VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

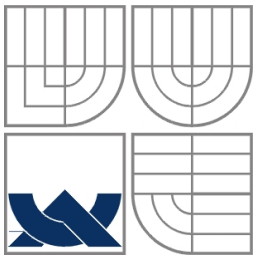
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

PROSTŘEDKY PRO IMPLEMENTACI ROZLOŽENÍ WEBOVÝCH STRÁNEK V JAVASCRIPTU

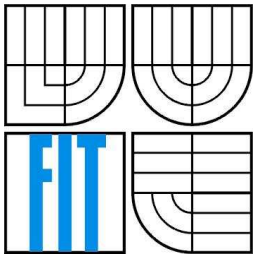
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE Bc. Gregor T. Kovács
AUTHOR

BRNO 2008



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

PROSTŘEDKY PRO IMPLEMENTACI ROZLOŽENÍ WEBOVÝCH STRÁNEK V JAVASCRIPTU

WEB PAGE LAYOUT FACILITIES IN JAVASCRIPT

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. Gregor T. Kovács

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. Radek Burget Ph.D.

BRNO 2008

Zadání diplomové práce/6733/2007/xkovac06

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2007/2008

Zadání diplomové práce

Řešitel: **Kovács Gregor T., Bc.**
Obor: Informační systémy
Téma: **Prostředky pro implementaci rozložení webových stránek v JavaScriptu**
Kategorie: Web

Pokyny:

1. Seznamte s jazykem CSS včetně jeho navrhovaných budoucích rozšíření.
2. Zhodnoťte existující nedostatky jazyka.
3. Prostudujte prostředky pro realizaci rozložení prvků uživatelského rozhraní v jazyce Java.
4. Navrhněte způsob realizace podobného řešení ve webových stránkách.
5. Navržené řešení vhodným způsobem implementujte v jazyce JavaScript.
6. Implementujte grafický editor rozložení prvků na stránce v jazyce JavaScript.
7. Zhodnoťte dosažené výsledky a navrhněte pokračování projektu.

Literatura:

- Bos, B. et al.: Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification, W3C Candidate Recommendation 19 July 2007
- Petr Staníček: CSS Kaskádové styly, Computer Press, 2003, ISBN 80-7226-872-4

Při obhajobě semestrální části diplomového projektu je požadováno:

- Bez požadavků.

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci ročníkového a semestrálního projektu (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVR-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Burget Radek, Ing., Ph.D.**, UIFS FIT VUT
Datum zadání: 24. září 2007
Datum odevzdání: 21. ledna 2008

L.S.

doc. Ing. Jaroslav Zendulka, CSc.
vedoucí ústavu

LICENČNÍ SMLOUVA
POSKYTOVANÁ K VÝKONU PRÁVA UŽÍT ŠKOLNÍ DÍLO

uzavřená mezi smluvními stranami

1. Pan

Jméno a příjmení: **Bc. Gregor T. Kovács**
Id studenta: 49581
Bytem: Páterova 86, 048 01 Rožňava
Narozen: 09. 06. 1984, Rožňava
(dále jen "autor")

a

2. Vysoké učení technické v Brně

Fakulta informačních technologií
se sídlem Božetěchova 2/1, 612 66 Brno, IČO 00216305
jejímž jménem jedná na základě písemného pověření děkanem fakulty:

.....
(dále jen "nabyvatel")

Článek 1
Specifikace školního díla

1. Předmětem této smlouvy je vysokoškolská kvalifikační práce (VŠKP):
diplomová práce

Název VŠKP: Prostředky pro implementaci rozložení webových stránek v JavaScriptu
Vedoucí/školitel VŠKP: Burget Radek, Ing., Ph.D.
Ústav: Ústav informačních systémů
Datum obhajoby VŠKP: .....

VŠKP odevzdal autor nabyvateli v:
tištěné formě počet exemplářů: 1
elektronické formě počet exemplářů: 2 (1 ve skladu dokumentů, 1 na CD)

2. Autor prohlašuje, že vytvořil samostatnou vlastní tvůrčí činností dílo shora popsané a specifikované. Autor dále prohlašuje, že při zpracovávání díla se sám nedostal do rozporu s autorským zákonem a předpisy souvisejícími a že je dílo dílem původním.
3. Dílo je chráněno jako dílo dle autorského zákona v platném znění.
4. Autor potvrzuje, že listinná a elektronická verze díla je identická.

Článek 2
Udělení licenčního oprávnění

1. Autor touto smlouvou poskytuje nabyvateli oprávnění (licenci) k výkonu práva uvedené dílo nevýdělečně užít, archivovat a zpřístupnit ke studijním, výukovým a výzkumným účelům včetně pořizování výpisů, opisů a rozmnoženin.
2. Licence je poskytována celosvětově, pro celou dobu trvání autorských a majetkových práv k dílu.
3. Autor souhlasí se zveřejněním díla v databázi přístupné v mezinárodní síti:
   - ☐ ihned po uzavření této smlouvy
   - ☐ 1 rok po uzavření této smlouvy
   - ☐ 3 roky po uzavření této smlouvy
   - ☐ 5 let po uzavření této smlouvy
   - ☐ 10 let po uzavření této smlouvy

   (z důvodu utajení v něm obsažených informací)
4. Nevýdělečné zveřejňování díla nabyvatelem v souladu s ustanovením § 47b zákona č. 111/ 1998 Sb., v platném znění, nevyžaduje licenci a nabyvatel je k němu povinen a oprávněn ze zákona.

Článek 3
Závěrečná ustanovení

1. Smlouva je sepsána ve třech vyhotoveních s platností originálu, přičemž po jednom vyhotovení obdrží autor a nabyvatel, další vyhotovení je vloženo do VŠKP.
2. Vztahy mezi smluvními stranami vzniklé a neupravené touto smlouvou se řídí autorským zákonem, občanským zákoníkem, vysokoškolským zákonem, zákonem o archivnictví, v platném znění a popř. dalšími právními předpisy.
3. Licenční smlouva byla uzavřena na základě svobodné a pravé vůle smluvních stran, s plným porozuměním jejímu textu i důsledkům, nikoliv v tísni a za nápadně nevýhodných podmínek.
4. Licenční smlouva nabývá platnosti a účinnosti dnem jejího podpisu oběma smluvními stranami.

V Brně dne: .....

....

Nabyvatel Autor

Abstrakt

Cieľom tejto práce je vytvorenie návrhu a implementácia aplikácií pre rozloženie webových stránok pomocou JavaScriptu. Práca obsahuje popis aktuálnych možností pozíciovania objektov s využitím možností kaskádových štýlov, obsahuje popis normy CSS 2.1., a poukazuje na nedostatky v pozíciovaní s využitím možností CSS. Ďalej sa zaoberá analýzou riešenia rozloženia objektov pri programovacom jazyku Java, využitím správcov rozloženia mriežkového typu GridLayout a GridBagLayout.

Na základe získaných poznatkov, sú vytvorené návrhy na riešenie rozloženia objektov pri tvorbe webových stránok na princípe mriežkového rozloženia. Rozloženie objektov na stránke sa rieši pomocou využitia rozšírenia HTML atribútov o nové vlastnosti pre pozíciovanie objektov, ako aj pomocou vytvorenia grafického editora pre rozloženie objektov. Riešenia sú implementované v jazyku JavaScript.

Klíčová slova

CSS, CSS pozíciovanie, layout, správca rozloženia, JavaScript, DOM, objektový model dokumentu, GridBagLayout, HTML, grafický editor,

Abstract

The aim of this work is to design and implement applications for the creation of web page layout facilities using JavaScript. The work includes the descriptions of the available methods of object positioning using the CSS given possibilities, the CSS 2.1 standard, and the difficulties of object positioning using CSS. Further, it includes the analysis of how the object placement is solved in the Java programming language using grid based layout managers GridLayout and GridBagLayout.

Based on the obtained knowledge, designs are created for the solving of object placement in the creation of web pages using the grid principle. The object placement is solved by defining new HTML attributes for position determination, and also by creating a graphical editor for object placement. All the solutions are implemented using JavaScript.

Keywords

CSS, CSS positioning, layout, layout manager, JavaScript, DOM, document object model, GridBagLayout, HTML, graphical editor

Citace

Gregor T. Kovács: Prostriedky pre implementáciu rozloženia webových stránok v JavaScripte, diplomová práca, Brno, FIT VUT v Brně, 2008.

Prostriedky pre implementáciu rozloženia webových stránok v JavaScripte

Prehlásenie

Prehlasujem, že som túto diplomovú prácu vypracoval samostatne pod vedením Ing. Radka Burgeta Ph.D.
Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

……
Gregor T. Kovács
5.1.2008

Poďakovanie

Chcel by som sa poďakovať Ing. Radkovi Burgetovi Ph.D.za poskytnutie odbornej pomoci a usmernenia pri tvorbe diplomovej práce.

Obsah

1. Úvod

Kaskádové štýly sú v informatike používané pre popis vzhľadu dokumentu. Najčastejšie sa využívajú pri tvorbe výzoru webových aplikácií väčšinou v kombinácii s jazykmi XHTML a HTML, alebo XML. Norma popisujúca kaskádové štýly má viacero vydaní, a je spravovaná organizáciou Word Web Consortium (W3C). Momentálne používaná norma je norma CSS 2.1. Kaskádové štýly v ich dnešnom stave neriešia dostatočne problematiku pozíciovania objektov na webových stránkach. V nasledujúcich kapitolách je rozobratá problematika týchto nedostatkov, je vytvorených niekoľko nových návrhov ako riešiť rozloženie objektov na webovej stránke. Tieto návrhy vychádzajú z popísanej štúdie rozloženia objektov správcami rozloženia pracujúcimi na mriežkovom princípe v jazyku Java. Navrhované riešenia sú implementované s využitím jazyka JavaScript, za podpory HTML a CSS.

1.1 Navodenie problematiky

Práca sa zaoberá štúdiou aktuálneho stavu CSS pozíciovania, popisuje vlastnosti a možnosti pozíciovania objektov podľa normy CSS 2.1. Cieľom práce je zjednodušiť spôsob umiestňovania objektov na stránke a vytvoriť intuitívnejší a jednoduchší postup pri umiestňovaní objektov, vychádzajúc z poznatkov riešenia rozloženia objektov v prípade programovacieho jazyka Java a jeho správcov rozloženia.

Budú navrhnuté nové spôsoby jednoduchšieho a efektívnejšieho riešenia problematiky pozíciovania a rozmiestňovania objektov na webovej stránke, ktoré majú za cieľ riešenie problémov pri tvorbe webových aplikácií (napr.: formulárov), kde je potrebné umiestniť väčšie množstvo objektov na stránke.

Momentálne dostupné možnosti CSS pozíciovania neriešia v dostatočnej miere problematiku umiestňovania väčšieho množstva objektov na stránke. Pri tvorbe zložitejších webových aplikácií sa často využívajú tabuľky ako prostriedok pre tvorbu kostry stránky, do ktorej sa potom umiestňujú objekty. Jedná sa síce o zastaralý , ale kvôli zložitosti CSS pozíciovania stále používaný spôsob rozloženia objektov na stránke.

Navrhnuté rozšírenia majú vytvoriť jednoduchší spôsob pozíciovania objektov pomocou generovania potrebného CSS kódu pre rozloženie objektov, ktoré budú schopné nahradiť tabuľky a iné zastarané spôsoby rozmiestňovania objektov.

1.2 Stručný popis kapitol

Druhá kapitola práce sa zaoberá popisom kaskádových štýlov, vysvetľuje ich význam a popisuje syntax a pravidlá určovania a prideľovania štýlových vlastností danému dokumentu (napr. HTML stránke) alebo objektom stránky. Obsahuje stručný popis normy CSS 2.1. so zameraním na popis modelu spracovania CSS a pravidiel pri spracovaní chýb v syntaxi definovaného CSS kódu. Zaoberá sa niektorými problémami a nedostatkami spojenými s normou CSS 2.1.

V tretej kapitole sa rozoberá problematika pozíciovania objektov na stránke. Sú popísané ako zastarané, tak aj aktuálne používané spôsoby rozmiestňovania objektov pri tvorbe web stránok.

Štvrtá kapitola sa zaoberá stručným popisom objektového modelu dokumentu, jeho základnými vlastnosťami, významom stromovej štruktúry dokumentu, ako aj významom tohto modelu pre JavaScript.

Kapitola piata sa zaoberá spôsobmi rozloženia objektov v jazyku Java. Zameriava sa hlavne na princípy rozmiestňovania objektov, založených na Java správcoch rozloženia (*layout manager*) pracujúcich s mriežkovým rozložením. Podrobne sa preberajú správcovia rozloženia *GridLayout* a *GridBagLayout*.

Šiesta kapitola je zameraná na návrh pozíciovania objektov na stránke vychádzajúc z poznatkov, získaných štúdiom rozloženia objektov mriežkovými správcami rozloženia jazyka Java. Popisujú sa podrobné návrhy pre pozíciovanie objektov pri tvorbe webových aplikácií, s využitím rozširujúcich HTML atribútov pre pozíciovanie objektov, ako aj pre tvorbu grafického editora zabezpečujúceho rozloženie objektov.

Kapitola siedma sa zaoberá implementáciou v šiestej kapitole navrhnutých riešení pozíciovania objektov pri tvorbe webových aplikácií.

V poslednej, ôsmej, kapitole je vyhodnotená práca a poznatky v nej získané. Sú uvedené niektoré možné budúce rozšírenia.

2. CSS - Kaskádové štýly

2.1 Úvod do CSS

V informatike sú Kaskádové štýly (Cascading Style Sheets, CSS) štýlový jazyk (stylesheet language) používaný pre popis vzhľadu dokumentu v rámci značkovacieho jazyka. Ich najčastejším využitím je tvorba výzoru, webových stránok písaných väčšinou v jazykoch HTML a XHTML, ale dajú sa použiť aj v spojení s jazykmi ako XML. Normu týkajúcu sa Kaskádových štýlov spravuje organizácia Word Wide Web Consortium (W3C).

Jazyk kaskádových štýlov má niekoľko vydaní. Pričom jednotlivé vydania stavali na im predchádzajúce vydania, typicky pridaním nových vlastností. Jednotlivé verzie sú značené ako CSS1 a CSS 2, CSS 2.1, pričom sa už pripravuje ďalšia nadstavba v podobe verzie CSS 3.

2.2 Vznik a vývoj CSS

Postupným rozširovaním sa siete internetu medzi ľuďmi sa na internete objavovalo čoraz viacej stránok s komerčným zameraním, pri ktorých sa dával väčší dôraz na vzhľad stránky. Zo začiatku sa ako o obsah tak aj výzor internetových stránok staral jazyk HTML. V dôsledku rozrastávania sa internetovej siete a rastúcich nárokov užívateľov bolo potrebné oddeliť od seba obsah a vzhľad stránok.

Jazyk HTML, ako aj každý z jeho prvkov má svoju základnú sémantiku popísanú v definícii jazyka, ktorá je daná aj základnými vlastnosťami a atribútmi jazyka. Tieto atribúty nie sú zamerané na vzhľad dokumentu, ale na zobrazenie informácií v dokumente, a predpokladajú, že o samotný vzhľad dokumentu sa postará prehliadač.

S postupným vývojom HTML dochádzalo k tlaku, aby vznikla možnosť do detailov ovplyvniť vzhľad stránok a správanie vizuálnych aj niektorých nevizuálnych prvkov. Z tohto dôvodu sa postupne pri každom prvku zvlášť oddelila množina atribútov, ktoré ovplyvňujú vzhľad prvku, ako aj niektoré nevizuálne vlastnosti prvku. Táto množina je množina atribútov tvoriaca štýl prvku. Táto množina, a teda v nej sa nachádzajúce atribúty popisujúce štýl atribútov, je jazyk kaskádových štýlov CSS.

Prvá prijatá norma popisujúca jazyk CSS bola prijatá roku 1996. Vypracovalo ju konzorcium World Wide Web Consortium ako návrh pre štandard kaskádových šablón (CSS – Cascading Style Sheets). V roku 1998 vznikol návrh normy CSS 2, ktorý už zahrňuje aj štýly pre iné nevizuálne typy médií. Posledná zverejnená norma týkajúca sa kaskádových štýlov je norma CSS 2.1, ktorá bola vydaná konzorciom World Wide Web Consortium v roku 2006. CSS 2.1 opravuje niektoré z

vlastností zavedených pri norme CSS 2, ktoré boli považované za nepoužiteľné, prípadne nepraktické. CSS 2.1 je takpovediac reakcia normy CSS 2 na prax. [3][9]

2.3 Pravidlá kaskádových štýlov

Kaskádové štýly sa používajú pre definovanie pravidiel ako prvok reprodukovať. Pojem reprodukovať môžeme chápať ako množinu zahrňujúcu pojmy, popisujúce spôsoby zobrazenia prvku (napr. zobraziť). Takýchto pravidiel môže byť veľa.

Kaskádovosť štýlov je daná hierarchickým usporadúvaním pravidiel pre štýly. Každý podriadený štýl predefinuje rovnako nazvané nadradené pravidlá pre istý prvok. Pravidlá sa interpretujú v kaskáde:

- externá šablóna štýlov
- lokálny štýl dokumentu
- lokálny štýl konkrétneho prvku

2.3.1 Spôsoby pričlenenia štýlových vlastností prvkom

Pri vytváraní štýlových vlastností prvku sa tieto dajú deklarovať na viacerých miestach. Zohľadňuje sa pritom hierarchia uvedená v kaskáde štýlov.

Miesta deklarácie štýlu prvku:

- Priamo v texte zdroja u formátovaného elementu pomocou atribútu
- V štýlovom predpise (stylesheet) v hlavičke stránky
- V externom štýlovom predpise

Najčastejšie používanou deklaráciou štýlu je použitie externého súboru **.css*, ale používajú sa aj druhé dve metódy.

V prípade použitia metódy vloženia popisu štýlu priamo do zdrojového textu formátovaného prvku sa dané štýlové vlastnosti deklarujú pomocou atribútu *style="..."*. Tento spôsob definovania štýlových vlastností prvku sa označuje ako priamy štýl, ale v dnešnej dobe sa používa už len v ojedinelých prípadoch.

Rozšírenejší spôsob deklarácie štýlových vlastností pre daný prvok je metóda deklarácie štýlového predpisu v hlavičke stránky. V tomto prípade sú atribúty určujúce štýlové vlastnosti jednotlivých prvkov na danej stránke uvedené na začiatku stránky v oblasti definovanej začínajúcim a koncovým tagom *<style>* a *</style>*.

Najrozšírenejšou formou deklarácie štýlov je použitie externého súboru **.css*, na ktorý sa jednotlivé dokumenty, obsahujúce prvky, pre ktoré sa majú jednotlivé štýlové vlastnosti uplatniť odkazujú pomocou tagu *<link>*. Výhodou tejto metódy, kde máme štýlový predpis umiestnený

v samostatnom súbore, na ktorý sa potom len odkazuje je, že na jeden súbor môžeme nalinkovať viacero dokumentov. Externý CSS súbor umožňuje mať údaje o vzhľade celej webovej stránky na jednom mieste v samostatnom súbore, čo nám umožňuje efektívnejšiu a rýchlejšiu úpravu vzhľadu stránky. Táto separácia zlepšuje prístupnosť obsahu dokumentu, umožňujúc väčšiu flexibilitu a kontrolu pri špecifikácii charakteristík jednotlivých prezentovaných prvkov, umožňuje redukovať komplexnosť a opakovateľnosť v obsahovej štruktúre dokumentu. Ďalej umožňuje pre ten istý dokument vytvorený v danom značkovacom jazyku, aby bol prezentovaný v rozdielnych vzhľadoch, závisiac na použitom kaskádovom štýle.

Na externý štýlový predpis je možné odkázať popri prvku <*link*> aj príkazom *@import*. Tento spôsob nám umožňuje vytvárať do seba zanorené šablóny štýlov v externom súbore. Príkaz *@import* umožňuje obmedzenie štýlu na určitý typ média, a teda sa príkaz *@import* uplatní len na prehliadači akceptujúcom dané médium. Druhou možnosťou obmedzenia štýlového predpisu pre dané médium je použitie príkazu *@media*.

2.3.1.1 Canvas

Canvas (plocha) sa pri každom druhu média podľa normy CSS 2.1 chápe ako priestor kam sa má formátovaná štruktúra vykresliť. Samotný canvas je síce teoreticky nekonečný, ale vykresľovanie prebieha už na vopred vyhraničenú konečnú plochu. Vykresľovanie na canvas sa deje v prípade agentov s využitím konštánt šírky a výšky.[3]

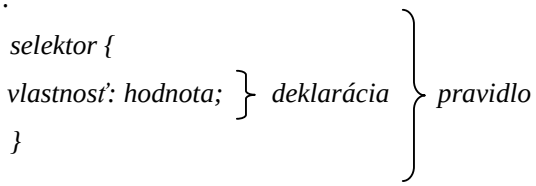
2.4 Definovanie syntaxe kaskádových štýlov

Ukážeme si jednoduchý popis gramatiky používanej pri každej z noriem CSS. Pričom uvedená syntax sa bude dodržiavať aj v budúcich verziách normy CSS (s potrebnými rozšíreniami).

Všetky vydania normy CSS (CSS 1, CSS 2, CSS 2.1) používajú syntax s rovnakým jadrom. Táto skutočnosť umožňuje užívateľským agentom parsovanie štýlových predpisov napísaných podľa normy CSS, ktoré v dobe vzniku daného užívateľského agenta ešte ani nemuseli existovať, pričom by daný agent časti štýlového predpisu, ktoré nedokáže spracovať zamietol. Táto vlastnosť umožňuje popri udržaní spätnej kompatibility rozvíjať a obohacovať normu CSS o nové vlastnosti.

2.4.1 Selektory

Definícia-štýlu :



Selektor je v najjednoduchšom prípade meno prvku. Komentáre sa dajú uplatniť v prípade, keď sú štýlové vlastnosti deklarované buď v externom **.css* súbore alebo v štýlovom predpise (stylesheet) v hlavičke stránky a majú podobu zátvoriek /* a */. Podľa konvencií uvedených v aktuálnej norme sa rozlišujú aj malé a veľké písmená.

Popri zmieňovanom jednoduchom selektore, v podobe mena prvku, pre ktorý sa majú štýlové vlastnosti uplatniť, poznáme aj zložitejšie podoby selektorov, akými sú:

- Viacnásobné selektory
- Kontextové selektory
- Atribútové selektory
- Univerzálne selektory

V prípade viacnásobného selektora môžeme tento chápať ako zoznam mien prvkov, pre ktoré sa majú dané vlastnosti uplatniť.

