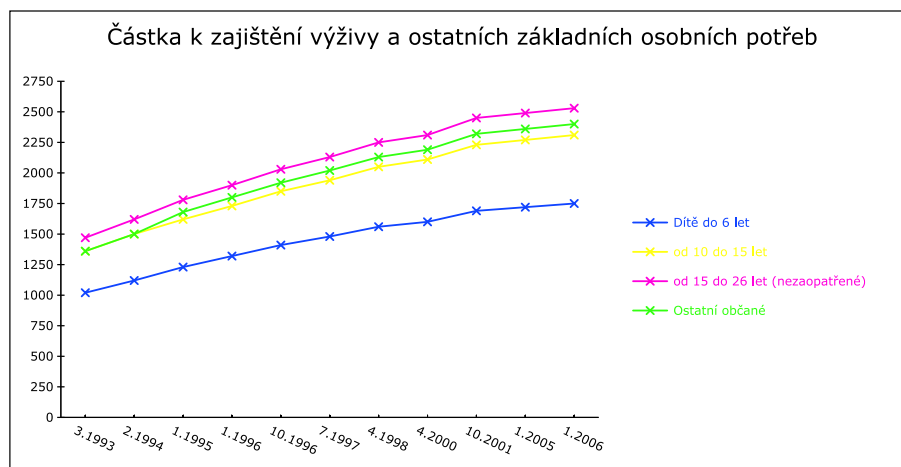
<xsl:template match="xh:table">
  <xsl:variable name="rows" select="(xh:tr|xh:tbody/xh:tr|
    xh:thead/xh:tr|xh:tfoot/xh:tr)"/>
  <msgr lineType="none" colType="block" shift="1.2" legend="botom">
    <title> <xsl:value-of select="
      (g:row2seq($rows[$grTitleRow])) [$grTitleCol]"/>
    </title>
    <names>
      <xsl:for-each select="
        (g:row2seq($rows[$namesRow])) [position() = $dataCols]">
        <name> <xsl:value-of select="."/> </name>
      </xsl:for-each>
    </names>
    <xsl:for-each select="$rows[position() = $dataRows]">
      <values>
        <title> <xsl:value-of select="g:row2seq(.)[$titlesCol]"/> </title>
        <xsl:for-each select="g:row2seq(.)[position() = $dataCols]">
          <value> <xsl:value-of select="translate(., '+', '.')"/> </value>
        </xsl:for-each>
      </values>
    </xsl:for-each>
  </msgr>
</xsl:template>

<xsl:function name="g:row2seq">
  <xsl:param name="row"/>
  <xsl:sequence select="
    for $a in $row/(xh:td|xh:th) return
      if ($a/@colspan) then (
        (for $b in (1 to $a/@colspan) return $a)
      ) else $a"/>
</xsl:function>

```

Pro ilustraci použijeme data ze stránky [http://www.finance.cz/home/hospodarstvi/prace/zivot\\_min/](http://www.finance.cz/home/hospodarstvi/prace/zivot_min/). Nejprve je nutné zdrojový HTML soubor převést do XHTML, aby jej šlo zpracovat

pomocí XSLT procesoru (viz například <http://www.w3.org/Status.html#TIDY>). Dále stačí na převedená data spustit předešlý skript s vhodnými parametry. Parametry mohou být nastaveny při volání stylu, jinak se použijí implicitní hodnoty. Ty odpovídají naší tabulce. Výsledek je na obrázku 2.12 – „Graf automaticky vytvořený z XHTML“.



**Obrázek 2.12. Graf automaticky vytvořený z XHTML**

# Kapitola 3

## Jak jsou styly naprogramovány

Cílem této kapitoly je alespoň v hrubých rysech popsat fungování stylů a usnadnit tak jejich pochopení, úpravu a případně jejich další vývoj.

### 3.1. Instalace a základní použití stylů

Styly a všechny potřebné soubory jsou ke stažení na serveru <http://code.google.com/p/graph2svg/>. Zde je možno stáhnout balík, ve kterém jsou adresáře s jednotlivými styly a ukázkami jejich použití a dále dokumentace, která obsahuje text této práce.

K používání stylů potřebujeme procesor podporující XSLT 2.0. Styly byly vyvíjeny a lazeny v procesoru Saxon 8.8, který je patrně nejsnáze dostupný XSLT procesor s podporou XSLT 2.0 a XPath 2.0. Saxon je ve své základní verzi (Saxon-B) bezplatně použitelný open-source projekt firmy Saxonica ([www.saxonica.com](http://www.saxonica.com)). Běží na platformách Java a .NET.

