České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra řídicí techniky

Diplomová práce

2007 **Antonín Karolík**

České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra řídicí techniky

Diplomová práce

Nástroj na kreslení Ganttových diagramů s využitím Metapostu

Vypracoval: Antonín Karolík
Vedoucí diplomové práce: Ing. Michal Kutil
Rok: 2007

Prohlášení

Prohlašuji, že jsem na přípravě diplomové práce pracoval samostatně a použil jsem pouze podklady (literaturu, projekty, software atd.) uvedené v přiloženém seznamu.

V Praze dne _____ _____

podpis

Poděkování

Chtěl bych poděkovat všem lidem, bez nichž by tato práce nemohla vzniknout. Děkuji především panu Ing. Michalu Kutilovi, vedoucímu mé diplomové práce za jeho vedení, připomínky a poznámky k práci. Dále bych rád poděkoval svým rodičům a všem blízkým za jejich morální podporu a vytvoření dobrých studijních podmínek.

Abstrakt

Postscriptový generátor ganttových diagramů (Psgantt) je perlovský skript, který umožňuje generovat ganttovy diagramy.

Vytváření diagramů je realizováno prostřednictvím METAPOSTu, což je programovací jazyk určený pro kreslení obrázků. Psgantt načte data ze vstupního souboru, který je ve formátu XML. K nim podle definic kaskádových stylů (CSS) přidá informace o způsobu jejich zobrazení. Všechny tyto údaje poté použije k vytvoření zdrojového souboru, který lze dále přeložit METAPOSTem. Výsledkem překladu je požadovaný obrázek ve formátu EPS. Název Psgantt je odvozen právě z naznačeného způsobu zpracování. Psgantt navíc umožňuje převedení takto získaného obrázku z formátu EPS do formátu PDF, který je dnes jedním z nejpoužívanějších formátů pro elektronickou sazbu.

Psgantt je navržen s důrazem na svou přenositelnost. Je možné jej používat jak v systémech založených na Unixu, tak v operačních systémech Microsoft Windows. Psgantt je vyvíjen v úzké spolupráci s TORSCHE Scheduling Toolboxem pro Matlab.

Abstract

Postcript Gantt chart generator (Psgantt) is a perl script enable to generate Gantt diagrams.

The diagrams are generated through METAPOST – programming language for pictures drawing. Psgantt loads data from the input file in XML format. Psgantt adds information about way of their appearance according the Cascading Style Sheets definition (CSS). All data will be used for creating the source file, which can be then compiled by METAPOST. The output is a picture in EPS format. The name Psgantt is taken from the process described above. Moreover Psgantt is able to transform this EPS pictures to PDF format, which is the most used format for electronic typesetting.

Psgantt emphasizes a good portability. It can be used in Unix like systems as well as in Microsoft Windows. Psgantt has been developed in close cooperation with TORSCHE Scheduling Toolbox for Matlab.

Katedra řídicí techniky *Školní rok:* 2004/2005

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student: Antonín Karolík

Obor: Technická kybernetika

Název tématu: Nástroj na kreslení Ganttových diagramů s využitím Metapostu

Zásady pro vypracování:

1. Prostudujte charakteristiky a vlastnosti Metapostu jako programovacího jazyka pro popis obrázků.
2. Navrhněte vhodný XML formát souboru popisující zdrojová data pro sestavení Ganttových diagramů.
3. Napište program generující Ganttovy diagramy na základě navrženého XML formátu s využitím Metapostu.

Seznam odborné literatury: Dodá vedoucí práce.

Vedoucí diplomové práce: Ing. Michal Kutil

Termín zadání diplomové práce: zimní semestr 2004/2005 (změna zadání: 11.05.2006)

Termín odevzdání diplomové práce: leden 2007

prof. Ing. Michael Šebek, DrSc.
vedoucí katedry

L.S.

prof. Ing. Zbyněk Škvor, CSc.
děkan

V Praze dne 17.05.2006

Obsah

Seznam obrázků

Seznam tabulek

1 Úvod

Využití rozvrhovacích algoritmů v praxi neustále narůstá. Jejich nejčastějším úkolem je vytvoření určitého rozvrhu tak, aby vyhověl zadaným omezením a požadavkům. Vytvořený rozvrh přiřadí jednotlivým strojům (procesorům) posloupnost úloh, které mají vykonat. Jelikož je dnes většina rozvrhovacích algoritmů zpracovávána počítači, jsou jejich výsledkem data, která mohou být pro běžného člověka nepřehledná a špatně čitelná. Proto vznikla potřeba tato data vizualizovat, a tím je více přiblížit lidskému vnímání. Jednou z možností je použít vhodný program. Předmětem této diplomové práce je popis programu Psgantt, který byl k tomuto účelu vytvořen.

Psgantt je program napsaný v programovacím jazyce Perl, který je určen pro vizualizaci rozvrhů. Obrázkům rozvrhů se podle významného amerického inženýra H. L. Gantta též často říká ganttovy diagramy. Jednotlivé prvky ganttových diagramů a jejich význam budou popsány v kapitole 2.

Vstupem programu Psgantt je textový soubor, který obsahuje vlastní data rozvrhu. Tyto data jsou v něm uložena pomocí konstrukcí jazyka XML (eXtended Markup Language). Vstupní soubor může být výsledkem zpracování rozvrhovací úlohy programem Matlab s nainstalovaným TORCHE Scheduling Toolboxem, nebo může být vytvořen ručně pomocí libovolného textového nebo XML editoru. Popis a náležitosti vstupního XML souboru jsou uvedeny v kapitole 4.

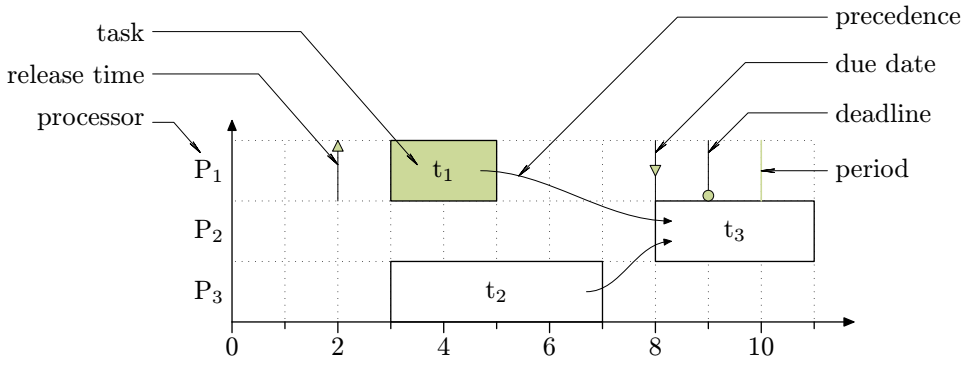
Jelikož data vstupního souboru nemusí obsahovat žádné informace o vzhledu, mohou k nim být tyto přidány z dalšího souboru, takzvané tabulky kaskádových stylů (CSS – Cascading Style Sheets). Informace uvedené v kaskádových stylech udávají, jak má pomocí XML definovaný objekt vypadat. Kromě popisu vzhledu a vlastností jednotlivých objektů, jakými jsou například barva, tloušťka čáry, velikost a další, poskytují kaskádové styly také jednoduchá pravidla pro výběr objektů, na které budou tyto vlastnosti aplikovány. Proto lze jednoduchým způsobem vybrat a zobrazit pouze ty objekty rozvrhu, které jsou pro nás důležité a nastavit pouze jejich vzhled. Tento přístup je výhodný zejména pro rozsáhlé rozvrhy, ve kterých je důležitá především rychlá orientace. Popisem pravidel výběru objektů a jejich vlastnostmi se zabývá kapitola 5.

Výstupem programu Psgantt může být buď obrázek, nebo zdrojový soubor, ze kterého byl tento obrázek vytvořen, nebo obojí. Podporovány jsou dva typy výstupních formátů: EPS (Encapsulated PostScript) a PDF (Portable Document Format). Oba jsou od firmy Adobe Systems Incorporated. Tyto formáty v dnešní době představují standardy pro uložení vektorové grafiky. Lze je ale snadno převést i do jiných vektorových nebo bitmapových formátů. Zdrojový soubor každého obrázku obsahuje instrukce programovacího jazyka METAPOST. Popisem základních prvků tohoto jazyka se zabývá kapitola 8. Příkazy METAPOSTstu mohou být také obsaženy ve vstupním XML souboru, a tak mohou snadno rozšířit obrázek rozvrhu o další grafické prvky.

2 Popis ganttových diagramů

Obrázek 2.1 zobrazuje jednoduchý ganttův diagram. V něm jsou označeny všechny prvky, které se v diagramu mohou objevit.

Ganttův diagram znázorňuje předpis, jak jsou jednotlivé úlohy v čase přiřazeny procesorům. Základními prvky ganttových diagramů jsou úloha (**task**) a stroj (**processor**). Každý task v diagramu je znázorněn boxem, který budeme někdy také nazývat taskbox. Pokud je daná úloha rozvrhovacím algoritmem rozdělena na více částí, může mít i více boxů. Popisek uprostřed boxu reprezentuje identifikátor úlohy nebo její části. Z polohy boxu vyplývá, kdy daná úloha začíná (start time) a kterému procesoru je přiřazena. Šířka boxu znázorňuje délku zpracování úlohy přiřazeným procesorem (processing time) a také čas jejího ukončení (completion time).



Obrázek 2.1: Ukázkový ganttův diagram

Zbývající prvky v diagramu představují omezení, která jsou na daný rozvrh kladena. Jimi mohou být:

Release time – okamžik dostupnosti úlohy, což čas, ve kterém může být úloha nejdříve spuštěna. Úloha tedy nesmí být spuštěna dříve, než je hodnota jejího release time.

Due date – okamžik požadovaného dokončení úlohy, což je čas, ve kterém by měla být úloha dokončena.

Deadline – poslední okamžik dokončení úlohy, je čas, do kterého musí být úloha bezpodmínečně dokončena.

Precedence – omezení, která představují vzájemné návaznosti jednotlivých úloh. Pokud se rozvh zabývá například stavbou domu, je jasné, že stavba střechy nesmí začít dříve, než jsou dokončeny zdi. Návaznosti úloh jsou v grafu reprezentovány šipkami. Směr šipky

udává pořadí úloh. Úloha, ze které šipka vychází musí být zpracována dříve, než úloha do které šipka vede.

Period – perioda úlohy, je hodnota používaná v cyklickém rozvrhování. V něm hodnota periody udává zároveň i hodnotu deadline, která se periodicky opakuje.

Hodnotami na x-ové ose může být čas, ale mohou jimi být například také počty iterací. Jelikož přesné jednotky neznáme, nebudeme je u x-ové osy uvádět. Hodnoty na y-ové ose tvoří procesory. Ty jsou číslovány sestupně. Procesor s nejvyšším pořadovým číslem je umístěn nejblíže k ose x, procesor s nejnižším číslem je od ní umístěn nejdále.

3 Motivační příklad využití programu Psgantt

Demonstrujme možnosti využití programu Psgantt na jednoduchém příkladu zabývajícím se rozvrhováním. V něm vyřešíme problém, jak rozvrhnout výrobu židle pro dva dělníky.

K vlastnímu řešení problému použijeme program Matlab, kde je nainstalován TORSCHE Scheduling Toolbox. TORSCHE Scheduling Toolbox rozšiřuje možnosti Matlabu o algoritmy rozvrhování. Program Psgantt dokáže s tímto toolboxem spolupracovat. TORSCHE Scheduling Toolbox totiž dokáže produkovat XML, které vyhovuje formátu popsaném v kapitole 4.

3.1 Zadání

Našim úkolem je rozvrhnout výrobu židle pro dva truhláře tak, aby trvala, pokud možno, co nejkratší dobu. Židle je složena ze čtyřech nohou, sedátka a opěradla. Předpokládejme, že oba truhláři jsou stejně zruční, to znamená, že výroba kterékoli části židle bude trvat každému z nich stejnou dobu. (V terminologii používané v rozvrhování bychom řekli, že oba procesory jsou identické, což znamená, že kterákoli jim přiřazená úloha bude vykonávána na každém z nich stejnou rychlostí.) Výroba každé nohy židle trvá 6 časových jednotek. Tyto úlohy označme leg_1, leg_2, leg_3, leg_4. Výroba sedátka (seat) zabere 15 časových jednotek. Opěradlo (backrest) lze vyrobit za 25 jednotek, přičemž víme, že jeho výroba může začít nejdříve v čase 20 od začátku výroby židle. (Například z důvodu čekání na materiál). Vyrobené části je ještě nutné poskládat dohromady. Poskládání všech nohou a sedadla, označme jej assembly_1, zabere 15 jednotek. K takto připravené podstavě židle je ještě nutné připevnit opěradlo, což zabere taktéž 15 jednotek. Tuto úlohu označme jako assembly_2. Ze zadání je jasné, že jednotlivá sestavování nemůžou začít dřív, než je dokončena výroba všech potřebných částí, což určuje precedenční závislosti. Příklad je převzatý z [17].

3.2 Řešení

Nyní, když známe zadání, můžeme začít formulovat rozvrhovací problém. K němu použijeme příkazy z TORSCHE Scheduling Toolboxu. Zdrojový kód pro program Mablab by vypadal následovně. Podrobný popis jednotlivých příkazů je možné najít v dokumentaci k TORSCHE Scheduling Toolboxu v [16], zde budou vysvětleny jenom ve stručnosti.

1. Nejprve definujeme jednotlivé úlohy

```
t1 = task(leg1,6);
t2 = task(leg2,6);
```

```
t3 = task(leg3,6);
t4 = task(leg4,6);
t5 = task(seat,6);
t6 = task(backrest,25,20);
t7 = task(assembly1,15);
t8 = task(assembly2,15);
```

2. Zadáme všechna precedenční omezení jednotlivých úloh.

```
prec = [0 0 0 0 0 0 1 0;...
        0 0 0 0 0 0 1 0;...
        0 0 0 0 0 0 1 0;...
        0 0 0 0 0 0 1 0;...
        0 0 0 0 0 0 1 0;...
        0 0 0 0 0 0 0 1;...
        0 0 0 0 0 0 0 1;...
        0 0 0 0 0 0 0 0];
```

3. Úlohy a omezení spojíme do jednoho celku.

```
T = taskset([t1 t2 t3 t4 t5 t6 t7 t8],prec);
```

4. Specifikujeme typ rozvrhovacího problému.

```
p = problem(P|prec|Cmax);
```

5. S danou množinou úloh, jejich omezeními a výše formulovaným problémem zavoláme rozvrhovací algoritmus. Ten určí jednotlivým procesorům (v našem případě truhlářům) pořadí zpracování jednotlivých úloh. K řešení jsme vybrali algoritmus zvaný List Scheduling.

```
S = listsch(T,p,2,SPT)
```

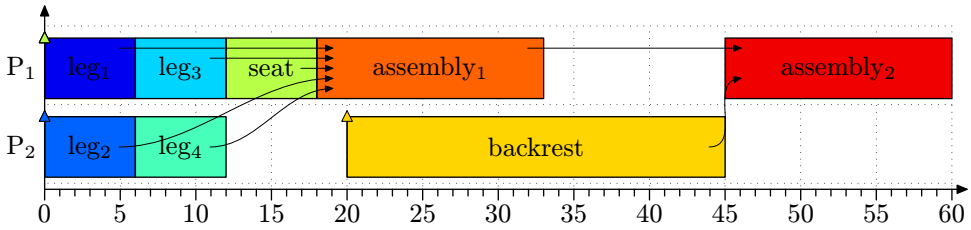
6. Výsledný rozrvh můžeme uložit do souboru XML, který pak bude vstupem programu Psgantt.

```
xmlsave('chair.xml',S);
```

   Soubor `chair.xml` je uložen na přiloženém CD-ROM.

3.3 Vizualizace

Spuštěním programu Psgantt se vstupním souborem `chair.xml` vygenerujeme obrázek diagramu. V tomto obrázku pak můžeme prostřednictvím CSS změnit nastavení vzhledu. Například velikosti jednotek x a y, přidání vertikálních mezer a jiné. Výsledný upravený diagram můžeme vidět na obrázku 3.1.



Obrázek 3.1: Výroba židle

4 Popis vstupního XML souboru

4.1 Úvod do XML

Vstupem programu Psgantt je textový soubor, nebo též dokument, který je ve formátu XML (eXtenzible Markup Language). V následujícím textu budou ve stručnosti popsány základní prvky jazyka XML. Účelem následující kapitoly je pouze nastínit, co XML vlastně je a co nám tento jazyk v souvislosti s Psganttem nabízí. Tento popis bude dále použit k vysvětlení syntaxe a významu jednotlivých částí XML tvořící vstup Psganttu. Úplný popis možností XML je možné nalézt v jeho specifikaci [1]. Je možné použít i její český překlad, který je uveden v [4].

4.1.1 Stručný popis jazyka XML

XML je obecný značkovací jazyk, který byl vyvinut a standardizován konsorciem W3C. Obecný v předchozí větě znamená, že pomocí XML můžeme vytvořit dokument s libovolně pojmenovanými značkami (někdy též nazývanými tagy). Princip XML umožňuje popsat strukturu libovolného dokumentu z hlediska věcného obsahu jeho jednotlivých částí. Vlastní obsah se ale nemusí zabývat jejich vzhledem, ten je definován pomocí kaskádových stylů. Kaskádové styly budou popsány dále v textu, v kapitole 5.

Jednoduše řečeno, jsou data v dokumentech XML reprezentovaná textovými řetězci, které jsou uzavřeny uvnitř značek. Značky jsou rovněž textové řetězce, které svým obsahem popisují data která obklopují. Rozlišujeme dva typy značek – značky počáteční a značky koncové. Počáteční značku tvoří text obklopený znaky `< >`, například `<jmeno>`. Koncová značka je obklopena znaky `<` a `/>`, například `</osoba>`. Konkrétní textový řetězec, spolu s příslušnými značkami, budeme dále označovat názvem *element*. Název elementu je spojen s obsahem jeho počáteční značky. Elementy mohou být do sebe libovolně vnořeny. Jestliže se však počáteční značka některého elementu nachází uvnitř jiného elementu, musí být uvnitř tohoto elementu i jeho značka koncová. Možnou součástí elementů jsou tzv. atributy. *Atribut* je dvojice název–hodnota připojená k počáteční značce příslušného elementu. Název atributu je od hodnoty oddělen rovnítkem a případně mezerami. Hodnoty atributů jsou uzavřeny ve dvojitých uvozovkách. Přesný popis elementů a atributů XML, jejich druhy a náležitosti, jsou plně popsány v [1].

Důležitou vlastností XML, na kterou je nutné dát pozor, je rozlišování malých a velkých písmen. Pokud se například názvy dvou elementů liší, byť jen ve velikosti jediného znaku, pak jsou považovány za různé. Stejné pravidlo platí obecně pro všechny názvy a hodnoty, které se v XML vyskytují.

Obecně je také důležité pořadí v jakém jsou elementy nebo atributy v souboru XML uvedeny. Toto pravidlo nemusí být striktně vyžadováno. Ve vstupním XML tedy mohou existovat části,

kde na pořadí uvedených elementů nebo atributů nezáleží. Pro nás bude pořadí závazné, pokud nebude v textu uvedeno jinak. Už nyní ale uveďme, že pro program Psgantt není důležité pořadí atributů.

Je možné rozlišit dva typy dokumentů XML. Jsou to datově orientované dokumenty a dokumenty orientované na sdělení. Prvně jmenovaný typ nabízí univerzální formát pro uložení složitých datových struktur, který je přenositelný mezi různými programovacími jazyky i počítačovými architekturami. Druhý typ je používán jako formát pro uložení například webových stránek, různých článků, učebnic a dalších jim podobných dokumentů. V této práci se budeme zabývat pouze prvním typem dokumentů.

Pokud dokument XML vyhovuje pravidlům uvedeným ve specifikaci, pak o takovém dokumentu říkáme, že je *platný* nebo také, že je správně formulovaný. Ze základních pravidel správně formulovaných dokumentů XML uveďme:

- každá počáteční značka musí mít odpovídající koncovou značku,
- elementy se nesmějí překrývat,
- musí mít právě jeden kořenový element,
- element nesmí mít dva atributy stejného názvu

a mnohé další (viz specifikace [1]).

Pro demonstraci právě uvedeného popisu si jako příklad uveďme následující text, který je platným dokumentem XML.

```
<osoby>
  <osoba id="o1">Jan Novák</osoba>
</osoby>
```

V něm řetězec `Jan Novák` představuje data nebo též vlastní obsah elementu `osoba`. Řetězce `<osoba>`, respektive `</osoba>` tvoří počáteční, respektive koncovou značku. Dále element `osoba` obsahuje atribut s názvem `id`, který má hodnotu `o1`. Element `osoba` je vnořen do elementu `osoby`, který je jeho rodičem.

Názvy elementů, atributů a jejich hodnoty si při formulaci XML volíme sami. Je tedy na nás, zda použijeme výše uvedený příklad nebo stejné informace raději uložíme takto:

```
<osoby>
  <osoba id="o1" jméno="Jan Novák"/>
</osoby>
```

Všimněme si, že se nyní obsah elementu `osoba` přesunul do hodnoty atributu `jméno`. Také koncová značka elementu chybí. Přesněji je počáteční značka zároveň i značkou koncovou, což je dáno přítomností znaku `/`. O takovém elementu říkáme, že má prázdný obsah nebo jednodušeji, že je prázdný. I tento příklad je platným dokumentem XML.

4.1.2 Validace dokumentů XML

Pro aplikaci, která načítá dokument XML je důležité, aby obsahoval všechna data, která aplikace pro svou činnost potřebuje. Navíc je nezbytně nutné, aby tato data byla ve správném tvaru. Po-

kud aplikace bude například zpracovávat objednávky, bude jistě někde ve vstupním dokumentu XML obsažen údaj o její ceně. Ta by měla představovat nezáporné číslo. Pokud bude namísto čísla uveden text „ahoj“, může aplikace objednávku odmítnout a označit ji za chybnou.

Aby aplikace dokázaly vzdorovat neplatným datům, procházejí dokumenty XML takzvanou validací. Validace umožňuje odhalit nekonzistence dat, které by mohly vadit při jejich dalším zpracování. Také výrazně zjednodušuje kontroly vstupních údajů na úrovni aplikace. Podstatou validace dokumentu XML je jeho porovnání se schématem. Schéma představuje formalizaci kontroly, co může daný XML dokument obsahovat. Jazyků definujících schémata XML dnes existuje několik. Z nejznámějších jmenujme například DTD, W3C XML Schema nebo Relax NG. Program Psgantt používá právě posledně jmenovaný Relax NG.

Relax NG je moderní, v porovnání s ostatními, velice elegantní schémový jazyk pro XML. Specifikace Relax NG byla vyvinuta standardizační organizaci OASIS a jeho standardizace ISO je v současnosti téměř u konce. Schéma v Relax NG představuje vzor, kterému musí vyhovět validní dokument XML. Tento vzor se skládá z dalších vzorů, kterým vyhovují jednotlivé části dokumentu nebo přímo jednotlivé elementy. Nejjednodušší vzory lze pomocí dalších vzorů vzájemně kombinovat. Opakováním rozložení vzorů lze popsat pravidla pro libovolné XML. Bližší popis validačních schémat Relax NG je možné najít v [5].

Pro nás je hlavně důležité, že je dokument XML před vlastním použitím validován. Pokud dokument validací projde, pak si můžeme být jisti, že obsahuje všechna potřebná data ve správném formátu. V opačném případě, můžeme takový dokument odmítnout.

4.2 Popis jednotlivých elementů

V této sekci budou popsány přípustné názvy, pořadí, význam a hodnoty jednotlivých elementů a atributů vstupního XML. Jejich obsah je pro Psgantt klíčový. Určuje, kolik bude mít rozvrh procesorů, kde se budou úlohy (tasky) v obrázku nalézat, jaká bude doba jejich zpracování, jaká budou jejich omezení a mnoho dalších informací, které podle kaptitoly 2 může ganttův diagram obsahovat.

Formát vstupního XML je navržen tak, aby umožňoval obecný popis ganttových diagramů, a to nezávisle na aplikaci jeho použití. Stejný vstupní soubor, tak může používat více aplikací. Na jednotlivá omezení tohoto obecného formátu, která jsou specifická pro program Psgantt, bude v textu výslovně upozorněno.