Štýl definovaný kontextovým selektorom sa uplatňuje pre prvky nachádzajúce sa v danom kontexte. Ako kontext sa rozumie výskyt prvku zanorene v tele iného prvku, pričom sa nemusí jednať len o jednorazové zanorenie. Pri uplatňovaní kontextového selektoru si tento môžeme predstaviť ako zoznam prvkov, medzi ktorými sa ale nenachádza žiaden operátor, a teda sú chápané ako zanorenie prvku v kontexte, od vonkajšieho smerom k vnútornému, pričom sa samotné štýlové vlastnosti uplatnia pre ten posledný prvok v zozname. Pri kontextovom selektore sa nemusí jednať a priame vnorenie, zoznam definujúci kontext prvku nemusí zodpovedať presnému popisu skutočného kontextového vnorenia, skutočné vnorenie môže obsahovať aj ďalšie prvky, ale dôležité je dodržanie poradia. Špecifickými prípadmi kontextových selektorov sú synovský a susedný selektor.

V prípade využitia synovského selektoru sa jedná o priame vnorenie. Operátor pre vyjadrenie vzťahu priameho vnorenia prvku je >.

Susedný selektor je tvorený pomocou využitia operátora +, znamená, že výskyty uvedených prvkov (spojených do jedného selektora práve operátorom +) musia byť v dokumente priamo vedľa seba.

Atribútový selektor nám umožňuje uplatnenie štýlových vlastností len pre prvky, ktoré obsahujú aj daný atribút, prípadne musí daný atribút obsahovať danú hodnotu.

Univerzálny selektor sa uplatňuje pre všetky prvky a je značený operátorom *.

2.4.2 Triedy

Kaskádové štýly nám umožňujú pre daný prvok definovať viacero štýlov, ktoré sa uplatňujú použitím pomenovaných tried. Konkrétnu triedu priradíme danému výskytu prvku použitím atribútu *class*, v ktorom uvedieme meno triedy pre daný výskyt prvku. V prípade, že chceme prvku priradiť viaceré

triedy, sa tieto uvedú v atribúte *class*, vymenované za sebou formou zoznamu a oddelené od seba čiarkou. Možné typy tried môžeme rozdeliť do nasledujúcich skupín:

- Regulárne triedy
- Pseudotriedy

V prípade regulárnych tried sa meno triedy udáva pomocou uvedenia mena triedy za bodku.

Pseudotriedy využívame v prípade, ak chceme upraviť štýl nie pomocou atribútov. V týchto prípadoch sa využívajú kľúčové slová, oddeľované od mien tried pomocou dvojbodky. Pseudotriedy bližšie definujú celý prvok a vyčleňujú ho z obecnej množiny prvkov jedného mena, podobne ako triedy. Kľúčové slová vyjadrujú určitú akciu alebo rozmer a väčšinou sú v angličtine. Môžu obsahovať pomlčku a nie sú case senzitívne.

2.4.3 Identifikátor ID

V prípade, že chceme definovať štýl pre prvok, je toto možné aj priradením štýlu pomocou atribútu *id*. V selektorovom výraze je takýto odkaz uvedený mriežkou # nasledovanou hodnotou atribútu *id*.

2.4.4 Obsahujúci blok

Ako obsahujúci blok daného elementu sa chápe jeho najbližší rodičovský blok, v ktorom je umiestnený. Pre jeho voľbu norma definuje niekoľko pravidiel, z ktorých sú najpodstatnejšie:

- Pokiaľ nie je daný element absolútne pozíciovaný, je jeho obsahujúcim blokom najbližší rodičovský blok elementu.
- Obsahujúci blok koreňového elementu jazyka vytvára priamo prehliadač.
- Pokiaľ je element pozíciovaný absolútne, jeho obsahujúcim blokom sa stáva najbližší rodičovský element s hodnotou vlastnosti *position* inou ako *static*. V prípade, že túto podmienku nespĺňa žiaden rodičovský element jeho obsahujúcim blokom sa stane základný obsahujúci blok.

2.5 Vlastnosti kaskádových štýlov

Vlastnosti kaskádových štýlov môžeme rozdeliť do nasledujúcich skupín:

- písmo: vzhľad písma je možné upravovať nastavením vlastností jeho veľkosti, roztiahnutia, štýlu, fontu či variantu písma.
- farba a pozadie: farby môžeme v CSS zadávať ako kľúčové slová, hexadecimálne alebo dekadické hodnoty.
- text: spôsoby definovania ako sa má text zarovnávať a prezentovať

- obdĺžniky a rozvrhnutie: pri kaskádových štýloch sa predpokladá, že každý jeden prvok bude vložený do obdĺžnikového priestoru.
- zoznam: položky v zoznamoch sa väčšinou zobrazujú ako blokové prvky s danou úvodnou značkou, pričom nám vlastnosti zoznamu umožňujú túto skutočnosť ovládať.
- klasifikácia značiek: tieto vlastnosti slúžia k určeniu, ako sa majú jednotlivé prvky a ich obsah klasifikovať.

2.6 Princípy návrhu CSS

CSS 2.1 ako aj CSS 2 a CSS 1 pred ňou, je založená na návrhových princípoch:

- Popredná a spätná kompatibilita: CSS 2.1 užívateľskí agenti sú schopní pochopiť aj štýlový predpis vytvorený zodpovedajúc norme CSS 1, tak ako aj agenti CSS 1, zvládajú prezeranie štýlového predpisu vytvoreného podľa normy CSS 2.1 s tým, že časti, ktorým nerozumejú sa zamietajú.
- Komplementárnosť k štruktúrovaným dokumentom: Štýlové predpisy doplňujú štruktúrované dokumenty (napr. HTML a XML aplikácie), poskytujúc im štylistické informácie, pričom by malo byť možné zmeniť štýlový predpis pre dokument bez zásahov do obsahovej časti dokumentu.
- Nezávislosť: Štýlový predpis je nezávislý na platforme, nástroji, či prehliadači, pričom ale norma CSS 2.1 už umožňuje špecifikovať rôzne druhy zobrazovania závisiac na skupine zobrazovacích prostriedkov, ktorým sa daná vlastnosť dá prideliť.
- Udržovateľnosť: Umožňuje nám ľahkú udržovateľnosť a jednoduché potrebné zmeny v štýle zobrazovania dokumentu v prípade, že je to potrebné, bez potreby zasahovania do obsahovej časti dokumentu.
- Jednoduchosť: CSS je jednoduchý, čitateľný a pisateľný jednoznačný štýlový predpis.
- Sieťový výkon: CSS poskytuje kompaktný popis ako zobrazovať obsah dokumentu
- Flexibilita: CSS sa môže uplatniť pre daný obsah viacerými spôsobmi. Kľúčovou vlastnosťou CSS je možnosť kaskádovania štýlových informácií určených v prednastavenom štýlovom predpise.
- Sýtosť: CSS poskytuje využitie širokého množstva uplatniteľných efektov zväčšujúc takto schopnosť fungovania webu ako vyjadrovacieho média. CSS povoľuje uplatnenie štýlov pri webových aplikáciách, aké poznáme len z desktopových aplikácií.
- Dostupnosť: Mnohé z vlastností CSS umožňujú tvoriť webové aplikácie viac prístupné pre užívateľov s handicapom.

2.7 Stručný popis normy CSS 2.1.

V tejto kapitole sa budeme zaoberať normou CSS 2.1., ktorá je nadstavbou momentálne najrozšírenejšej a prehliadačmi najviac podporovanej normy CSS 2. Podporuje štýly pre rôzne médiá, a teda má autor možnosť úpravy svojich dokumentov pre potreby internetových prehliadačov, pre tlačenie dokumentu alebo pre prezeranie na vreckových zariadeniach (napr. PDI).

CSS 2.1 opravuje niektoré chyby, ktoré sa vyskytovali vo verzii CSS 2. a pridáva do normy niektoré už rozšírené a používané rozšírenia, ktoré sa ale nestihli do predchádzajúcej verzie normy (CSS 2) zakotviť.

Znalosti uvedené v tejto kapitole sa opierajú o fakty prezentované v norme CSS 2.1. vydanej konzorciom W3C, odkiaľ sú čerpané aj uvedené príklady pre spracovávanie chýb pri parsovaní [15].

2.7.1 Porovnanie CSS 2.1 s verziou CSS 2

Komunita CSS sa podrobne zoznámila s normou CSS 2, ktorá sa úspešne presadila v tvorbe webových aplikácii a od jej prvotného uvedenia roku 1998 sa stala rozšíreným štandardom pri tvorbe vzhľadu webových stránok. Od uvedenia CSS 2 boli mnohé chyby, vyskytujúce sa v jazyku, opravené prostredníctvom rôznych publikácií.

Niektoré z týchto úprav budú spracované až chystanou normou CSS 3. Iné publikované chyby si ale vyžadovali včasnejšie riešenie, čo viedlo k vzniku CSS 2.1., ktoré sa snaží spracovať a riešiť situáciu prístupom:

- Zachovania kompatibility s tými časťami CSS 2, ktoré sú vysoko rozšírené a implementované.
- Zahrnutím všetkých publikovaných chýb týkajúcich sa CSS 2.
- V prípade, že sa implementácie vo vysokej miere líšia od špecifikácie CSS 2, sa tieto špecifikácie pre normu CSS 2.1. upravili, aby zodpovedali zaužívaným praktikám používania.
- Odstránenie špecifikácii z CSS 2, ktoré sa neujali, a sú teda nepoužívané.
- V prípade prvkov z normy CSS 2., ktoré sú už prekonané a budú v chystanej norme CSS 3 nahradené, sa do normy CSS 2.1. zahrnuli tieto nové verzie tak, že pre daný prvok, rys jazyka CSS už existujú.
- Pridanie malého množstva nových vlastnosti, ktoré sa ukázali ako potrebné pre lepšiu prácu s CSS 2.

V dnešnej dobe sa odporúča používanie normy CSS 2.1 kvôli širšej podpore medzi internetovými prehliadačmi a lepšej kompatibilite na pripravovanú normu CSS 3.

2.7.2 Model spracovania CSS 2.1

V tejto podkapitole sa budeme zaoberať prácou jedného z možných modelov užívateľských agentov pre spracovanie CSS (CSS 2.1 processing model). Uvedený model je konceptuálny príklad fungovania spracovania kaskádových štýlov, skutočné implementácie sa môžu líšiť. Užívateľský agent pracuje v nasledujúcich krokoch:

1. Rozparsuje zdrojový dokument a vytvorí stromovú štruktúru (document tree) . Takýto strom obsahuje všetky prvky dokumentu, pričom každý z nich má presne jeden rodičovský prvok, okrem koreňového prvku, ktorý rodičovský prvok nemá.
2. Identifikuje sa druh cieľového média. Pri kaskádových štýloch existuje možnosť preddefinovania na akom druhu média (obrazovka, výstup na tlačiareň, TV) a ako sa má dokument zobrazovať.
3. Vráti všetky asociácie štýlov, ktoré sa majú pre daný druh média uplatniť.
4. Anotuje sa každý prvok stromovej štruktúry priradením hodnoty(pre každý jej prvok), ktorý sa dá pre daný druh média uplatniť. Časť priraďovaných hodnôt sa získava pomocou príslušných formátovacích algoritmov pre dané cieľové médium.
5. Z anotovanej stromovej štruktúry dokumentu sa generuje formátovacia štruktúra. Táto je často vysoko podobná pôvodnej stromovej štruktúre dokumentu, ale môže sa aj odlišovať. Dôležitosť formátovacej štruktúry je daná jej vlastnosťou, že už nemusí mať stromovú štruktúru. Jej štruktúra závisí od druhu implementácie. Ďalšou výhodou tejto štruktúry je, že môže obsahovať podľa potreby aj viac prípadne menej informácii, než obsahovala stromová štruktúra.
6. Formátovaná štruktúra sa predá na cieľové médium (napr. zobrazí sa formátovaný dokument na ploche, alebo sa vytlačí výsledok atď.).

Zo spomenutých krokov sa vykonávanie krokov 2 až 5 deje podľa špecifikácií normy CSS 2.1, kroky 1 a 6 nie sú zahrnuté v norme CSS 2.1. Postup priebehu kroku 1 je popísaný v špecifikácii DOM (kapitola 4.).

2.7.3 Konformita užívateľského agenta

Nasledujúce body popisujú prispôsobenie sa užívateľských agentov norme CSS 2.1. Vo všeobecnosti by mal užívateľský agent používaný s normou CSS 2.1 zahŕňať nasledujúce špecifikácie:

1. Musí byť schopný rozpoznať jednotlivé médiá, typy noriem.
2. Pre každý zdrojový dokument sa musí pokúšať o vrátenie všetkých asociovaných štýlových predpisov, ktoré sú vhodné pre dané zobrazovacie médium. V prípade, že nedokáže vrátiť vyžadovaný štýlový predpis, musí zvoliť iný dostupný štýlový predpis pre zobrazenie dokumentu.
3. Musí parsovať štýlový predpis v súlade so špecifikáciou.

4. Pre každý prvok v stromovej štruktúre dokumentu (document tree) musí priradiť hodnotu pre každú aplikovateľnú vlastnosť podľa popisu vlastnosti a pravidiel určených v norme CSS 2.1.
5. V prípade, že zdrojový dokument prichádza s možnosťou striedať štýlové predpisy, musí užívateľský agent povoliť možnosť výberu, ktorý zo štýlov sa má aplikovať.
6. Užívateľský agent musí povoliť užívateľovi vypnúť vplyv autorovho štýlového predpisu. [15]

2.7.4 Pravidlá pre spracovanie chýb pri parsovaní

V niektorých prípadoch užívateľskí agenti musia ignorovať časť štýlového predpisu. Zo špecifikácie normy vyplýva, že užívateľský agent musí parsovať aj nelegálnu časť štýlového prepisu z dôvodu možnosti nájdenia jeho začiatku a konca, ale mimo toho sa má agent správať, ako by tam daná nelegálna časť vôbec nebola. Norma CSS 2.1 rezervuje pre budúce použitie všetky kombinácie *vlastnosť*: *hodnota* a kľúčové slová začínajúce zavináčom (napr. *@-kľúčové slovo*) ako aj všetky identifikátory, ktoré neobsahujú na svojom začiatku pomlčku alebo podčiarknutie (*-identifikátor*, *_identifikátor*). Implementácie pracujúce s CSS musia ignorovať všetky takéto kombinácie, vynímajúc v budúcnosti zavedené rozšírenia CSS.

Pre zaistenie, že v budúcnosti bude možné pridávať nové vlastnosti a nové hodnoty pre už existujúce vlastnosti, je potrebné, aby sa užívateľskí agenti riadili nasledujúcimi pravidlami pre daný prípad:

2.7.4.1 Neznáma vlastnosť

Užívateľskí agenti musia ignorovať všetky deklarácie s neznámou vlastnosťou. Napríklad v prípade, že náš štýlový predpis obsahuje:

```
h1 { color: red; rotation: 70minutes }
```

musí tento užívateľský agent spracovať, ako keby štýlový predpis obsahoval len vlastnosť:

```
h1 { color: red }
```

2.7.4.2 Nepovolená hodnota pre vlastnosť

Užívateľskí agenti musia ignorovať všetky deklarácie s neznámou hodnotou pre danú vlastnosť. Napríklad v prípade, že náš štýlový predpis obsahuje:

```
img { float: left }       /* správne podľa  CSS 2.1 */
img { float: left here }  /* "here" nie je povolená hodnota vlastnosti  'float' */
img { background: "red" } /* kľúčové slová nesmú byť v úvodzovkách */
img { border-width: 3 }   /* pre hodnoty vyjadrujúce dĺžku je potrebné uviesť
jednotku */
```

Parser pracujúci podľa normy CSS 2.1 by vyhodnotil prvé pravidlo a ignoroval by všetky zvyšné nepovolené vlastnosti, správal by sa akoby štýlový predpis obsahoval len vlastnosť:

img { float: left }

2.7.4.3 Zdeformované deklarácie

Užívateľský agent musí spracovávať neočakávané tokeny, na ktoré narazí počas parsovania danej deklarácie až po jej koniec, pričom musí overovať pravidlá pre párny počet znakov: (), [], {}, "". Uvedené príklady nám ukazujú najčastejšie spôsoby chybných deklarácií:

p { color:green }

p { color:green; color } / chýba tu dvojbodka ':', pred hodnotou */*

p { color:green; color: } / chýba hodnota */*

p { color:green; color{;color:maroon} } / neočakávané tokeny { } */*

2.7.4.4 Nepovolené kľúčové slovo začínajúce zavináčom

Užívateľský agent musí ignorovať neplatné kľúčové slovo začínajúce zavináčom (napr. *@-kľúčové slovo*) spolu so všetkým, čo za ním v deklarácii nasleduje až po nasledujúcu bodkočiarku alebo blok ({...}) vrátane, podľa toho, čo sa vyskytne skôr. Napríklad v prípade výskytu nasledujúcej deklarácie v danom štýlovom predpise:

```
@three-dee {
  @background-lighting {
    azimuth: 30deg;
    elevation: 190deg;
    }
 h1 { color: red }
}
h1 { color: blue }
```

Agent zistí, že kľúčové slovo začínajúce zavináčom '*@three-dee*' nie je uvedené v norme CSS 2.1, a teda bude aj spolu s obsahom k nemu patriacim ignorované. Užívateľský agent, pracujúci podľa CSS 2.1normy, po ignorovaní chybne deklarovanej časti vráti:

h1 { color: blue }

2.7.4.5 Neočakávaný koniec štýlového prepisu

Užívateľskí agenti musia ukončiť všetky otvorené konštrukcie deklarácií na konci štýlového predpisu. Napríklad:

h1 { color: blue

bude spracovaný rovnako, ako bola spracovaná aj deklarácia:

h1 { color: blue }

2.7.4.6 Neočakávaný koniec reťazca

Užívateľskí agenti musia ukončiť všetky reťazce, keď dorazia na koniec riadku a zamietnu konštruktér (deklarované pravidlo), ktorý daný reťazec obsahoval, a teda v prípade deklarácie:

```
p {
color: green;
font-family: 'Courier New Times  /* chýba zakončenie reťazca: ';  */
color: red;
 color: green;
}
```

bude uvedené do podoby:

```
p {
color: green;
 color: green;
}
```

a to z dôvodu, že druhá deklarácia (agentom chápaná od začatia deklarácie font-family až po bodkočiarku ukončujúcu deklarovanie 'color: red') nie je validná, a teda sa zamietne. [15]

2.8 Nedostatky CSS

Väčšina problémov pri práci s kaskádovými štýlmi sa dá v skutočnosti odvodiť na používaný prehliadač, ktorý nie vždy podporuje v úplnej miere špecifikácie udávané normou CSS, prípadne na chyby vzniknuté nesprávnym definovaním daných CSS vlastností.

2.8.1 Nedostatky normy CSS 2.1

Medzi nedostatky normy CSS 2.1. ([6] [15]) môžeme zaradiť:

- Limitovaná kontrola vertikálneho zarovnania

  Kým umiestňovanie prvkov v horizontálnej polohe je ovládateľné, ich vertikálne umiestňovanie je často neintuitívne, zložité či dokonca nezvládnuteľné. Jednoduché úlohy ako vertikálne vycentrovanie prvku vyžadujú pre zrealizáciu komplikované a neintuitívne pravidlá.

- Selektory nedokážu stúpať

  CSS neponúka žiadnu možnosť výberu rodiča alebo predchodcu pre prvok, ktorý by spĺňal isté kritériá. Chýba vyššie vyvinutá schéma selektorov (ako napríklad jazyk XPath,

využívaný pre adresovanie častý XML dokumentov), ktorá by umožňovala sofistikovanejšie štýlové predpisy (stylesheet). Vývojári aktuálny stav odôvodňujú nepripravenosťou prehliadačov.

- Jedna bloková deklarácia nemôže dediť od druhej

  Dedenie štýlov je vykonávané prehliadačom na základe obmedzení DOM hierarchie prvkov, a špecifickosti selektorov pravidiel. Len užívateľ daného bloku môže na ne odkazovať začlenením mien tried do atribútu triedy v DOM prvku.

- Absencia výrazov

  Nie je možnosť špecifikovať hodnoty vlastníctva ako jednoduché výrazy.

- Nedostatok ortogonality

  Často skončia viaceré vlastnosti popisujúce tú istú činnosť. Napríklad vlastnosti: *position*, *display* a *float* sa využívajú pre špecifikáciu umiestnenia modelu, pričom sa ale väčšinou nedajú rozumne kombinovať. Napríklad by sa prvok *display: table-cell* nemohol znášať, alebo mať určenú pozíciu *position: relative*, a tak isto by sa prvok *float: left* nemal meniť v reakcii na zmeny nastavenia *display*.

- Neočakávaný kolaps okrajov

  je často problém, ktorého odstránenie je bez vzniku nechcených vedľajších efektov často nemožné.

- Chýbajúce viaceré pozadia pre jeden prvok

  Stránky s náročným grafickým vzhľadom často vyžadujú viaceré pozadia pre každý prvok, ale CSS podporuje len jedno.

- Chýbajúca možnosť špecifikovania rôznej veľkosti fontov pre rôzne rodiny fontov

  V prípade typu písma je možné nadefinovať viaceré typy písma, fontov, z dôvodu, že by nami prvotne zvolený typ písma nebol podporovaný niektorými z prehliadačov. Toto riešenie ale chýba v prípade veľkosti písma, a teda nemáme možnosť nastaviť pre jednotlivé alternatívy k pôvodnému písmu ich veľkosť, akou sa majú zobrazovať. Samotný problém vzniká v tom, že napr. pri veľkosti 12 pixlov sa bude úplne ináč renderovať písmo typu *Verdena*, než *Garamond*.

- Chýbajúca možnosť kontroly stĺpcov

  V CSS nie je možnosť jednoduchého vytvárania stĺpcov. Nie je umožnené jednoducho špecifikovať niečo v štýle: „Chcem mať vedľa seba tri stĺpce nezávislé na veľkosti okna webového prehliadača“. Riešenie tohto problému je síce možné, ale dá sa povedať nie príliš intuitívnym spôsobom, pomocou vytvorenia stĺpcov podobného vzhľadu pomocou *float* a *clear*, ale tento prístup nesie so sebou možnosť stať sa zdrojom množstva nasledujúcich problémov.

- Stĺpce rovnakej výšky

V prípade, že máme dva stĺpce umiestnené vedľa seba sa nám vyskytne problém ako zabezpečiť, aby tieto stĺpce mali rovnakú výšku. Pri práci s tabuľkami sa v prípade, že sa nám v niektorej z buniek tabuľky, v danom stĺpci a v danom riadku vyskytol pridlhý text sa tá bunka spolu so zvyškom daného riadku tabuľky prispôsobila veľkosťou potrebám pre daný text. Teda nenarástla len výška jednej bunky, ale celého riadku tabuľky. Riešenie tohto typu v CSS, napríklad pri použití prvkov *div* pre vytvorenie stĺpcov, nie je možné. Nevieme vytvoriť vzťah medzi jednotlivými stĺpcami.