Všechny tři styly jsou naprogramovány tak, aby je bylo možné volat v jiných stylech. Obsahují jednu hlavní pojmenovanou šablonu, v jejímž parametru je nutno předat data příslušného grafu. Například pro zpracování grafu typu MSGR je hlavní šablona spolu s dalšími pomocnými šablonami a funkcemi v souboru `msgr2svg.xsl`. Ve stylu, který uživatel používá k zpracování vlastních dat (například styl transformující dokumentaci v DocBooku, styl generující HTML stránky a podobně), které obsahují graf, stačí vložit soubor `msgr2svg.xsl` a zavolat šablonu pojmenovanou `msgr2svg`. To již bylo ukázáno v sekci 2.4 – „Automatické generování grafů z HTML tabulek“. Pro zpracování samostatného souboru lze použít jednoduchý styl `use_msgr2svg.xsl`. Používáme-li Saxon na .NET platformě a je-li v souboru `graph.xml` uložen graf typu MSGR, provedeme převod příkazem:

```
transform -o picture.svg graph.xml use_msgr2svg.xsl
```

Výsledný obrázek je pak v souboru `picture.svg`, který můžeme zobrazit v internetovém nebo jiném prohlížeči s podporou SVG nebo jej dále zpracovat a použít.

### 3.2. Celková architektura stylů

Jak už bylo uvedeno v úvodu, rozhodli jsme se naprogramovat styly v XSLT 2.0. Programové konstrukce, které tento jazyk nabízí oproti XSLT 1.0 jsou velmi užitečné a v mnohém usnadnily tvorbu stylů. Jedná se zejména o:

- možnost definovat vlastní funkce
- přímé použití nových funkcí, které v XSLT 1.0 nebyly definovány
- příkaz **if then else** definovaný v XPath 2.0
- práci se sekvencemi v XPath 2.0, zejména použití cyklu **for return**

První dvě možnosti byly v podstatě snadno použitelné i v XSLT 1.0 díky různým dobrovolným rozšířením XSLT procesorů, dnes už jsou ale implementovány přímo ve standardu XSLT 2.0. Ovšem přesto používáme i rozšiřující matematické funkce (jmenný prostor <http://exslt.org/math>), jelikož potřebujeme pracovat s goniometrickými funkcemi ( $\sin$  a  $\cos$ ) a pro automatický výpočet mezi os s exponenciální funkcí ( $\text{power}$ ) a logaritmem ( $\log$ ).

Možnost psaní podmínek přímo v XPath výrazech umožňuje v mnohých případech nahradit podmíněné zpracování pomocí `xsl:if` nebo `xsl:choose`. Pomocí příkazu **for return** zase častokrát snadno obejdeme cyklické zpracovávání v `xsl:for-each` nebo ještě nepřehlednější implementaci cyklu pomocí rekurentního volání pojmenovaných šablon.

Volání stylu pomocí pojmenované šablony v sobě nese určitý problém. Ve stylu totiž nemůžeme používat globální proměnné, jejichž hodnota by závisela na zpracováváných datech. A těch, jak uvidíme později, není málo. Všechny proměnné jsou tedy vypočítávány uvnitř hlavní šablony a při případném volání jiných pojmenovaných šablon musí být předány jako parametr. Při předávání většího počtu parametrů se ovšem zbytečně prodlužuje a znepráhledňuje kód. Proto je většina výpočtů provedena uvnitř hlavní šablony.

Uvnitř hlavní šablony jsou nejprve definovány konstanty, pak je proveden výpočet „globálních“ proměnných a následuje generování jednotlivých grafických prvků SVG dokumentu: kořenový element, mřížky os, výplně ploch, sloupce, osy souřadnic, spojnice bodů, značky bodů a legenda. Při generování legendy jsou vykreslovány pouze zadané názvy datových řad. Piktogramy, které znázorňují způsob vykreslení datové řady, jsou vykreslovány spolu s datovou řadou. Můžeme tak zajistit, že jsou společné vlastnosti piktogramů a hodnot datové řady (například barva) označeny jednotně (v jedné skupině) a je možno je při případné editaci společně změnit.

Při generování výsledného SVG dokumentu bývá často několik možností, jak příslušného vzhledu dosáhnout. Naší snahou je v takových případech zvolit optimální řešení z hlediska délky kódu, snadnosti případné pozdější editace (ať už pomocí nějakého SVG editoru nebo ručně) a jednoduchosti SVG kódů. Touto jednoduchostí máme na mysli snahu zamezit zbytečnému používání složitějších elementů, které by případně nemusely být implementovány v programech zobrazujících SVG.