Dříve než začneme s popisem elementů, popišme si ve stručnosti možnosti syntaktického zápisu vstupního XML, který bude používán v následujícím textu.

Vlastní obsah XML souboru je sázen strojopisem. K řetězci koncové značky každého elementu může být připojen jeden z následujících znaků `?`, `+` nebo `*`, které udávají možný počet opakování daného elementu. Jejich význam popisuje tabulka 4.1. Pokud za elementem nebude uveden žádný znak, znamená to, že musí být uveden právě jednou. Znaky charakterizující počet opakování jsou u elementů uvedeny vždy, když jsou popisovány vzhledem k rodičovskému elementu.

Nepovinné části jsou ohraničeny závorkami `[` a `]`. Jejich význam je totožný s významem znaku `?`, nicméně jsou přehlednější, proto jim dáme v popisu XML přednost. Pokud je mož-

`?`	element může být uveden maximálně jednou nebo nemusí být uveden vůbec
`+`	element musí být uveden minimálně jednou
`*`	element může být uveden jednou, vícekrát nebo též nemusí být uveden vůbec

Tabulka 4.1: Počet výskytů elementů

ných hodnot více, může být jejich seznam uveden rovnou v zápisu syntaxe. Jednotlivé možnosti jsou vzájemně odděleny svislou čarou `|` (např. `yes|no`). Pokud je obsah složitější, pak je místo seznamu antikvou vysázen pouze jejich popisek (např. „název kódování“). Popis také může obsahovat řetězec „`...`“, který zastupuje další obsah, který bude v textu popsán později.

4.2.1 Deklarace XML

Každý dokument v XML by měl začínat takzvanou deklarací. Tato deklarace je nepovinná, ale pokud je v dokumentu uvedena, pak musí být na prvním místě, jako první element dokumentu. Deklarace říká, jaké verzi specifikace XML dokument vyhovuje, jaké je jeho kódování a zda-li je dokument samostatný, či zda obsahuje odkazy na externí soubory.

Syntaxe:

```
<?xml [version="1.0"] [encoding="název kódování"] [standalone="yes|no"]?>
```

Otazník je v tomto případě součástí počáteční a koncové značky elementu. Neuvádí proto možný počet opakování. Deklarace XML může být v dokumentu uvedena maximálně jednou.

Pro Psgantt je důležitý zejména atribut `encoding`, který udává v jaké kódové sadě je dokument napsán. Atribut `encoding` je, stejně jako ostatní, nepovinný. Pokud je vynechán, pak se předpokládá, že je použito kódování `utf-8`. Další Psganttem podporovaná kódování jsou `ISO-8859-2` a `windows-1250`. Jmenovaná kódování obsahují všechny znaky české/slovenské (a tudíž i anglické) abecedy, které jsou běžně dostupné v Psganttem podporovaných operačních systémech.

Další atributy uvedené v deklaraci XML nejsou pro Psgantt důležité a jsou jím ignorovány.

4.2.2 Kořenový element (matlabdata)

Kořenový element je takový element, do něhož jsou uzavřeny všechny ostatní elementy. Každý dokument v XML musí obsahovat právě jeden kořenový element.

Syntaxe:

```
<matlabdata [date="datum vytvoření"] [proccessor="název procesoru"]
            [ver="verze"]>
    <style>...</style>*
    <taskset>...</taskset>+
    [<graph>...</graph>]
</matlabdata>
```

Rodič: (žádný)

Kořenovým elementem dokumentu XML určeným pro program Psgantt je element `matlabdata`. (V obecném XML formátu ganttových diagramů může mít kořenový element i jiné jméno. Stejné pravidlo platí i pro jméno elementu `taskset`.) Nepovinnými atributy elementu `matlabdata` jsou `date`, `proccessor` a `ver`, jejichž hodnotami mohou být libovolné (tedy i prázdné) textové řetězce. Atributy postupně udávají, kdy byl dokument vytvořen, jakým programem a jaké verzi by měl obsah tohoto elementu odpovídat. V této práci je popsána verze 0,2 návrhu elementu `matlabdata`.

Element `matlabdata` může obsahovat jeden, žádný, nebo i více elementů `style`. Jeho data představují definice kaskádových stylů, které specifikují vzhled ganttových diagramů. Pokud elementů `style` bude uvedeno více, bude se jejich význam „sčítat“. To znamená, že výsledek bude stejný, jako kdybychom uvedli element `style` pouze jeden, s obsahem ostatních elementů `style`. Co konkrétně je myšleno slovem „sčítat“ bude patrné z popisu kaskádových stylů, který bude uveden později, v kapitole 5.

Element `taskset` obsahuje vlastní data ganttova diagramu, který chceme vykreslit. Elementů `taskset` může být uvedeno více, minimálně však jeden.

Kořenový element `matlabdata` může obsahovat maximálně jeden element `graph`. Obsah elementu `graph` tvoří data definovaná uživatelem. Obsahem ani strukturou tohoto elementu se Psgantt nezabývá, proto se jím v dalším textu zabývat nebudeme.

4.2.3 Element taskset

Element `taskset` obsahuje všechna data ganttova diagramu. Pro program Psgantt je to element povinný. Pokud bude těchto elementů ve vstupním souboru uvedeno více, bude pro každý z nich vytvořen jeden obrázek. Jméno obrázku bude v tomto případě doplněno řetězcem `_ts_x`, kde místo `x` bude uvedeno pořadové číslo daného elementu `taskset`. Pokud ve vstupním souboru nebude žádný element `taskset`, pak nebude takový soubor validní a Psgantt jej odmítne zpracovat.

Syntaxe:

```
<taskset [id="id"] [ver="verze"] [class="třída"] [style="styl"]>
    [<name>název diagramu</name>]
    <task>...</task>+
    [<precedenceconstrains>...</precedenceconstrains>]
    [<schedule>...</schedule>]
    <note>poznámka</note>*
    [<graphicparam>...</graphicparam>]
    [<tsuserparam>...</tsuserparam>]
</taskset>
```

Rodič: `matlabdata`

Atribut `ver` specifikuje verzi návrhu elementu. Námi popisovaná verze je 1,0. Význam atributů `id`, `class` a `style` bude vysvětlen v kapitole 5 věnované kaskádovým stylům. Proto od následujícího elementu dále, již nebude atributům `id`, `class` ani `style` věnovaná pozornost.

Nepovinný element `name` obsahuje název daného ganttova diagramu. Element `task` obsahuje všechny informace o daném tasku, jak byly uvedeny v kapitole 2. Elementu `task` bude dále věnovaná samostatná sekce, proto jej nyní přeskočíme. Následujícím elementem je nepovinný element `precedenceconstrains`. Ten popisuje všechny precedenční omezení, která se v daném diagramu mohou vyskytnout. Každý element `taskset` může mít tento element maximálně jeden. I jemu bude věnována samostatná část. V pořadí čtvrtým elementem je element `schedule`. Jeho obsahem mohou být informace o rozvrhu, ale také může upřesňovat vzhled ganttova diagramu. Element `schedule` může být uveden maximálně jeden. Rozborem jeho obsahu se budeme zabývat později.

Zbývajícími elementy jsou `note`, `graphicparam` a `tsuserparam`. Element `note` může obsahovat libovolný text s poznámkou o aktuálním elementu `taskset`. Poslední dva jmenované elementy, pokud jsou uvedeny, mohou obsahovat libovolnou strukturu dalších elementů, které mohou být využity jinými aplikacemi. Obsah elementů `note`, `graphicparam` a `tsuserparam` není pro Psgantt důležitý, proto se jimi v dalším textu zabývat nebudeme.

4.2.4 Element task

Element `task` obsahuje všechny informace o dané úloze. Každá úloha může mít, kromě elementů `releasetime`, `duedate` a `deadline`, také svůj vlastní „rozvrh“, který je uložen v elementu `schedule`. Jeho obsah popisuje jednotlivé části, na které může být úloha rozvrhovacím algoritmem rozdělena. Element `schedule` je (pro program Psgantt) zároveň jediným povinným elementem každého elementu `task`. (Pro jiné aplikace může být element `schedule` nepovinný.) Element `schedule` musí být vždy uveden právě jeden. Dalším užitečným elementem je element `graphicparam`, který v sobě může obsahovat element s barvou tasku. Důležitým elementům v tomto odstavci bude v dalším textu věnována samostatná sekce.

Syntaxe:

```
<task [id="id"] [class="třída"] [style="styl"]>
    [<name>jméno úlohy</name>]
    <proctime>číselná hodnota</proctime>*
    [<releasetime>číselná hodnota</releasetime>]
    [<deadline>číselná hodnota</deadline>]
    [<duedate>číselná hodnota</duedate>]
    [<weight>číselná hodnota</weight>]
    <schedule>...</schedule>
    [<asap>číselná hodnota</asap>]
    [<alap>číselná hodnota</alap>]
    [<userparam>...</userparam>]
    <note>text poznámky </note>*
    [<graphicparam>...</graphicparam>]
</task>
```

Rodič: `taskset`

Nepovinný element `name` udává jméno příslušné úlohy. Podobně element `note` obsahuje text s poznámkou k úloze. Hodnota elementu `proctime` specifikuje dobu provádění dané úlohy jí přiřazeným procesorem. Pokud se jedná o jiný procesor, než je jí přiřazený rozvrhovacím algoritmem, může být element `proctime` doplněn o atribut `processor`, jehož hodnota představuje číslo daného procesoru. Dalším z nepovinných elementů je element `weight`, který obsahuje číselnou hodnotu udávající váhu (prioritu) tasku. Elementy `asap`, respektive `alap`, určují čas, kolem kterého je úloha do rozvrhu umísťováná co možná nejdříve, respektive co možná nejpozději. Obsah elementu `userparam` může obsahovat libovolnou XML strukturu s uživatelskými daty.

Všechny elementy popsané v předchozím odstavci jsou pro činnost Psganttu nepodstatné. Mohou sloužit jako zdroj informací pro rozvrhovací algoritmus nebo pro jiné aplikace. Těmito se v dalším popisu zabývat nebudeme.

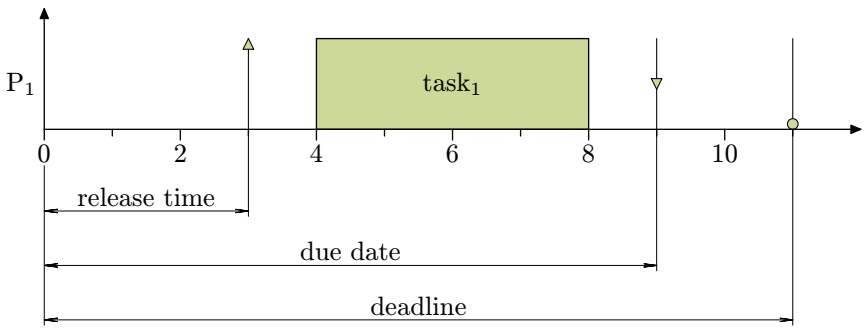
4.2.5 Element releasetime

Hodnota elementu `releasetime` udává čas, ve kterém může být daná úloha nejdříve spuštěna. Tato hodnota musí být nezáporné číslo a může být v obrázku reprezentována pomocí značky. Vzhled značky je předepsán hodnotami kaskádových stylů. Defaultní značkou (viz obrázek 4.1) je trojúhelník otočený směrem nahoru, který je doplněný svislou čarou.

Syntaxe:

```
<releasetime [id="id"] [class="třída"] [style="styl"]>
    číselná hodnota
</releasetime>
```

Rodič: `task`



Obrázek 4.1: Ganttův diagram s hodnotami release time, deadline a due date

4.2.6 Element deadline

Hodnota elementu `deadline` udává čas, do kterého musí být daná úloha ukončena. Tato hodnota musí být nezáporné číslo a může jí být v obrázku přiřazena značka. Další Psganttem akceptovanou hodnotou je řetězec `Inf`. Pokud element `deadline` obsahuje tuto hodnotu, pak se program chová stejně, jako kdyby element nebyl vůbec uveden. To je, nebude pro něj vykreslena žádná

značka ani čára. Vzhled značky je opět předepsán pomocí kaskádových stylů. Defaultní značkou (viz obrázek 4.1) je kolečko, doplněné svislou čarou.

Syntaxe:

```
<deadline [id="id"] [class="třída"] [style="styl"]>
    číselná hodnota
</deadline>
```

Rodič: `task`

4.2.7 Element duedate

Hodnota elementu `duedate` udává čas, ve kterém by měla být daná úloha dokončena. Tato hodnota musí být nezáporné číslo a může být v obrázku reprezentována pomocí značky. V případě hodnoty `Inf` platí pro element `duedate` to samé, co pro element `deadline`. Vzhled značky je opět předepsán hodnotami kaskádových stylů. Defaultní značkou (viz obrázek 4.1) je trojúhelník otočený směrem dolů, doplněný svislou čarou.

Syntaxe:

```
<duedate [id="id"] [class="třída"] [style="styl"]>
    číselná hodnota
</duedate>
```

Rodič: `task`

4.2.8 Element task/schedule

Obsah tohoto elementu je tvořen popisem samotné úlohy. Z něj například vyplývá, na kolik částí je daná úloha rozdělena, jaké má mít box reprezentující úlohu (nebo její část) rozměry, kterým procesorům jsou případné části přiřazeny, atd. Přesný popis jednotlivých vlastností bude předmětem následujících sekcí.

Ve vstupním XML souboru existuje ještě jeden element `schedule`, a to synovský element elementu `taskset`. Oba elementy, byť mají stejné jméno, mají úplně rozdílný význam. Abychom je od sebe navzájem rozlišili, bude vždy jméno jejich elementu uvedeno až za jménem jejich elementu rodičovského. Pro názornost budou ještě jména oddělena lomítkem. Proto budeme vždy dále uvádět `task/schedule`, respektive `taskset/schedule`.

Syntaxe:

```
<schedule [id="id"] [class="třída"] [style="styl"]>
    [<period>číselná hodnota</period>]
    <item>...</item>+
</schedule>
```

Rodič: `task`

4.2.9 Element period

Nepovinný element `period` udává, hodnotu periody úlohy. Perioda se používá pouze v cyklickém rozvrhování. Pokud je tento element uveden, a jeho hodnota je nezáporná, pak jsou na patřičných místech v diagramu zobrazeny svislé čáry, které mají stejnou barvu jakou má příslušný box (viz obrázek 4.2). Grafické parametry elementu `period` mohou být opět nastaveny pomocí kaskádových stylů.

Syntaxe:

```
<period [id="id"] [class="třída"] [style="styl"]>
    číselná hodnota
</period>
```

Rodič: `task/schedule`

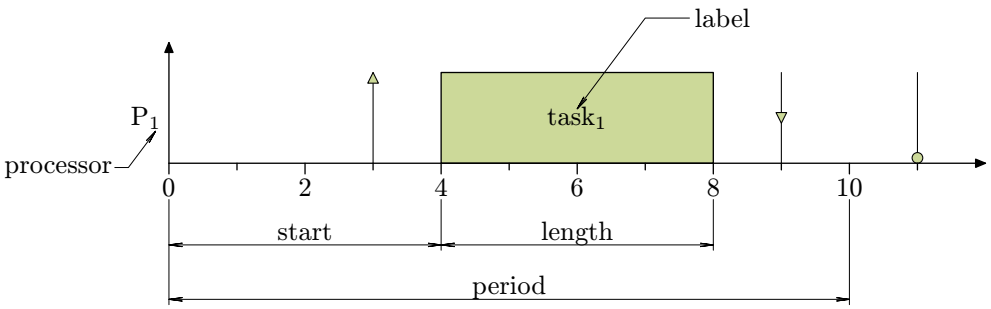
4.2.10 Element item

Element `item` je jediný povinný element elementu `task/schedule`. Počet elementů `item` udává, na kolik částí je daná úloha rozdělena. Povinný atribut `order` specifikuje pořadí jednotlivých `item`ů uvnitř elementu `task/schedule`. Hodnotami atributu `order` mohou být pouze celá kladná čísla. Elementů `label` může být v obecném návrhu XML více, program Psgantt však vyžaduje, aby byl maximálně jeden.

Syntaxe:

```
<item order="pořadové číslo" [id="id"] [class="třída"] [style="styl"]>
    <start>číselná hodnota</start>
    <length>číselná hodnota</length>
    <processor>číselná hodnota</processor>
    [<label>text popisku</label>]
</item>
```

Rodič: `task/schedule`



Obrázek 4.2: Ganttův diagram s hodnotami period, start a length

Příklad 1.

Obrázku 4.2 odpovídá následují obsah elementu `task`:

```
<task>
    <releasetime>3</releasetime>
    <deadline>11</deadline>
    <duedate>9</duedate>
    <schedule>
        <period>10</period>
        <item order="1">
            <start>4</start>
            <length>4</length>
            <processor>1</processor>
            <label format="tex">task$_1$</label>
        </item>
    </schedule>
</task>
```

4.2.11 Element start

Element `start` je pro každý element `item` povinný. Jeho hodnotou musí být nezáporné číslo, které udává, v jakém čase se daná úloha (nebo její část) začala zpracovávat, viz obrázek 4.2. Hodnotu elementu `start` můžeme také chápat jako x-ovou souřadnici levého dolního rohu boxu.

Syntaxe:

```
<start>
    číselná hodnota
</start>
```

Rodič: `item`

4.2.12 Element length

Syntaxe:

```
<length>
    číselná hodnota
</length>
```

Rodič: `item`

Element `length` je také povinným elementem. Jeho hodnotou musí být nezáporné číslo, které udává délku zpracování úlohy (nebo její části), viz obrázek 4.2. Délka zpracování je současně také délkou boxu.

4.2.13 Element processor

Element `processor` je povinným elementem. Jeho hodnota musí být nezáporné celé číslo. To specifikuje procesor, kterým je úloha (nebo její část) zpracovávána. Pokud vstupní XML soubor

neobsahuje jména jednotlivých procesorů (viz element `axisx/labels` v kapitole 4.2.23), pak jsou jména procesorů Psganttem vytvořena automaticky, přičemž součástí každého jména je i číslo procesoru. Popisek procesoru s jeho číslem je pro ilustraci uveden na obrázku 4.2.

Syntaxe:

```
<processor>
    číselná hodnota
</processor>
```

Rodič: `item`

Číslo procesoru zároveň udává vertikální polohu boxu. Procesor s číslem jedna (což je nejnižší číslo) je v obrázku umístěn nejvýš, procesor s nejvyšším číslem je v něm naopak umístěn nejníže. Jaká bude vertikální poloha daného boxu, tedy závisí na celkovém počtu procesorů.

4.2.14 Element item/label

Element `label`, pokud je uveden, obsahuje popisek s názvem úlohy (nebo její části). Ten může být v elementu `item` uveden maximálně jeden. (Pro jiné aplikace jich může být uvedeno více.) Element může obsahovat nepovinný atribut `format`, jehož dvě možné hodnoty `verbatim` a `tex` předepisují způsob vysázení popisku. Pokud atribut `format` není uveden, je defaultně nastaven způsob `verbatim`. Hodnota atributu `format` může být nastavena pomocí kaskádových stylů.

Syntaxe:

```
<label [format="verbatim|tex"] [id="id"] [class="třída"] [style="styl"]>
    text popisku
</label>
```

Rodič: `item`

Obecně všechny texty, tedy i obsah elementu `label`, jsou do obrázku vysázeny pomocí typografického systému TEX. Autorem TEXu je Donald E. Knuth z americké Standfordovy univerzity. (Ve skutečnosti je text sázen pomocí formátu CSLATEX, což je rozšíření standardního TEXu o pravidla české sazby, fonty a podpůrná makra.) TEX i přes to, že vznikl před více než pětadvaceti lety, patří stále mezi nejlepší sázecí programy. K jeho hlavním přednostem patří, že je šířen zdarma, umožňuje dávkové zpracování sazby a jeho funkčnost jde snadno rozšířit.

Zjednodušeně řečeno, můžeme TEX považovat za kompilátor textů. Jeho vstupem je běžný (ASCII) text, jak ho známe například z textového editoru Notepad. Výstupem je pak text vysázený v bitmapě nebo v některém z vektorových formátů (PDF nebo PostScript). Pravidla sazby lze ovlivňovat pomocí příkazů, které mohou být součástí vstupního textu. Jejich popis je možné nalézt v literatuře [6] nebo také v [7]. Protože TEX má, stejně tak jako jiné kompilátory, svou vlastní syntaxi příkazů, je nutné tuto syntaxi ve vstupním textu dodržet. Nedodržením pravidel syntaxe riskujeme, že TEX průběh sazby přeruší a místo vysázeného textu zobrazí pouze chybové hlášení. Je tedy nutné TEXu nějakým způsobem přikázat, aby vstupní text sázel, aniž by jej interpretoval. Tento způsob práce nastane, pokud bude v atributu `format` nastavena

hodnota `verbatim`. Uvedením hodnoty `tex` zapneme pro TEX běžný způsob sazby, čímž na sebe bereme zodpovědnost za obsah elementu `label`.

Uveďme si pro názornost jednoduchý příklad. Předpokládejme, že element `label` obsahuje řetězec „`P$_1$`“. Pokud bude mít atribut `format` hodnotu `verbatim`, pak jako výsledek sazby obdržíme „`P$_1$`“, což je přesně to, co jsme do elementu `label` napsali. Pokud nyní nastavíme atribut `format` na hodnotu `tex`, pak bude výsledkem sazby „P_1“ (viz např. obrázek 4.2). TEX totiž vysází řetězec obklopený znaky dolaru '`$`' v takzvaném matematickém režimu, ve kterém interpretuje znak podtržítka '`_`' jako příkaz k sazbě dolního indexu. Pomocí vhodných příkazů TEXu (respektive CSLATEXu) je možné pohodlně sázet texty například italikou, bezpatkovým, skloněným, tučným písmem nebo jinými dostupnými fonty. Pravidla syntaxe a popis systému TEX přesahuje rámec této práce. Podrobnosti lze najít například v literatuře [6, 8].

Jestliže se v textu popisků vyhneme znakům se speciálním významem, můžeme režim `tex` bez obav používat. Speciálními znaky v TEXu jsou '`\`', '`#`', '`{`', '`$`', '`}`', '`%`', '`&`', '`_`' a '`^`'. Pomocí těchto znaků lze ovlivnit činnost TEXu. Nebude-li v textu žádný ze speciálních znaků uveden, budet se TEX chovat jako bežný „textový editor“, který jednoduše vysází obsah popisku antikvou.

Pro úplnost ještě uveďme, že nezávisle na hodnotě atributu `format`, je možné vysázet obsah elementu `label`, pouze v takzvaném horizontálním módu (viz [9]). Horizontální mód představuje režim sazby, ve kterém je zpracováván pouze obsah jednoho odstavce. Pokud bude obsah elementu `label` obsahovat příkaz vyžadující sazbu například ve vertikálním módu (tj. při práci s více odstavci), obdržíme pouze chybové hlášení. Uvedená skutečnost pro nás neznamená prakticky žádné omezení, protože asi jen málokdy budeme chtít, aby byl obsah popisku úlohy delší než jeden odstavec. Přesný popis činnosti TEXu je popsán v literatuře [10].

4.2.15 Element task/graphicparam

Z pohledu Psganttu je v nepovinném elementu `task/graphicparam` zajímavý pouze jediný element, kterým je element `color`. Ten může obsahovat barvu, kterou je vykreslen příslušný task. Element `color` musí být v `task/graphicsparam` uveden jako první element. Pokud bude v elementu `graphicparam` uvedeno víc elementů `color`, bude výsledná barva tasku nastavena podle elementu, který byl uveden jako poslední. Ostatní obsah elementu `task/graphicparam` může být libovolný.

Syntaxe:

```
<graphicparam>
    <color>...</color>*
    ...
</graphicparam>
```

Rodič: `task`

4.2.16 Element color

Obsah elementu `color` definuje barvu, kterou je vykreslena příslušná úloha. Stejnou barvou jsou pak také vykresleny i značky symbolů release time, deadline, due date a period. Barva všech

jmenovaných prvků ganttova diagramu může být změněna pomocí kaskádových stylů.

Element `color` může mít nepovinný atribut `type`. Tomu mohou být přiřazeny hodnoty `rgb`, `shortname` nebo `longname`. V závislosti na hodnotě atributu `type`, můžeme syntaktický zápis rozdělit na čtyři případy.