2.8.2 Navrhované rozšírenia CSS

Popri momentálne používanej a platnej norme CSS 2.1 sa už dlhšiu dobu pripravuje nová norma CSS 3, ktorá má rozšíriť vlastnosti a možnosti ponúkané aktuálne platnou verziou. V tejto podkapitole sú uvedené niektoré pripravované rozšírenia pripravované s normou CSS 3 konzorciom W3C.

Jednou z najpodstatnejších zmien v norme CSS 3 bude–zavedenie modularizácie, delenia, kaskádových štýlov do viacerých dokumentov. Výhodou tohto rozšírenia bude jednoduchšie definovanie podpory danej aplikácie kaskádovými štýlmi. Jednotlivé moduly sa vyvíjajú relatívne samostatne, tvoriac aj samostatné dokumenty novej pripravovanej normy.

Obohatené budú aj selektory. Modul, ktorý ich popisuje, obsahuje nové selektory, pseudotriedy, pseudoelemnty ako aj možnosť špecifikácie menného priestoru (namespace).

Pripravovaná nová norma CSS 3 zatiaľ nie je schválená ani podporovaná dnešnými internetovými prehliadačmi, aj keď niektoré z navrhovaných rozšírení sú podporované Mozilla Firefoxom, v menšom množstve aj novým Internet Explorerom 7, ale kvôli nejednoznačnosti v podpore prehliadačmi a chýbajúcemu koncovému a schválenému dokumentu o novej norme CSS 3, sa jej používanie naďalej neodporúča. [11]

3. Problematika rozmiestnenia objektov na stránke

V tejto kapitole budú prezentované rôzne prístupy a problematiky týkajúce sa rozmiestňovania objektov na webových stránkach. Cieľom skúmania tejto problematiky a jednotlivých spôsobov umiestňovania objektov na stránke je vytvorenie návrhu pre efektívnejší spôsob rozloženia objektov na stránke.

3.1 Zastarané spôsoby umiestňovania objektov

CSS rozmiestnenie objektov vzniklo s cieľom nahradiť zastarané spôsoby umiestňovania objektov na stránky, ako sú napríklad metóda delenia stránky na rámy (frames), ktorá sa už v dnešnej dobe pre rozmiestnenie objektov neodporúča, alebo metóda tabuliek, pri ktorej sa objekt umiestňuje do buniek tabuliek vytvorených ako kostra stránky.

3.1.1 Metóda pomocou frame

Metóda takzvaných rámcov *frame*, je pri pohľade na vývoj dnešných webových stránok zastaraným a nepoužívaným spôsobom, ktorý bol vo svojej funkčnej podstate úplne nahradený sofistikovanejšími spôsobmi umiestňovania objektov na stránku s využitím tabuliek (tiež už nie aktuálny spôsob umiestňovania), alebo pomocou CSS rozmiestňovania objektov.

Samotná metóda rámcov mala svoj princíp umiestňovania objektov na stránku založený na delení pôvodnej stránky do viacerých obdĺžnikových objektov, rámcov, pričom sa do každého z nich mohol načítať obsah inej stránky [5].

Využitie rámcov frame má pri tvorbe webových aplikácií okrem pozíciovania aj mnohé iné spôsoby využitia, ktoré sa aj dnes využívajú

3.1.2 Tabuľky

Tabuľka sa pri tvorbe stránok využíva na tvorbu tabuliek v klasickom slova zmysle (napr. pre reprezentáciu tabuľkových hodnôt), ale aj ako spôsob tvorenia kostry pre rozloženie objektov na stránke, pričom sa vytvorí akási tabuľka zobrazujúca stránku delenú na jej jednotlivé časti. Takto vytvoreným bunkám sa potom priradí do obsahu daný objekt, čím sa zabezpečí presné pozíciovanie objektu vzhľadom k ostatným objektom na stránke, vyplňujúcim ostatné bunky tabuľky.

Tvorba webových stránok v dobe pred zavedením CSS by bola bez tvorby tabuliek nepredstaviteľná. Tabuľky predstavovali jediný spôsob tvorby viacstĺpcového rozloženia stránky.

Využívanie tabuliek ako spôsob pozíciovania objektov na stránke je často zaužívané aj v dnešných implementáciách. Tento spôsob umiestňovania objektov na stránku je v dnešnej podobe používania ovplyvnený kaskádovými štýlmi, ktoré sa využívajú pre definovanie niektorých vlastností tabuľky ako napr. vlastnosti *width* či *height* pre definovanie šírky či výšky.

Napriek ich častému využívaniu sa používanie tabuliek neodporúča, z dôvodu zložitosti úprav, neprehľadnosti kódov a malo by byť pri implementácii rozmiestňovania objektov na stránke úplne nahradené CCS rozmiestňovaním objekt. ([17], [18]).

3.2 CSS rozmiestňovanie objektov

CSS rozmiestňovanie objektov je podrobne popísané v štandarde kaskádových štýlov CSS 2.1. (pozri [15]) a je podporované majoritnou väčšinou moderných internetových prehliadačov.

CSS rozmiestňovanie objektov je v dnešnej dobe najrozšírenejší a najefektívnejší spôsob umiestňovania objektov na stránkach. Tento spôsob umožňuje umiestňovať akýkoľvek objekt (ako objekt chápeme napr. tabuľku, obrázok, text, atď.) kamkoľvek na stránku.

Môžeme rozlíšiť základné druhy rozmiestňovania objektov, a to pozíciovanie *absolútne* a *relatívne* a rozmiestňovanie objektov využitím vlastnosti *float* a *clear*. [2]

3.2.1 Absolútne pozíciovanie

Absolútne pozíciovanie umiestňuje objekt do stránky na zadané súradnice bez ohľadu na objekty v jeho okolí, umožňuje nám teda umiestniť objekt kam chceme. Pri zadávaní pozície konkrétneho objektu sa využívajú hlavne vlastnosti *top* a *left* (prípadne *right* a *bottom*), ktoré sa využijú pre určenie vzdialenosti objektu od horného a ľavého okraja stránky [15]

3.2.2 Relatívne pozíciovanie

V prípade relatívneho pozíciovania sa jedná skôr o posunutie objektu, než jeho pozíciovanie. V tomto prípade element (pozíciovaný objekt) nie je vyzdvihnutý z toku dokumentu. Vo všeobecnosti sa odporúča uprednostniť pred relatívnym pozíciovaním objektu absolútne pozíciovanie.[9]

3.2.3 Rozmiestňovanie objektov pomocou *float* a *clear*

Vlastnosť *float* vytvára z bežných objektov objekty plávajúce. Takýto objekt je potom zobrazovaný ako obdĺžnik odsunutý k okraju a ostatný obsah ho oblieva. Možné hodnoty float sú *none* a hodnoty *left* či *right* určujú polohu objektu. Tento spôsob má ale isté nedostatky pri pokuse o jeho uplatnenie pre väčšie množstvo objektov, pričom je zložité dosiahnuť stav, že by sa tieto objekty v prípade nerovnakej veľkosti vedeli rozložiť formou tabuľky (horizontálne zarovnanie riadkov).

Vlastnosť *clear* určuje, kde sa má prvok zobraziť, ale až pod plávajúcim prvkom (prvok umiestnený pomocou *float*). [1]

3.3 Nedostatky v CSS rozmiestňovaní objektov

CSS rozmiestňovanie objektov je momentálne najefektívnejší a odporúčaný spôsob definovania rozloženia objektov na stránke. Umožňuje súradnicové definovanie presnej, dopredu určenej polohy každého objektu, definovanie jeho veľkosti a jeho správania sa v dokumente voči ostatným objektom ako aj voči samotnému dokumentu. Možnosť určenia reakcie objektu na ovplyvnenie jeho veľkosti či polohy v prípade zmeny veľkosti dokumentu.

Napriek svojim mnohým výhodám, v porovnaní so zastaranými spôsobmi, prináša CSS rozmiestňovanie objektov so sebou aj isté nevýhody, akou je nie vždy intuitívne definovania polohy, nedostatok ortogonality (zmieňované v kapitole 2), ako aj chýbajúca možnosť jednoduchého definovania viacerých objektov v dokumente.

V prípade definovania viacerých objektov rozložených na stránke je potrebné pri definovaní ich pozície v dokumente zohľadniť a zakalkulovať do výpočtov polohy a veľkosti ostatných objektov v dokumente. Napr. ak máme rozdeliť obsah stránky (prípadne iného nadradeného objektu, možné pri vnorení), medzi štyri objekty v podobe dvoch riadkov po 2 objektoch, môžeme ich polohy definovať v štýlovom predpise nasledovne:

```
#obj1 { top: 0px; left: 0px; width: 40px ; height: 30px; }
#obj2 { top: 0px; left: 40px; width: 40px ; height: 30px; }
#obj3 { top: 30px; left: 0px; width: 40px ; height: 30px; }
#obj4 { top: 30px; left: 40px; width: 40px ; height: 30px; }
```

Na príklade vidieť, že napr. pre určenie polohy objektu číslo 4 (obj4), bolo potrebné poznať a počítať so súradnicami určujúcimi veľkosť a polohu ostatných objektov. Objekt číslo 4 sa nachádza pod objektom číslo 2 (obj2) a vedľa objektu číslo 3 (obj3), a teda výška a šírka týchto objektov nám určuje súradnice pre polohu nášho objektu číslo 4.

Komplikovanosť a náročnosť výpočtov súradníc rastie súbežne s počtom rozmiestňovaných objektov na stránke, (napr. vytváranie formulára pre registráciu zákazníka, kde by bolo potrebné pre určenie polohy objektu v ôsmom rade spočítať celkovú veľkosť prvých šiestich riadkov).

Ďalším z nedostatkov CSS rozmiestňovania objektov je chýbajúca možnosť intuitívneho a jednoduchého definovania rozloženia objektov do stĺpcov. Norma CSS 2.1 neponúka žiadne uspokojujúce riešenie ako vytvoriť definíciu tabuľkového typu, kde by bolo možné určiť vopred daný počet prednastavených stĺpcov deliacich stránku alebo iný im nadriadený objekt pomerovo alebo staticky na menšie jednotky, na stĺpce, do ktorých sa potom budú môcť objekty umiestňovať.

Nedostupnosť jednoduchej metódy dosiahnutia spomínaného rozloženia tabuľkového typu je jedným z hlavných dôvodov používania tabuliek pre vytvorenie kostry na rozloženie objektov na stránke.

4. Objektový model dokumentu DOM

4.1 Úvod do objektového modelu dokumentu

DOM, objektový model dokumentu (Document Object Model), predstavuje štandardizované programátorské rozhranie pre prístup k štruktúre a udalostiam dokumentu. Objektový model dokumentu využíva koncepciu objektovo orientovaného programovania, pričom ale nie je viazaný na konkrétny programovací jazyk.

Jedná sa o takzvané aplikačné programové rozhranie (API - aplication programming interface), ktoré definuje všeobecný štandard pre prístup k akémukoľvek platnému dokumentu HTML a k správne skonštruovanému dokumentu XML. Je to užitočný nástroj, vďaka ktorému je možné jednoducho pristupovať k častiam HTML či XML dokumentu, s možnosťou tieto upravovať a mazať, či pridávať nové vlastnosti. [16] [3]

4.2 Vznik objektového modelu dokumentu

Samotná potreba vytvorenia objektového modelu dokumentu vznikala po vzniku dynamického HTML, ktorý v sebe kombinoval kaskádové štýly, HTML, a JavaScript za účelom možnosti dynamického pracovania s dokumentmi. Ideou objektového modelu dokumentu bolo vytvorenie zjednodušenej možnosti pre dynamické pracovanie s dokumentom na strane užívateľa. Postupne začali vo webových prehliadačoch ako internet Explorer a Netscape Navigator vznikať rozhrania, ktoré mali umožniť skriptom na strane klienta pristupovať k jednotlivým častiam štruktúry zobrazeného dokumentu a tieto v prípade potreby do istej miery aj upraviť. Problémy takto vzniknutých rozhraní pre prístup k častiam štruktúry dokumentu ale neboli nijako koordinované a kontrolované, teda chýbala kompatibilita medzi jednotlivými druhmi prehliadačov.

S cieľom skoordinovať a zjednotiť tieto prístupové metódy a rozhranie pre prácu so štruktúrou zobrazeného dokumentu bol vytvorený štandard objektového modelu dokumentu (Document Object Model), ktorého spravovanie má na starosti konzorcium W3C. Po vytvorení jednotného štandardu sa umožnilo používanie vytvorených skrípt jednotne vo väčšine webových prehliadačov.

4.2.1 Väzby objektového modelu dokumentu na programovací jazyk

Už spomínanou hlavnou výhodou štandardu objektového modelu dokumentu je jeho nezávislosť na programovacom jazyku, možnosť implementovať ho v ľubovoľnom programovacom jazyku, ktorý podporuje objektovo orientované programovanie.

4.3 Dostupné verzie špecifikácie objektového modelu dokumentu

Prvá verzia špecifikácie bola vytvorená roku 1998. Postupom času sa samozrejme táto pôvodne vytvorená verzia rozširovala, čo viedlo k vytvoreniu viacerých úrovní štandardu, pričom každá z úrovní reflektuje rozšírenie pôvodného štandardu o nové vlastnosti. Pri rozširovaní pôvodného štandardu sa pri každej zo vzniknutých úrovní zachovala spätná kompatibilita. Do dneška existujú štyri úrovne štandardu značené DOM Level 0 až DOM Level 3.

- **DOM Level 0:**

V prípade nultej úrovne sa ešte nejedná o štandard, vznikla za účelom zjednotenia rôznych spôsobov prístupu k štruktúre zobrazovaného dokumentu a rôznych rozhraní implementovaných v jednotlivých druhoch prehliadačov. Táto úroveň slúžila ako základný stavebný kameň pre tvorbu prvého štandardu zaoberajúceho sa s objektovým modelom dokumentu.

- **DOM Level 1:**

Prvá úroveň značí prvý vydaný štandard zaoberajúci sa objektovým modelom dokumentu z roku 1998. Štandard definuje základnú triedu pre popis dokumentu značenú Core, ako aj rozšírenia pre popis HTML dokumentov. Prvá úroveň do seba zahŕňa aj spätnú kompatibilitu s nultou úrovňou.

- **DOM Level 2:**

Druhá úroveň štandardu predstavuje rozšírenie prvej úrovne pri zachovaní spätnej kompatibility. Špecifikácia bola obohatená o ďalšie časti. Rozšírila sa základná trieda Core ako aj prostriedky pre popis HTML dokumentu o nové vlastnosti, pribudlo rozhranie pre prístup k definícii zo štýlu dokumentu so zameraním na kaskádové štýly CSS (značené Style), rozhranie pre spracovávanie udalostí (značené Events), ako aj rozšírenie rozhrania pre prechod stromom dokumentu a definovania úsekov dokumentu.

- **DOM Level 3:**

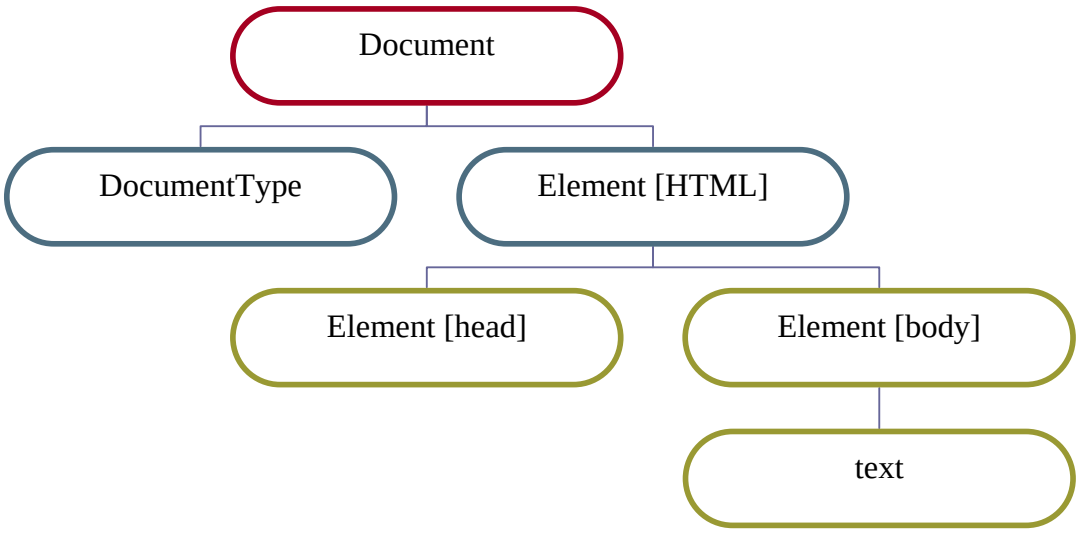
Štandard tretej úrovne bol zverejnený roku 2004 a je aj momentálne platným štandardom pre prácu s objektovým modelom dokumentu. Popri zachovaní spätnej kompatibility s predchádzajúcou úrovňou sa štandard obohatil o ďalšie vlastnosti, z ktorých je vhodné vyzdvihnúť možnosť validácie modelovaného dokumentu.

Popri uvedených úrovniach existujú aj iné implementácie objektového modelu dokumentu, značené ako podmnožiny uvedených špecifikácií. Pri používaní objektového modelu v kombinácii s konkrétnym programovacím jazykom je potrebné overiť, ktoré časti a rozhrania štandardu sú pre daný jazyk implementované. V dnešnej dobe väčšina implementácií zodpovedá štandardu DOM Level 2, toto je aj úroveň, s ktorou sa budeme podrobnejšie zaoberať. [16]

4.4 Základné dátové typy

4.4.1 DOM Tree

Dokument je chápaný objektovým modelom, ako usporiadaný strom popisujúci štruktúru dokumentu, pričom každý z uzlov môže mať nula až n podstromov, pri ktorých je dôležité aj ich poradie. Uzly v strome sa tvoria objektmi rôznych tried a reprezentujú rôzne štruktúry v dokumente.[3]



Obr. 4.1: DOM tree, zdroj [3]

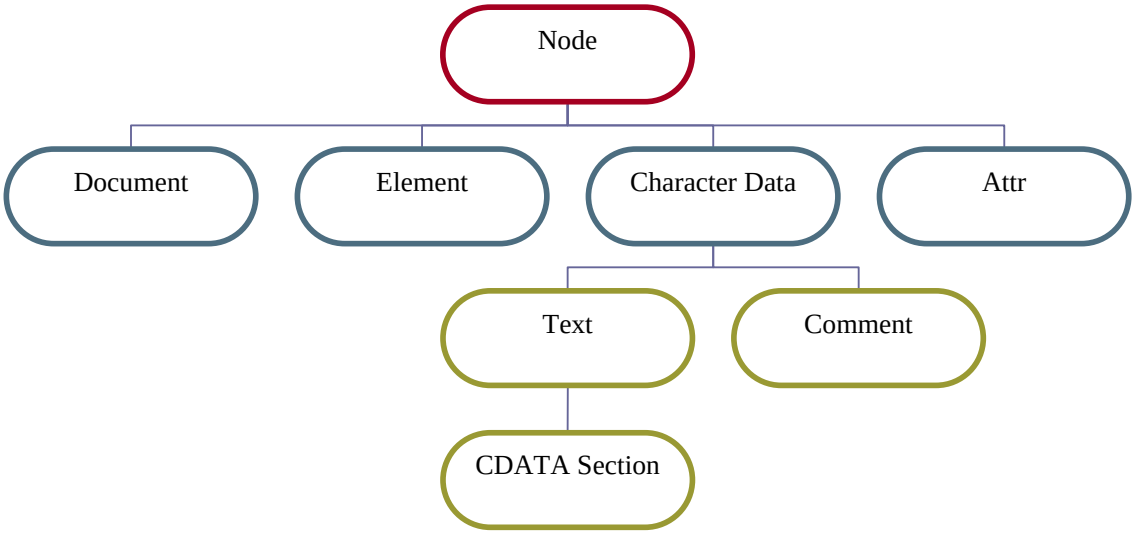
Na obrázku 4.1 je jednoduchý príklad stromovej štruktúry, nazývanej DOM tree, jednoduchého HTML dokumentu, podľa štandardu objektového modelu dokumentu, zodpovedajúceho nasledujúcemu pseudokódu:

```
<!DOCTYPE …>
<html>
  <head>
    …
  </head>
  <body>
    Text…
  </body>
</html>
```

Z obrázku 4.1. je vidieť, že uzly stromu sú objektmi rôznych typov (ale súčasne sú aj typu Node). Koreňovým uzlom je uzol typu *Document* reprezentujúci dokument ako celok, obsahujúci dve základné štruktúry

- DocumentType: - je objektom triedy obsahujúcim definíciu typu dokumentu
- Element [HTML]: - je koreňovým elementom <html> , ktorý v sebe zahŕňa obsah celého dokumentu, reprezentovaného objektom triedy Element.

Všetky triedy objektov, ktoré môžu tvoriť uzly stromu objektového modelu dokumentu, sú odvodené od spoločnej nadtriedy Node. Na obrázku 4.2. je zobrazená hierarchia dedičnosti medzi základnými triedami objektov, použiteľných ako uzly stromu:



Obr. 4.2: hierarchia dedičnosti medzi základnými triedami objektov, zdroj [3]

Objekt triedy *Document* tvorí vždy koreňový uzol stromu objektového modelu dokumentu. A každý z uzlov stromu má vlastnosť *ownerDocument* obsahujúcu odkaz na koreňový uzol typu Document.

Element je najčastejšie sa vyskytujúcim uzlom, reprezentujúcim ľubovoľný element HTML alebo XML, vždy má meno, prípadne aj atribút.

Trieda CharacterData sa delí na podtriedy Text, ktorá slúži pre reprezentáciu textového obsahu elementu a podtrieda Comment obsahuje vložený komentár.

Trieda CDATA Section sa využíva v prípade vloženia XML sekcie textu.

Trieda Attr reprezentuje atribút elementu, pričom objekty tejto triedy už netvoria samostatné uzly stromu, sú priradené elementom. Rozhranie tejto triedy umožňuje zistiť hodnoty daného atribútu, a či bola explicitne definovaná dokumentom, vieme zistiť aj to, ktorému objektu typu Element daná vlastnosť patrí.

4.4.1.1 Atribúty Elementu

Uzly typu Element zodpovedajú vždy HTML alebo XML elementom, a môžu teda mať atribúty, ku ktorým je možné pristupovať pomocou vlastnosti attribute, obsahujúcej kolekciu objektov typu Attribute. K atribútom sa môže pristupovať na základe znalosti ich mena. Kolekcia obsahujúca objekty typu Attribute, je odvodená od triedy Node. Metódy tejto triedy môžeme použiť pre zisťovanie hodnoty, prideľovanie či modifikovanie atribútu.