Například pro vygenerování červeného obdélníku můžeme použít kterýkoliv z následujících příkazů:

```
<svg:rect x="20" y="20" width="120" height="60" fill="red" stroke="black"/>
<svg:path d="M20,20 h120 v60 h-120 z" fill="red" stroke="#000"/>
```

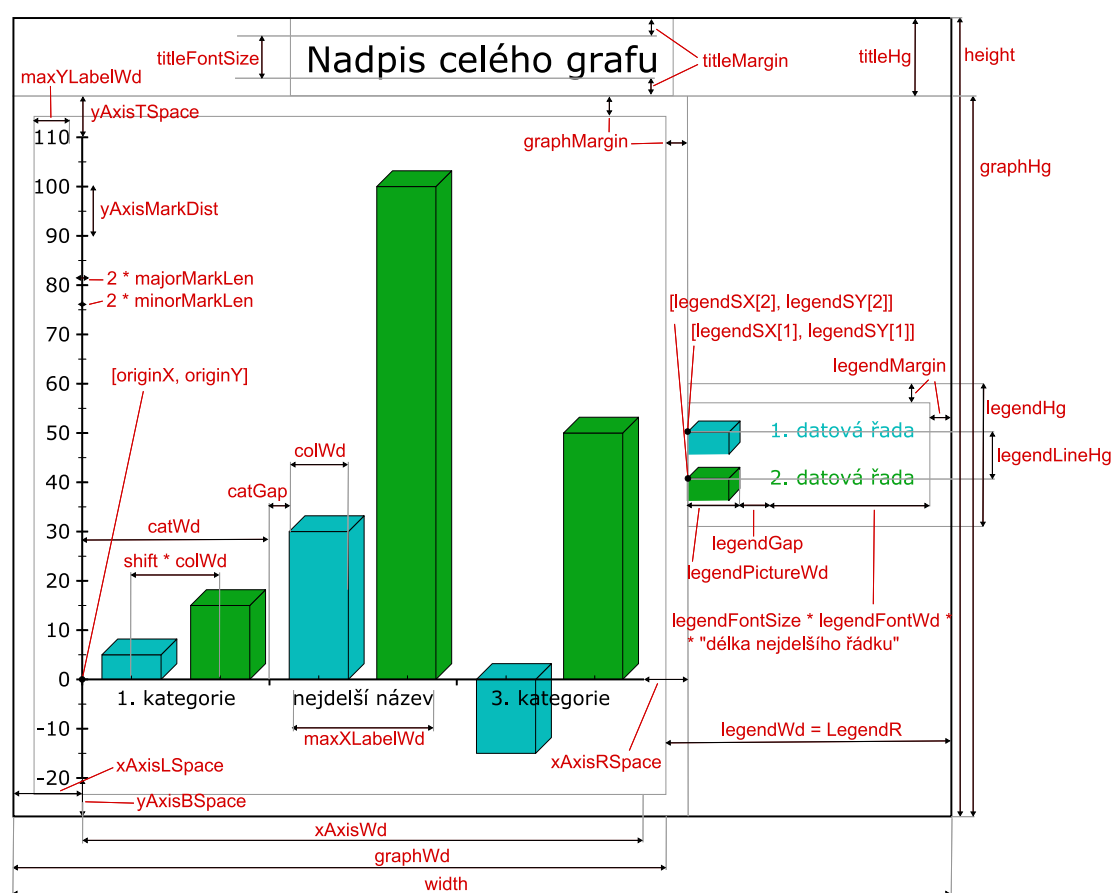
Podstatnějším problémem může být volba, jestli uvádět vlastnosti jednotlivých grafických elementů přímo u nich nebo je sdružit do skupiny (`svg:g`) a pomocí atributů skupiny nastavit parametry najednou. Zde používáme raději druhou možnost, protože kromě úspory místa lze pak snáze editovat výsledné SVG. Sdružujeme samozřejmě pouze prvky logicky související,



například všechny body jedné datové řady. Standard SVG umožňuje každý grafický prvek vykreslit od bodu [0, 0] (ušetří se tak některé atributy, které jsou implicitně nastaveny na 0) a pak jej pomocí atributu `transform` elementu `svg:g` přesunout na požadované místo. Výsledný kód ale nebývá kratší ani přehlednější. Tento princip je použit pouze u vykreslování sloupců, které jsou vystínované pomocí barevného přechodu (`svg:linearGradient`).