1. Atribut `type` obsahuje hodnotu `rgb`.

Syntaxe:

```
<color type="rgb">
    <r>číselná hodnota</r>
    <g>číselná hodnota</g>
    <b>číselná hodnota</b>
</color>
```

Rodič: `task/graphicparam`

V tomto případě je barva definovaná pomocí složek `r`, `g` a `b`. Hodnota jednotlivých složek musí mýt číslo z intervalu $\langle 0; 1 \rangle$. Hodnoty (0,0,0) představují černou barvu, hodnoty (1,1,1) představují bílou barvu.

2. Atribut `type` obsahuje hodnotu `shortname`.

Syntaxe:

```
<color type="shortname">
    r|g|b|c|m|y|k|w
</color>
```

Rodič: `task/graphicparam`

Barva je daná přímo obsahem elementu `color`. Jednotlivé hodnoty postupně odpovídají barvám: red, green, blue, cyan, magenta, yellow, black nebo white.

3. Atribut `type` obsahuje hodnotu `longname`.

Syntaxe:

```
<color type="longname">
    red|green|blue|cyan|magenta|yellow|black|white
</color>
```

Rodič: `task/graphicparam`

4. Atribut `type` není uveden.

Syntaxe:

```
<color>
    #rgb|#rrggbb|aqua|black|blue|fuchsia|gray|green|lime|
    maroon|navy|olive|purple|red|white|yellow|silver|teal
</color>
```

Rodič: `task/graphicparam`

V případě kdy atribut `type` chybí, může být barva zadaná jednou hodnotou z výčtu. Hodnoty výčtu odpovídají barvám známým z jazyka HTML (HyperText Markup Language). První dvě hodnoty výčtu mohou místo jednotlivých písmen 'r', 'g', 'b', obsahovat hexadecimální číslice. (Těmi jsou `0`, `1`, `2`, `3`, `4`, `5`, `6`, `7`, `8`, `9`, `a`, `b`, `c`, `d`, `e`, `f`.) Zápis `#ffffff` představuje bílou barvu, ta by se pomocí zkráceného způsobu zápisu napsala jako `#fff`. Každá barevná složka je v hexadecimálním zápisu zastoupena dvěmi, respektive jedním, hexadecimálním číslem.

Jestliže k elementu `task` nebude existovat žádný element `graphicparam/color`, bude jeho barva bílá, pokud mu nebude v kaskádových style nastavena jiná hodnota.

4.2.17 Element precedenceconstrains

Obsah elementu `precedenceconstrains` tvoří všechna precedenční omezení daného ganttova diagramu. Jak bylo uvedeno v kapitole 2, precedence představují pořadí ve kterém musí být úlohy zpracovány. Pokud je element `precedenceconstrains` uveden, pak musí obsahovat minimálně jeden element `precedence`.

Syntaxe:

```
<precedenceconstrains [id="id"] [class="třída"] [style="styl"]>
    <precedence>...</precedence>+
</precedenceconstrains>
```

Rodič: `taskset`

4.2.18 Element precedence

Element `precedence` popisuje právě jednu precedenční závislost (precedenci). V obrázku je precedence znázorněna pomocí šipky (viz obrázek 2.1). Její grafické parametry mohou být opět nastaveny pomocí kaskádových stylů. Zpracování úlohy, ze které šipka vychází, musí být dokončeno dříve (nebo nejpozději ve stejnou dobu), než se začne zpracovávat úloha do které šipka vede. Ze které úlohy šipka vychází, je dáno obsahem povinného atributu `from`. Úlohu, do které šipka vede, udává povinný atribut `to`. Vlastními hodnotami atributů `from` a `to` musí být hodnoty atributů `id` příslušných elementů `task`.

Syntaxe elementu `precedence` závisí na tom, zda-li je k ní přiřazena určitá číselná hodnota (1. případ) nebo není (2. případ). Její hodnota udává váhu dané precedence. Pokud váha není uvedena, je považovaná za jednotkovou. V případě, kdy má váha precedence jinou hodnotu než jedna, může být v obrázku zobrazena. Možnosti zobrazení váhy lze opět nastavit kaskádovými styly.

Syntaxe (1. případ):

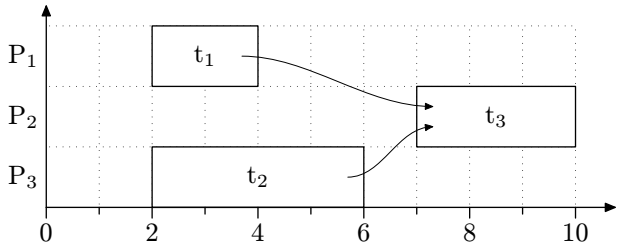
```
<precedence from="odkud" to="kam" [id="id"] [class="třída"] [style="styl"]>
    číselná hodnota
</precedence>
```

Rodič: `precedenceconstrains`

Syntaxe (2. případ):

```
<precedence from="odkud" to="kam" [id="id"] [class="třída"] [style="styl"]/>
```

Rodič: `precedenceconstrains`



Obrázek 4.3: Ganttův diagram s precedencemi

Příklad 2.

Za předpokladu, že uvedené tasky t_1, t_2 a t_3 mají postupně počáteční značky

`<task id="t1">`, `<task id="t2">` a `<task id="t3">`,

bude obrázku 4.3 odpovídat následující obsah elementu `precedenceconstrains`:

```
<precedenceconstrains>
    <precedence from="t1" to="t3"/>
    <precedence from="t2" to="t3"/>
</precedenceconstrains>
```

4.2.19 Element taskset/schedule

Tento element blíže popisuje daný ganttův diagram. Mohou v něm být obsaženy informace jak o rozvrhovacím algoritmu, kterým byl diagram vytvořen, tak například názvy procesorů. Dále v něm mohou být obsaženy grafické parametry, jakými jsou například rozměry jednotek os x a y, parametry mřížky rozvrhu, ale také příkazy jazyka METAPOST, pomocí kterých lze do obrázku dále kreslit.

Syntaxe:

```
<schedule [id="id"] [class="třída"] [style="styl"]>
    [<description>popis rozvrhovacího algoritmu</description>]
    [<time>číselná hodnota </time>]
    [<iterations>číselná hodnota </iterations>]
    [<memory>číselná hodnota </memory>]
    [<gantt>...</gantt>]
</schedule>
```

Rodič: `taskset`

Element `description` popisuje rozvrhovací algoritmus. Element `time` udává dobu, kterou vytvoření rozvrhu algoritmu tvrvalo. Obsah elementu `memory` poskytuje informaci o velikosti

paměti (v bajtech), která byla daným algoritmem alokována. Tyto elementy mají informativní charakter a Psgantt se jimi nezabývá.

Posledním elementem v `taskset/schedule` je nepovinný element `gantt`. Ten může upřesňovat vzhled ganttova diagramu. Jemu bude věnována následující sekce.

4.2.20 Element gantt

Tento element může obsahovat nepovinné elementy `axisx` a `axisy`, které udávají nastavení os x a y ganttova diagramu. Dále pak může obsahovat nepovinný element `mpcommand`, pomocí kterého lze do obrázku přidávat další příkazy jazyka METAPOST.

Syntaxe:

```
<gantt [id="id"] [class="třída"] [style="styl"]>
    [<axisx>...</axisx>]
    [<axisy>...</axisy>]
    <mpcommand>...</mpcommand>*
</gantt>
```

Rodič: `taskset/schedule`

4.2.21 Element axisx

Element `axisx` upřesňuje nastavení x-ové osy ganttova diagramu.

Syntaxe:

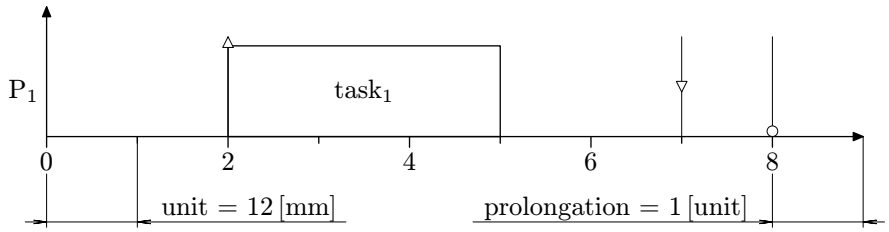
```
<axisx [unit="délka"] [prolongation="délka"] [cutfrom="odkud"] [cutto="kam"]
       [id="id"] [class="třída"] [style="styl"]>
    [<ticks>číselná hodnota </ticks>]
    [<labels>...</labels>]
    <grid/>*
</axisx>
```

Rodič: `gantt`

Atribut `unit` udává délku jednotky x-ové osy v milimetrech. Její hodnotou musí být nenulové kladné číslo. Pokud atribut `unit` nebude uveden, pak se místo něj použije hodnota `xunit` definovaná v kaskádových stylech, viz kapitola 5.2.1. V takto nastavených jednotkách [unit] jsou vyjádřeny všechny x-ové hodnoty ganttova diagramu. Jsou to například hodnoty elementů `releasetime`, `duedate`, `deadline`, `length` a dalších. Výjimky představují např. tloušťky čar a jim podobné hodnoty, které jsou udávány v takzvaných big pointech (bp). Jeden bp má délku 1/72 palce, což je přibližně 0,353 milimetrů).

Atribut `prolongation` udává délku o kterou bude prodloužena x-ová osa diagramu. Velikost je udávána v jednotkách [unit], která byla popsána dříve. Prodloužení je bráno vzhledem k poslednímu zobrazenému prvku vpravo. (Prvky zde označují jednotlivé symboly ganttova diagramu. K nim ale nepatří další grafické objekty, které mohou být do obrázku přidány pomocí

příkazů METAPOSTu.) Pokud nebude tento atribut uveden, bude místo něj použita hodnota `xover` z kaskádových stylů, viz kapitola 5.2.1.



Obrázek 4.4: Diagram z jednotkami x-ové osy

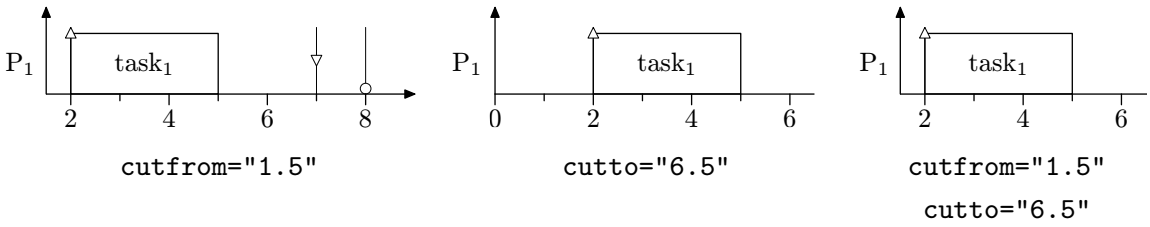
Příklad 3.

Pokud bude atribut `unit="12"` a atribut `prolongation="1"`, bude mít obrázek jednotky, které jsou znázorněny na obrázku 4.4.

Hodnota atributu `cutfrom` specifikuje místo na ose x, od kterého bude diagram ořezán. Jeho hodnota může být kladné číslo, včetně nuly. Pokud atribut `cutfrom` nebude uveden, pak obrázek nebude z levé strany ořezán.

Atribut `cutto` představuje místo na ose x, do kterého bude trvat ořezání obrázku. Jeho hodnotou může být pouze kladné nenulové číslo. Pokud atribut `cutto` nebude uveden, pak obrázek nebude ořezán z pravé strany.

Pokud bude uveden pouze jediný z dvojice atributů `cutfrom` a `cutto`, pak bude obrázek ořezán od této hodnoty napravo až do konce obrázku, resp. od této hodnoty nalevo (až do jeho začátku). Ořezání je možné z výhodou použít u velmi rozsáhlých rozvrhů, u kterých nás zajímá pouze jejich určitá část.



Obrázek 4.5: Diagram s ořezáním v ose x

Příklad 4.

Nastavme u předchozího obrázku atribut `unit="6.5"` a sledujme jaký vliv na něj budou mít hodnoty atributů `cutfrom` a `cutto`. Výsledky jsou zobrazeny v obrázku 4.5.

4.2.22 Element ticks

Syntaxe:

```
<ticks [labeled="číselná hodnota"]>
    číselná hodnota
</ticks>
```

Rodič: `axisx`, `axisy`

V případě, že je rodičovským elementem `axisx`, má element `ticks` význam popsaný v následujících odstavcích. V elementu `axisy` tento element (pro program Psgantt) žádný význam nemá.

Obsah nepovinného elementu `ticks` udává četnost kratších čárek, které jsou zobrazeny pod osou x. Pokud bude například jeho hodnota číslo 2, pak to znamená, že malá čárka bude na ose x umístěna každou druhou jednotku. (Pokud by byla 3, pak každou třetí, atd.) Hodnota elementu `ticks` musí být kladné celé číslo. Jestliže element nebude uveden, bude četnost čárky nastavena podle hodnoty `xmarks-small` v kaskádových stylech, viz kapitola 5.2.1.

Nepovinný atribut `labeled` udává, četnost delších čárek a popisků pod osou x. Jeho hodnota je vztažena k hodnotě elementu `ticks` a říká, na kolikáté malé čárce má být zobrazen popisek a zároveň i delší čárka. Pokud bude atribut `labeled` chybět, bude jeho hodnota nastavena na hodnotu elementu `ticks`.

Například obrázkům v 4.5 odpovídá nastavení `<ticks labeled="2">1</ticks>`.

4.2.23 Element axisx/labels

Pomocí tohoto elementu můžeme nastavovat své vlastní popisky osy x a jejich polohu vůči dříve popsaným čárkám.

Syntaxe:

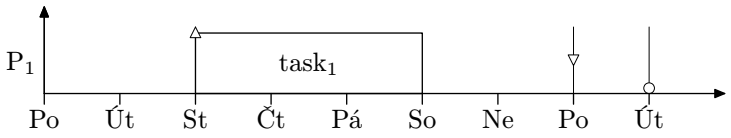
```
<labels [between="true|false"]>
    <label>text popisku</label>+
</labels>
```

Rodič: `axisx`

Element `axisx/labels` musí obsahovat alespoň jeden element `label`. Ten obsahuje text s vlastním popiskem a platí pro něj stejná pravidla jako pro element `item/label`.

Popisky na ose x jsou postupně brány podle pořadí elementů `label` v `labels`. X-ová hodnota, na kterou se popisek umístí, je dána nastavením elementu `ticks` popsaným výše. Pokud je elementů `label` méně než odpovídá počtu dlouhých čárek, jsou popisky brány v pořadí zase od začátku. Takto lze vytvořit popisky, které se na ose x periodicky opakují.

Nepovinný atribut `between` udává, jestli budou popisky vůči poloze dlouhých čárek centrovány (hodnota `true`) či nikoliv (hodnota `false`).

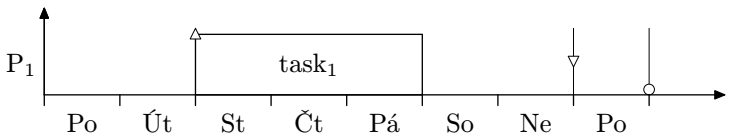


Obrázek 4.6: Příklad popisků osy x

Příklad 5.

Jestliže bude mít element `axisx` následující obsah, dostaneme obrázek 4.6.

```
<axisx unit="10">
    <ticks>1</ticks>
    <labels>
        <label format="tex">Po</label>
        <label format="tex">Út</label>
        <label format="tex">St</label>
        <label format="tex">Čt</label>
        <label format="tex">Pá</label>
        <label format="tex">So</label>
        <label format="tex">Ne</label>
    </labels>
</axisx>
```



Obrázek 4.7: Centrované popisky osy x

Příklad 6.

Pokud nyní do elementu `labels` přidáme text `between="true"`, obdržíme obrázek 4.7. Obsahy elementů `label` zůstávají stejné jako v příkladě 5.

```
<axisx unit="10">
    <ticks>1</ticks>
    <labels between="true">
        .
        .
        .
    </labels>
</axisx>
```

4.2.24 Element axisx/grid

Nepovinný element `axisx/grid` nastavuje na ose x mřížku (grid).

Syntaxe:

```
<grid [tick="číselná hodnota"] [from="číselná hodnota"] [to="číselná hodnota"]
      [id="id"] [class="třída"] [style="styl"]/>
```

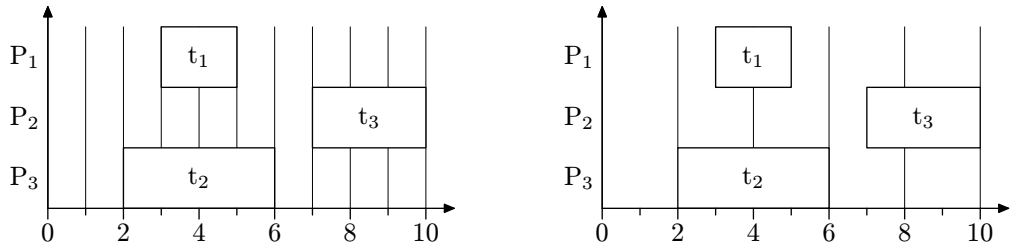
Rodič: `axisx`

Hodnota atributu `tick` udává četnost gridu. Přesněji řečeno, udává počet ticků (malých čárek) na kterých bude vykreslena jedna čára gridu. Pokud atribut `tick` nebude uveden, vykreslí se čáry gridu podle hodnoty atributu `labeled` elementu `ticks`. Jestliže nebude uveden ani ten, bude grid vykreslen u každého ticku (tj. bude hodnota atributu `tick` rovna jedné).

Atribut `from` udává, od kterého ticku (včetně) se má grid vykreslovat. Pokud nebude uveden, bude grid vykreslen od prvního ticku. Naopak hodnota atributu `to` udává, do kterého ticku (včetně) bude grid vykreslen. Pokud nebude atribut `to` uveden, bude se grid vykreslen až do posledního ticku v grafu.

V případě, že budeme chtít vykreslit pouze grid s jedinou čarou, pak musíme uvést oba atributy `from` a `to`, ty musí mít navíc i stejnou hodnotu.

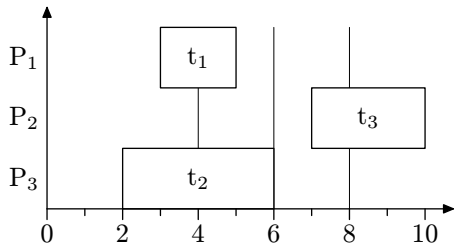
Vzhled čar zadaných pomocí elementů `grid` může být opět nastaven kaskádovými styly.



Obrázek 4.8: Gridy v ose x

Příklad 7.

Levý obrázek v 4.8 odpovídá nastavení `<grid tick="1"/>`. Pravému obrázku v 4.8 odpovídá nastavení `<grid tick="2"/>`.



Obrázek 4.9: Mřížka v ose x zadaná pomocí rozsahu

Příklad 8.

Obrázek 4.9 odpovídá následujícímu nastavení elementu `grid`.

```
<grid tick="2" from="4" to="8"/>
```

4.2.25 Element axisy

Element `axisy` upřesňuje nastavení y-ové osy ganttova diagramu. Jeho nastavení je podobné jako u elementu `axisx`.

Syntaxe:

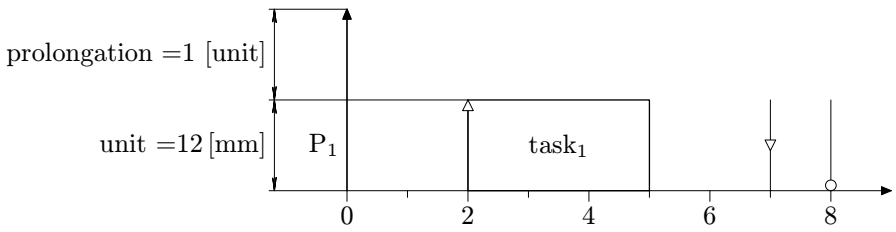
```
<axisy [unit="délka"] [prolongation="délka"] [cutfrom="odkud"] [cutto="kam"]
       [id="id"] [class="třída"] [style="styl"]>
    [<ticks>číselná hodnota </ticks>]
    [<labels>...</labels>]
    <grid/>*
</axisy >
```

Rodič: `gantt`

Nastavení nepovinného elementu `ticks` v případě y-ové osy Psgantt nebere v úvahu, takže jej v dalším výkladu přeskočíme.

Atribut `unit` udává délku jednotky y-ové osy v milimetrech, která je současně výškou všech boxů diagramu. Její hodnotou musí být nenulové kladné číslo. Pokud atribut `unit` nebude uveden, pak se místo něj použije hodnota `yunit` definovaná v kaskádových stylech, viz kapitola 5.2.1. K takto nastavených jednotkách [unit] jsou vztaženy všechny y-ové hodnoty ganttova diagramu. Jsou to například výšky značek odpovídající elementům `releasetime`, `duedate`, `deadline` a dalších. Na výjimky tohoto pravidla bude upozorněno.

Atribut `prolongation` udává délku o kterou bude prodloužena y-ová osa diagramu. Velikost je udávána v jednotkách [unit], která byla popsána dříve. Prodloužení je bráno vzhledem k nejvýše zobrazenému prvku. (Prvky zde označují jednotlivé symboly ganttova diagramu. K nim ale nepatří další grafické objekty, které mohou být do obrázku přidány pomocí příkazů METAPOSTu.) Pokud nebude tento atribut uveden, bude místo něj použita příslušná hodnota `yover` z kaskádových stylů, viz kapitola 5.2.1.



Obrázek 4.10: Diagram s jednotkami y-ové osy

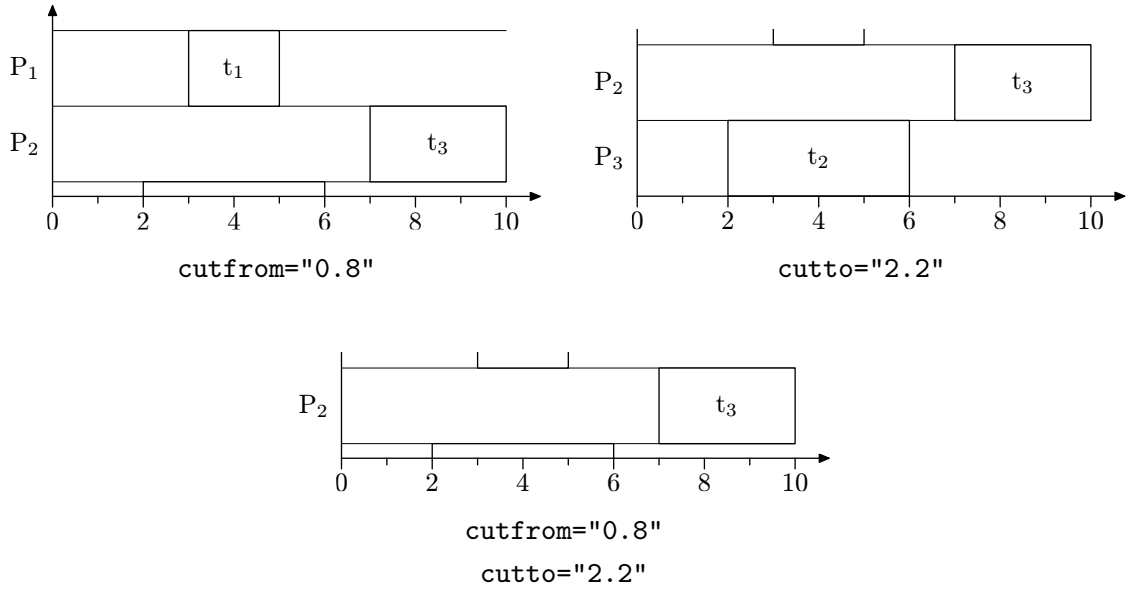
Příklad 9.

Pokud bude atribut `unit="12"` a atribut `prolongation="1"`, bude mít diagram jednotky, které jsou znázorněny na obrázku 4.10.

Hodnota atributu `cutfrom` specifikuje místo na ose y, od kterého bude diagram ořezán. Jeho hodnota může být kladné číslo, včetně nuly. Pokud atribut `cutfrom` nebude uveden, pak obrázek nebude z dolní strany ořezán.