4.4.2 Rozhrania odvodených tried

Okrem rozhrania triedy Node môžeme využívať aj rozhrania tried od nej odvodených, a teda využívať aj ich špecializované vlastnosti a metódy. V prípade triedy Element môžeme využívať vlastnosti *tagName* obsahujúce meno HTML alebo XML značky tvoriacej daný element. Užitočná metóda je *getElementsByTagName*(), ktorá nám vracia indexovanú kolekciu všetkých podriadených elementov daného elementu.

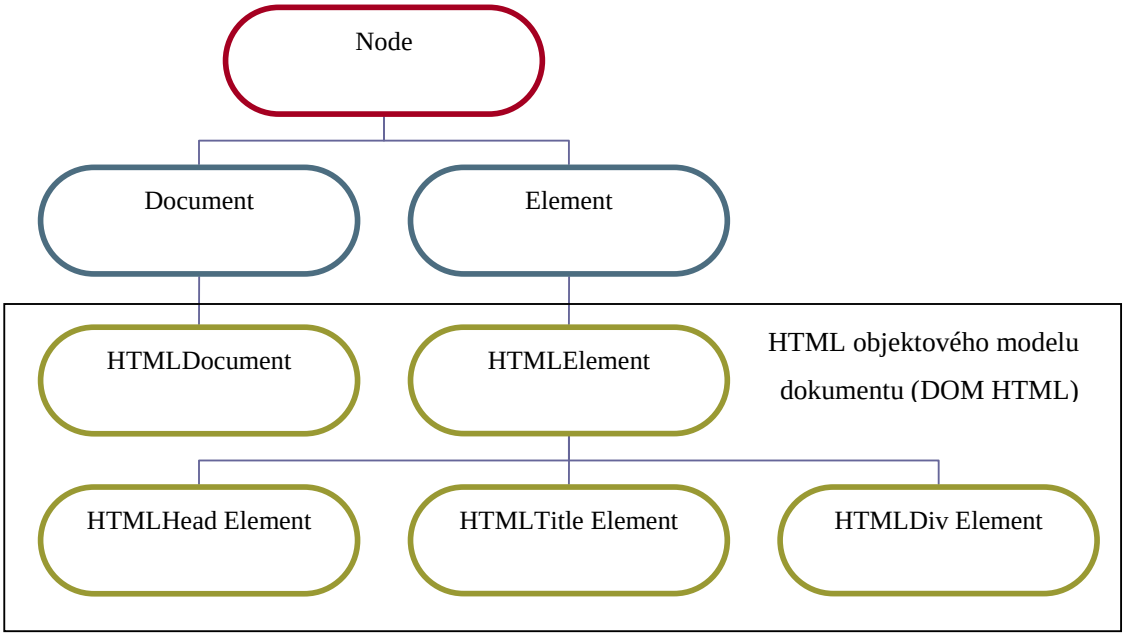
Koreňová trieda Document nám ponúka metódy pre získavanie niektorých podstromov definícií dokumentu alebo koreňového elementu. Máme k dispozícii metódy *getElementsByTagName()* používané aj v triede Element, alebo metódu *getElementById*(), umožňujúcu získať konkrétny element identifikovaný jednoznačne identifikátorom. Ako jednoznačný identifikátor sa chápe hodnota atribútu, deklarovaná ako atribút typu ID. U HTML sa atribút označuje ako *id*.

4.4.3 Špecifikácia HTML objektového modelu dokumentu (DOM HTML)

Definuje typy objektov odvodených od triedy Document a Element, modelujúce priamo dokument v jazyku HTML a jeho elementy. Kým základný objektový model dokumentu je určený pre modelovanie ľubovoľného dokumentu, táto špecifikácia (DOM HTML) obsahuje triedy pre

modelovanie konkrétnych HTML elementov, umožňujúce efektívnejšie pracovať so špecifickými vlastnosťami ako atribúty alebo podstromy konkrétnych HTML elementov.

HTML objektový model dokumentu rozširuje hierarchiu uzlov nasledujúcim spôsobom:



Obr. 4.3: DOM HTML, zdroj [3]

Okrem zobrazených tried sú definované aj triedy odvodené od HTMLElement, slúžiace pre reprezentáciu všetkých netriviálnych elementov jazyka HTML, ktoré buď obsahujú ďalšie špecializované elementy, alebo môžu mať aj iný ako generický atribút.

4.4.3.1 HTML Dokument

Trieda HTMLDocument bola definovaná pre modelovanie špecifických vlastností HTML dokumentu. Objekt tejto triedy je odvodený od koreňového uzlu stromu objektového modelu HTML dokumentu a obsahuje aj nové vlastnosti ako napr.: DOMString title; obsahujúcu titulok dokumentu definovaný elementom <*title*> [16].

4.4.3.2 HTML Element

Trieda HTMLElement slúži pre reprezentáciu ľubovoľného elementu HTML dokumentu. Oproti triede Element obsahuje mnohé nové vlastnosti zodpovedajúce atribútom, ktoré je možné špecifikovať pre ľubovoľný element. Tieto vlastnosti sú pomenované podľa názvu týchto elementov ako napr. *id*, *title*, *dir*, jedinou výnimkou je atribút *class*, ktorému zodpovedá vlastnosť *className*.

Pre všetky HTML elementy, ktoré môžu mať ďalšie atribúty, alebo môžu obsahovať špeciálne vnorené podštruktúry elementov, sú definované triedy odvodené HTMLElement, ako napr. trieda HTMLTableElement, reprezentujúca tabuľku vytvorenú HTML elementom <table>. [16][3].

4.5 Význam objektového modelu pre JavaScript

Objektový model dokumentu predstavuje štandard práce s HTML a XML dokumentmi, ktorého rozhranie je implementované vo všetkých v dnešnej dobe rozšírených internetových prehliadačoch (Internet Explorer, Mozilla Firefox, Opera). Tento fakt nám umožňuje obsluhovať daný dokument s využitím objektového modelu dokumentu pomocou JavaScriptu. Vďaka faktu, že sa pri objektovom modeli dokumentu jedná o jediný štandard, podporovaný až na menšie odlišnosti rovnako všetkými prehliadačmi, je možné písať JavaScript spôsobom, že bude mať jednotný kód, ktorý bude rovnako funkčný vo väčšine moderných prehliadačov. [16]

5. Rozloženie prvkov v jazyku Java

Kapitola sa zaoberá riešením rozloženia prvkov v jazyku Java a možnosťami uplatnenia týchto princípov v prípade rozloženie prvkov na webových stránkach.

V prípade jazyka Java je najvhodnejšia voľba pre rozmiestňovanie prvkov použitie niektorého zo správcov rozloženia (layout manager).

5.1 Správcovia rozloženia

V Jave sú dostupní viacerí správcovia rozloženia, v štandardných balíkoch (*java.awt* a *javax.swing*), medzi najpoužívanejšie patria:

- **FlowLayout**

  FlowLayout (správca pre plávajúce rozloženie), usporadúva komponenty podobne akoby išlo o jednotlivé znaky v texte. Začne napr. v ľavom hornom rohu a pokračuje ďalej v riadku, a v prípade, že došlo miesto prechádza do ďalšieho riadku. Umožňuje nastavenie medzier medzi komponentmi ako aj vodorovné zarovnanie.

- **GridLayout**

  GridLayout (rozloženie do mriežky), je založený na princípe usporiadania komponentov do virtuálnej mriežky. Celková plocha kontejneru (objekt, obsahujúci "v sebe" nejaké ďalšie objekty) je rozdelená na obdĺžniky a v každom z nich sa nachádza jeden komponent. Je možné nastaviť počet riadkov a počet stĺpcov.

- **GridBagLayout**

  GridBagLayout (rozloženie do zobrazovacích oblastí), je komplexnejší správca rozloženia. Ako základ sa opäť pracuje s vytvorenou virtuálnou mriežkou, ale v tomto prípade je umožnené, aby komponent zaberal aj viac než len jednu bunku. Je umožnené aplikovať viaceré rôzne pravidlá pre správanie sa správcu, čo umožňuje vytvorenie zložitého rozloženia komponentov.

- **BoxLayout**

  BoxLayout (“škatuľové“ rozdelenie) je jednoduchý správca rozloženia umiestňujúci komponenty do riadkov alebo stĺpcov (závisiac na nastavení).

- **BorderLayout**

  BorderLayout (rozloženie do piatich oblastí), je rozdelenie kontejnera na päť oblastí, jednu centrálnu a štyri okrajové. Používa sa pre hlavné aplikačné okná.

Z uvedených správcov rozloženia dostupných v jazyku Java majú niektoré podobné chápanie rozmiestňovania komponentov v Jave, aké nájdeme aj pri rozmiestňovaní objektov na webovej stránke pomocou CSS. Správca pre plávajúce rozloženie FlowLayout, je podobný pozíciovaniu

pomocou CSS vlastnosti *float* a *clear* pri tvorbe stránok. Rozloženie do mriežky pomocou GridLayout, má princíp podobný rozloženiu objektov na webových stránkach s použitím tabuliek, už nepoužívaný zastaraný spôsob, alebo s použitím CSS pozíciovania, pričom sa ale budú vyskytovať problémy vyplývajúce z nedostatkov tejto metódy (viz. kapitola 3).

5.2 Mriežkové rozloženie komponentov

Z v Jave poskytovaných správcov rozloženia sa zameriame na správcov rozloženia pracujúcich na mriežkovom princípe. Sem patria GridLayout a GridBagLayout zaisťujúci efektívne rozloženie objektov do požadovanej štruktúry s využitím virtuálnej mriežky tvoriacej kostru rozloženia. Lepšie pochopenie princípu fungovania týchto správcov rozloženia sa využije ako základ pri návrhu efektívneho a jednoduchého rozloženia objektov na webovej stránke. Cieľom je na základe získaných poznatkov o princípe fungovania správcov rozloženia v jazyku Java vytvoriť návrh ako podľa podobných princípov rozmiestňovať objekty pri tvorbe webovej stránky (problematika podrobne rozoberaná v kapitole 6.).

5.2.1 Správca rozloženia GridLayout

Správca rozloženia GridLayout (rozloženie do mriežky), je založený na princípe usporiadania komponentov do virtuálnej mriežky. Pre každý komponent umiestnený do takejto mriežky je určený stĺpec a riadok, do ktorého patrí (teda súradnice konkrétnej bunky mriežky). Komponent vyplňuje celkové nevyužité miesto v danej bunke mriežky. Pre jednotlivé bunky mriežky platí pravidlo, že majú rovnakú výšku aj šírku a sú vždy čo možno najväčšie, závisiac na aktuálnej veľkosti kontejnera.

GridLayout podporuje možnosť určenia orámovania, miesta medzi jednotlivými bunkami mriežky s využitím triedy *GridLayout*:

- *setVgap* / *setHgap* - umožňujú určenie medzery medzi jednotlivými bunkami definovaním vertikálneho alebo horizontálneho voľného priestoru okolo buniek.

5.2.2 Správca rozloženia GridBagLayout

Správca rozloženia GridBagLayout (rozloženie do zobrazovacích oblastí) sa v Jave používa pre prípad, že chceme rozmiestniť viacero komponentov v danom kontejneri. Pracuje pomocou akejsi virtuálnej mriežky, umožňujúc nastavenie veľkosti jednotlivých riadkov a stĺpcov, do ktorej sa potom umiestňujú jednotlivé komponenty. Pre každý komponent sa môže nastaviť jeho veľkosť, natiahnutelnosť (o koľko nám komponent bude môcť narásť, natiahnuť sa v prípade zmeny veľkosti mriežky, napr. zmena zobrazovaného okna) a jeho okraje. Každý komponent vložený do mriežky sa

môže natiahnuť cez jej viaceré stĺpce alebo riadky. Pre komponent umiestnený pomocou tohto správcu rozloženia sa dajú určiť pravidlá jeho správania sa využitím triedy *GridBagConstraints*:

- *gridx/gridy* –ktoré nám určujú, v ktorej bunke mriežky bude uložený horný roh komponentu. Premenná *gridx* nám určuje číslo stĺpca a *gridy* číslo riadku.
- *gridwidth/gridheight* – špecifikujú, cez koľko stĺpcov/riadkov bude daný komponent natiahnutý. Je možné použiť veľmi užitočnú konštantu *GridBagConstraints.REMAINDER*, ktorá v prípade *gridwidth* udáva, že komponent je posledný v danom riadku a v prípade *gridheight* sa jedná o posledný komponent v danom stĺpci.
- *weightx/weighty* - špecifikujú, či je možné "natiahnuť" stĺpec/riadok, (v ktorom je daný komponent umiestnený) až do limitu voľného miesta na paneli.
- *anchor* - umožňuje zadať umiestnenie komponentu vo vzniknutom priestore pomocou konštánt *NORTH* (zarovnanie smerom hore), *NORTHEAST* (zarovnanie smerom k hornému pravému rohu bunky mriežky), *EAST* (zarovnanie smerom doprava), *SOUTHEAST* (zarovnanie smerom k dolnému pravému rohu), *SOUTH* (zarovnanie smerom spodku obmedzujúcej bunky mriežky), *SOUTHWEST* (zarovnanie smerom k dolnému ľavému rohu), *WEST* (zarovnanie smerom doľava), *NORTHWEST* (zarovnanie smerom k hornému ľavému rohu), a *CENTER* (zarovnanie na stred), určujúcich smer zarovnania komponentu.
- *ipadx/ipady* - umožňuje zadať počet pixlov, o ktoré sa daný komponent zväčší do strán *ipadx*, alebo do výšky *ipady*.

Uvedené premenné a ich vlastnosti nepopisujú celkový prehľad dostupných premenných a ich vlastností poskytovaných triedou *GridBagConstraints*. Zamerali sme sa len na premenné a vlastnosti, ktoré sú svojou funkcionalitou užitočné a využiteľné pri návrhu nových rozširujúcich vlastností CSS pozíciovania. Pre podrobnejší prehľad možností a fungovania Layout Manageru *GridBagLayout* ako aj pre detailný popis triedy *GridBagConstraints*, pozri [14].

5.2.3 Uplatnenie mriežkového rozloženia v prípade webových stránok

Chýbajúca možnosť vytvárania akejsi mriežky, tabuľkovej kostry pre rozloženie objektov na stránke je jedným z hlavných nedostatkov CSS pozíciovania, a príčinou, prečo sa ešte stále stretávame s využitím HTML tabuľky pre tvorbu kostry stránky k rozloženiu objektov.

Riešením tohto problému by bolo vytvorenie spôsobu, ktorý umožňuje definovanie pozície objektu v akejsi virtuálnej mriežke jednoduchým spôsobom, podobným možnostiam pravidiel GridBagLayout. Pri väčšine implementovaných internetových stránok sa problém rozloženia objektov týka práve rozloženia tabuľkového typu (do mriežky), objekty chceme mať usporiadané do riadkov a stĺpcov.

Máme tu potrebu vytvorenia metódy umožňujúcej delenie stránky na menšie časti reprezentujúce jednotlivé bunky mriežky, do ktorých chceme naše objekty umiestniť. Pred samotným umiestňovaním objektov sa vytvorí návrh, v akom usporiadaní by sa na stránke mali nachádzať, a na základe takto získaných znalosti by sa určovala poloha objektov vo virtuálnej mriežke tvoriacej kostru nášho dokumentu. Ako jednoduchý príklad si môžeme predstaviť vytváranie prihlasovacieho formuláru, ktorý by mal obsahovať hlavičku a niekoľko vstupných polí pre vyplnenie mena, priezviska, číslo mobilného telefónu a dátum narodenia. Po získaní týchto informácií si náš formulár môžeme predstaviť ako tabuľku s dvoma stĺpcami a šiestimi riadkami.
Z pohľadu virtuálnej mriežky dokumentu by naša stránka mohla vyzerať nasledovne:

Hlavička obsahujúca nadpis: *Formulár*	
Nápis: *Meno*	Pole pre hodnotu: *Meno*
Nápis: *Priezvisko*	Pole pre hodnotu: *Priezvisko*
Nápis: *Telefónne číslo*	Pole pre hodnotu: *Telefónne číslo*
Nápis: *Dátum narodenia*	Pole pre hodnotu: *Dátum narodenia*
	Tlačidlo *Uložiť*

Tabuľka 5.1

Virtuálna mriežka znázornená v tabuľke 5.1. nám ukazuje formu kostry potrebnej pre daný formulár. V prípade, že by stránka obsahovala takúto kostru rozloženia objektov, by sa objekty mohli umiestňovať na správne miesto len pomocou definovania súradníc, v ktorom riadku a v ktorom stĺpci mriežky sa nachádza bunka, do ktorej má byť daný objekt umiestnený.

5.2.4 Aplikovanie princípu fungovania GridBagLayout

Pre tvorbu riešenia mriežkového rozloženia objektov na webovej stránke budeme vychádzať z vlastností a možností poskytovaných mriežkovým správcom rozloženia v Jave GridBagLayout. Tento správca rozloženia nám slúži ako základný model pre navrhnutie pozíciovania objektov na webovej stránke. GridBagLayout v sebe zahŕňa mnohé užitočné vlastnosti, uľahčujúce definovanie rozloženia objektov v dokumente.

Využitie tvorby virtuálnej mriežky, (kde sa v prípade GridBagLayout pomocou vlastností *gridx* a *gridy* umožňuje určovanie presnej polohy objektu) by v prípade tvorby webových stránok znamenalo značné uľahčenie určovania polohy objektu, na rozdiel od bežne používaného spôsobu pozíciovania využitím CSS vlastností. V prípade CSS pozíciovania je jeden z veľkých problémov počítanie súradníc objektu a chýbajúca možnosť vytvorenia tabuľkovej kostry pre objekty, pričom by v oboch týchto prípadoch mohla našu prácu s umiestňovaním objektov uľahčiť vlastnosť určovania súradníc typu “do ktorého riadku, v ktorom stĺpci“ sa má objekt umiestniť.

Ako doplňujúca vlastnosť k súradnicovému umiestňovaniu objektov do našej virtuálnej mriežky by bolo potrebné tiež doplniť možnosť nastavovania parametrov podobných *gridwidth* a *gridheight*, ktoré by umožňovali nastaviť rozťahovanie sa bunky obsahujúcej objekt cez viaceré stĺpce i riadky mriežky dokumentu. Realizácia pozíciovania webových objektov na stránke popísaným spôsobom by umožňovala efektívne a rýchle rozmiestňovať objekty po stránke, (prípadne v obmedzujúcom kontejneri), bez potreby využitia predvytvorenej tabuľky ako kostry dokumentu alebo nutnosti zložitého počítania vzdialeností objektu od strán dokumentu.

Ďalšie možné rozšírenie by bolo obohatenie pozíciovania aj o parametre typu *weightx* a *weighty* v kombinácii s parametrami *ipadx* a *ipady*, teda vytvorenie vlastností umožňujúcich v prípade zmeny veľkosti strany prispôsobiť aj šírku stĺpcov (podrobnejší popis v ďalšej podkapitole).

Doplnenie CSS pozíciovania o vlastnosť *anchor* by umožňovalo definovať pozíciu objektu jedným príkazom, a to ako jeho ukotvenie k určitej strane alebo rohu dokumentu, či obsahujúcemu bloku.

6. Návrh riešenia rozloženia webovej stránky

V tejto kapitole sa budeme zaoberať možnosťami umiestňovania objektov na stránke, založenými na poznatkoch získaných zo štúdia princípu fungovania správcov rozloženia jazyka Java pracujúcich na mriežkovom princípe (pozri kapitolu 5). Podľa získaných poznatkov vytvoríme návrh, ako realizovať tento princíp pre webové stránky.

V kapitole budú prezentované viaceré spôsoby riešenia problematiky umiestňovania objektov na webovej stránke pomocou rozširujúcich vlastností pozíciovania zavedením nových nepárových HTML atribútov, či cez grafický editor pre mriežkové usporiadanie objektov.

6.1 Doplňujúce vlastnosti pre pozíciovanie

Jedným zo spôsobov pozíciovania objektu na stránke je vytvorenie nových HTML atribútov pre definovanie doplňujúcich vlastností polohy objektu, na základe ktorých bude možné určiť umiestnenie objektu vo virtuálnej mriežke dokumentu. Tieto atribúty by mali byť podobné tým zo správcu rozloženia GridBagLayout.

6.1.1 Popis rozširujúcich vlastností

Rozširujúce hodnoty udávajúce polohu objektu sa aplikujú ako atribúty pre daný objekt, ktorý pomocou nich chceme umiestniť, alebo sa použijú pre určenie presnej polohy bunky mriežky, do ktorej chceme daný objekt umiestniť. V nasledujúcej časti bude prezentovaný stručný prehľad týchto atribútov ako aj špecifikácia ich významu a povoľovaných hodnôt.

6.1.1.1 obj_x

Bude sa využívať pre určenie vodorovnej polohy, v ktorom stĺpci sa nachádza bunka obsahujúca daný objekt, určuje nám X-ovú súradnicu. V prípade, že sa jedná o objekt vložený do bunky roztiahnutej cez viaceré stĺpce, sa hodnota chápe ako hodnota stĺpca, v ktorom bunka začína. Atribút môže obsahovať pozitívne celočíselné hodnoty, začínajúc hodnotou 0. Hodnota *obj_x* je povinná. Napríklad objekt nachádzajúci sa v 4. stĺpci bude mať hodnotu:

<... obj_x="3" ....>

Číslo tri značí štvrtý stĺpec, čísluje sa od hodnoty nula.

6.1.1.2 obj_y

Túto hodnotu využívame pre určenie horizontálnej polohy, v ktorom riadku sa nachádza bunka obsahujúca daný objekt, určuje nám Y-ovú súradnicu, jedná sa o povinnú hodnotu. V prípade, že sa jedná o objekt vložený do bunky, roztiahnutej cez viaceré riadky, sa hodnota chápe ako hodnota riadku, v ktorom objekt začína. Vlastnosť môže obsahovať pozitívne celočíselné hodnoty, začínajúc hodnotou 0. Napríklad objekt nachádzajúci sa v 3. riadku bude mať hodnotu:

<... obj_y="2" ....>

Hodnota dva, je z dôvodu číslovania od nuly.

6.1.1.3 col_width

Col_width musí mať celočíselnú nenulovú pozitívnu hodnotu, ktorá nám určuje cez koľko stĺpcov je bunka, do ktorej je objekt umiestnený roztiahnutá. Jedná sa o poslednú z povinných atribútov.

6.1.1.4 col_height

Col_height je nepovinný atribút s celočíselnou nenulovou pozitívnou hodnotou, ktorá nám určuje cez koľko riadkov je bunka obsahujúca objekt roztiahnutá.

6.1.1.5 obj_weight

Atribút *obj_weight* je nepovinný. V prípade jeho využitia môžeme pomocou tohto atribútu určiť pomerné rozdelenie stĺpcov alebo správane sa objektov v bunke , môže obsahovať hodnotu *static* alebo pozitívnu nenulovú číselnú hodnotu (nemusí byť celočíselná). Atribút *obj_weight* môžeme priradiť len objektom s hodnotou *col_height* rovnou jedna.