### 3.3. Význam důležitých proměnných

Pro pochopení funkce stylu a k jeho případným úpravám je užitečné vědět, co která proměnná znamená. Přestože bylo naší snahou pojmenovávat proměnné výstižně (a to hlavně ty, které se používají na více místech), nemusí být vždy jasné, co daná proměnná uchovává. Při pojmenovávání proměnných používáme následující zkratky: T = Top, B = Bottom, L = Left, R = Right, Wd = Width, Hg = Height, Cat = Category. Rychlou orientaci (snad) usnadní obrázek 3.1 – „Význam některých proměnných“. Jsou zde znázorněny některé důležité proměnné. Pochopitelně takto můžeme zobrazit pouze ty proměnné, které označují nějakou délku. Je-li v obrázku některá dvojice proměnných uzavřena v hranatých závorkách, označuje souřadnice nějakého bodu vzhledem k počátku (v SVG to je levý horní roh). Na takový bod vede ukazatel a je označen černým puntíkem.



Obrázek 3.1. Význam některých proměnných

V obrázku je vykreslen typický graf typu MSGR doplněný o popis důležitých vzdáleností. U ostatních stylů jsou proměnné pojmenovány analogicky. Ve stylu XYGR nejsou na x-ovou osu vynášeny kategorie řad, ale je měřitelná stejně jako y-ová osa na našem obrázku. Ve stylu OSGR jsou grafy typu "norm" pouze o něco jednodušší, u grafů typu "pie" zůstávají stejné pouze proměnné pro základní rozvržení obrázku, ale proměnné uvnitř grafu jsou jiné. Základ tvoří souřadnice středu grafu a jeho poloměr. Proměnných ale není tolik a není proto problém se v nich vyznat.

Poměrně složité je zpracování měřitelných datových hodnot. Je nutno určit jejich nejmenší a největší hodnotu, podle hodnoty parametru `xAxisType` nebo `yAxisType` upravit meze. Dále vypočítat vhodný krok pro zobrazení legendy a podle toho upravit maximální a minimální zobrazovanou hodnotu. Výsledkem tohoto procesu jsou pro každou měřitelnou osu dvě konstanty, které se používají pro přepočtení skutečných souřadnic daného bodu na obrazové souřadnice. Bod o souřadnicích  $[x, y]$  (u grafu typu XYGR) je takto převeden na bod  $[xShift + xKoeff * x, yShift + yKoeff * y]$ .

Při vykreslování bodů, čar a sloupců v MSGR stylu používáme často následující proměnné: `sn` (series number) pro pořadí právě vykreslované datové řady (od 0), `cn` (color number) pro pořadí barvy právě vykreslované datové řady v barevném schématu, `vn` (value number) pro pořadí aktuální hodnoty v rámci datové řady.

### 3.4. Popis některých funkcí a pojmenovaných šablon

V této podkapitole vysvětlujeme chování a použití některých funkcí. Má význam především pro toho, kdo by chtěl hlouběji zkoumat fungování stylů.

předzpracování dat pomocí speciálních šablon

Dříve, než jsou dělány jakékoliv úpravy s daty, proběhne jejich předzpracování. Jestliže má být některá osa zobrazována jako logaritmická (`yAxisType = "log"`) nebo je u MSGR grafu zvolen skládaný graf (atribut `stacked` je roven "sum" nebo "percentage"), jsou hodnoty příslušným způsobem přepočteny. K tomu se používá sada šablon pracujících ve zvláštním režimu, které pro zajímavost uvádíme (pochází opět ze stylu MSGR):

```
<xsl:template match="value" mode="processValues">
  <xsl:param name="graph"/>
  <value>
    <xsl:apply-templates select="@*|*" mode="processValues"/>
    <xsl:variable name="pos" select="count(preceding-sibling::value)+1"/>
    <xsl:value-of select="
      if ($graph/@stacked='sum') then
        sum(../preceding-sibling::values/value[$pos], .) else
      if ($graph/@stacked='percentage') then (
        sum(../preceding-sibling::values/value[$pos], .) div
        sum(../../values/value[$pos]) ) else
      if ($graph/@yAxisType='log') then
        m:Log10(if ((.) != 0) then math:abs(.) else 1) else (.)"/>
  </value>
</xsl:template>
```

```
<xsl:template match="values" mode="processValues">
  <xsl:param name="graph"/>
  <values>
    <xsl:apply-templates select="@*|*|text()" mode="processValues">
      <xsl:with-param name="graph" select="$graph"/>
    </xsl:apply-templates>
  </values>
</xsl:template>
<xsl:template match="*|@" mode="processValues">
  <xsl:copy-of select="."/>
</xsl:template>
```

Výsledek je uložen do proměnné `gra`, která se dále používá místo vstupních dat.

šablona `drawPoint`

Slouží k vykreslení bodu daného typu na zadané místo. Jsou v ní i konstanty ovlivňující velikosti jednotlivých typů bodů.