Atribut `cutto` představuje místo na ose y, do kterého bude trvat ořezání obrázku. Jeho hodnotou může být pouze kladné nenulové číslo. Pokud atribut `cutto` nebude uveden, pak obrázek nebude ořezán z horní strany.

Pokud bude uveden pouze jediný z dvojice atributů `cutfrom` a `cutto`, pak bude obrázek ořezán od této hodnoty nahoru až do konce obrázku, resp. od této hodnoty dolů (až do jeho začátku). Ořezání je možné z výhodou použít u velmi rozsáhlých rozvrhů, u kterých nás zajímá pouze jejich určitá část.



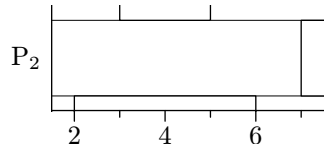
Obrázek 4.11: Diagram s ořezáním v ose y

Příklad 10.

Vezměme si například diagram z obrázku 4.9, ten budeme chtít v y-ové ose ořezat. Doplňme jej pro názornost ještě mřížkou na y-ové ose. V něm nastavme jednotky os x a y postupně na 6 a 10 milimetrů. Sledujme jaký na něj budou budou mít vliv hodnoty atributů `cutfrom` a `cutto`. Výsledky ořezání jsou uvedeny na obrázku 4.11.

Příklad 11.

Ořezání os x a y můžeme vzájemně kombinovat. Vezměme si opět diagram z obrázku 4.9. Pro atributy `cutfrom="1.5"` a `cutto="7.5"` elementu `axisx` a atributy `cutfrom="0.8"` a `cutto="2.2"` vznikne obrázek 4.12. (Jednotky jsou stejné jako v předcházejícím příkladu.)



Obrázek 4.12: Diagram ořezaný v obou osách

4.2.26 Element axisy/labels

Pomocí tohoto elementu můžeme nastavovat názvy procesorů. Ty se zobrazí vlevo od osy y. Jejich vertikální poloha je vždy centrována vzhledem k výšce boxů.

Syntaxe:

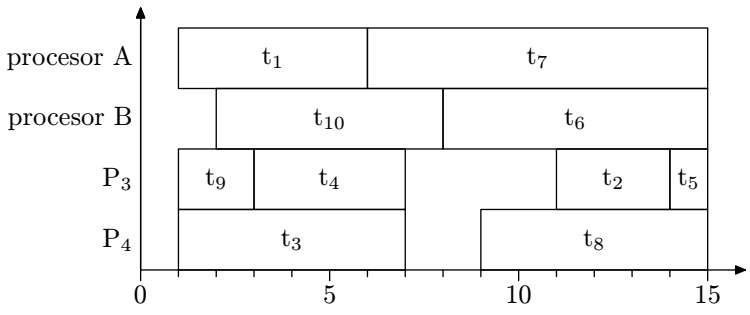
```
<labels [between="true|false"]>
    <label>text popisku</label>+
</labels>
```

Rodič: `axisy`

Uvnitř elementu `axisy/labels` musí být alespoň jeden element `label`. Ten obsahuje text s popiskem procesoru. Pro element `label` platí stejná pravidla jako pro element `item/label`.

Popisky jednotlivých procesorů jsou postupně brány podle pořadí, v jakém jsou uvedeny elementy `label` v `labels`. Narozdíl od elementu `axisx/labels` se v elementu `axisy/labels` popisky periodicky neopakují. Pokud bude počet elementů `label` v `axisy/labels` menší než je počet procesorů, budou zbývajícím procesorům přiřazeny defaultní jména. Ty mají tvar P_n, kde n je číslo procesoru. Z výše uvedeného vyplývá, že pokud budeme chtít změnit defaultní jméno procesoru P_1 (tj. nejspodnějšího procesoru), pak musíme v `labels/label` uvést popisky pro všechny procesory.

Nepovinný atribut `between` je programem ignorován.



Obrázek 4.13: Změna názvů procesorů

Příklad 12.

Nastavme element `axisy` následujícím způsobem. Jemu odpovídající popisky procesorů jsou zobrazeny na obrázku 4.13.

```
<axisx>
    <labels>
        <label format="tex">procesor A</label>
        <label format="tex">procesor B</label>
    </labels>
</axisx>
```

Všimněme si, že se změnily pouze jména prvních dvou procesorů. Ostatním zůstaly přiřazeny jejich defaultní jména.

4.2.27 Element axisy/grid

Nepovinný element `axisy/grid` nastavuje na ose y mřížku (grid).

Syntaxe:

```
<grid [tick="číselná hodnota"] [from="číselná hodnota"] [to="číselná hodnota"]
      [id="id"] [class="třída"] [style="styl"]/>
```

Rodič: `axisy`

Hodnota nepovinného atributu `tick` je programem ignorována.

V případě y-ové osy jsou gridy číslovány následujícím způsobem. Grid s pořadovým číslem 0 je umístěn nad nejvyšším procesorem, tj. P_1. Grid s číslem 1 pod ním, grid číslem 2 pod dalším procesorem, to je P_2, atd. Obecně tedy platí, že pořadové číslo gridu musí leže v intervalu $\langle 0, p \rangle$, kde p je celkový počet procesorů v diagramu.

Atribut `from` udává, od kterého pořadového čísla (včetně) se má grid vykreslovat. Pokud nebude uveden, bude grid vykreslen od nultého. Naopak hodnota atributu `to` udává, do kterého pořadového čísla (včetně) bude grid vykreslen. Pokud nebude atribut `to` uveden, bude se grid vykreslen až do posledního čísla v grafu, které je rovno počtu procesorů.

V případě, že budeme chtít vykreslit pouze grid s jedinou čarou, pak musíme uvést oba atributy `from` a `to`, ty musí mít navíc i stejnou hodnotu.

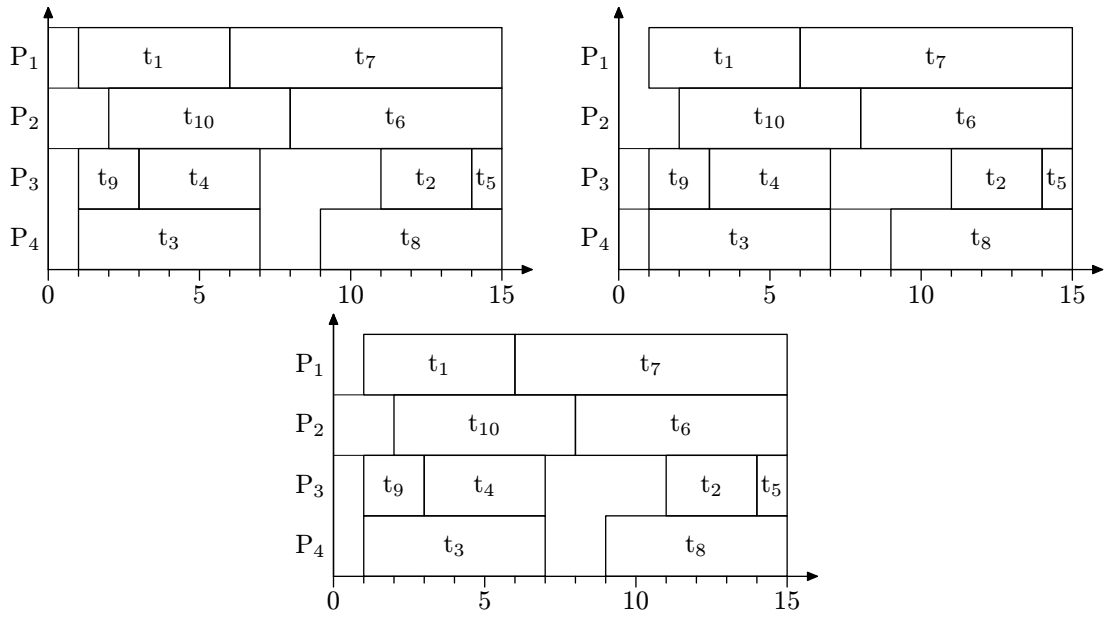
Vzhled čar zadaných pomocí elementů `grid` může být opět nastaven kaskádovými styly, jímž je věnována samostatná kapitola 5.2.11. Pomocí kaskádových stylů mohou být také nastaveny mezery nad a pod čarou gridu.

Příklad 13.

Vezměme si například diagram z obrázku 4.13. Levému obrázku v 4.14 odpovídá nastavení `<grid to="2"/>`. Pravému obrázku v 4.14 odpovídá nastavení `<grid from="2"/>`. Nastavení dolního obrázku je `<grid from="1" to="2"/>`.

4.2.28 Element mpcommand

Tento element může obsahovat příkazy jazyka METAPOST, pomocí kterých můžeme do obrázku přidávat další grafiku. METAPOSTu bude věnovaná samostatná kapitola 8.



Obrázek 4.14: Gridy v ose y

5 Popis kaskádových stylů

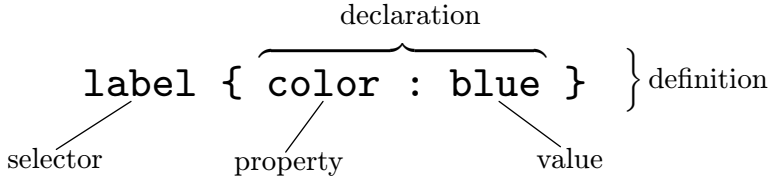
5.1 Vlastnosti CSS

5.1.1 Úvod do CSS

Většina značek popsaných v kapitole 4 popisuje pouze sémantický význam obsahu, který je v nich uzavřen. Pro nás je však nutné jejich obsah nějakým způsobem zformátovat a poté ve formě obrázku zobrazit uživateli. Za tímto účelem musí proběhnout nějaký krok, ve kterém jsou na vstupní XML soubor aplikovány určité formátovací informace. Tento krok převede značkování významové na značkování formátovací. Jednou z možností jak provést tento převod je použít kaskádové stylové šablony (Cascading Style Sheets), zkráceně CSS.

CSS představují jazyk, pomocí kterého můžeme definovat vlastnosti (property) každého elementu použitého ve vstupním XML souboru. Počet vlastností, jejich názvy a hodnoty (value) si přitom můžeme volit sami. Kterého elementu se dané vlastnosti týkají, udává takzvaný selektor. Ten je definován pomocí vzoru, kterému musí daný XML element vyhovět. (Jestliže element vzoru nevyhoví, nebudou jeho vlastnosti daným selektrorem nastaveny.) Úplná syntaxe vzorů je poměrně složitá. V práci budou uvedeny pouze ty vzory, pro které má program Psgantt zabudovánu podporu. Ostatní vzory jsou programem ignorovány. Kompletní seznam vzorů je možné najít ve specifikaci k CSS [12].

5.1.2 Syntaxe CSS



Obrázek 5.1: Popis syntaxe CSS

CSS se skládá z posloupnosti pravidel (definition). Každé pravidlo se může týkat jednoho nebo více elementů. Pravidlo začíná selektorem (selector), který specifikuje skupinu elementů. Selektor je následován deklarací vlastností (declaration), které určují vlastnosti (property) vybrané skupiny elementů. Deklarace vlastností je uzavřena ve složených závorkách. Pokud má selektor více deklarací, jsou navzájem odděleny středníky (za poslední deklarací středník už být nemusí). Hodnoty (value) jsou od názvů vlastností odděleny dvojtečkou. Pokud má vlastnost více hodnot, jsou navzájem odděleny mezerou. Selektorů může být v definici uvedeno více, v tom případě musí být odděleny čárkou.

Příklad 1.

Platnou definicí jednoho pravidla CSS by mohl být například následující text:

```
jmeno_elementu {
    vlastnost1 : hodnota1 hodnota2;
    vlastnost2 : hodnota
}
```

Toto pravidlo se bude aplikovat na všechny elementy, které mají jméno `jmeno_elementu`. Vlastnostmi těchto elementů pak budou `vlastnost1` a `vlastnost2`. Kde vlastnost `vlastnost1` má hodnoty `hodnota1` a `hodnota2`. Vlastnost `vlastnost2` má pouze jedinou hodnotu `hodnota`.

5.1.3 Způsoby vložení CSS do XML

Kaskádový styl můžeme k dokumentu XML připojit několika způsoby Můžeme jej definovat přímo v dokumentu (1) nebo v externím souboru (2). Dalším Psganttem podporovaným způsobem, je uvedení jména externího souboru jako parametru příkazové řádky (3). Poslední možností je, definovat styl přímo určitému elementu (4). Všechny způsoby můžeme vzájemně kombinovat.

1. Definice stylů můžeme zapsat přímo do elementu `matlabdata/style`.

2. Pokud budou definice uloženy v nějakém externím souboru (např. `soubor.css`), pak je můžeme k danému XML souboru připojit uvedením pravidla

   ```
   @import url("soubor.css");
   ```

   uvnitř zmíněného elementu `style`. Řetězec uvnitř `url("...")` musí obsahovat jméno souboru, doplněné o relativní cestu vůči vstupnímu XML souboru. Uvedení absolutní cesty nebo URL nejsou Psganttem podporovány. To samé platí i v případě vícenásobného zanoření importovaných souborů. Pravidlo `@import` musí být uvedeno v CSS vždy jako první.

3. Tento způsob je založen na uvedení jména importovaného souboru jako parametru příkazové řádky. (Popis spouštění Psganttu z příkazové řádky a její parametry jsou popsány v 7.1.) Oproti bodu (2) se tento způsob liší pouze tím, že v něm můžeme uvádět také úplnou cestu k souboru CSS.

4. Posledním způsobem je zapsaní textu s deklarací vlastností (viz obrázek 5.1) do atributu `style` konkrétního elementu, jehož vlastnosti chceme změnit.

Příklad 2.

Následující text uvádí jednotlivé způsoby vložení CSS do vstupního XML. Předpokládejme, že obsahem souboru `soubor.css` je text:

```
label {color : blue}.
```

Protože v pořadí třetí uvedený způsob vložení CSS se zapisuje pouze do příkazové řádky (ne tedy do vstupního XML), nebude v tomto příkladu uveden.

1. způsob:

```
<style>
  label{
    color : red
  }
</style>
```

2. způsob:

```
<style>
  @import url("soubor.css");
</style>
```

4. způsob:

```
<label style="color:white">
  ...
</label>
```

5.1.4 Hledání vlastností elementu

Předpokládejme, že jsou do vstupního XML souboru vloženy všechny definice CSS z předcházejícího příkladu. Text vstupního XML je pro názornost uveden na obrázku 5.2.

```
<matlabdata>
  <style>
     @import url("soubor.css");
     label { color : red }
  </style>
  <taskset>
     <task>
        <schedule>
           <item>
             ...
             <label style="color:white">...</label>
           </item>
        </schedule>
     </task>
  </taskset>
</matlabdata>
```

Obrázek 5.2: Ukázkový XML soubor

V něm jsme nastavili elementu `label` vlastnost `color`. Jestliže budeme chtít obsah elementu `label` vytisknout, musíme přesně vědět, kterou z uvedených barev mu máme přiřadit. Pořadí

vyhledávání vlastností daného elementu udává následující seznam. Pokud je v nějakém bodu hledaná vlastnost nalezena, je vlastnosti nastavena příslušná hodnota a vyhledávání je ukončeno.

1. Nejprve je prohledán obsah atributu `style` daného elementu.
2. Vlastnost se hledá mezi vlastnostmi rodičovského elementu. V případě neúspěchu se hledání rekurzivně opakuje do té doby, dokud není hledaná vlastnost nalezena, nebo dokud nejsou prohledány vlastnosti kořenového elementu.
3. Prohledává se obsah případného externího souboru, který byl připojen pomocí způsobu číslo 3.
4. Jsou prohledávány jednotlivé definice pravidel CSS z elementu `matlabdata/style`.
5. Je prohledán CSS soubor s defautlním nastavením každého elementu.

Body 3, 4 a 5 jsou ještě dále rozšířeny o takzvanou specificitu selektoru. Její hodnota rozhoduje o tom, který selektor bude vybrán, v případě, že jich danému elementu vyhoví více. V tomto případě se elementu nastaví vlastnosti selektoru s nejvyšší specificitou. Hodnoty specificit všech Psganttem podporovaných selektorů jsou zobrazeny v tabulce 5.1. Jednotlivé typy selektorů budou popsány dále, v kapitole 5.1.5.

specificita	typ selektoru
101	`element#id`
100	`#id`
11	`element.class`
10	`.class`
2	`parent_element > element`
1	`element`

Tabulka 5.1: Hodnoty specificit selektorů CSS

5.1.5 Typy selektorů

Selektory udávají pravidla, podle kterých se rozhodne, zda elementu přiřadí vlastnosti daného selektroru, či nikoliv.

V Psganttem podporovaných selektorech (viz tabulka 5.1) se mohou vyskytnout následující čtyři prvky z XML. Ty jsou zobrazeny v tabulce 5.2.

Selektor Element

Selektor element je nejjednodušším selektorem. Selektor typu element obsahují všechny definicice CSS, které lze popsat zápisem

```
element { ... }
```

`id`	zastupuje hodnotu atributu `id`. Ten by měl svým obsahem (spolu se jménem elementu) jednoznačně identifikovat každý element.
`class`	zastupuje hodnotu atributu `class`, který by měl obsahovat název třídy.
`element`	zastupuje jméno elementu.
`parent_element`	zastupuje jméno rodičovského elementu.

Tabulka 5.2: Prvky XML v selektorech CSS

V něm může být text `element` nahrazen libovolným jménem elementu v XML. Vlastnosti selektorů typu element se aplikují na elementy, jejichž jméno je stejné jako jméno v řetězci `element`.

Příklad 3.

Pokud bude CSS obsahovat definici `label{color:blue}`, pak se nastaví vlastnost `color` všech elementů `label` na hodnotu `blue`.

Selektor ID

Selektor typu ID obsahují všechny definicice CSS, které lze popsat zápisem

```
element#id { ... }
```

nebo

```
#id { ... }
```

V něm může být text `element` nahrazen libovolným jménem elementu a text `it` může být nahrazen libovolným atributem `id`.

Příklad 4.

Pokud bude CSS obsahovat definici `label#l1{color:blue}`, pak se nastaví se na `blue` vlastnost `color` všech elementů `label`, jejichž atribut `id` má hodnotu `l1`.

V případě `#it{color:blue}` se nastaví na `blue` vlastnost `color` všech elementů (tedy i jiných než `label`), jejichž hodnota atributu `id` je `l1`.

Selektor Class

Selektory typu Class obsahují všechny definicice CSS, které lze popsat zápisem

```
element.class { ... }
```

nebo

```
.class { ... }
```

V něm může být text `element` nahrazen libovolným jménem elementu a text `class` může být nahrazen libovolným atributem `class`.

Příklad 5.

Pokud bude CSS obsahovat definici `label.important{color:blue}`, pak se nastaví na `blue` vlastnost `color` všech elementů `label`, jejichž atribut `class` má hodnotu `important`.

V případě `.important{color:blue}` se nastaví na `blue` vlastnost `color` všech elementů (tedy i jiných než `label`), jejichž hodnota atributu `class` je `important`.

Selektor Parent element

Selektor typu Parent element obsahují všechny definicice CSS, které lze popsat zápisem

```
parent > element { ... }
```

V něm mohou být texty `element` a `parent` nahrazeny libovolnými jmény elementů z XML.

Příklad 6.

Pokud bude CSS obsahovat definici `item > label{color:blue}`, pak se nastaví se na `blue` vlastnost `color` všech elementů `label`, které mají rodičovský element `item` (tj. všechny elementy `item/label`).

5.2 Vlastnosti CSS jednotlivých elementů

V následujících kapitolách budou popsány vlastnosti elementů, které byly uvedeny v kapitole 4.

5.2.1 Vlastnosti elementu taskset

Vlastnosti tohoto elementu blíže specifikují vzhled ganttova diagramu. Jeho defaultní definice CSS je:

```
taskset{
    visible : yes;
    xunit : 5;
    yunit : 8;
    xover : 1;
    yover : 0.33;
    xmarks-small : 1;
    xmarks-big : 5;
    vspaces : 0;
    xlabels-scale : 1
}
```

Vlastnost visible

Přípustné hodnoty této vlastnosti jsou `yes` nebo `no`. Pokud bude mít `visible` hodnotu `no`, pak se obsah elementu `taskset` přeskočí a nebude zpracován. To znamená, že pro něj Psgantt nevytvoří obrázek ani zdrojový soubor pro METAPOST.

Vlastnosti xunit a yunit

Hodnoty `xunit`, respektive `yunit`, udávají rozměry jednotek x-ové, respektive y-ové osy. Jejich význam je tedy stejný, jako význam atributů `unit` elementů `axisx` a `axisy`. Ty byly popsány v sekcích 4.2.21 a 4.2.25. Defaultními hodnotami jsou 5 a 8 milimetrů.

Vlastnosti xover a yover

Hodnoty `xover`, respektive `yover`, udávají rozměry, o které bude přesahovat x-ová, respektive y-ová osa, nad posledním horizontálním, resp. vertikálním prvkem. Význam vlastností `xover` a `yover` je stejný, jako význam atributů `prolongation` elementů `axisx` a `axisy` (viz sekce 4.2.21 a 4.2.25). Defaultními hodnotami jsou 1 a 0,33.

Vlastnost xmarks-small

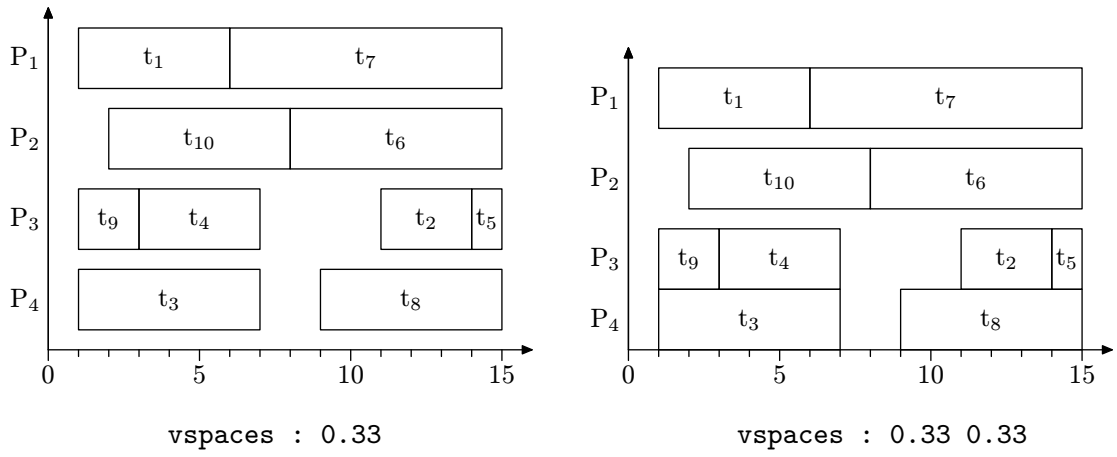
Hodnota `xmarks-small` udává četnost menších čárek (ticků) na x-ové ose. Má tedy stejný význam jako hodnota elementu `ticks`, viz sekce 4.2.22. Defaultní hodnota je 1.

Vlastnost xmarks-big

Hodnota `xmarks-big` udává četnost větších čárek na x-ové ose a jim přislušných popisků. Hodnota nastavená v `xmarks-big` se použije jen v tom případě, pokud ve vstupním XML souboru nebude uveden element `ticks` (viz sekce 4.2.22), který hodnotu `xmarks-big` předefinuje. Její defaultní hodnota je 5.

Vlastnost vspaces

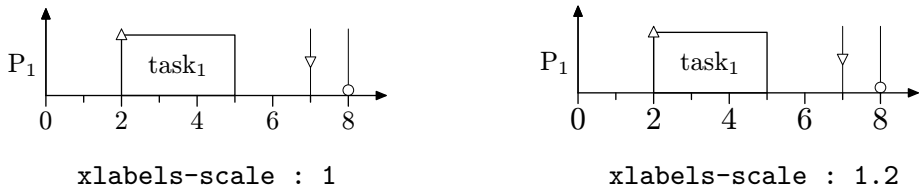
Hodnoty vlastnosti `vspaces` udávají velikosti vertikálních mezer pod každým procesorem. Pokud bude její hodnota jediné číslo, bude nastavena pro všechny mezery. Pokud bude jejich hodnot více, budou podle nich nastaveny jen jim příslušné mezery. Defaultní hodnotou je 0. Chování vlastnosti `vspaces` ilustruje obrázek 5.3.