- *static* – je jediná platná nečíselná hodnota, ktorú môžeme atribútu *obj_weight* priradiť. Hodnota sa bude vzťahovať na všetky objekty umiestnené na stránke pomocou našej metódy. Jej význam je, že sa pre každý objekt zachová jeho šírka nastavená vlastnosťou *width* v CSS a šírku stĺpca bude určovať najširší objekt v danom stĺpci.
- *Číselná hodnota*- nám určuje váhu stĺpca, v ktorom sa bunka nachádza. Nie je možné priradiť hodnotu bunke, ktorá je roztiahnutá cez viaceré stĺpce. Číselná hodnota, ktorú zadáme sa vzťahuje na stĺpec, v ktorom sa bunka nachádza, a je teda dostačujúce ju definovať pre každý stĺpec len raz. V prípade, že by bola hodnota uvedená pre ten istý stĺpec viackrát (pri každom ďalšom definovaní umiestnenia objektu do daného stĺpca), sa vždy berie do úvahy posledná zadaná hodnota pri vyhodnocovaní stĺpca smerom zhora dole (po riadkoch). Číselné hodnoty určujú pomerné rozdelenie dostupnej zobrazovacej oblasti medzi stĺpcami. V prípade nezadania žiadnej hodnoty pre žiaden zo stĺpcov sa tieto rozdelia rovnomerne. V prípade, že

zadáme hodnotu len pre niektoré stĺpce, sa ostatným priradí hodnota nula, a teda nebudú zobrazené.

6.1.1.6 anchor

Jedná sa o nepovinný atribút umožňujúci umiestnenie objektu ukotvením k jednej zo strán alebo k rohu stránky alebo obsahujúceho bloku. Môže nabrať hodnoty *bottom_left*, *bottom_right*, *bottom*, *right*. Tieto hodnoty svojím správaním presne zodpovedajú vlastnostiam uvedeným v podkapitole 3.5. Hodnota *top_left* podporovaná nie je, keďže sa objekty začínajú rozmiestňovať vždy od ľavého horného rohu, ktorý zodpovedá polohe bunky so súradnicami:

<... obj_x="0" obj_y="0" ....>

Riešenie by malo fungovať nasledovne. Majme "objekt1", ktorý chceme umiestniť na našej stránke vedľa objektu nachádzajúceho sa v siedmom rade a v štvrtom stĺpci našej stránky. Objekt1 by mal zaberať ako na šírku tak aj na výšku vždy dve bunky virtuálnej mriežky, a to pri zachovaní veľkosti buniek mriežky. Toto môžeme dosiahnuť nasledujúcim spôsobom:

Vlastnosti pre určenie polohy objektu ako aj ďalšie doplňujúce vlastnosti týkajúce sa jeho polohy na stránke bude potrebné definovať v HTML kóde ako atribúty bunky, do ktorej je objekt umiestnený, a teda v našom prípade by pribudli nasledujúce vlastnosti k deklarácii objektu v HTML kóde:

< div	*id=" objekt1"*	začínajúci div tag a určenie Id objektu
	obj_x="3".....	poloha v stĺpci 4
	obj_y="6"	poloha v riadku 7
	col_width="2".....	šírka objektu zaberie 2 bunky virtuálnej mriežky (stĺpec 7 a 8)
	col_height="2".....	výška objektu zaberie 2 bunky virtuálnej mriežky (riadok 4 a 5)
	obj_weight="static" >.....	šírka objektu nastavená pomocou CSS zostane vždy zachovaná
	....." objekt"....	
* *		ukončujúci div tag

Takto zapísaná deklarácia v HTML by mohla plniť funkciu určovania polohy kam bude objekt umiestnený.

Uvedené nové atribúty a ich hodnoty by sa vyhodnocovali pomocou JavaScriptu, ktorý by bol pripojený k stránke a po spracovaní zadaných hodnôt atribútov pre určovanie polohy objektu by tieto vyhodnocoval a vytvoril by doplňujúce CSS vlastnosti určujúce polohu.

6.1.2 Doplňujúce parametre pre umiestňovanie

Pre úspešnú realizáciu umiestňovania objektov pomocou parametrov definovaných ako HTML atribúty a ich hodnoty, je potrebné zadefinovať ďalšie parametre.

6.1.2.1 Trieda layoutobj

Jedná sa o vlastnosť zavedenú z dôvodu potreby určenia, ktoré objekty sa majú umiestniť na stránkach pomocou metódy pozíciovania. Z tohto dôvodu je potrebné, aby všetky bunky, ktoré majú tvoriť kostru tabuľky pre rozmiestňovanie objektov patrili do tej istej triedy, a teda aby obsahovali vlastnosť *class* s hodnotou *layoutobj*.

<... class="layoutobj" ....>

Táto vlastnosť nám nastavuje, že sa jedná o bunku tabuľkovej kostry, do ktorej sa má umiestniť daný objekt.

6.1.2.2 Id objektu

Atribút *id* umožňujúci jednoznačnú identifikáciu daného objektu je z hľadiska využitia layout manageru *layoutm.js* nepotrebný ,a umožňuje nám uplatniť doplňujúce CSS vlastnosti pre konkrétnu bunku tabuľky, ako sú napríklad farba pozadia alebo *padding* alebo šírka či výška objektu.(vlastnosť *margin* nie je podporovaná, pozri: 7.2.2.1)

6.1.2.3 Target

Ďalším zavedeným atribútom je *target*, jedná sa o nepovinný atribút, ktorého hodnota udáva cieľový objekt, do ktorého sa má umiestniť naša tabuľková kostra. Ako jeho hodnota sa udáva *id* objektu, do ktorého chceme naše objekty rozložiť.

Target sa definuje raz pre ľubovoľný z objektov, ktoré chceme umiestniť. Keby sme vlastnosť target definovali viackrát, zohľadnil by sa posledný uvedený variant. V prípade, že target nie je definovaný sa objekty rozložia do okna prehliadača. V prípade, že chceme naše objekty rozložiť do objektu s *id=“test“*, to deklarujeme nasledovne:

<... target="test" ....>

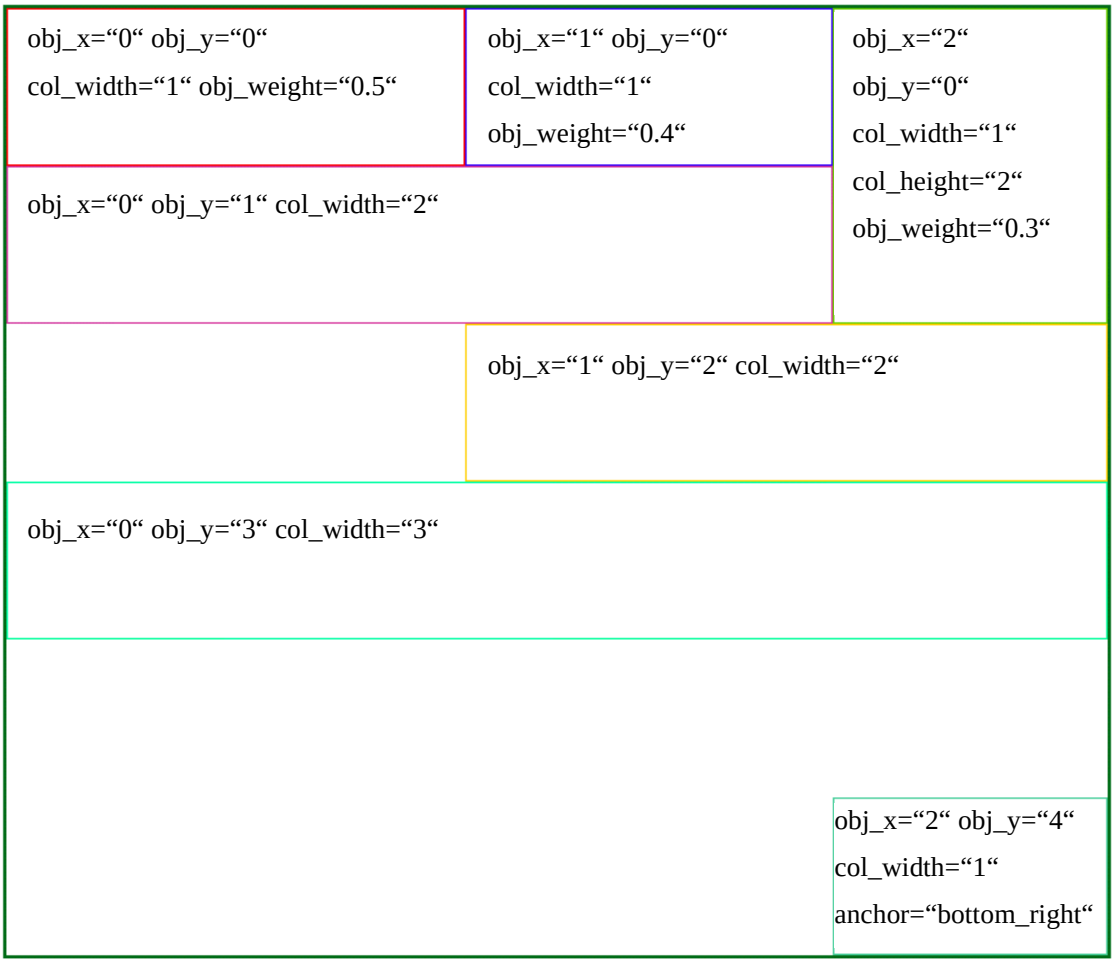
6.1.3 Výška riadkov vo virtuálnej mriežke

Výška jednotlivých riadkov vo vytvorenej virtuálnej mriežke, do ktorej sú objekty umiestňované, sa vždy rovná najvyššiemu objektu v riadku. Výnimkou je prípad, keď naša tabuľková kostra obsahuje bunku, ktorá je roztiahnutá cez viaceré riadky tabuľky, pričom je jej celková výška väčšia, než súčet výšok najvyšších objektov pre jednotlivé riadky, cez ktoré sa objekt rozťahuje. V tomto prípade sa k výške posledného riadku, v ktorom sa roztiahnutý objekt ešte nachádza pripočíta výška, o ktorú roztiahnutý objekt prevyšuje súčet výšok jednotlivých riadkov (cez ktoré sa rozťahuje).

6.1.4 Význam mriežkového rozloženia

Pre rozmiestnenie objektov na stránke sa používa mriežkové rozloženie, ktoré sa vytvorí podľa zadaných hodnôt, a do ktorého buniek sa umiestňujú objekty.

Nasledujúci obrázok 6.1. znázorňuje vytvorenú virtuálnu mriežku (akoby kostru pre rozloženie objektov) s bunkami, do ktorých je možné umiestňovať objekty. V každej z buniek sú uvedené rozširujúce vlastnosti, pomocou ktorých je vytvorená.



Obr. 6.1

Virtuálna mriežka vytvára sama seba, a to na základe zistených súradníc kam sa majú objekty umiestňovať (hodnoty parametrov *obj_x / obj_y*) a hodnôt určujúcich veľkosti jednotlivých buniek (na koľko riadkov/stĺpcov je bunka roztiahnutá). Podľa týchto hodnôt sa zistí veľkosť mriežkovej kostry, kde najväčšie hodnoty určujú súradnice umiestnenia objektu a určujú aj počet riadkov a stĺpcov tabuľky. Pomocou parametrov *col_width* a *col_height* sa definuje, ktoré z buniek sa majú spojiť do jednej väčšej bunky, ktorá je roztiahnutá cez viaceré stĺpce alebo riadky.

Vytvorená virtuálna mriežka sa skladá z troch stĺpcov a z piatich riadkov. Jednotlivé stĺpce majú svoju šírku určenú podľa jednotlivých pridelených hodnôt atribútu *obj_weight*. Výška riadku vychádza z výšky najvyššieho objektu v riadku. Tiež je znázornená funkčná podstata atribútu *anchor*, ktorá má za dôsledok ukotvenie bunky do ľavého dolného rohu zobrazovacej plochy.

6.1.5 Editor doplňujúcich atribútov pozíciovania

Vhodným ošetrením pre pozíciovanie pomocou využitia doplňujúcich vlastností určovaných HTML atribútmi, by bolo vytvorenie webovej aplikácie editora, ktorý by pre zadané doplňujúce atribúty pozíciovania vrátil potrebný CSS kód pre realizáciu tohto rozloženia. Vytvorenie takejto aplikácie by zaručilo, že objekty budú na stránke správne rozložené a to aj bez potreby pripojenia potrebného JavaScriptu k stránke.

Samotné rozloženie objektov na stránke by teda nebolo závislé na tom, či je v prehliadači povolená podpora pre JavaScript.

Samotná webová aplikácia by mala mať formu, ktorá by umožňovala na jednej stránke vidieť súčasne ako vstupný kód obsahujúci popis rozloženia objektov s využitím rozširovacích HTML atribútov, tak aj výsledok zobrazenia spolu s potrebným CSS kódom pre dosiahnutie daného výsledku. Takto zrealizovaná aplikácia by umožňovala jednoduchým spôsobom upraviť kód pre určenie polohy objektov a zaručila by získanie CSS kódu pre požadovaný spôsob rozloženia objektov.

Stránku si môžeme predstaviť ako dva stĺpce, pričom v pravom sa nachádza zobrazovaný výsledok a v ľavom stĺpci je vstupné textové pole pre vloženie kódu, ako aj textové pole pre zobrazenie výsledného CSS kódu.

6.2 Grafický editor mriežkového usporiadania

Druhým zo spôsobov pozíciovania objektu na stránke je vytvorenie grafického editora pre umiesťovanie objektov do virtuálnej mriežky. Editor by mal umožniť intuitívne a jednoduché vytvorenie virtuálnej mriežky podľa potreby a umožniť umiestňovanie objektov do jednotlivých buniek tejto mriežky.

Editor by mal umožňovať intuitívnu a ľahkú prácu s virtuálnou mriežkou, presné a jednoduché rozmiestnenie objektov v jednotlivých bunkách mriežky a export kódu potrebného pre zachovanie

navrhnutého rozloženia objektov. Pre splnenie požiadaviek intuitívneho a ľahkého ovládania by sa malo čo najviac obmedziť používanie klávesnice a prispôsobiť editor pre jeho ovládanie pomocou jednoduchých operácií využívajúcich myšku. Tri hlavné skupiny činnosti, ktoré by mal takýto editor spĺňať, sú:

- Intuitívna tvorba virtuálnej mriežky (vytvorenie a úprava veľkosti aj rozloženia buniek)
- Rýchle a jednoduché rozloženie jednotlivých objektov do buniek mriežky
- Export vzniknutého kódu pre rozloženie objektov.

6.2.1 Virtuálna mriežka

Vychádzajúc z poznatkov nadobudnutých štúdiom správcov rozloženie mriežkového typu v jazyku Java, sa ako hlavný cieľ vytvorenia grafického editora javí navrhnutie jednoducho a intuitívne ovládateľného spôsobu pre tvorbu virtuálnej mriežky. Editor by mal umožňovať efektívne a rýchle vytvorenie tejto mriežky, ako aj spôsoby pre jej jednoduchú úpravu.

Samotnú virtuálnu mriežku si môžeme predstaviť ako maticu alebo tabuľku s vopred daným počtom stĺpcov a riadkov. Jednotlivé bunky vytvorenej mriežky musia byť upravovateľné, čo sa týka nastavovania ich veľkosti alebo spojenia s ďalšou susednou bunkou v prípade, že chceme vytvoriť spojitú bunku, siahajúcu cez viaceré riadky či stĺpce (alebo oboje) danej mriežky. Tiež musí byť poskytnutá možnosť ako do danej bunky vytvorenej virtuálnej mriežky umiestniť jednotlivé objekty.

6.2.1.1 Tvorba virtuálnej mriežky

Prvým krokom tvorby virtuálnej mriežky bude určenie počtu riadkov a stĺpcov tvoriacich túto maticu. Tieto údaje bude treba definovať pomocou číslic reprezentujúcich daný počet stĺpcov a riadkov. Tieto vstupné údaje by mali byť jediné, pri definovaní ktorých sa počas tvorby virtuálnej mriežky využije klávesnica. Po vytvorení mriežky s daným počtom stĺpcov a riadkov sa nám poskytne niekoľko možností ako s touto maticou narábať z hľadiska zmeny jej veľkosti, či spájania jednotlivých buniek mriežky do väčšej jednotnej bunky.

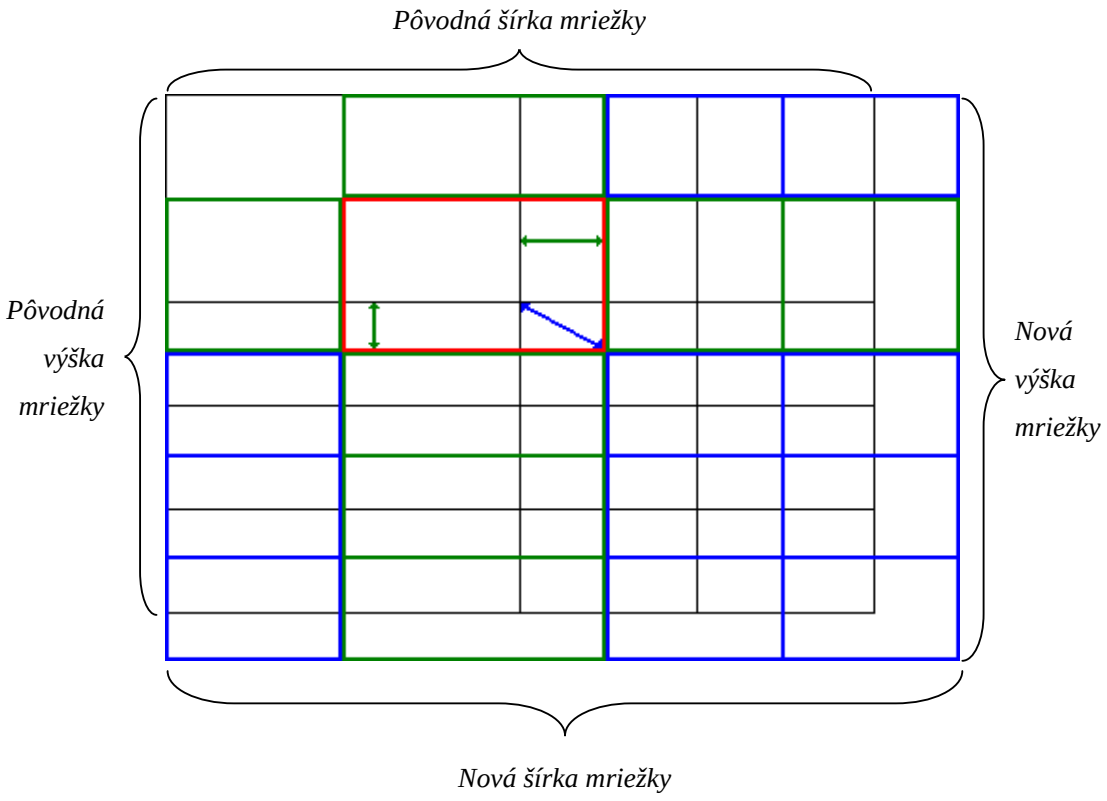
6.2.2 Zmena veľkosti vytvorenej virtuálnej mriežky

Zmena veľkosti buniek mriežky musí byť navrhnutá umožňujúc presnú a ľahkú zmenu potrebných veľkostí buniek Spomenieme dva hlavné spôsoby pre zmenu vlastností veľkosti mriežky.

V prvom prípade sa mení veľkosť konkrétne vybratej bunky, toto by sa malo realizovať pomocou možnosti uchytenia pravého dolného rohu bunky myškou a jej natiahnutie či zmenšenie ako vo vertikálnom tak aj v horizontálnom smere. Počas doby upravovania veľkosti bunky sa bude zobrazovať vždy aktuálna veľkosť bunky udávaná v pixloch. Zvyšok mriežky sa prispôsobí novej veľkosti bunky, ktorú sme upravovali. Zvyšné bunky nachádzajúce sa v riadku a v stĺpci upravovanej bunky sa musia upraviť a prispôsobiť novej veľkosti riadku či stĺpca vychádzajúc z nových veľkostí

upravovanej bunky. To znamená, že napr. v prípade, že sa menila výška danej bunky, sa všetky ostatné bunky nachádzajúce sa v riadku kam patrí aj upravovaná bunka, musia upraviť a ich nová výška musí zodpovedať novonadobudnutej výške upravovanej bunky (v prípade šírky sa uplatňuje podobný princíp pre daný stĺpec). Ďalším opatrením pri zmene veľkosti niektorej z buniek mriežky je vertikálny posun polohy buniek nachádzajúcich sa v priestore vymedzenom smerom zhora riadkom, do ktorého upravovaná bunka patrí, ako aj horizontálny posun buniek nachádzajúcich sa v priestore napravo od stĺpca, do ktorého upravovaná bunka patrí. Všetky tieto obmedzenia sú potrebné pre zachovanie mriežkového tvaru v prípade zmeny veľkosti bunky vo vytvorenej virtuálnej mriežke.

Pre lepšie pochopenie si pozrime obrázok 6.2., kde máme čiernou farbou na pozadí znázornenú pôvodne vytvorenú maticu o veľkosti 4x5. Bunka značená červenou farbou nám reprezentuje vybranú bunku, pre ktorú meníme jej veľkosť ako v horizontálnom, tak aj vertikálnom smere. Na obrázku vidíme, že bunky patriace do stĺpca či riadku upravovanej bunky zmenili svoju veľkosť prispôsobujúc sa veľkosti upravovanej bunky. Pre lepšie zviditeľnenie sú bunky nových rozmerov značené zelenou farbou ako aj šípky v upravovanej bunke znázorňujúce o koľko bunka vo vertikálnom či horizontálnom smere narastá. Pri bunkách zapadajúcich do oblasti napravo od stĺpca či pod riadkom upravovanej bunky, došlo k posunu ich polohy. Tento posun je znázornený vykreslením posunutých buniek modrou farbou.



Obr. 6.2

Druhým spôsobom zmeny veľkosti mriežky je možnosť úpravy veľkosti vybraného stĺpca či riadku. Pre umožnenie tejto operácie budú na konci riadkov i stĺpcov umiestnené tlačidlá umožňujúce

meniť veľkosť stĺpca či riadku vedľa ktorého sa nachádzajú. V prípade zmeny veľkosti daného riadku sa ostatné bunky, nachádzajúce sa pod daným riadkom posunú na novú polohu, aby sa dodržala mriežková štruktúra (v prípade zmeny veľkosti daného stĺpca sa pre stĺpce napravo od neho uplatňuje obdobný postup).