šablona `drawCol`

Šablona vykreslí sloupec daného tvaru a typu. Sloupec je vykreslen do počátku souřadnic a poté posunut na správné místo. To děláme proto, že v počátku je definován lineární přechod u sloupců typu "cylinder" a "cone", který je společný pro všechny sloupce jedné řady.

funkce `m:LineType`

Tato funkce v závislosti na požadovaném typu čáry vrací řetězec, který musí být vložen do atributu `stroke-dasharray` při vykreslování čar. Určuje se podle něj, jak má být čára přerušována.

funkce `m:Step`

Funkce `m:Step` počítá vhodný krok k popsání os. Nejdříve se vypočte odhad kroku podělením délky osy a požadovaného počtu hlavních značek. Potom se krok upraví tak, aby jeho násobky byly příslušné mocniny desítky.

funkce `m:GMax`

Zaokrouhluje horní maximum dat na celé kroky nahoru. Používá se při určování největší a nejmenší hodnoty os.

# Závěr

Data ekonomického charakteru jsou snadněji analyzovatelná, pokud jsou zobrazena graficky, tedy formou grafu. Přestože nástrojů pro tvorbu grafů existuje hodně, poněkud chybí dostatečně variabilní nástroj, který by převáděl XML data do SVG pomocí XSLT. V práci byla navržena sada tří typů XML souborů, které pokrývají většinu běžně používaných grafů. Přitom jsme vycházeli z typu zobrazovaných dat a snažili jsme se vytvořit návrh, který reprezentuje data doplněná o atributy, které určují, jak má graf vypadat. Jádrem práce jsou ovšem transformační skripty, které vypočítávají výsledný vzhled a vykreslují grafy v SVG formátu.

Snažili jsme se zahrnout co nejširší spektrum grafů a tomu přizpůsobit volbu parametrů. Styly by přesto mohly obsahovat mnoho dalších parametrů, které řídí vzhled grafů, barvy výplní, jednotlivé rozměry a podobně. Jejich používání by však potom bylo dle našeho názoru poměrně složité a nepřehledné. Ostatně je možné přizpůsobit grafy daným potřebám, buď pomocí dodatečné editace výsledného obrázku nebo přímo úpravou skriptů. Stejně tak je možná i jiná koncepce takovéto transformace, například přímý popis prvků grafu.

Jelikož jsou výsledné skripty zveřejněny, může je každý používat pro tvorbu grafů, které potřebuje, nebo je zakomponovat do svého informačního systému. Otevírá se tedy prostor pro další vývoj těchto stylů, jejich obohacení o nové možnosti, případně navržení nových stylů pro jiné speciální typy dat.

# Literatura

- [1] Harris R.: *Information Graphics — A Comprehensive Illustrated Reference*. New York, Oxford University Press 1999, 448 s. ISBN 0-19-513532-6.
- [2] Franců M.: *Office grafy a diagramy — Excel Word PowerPoint*. 1. vyd. Praha, Grada 2005, 116 s. ISBN 80-247-1189-3.
- [3] Kosek J.: *XML pro každého — podrobný průvodce*. Praha, Grada Publishing 2000, 164 s. ISBN 80-7169-860-1. <http://www.kosek.cz/xml/>.
- [4] Bray T., Paoli J., Sperberg-McQueen C. M., Maler E.: *Extensible Markup Language (XML) 1.0 (Second Edition)* W3C 2000, <http://www.w3.org/TR/REC-xml>.
- [5] Kosek J.: *XSLT v příkladech*. <http://www.kosek.cz/xml/xslt/>.
- [6] Clark J.: *XSL Transformation (XSLT) Version 1.0*. W3C 1999, <http://www.w3.org/TR/xslt/>.
- [7] Kay M. ed.: *XSL Transformations (XSLT) Version 2.0, W3C Proposed Recommendation*. W3C 2006, <http://www.w3.org/TR/xslt20/>.
- [8] Nič M.: *XSLT 2.0 Tutorial*. <http://zvon.org/xxl/XSL-Ref/Tutorials/>.
- [9] Berglund A., Boag S., Chamberlin D., Fernández M. F., Kay M., Robie J., Siméon J.: *XML Path Language (XPath) 2.0, W3C Proposed Recommendation*. W3C 2006, <http://www.w3.org/TR/xpath20/>.
- [10] *Saxon 8.8 documentation*. Saxonica 2006, <http://www.saxonica.com>.
- [11] Ferraiolo J., Fujisawa J., Jackson D. a kol.: *Scalable Vector Graphics (SVG) 1.1 Specification, W3C Recommendation*. W3C 2003, <http://www.w3.org/TR/svg11/>.