Obrázek 5.3: Vlastnost vspaces

Vlastnost xlabels-scale

Hodnota této vlastnosti specifikuje zvětšení popisků x-ové osy. Přípustné hodnoty jsou nezáporná čísla. Defaultní hodnotou je číslo 1. Ukázka vlastnosti `xlabels-scale` je uvedena na obrázku 5.4.



Obrázek 5.4: Vlastnost xlabels-scale

5.2.2 Vlastnosti elementu task

Vlastnosti tohoto elementu specifikují viditelnost `visible` a barvu `taskcolor` elementu `task`.

Vlastnost visible

Možné hodnoty pro `visible` jsou `yes` a `no`. V případě hodnoty `no` nebude `task` vykreslen. Také nebude vykreslen žádný z elementů `releasetime`, `deadline`, `duedate`, `period`, `item` ani `item/label`. (Pokud jejich vlastnosti nebudou nastaveny jinak.)

Defaultní vlastnosti elementu `task` jsou:

```
task{
   visible : yes;
   taskcolor : white
}
```

Vlastnost taskcolor

Hodnoty `taskcolor` mohou být stejné jako hodnoty popsané v bodu 4 u elementu `color` (viz sekce 4.2.16). Pro hodnoty barev jsou navíc ve specifikaci CSS definovány další způsoby zápisu. Ty mohou být ve tvaru `rgb(`r_1`,`g_1`,`b_1`)` a `rgb(`r_2`%,`g_2`%,`b_2`%)` V prvním případě musí hodnoty r_1, g_1, b_1 ležet v intervalu $\langle 0; 255 \rangle$. V druhém případě musí hodnoty r_2, g_2, b_2 ležet v intervalu $\langle 0; 100 \rangle$. Psgantt navíc pro barvy definuje další dvě hodnoty, kterými jsou `taskcolor` a `none`.

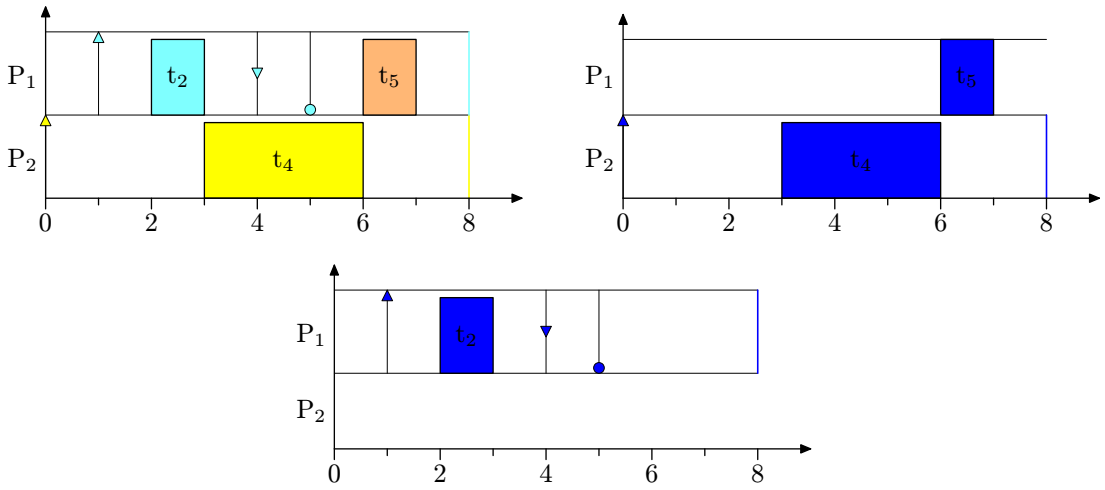
První z nich zastupuje barvu, která je přiřazena aktuálnímu tasku. Proto následující deklarace vlastnosti `taskcolor:taskcolor` u elementu `task` nemá smysl. (Jednalo by se vlastně o nekonečnou rekurzi. V tomto případě bude tasku nastavena bílá barva). Tato hodnota je využívána elementy, kteří jsou potomky elementu `task`.

Druhá hodnota `none` deklaruje průhlednost barvy. Ta udává, že se plocha (např. boxu, deadline, releasetimu, atd.) nevyplní žádnou barvou. Bude pouze vykreslena její kontura. (U vlastností barvy čar, a jim podobných, bude hodnota `none` interpretována jako černá barva.)

Příklad 7.

Vezměme si ganttův diagram z obrázku 5.5 vlevo. Pro něj řešme následující zadání: Chceme, aby tasky t_4 a t_5 měly modrou barvu a aby task t_2 z obrázku zmizel. Pokud víme, že má task t_2 hodnotu atributu `id` rovnu `t2`, můžeme zadání vyřešit pomocí následujícího CSS. Výsledek je zobrazen v obrázku 5.5 vpravo.

```
task#t2 { visible: no }
task { taskcolor: blue }
```



Obrázek 5.5: Diagramy k příkladům 7 a 8

Všimněme si, že z levého obrázku kromě boxu úlohy t_2, zmizely také příslušné značky k elementům `releasetime`, `deadline`, `duedate` a `period`. Ty podle mechanizmu popsaného v kapitole 5.1.4 od elementu `task` zdědí vlastnost `visible` s hodnotou `no`.

Příklad 8.

Vezměme si stejný obrázek jako v předchozím příkladu. Tentokrát budeme chtít, aby měl naopak task t_2 modrou barvu, a aby ostatní tasky zmizely. Řešením může být následující CSS. Výsledek je zobrazen na obrázku 5.5 uprostřed.

```
task#t2 { visible: yes; taskcolor : blue }
task { visible : no }
```

Aplikací stejného postupu na velmi rozsáhlé ganttovy diagramy, můžeme velmi rychle zobrazit pouze ty tasky, které nás skutečně zajímají. Méně důležité tasky skryjeme. V řešení úloh podobného typu spočívá hlavní přednost kaskádových stylů. Díky dědění vlastností, je možné ušetřit spoustu psaní a úprav, které bychom jinak museli udělat.

Od tohoto odstavce dále budou popisovány pouze vlastnosti takových elementů, se kterými je v ganttově diagramu spojen konkrétní grafický objekt. Tím se vyhneme popisu vlastností takových elementů, které z pohledu kaskádových stylů slouží pouze k tomu, aby od nich mohly jejich

synovské elementy tyto vlastnosti dědit. Těmito „ryze dědícími“ elementy jsou `task/schedule`, `precedenceconstrains`, `taskset/schedule`, `gantt`, `axisx`, `axisy` a `labels`.

5.2.3 Vlastnosti elementu releasetime

Elementu `releasetime` je pomocí kaskádových stylů přiděleno celkem deset vlastností. Jejich defaultní hodnoty jsou uvedeny v následující definici CSS. Značka odpovídající elementu `releasetime` se obecně skládá ze dvou částí: symbolu a svislé čáry.

```
releasetime{
  visible : yes;
  height : 1.1;
  color: black;
  mark-style-type: arrow-up;
  mark-style-color: black;
  mark-style-size: 4;
  mark-style-position: 100%;
  mark-style-background-color: taskcolor;
  pen-width : 0.25;
  ignore-margin : no
}
```

Vlastnost visible

Hodnota vlastnosti `visible` v případě elementu `releasetime` nastavuje, zda bude značka viditelná (hodnota `yes`), či nikoliv (hodnota `no`). Pokud se u ostatních elementů význam této vlastnosti nezmění, nebudeme ji už dále uvádět.

Vlastnost height

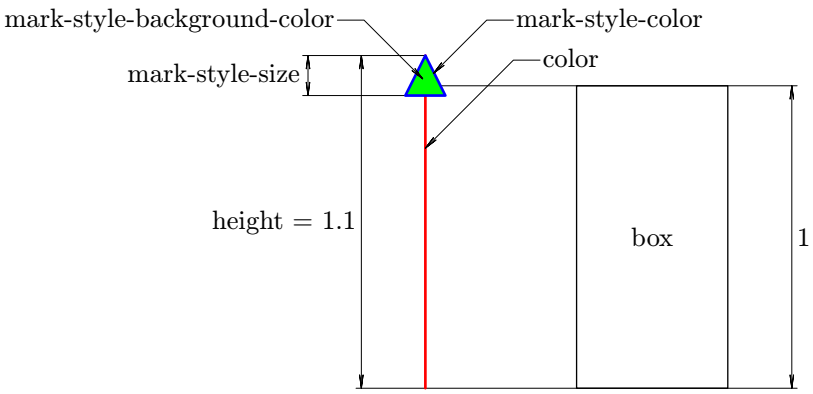
Hodnota vlastnosti `height` udává celkovou výšku značky, která je vztažená k výšce boxu (viz obrázek 5.6). Hodnota musí být nezáporné číslo. Pokud bude mít `height` hodnotu 0, pak značka nebude vůbec vykreslena. Defaultní hodnota je 1,1. To znamená, že výška značky bude v tomto případě o 10% větší než je výška boxu.

Vlastnosti color, mark-style-color, mark-style-background-color

Význam těchto vlastností je patrný z obrázku 5.6. Jejich defaultní hodnoty jsou postupně `black`, `black` a `taskcolor`. (V obrázku 5.6 jim jsou přiřazeny hodnoty `red`, `blue` a `green`.) Přípustné hodnoty jsou stejné jako u vlastnosti `taskcolor` elementu `task`. Pokud bude mít vlastnost `mark-style-background-color` hodnotu `none`, nebude symbol vyplněn žádnou barvou. (Vykreslí se pouze jeho kontura). Navíc bude svislá čára vhodným způsobem rozdělena tak, aby její části nezasahovaly dovnitř symbolu.

Vlastnost pen-width

Hodnota vlastnosti `pen-width` nastavuje šířku pera, kterou jsou kresleny všechny části značky (tj. symbol i svislá čára). Hodnota je uvedena v jednotkách bp. Defaultní hodnotou je 0,25 bp. (Na obrázku je vlastnosti `pen-width` přiřazena hodnota 1 bp.)



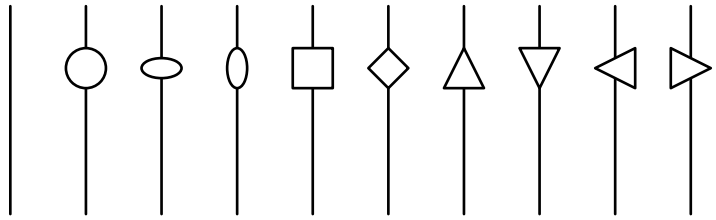
Obrázek 5.6: Značka elementu releasetime

Vlastnost mark-style-size

Tato vlastnost udává velikost symbolu. (Na obrázku 5.6 výšku a šířku trojúhelníka). Její hodnota se udává v jednotkách bp. Defaultní hodnotou je 4 bp.

Vlastnost mark-style-type

Hodnoty této vlastnosti specifikují tvar symbolu. V případě elementu `releasetime` je jím defaultně trojúhelník otočený nahoru (`arrow-up`). Seznam možných hodnot je `none`, `circle`, `xellipse`, `yellipse`, `square`, `diamond`, `arrow-up`, `arrow-down`, `arrow-left` a `arrow-right`, viz obrázek 5.7. V případě hodnoty `none` není vykreslen žádný symbol, ale pouze svislá čára.

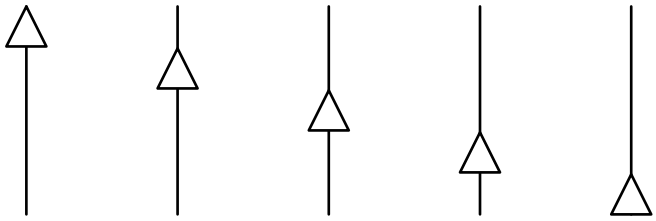


Obrázek 5.7: Symboly značek

Vlastnost mark-style-position

Hodnota vlastnosti `mark-style-position` specifikuje polohu symbolu vůči hodnotě `height`. (Pokud bude její hodnota například `50%`, pak bude symbol značky umístěn přesně uprostřed značky, jejíž výška je zadaná pomocí `height`.) Přípustné hodnoty musí ležet v intervalu ⟨`0%`;`100%`⟩.

Defaultní hodnota `mark-style-position` je `100%`. Na obrázku 5.8 je zobrazen symbol `arrow-up` postupně s hodnotami `100%`, `75%`, `50%`, `25%` a `0%`.

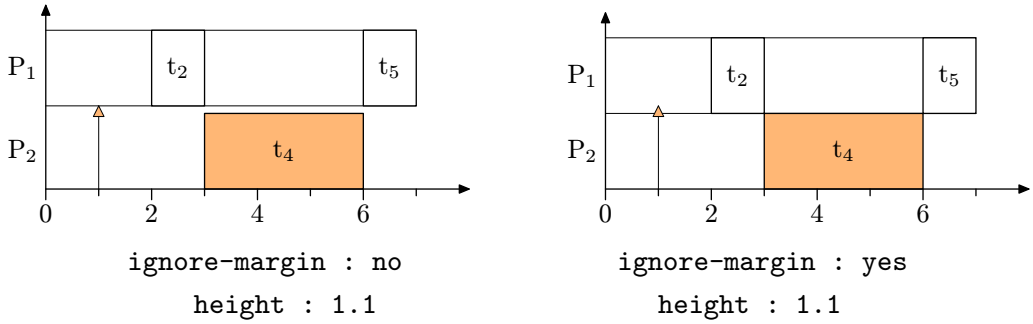


Obrázek 5.8: Pozice symbolů značek

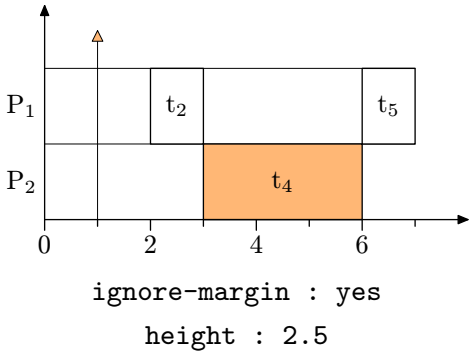
Vlastnost ignore-margin

Poslední vlastností je `ignore-margin`. Ta udává, jestli bude výška značky započítána do výšky příslušné k danému procesoru (hodnota `no`), nebo nikoliv (hodnota `yes`). Význam vlastnosti `ignore-margin` ilustruje obrázek 5.9, který je pro názornost doplněn gridem na y-ové ose. Defaultní hodnotou `ignore-margin` je `no`.

Význam této property oceníme zejména tehdy, když budeme chtít symbol některého elementu `releasetime` pro větší přehlednost posunout směrem nahoru, například nad nejvýše umístěný procesor. To ilustruje obrázek 5.10.



Obrázek 5.9: Vlastnost `ignore-margin`



Obrázek 5.10: Příklad posunutí symbolu nahoru

5.2.4 Vlastnosti elementu deadline

Element `deadline` má definovány stejné vlastnosti jako element `releasetime` (viz sekce 5.2.3). Význam vlastností těchto elementů je naprosto shodný, proto jej už nebudeme znovu uvádět. Jejich vlastnosti ze se navzájem liší pouze defaultně přidělenými hodnotami. Jak vypadá element `deadline` s defaultním nastavením, je vidět v obrázku 5.11. Defaultní kaskádový styl pro elementu `deadline` je:

```
deadline{
  visible : yes;
  height : 1.1;
  color: black;
  mark-style-type: circle;
  mark-style-color: black;
  mark-style-size: 4;
  mark-style-position: 0%;
  mark-style-background-color: taskcolor;
  pen-width : 0.25;
  ignore-margin : no
}
```

5.2.5 Vlastnosti elementu duedate

Pro vlastnosti elementu `duedate` platí to samé, co pro vlastnosti elementu `deadline` (viz předchozí sekce). Defaultní hodnoty vlastností (viz obrázek 5.11) jsou nastaveny v následovně:

```
duedate{
  visible : yes;
  height : 1.1;
  color: black;
  mark-style-type: arrow-down;
  mark-style-color: black;
  mark-style-size: 4;
  mark-style-position: 50%;
  mark-style-background-color: taskcolor;
  pen-width : 0.25;
  ignore-margin : no
}
```

5.2.6 Vlastnosti elementu period

Element `period` má definovány vlastnosti, které jsou uvedeny v následujícím kaskádovém stylu. Příslušný grafický objekt k elementu `period` je zobrazen například na obrázku 5.11. Tvoří jej

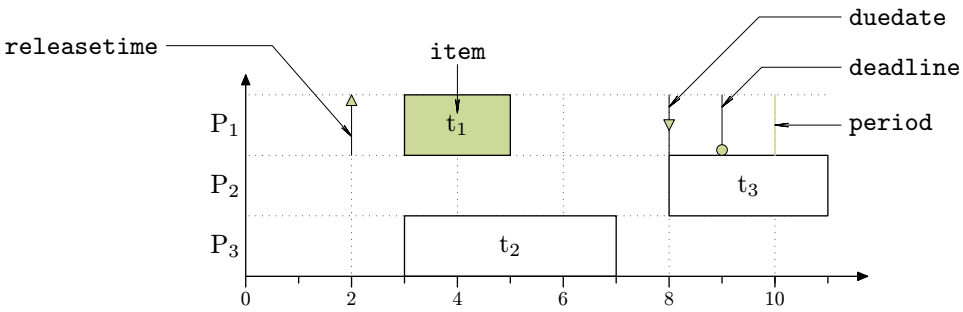
pouze svislá čára, která se v diagramu periodicky opakuje. Její vlastnosti jsou stejné jako vlastnosti svislé čáry například u elementu `releasetime`. Ty byly popsány v kapitole 5.2.3. Proto se jimi znovu zabývat nebudeme.

```
period{
  visible : yes;
  color : taskcolor;
  ignore-margin : yes;
  height : 1.1;
  ignore-margin : yes;
  pen-width : 0.6
}
```

5.2.7 Vlastnosti elementu item

Vlastnosti elementu `item` definují vzhled boxu, který reprezentuje danou úlohu, nebo její část (viz obrázek 5.11). Vlastnost `color` udává barvu boxu, vlastnost `visible` jeho viditelnost. Defaultní hodnoty vlastnosti jsou tyto:

```
item{
  visible : yes;
  color : taskcolor
}
```



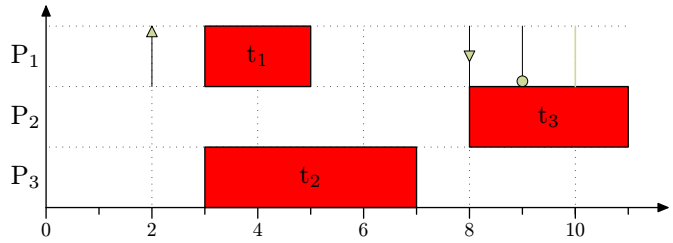
Obrázek 5.11: Diagram s defaultním nastavením elementů tasku

Příklad 9.

Vezměme si předchozí obrázek 5.11. V něm budeme nyní chtít změnit barvy všech boxů na červenou barvu. To je možné, za pomoci následujícího pravidla CSS:

```
item{
  color : red
}
```

Výsledný obrázek je zobrazen na obrázku 5.12. V něm si všimněme, že se změnily pouze barvy elementů `item`, které jsou zobrazeny pomocí boxů. Nezměnily se však barvy elementů `releasetime`, `deadline`, `duedate` a `period`, které zůstaly zachovány.



Obrázek 5.12: Příklad nastavení barvy boxů

5.2.8 Vlastnosti elementu item/label

Tento element obsahuje popisek, který se zobrazí uvnitř boxu (viz obrázek 5.11). Vlastnosti tohoto elementu a jejich defaultní nastavení jsou dány tímto pravidlem CSS:

```
label{
  visible : yes;
  format : verbatim;
  color : black;
  top : 0.75;
  left : 0;
  shift-enabled : yes;
  scale : yes;
}
```

Vlastnost format

Vlastnost `format` udává způsob vysázení popisku. Jeho povolenými hodnotami jsou `verbatim` a `tex`. Pro vlastnost format platí stejná pravidla, jako pro atribut `format` elementu `item/label`, o kterém jsme mluvili v kapitole 4.2.14.

Vlastnost shift-enabled

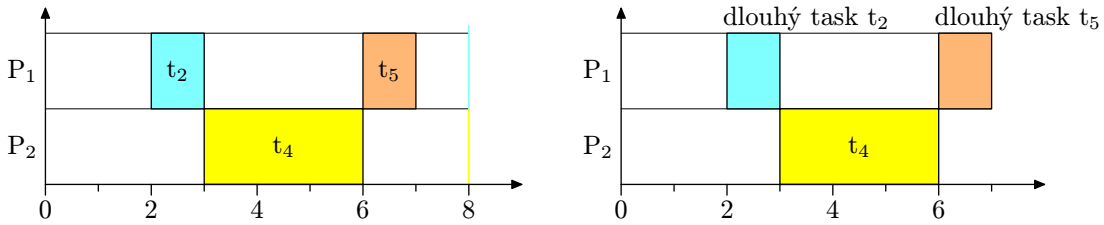
Tato vlastnost povoluje (hodnota `yes`), respektive nepovoluje (hodnota `no`) posunutí popisku v případě, kdy se popisek nevejde do boxu. Podrobněji bude tato vlastnost popsána v příští sekci.

Vlastnosti top a left

Vlastnosti `top` a `left` udávají, kam se má popisek posunout, pokud se jeho šířka nevejde do daného boxu a pokud bude zároveň povoleno jeho posunutí vlastností `shift-enabled`. (Kontrolována je pouze šířka popisku, ne jeho výška.) V případě splnění obou podmínek se popisek posune o `left` jednotek doleva a `top` jednotek nahoru. Pokud se do boxu popisek vejde, pak k žádnému posunutí nedojde. V tomto případě bude popisek umístěn do středu boxu. Posunutí ilustruje následující příklad.

Příklad 10

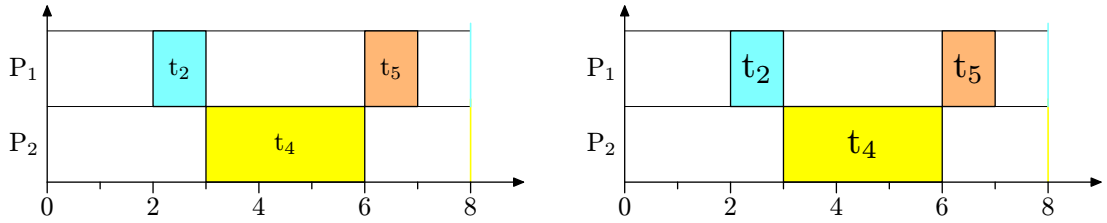
Podívejme se na obrázek 5.13 vlevo. V něm je vidět, že se všechny popisky vešly do boxů. Změňme nyní popisky u tasku t_2 a t_5 tak, aby se do svých boxů nevešly. Současně nastavme hodnoty posunutí `left` na `-1` a `top` na `0.7`. Na obrázku 5.13 vpravo pak můžeme vidět, že se popisky opravdu posunuly o námi nastavené hodnoty.



Obrázek 5.13: Posunutí popisků

Vlastnost scale

Tato vlastnost udává zvětšení (hodnota je větší než jedna), respektive zmenšení (hodnota je menší než jedna) popisku. Její hodnota musí být kladné nenulové číslo.



Obrázek 5.14: Změna velikosti popisků

Příklad 11

Vezměme si obrázek 5.14 vlevo. V něm jsou popisky jednotlivých boxů vysázeny defaultní velikostí písma, jejíž velikost je rovna 10 pt, to odpovídá jednotkové velikosti `scale`. (pt je takzvaný tiskařský bod. Jeho délka je jen o něco málo menší než je velikost 1 bp. Jeho přesný rozměr je 1/72,27 palce, což je přibližně 0,351 milimetru.) Nastavme nyní `scale` na hodnotu `1.4`. Tímto bychom měli dostat popisky vysázené velikostí 14 pt. Výsledný obrázek je vidět v 5.14 vpravo.

5.2.9 Vlastnosti elementu labels/label

Vlastnosti elementu `labels/label` jsou stejné, jako vlastnosti elementu `item/label`, který byl popsaný dříve, v kapitole 5.2.8. Jediným rozdílem je, že v elementu `labels/label` jsou ignorovány vlastnosti `top`, `left` a `shift-enabled`.