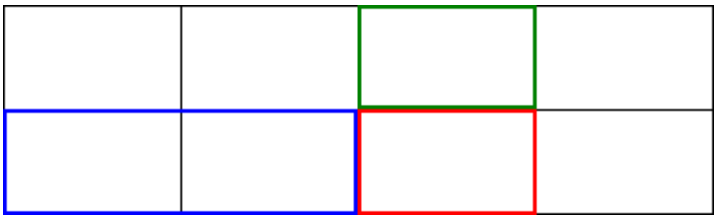
6.2.3 Spojovanie mriežok

Popri zmene veľkosti mriežky, je druhá z potrebných operácií pri tvorbe virtuálnej mriežky pre umiestňovanie objektov na stránke, spôsob umožňujúci spájanie viacerých mriežok do jednej väčšej mriežky roztiahnutej cez viaceré stĺpce či riadky. Táto operácia je potrebná, keďže väčšinou sa snažíme objekty rozložiť komplexnejšie, napr. vytvorenie spojitej veľkej bunky cez celý riadok, tvoriacej akúsi hlavičku pre nasledujúce objekty, a pre vytvorenie virtuálnej mriežky umožňujúcej takéto rozloženie je možnosť spájania buniek nevyhnutnosťou. Pri spájaní buniek môžeme rozlišovať prípady jednoduchého a komplexného spájania. Pri spájaní vždy pripájame jednoduché (ešte nespojené) bunky k otcovskej bunke. Otcovská bunka je bunka nachádzajúca sa napravo alebo nad bunkou, s ktorou ju chceme spojiť.

Spájanie bude prebiehať využitím tlačidiel pre určenie pripojenia buď buniek napravo alebo pod otcovskou bunkou. Pri spájaní buniek sa musia dodržať nasledujúce základné pravidlá:

- Nesmie sa spojiť nespojená otcovská bunka s už spojenou bunkou.
- Nesmú sa spojiť do jednej bunky dve spojené bunky (aj otcovská aj spájaná bunka sa skladá z viacerých buniek).

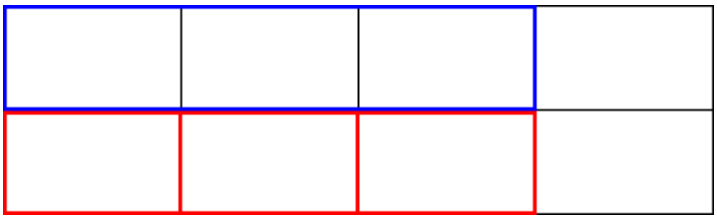
Pri jednoduchom spájaní buniek, keď k otcovskej bunke pripájame vždy naraz len jednu bunku, sa jedná o operáciu spájania buniek patriacich do rovnakého riadku či stĺpca. Dôležité je počas spájania kontrolovať a upravovať možnosti spájania okolitých buniek tak, aby boli vždy dodržané základné pravidlá spájania.



Obr. 6.3

Pre lepšie pochopenie si pozrime obrázok 6.3., kde máme čiernou farbou na pozadí znázornenú vytvorenú maticu o veľkosti 4x2. Modrou značená veľká bunka znázorňuje spojené dve bunky, ku ktorým v znázornenom kroku chceme pripojiť ďalšiu bunku značenú červenou farbou (modrá veľká bunka je teda otcovskou a k nej pripájame červenú jednoduchú bunku).Bunky nachádzajúce sa nad modrou bunkou už nemajú možnosť spájania smerom dole a po pripojení červenej bunky stráca túto schopnosť aj bunka nachádzajúca sa nad ňou značená zelenou farbou, aby sa dodržali základné pravidlá spájania buniek.

V prípade komplexného spájania vytvárame bunku, ktorej rozmery siahajú súčasne cez viaceré riadky i stĺpce. Rovnako ako aj v predchádzajúcom prípade aj tu platia základné pravidlá spájania buniek. Ku komplexnému spájaniu dochádza v prípade, ak chceme spojiť jedným smerom už spojenú bunku (bunka vznikla vertikálnym alebo horizontálnym jednoduchým spojením buniek), druhým smerom (v prípade, že bunky boli spojené jednoduchým spojením v horizontálnom smere, chceme uskutočniť vertikálne spojenie, alebo naopak). V tomto prípade dochádza k pripojeniu všetkých jednoduchých buniek nachádzajúcich sa vedľa alebo pod otcovskou bunkou ,ku ktorej ich chceme pripojiť.



Obr. 6.4

Na obrázku 6.4 vidíme príklad vertikálneho komplexného spojenia, keď modrá farba značí otcovskú bunku a červená farba jednoduché bunky, ktoré sa v nadchádzajúcom kroku spoja s otcovskou bunkou.

6.2.3.1 Zmena veľkosti pri spojených bunkách

V prípade spojených buniek nie je možné zmeniť ich veľkosť pomocou uchytenia za pravý dolný roh bunky (8.2.2). Toto ošetrenie je nutné z dôvodu, že by sa v prípade spojitej bunky nedalo určiť, ako sa má vykonaná zmena veľkosti bunky premietnuť na ostané bunky okolo. V prípade zmeny veľkosti spojenej bunky, ktorá pozostáva s dvoch horizontálne spojených jednoduchých buniek by sa nedalo určiť, ako sa majú zmeny vo veľkosti šírky rozložiť medzi dva stĺpce, medzi ktoré patria aj bunky, ktoré tvoria spojenú bunku.

V prípade, že chceme zmeniť veľkosť spojenej bunky, to môžeme dosiahnuť pomocou použitia úprav veľkosti vybraného stĺpca či riadku (viz. 6.2.2).

6.2.4 Umiestňovanie objektov do mriežky

Po vytvorení mriežky je potrebné zaistiť spôsob, ako jednoducho umiestniť objekty v mriežke, ako s umiestnenými objektami narábať. Efektívny spôsob ako vkladať objekty do mriežky, je umiestnenie tlačidla pre vloženie nového objektu do každej z buniek mriežky, do ktorej môžeme objekty umiestňovať.

Samotné vkladanie objektov sa bude realizovať pomocou vstupného textového rozhrania, kam sa vloží zdrojový kód obsahujúci potrebný popis daného objektu. Takto popísaný objekt sa potom umiestni do danej bunky, pri ktorej bola voľba vloženia objektu zvolená.

Pre objekty umiestňované do buniek mriežky platia dve základné pravidlá:

- Jedna bunka mriežky môže obsahovať ľubovoľný počet objektov, pričom sú tieto usporiadané za sebou.
- Žiaden z vložených objektov nesmie presahovať cez hranice bunky, do ktorej bol objekt vložený.

Po vložení objektu do bunky treba riešiť prípadné zmeny týkajúce sa polohy a obsahu objektu, prípadne spôsobu jeho odstránenia. Pre možnosť premiestňovania vybraného objektu do inej bunky mriežky v prípade potreby, alebo zmenu jeho zaradenia v rámci danej bunky (prípad, keď jedna bunka obsahuje viaceré objekty) bude na bunke zobrazené tlačidlo umožňujúce uchopenie objektu a jeho preloženie do inej bunky alebo na koniec. Poradie objektov v rámci bunky sa určí pomocou myšky.

Objekt bude tiež obsahovať ďalšie dve tlačidlá umožňujúce voľby jeho editácie či odstránenie z mriežky. V prípade editácie objektu sa poskytne užívateľovi možnosť zmeny zdrojového kódu daného objektu.

Ďalšou problematikou, ktorú treba vyriešiť, bude umožnenie spájania buniek či zmena ich veľkosti, aj v prípadoch, že tieto už obsahujú vložené objekty. Zmena veľkosti bunky po vložení objektov bude ovplyvňovať rozloženie ako aj zobrazovanú časť vložených objektov. V prípade zmenšenia sa bunky pod veľkosť vloženého objektu sa jeho prečnievajúce časti, ktoré by boli mimo rozsahu bunky, skryjú. Pri spojení dvoch buniek, ktoré už obsahujú nejaké vložené objekty, sa objekty spojenej bunky doplnia za objekty otcovskej bunky, ku ktorej sa ich bunka pripája.

6.2.5 Export vzniknutého kódu rozložených objektov

Po dokončení tvorby mriežky a vložení potrebných objektov na im predurčené miesta bude potrebné takto vytvorené rozloženie zobraziť a upraviť do tvaru, aby sa dalo ľahko preniesť a vložiť na potrebné miesto vo vytváranej aplikácii. Riešením problému je vytvorenie exportu potrebného kódu pre rozloženie objektov spolu s vloženým kódom umiestnených objektov tak, aby sa tento mohol vložiť na potrebné miesto. Takto exportovaný kód by mal obsahovať okrem kódu vloženého pri definovaní vlastností objektov užívateľom, len kód zodpovedajúci normám jazykov HTML a CSS tak, aby sa dal aplikovať s istotou do čo najširšieho množstva možných použití pre dané internetové aplikácie.

7. Implementácia

Implementácia je založená na získaných poznatkoch počas analýzy problematiky rozmiestňovania objektov na stránke metódou využívajúcou rozširujúce vlastnosti pozíciovania, ako aj využitím možností grafického editora (kapitola 6.).

Implementované riešenia sú založené na poznatkoch nadobudnutých počas štúdia možností umiestňovania objektov v jazyku Java a na základe vytvorených návrhov.

Prvým je rozšírenie vlastností pozíciovania objektov na stránke využitím programu písaného v jazyku JavaScript spracovávajúc zavedené HTML atribúty, pomocou objektového modelu dokumentu. Tento spôsob riešenia bude realizovaný ako doplňujúci JavaScript, ktorý bude pripojený k danej stránke a bude realizovať rozloženie objektov, vždy pri načítaní stránky. Druhým spôsobom riešenia rozloženia objektov na stránke s využitím HTML atribútov je spôsob vytvorenia webovej aplikácie slúžiacej ako editor pre rozloženie objektov na stránke vytvárajúc výsledný CSS kód pre rozloženie objektov. Tento spôsob by zabezpečoval správne rozloženie objektov na stránke nezávisiac na podpore JavaScriptu.

Druhou implementovanou formou riešenia je vytvorenie grafického editora pre rozmiestnenie objektov na stránke. Editor bude fungovať podľa popisu vychádzajúc z popisu v predchádzajúcej kapitole a jeho implementácia prebehne tiež v jazyku JavaScript s využitím objektového modelu dokumentu.

7.1 Použité programovacie technológie

Pri tvorbe alternatívneho riešenia som využil programovacie techniky JavaScript s využitím DOM-u. Koncept objektového modelu dokumentu (DOM) sa využíva k pristupovaniu k jednotlivým objektom a ich parametrom pomocou stromovej štruktúry dokumentu, jeho podrobný popis je uvedený v kapitole 4. Objektový model dokumentu DOM.

7.1.1 Stručný úvod do JavaScriptu

JavaScript je interpretovaný programovací jazyk so základným objektovo orientovaným konceptom. Klientska verzia tohto jazyka (tj. verzia pracujúca s DOM, pojmy ako udalosť prehliadača, dokument, okno apod.) je súčasťou väčšiny všeobecne rozšírených prehliadačov ako sú napr. Internet Explorer a Mozilla Firefox.

Jadro jazyka (tj. časť univerzálna a nezávislá od internetového prehliadača) je syntakticky veľmi podobné C, C++ alebo Jave. Podobnosť však končí na úrovni syntaxe. JavaScript je jazykom,

ktorý má potlačenú typovú kontrolu. Ďalej ide o interpretovaný jazyk, ktorý mnohé myšlienky preberá z jazyka Perl.

JavaScript ďalej nie je zjednodušenou verziou Javy. Ide o samostatný jazyk, ktorý vznikal pod názvom LiveScript. Názov bol zmenený tesne pred jeho uvedením na trh z marketingových dôvodov. Rovnako aj predstava, že ide o nejaký jednoduchý jazyk je chybná. Je rovnako zložitý ako ostatné univerzálne programovacie jazyky. V produktoch Microsoft je možné sa stretnúť s komerčným názvom pre tento jazyk JScript. [4]

7.2 Implementovanie rozloženia prvkov pomocou HTML atribútov

Vytvorená aplikácia umožňuje rozloženie objektov pomocou špecifikácií uvedených v kapitole 8.1. Jedná sa o JavaScript s aplikáciu *layoutm.js*, ktorá sa pripojí k HTML dokumentu, a pri nabehnutí stránky sa spustí jeho funkcia *layout_maker()*. Do HTML kódu sa pridajú nasledujúce deklarácie:

```
<script type="text/javascript" src="./layoutm.js">
....
<BODY onLoad="layout_maker()">
```

Správca rozloženia *layoutm.js* vyhodnotí parametre a ich hodnoty určujúce rozloženie objektov do buniek mriežkovej kostry a doplní k HTML kódu ďalšie CSS vlastnosti, ktoré zabezpečia správne rozloženie objektov na stránke. Aplikácia bola testovaná pre aktuálne verzie prehliadačov Internet Explorer a Mozilla Firefox Opera a Netscape Browser.

7.2.1 Princíp fungovania správcu rozloženia *layoutm.js*

Po nabehnutí stránky sa z nej zavolá javascript *layoutm.js*, ktorý má za úlohu vytvorenie tabuľkovej kostry a umiestnenie objektov do nej, na vopred určené správne miesto. Ako prvá sa priamo zo stránky zavolá funkcia *layout_maker()*, ktorá má za úlohu zo všetkých objektov, definovaných párnymi tagmi * objekt * vybrať tie, ktoré patria do triedy *layoutobj*:

```
<... class="layoutobj"  ....>
```

a majú sa rozložiť po stránke pomocou skriptu *layoutm.js*. V priebehu vyberania objektov, ktoré sa budú rozmiestňovať sa volá funkcia *calc_layout()*, ktorá nám zistí veľkosť tabuľky (počet stĺpcov a riadkov) a vytvorí základnú verziu tabuľkovej kostry pre rozloženie objektov.

Jadrom aplikácie je funkcia *find_first_obj()*, volaná po vyselektovaní všetkých objektov, ktoré sa majú umiestniť pomocou našej aplikácie. Táto funkcia prejde všetky objekty, ktoré chceme umiestniť na stránke v dvoch prechodoch, pričom objekty vždy vyberá v poradí ako sa budú zobrazovať na stránke (zaradom, začínajúc od horného ľavého rohu). V prvom prechode sa pomocou funkcie *collect_weight()* a ňou volaných pomocných funkcií zistí pomerové rozdelenie stĺpcov v tabuľkovej kostre.

V druhom priechode sa potom pre každý objekt zavolá funkcia *place_object()*, ktorá nám vypočíta polohu bunky, v ktorej sa má objekt nachádzať a vygeneruje potrebný CSS kód pre umiestnenie objektu.

Funkcia *place_object()* je najrozsiahlejšou funkciou aplikácie, volá mnohé pomocné funkcie pre výpočet rozloženia tabuľky, počíta veľkosti riadkov ako aj doplňujúce vlastnosti typu *anchor*. Tiež má na starosti overovanie vlastnosti *target*, ktorá určuje, kam sa má výsledné rozloženie zobraziť.

Samotné zobrazenie má na starosti funkcia *target_area()*, ktorá vloží výsledné rozloženie do určeného cieľového objektu alebo priamo do tela stránky.

7.2.2 Špecifikácia používania

Pre správnu prácu správcu rozloženia *layoutm.js* je potrebné tento JavaScript nalinkovať na HTML stránku a zavolať funkciu *layout_maker()*. Dôležité je mať správne určené vstupné parametre a prehliadač musí podporovať JavaScript.

Je potrebné správne nastaviť súradnicové bunky tabuľky (*obj_x* a *obj_y*) kam chceme objekt umiestniť, ako aj veľkosť objektu, teda cez koľko riadkov a stĺpcov tabuľky sa bude bunka obsahujúca daný objekt rozťahovať.

Aplikácia vytvára CSS kód určujúci rozloženie a veľkosť jednotlivých buniek, do ktorých umiestňuje dané objekty. Každý z objektov tvoriaci bunku tabuľky je objekt, ktorý môže obsahovať ľubovoľné *id*, a teda sa mu v CSS môžu prideliť ďalšie vlastnosti ako šírka či výška objektu, prípadne farba pozadia či *padding*.

Poloha objektu, a teda bunka mriežky, ktorá objekt obsahuje, je určená v CSS pomocou súradníc HTML rozšírenia pre jednoduché určenie polohy. V CSS vlastnostiach je definovaný *padding* pre vzdialenosť od ostatných buniek, ako aj výška bunky (z dôvodu efektívnejšieho zobrazenia príkladu.), šírka vyplýva z počtu umiestňovaných objektov, a v prípade ich uvedenia aj z nastavenia hodnôt parametru *obj_weight* pre stĺpce tabuľkovej kostry.

Príklad definície objektu umiestneného do bunky pomocou *layoutm.js*:

- CSS:

  # tableCell { background: blue; height: 500px; padding: 12px; }

  # innerObj {background: green; height: 100%; width: 100%; }

- HTML:

```
<div id="tableCell"
    class="layoutobj"
    obj_x="1"
    obj_y="1"
    col_width="1" >
        <div id="innerObj">
            dsf
        </div>
</div>
```

definovanie veľkosti a polohy bunky v tabuľkovej kostre

objekt umiestnený do bunky tabuľkovej kostry (*dsf* – je obsah objektu *innerObj*)

Vytvorená mriežková kostra sa umiestni buď do predurčeného obsahujúceho bloku alebo priamo do stránky. V prípade, že sa jedná o umiestnenie do obsahujúceho bloku, si ten pre nášho správcu rozloženia určíme pomocou jeho *id*, ktorého hodnotu uvedieme ako hodnotu parametra *target*. Pri tvorbe tabuľkovej kostry sa pri výpočtoch pre šírku stĺpcov vychádza vždy zo šírky objektu (stránky alebo obsahujúceho bloku) kde sa bude tabuľka vytvárať, táto šírka sa potom delí medzi jednotlivé stĺpce (výnimkou je prípad použitia hodnoty *static*). V prípade použitia parametra *obj_weight* pre určenie pomeru medzi šírkami stĺpcov, sa pomocou hodnôt tohto parametra vypočíta percentuálna šírka jednotlivých stĺpcov. V prípade, že sa takto vytvorená tabuľka neumiestni do obsahujúceho bloku s pevnou šírkou, sa šírky stĺpcov budú prispôsobovať aktuálnej šírke objektu (stránky alebo obsahujúceho bloku), kde sa zobrazujú.

obj_x=”0” obj_y=”0” col_width=”1” obj_weight=”0.5”	obj_x=”1” obj_y=”0” col_width=”1” obj_weight=”0.4”	obj_x=”2” obj_y=”0” col_width=”1” obj_weight=”0.3”

obj_x=”0” obj_y=”0” col_width=”1” obj_weight=”0.5”	obj_x=”1” obj_y=”0” col_width=”1” obj_weight=”0.4”	obj_x=”2” obj_y=”0” col_width=”1” obj_weight=”0.3”

Obr. 7.1.

Príklad na obrázkoch 7.1. ukazuje zmenu šírky stĺpcov v závislosti na zmene veľkosti obsahujúceho bloku, pričom sa ale dodržiavajú pomerové vlastnosti medzi šírkami stĺpcov, ktoré sú dané parametrom *obj_weight* (bunky obsahujú jeho hodnotu pre aktuálny stĺpec).

K aplikácii bol vytvorený aj stručný manuál obsahujúci návod pre použitie rozširujúcich HTML atribútov pre pozíciovanie.

7.2.2.1 Problém s margin

Pre jednotlivé objekty tvoriace bunky tabuľkovej kostry, do ktorej sa umiestňujú iné objekty je možné okraje formátovať pomocou vlastnosti *padding*, a takto nastaviť presnejšie umiestnenie zobrazovaného objektu v bunke. Vlastnosť *margin* nie je podporovaná, jej nastavenie pre objekt bunky tabuľkovej kostry nie je povolené, z dôvodu nesprávnej a nerovnakej podpory rôznymi prehliadačmi, ako aj nepotrebnosťou vlastnosti, ktorú v prípade tabuľkovej bunky zastupuje vlastnosť padding, vhodná pre definíciu vzdialenosti okraja umiestňovaného objektu od okraja bunky.

Rovnaké správanie sa vlastnosti *padding* bolo potrebné zabezpečiť funkciou, ktorá ku každej stránke doplní neštandardnú CSS vlastnosť (*moz-box-sizing: border-box; box-sizing: border-box;*) zaručujúcu rovnaké spracovanie vlastnosti padding rôznymi prehliadačmi.

7.2.3 Kompatibilita s prehliadačmi

Aplikácia *layoutm.js* bola testovaná na viacerých dostupných prehliadačoch, a to Internet Explorer, Mozilla Firefox a Opera. Aplikácia je vo všetkých prehliadačoch podporovaná rovnako a správne, až na niektoré špecifické výnimky, ku ktorým prichádza z dôvodu nie vždy správnej a pri každom prehliadači rovnakej podpory normy CSS 2.1 či špecifikácie objektového modelu dokumentu.

V prípade, že žiadnej z buniek tvoriacich tabuľkovú kostru stránky nepriradíme nejakú nenulovú výšku (v podstate na konkrétnej šírke nezáleží, keďže ju náš *layoutm.js* skript prepíše) náš layout manager nebude fungovať v prehliadači Internet Explorer, z dôvodu odlišnej implementácie funkcií objektového modelu dokumentu *element.clientHeight*, a *element.clientWidth*, ktorý v Exploreri v tomto prípade vracia ako výšku, vždy hodnotu nula.

7.2.3.1 Validita stránok

Kvôli rozšíreniam HTML novými atribútmi nie sú stránky vytvorené s rozšíreniami pre umiestňovanie objektov pomocou *layoutm.js* validné. Validátor chápe rozšírenia ako podľa príslušných noriem nešpecifikované atribúty, a teda ich označuje za chybu.

Tento problém nenastáva v nasledujúcich postupoch rozmiestnenia objektov využitím webového editora doplnkových vlastností pre pozíciovanie, ani v prípade pozíciovania využitím grafického editora pre rozmiestenie objektov. V prípade týchto nasledujúcich postupov sa samotné rozmiestnenie objektov na zobrazovanej stránke rieši použitím aplikácií generovaných CSS kódu. Nie je potrebné, aby stránka obsahovala akékoľvek neštandardné vlastnosti.

7.3 Implementovanie webového editora pozíciovania pomocou HTML atribútov

Vytvorená webová aplikácia *layout code editor* umožňuje rozloženie objektov na stránke s využitím rozširujúcich vlastností pozíciovania, ktoré boli navrhnuté v kapitole 6.1. (časť 6.1.5.), a ktoré sa využívajú aj pri určovaní polohy objektu s využitím správcu rozloženia *layoutm.js* (viz. kap. 7.2.).

Editor pozostáva z troch hlavných častí, ktoré zastávajú funkcie, textového poľa pre zadávanie kódu užívateľom, potrebného pre požadované rozmiestnenie objektov na stránke. Druhou je vložený *frame* v stránke, do ktorého sa zobrazuje výsledné zobrazenie podľa popisu užívateľa zadaného už do spomínaného textového poľa pre vstupný kód. Treťou dôležitou súčasťou editora je textové pole kde sa zobrazuje výsledný CSS kód potrebný pre pozíciovanie objektov na stránke.