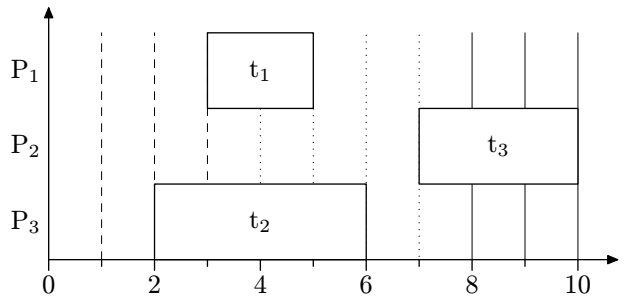
5.2.10 Vlastnosti elementu axisx/grid

Element `axisx/grid` vykreslí v ganttově diagramu jednu, nebo více horizontálních linek. Defaultní vlastnosti těchto linek jsou definovány následujícím pravidlem CSS:

```
grid{
  visible : yes;
  color : black;
  line-style: solid;
  margin-top : 0;
  margin-bottom : 0;
  pen-width : 0.25
}
```

Vlastnost line-style

Tato vlastnost udává styl linek gridu. Jejími přípustnými hodnotami jsou `dashed`, `dotted` a `solid`. Ty postupně znamenají, že bude grid vykreslen čárkovanou, tečkovanou nebo plnou čarou. Defaultně je nastavena hodnota `solid`. Všechny tři způsoby uvádí obrázek 5.15.



Obrázek 5.15: Styly čar gridů

Pro zajímavost ještě uveďme, jaké nastavení elementu `axisx` obrázku 5.15 odpovídá.

```
<axisx>
  <grid from="1" to="3" style="line-style:dashed"/>
  <grid from="4" to="7" style="line-style:dotted; pen-width:0.5"/>
  <grid from="8" to="10"/>
</axisx>
```

Vlastnosti `margin-top` a `margin-bottom` mají uplatnění pouze u gridů na y-ové ose. Popis zbylých vlastností byl uveden v dřívějších kapitolách (např. `pen-width` v kapitole 5.2.3).

5.2.11 Vlastnosti elementu axisy/grid

Vlastnosti tohoto elementu jsou stejné jako vlastnosti elementu `asisx/grid` (viz kapitola 5.2.10). Jediný rozdíl představují vlastnosti `margin-top` a `margin-bottom`, které si nyní vysvětlíme. Ještě předtím, si zopakujme základní informace o gridech, které byly uvedeny v kapitole 4.2.27.

Nejvyšší grid v diagramu je umístěn nad nejvyšším procesorem. Pořadové číslo tohoto gridu je 0. Grid s pořadovým číslem 1 je umístěn pod tento procesor, atd. Až nakonec nejníže umístěný grid bude mít pořadové číslo n, kde n je celkový počet procesorů. Tento grid bude umístěn pod prvním procesorem.

Vlastnost margin-top

Tato vlastnost udává, velikost vertikální mezery, která bude nad linkou a nejbližším vyšším procesorem. Umístění `margin-top` je zobrazeno na obrázku 5.16.

Vlastnost margin-bottom

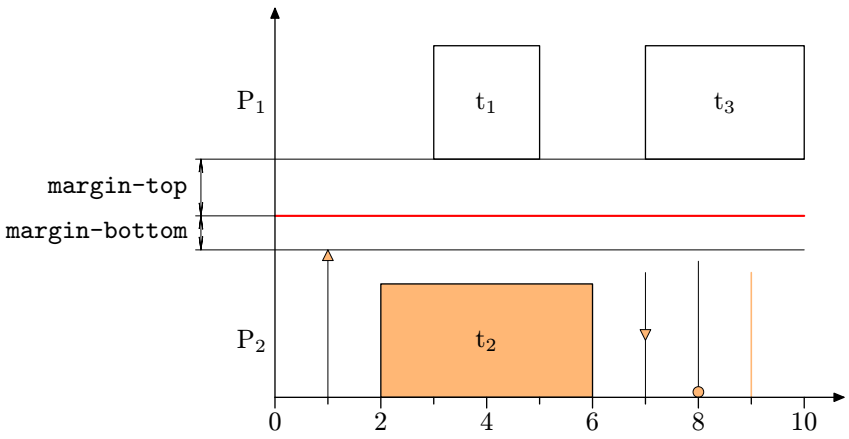
Vlastnost `margin-bottom` udává, velikost vertikální mezery, která bude mezi horizontální linkou a nejvyšším prvkem procesoru, který je umístěn nejblíž pod linkou. Umístění `margin-bottom` je vidět z obrázku 5.16.

Dále pro vykreslování gridu platí následující pravidla:

1. Pokud bude mít grid pořadové číslo n (tj. nejnižší grid) a zároveň bude nulová jeho hodnota `margin-bottom`, pak tento grid nebude vykreslen. Musel by totiž překrýt x-ovou osu, což v našem případě není vhodné. Pokud bude mít tento grid nenulovou hodnotu `margin-top`, bude nad osu x pouze přidána mezera o velikosti `margin-top`.

2. V případě, kdy bude mít grid pořadové číslo 0 (tj. nejvyšší grid), bude ignorovaná jeho hodnota `margin-top`.

3. Pokud bude v elementu `axisx` existovat nějaký element `grid`, bude ignorováno nastavení vertikálních mezer podle vlastnosti `vspaces` elementu `taskset`. Není totiž možné nastavit mezeru mezi sousedními procesory zároveň dvěma různými způsoby.



Obrázek 5.16: Gridy y-ové osy

5.2.12 Vlastnosti elementu precedence

Tento element reprezentuje jediné precedenční omezení ganttova diagramu. Precedenční omezení byly postupně popsány v kapitolách 2 a 4.2.18. Defaultní definice vlastností elementu `precedence` udává následující kód:

```
precedence{
  visible : yes;
  color : black;
  pen-width : 0.25;
  mark-style-size : 3;
  from-x-over : 6;
  to-x-over : 6;
  from-y-shift : 0;
  to-y-shift : 0;
  path-text : '{right}...{right}';
  weight-text-enabled : yes;
  weight-text-position : top;
  weight-text-offset : 2;
  weight-text-scale : 0.6;
  weight-mark-size : 3;
  weight-mark-time : 0.5
}
```

Vlastnost mark-style-size

Hodnota vlastnosti `mark-style-size` udává délku šipky. Tím je myšlena velikost symbolu u koncového bodu precedence. Délka je udávána v jednotkách bp. (Viz obrázek 5.17)

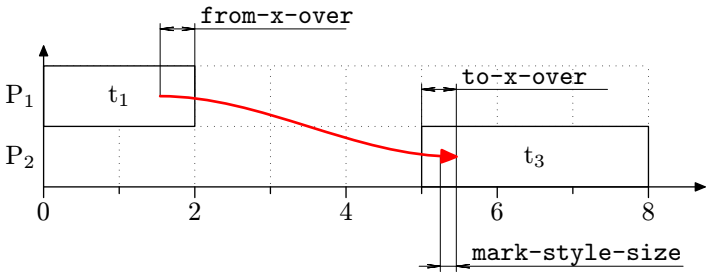
Vlastnost from-x-over

Hodnota vlastnosti `from-x-over` udává vzdálenost, o kterou je počáteční bod precedence posunutý směrem dovnitř boxu, ze kterého vede. Hodnoty `from-x-over` jsou udávány v jednotkách bp. Minimální hodnotou je číslo 0. Maximální hodnota je limitována násobkem 0,4 šířky počátečního boxu.

Vlastnost to-x-over

Hodnota vlastnosti `to-x-over` udává vzdálenost, o kterou je koncový bod precedence posunutý směrem dovnitř boxu, do kterého vede. Hodnoty `to-x-over` jsou udávány v jednotkách bp. Minimální hodnotou je číslo 0. Maximální hodnota je limitována násobkem 0,4 šířky koncového boxu.

Vlastnosti `mark-style-size`, `from-x-over` a `to-x-over` jsou zobrazeny na obrázku 5.17.



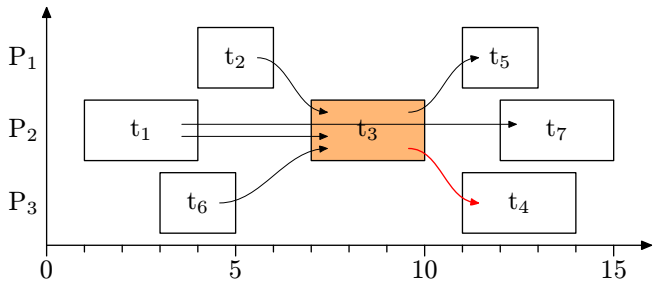
Obrázek 5.17: Vlastnosti precedencí

Vlastnost from-y-shift

Pomocí hodnoty `from-y-shift` můžeme posunovat počáteční bod precedence ve vertikálním směru. Jednotka, ve které se hodnota `from-y-shift` udává, závisí na tom, kolik precedencí má procesor, k němuž přísluší její počáteční bod. Počet precedencí tohoto procesoru zjistíme sečtením všech vertikálních úrovní nebo také hladin, ve kterých je alespoň jedna precedence. Jednotkou pak je výška boxu vydělená počtem těchto úrovní zvýšeným o jedničku.

Příklad 12.

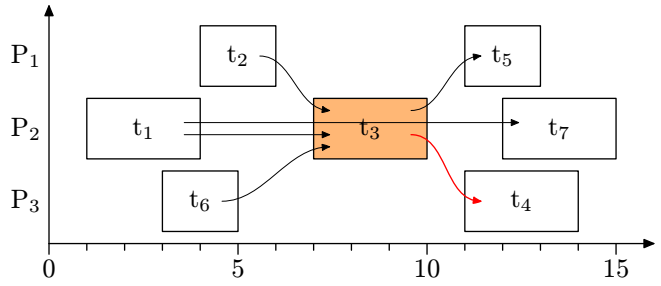
Vezměme si diagram z obrázku 5.18. V něm budeme chtít posunout počáteční bod vyznačené precedence nahoru. V obrázku vidíme, že její počáteční bod přísluší k procesoru s číslem 2. Ten je svými čtyřmi precedencemi rozdělen na 5 dílů. Jednotka posunutí má velikost právě jednoho dílu, to je 1/5 výšky procesoru. Pokud nastavíme hodnotu `from-y-shift` na hodnotu -1, posune se počáteční bod precedence směrem nahoru. Kladné hodnoty `from-y-shift` by precedenci posunuly směrem dolů. Směr posunutí tedy odpovídá číslovaní procesorů. S rostoucí hodnotou bude blíž k ose x, s klesající od ní bude víc vzdálen. Výsledek posunu znázorňuje obrázek 5.19.



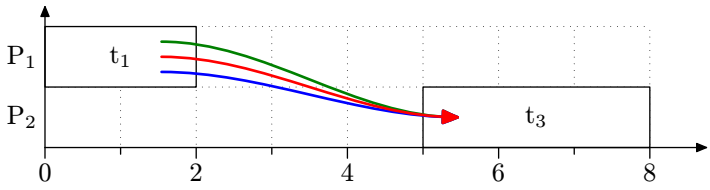
Obrázek 5.18: Vertikální posunutí počátečního bodu precedence – zadání

Příklad 13.

Ukažme si posunutí ještě na jednom příkladu, ten je vidět na obrázku 5.20. V něm má červenou barvu původní precedence. Zelená precedence vznikla posunutím počátečního bodu červené precedence o $-0{,}5$ jednotek nahoru. Modrá posunutím jejího počátečního bodu o 0,5 jednotek dolů.



Obrázek 5.19: Vertikální posunutí počátečního bodu precedence – výsledek



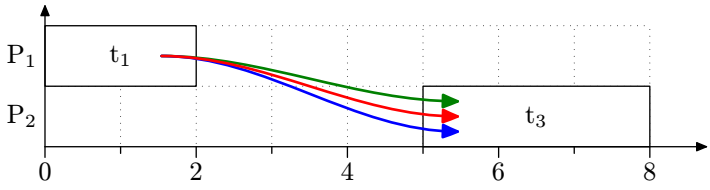
Obrázek 5.20: Další příklad posunutí

Vlastnost to-y-shift

Tato vlastnost je analogická k vlastnosti `from-y-shift`. Narozdíl od ní `to-y-shift` vertikálně posunuje koncový bod precedence. Způsob výpočtu jednotky posuvu je analogický jako v případě `from-y-shift`.

Příklad 14.

Pro hodnoty `to-y-shift` rovny $-0{,}5$ a $0{,}5$ jsou polohy precedencí znázorněny na obrázku 5.21. Precedenci s hodnotou $-0{,}5$ je nastavena zelená barva, precedenci s hodnotou $0{,}5$ barva modrá.



Obrázek 5.21: Vertikální posunutí koncového bodu precedence

Vlastnost path-text

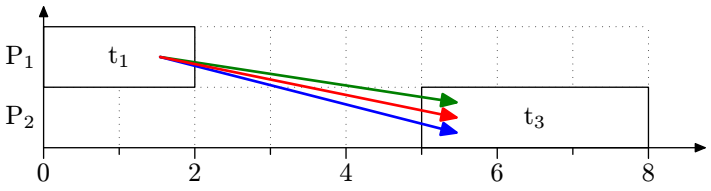
Hodnotou této vlastnosti je řetězec, který specifikuje tvar křivky precedence. Přesná syntaxe křivky je popsaná v uživatelském manuálu METAPOST [13]. Zde se spokojíme s vysvětlením její defaultní hodnoty, která je `'{right}...{right}'`.

Řetězec `{right}` udává, že křivka povede z počátečního bodu křivky ve směru doprava. (Její tečný vektor v tomto bodě bude rovnoběžný osou x). Tři tečky `...` udávají, že se bude jednat o Bézierovu kubiku s maximální tenzí (viz [13]). Poslední část `{right}` udává, jaký směr bude mít křivka ve svém koncovém bodě.

Změnou hodnoty `path-text` je možné specifikovat libovolný průběh křivky precedence mezi jejím počátečním a koncovým bodem. Syntaxe `path-text` musí odpovídat syntaxi METAPOST-ového typu path (viz [13]).

Příklad 14.

Pokud budeme chtít změnit tvar precedence z křivky na přímku, pak stačí nastavit `path-text` na hodnotu `'--'`. Změnu tvaru precedencí (z obrázku 5.21) z křivek na přímky ilustruje obrázek 5.22.



Obrázek 5.22: Změna tvaru precedence

Vlastnost weight-text-enabled

Vlastnost `weight-text-enabled` povoluje (hodnota `yes`), respektive zakazuje (hodnota `no`) zobrazení čísla s váhou precedence. V případě povolení budou vykresleny pouze precedence s váhou, jejíž hodnota je různá od jedné.

Vlastnost weight-mark-time

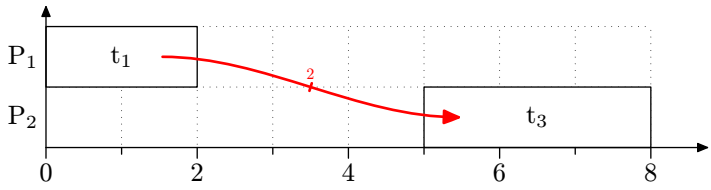
Hodnota `weight-mark-time` udává takzvaný čas METAPOSTové proměné path (viz [13]). Dříve než si vysvětlíme k čemu `weight-mark-time` vlastně slouží, popišme, co tímto časem myslíme.

Zjednodušeně řečeno je čas číslo, které může nabývat hodnot 0 až $n-1$, kde n je počet bodů, kterými je definovaný průběh cesty. Defaultně je cesta precedence složena ze dvou bodů, což znamená, že hodnoty `text-mark-time` musí v tomto případě ležet v intervalu $\langle 0; 1\rangle$. Pokud by byl počet bodů například 3, pak to byl interval $\langle 0; 2\rangle$, atd. (Čas je v nauce o křivkách nazýván parametrem křivky. Takové křivky jsou pak známy jako křivky parametrické.)

Hodnota `weight-mark-time` udává čas, pomocí kterého je definován bod na křivce precedence, ve kterému bude umístěn popisek s číslem váhy. Předpokládejme defaultní nastavení precedence se dvěmi body. V tomto případě hodnota 0 udává, že bude popisek s váhou umístěn v jejím počátečním bodě, hodnota 1, že bude umístěn v bodě koncovém. Hodnoty mezi 0 a 1 určují body křivky mezi jejím počátečním a koncovým bodem.

Příklad 14.

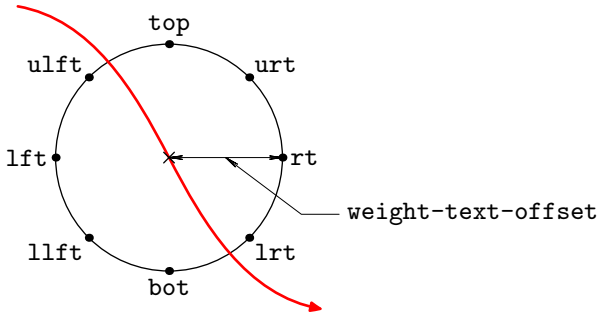
Předpokládejme, že má precedence váhu 2. Pro defaultní nastavení precedence (tj. hodnota `weight-mark-time` je 0,5) bude popisek s váhou hrany umistěn k bodu, který leží na křivce uprostřed mezi jejím počátečním a koncovým bodem. To je vidět na obrázku 5.23.



Obrázek 5.23: Poloha popisku s váhou precedence

Vlastnost weight-text-position

Tato vlastnost specifikuje, kde bude popisek s váhou hrany umístěn vzhledem k bodu definovaném pomocí `weight-mark-time`. Přípustnými hodnotami jsou `lft`, `rt`, `top`, `bot`, dále pak `llft`, `lrt`, `ulft` a `urt`. Defaultní hodnotou je `top`. Význam jednotlivých vlastností vystihuje obrázek 5.24. Bod precedence definovaný pomocí `weight-mark-time` je v něm označen křížkem.



Obrázek 5.24: Možnosti umístění popisku s váhou precedence

Vlastnost weight-text-offset

Hodnota `weight-text-offset` udává velikost posunutí popisku s váhou precedence vůči bodu definovaném pomocí `weight-text-time`. Je udávána v jednotkách bp. Defaulní velikost je 2 bp. Její význam je patrný z obrázku 5.24.

Vlastnost weight-text-scale

Tato vlastnost udává velikost zvětšení (hodnota větší než 1), respektive zmenšení (hodnota menší než 1) popisku s váhou hrany. Hodnota `weight-text-scale` musí být větší než nula. Defautlně je nastavena na 0,6.

Vlastnost weight-mark-size

Pomocí hodnoty `weight-mark-size` se nastavuje velikost čárky, která se vykreslí kolmo k precedenci v bodě, který je určený vlastností `weight-mark-time`. Hodnota je vyjádřena v jednotkách bp. Defaultní hodnotou `weight-mark-size` je 0,5.

Vlastnost color

Hodnota `color` udává, jakou barvou budou vykresleny všechny části precedence. Jimi jsou vlastní křivka, šipka, čárka a popisek s váhou. Defaultně je jí přiřazena hodnota `black`. Možnostmi nastavení barev jsme se zabývali ve vlastnosti `taskcolor` elementu `task` (viz kapitola 5.2.2).

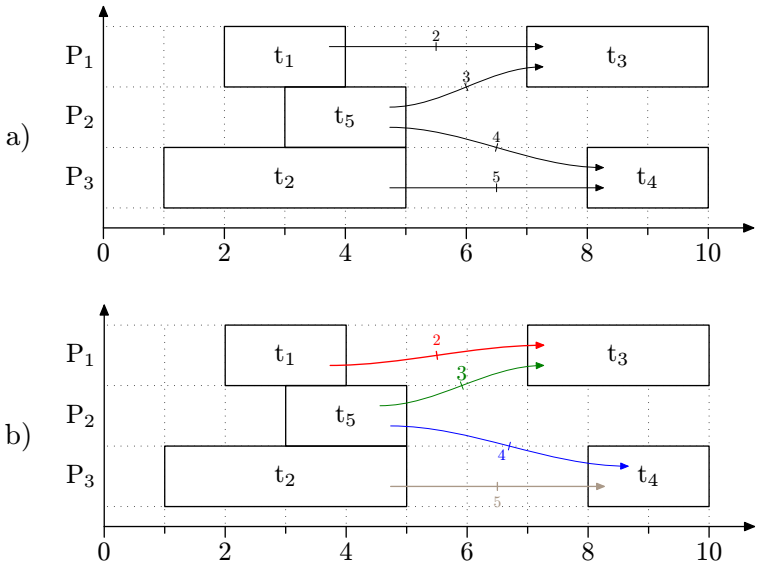
Příklad 15.

Vezněme si ganttův diagram z obrázku 5.25 a), ve kterém je použito defaultní nastavení CSS. Precedencím může ve vstupním XML souboru odpovídat kód:

```
<precedence from="t1" to="t3">2</precedence>
<precedence from="t5" to="t3">3</precedence>
<precedence from="t5" to="t4">4</precedence>
<precedence from="t2" to="t4">5</precedence>
```

Nastavme nyní jednotlivým precedencím atribut `style`. Pořadí atributů `style` odpovídá pořadím výše uvedených precedencí.

```
style="color:red; weight-text-offset:5; pen-width:0.5; from-y-shift:1"
style="color:green; weight-text-scale:0.8; from-x-over:10"
style="color:blue; weight-text-position:llft; to-x-over:15"
style="color:#a98; pen-width:0.5; weight-text-position:bot; weight-text-offset:4"
```



Obrázek 5.25: Příklad změny vlastností precedencí

6 Praktické příklady

Jelikož se v průběhu vývoje Psganttu množily dotazy, například, jak do obrázku přidat kótu, popisek s šipkou nebo legendu, a podobné, rozhodl jsem se, že několik praktických příkladů uvedu v samostatné kapitole, která bude demonstrovat možnosti programu Psgantt.

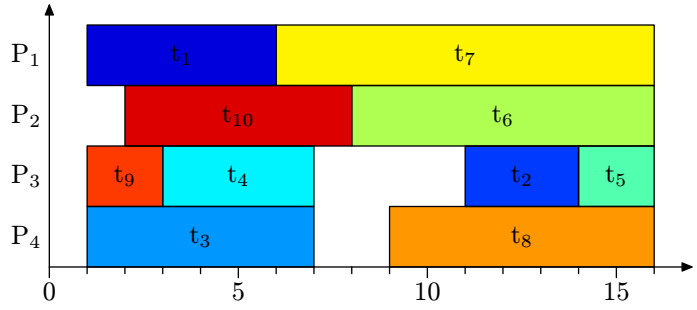
Tato kapitola obsahuje dva příklady, ze kterých je možné usoudit, jakým způsobem postupovat, když budeme do obrázku chtít vložit další grafické prvky, a při tom využít možnosti Psganttu a METAPOSTu.

Podrobnosti k příkazům METAPOSTu lze najít v kapitole 8, nebo v literatuře [13].

Příklad 1.

Mějme ganttův diagram z obrázku 6.1. (Jemu odpovídající XML soubor lze najít na přiloženém CD. Jméno souboru je `ex1_1.xml`.) Našim úkolem bude:

1. Změnit popisky na ose x tak, aby každá druhá jednotka odpovídala jednomu dni. Jména jednotlivých dnů umístit doprostřed mezi jim odpovídající čárky a dále zvýraznit modrou barvou a italikou víkendové dny So a Ne.
2. Změnit jména procesorů tak, aby místo P_1 byl Stroj A, místo P_2 byl Stroj B, atd.
3. Kolem procesorů vložit gridy a vertikální mezery o velikosti 0,3 výšky boxu. Kolem procesoru jménem Stroj B vložit navíc červený grid s tloušťkou pera 0,5 pt.
4. Jméno úlohy t_6 vysázet červeně a umístit k němu šipku s popiskem „důležitá úloha“.

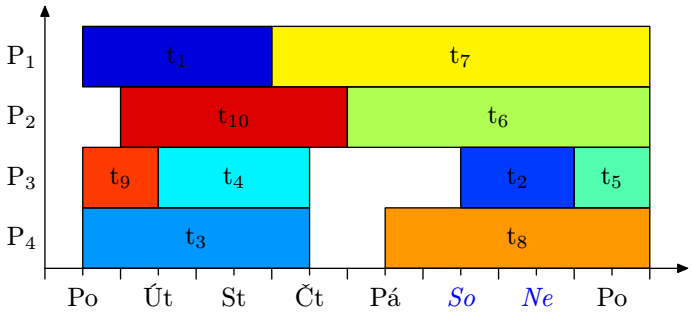


Obrázek 6.1: Výchozí diagram pro příklad 1

Řešení: Pro jednoduchost předpokládejme, že obsah elementu `matlabdata/style` je: `label{format:tex}`, tj. všechny popisky se budou sázet TEXem.

1. První bod lze vyřešit vložením elementu `axisx` s následujícím obsahem do elementu `taskset/schedule/gantt`. Výsledek vidíme na obrázku 6.2. (Jméno vstupního XML souboru je `ex1_2.xml`.)

```
<axisx>
  <ticks labeled="2">1</ticks>
  <labels between="true">
    <label>Po</label>
    <label>Út</label>
    <label>St</label>
    <label>Čt</label>
    <label>Pá</label>
    <label style="color:blue">{\it So}</label>
    <label style="color:blue">{\it Ne}</label>
  <labels/>
</axisx>
```

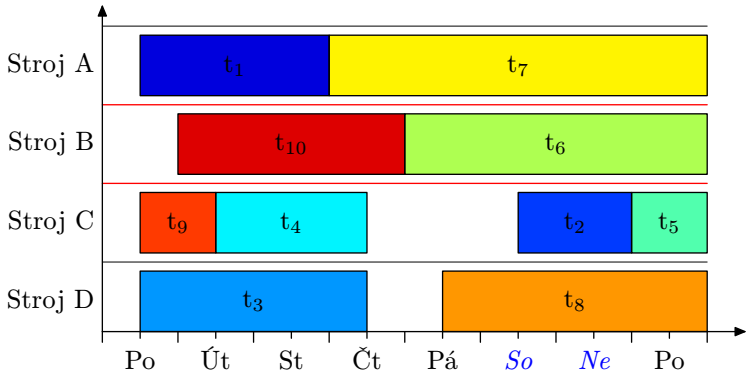


Obrázek 6.2: Diagram pro příklad 1 – bod 1

2.-3. Druhý a třetí bod lze vyřešit vložením elementu `axisy` s následujícím obsahem do elementu `taskset/schedule/gantt`. Výsledek vidíme na obrázku 6.3. (Soubor `ex1_3.xml`.)

```
<axisy style="margin-bottom:0.15; margin-top:0.15">
  <labels>
    <label>Stroj A</label>
    <label>Stroj B</label>
    <label>Stroj C</label>
    <label>Stroj D</label>
  <labels/>
  <grid from="0" to="0"/>
  <grid from="1" to="2" style="color:red; pen-width:0.5"/>
  <grid from="3" to="3"/>
</axisy>
```

4. Změny barvy popisku úlohy t_6 lze docílit vložením textu `#t6{color:red}` do elementu `matlabdata/style`. Vložení šipky s popiskem zařídí element `mpcommand` s následujícím obsahem. Výsledný diagram je vidět na obrázku 6.4.

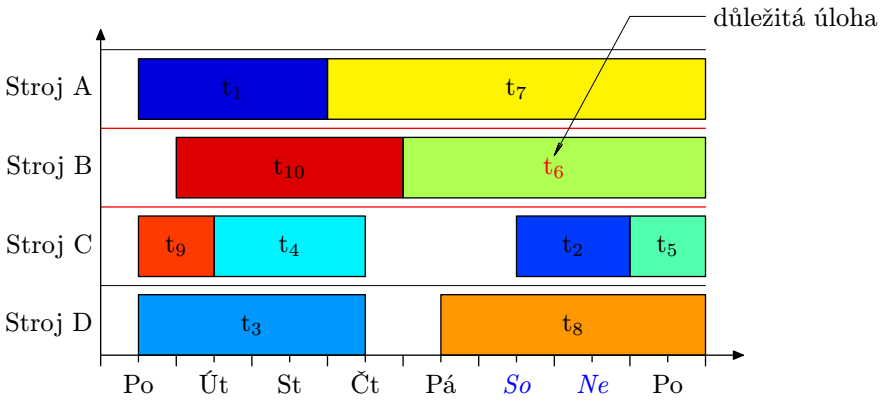


Obrázek 6.3: Diagram pro příklad 1 – body 2 a 3

```
<mpcommand>
  drawleftarrow((12xu,3.3yu)--(14xu,5.6yu)--(16xu,5.6yu),0.25,black);
  label.rt(btex důležitá úloha etex,(16xu,5.6yu));
</mpcommand>
```

Příkaz `drawleftarrow` vykreslí barvou `black`, perem tloušťky `0.25` bp, lomenou čáru s vrcholy v bodech `(12xu,3.3yu)`, `(14xu,5.6yu)` a `(16xu,5.6yu)`. U prvního vrcholu zleva bude umístěna šipka. Pokud bychom chtěli mít šipku i z druhé strany (tj. například kótu), pak stačí místo `drawleftarrow` psát `drawdoublearrow`.

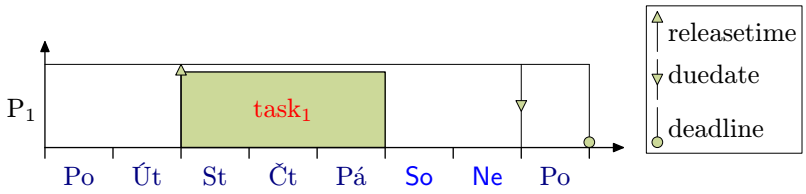
Příkaz `label` vysází obsah mezi `btex` a `etex` TEXem a umístí jej vpravo (`rt`) od bodu `(16xu,5.6yu)`.



Obrázek 6.4: Diagram pro příklad 1 – bod 4

Příklad 2.

Na tomto příkladě si ukážeme, jakým způsobem do obrázku vložit legendu s popisem releasetime, deadline a duedate. Výsledný diagram je vidět na obrázku 6.5.



Obrázek 6.5: Diagram s legendou

Řešením je v elementu `mpcommand` uvést následující příkazy METAPOSTu. (Soubor `ex2.xml`.)

```
<mpcommand>
  picture legend;
  legend := image(
    picture legend;
    circle((0,0),(0,15),0,4,(0.8,0.85,0.6),(0,0,0),(0,0,0),0,0.25);
    label.rt(btex deadline etex,(1,7.5));
    arrow_down((0,18),(0,33),0.5,4,(0.8,0.85,0.6),(0,0,0),(0,0,0),0,0.25);
    label.rt(btex duedate etex,(1,27.5));
    arrow_up((0,36),(0,51),1,4,(0.8,0.8,0.6),(0,0,0),(0,0,0),0,0.25);
    label.rt(btex releasetime etex,(1,43.5));
  );
  z0 = (9xu,0yu);
  draw legend shifted z0;
  draw (bbox legend) shifted z0 withpen pencircle scaled 0.25;
</mpcommand>
```

Nejprve vytvoříme nový obrázek `legend`. Do něj postupně vložíme jednotlivé značky (příkazy `circle`, `arrow_up` a `arrow_down`) a k nim příslušné popisky (příkazy `label`). Poté definujeme bod `z0`. Poslední dva příkazy vykreslí obrázek `legend` v bodě `z0` a kolem něj ještě rámeček.

Dále si vysvětleme pouze parametry použitých příkazů `circle`, `arrow_up` a `arrow_down`, jejichž parametry jsou stejné. (Stejné parametry mají i makra, jejichž jména jsou stejná jako hodnoty vlastnosti `mark-style-type` elementů releasetime, deadline a duedate, s jediným rozdílem, že místo znaku `'-'` použijeme znak `'_'`. Viz sekce 5.2.3.)

První dva parametry udávají body odkud kam se má kreslit svislá čára značky. Například u `circle` to jsou `(0,0)` a `(0,15)`. (Všechny jednotky jsou zadány absolutně v bp).

V pořadí třetí parametr říká, kde mezi zadanými body čáry se bude nacházet symbol značky. (V našem případě je to `0`, což znamená, že symbol značky bude umístěn v prvním bodě, tj. `(0,0)`. Hodnota 1 by znamenala, že bude značka umístěna v bodě `(0,15)`.)

Čtvrtý parametr (`4`), udává velikost symbolu značky. Jeho význam je stejný jako význam vlastnosti `mark-style-size`, viz sekce 5.2.3.

Následují tři barvy, jejichž význam postupně odpovídá vlastnostem `mark-style-color`, `color` a `mark-style-background-color`, viz obrázek 5.6.

Předposlední parametr od konce udává, jestli se bude symbol vyplňovat barvou (hodnota `0`), nebo jestli bude vykreslena pouze jeho kontura (hodnota `1`).

Konečně poslední parametr specifikuje šířku pera.

7 Program Psgantt

Program Psgantt je napsaný ve skriptovacím jazyce Perl. Psgantt v sobě zahrnuje podporu pro programovací jazyk METAPOST a typografický systém TEX, který má také svůj vlastní makrojazyk. Z každého jmenovaného Psgantt přebírá následující vlastnosti.

Z jazyka Perl využívá velké rychlosti zpracování textů, která je, v porovnání s jinými programovacími jazyky, velmi vysoká. Z jazyka METAPOST si Psgantt odnáší snadný způsob zpracování vektorové grafiky a jeho snadnou rozšiřitelnost. Konečně, z TEXu Psgantt využívá vlastností kvalitní sazby, kterými jsou především jeho snadná přizpůsobitelnost pro téměř jakýkoliv dnes známý jazyk (včetně například čínštiny a arabštiny).

7.1 Spouštění Psganttu

Program Psgantt je možné spouštět z příkazové řádky.

- Na systémech Unixového typu můžeme Psgantt spouštět z příkazové řádky zapsáním následujícího příkazu

  ```
  ./psgantt.pl [-mpd] xmlfile[.xml] [cssfile[.css]]
  ```

- V systémech Microsoft Windows může mít příkazová řádka tvar

  ```
  perl psgantt.pl [-mpd] xmlfile[.xml] [cssfile[.css]]
  ```

V příkazové řádce představuje `xmlfile` vstupní XML soubor. Standardně se předpokládá, že vstupní soubor bude mít příponu `xml`, kterou není nutné v příkazu uvádět. Pokud má vstupní soubor jinou příponu, pak ji do příkazu uvést musíme. Tento vstupní soubor musí odpovídat formátu, který byl popsán v kapitole 4. Jméno vstupního XML souboru může být doplněno relativní nebo absolutní cestou k tomuto souboru.

Druhým, nepovinným, parametrem je `cssfile`, což je soubor, ve kterém mohou být obsaženy definice pravidel pro CSS. Pomocí tohoto CSS souboru můžeme upravovat vzhled výsledného obrázku, jak bylo uvedeno v kapitole 5. Standardně se předpokládá, že má příponu `css`.

Pokud v příkazové řádce neuvedeme žádné přepínače, pak bude vygenerován pouze zdrojový soubor pro program METAPOST. Jméno tohoto souboru bude stejné jako jméno vstupního XML souboru, jeho přípona přípona však bude `mp`.

Pokud budeme chtít takto vytvořený soubor zkompilovat METAPOSTem, pak musíme za jménem skriptu `psgantt.pl` ještě uvést přepínač `-m`. V tomto případě bude výsledkem činnosti programu kromě zdrojového souboru také obrázek s ganttovým diagramem. Jméno obrázku bude opět stejné jako jméno vstupního XML souboru, pouze bude mít příponu `eps`.

Uvedení přepínače `-p`, kromě zpracování zdrojového souboru METAPOSTem, navíc převede vygenerovaný obrázek z formátu EPS do formátu PDF. Výsledkem práce programu tedy v tomto případě bude soubor s příponou `pdf`.

Přepínačem `-d` programu nařídíme, aby po sobě smazal všechny pomocné soubory, které si v průběhu své činnosti vytvořil. Ve výsledku tak zůstane vždy pouze ten soubor, který jsme od programu chtěli. To je buď zdrojový soubor pro METAPOST, nebo EPS obrázek, nebo PDF obrázek.

7.2 Popis funkce programu

V této kapitole si ve stručnosti popíšeme, jak program vlastně pracuje. Nebudeme zabíhat do podrobností, půjde nám hlavně o nastínění principu jeho činnosti. Podrobnější informace o jednotlivých částech programu jsou uvedeny v příloze.

Psgantt se nejprve přesvědčí, zda-li vstupní soubor vůbec existuje. Pokud ano, pak jej nejprve zvaliduje podle přiloženého schématu, které je nezbytnou součástí programu. (Princip validace byl popsán v kapitole 4.1.2.) Pokud validace vstupního XML souboru dobře dopadne, můžeme si být jistí, že data, které obsahuje, jsou ve správném tvaru. V opačném případě program vypíše chybové hlášení a poté svou činnost ukončí.

Poté jsou programem načteny a uloženy defaultní pravidla kaskádových stylů (viz kapitola 5), které budou použity v případě, že vstupní soubor žádné CSS neobsahuje, a nebo v příkazové řádce není uveden žádný externí CSS soubor.

Nyní program začne zpracovávat informace uložené ve vstupním XML souboru. K tomu je využíván takzvaný Document Object Model (DOM), viz [2], což je model reprezentace XML (někdy také nazývaný interface) vytvořený a podporovaný konzorciem W3C. XML parser uloží vyparsovaná data do stromové struktury, která odpovídá této specifikaci DOM.

Poté se program podívá, zda-li ve stromu XML existuje nějaký element `matalbdata/style`, který obsahuje definice CSS. Pokud ano, pak načte a uloží informace v něm obsažené. Podobným způsobem je poté zpracován i případný externí CSS soubor, zadaný z příkazové řádky. Po zpracování informací z CSS můžeme začít zpracovávat informace uložené ve vstupním XML.

Program postupně prochází strom XML a ukládá data, která jsou pro něj, podle kapitoly 4.2, důležitá. K těmto datům se snaží podle algoritmu, popsaném v kapitole 5.1.4, vyhledat a nastavit jejich vlastnosti. Všechny potřebné informace si program ukládá do své vlastní stromové struktury.

Po dokončení zpracování vstupního XML má program vytvořen kompletní strukturu s daty XML a jejich CSS vlastnostmi. Nyní program začne tyto informace zpracovávat, aby je poté mohl použít k vytvoření zdrojového souboru pro METAPOST.

Vlastní vytváření zdrojového souboru je složeno z několika částí. Nejprve jsou do souboru vloženy odkazy na externí METAPOSTová makra, která rozšiřují jeho činnost o práci s boxy, a také umožňují vytvářet obrázky velkých rozměrů (viz kapitola věnovaná METAPOSTu 8). Poté jsou do něj přidány makra TEXu pro zpracování popisků formátem CSLATEX. Nakonec jsou do souboru zkopírovány vlastní METAPOSTová makra Psganttu, která obsahují podporu

například pro kreslení precedencí, releasetime a dalších grafických objektů ganttova diagramu. V další části je uveden kód pro jeden obrázek, v něm jsou ve správném pořadí zapisovány příkazy METAPOSTu, které využívají do něj dříve vepsaných maker. Parametry těchto maker jsou data, které jsme získali podle popisu uvedeném v předchozích odstavcích. Poslední část kód tohoto obrázku uzavírá. Tím jsme vytvořili kompletní zdrojový soubor jednoho ganttova diagramu. Nyní můžeme začít s jeho případnou kompilací.

Výsledkem kompilace zdrojového kódu METAPOSTem obdržíme soubor s obrázkem ve formátu EPS, který má příponu `1`. Tu program změní na příponu `eps`.

Pokud je v příkazové řádce uveden přepínač `-p`, je obrázek pomocí standardních nástrojů TEXu, převeden z formátu EPS do formátu PDF.

7.3 Instalace a požadavky

Díky výše zmíněným závislostem na jiných programovacích nástrojích, je nutné, aby byly všechny výše uvedené řádně nainstalovány.

7.3.1 TEX a METAPOST

Pokud budeme chtít využívat možnosti generování obrázků v některém z vektorových formátů Encapsulated PostScript (EPS) nebo Portable Document Format (PDF), je nutné abychom nainstalovali nějakou distribuci typografického systému TEX. Pokud budeme chtít vygenerovat pouze zdrojový soubor pro program METAPOST, nemusíme TEX instalovat.

V prvním jmenovaném případě budeme tedy typografický systém TEX potřebovat. Schválně nehovořím o jediném programu, ale o jejich systému. TEX totiž tvoří poměrně rozsáhlý seznam programů, které spolu jistým způsobem spolupracují při vytváření sazby. Kromě jiných programů v sobě TEX obsahuje i METAPOST, který je jeho standardní součástí.

V této práci se samotnou instalací TEXu zabývat nebudeme. K tomuto účelu je možné najít pro každý operační systém instalační příručku. TEXem se zabývá a vyvíjí jej organizace TEX User Group (TUG). V České republice existuje její odnož, která má název Československé sdružení uživatelů TEXu (CSTUG).

Pro operační systémy založené na Unixu doporučuji použít distribuci TEXu zvanou TEXlive, která v sobě obsahuje METAPOST požadované verze 0.901. Doporučit nelze distribuci zvanou teTEX, která tuto verzi (ke dni 17.1.2007) METAPOSTu neobsahuje. Návod na instalaci TEXlive je možné najít v [18].

V operačních systémech Microsoft Windows doporučuji použít distribuci zvanou MiKTEX verze 2.4 nebo 2.5. Návod na její instalaci lze najít v [19].

Všechny součásti TEXu potřebné pro správnou činnosti Psganttu jsou CSLATEX, CSPlain, ConTEXt a již zmíněný METAPOST. Všechny tyto součásti jsou obsaženy v plné verzi instalace TEXlive i MiKTEXu. V případě MiKTEXu je dokonce možné instalovat pouze jeho minimální verzi. Po spuštění programu Psgantt si MiKTEX dokáže zjistit, které součásti chybí a zeptá se, zda si je přejeme doinstalovat.

7.3.2 Perl

Perl je taktéž dostupný pro oba zmíněné operační systémy. V případě systémů založených na Unixu je již jejich standardní součástí, takže jej není nutné instalovat. Pro systémy Windows je možné zdarma získat produkt ActivePerl [22] od firmy ActiveState. Doporučená verze ActivePerlu je 5.8.7 Build 813.

Psganttem využívané externí balíky (package) Perlu jsou:

- XML::LibXML – je balík, který umožňuje použití knihovny libxml2 (z jazyka C) v Perlu. Libxml2 je knihovna z dílny autorů GNOME, která v sobě obsahuje XML parser s interface podle DOM, validaci Relax NG, XPATH a další aplikace XML. Potřebná verze balíku XML::LibXML je 1.58 nebo vyšší. Dokumentaci a zdrojové kódy je možné najít v [23].
- CSS::SAC – je balík, který obsahuje událostmi řízený CSS parser. Tento parser vyhovuje specifikaci interface SAC (Simple Api for CSS, viz [3]) od konzorcia W3C. Potřebná verze balíku CSS::SAC je 0.06. Dokumentaci a zdrojové kódy je možné najít v [24].

Popišme si v jednotlivých bodech postup instalace balíčků nutných pro běh programu Psgantt.

Instalace v Unixových systémech

K instalaci na Unixových systémech je zapotřebí mít vhodná práva, nejlepší je, být přihlášen jako uživatel root.

1. Nejprve musíme nainstalovat knihovnu libxml2. Ta není součástí Perlu. Nejvýhodnější je k tomuto účelu použít pro danou linuxovou distribuci typický manažer instalačních balíčků. (V mé linuxové distribuci Ubuntu jsem použil program Synaptic.) Potřebnými součástmi knihovny jsou: `libxml2` a `libxml2-dev`. Po jejich úspěšné instalaci se můžeme pustit do instalace samotných balíčků Perlu.

2. V tomto bodě spustíme Perlovský instalátor balíčků, což provedeme zadáním následujícího příkazu do příkazové řádky.

   ```
   perl -MCPAN -e shell
   ```

   (Pokud instalátor spouštíme poprvé, může se nás v průběhu spouštění zeptat na některé doplňující informace. U všech odpovědí, kromě výběru místa odkud se budou balíčky stahovat, stačí pouze stisknout klávesu enter.)

3. Po úspěšném spuštění se nám zobrazí další příkazový řádek, do kterého nyní stačí zadat

   ```
   install XML::LibXML
   ```

   čímž nainstalujeme balík XML::LibXML. Na případné dotazy instalátoru stačí opět odpovědět klávesou enter.

4. Stejným způsobem jako v předcházejícím bodě nainstalujeme balík XML::SAC, pro který by příkazová řádka instalátoru vypadala takto:

   ```
   install CSS::SAC
   ```

5. Činnost instalátoru ukončíme zadáním příkazu `q`.

Instalace v systémech Microsoft Windows

Po nainstalování ActivePerlu (verze 5.8.7 Build 813 – instalační instrukce pro jiné verze ActivePerlu se mohou mírně lišit) nás čekají podobné kroky jako v předchozí sekci. Jediným rozdílem je skutečnost, že pro systémy Windows už existují zkompilované knihovny libxml2, takže ji už instalovat nemusíme.

1. Balík XML::LibXML můžeme (spolu s dll knihovnami libxml2 a jiným softwarem pro XML v Perlu) nainstalovat stažením a rozpakováním zip archívu [25] do adresáře `c:\perl`, kam se ActivePerl standardně nainstaluje.

2. Dalším krokem je nainstalování balíku CSS::SAC, tu provedeme pomocí příkazu

   ```
   ppm install CSS-SAC
   ```

8 METAPOST

METAPOST je systém implementující jazyk pro kreslení obrázků s výstupem do PostScriptu. Jeho autorem je John D. Hobby. METAPOST vznikl z jazyka METAFONT, jehož autorem je Donald E. Knuth. Ten METAFONT vyvíjel již od roku 1977 společně s typografickým systémem TEX. Tato skutečnost je důvodem, proč mají tyto dva systémy tolik společného a jejich možnosti se proto vzájemně doplňují. Z důvodů zmíněné spolupráce s TEXem je METAPOST obzvlášť výhodný pro vytváření obrázků, ve kterých se objevují texty v kombinaci s grafikou. Samotný Knuth prozradil, že k vytváření obrázků nepoužívá žádný jiný program než METAPOST. Ten zároveň poskytuje přístup ke všem možnostem PostScriptu, což je velmi výhodné pro převod vzniklého obrázku do formátu PDF.

Výše uvedené skutečnosti hrály jednu z rozhodujících rolí při výběru vhodného kreslícího softwaru pro program Psgantt. Dokumentaci k programu METAPOST je možné najít v uživatelském manuálu [13]. Stručný úvod je také obsažen v česky psané diplomové práci [15]. Detailní popis jednotlivých možností jazyka METAPOST je možné najít v [11].

8.1 Spouštění METAPOSTu

Jméno překladače METAPOSTu je `mpost`, které je ve všech distribucích TEXu stejné. Při kompilaci zdrojového souboru je možné specifikovat, který formát TEXu má METAPOST použít při sazbě popisků. My této možnosti využijeme a budeme kompilátor volat pomocí následujícího příkazu

```
mpost -tex=cslatex filename[.mp]
```

V něm řetězec `filename` nahradíme jménem překládaného souboru, jehož popisky budou vysázeny formátem CSLATEX. Přípona `mp` je nepovinná, nemusíme ji tedy uvádět.

Zdrojový soubor METAPOSTu obsahuje posloupnost příkazů, které jsou navzájem odděleny středníky. Základní možnosti METAPOSTu si nyní ve stručnosti probereme.

8.2 Datové typy

Základními datovými typy METAPOSTu jsou:

- `numeric` pro uložení čísla,
- `pair` pro uložení souřadnic,
- `path` pro uložení parametrické křivky,

- `picture` pro uložení obrázku,
- `color` pro uložení barvy,
- `string` pro uložení řetězce,
- `boolean` pro uložení hodnoty true nebo false,
- `pen` pro uložení pera,
- `transform` pro uložení afinní transformace.

Proměnné se v METAPOSTu deklarují příkazem

```
numeric a;
```

Ve kterém `a` reprezentuje jméno proměnné a `numeric` jeho datový typ. Text `numeric` může být pochopitelně nahrazen kterýmkoli z výše uvedených datových typů. Dalším možným způsobem deklarace, je deklarace pole daného datového typu. Například následující příkaz

```
numeric a[];
```

deklaruje proměnné `a1`, `a2`,..., které jsou typu numeric. Analogické je to i s ostatními datovými typy.