Editor *layout code editor* po zadaní vstupného kódu pre rozmiestnenie objektov na stránke tento na príkaz (stlačenie tlačidla *Compile*) vyhodnotí pomocou JavaScriptovej aplikácie podobnej *layoutm.js*, a výsledné rozloženie pošle pre vizualizáciu do vloženého frame okna stránky, v ktorom sa výsledok použitím JavaScriptu a vlastností *frame* zobrazí, a súčasne sa samotný vytvorený CSS kód potrebný pre toto zobrazenie pošle do textového poľa pre zobrazenie výsledného CSS kódu pre pozíciovanie v textovej (prehliadačom nespracovanej) podobe.

Aplikácia bola testovaná pre aktuálne verzie prehliadačov Internet Explorer a Mozilla Firefox a Opera.

7.3.1 Princíp fungovania webového editora

Editor *layout code editor* funguje na podobnom princípe ako Try It editory, ktoré sa vyskytujú na internete a umožňujú skúšať správnosť fungovania daného kódu, (príkladom takéhoto editoru je Try It editor od W3C school pre skúšanie JavaScript aplikácií, pozri [19]). Tieto editory pozostávajú väčšinou z dvoch častí z textového poľa pre užívateľský vstup. Vloženie zdrojového kódu, ktorý chcem preložiť a okna tvoreného s frame, kde sa zobrazuje výsledok. Náš editor je ešte doplnený o tretiu časť, o textové pole, kde sa zobrazí výsledný CSS kód pre rozloženie objektov.

Správne fungovanie editora je zabezpečené využitím programovacieho jazyka JavaScript, objektového modelu dokumentu a vlastností prvku frame. Užívateľ vloží vstupný kód, ktorý obsahuje minimálne popis pre rozloženie objektov na stránke (odpovedajúc popisom s kapitol 6.1. a 7.2.), ako aj prípadné ďalšie vlastnosti potrebné pre popis objektu. Podporované sú všetky skriptovacie a iné jazyky, pre ktoré platí, že ich preklad sa deje na klientskej strane a je podporovaný daným internetovým prehliadačom, kde je editor spustený. Väčšinou to sú jazyky HTML, CSS a JavaScript. Potom, čo sa vložil vstupný kód, sa samotná aplikácia spustí kliknutím na tlačidlo *Compile*. Týmto kliknutím sa zavolá k stránke pripojený JavaScript *compiler.js*, ktorý má za úlohu zobrať užívateľom

zadaný vstup z textového poľa, a vložiť ho ako časť novo vznikajúceho kódu stránky, ktorá sa potom zobrazí vo vloženom frame okne stránky. JavaScript *compiler.js* vytvorí nový *iframe* pomocou:

var iframe = window.frames.resultbox;

do ktorého sa postupne vloží s využitím *document.write* najprv hlavička vznikajúcej stránky, ktorá obsahuje pripojenie upravenej verzie skriptu *layoutm.js* (pozri 7.1.). Časť pre telo stránky (*body*) bude doplnená o volanie upravenej funkcie *layout_maker()* (pozri 7.1.). Samotné telo stránky je tvorené práve kódom zadaným užívateľom, ktorý sa vloží na toto miesto. Takto vytvorená stránka sa vloží do frame okna na stránke editora a súčasne sa prevedie k nej pripojený upravený skript *layoutm.js* , ktorý zobrazí výsledné rozloženie objektov podobne ako v prípade implementovaného rozloženia prvkov pomocou HTML atribútov (kap. 7.1.). Samotný skript je upravený a obohatený o funkcie zaisťujúce zobrazenie vytvoreného CSS kódu pre rozmiestnenie objektov podľa zadaných rozširujúcich vlastností užívateľa do predurčeného textového poľa na stránke editora.

Screenshot aplikácie je zobrazený v prílohách.

7.3.2 Špecifikácia používania

Editor dokáže správne pracovať so vstupným textovým súborom, pokiaľ preňho platia pravidlá popísané v kapitole 7.2. o používaní doplňujúcich HTML atribútov pre pozíciovanie objektov, a ďalšie časti vstupného kódu sú podporované prehliadačom kde je aplikácia spustená.

Vstupný kód užívateľ vloží do predurčeného textového poľa. Kliknutím tlačidla Compile sa vstupný kód vykoná, zobrazí sa rozloženie objektov, a v textovom okne pre CSS výstup bude zobrazený novovzniknutý CSS kód zabezpečujúci správne rozloženie objektov podľa užívateľových špecifikácií.

7.3.3 Kompatibilita s prehliadačmi

Webový editor *layout code editor* pre pozíciovanie bol testovaný na viacerých dostupných prehliadačoch, na aktuálnych verziách prehliadačov Internet Explorer, Mozilla Firefox a Opera. Pre samotnú aplikáciu platia rovnaké pravidlá a obmedzenia ohľadne kompatibility s prehliadačmi ako sú uvedené v kapitole 7.2.3.

7.4 Implementovanie grafického editora pre rozloženie objektov na stránke

Implementovaný grafický editor *LayIt graphical editor v.0.1* umožňuje užívateľovi vytvorenie mriežkovej štruktúry a vkladanie objektov do nej, s cieľom vytvorenia presného rozloženia prvkov na báze mriežkového rozloženia podľa vytvoreného návrhu, uvedeného v kapitole 6.2.

Samotný grafický editor je vytvorený kombináciou HTML CSS a JavaScriptu využívajúc objektový model dokumentu. Editor je dostupný na HTML stránke *layit_editor.html* a samotné rozloženie objektov a tvorba mriežky sa realizuje pomocou jednotlivých funkcií pripojeného JavaScriptu *grid_generator.js*, ktorý v sebe obsahuje potrebné funkcie pre riešenie problematiky grafickej editácie rozloženia objektov, ktorá je popísaná v analýze (6.2.).

Práve spomínaný JavaScript zabezpečí správne správanie sa editora pri tvorbe rozloženia objektov ako aj pri samotnom exporte vzniknutého rozloženia do CSS kódu.

7.4.1 Princíp fungovania grafického editora

Grafický editor je založený na možnostiach poskytnutých objektovým modelom dokumentu a vlastnostiach jazyka JavaScript. Pri spustení aplikácie sa od užívateľa vyžaduje definovať presný počet stĺpcov a riadkov mriežky, ktorá po budúcich úpravách bude slúžiť ako základ pre rozloženie objektov na stránke. Na základe týchto údajov sa po spustení vykreslenia mriežky (kliknutím na tlačidlo *Generate grid*) v JavaScripte *grid_generator.js* vykonajú potrebné funkcie pre vykreslenie základnej mriežky daných rozmerov.

7.4.1.1 Vykreslenie základnej mriežky

Samotné vykreslenie je proces, pri ktorom sa vytvoria a pomocou absolútneho pozíciovania umiestnia vedľa seba vytvorené *div* objekty, reprezentujúce jednotlivé bunky základnej mriežky. Bunky sa umiestňujú v pôvodnej DOM štruktúre stránky ako nové synovské uzly pre uzol *grid_container*. Pre každú takto vytvorenú bunku, *div* objekt, platia rovnaké základné vlastnosti. Prednastavená veľkosť buniek je 220x150 pixlov. Každá z buniek obsahuje ikonky pre zmenu veľkosti bunky (obrázok 7.2.) a pre vkladanie nových objektov do bunky (obrázok 7.3.). Každá z buniek, okrem tých, ktoré sa nachádzajú v pravom stĺpci, obsahuje ikonku pre spojenie s ďalšou bunkou napravo (obrázok 7.4.), a každá z buniek okrem tých, ktoré sa nachádzajú v poslednom riadku, obsahuje ikonku (obrázok 7.5.) pre spojenie bunky s bunkou, ktorá sa nachádza pod ňou.

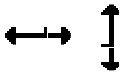

Obr. 7.2.


Obr. 7.3.


Obr. 7.4.


Obr. 7.5.

Popri vykreslení mriežky sa tiež vykreslia okolo vytvorenej mriežky posúvacie prvky (obrázok 7.6.), umožňujúce zmenu veľkosti vždy pre konkrétny riadok alebo stĺpec, vedľa ktorého sa nachádzajú.



Obr. 7.6.

Tieto posúvače spĺňajú úlohu pre uľahčenie jednotného upravovania veľkosti daného riadku či stĺpca mriežky a tiež sú potrebné v prípade zmeny veľkosti pospájanej bunky.

Po vykreslení mriežky ako aj počas ďalšej práce s ňou sa v hornom ovládacom paneli zobrazí vždy práve aktuálna celková veľkosť zobrazovanej mriežky.

7.4.1.2 Rozpoznávanie požiadaviek užívateľa

Samotný editor pracuje pri tvorení rozloženia objektov a tvorbe mriežky s množstvom ikoniek, pre ktoré je potrebné určiť kedy s ktorou užívateľ pracuje a aká činnosť sa má vykonať. Pre zaistenie tejto požiadavky sa pri načítaní stránky volá funkcia *grabID()*, ktorá pri každom pohybe s myškou zaznamenáva jej aktuálnu polohu a aký objekt sa pod ňou nachádza. V prípade, že užívateľ myškou klikne, sa zavolá ďalšia funkcia *idnetify_operation()*, ktorá na základe hodnoty získanej funkciou *grab_id()* rozhodne, čo za činnosť sa má vykonať a zavolá potrebné ďalšie funkcie.

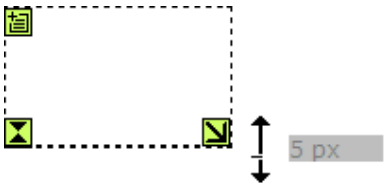
Samotná vytvorená mriežka zachováva svoju maticovú podobu, aj vďaka spôsobu pomenovávania jednotlivých *div* objektov (správne priradená hodnota *id*), ktoré ju tvoria a priradenia týchto objektov aj použitých ikoniek, ktoré sa v bunkách nachádzajú do predurčených skupín (hodnota *class*). Pre skupinové zaraďovanie objektov platí, že objekty rovnakého druhu patria do rovnakej skupiny (napr. všetky ikonky pre pridávanie objektov do mriežky sú *class="append_content"*). Toto skupinové zaradenie daného objektu sa využíva pri určovaní (pomocou funkcie *idnetify_operation()*) aký typ činnosti sa má pri kliknutí myškou na daný objekt vykonať. Samotné unikátne *id* jednotlivých objektov sa potom použije v jednotlivých funkciách, ktoré sa volajú, aby sa určilo, s ktorým konkrétnym objektom sa má užívateľom požadovaná činnosť uskutočniť. Pre *id* identifikujúce jednotlivé bunky mriežky platí, že je tvorené akýmsi číselným kódom, z ktorého si aplikácia vie odvodiť súradnice, o ktorú bunku mriežky sa jedná. Napríklad mriežka s *id="1.04cr"* značí, že sa bunka nachádza v 2. stĺpci (vyplýva z hodnoty 1. pred bodkou, značiacu číslo stĺpca číslované od nuly), a v 5. riadku mriežky (značené číslom 04 za bodkou, čísluje sa opäť od nuly). Toto presné značenie buniek hrá kľúčovú rolu pri spájaní buniek mriežky.

7.4.1.3 Zmena veľkosti vytvorenej mriežky

Pre vytvorené mriežky platí, že pokiaľ nevznikli spojením viacerých buniek, sa ich veľkosť dá meniť pomocou ikonky v pravom dolnom rohu (obrázok 7.2.). V inom prípade je možné ich veľkosť meniť pomocou posúvacích prvkov (obrázok 7.6.).

Pri zmene veľkosti využitím ikonky v pravom dolnom rohu danej bunky, s ktorou pracujeme sa počas trvania zmeny veľkosti zobrazí v bunke vždy práve aktuálna veľkosť bunky (obrázok 7.7.). Pri samotnej zmene veľkosti sa počas trvania zmeny mení plynule veľkosť bunky, s ktorou pracuje, kým veľkosť ostatných buniek sa prispôsobí až po dokončení činnosti (potreba popísaných činností sa podrobne rozoberá v kapitole 6.2.2.). Zmenu veľkosti mriežky zaisťuje viacero funkcií, ktoré menia jednotlivé polohové (CSS vlastnosti *top* a *left*) a veľkostné vlastnosti (CSS vlastnosti *width* a *height*) *div* objektov, ktoré mriežku tvoria.

Obr. 7.7.



Obr. 7.8.

Druhým spôsobom zmeny veľkosti bunky, je zmena pomocou posúvacích prvkov, pri výbere ktorých sa umožňuje zmena veľkosti vybraného stĺpca či riadku. Počas zmeny veľkosti riadku či stĺpca sa vedľa posúvača zobrazí aktuálny údaj o koľko pixlov daný riadok alebo stĺpec narastá (obrázok 7.8.), prípadne sa zmenšuje (zobrazené ako záporná hodnota). Tento spôsob zmeny veľkosti má hlavne slúžiť ako doplnková možnosť zmeny veľkosti umožňujúca zmenu veľkosti bunky, ktorá vznikla spojením viacerých buniek, a teda sa už pri nej neposkytuje možnosť zmeny veľkosti (podrobné vysvetlenie, pozri 6.2.2.) pomocou ikonky v pravom dolnom rohu bunky.

Pri zmene veľkosti bunky, ktorá už obsahuje nejaký vložený objekt sa na postupe nič nemení a ďalej sa riadime podľa pravidiel z kapitoly 6.2.2

Pre vytvorené bunky platí, že ich minimálna povolená veľkosť je 70x70 pixlov (v prípade potreby zmeny sa hodnoty dajú upraviť vo funkcii *get_possition_on_resize()*).

7.4.1.4 Spájanie buniek

Vytvorené bunky mriežky obsahujú ikony (obrázky 7.4. a 7.5.) umožňujúce ich spojenie do väčšej bunky siahajúcej cez viaceré riadky či stĺpce mriežky. V prípade takéhoto jednoduchého spojenia sú dodržané princípy a pravidlá, ktoré sú podrobne rozpísané v kapitole 6.2.3. Samotné jednoduché spájanie buniek je založené na princípe, že sa bunka ,ktorá má byť pripojená k inej bunke zoberie a jej pôvodná DOM štruktúra sa odoberie otcovskému prvku (teda uzlu *grid_container* pozri 7.4.1.) a vloží sa ako nový synovský uzol bunky ku ktorej ju pripájame. Novovzniknutej otcovskej bunke sa upravia jej rozmery (*width* alebo *height*, závisiac od smeru, ktorým spájam) o veľkosť pričlenenej bunky. Tento postup je potrebný pre zaistenie správneho správania sa mriežky pri budúcich zmenách. V budúcnosti, keď nastanú zmeny týkajúce sa aj bunky mriežky, ktorá je pripojená k nejakej ďalšej bunke mriežky (napr. meníme veľkosť stĺpca, ktorý obsahuje aj bunku pripojenú k inej bunke mimo stĺpca), sa vďaka overovaniu otcovského uzla DOM štruktúry vie, že sa potrebné zmeny majú uplatniť nie pre samotnú bunku, ale pre jej otcovský uzol, teda bunku, ku ktorej je pripojená.

Jednoznačná a presná identifikácia každej bunky je zabezpečená pomocou jej *id*, ktoré je potrebné aj pre určenie vzťahov otec-syn medzi spojenými bunkami. Umožní určiť vzájomné polohy dvoch spojených buniek (bunky sú v rovnakom riadku alebo stĺpci), aby sa potrebné zmeny správne uplatnili (napr. v prípade, že mením veľkosť stĺpca, ktorý obsahuje aj bunku pripojenú k inej bunke mimo stĺpca aplikácia zistí, že sa u otcovského objektu má uplatniť zmena jeho šírky).

V prípade komplexného spájania prebieha podobný proces dodržiavajúc zásady uvedené v kapitole 8.2.3., pričom sa ale pri spájaní v jednom kroku pracuje s viacerými bunkami mriežky.

Pre prípady, keď sa spojujú bunky, ktoré už v sebe obsahujú vložený objekt, sa tieto zachovávajú, a v prípade, že daný objekt pôvodne patril k bunke, ktorá sa pripája (mení sa na synovskú bunku), sa tento objekt odoberie danej bunke, a vloží sa ako posledný z objektov obsiahnutých v novovzniknutej spojenej bunke, (ktorá je otcovskou bunkou).

7.4.1.5 Vkladanie objektov do bunky

Pri vkladaní objektov do danej vybranej bunky (pre podrobnú špecifikáciu postupu pozri 6.2.4.) sa volí daná ikonka obsiahnutá v každej z buniek mriežky a užívateľovi sa zobrazí okno s textovým poľom, kam sa má vložiť zdrojový kód daného objektu, ktorý sa má do danej bunky umiestniť. Užívateľovi sa ponúka možnosť uložiť (tlačidlo *Save element*) vytvorený objekt, alebo návrat bez akejkoľvek vykonanej zmeny (tlačidlo *Cancel*) oproti stavu pred voľbou pre vloženie objektu.

V prípade, že užívateľ zadá kód pre vytvorenie nejakého objektu a daný objekt uloží, aplikácia zoberie vložený kód a ten vloží do novovytvoreného *div* objektu, ktorý sa ako plávajúci prvok vloží do vybranej bunky mriežky, ako jej synovský prvok. Samotný *div* objekt ,do ktorého sa objekt vloží, je potrebný pre umožnenie budúceho narábania s vytvoreným objektom. Potom, čo sa takto vytvorený objekt vložil do danej bunky, sa v objekte zobrazia ikonky pre ďalšie operácie s objektom. Vytvorený objekt je možné editovať (ikonka znázornená na obrázku 7.9), odstrániť (obrázok 7.10) ako aj premiestniť (obrázok 7.11).


Obr. 7.9.


Obr. 7.10.


Obr. 7.11.

Editácia vytvoreného objektu umožní znovunačítanie objektu do textového poľa, kde má užívateľ možnosť zdrojový kód objektu editovať a prípadné zmeny uložiť (tlačidlo *Save element*), alebo sa vráti do stavu spred editácie (tlačidlo *Cancel*).

Voľba odstránenia vybraného objektu po kontrolnom overení voľby odstráni z bunky daný synovský objekt.

Možnosť premiestňovať objekt znamená, že sa užívateľovi poskytne možnosť daný objekt za ikonku uchopiť a premiestniť do ľubovoľnej inej bunky vytvorenej mriežky. Pri voľbe tejto možnosti sa objekt preradí ako nový synovský uzol bunky, nad ktorou ju užívateľ upustí (známe aj ako "*drag and drop*"). V prípade, že užívateľ objekt upustí mimo mriežky, sa zmena nekoná, a objekt bude naďalej zaradený ako synovský uzol pôvodnej bunky. Ďalším využitím možnosti premiestňovania

objektu je zmena poradia objektov v prípade, keď jedna bunka obsahuje viacero objektov. Vložené objekty v jednej bunke sú radené podľa poradia ich pridania ako synovského uzla objektu danej bunky. V prípade uchopenia a upustenia daného objektu bunky, sa tá odoberie a znovu pridá ako synovský uzol objektu bunky, a teda sa zaradí na posledné miesto.

7.4.1.6 Exportovanie rozložených zoskupení

Po dokončení rozmiestnenia objektov na stránke sa užívateľovi poskytuje možnosť vytvorené rozmiestnenie objektov exportovať vo forme zdrojového kódu, ktorý bude obsahovať ako zdrojové kódy popisujúce jednotlivé objekty vložené užívateľom, tak aj dodatočný CSS kód vytvorený aplikáciou potrebný pre správne pozíciovanie objektov na stránke.

Samotný export volá funkciu, ktorá z vytvoreného objektového rozloženia odstráni všetky nadbytočné pomocné prvky, akými sú zobrazená mriežka a jednotlivé ikony pre voľbu operácií nad objektmi či mriežkou v rozložení. Od nadbytočnosti očistený zdrojový kód objektov a kód rozloženia sa potom zobrazí užívateľovi do textového poľa, odkiaľ ho užívateľ môže vyňať a umiestniť do svojej vytváranej web aplikácie.

7.4.1.7 Náhľad

Počas tvorby aplikácie sa užívateľovi poskytuje možnosť vypínať a zapínať zobrazovanie pomocných prvkov (tlačidlo *Design Off* / *Design On*) pri rozmiestnení, teda ohraničenia mriežky, a ikonky. Táto možnosť umožňuje si prezeranie objektového rozloženia po jej vložení do klientskej aplikácie.

7.4.2 Špecifikácia používania

Grafický editor *LayIt graphical editor v.0.1* vyžaduje pre svoju správnu prácu aby, mal prehliadač povolenú podporu pre JavaScript. Samotný editor pracuje podľa špecifikácií určených v kapitole 6.2. a popisu z predchádzajúcej časti 7.4.1.

Pre vytvorenie požadovaného rozloženia objektov musí užívateľ vytvoriť požadovanú základnú mriežku určením počtu stĺpcov a riadkov. Takto vzniknutú mriežku potom môže ďalej upraviť upravením veľkostí jednotlivých buniek mriežky ako aj ich spájaním medzi sebou. Po vytvorení požadovanej mriežky sa do jednotlivých buniek mriežky vložia požadované objekty pre rozloženie. Po rozložení požadovaných objektov sa výsledné rozloženie exportuje vo forme zdrojových kódov, ktoré užívateľ môže vložiť do svojej aplikácie. Aplikácia obsahuje aj stručný návod na použitie (tlačidlo *Help*).

Screenshot aplikácie je zobrazený v prílohách.

7.4.3 Kompatibilita s prehliadačmi

Grafický editor *LayIt graphical editor v.0.1* je podporovaný aktuálnymi verziami prehliadačov Internet Explorer, Mozilla Firefox a Opera. Vo vymenovaných prehliadačoch aplikácia pracuje

rovnako a správne, vyskytujú sa drobné odlišnosti, ku ktorým prichádza z dôvodu nie vždy správnej a pri každom prehliadači rovnakej podpory normy CSS 2.1 či špecifikácie objektového modelu dokumentu. Tieto ale nemajú vplyv na správne fungovanie aplikácie.

8. Záver

Práca obsahuje popis problematiky pozíciovania, návrh na jej riešenie a realizáciu riešenia s využitím možností jazyka JavaScript.

8.1 Vyhodnotenie výsledkov

V práci boli popísané vlastnosti kaskádových štýlov, norma CSS 2.1. popisujúca pravidlá pre prácu s kaskádovými štýlmi, jej špecifiká a charakteristiky, so zameraním na podrobný popis problematiky využitia CSS pozíciovania (CSS vlastností pre určovanie polohy objektu na stránke). Po súhrne nedostatkov pozíciovania objektov s využitím možností kaskádových štýlov je uvedený popis možností rozloženia objektov v programovacom jazyku Java, s využitím jeho mriežkových správcov rozloženia GridLayout a GridBagLayout. Na základe štúdia možností mriežkového rozloženia v Jave sa vytvorili návrhy pre uplatnenie podobných princípov v prípade rozloženia objektov webovej stránky. Boli vytvorené viaceré návrhy pre zabezpečenie požadovaného rozloženia objektov stránky, s využitím novovytvorených HTML atribútov pre pozíciovanie, ako aj rozloženie objektov pomocou vytvorenia vhodného grafického editora.