My budeme v další práci používat hlavně prvních pět uvedených datových typů. Popis ostatních typů je možné najít ve výše uvedené literatuře.

8.2.1 Typ numeric

Tento typ může obsahovat číslo, které je interně uloženo jako celočíselný násobek zlomku $\frac{1}{65536}$. Maximální absolutní hodnota tohoto čísla by neměla přesáhnout hodnotu 4096. Jednotka, ve které jsou hodnoty uloženy, je bp (1 bp $\doteq$ 0,352 mm). Deklarace proměnné typu numeric není povinná. To znamená, že pokud deklarujeme proměnnou bez uvedení jejího typu, bude METAPOST předpokládat, že je typu numeric. Kromě uložení číselných hodnot, mohou proměnné typu numeric specifikovat měřítko, čehož program Psgantt využívá. Například pomocí příkazů

```
xu := 10mm;
yu := 16mm;
```

jsou v Psganttu definovány jednotky x-ové a y-ové osy. Tyto příkazy deklarují proměnné `xu` a `yu`, a současně jim také přiřazují hodnoty 10 a 16 milimetrů. V dalších příkazech pak můžeme místo `mm` použít jednotek `xu` nebo `yu`. Například příkaz

```
a := 2xu;
```

přiřadí proměnné `a` hodnotu odpovídající $2 \cdot 10 = 20$ milimetrů.

Předpokládejme, ze jsou všechny rozměry obrázku zadané v „našich“ jednotkách `xu` a `yu`. Pak změnou jejich hodnoty můžeme ovlivnit rozměry celého obrázku, aniž bychom museli hodnoty všech souřadnic v obrázku ručně přepisovat.

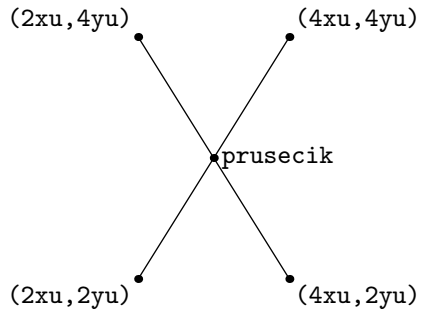
8.2.2 Typ pair

Typ pair reprezentuje jeden bod v rovině. Ten může být tvořen dvěmi číselnými hodnotami nebo proměnnými datového typu numeric. Body můžeme vzájemně sčítat, odčítat, a dále násobit a dělit číslem. Ukázka deklarace a přiřazení proměnné typu pair je vidět na příkazech

```
pair bod;
bod = (1xu,3yu);
```

Zde si musíme dát pozor na znak rovnítka, který METAPOST interpretuje zvláštním způsobem jako soustavu lineárních rovnic. Řešení soustavy poté přidělí jednotlivým souřadnicím bodu. Možnost automatického řešení soustav lineárních rovnic je jednou z dalších výhod jazyka METAPOST. Například průsečík úsečky s body `z1` a `z2` lze zjistit velmi jednoduše příkazem `1/2[z1,z2]`. Uveďme si příklad výpočtu průsečíků dvou přímek, které jsou zadány dvojicí bodů znázorněných na obrázku 8.1.

```
pair prusecik;
prusecik = whatever[(2xu,2yu),(4xu,4yu)]
         = whatever[(2xu,4yu),(4xu,2yu)];
```



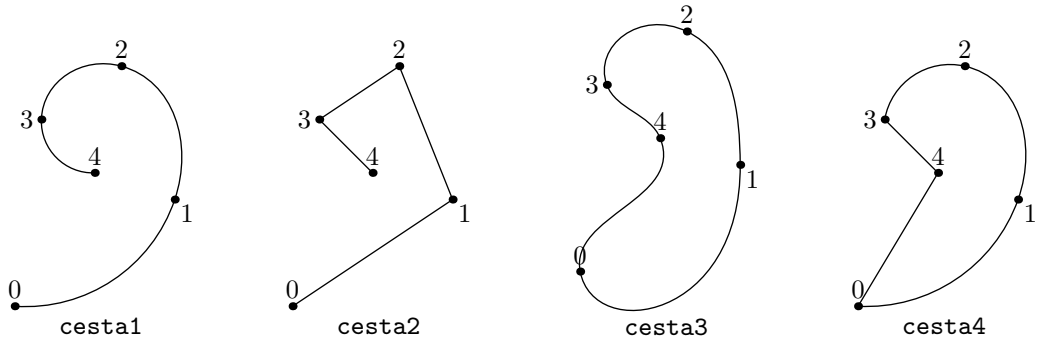
Obrázek 8.1: Výpočet průsečíků dvou přímek

8.2.3 Typ path

Tento datový typ reprezentuje lomenou čáru nebo křivku, danou parametrickým vyjádřením. METAPOST kreslí Béziérovy kubiky procházející zadanými body, které jsou typu pair. Sám si vypočítá kontrolní body křivky tak, aby výsledná křivka byla „co nejhezčí“.

Definice spojité a lomené křivky mohou vypadat takto

```
path cesta[];
cesta1 := (0,0)..(60,40)..(40,90)..(10,70)..(30,50);
cesta2 := (0,0)--(60,40)--(40,90)--(10,70)--(30,50);
cesta3 := (0,0)..(60,40)..(40,90)..(10,70)..(30,50)..cycle;
cesta4 := (0,0)..(60,40)..(40,90)..(10,70)--(30,50)--cycle;
```



Obrázek 8.2: Parametrické křivky

Průběh křivek je znázorněn na obrázku 8.2. Čísla v obrázku odpovídají takzvanému parametru, nebo také času. Ten může nabývat hodnot z intervalu $\langle 0; n\rangle$, kde n je celkový počet bodů křivky.

Křivky a lomené čáry můžeme v cestě libovolně střídat kombinacemi `..` a `--`. Pokud budeme chtít uzavřenou křivku, pak stačí za její poslední bod doplnit text `..cycle`, respektive `--cycle`, který propojí její počáteční bod s koncovým.

METAPOST dále nabízí několik způsobů, jak tvar křivky upravit podle svých představ. Sklon tečny v daném bodě ovlivníme uvedením textu `{dir` α`}` před nebo za bodem, kde α představuje úhel ve stupních. Můžeme také použít již předdefinované směry `up`, `down`, `left` a `right`.

Zaměníme-li text `..` textem `..tension..`a, kde a je kladné číslo, změníme napětí mezi dvěmi body. Maximální napětí křivky lze nastavit pomocí `...`, což je zkratka pro `tension infinity`. Další možnosti lze najít v manuálu [13].

8.2.4 Typ picture

Typ picture představuje speciální datový typ, do kterého je možné uložit celý obrázek, nebo jeho část. Ukládání je výhodné v případě, kdy si přejeme seskupit několik grafických prvků (čar, bodů, popisků, apod.) a pak s nimi pracovat jako s jedním celkem. Do proměnné typu picture se ukládají například všechny popisky (labely), ale také je do něj uložen celý aktuální obrázek, který má přiděleno jméno `currentpicture`. Způsob přiřazení popisku do obrázku je vidět v následujícím kódu

```
picture obrazek;
obrazek := thelabel(btex text labelu etex, bod);
```

Pokud bude text labelu obsahovat řetězec `popisek` a bod bude například `(0,0)`, bude v proměnné `obrázek` uložen obrázek s tímto textem, který bude umístěn v počátku souřadného systému.

Uložení objektů do obrázku umožňuje příkaz `image`. Ukázkový kód uložení objektů do obrázku bude uveden později, v sekci 8.3.1.

8.2.5 Typ color

Typ color je podobný jako typ pair, s tím rozdílem, že má tři souřadnice, které tvoří postupně červenou, modrou a zelenou složku. Hodnoty jednotlivých složek musí ležet v intervalu $\langle 0; 1\rangle$. Černá barva má hodnotu `(0,0,0)`, bílá barva `(1,1,1)`.

8.3 Příkazy pro kreslení

Dosud jsme se seznámili pouze s datovými typy, které se mohou ve zdrojovém kódu vyskytnout. Pokud nějaké proměnné typu path přiřadíme cestu, nebo do nějaké proměnné typu picture vložíme popisek nebo jiný obrázek, nic se do výsledného obrázku nevykreslí. K samotnému vykreslení slouží příkazy `draw`, `fill` a další. Pro naše účely budou dostačující tyto dva. Ostatní kreslící příkazy METAPOSTu lze najít například v [13].

8.3.1 Příkaz draw

Tento příkaz vykreslí do obrázku `currentpicture` svůj argument. Argumentem může být buď proměnná typu path nebo picture, nebo jim odpovídající výrazy. Výrazem zde myslíme text, který se v předchozích příkladech objevoval napravo od znaku přiřazení `:=`.

Příkaz, který by vykreslil křivku `cesta1` vypadá takto

```
draw cesta1;
```

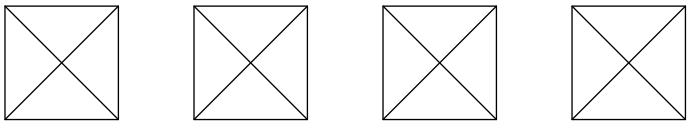
Pokud bychom místo proměnné `cesta1` uvedli rovnou výraz, pak by měl příkaz tvar:

```
draw (0,0)..(60,40)..(40,90)..(10,70)..(30,50);
```

Nyní si uveďme slibovaný příklad na využití datového typu picture. Dejme tomu, že budeme chtít, aby výsledný obrázek obsahoval čtyři čtverce, každý o straně 1,5 cm. V každém čtverci mají být vykresleny jeho úhlopříčky. Tyto čtverce chceme umístit vedle sebe. Mezi čtverci má být navíc vynecháno místo o velikosti 1 cm. Náš problém může vyřešit následující kód, jehož výsledek je vidět na obrázku 8.3.

```
z1 = (0,0);          z2 = (1.5cm,0);
z3 = (1.5cm,1.5cm); z4 = (0,1.5cm);
picture ctverec;
ctverec := image ( draw z1--z2--z3--z4--cycle;
                   draw z1--z3;
                   draw z2--z4; );
draw ctverec;
draw ctverec shifted (2.5cm,0);
draw ctverec shifted (5cm,0);
draw ctverec shifted (7.5cm,0);
```

Výhodou použití příkazu `image` je možnost uložení libovolného obsahu do proměnné typu picture, kterou pak můžeme libovolně transformovat. Transformace budou popsány v kapitole 8.4.



Obrázek 8.3: Příklad použití datového typu picture

8.3.2 Příkaz fill

Tento příkaz vyplní uzavřenou cestu danou barvou. Jeho syntaxe je

`fill` cesta `withcolor` barva`;`

8.4 Transformace

V METAPOSTu můžeme používat následující transformace:

- posunutí o vektor (a, b): (x, y) `shifted` $(a, b) \longrightarrow (x + a, y + b)$,
- stejnolehlost s koeficientem a: (x, y) `scaled` $a \longrightarrow (ax, ay)$,
- změna měřítka na ose x: (x, y) `xscaled` $a \longrightarrow (ax, y)$,
- změna měřítka na ose y: (x, y) `yscaled` $a \longrightarrow (x, ay)$,
- otočení o úhel θ (ve stupních) kolem počátku, proti směru hodinových ručiček: (x, y) `rotated` $\theta \longrightarrow (x \cos\theta - y \sin\theta, x \sin\theta + y \cos\theta)$,
- zkosení s koeficientem a: (x, y) `slanted` $a \longrightarrow (x + ay, y)$,
- rotace a stejnolehlost: (x, y) `zscaled` $(a, b) \longrightarrow (ax - by, bx + ay)$,
- osová souměrnost podle přímky procházející body a a b: (x, y) `reflectedabout` (a, b),
- otočení o úhel θ (ve stupních), kolem bodu a, proti směru hodinových ručiček: (x, y) `rotatedaround` (a, θ)

Transformace mohou být aplikovány na proměnné nebo výrazy, ke kterým jsou připojovány zleva (viz transformace `shifted` v příkladu se čtverci). Přesný popis syntaxe a způsobu použití je uveden v [13].

8.5 Makra

METAPOST má v sobě zabudován vlastní makrojazyk, který nám umožňuje definovat nové příkazy. Této možnosti využívá i program Psgantt, který například definuje makro `circle` pro vykreslení defaultní značky pro deadline. Kdykoli tedy můžeme vytvořit nové makro, které stávající možnosti programu rozšiřuje.

Nebudeme zde popisovat veškeré aspekty definování nových maker, pouze nastíníme základní princip. Vytvoříme makro, které pojmenujeme `drawcircle`, které vykreslí kružnici se středem `s` a průměrem `d`.

```
vardef drawcircle (expr s, d) =
    draw fullcircle scaled d shifted s;
enddef;
```

Parametry makra jsou uvedeny v závorce hned za jeho jménem. Typy parametrů mohou být celkem tři. Jsou jimi

- `expr` – což může být výraz libovolného datového typu,
- `suffix` – který může obsahovat libovolné jméno proměnné,
- `text` – který může obsahovat libovolný text (přesněji řečeno libovolný seznam tokenů). Součástí tohoto parametru tak může být například jeden nebo i více příkazů.

V případě, že budeme chtít, aby makro nějaký výraz vracelo, uvedeme jej jako poslední příkaz v makru, těsně před ukončující `enddef`. Za tímto posledním příkazem už nesmí být středník, jinak by došlo místo vrácení hodnoty k jeho provedení.

Pokud bude vytvářené makro potřebovat nějaké vlastní proměnné, například pro uložení mezivýsledků, je nutné před jejich deklarací uvést příkaz:

`save` jméno`;`

kde jméno nahradíme skutečným jménem proměnné. Příkaz `save` uloží hodnotu proměnné, kterou měla před provedením tohoto příkazu, a po ukončení makra jí ji zase vrátí zpátky.

Námi definované makro `drawcircle` bychom v programu volali například uvedením příkazu:

```
drawcircle((0,0),20);
```

8.6 Boxy

Boxy představují rozšíření základních METAPOSTových maker o možnosti lepší práce s obdelníky, ovály nebo kružnicemi. Nás budou zajímat hlavně prvně uvedené obdelníky. Základní myšlenkou obdelníkových boxů je jejich snadné vytvoření, které umožňuje příkaz:

`boxit.`⟨suffix⟩`(`⟨picture expression⟩`);`

V něm ⟨suffix⟩ zastupuje jméno boxu a ⟨picture expression⟩ libovolný výraz typu picture, kolem kterého bude box vykreslen. Tento příkaz deklaruje body ⟨suffix⟩`.c`, ⟨suffix⟩`.n`, ⟨suffix⟩`.e`, ..., (viz obrázek 8.4), které mohou být použity pro následné určení polohy boxu.

V našem případě Psgantt do zdrojového souboru napíše například příkazy

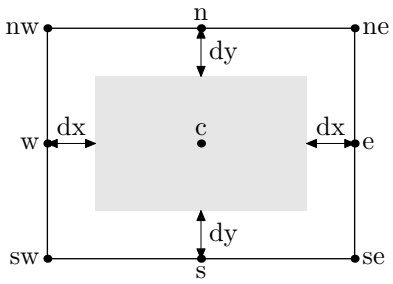
```
boxit.b7();
b7.sw = (12xu,0yu);
b7.ne = (27xu,1yu);
```

které vytvoří prázdný box (bez obrázku), jehož rozměry a poloha budou dány body `b7.sw` a `b7.ne`. Do nich jsou postupně dosazeny hodnoty `item/start` a `item/start` + `item/length`.

Příkazem

`drawboxed(`⟨suffix list⟩`);`

pak vykreslíme jeden nebo více boxů.



Obrázek 8.4: Boxem obklopený obrázek a jím definované body a hodnoty

Pro naše účely Psgantt nastaví hodnoty `b7.dx` a `b7.dy` rovny nule, takže pak bude mít box přesně požadované rozměry. Výška boxu je vždy jednotková, tj. `1yu`.

Možnosti boxů oceníme například při kreslení precedencí, kdy stačí znát pouze číslo (jméno) boxu a vzdálenost od jeho horního okraje. Tyto informace nám stačí k definici počátečního, respektive koncového bodu precedence. Pokud pak budeme chtít v obrázku například zvětšit vertikální mezery, změní se pouze definice bodu `b7.ne` a `b7.sw`, zbylý kód zůstane beze změny.

Bližší informace o možnostech boxů je možné najít v [13].

8.7 Rozšíření METAPOSTu

Makrojazyk umožňuje velmi široké rozšíření možností základního METAPOSTu.

Velmi zdařilým, ale také obsáhlým, souborem maker je MetaFun, jehož autorem je Hans Hagen. MetaFun rozšiřuje základní možnosti METAPOSTu například o barevné schéma CMYK, makra pro podporu formátu PDF, jako například transparentní barvy, a mnohé další. Podrobnosti lze najít v jeho dokumentaci [14].

Jako příklad dalších rozšíření METAPOSTu můžeme uvést makra pro vytváření 3D grafiky, UML diagramů, fraktálů a také vytváření animací.

9 Závěr

V této práci byl vytvořen XML formát, pomocí kterého mohou být popsány ganttovy diagramy. Tento návrh se neomezuje pouze pro jedinou aplikaci (Psgantt), ale počítá se s jeho možným využitím i v jiných aplikacích. Jelikož vstupní XML soubor nemusí obsahovat žádné informace určující vzhled ganttova diagramu, byly navrženy kaskádové styly (CSS), pomocí nichž lze nastavit vzhled jednotlivých částí diagramu, tak i jeho celku. V aktuální verzi Psganttu je možné nezávisle na sobě nastavit více než 60 vlastností kaskádových stylů. Pro CSS byla navíc vypracována podpora několika selektorů, s jejichž pomocí můžeme vybírat elementy vstupního XML souboru, na které se mají vlastnosti CSS aplikovat. Je tedy možné vykreslovat pouze ty části diagramu, které nás skutečně zajímají.

Dále byl ověřen předpoklad výhodnosti použití jazyka METAPOST, jako nástroje pro vlastní tvorbu obrázků. Vzniklé obrázky jsou ve formátu EPS, ale je možné je převést i do formátu PDF. Narozdíl od formátu EPS, formát PDF již v sobě obsahuje všechny fonty a je tedy plně přenositelný. Pro jazyk METAPOST byly vytvořeny makra, které kreslení ganttových diagramů usnadňují a mohou být kdykoli nahrazena jinými, komplexnějšími.

Pro automatické generování obrázků ganttových diagramů byl vytvořen program Psgantt, který v sobě zahrnuje podporu pro výše uvedené XML, CSS a METAPOST. Výhodou programu Psgantt je, že je napsán v programovacím jazyce Perl, jehož zdrojové soubory mohou být přenositelné mezi různými operačními systémy. Psgantt byl nezávisle testován na operačním systému Windows XP a na systému GNU/Linux (Ubuntu). Na obou byl plně funkční.

Psgantt byl testován vstupními XML soubory, které byly vytvářeny pomocí TORSCHE Scheduling Toolboxu pro Matlab. Při testování byla také využívána schopnost tohoto toolboxu generovat náhodné rozvrhy.

Možná rozšíření programu Psgantt jsou: Vypracování podpory pro další selektory CSS, které umožní snadnější výběr elementů, k čemuž bude nutné navrhnout novou strukturu ukládání dat CSS. Dalším zlepšením může být vypracování podpory pro takzvané z-indexy v CSS, které by umožňovaly ovlivňovat pořadí vykreslování jednotlivých grafických objektů. Z hlediska výstupního PDF formátu by byla jistě výhodná možnost používání transparentních barev, kterou umožňuje rozšíření METAPOSTu zvané Metafun.

Program Psgantt může být využit při výuce předmětů na katedře řízení FEL ČVUT, které se zabývají rozvrhováním.

Literatura

[1] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, editors. *Extensible Markup Language (XML) 1.0 Four Edition.* W3C (World Wide Web Consortium), 2006. [online] <http://www.w3.org/TR/REC-xml>

[2] Philippe Le Hégaret, Ray Whitmer, Lauren Wood. *Document Object Model (DOM).* [online] <http://www.w3.org/DOM/>

[3] Philippe Le Hégaret. *The Simple API for CSS.* [online] <http://www.w3.org/Style/CSS/SAC/>

[4] Elliote Rusty Harold & W. Scott Means. (Překlad Martin Blažík). *XML v kostce.* Computer Press, 2002. ISBN-807226-712-4.

[5] OASIS Committee Specification. *RELAX NG Specification.* 3 December 2001. [online] <http://www.oasis-open.org/committees/relax-ng/spec-20011203.html>.

[6] Michael Doob. (Překlad Josef Daneš, Jiří Veselý). *Jemný úvod do TEXu.* Československé sdružení uživatelů TEXu, 1993.

[7] Jiří Rybička. *LATEX pro začátečníky.* Konvoj, 2003, 3. vydání. ISBN 80-7302-049-1.

[8] Petr Olšák. *Typografický systém TEX.* Konvoj, 2000, 2. vydání. ISBN-80-85615-91-6.

[9] Petr Olšák. *TEXbook naruby.* Konvoj, 2001, 2. vydání. ISBN-80-7302-007-6.

[10] Donald E. Knuth. *The TEXbook*, volume A of *Computers and Typesseting.* Addison-Wesley, Reading, MA, USA, 1986. ISBN 0-201-13447-0.

[11] Donald E. Knuth. *The METAFONTbook*, volume C of *Computers and Typesseting.* Addison-Wesley, Reading, MA, USA, 1986. ISBN 0-201-13445-4.

[12] Bert Bos, Håkon Wium Lie, Chris Lilley, Ian Jacobs, editors. *Cascading Style Sheets, level 2 CSS2 Specification.* W3C (World Wide Web Consortium), 1998. [online] <http://www.w3.org/TR/REC-CSS2>

[13] John D. Hobby. *A User's Manual for MetaPost.* AT&T Bell Laboratories Computing Science. Technical Report 162, 1992. [online] <http://tug.org/docs/metapost/mpman.pdf>

[14] Hans Hagen *MetaFun.*
Pragma. Advanced Document Engineering, Hasselt NL, 2002. [online]
<http://www.pragma-ade.com/general/manuals/metafun-s.pdf>

[15] Miroslava Krátká. *Diplomová práce.*
Tvorba obrázků pro matematické texty pomocí METAPOSTu. Brno, 2001. [online]
<http://mirka.janik.cz/dp>

[16] Michal Kutil, Přemysl Šůcha, Michal Sojka a Zdeněk Hanzálek. *TORSCHE Scheduling Toolbox for Matlab Users Guide.* DCE, CTU in Prague. [online]
<http://rtime.felk.cvut.cz/scheduling-toolbox>

[17] M. Stibor, M. Kutil. *TORSCHE Scheduling Toolbox: List Scheduling.* In proceeding of Process Control 2006 [CD-ROM]. Pardubice, University of Pardubice. [online]
<http://rtime.felk.cvut.cz/scheduling-toolbox/download/articles/2006/process-control-list-schedulig.pdf>

[18] Sebastian Rahtz & Karl Berry, editoři. *TEX-Collection 2005. Příručka TEXLive.* [online]
<http://tug.org/texlive/doc/texlive-cz/live.pdf>

[19] Christian Schenk. *MikTEX 2.5 Manual* [online]
<http://docs.miktex.org/manual/>

Použitý software

[20] MiKTeX <http://miktex.org>

[21] TeXLive <http://tug.org/texlive>

[22] ActivePerl <http://www.activestate.com/Products/ActivePerl>

[23] XML::LibXML
<http://search.cpan.org/author/PAJAS/XML-LibXML-1.62001/LibXML.pod>

[24] CSS::SAC <http://search.cpan.org/author/BJOERN/CSS-SAC-0.06/SAC.pm>

[25] XML XSH
<http://www.structuredocs.com/wordpress/wp-content/uploads/2006/12/xsh-files-12-12-06.zip>

A CD-ROM

Přílohou diplomové práce je CD-ROM, který obsahuje:

- všechny zdrojové soubory programu Psgantt,
- dokumentaci k těmto zdrojovým souborům,
- zdrojové soubory vlastní diplomové práce pro typografický systém TEX,
- diplomovou práci ve formátu PDF,
- zdrojové soubory k příkladům,
- perlovské balíky, potřebné k běhu programu Psgantt.