V prípade riešenia pomocou rozširujúcich HTML atribútov pre pozíciovanie sa jedná o vytvorenie nových atribútov pre určenie polohovacích vlastností objektu, podobných vlastnostiam využívaným mriežkovým editorom GridBagLayout jazyka Java. Vytvorený návrh je riešený implementáciou JavaScript aplikácií pre prevod rozširujúcich neštandardných HTML atribútov pre pozíciovanie na CSS kód zabezpečujúci dané rozloženie. Vytvorenie CSS vlastností pre pozíciovanie je zabezpečené aplikáciou *layoutm.js*, ktorá sa pripája k vytvorenej webovej stránke, a pri jej načítaní sa vykoná JavaScript aplikácia potrebná pre tvorbu CSS kódu pre rozloženie objektov. Nevýhodou tejto metódy je, že v prípade zakázanej podpory JavaScriptu zo strany prehliadača sa aplikácia nevykoná, nedôjde k požadovanému rozmiestneniu objektov.

Druhým spôsobom zaistenia správneho rozloženia je vytvorený editor rozširujúcich HTML atribútov pre pozíciovanie, ktorý dané rozloženie prevedie na CSS kód a súčasne zobrazí užívateľovi náhľad tohto rozloženia. Výhodou editora je, že po jeho využití sa do samotnej aplikácie pridáva už len vytvorený CSS kód, bez potreby pripájania JavaScript aplikácie k stránke. Stránka tiež nemusí obsahovať ani neštandardné HTML rozšírenie pre pozíciovanie.

Pri riešení rozloženia objektov na stránke pomocou vytvoreného grafického editora sa jedná o aplikáciu umožňujúcu jednoduchým spôsobom rozložiť objekty na stránke do užívateľom vytvorenej mriežkovej kostry. Je poskytnutá aj možnosť editácie už vložených objektov. Takto vytvorené rozloženie sa potom exportuje do potrebného zdrojového kódu, obsahujúceho zdrojové kódy užívateľom vložených objektov obohatený o potrebné CSS vlastnosti zabezpečujúce ich správne

rozloženie na stránke. Pre zabezpečenie funkčnosti grafického editora je potrebné mať povolenú podporu pre JavaScript v prehliadači, kde aplikácia beží. Exportovaný zdrojový kód obsahujúci objekty spolu s ich požadovaným rozložením pre svoje správne zobrazenie už nevyžaduje podporu JavaScriptu.

Využitie aplikácií sa predpokladá pri webových aplikáciách kladúcich vyššie nároky na prehliadač, dá sa predpokladať podpora JavaScriptu.

8.2 Ďalšie možnosti vývoja

Pri samotnej problematike rozloženia objektov na stránke sa očakávajú mnohé zjednodušenia pomocou nových vlastností zatiaľ len pripravovanej normy CSS 3.

Z hľadiska vytvorených aplikácií by sa v budúcnosti dali vytvoriť ďalšie web aplikácie pre jednoduchý spôsob tvorby základných a bežne používaných objektov webových stránok (vytvorenie možnosti ako si jednoducho bez nutnosti ovládania programovacích techník vytvoriť potrebné základné webové objekty ako napr. tabuľka, jednoduché formuláre a pod.). Kombinácia aplikácie pre tvorbu objektov webovej stránky s vytvoreným grafickým editorom pre pozíciovanie objektov by poskytla možnosť tvorby jednoduchších webových stránok užívateľom bez požiadavky, aby ovládali potrebné programovacie techniky.

Literatúra

[1] Budd A. CSS : filtry, hacky a pokročilé postupy Zoner Press, 2007, ISBN: 978-80-86815-54-1

[2] Dudek J. Interval.cz, CSS 2 pozivováni elementů, , Stránka je dostupná online na URL: http://interval.cz/clanky/css2-pozicovani-elementu/ (máj 2007)

[3] Hruška T., Burget R.: Internetové aplikace II, 2007. Dokument je dostupný online na URL: https://www.fit.vutbr.cz/study/courses/WAP/private/opory/OporaWAP2SGMLHTMLCSSDOM.pdf

[4] Hruška T.: Internetové aplikace VI, 2007. Dokument je dostupný online na URL: https://www.fit.vutbr.cz/study/courses/WAP/private/opory/OporaWAP6ProgramovaniKlienta.pdf

[5] Janovský D.: Jak psát web. Stránka je dostupná online na URL: http://www.jakpsatweb.cz (máj 2006)

[6] Nyman R., CSS Shortcomings, 2. 3. 2007, Stránka je dostupná online na URL: http://www.robertnyman.com/2006/09/06/css-shortcomings/ (máj 2007)

[7] Prokop M. Interval.cz, Přehled standardu W3C, , Stránka je dostupná online na URL: http://interval.cz/clanky/prehled-standardu-w3c/ (máj 2007)

[8] Škultéty R. JavaSript : programujeme internetové aplikace Computer Press, 2004, ISBN: 80-251-0144-4

[9] Staníček P. CSS Kaskádové styly, Computer Press, 2003, ISBN 80-7226-872-4

[10] Wyke-Smith Ch. CSS : využijte kaskádové styly naplno! , Computer Press, 2006, ISBN: 80-251-1297-7

[11] CSS3.info, CSS3 Preview, Stránka je dostupná online na URL: http://www.css3.info/preview/ (máj 2006)

[12] Mozilla developer center, DOM, Stránka je dostupná online na URL: http://developer.mozilla.org/en/docs/DOM (máj 2007)

[13] Mozilla.org, JavaScript-DOM Prototypes in Mozilla, Stránka je dostupná online na URL: http://www.mozilla.org/docs/dom/mozilla/protodoc.html (máj 2007)

[14] The Java Tutorials, How to Use GridBagLayout, Stránka je dostupná online na URL: http://java.sun.com/docs/books/tutorial/uiswing/layout/gridbag.html (máj 2007)

[15] W3C, Cascading Style Sheets level 2 revision 1, Stránka je dostupná online na URL: http://www.w3.org/TR/2006/WD-CSS21-20061106 (máj 2007)

[16] W3C, Document Object Model, Stránka je dostupná online na URL: http://www.w3.org/DOM/ (máj 2007)

[17] HTML, CSS and tables – the beauty of data, Stránka je dostupná online na URL: http://icant.co.uk/forreview/tables/ (november 2007)

[18] W3C, HTML table specification, Stránka je dostupná online na URL: http://www.w3.org/TR/html4/struct/tables.html (november 2007)

[19] W3C TryIt Editor v1.4, Stránka je dostupná online na URL: http://www.w3schools.com/js/tryit.asp?filename=tryjs_text (november 2007)

Zoznam príloh

Príloha 1 – Ukážky použitia *layoutm.js*

Nasledujúci jednoduchý formulár na obrázku P1.1 bol vytvorený pomocou layout manageru *layoutm.js*. Mimo CSS kódu popisujúceho obsahujúci blok, sa pre tvorbu mriežkovej kostry a pre popis v ňom zobrazených objektov použil kód:

```
h1 { text-align: center; font-size-adjust: 0.99; }
input, textarea {width: 100%; }
.nop {width: 20%; }
#kostra { background: #66cc00; width: 100px; padding: 4px; }
```

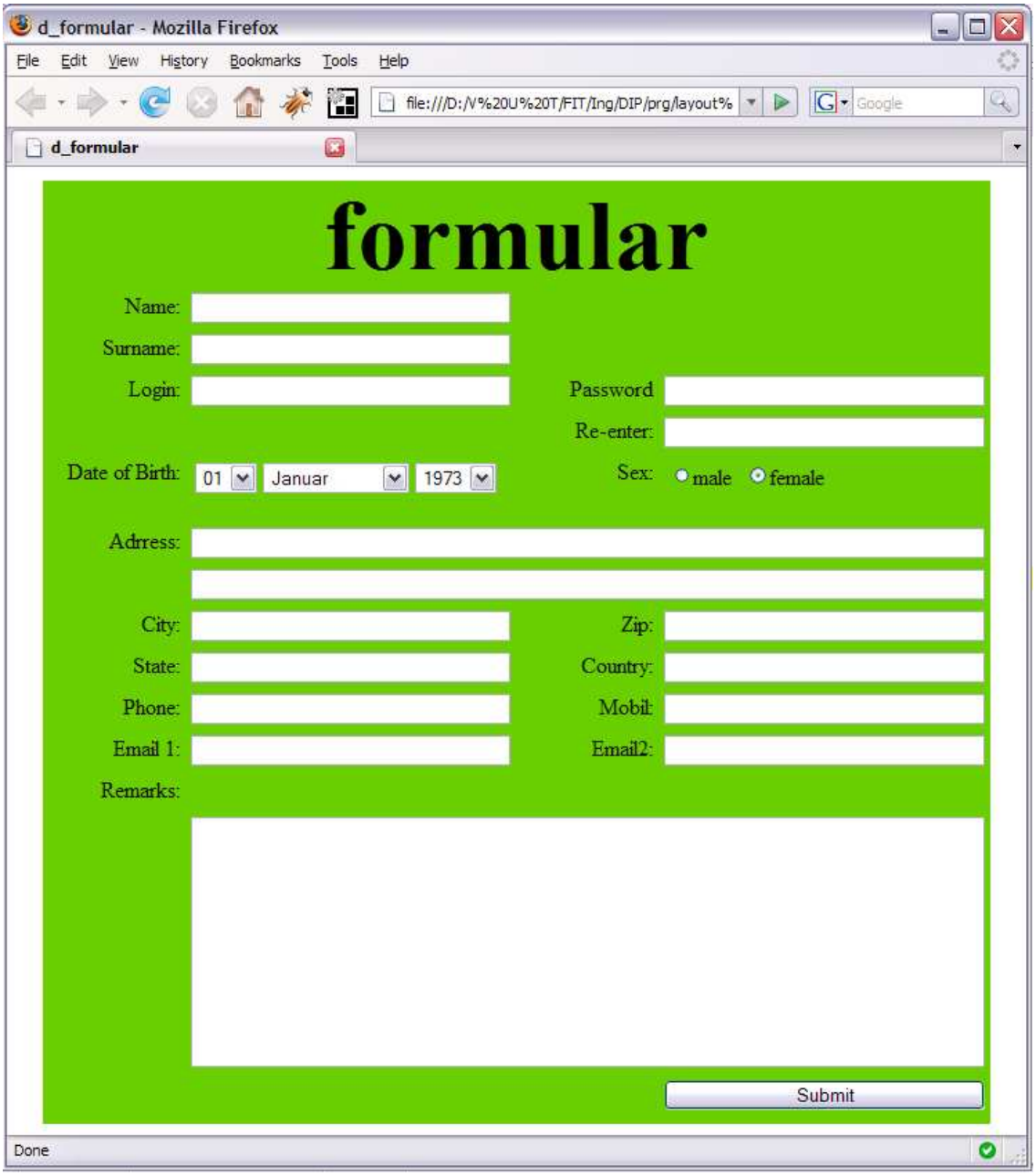
Samotný formulár zobrazený na obrázku P1.1 je vytvorený s použitím rozširujúcich vlastností pozíciovania pomocou pridaných HTML atribútov, ktoré slúžia pre mriežkové pozíciovanie objektov, a sú spracované pomocou JavaScript aplikácie *layoutm.js*.

Napríklad časť formulára určená pre udávanie mena užívateľa (*name*) je vytvorená pomocou rozširujúcich vlastností pre HTML pozíciovanie:

```
<div id="obj2" class="layoutobj" obj_x="0" obj_y="1" col_width="1">
        Name:
</div>

<div id="obj2" class="layoutobj" obj_x="1" obj_y="1" col_width="1">
        <input type="text" name="login" value="">
</div>
```

Z uvedených doplňujúcich HTML atribútov vyplýva, že sa jedná o prvky v prvom a druhom stĺpci druhého riadku vytvorenej mriežky. Podrobnejšie sa vytvorený formulár dá študovať v priložených demonštračných príkladoch, nachádzajúcich sa na CD prílohe v adresári “Demo príklady“.

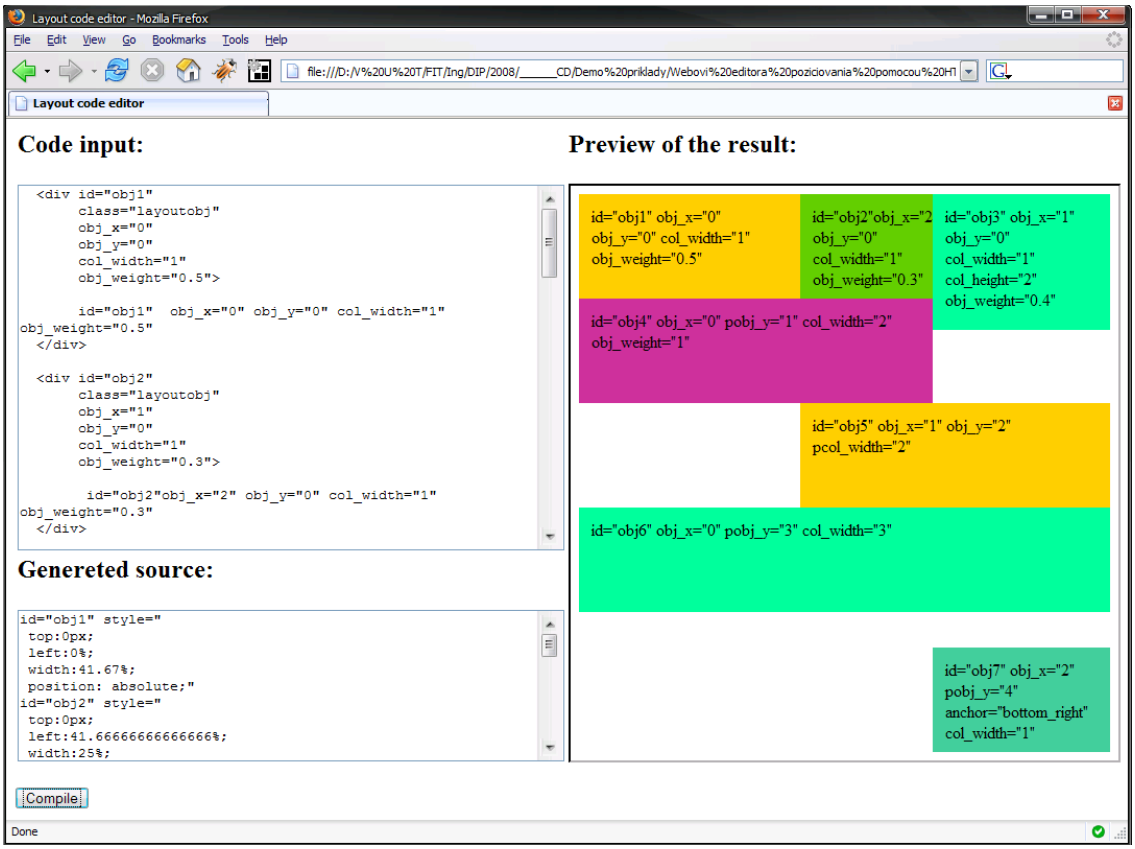

d_formular - Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///D:/V%20U%20T/FIT/Ing/DIP/prg/layout%
Google
d_formular
formular
Name:
Surname:
Login:
Password
Re-enter:
Date of Birth:
01
Januar
1973
Sex:
male
female
Adrress:
City:
Zip:
State:
Country:
Phone:
Mobil:
Email 1:
Email2:
Remarks:
Submit
Done

Obr. P1.1.

Príloha 2 – Screenshot aplikácia *layout code editor*

Na obrázku P2.1. je zobrazená aplikácia editora pre pozíciovanie pomocou rozširujúcich HTML atribútov. V pravom stĺpci je v hornom textovom poli vidieť vstupný zdrojový kód pre rozloženie objektov, v textovom poli pod ním sa nachádza generovaný výstup, zdrojový kód vstupných objektov obohatený o CSS vlastnosti pre požadované a zobrazované rozloženie a bez rozširujúcich HTML atribútov.

Rozloženie objektov s použitím vytvoreného výstupu pre samotnú aplikáciu, by zaistilo správne rozloženie bez potreby JavaScriptu a zdrojový kód, ktorý by neobsahoval žiadne neštandardné HTML vlastnosti.

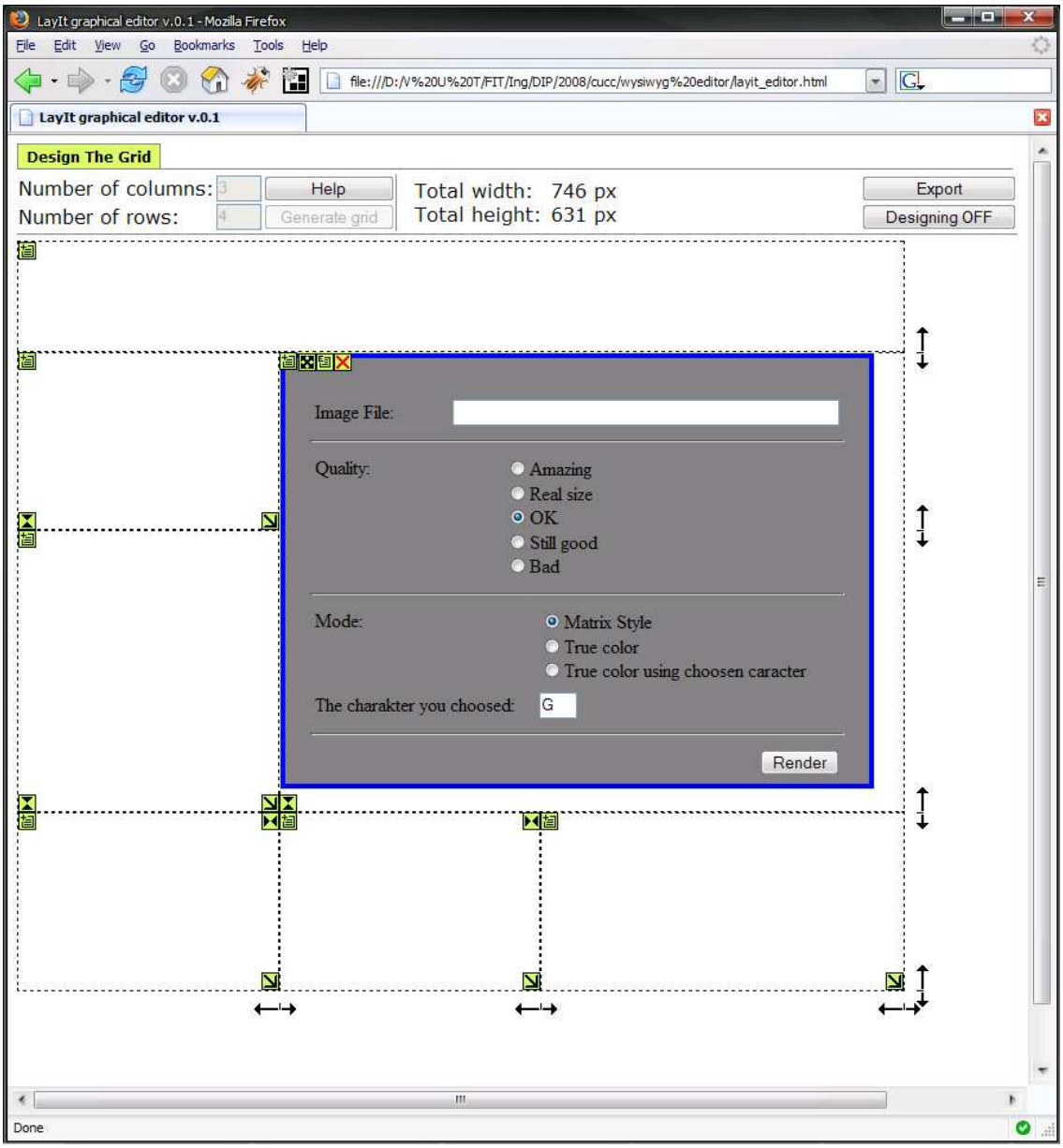


Obr. P2.1.

Zobrazený príklad je uvedený ako príklad dostupný na CD prílohe v adresári "Demo príklady".

Príloha 3 – Screenshot aplikácia *LayIt graphical editor v.0.1*

Na obrázku P3.1. je zobrazená aplikácia grafického editora *LayIt graphical editor v.0.1*. Je vidieť, že v aplikácii bola vytvorená základná mriežka o veľkosti 3 stĺpcov a 4 riadkov. Celková veľkosť vytvorenej mriežky je uvedená v hornom paneli, veľkosť zobrazenej mriežky je 756 pixlov na šírku a 631 pixlov na výšku.



Obr. P3.1.

Vytvorená mriežka už prešla zmenami veľkosti buniek (je vidieť, že jednotlivé riadky a stĺpce majú odlišné šírky a výšky, a teda bola využitá možnosť zmeny veľkosti stĺpcov i riadkov) aj spojením buniek. Najväčšia bunka mriežky, ktorá vznikla spojením štyroch pôvodných buniek mriežky v sebe obsahuje aj vložený objekt. Tento je tvorený zložitejšou tabuľkou, pre znázornenie, že sa v aplikácii dajú zobraziť a pracovať aj so zložitejšími objektami. Na snímke vidieť jednotlivé ovládacie prvky aplikácie.

Tabuľka ako zložitejší objekt je dostupná v ukážke pre aplikáciu grafického editora uloženej na CD prílohe v adresári “Demo príklady“.

Príloha 4 – Obsah CD

Súčasťou diplomovej práce je aj priložené CD, obsahujúce adresáre s obsahom:

- Demo príklady:
  - Grafický editor pre rozloženie objektov – adresár obsahuje ukážkový príklad pre využitie tohto editora.
  - Správca rozloženia layoutm.js – adresár obsahuje ukážky tvorby mriežky a rozloženia objektov využitím správcu rozloženia layoutm.js. Súčasťou adresára je aj manuál použitia layoutm.js
  - Webový editor pozíciovania pomocou HTML atribútov – adresár obsahuje ukážkový príklad pre prácu s webovým editorom
- Diplomová práca: -adresár obsahuje elektronickú podobu textovej časti diplomovej práce
- Zdrojové kódy programu:
  - Grafický editor pre rozloženie objektov – adresár obsahuje všetky zdrojové kódy vytvorenej aplikácie
  - Správca rozloženia layoutm.js – adresár obsahuje zdrojový súbor layoutm.js a manuál použitia správcu rozloženia layoutm.js
  - Webový editor pozíciovania pomocou HTML atribútov – zdrojové kódy aplikácie