

Софийски Университет "Св. Климент Охридски"
Факултет по Математика и Информатика
Катедра "Компютърна информатика"
Специализация "Информационни системи"

ДИПЛОМНА РАБОТА

на тема

**Управление на съдържанието на уеб
сайт посредством универсални шаблони**

Боян Николаев Бонев
фак. № 41469

Научен ръководител:
доц. д-р Владимир Димитров

Ръководител на катедра:
доц. д-р Димитър Добрев

София, 2002

СЪДЪРЖАНИЕ

Постановка на задачата	3
Задачи за изпълнение	3
Увод в темата	4
• Общи сведения за темата	4
• Обзор на основните концепции в разработката	6
• Благодарности	8
Глава 1 – Описание на модела	10
• Основни компоненти	10
• Език	11
• Графични изображения	11
• Файлове	12
• Шаблони	13
• Страници (блокове)	15
• Метаезик - дефиниция	16
• Метаезик - семантика	20
Глава 2 - Реализация	24
• Общи сведения	24
• Избор на програмен език и СУБД	24
• Представяне в релационна СУБД	27
• Метаезик	30
• Файлове и графични изображения	31
• Оптимизации за бързодействие	32
• Административен интерфейс	33
Глава 3 - Реализация в системата на конкретен интернет сайт	40
• Избор на пример от практиката	40
• Структура на сайта	40
• Реализация на вътрешни страници	43
• Реализация на блок меню	45
• Реализация на галерия от графични изображения	45
Заклучение	47
• Постигнати резултати	47
• Възможности за развитие	47
Приложение	49
• Изходен текст на подсистемата за обработка на метаезика	49
Литература	59

Тема на дипломната работа – управление на съдържанието на уеб сайт посредством универсални шаблони

Постановка на задачата

Да се разработи система за съставяне на универсални шаблони за представяне съдържанието на уеб сайт. Основната цел на системата е да се намери оптималния баланс между универсалност и простота на модела. Практическата приложимост на така избрания модел може да бъде проверена след реализацията му с използване на конкретни примери от практиката. Дефинирането на ролите в системата – управление на шаблони за форматиране и управление на съдържание се налага не само от разликите в изпълняваните дейности, но и в изискванията към двете роли. Управлението на данните в системата изисква познаване на самите данни, както и на структурата на системата, докато управлението на шаблоните за форматиране допълнително изисква сериозни познания по HTML, HTML 4 и DHTML.

Задачи за изпълнение

1. Избор на формален модел за описание на шаблоните и данните, към които се прилагат. Разделяне на данните от информацията за форматирането им още на ниво модел на описание.
2. Ясно разделяне на ролите в системата – управление на шаблони за форматиране и управление на съдържание.

3. Реализация на програмна система, включваща съхранение на шаблоните и данните в релационна СУБД, административни потребителски интерфейси за двете роли в системата и система за публикуване на съхранените вече данни в уеб.
4. Реализация в системата на конкретен интернет сайт.

Увод в темата

- **Общи сведения за темата**

Бързото развитие и широко навлизане в практиката на интернет уеб сайтовете, като основен начин за публикуване на информация, поставят нови изисквания към актуалността на информацията и технологиите за поддръжка. В началото на интернет ерата уеб сайтът представляваше съвкупност от статични страници, графични изображения и файлове. Обновяването му ставаше сравнително рядко от квалифицирани специалисти, добре запознати с технологията и езика HTML. За редактиране на съдържанието се използваха обикновени текстови редактори, без възможност за наблюдение на резултата при промяна (WYSIWYG – What You See Is What You Get). Като първа основна стъпка в развитието можем да разглеждаме въвеждането на програмен код от страна на уеб сървъра (CGI – Common Gateway Interface). Тази технология даде възможност за динамично съдържание, генерирано от програма и зависещо както от параметрите на конкретната заявка, така и от състоянието на база от данни или външни ресурси. Изнасянето на част от съдържанието в база от данни е първа стъпка към разделяне на данните от форматирането им в HTML. Все по-широкото навлизане и съответното разширяване на кръга от

хора създаващи или обновяващи съдържанието на уеб сайтове наложи WYSIWYG HTML редакторите. При тях редактирането е интерактивно, като се работи директно върху крайния вариант на страницата, без да е необходимо познаване на HTML и непрекъснато преглеждане на резултата. Тази стъпка напред още повече увеличи кръга от хора, създаващи и поддържащи уеб сайтове. Днес технологията еволюира до вграждане на езиците за програмиране на динамични уеб сайтове в самите уеб сървъри. Самите езици развиха своите възможности, включвайки в себе си всички основни примитиви на уеб програмирането – докато при CGI програмиста трябваше да се грижи както за логиката на приложението, така и за интерфейса със самия уеб сървър, днес всички рутинни дейности са почти напълно автоматизирани. Въпреки добрата интеграция между уеб сървър и език за програмиране и наличието на визуални HTML редактори, създаването и редактирането на един уеб сайт остава една непосилна задача за много голям процент от хора. Тенденцията към улесняване на публикуването на информация в уеб и сайтовете с динамично съдържание водят до разделянето на ролите на специалист по съдържанието на сайта и специалисти по HTML, уеб програмиране и външен вид. Това разделяне на ролите води и до разделяне на данните от тяхната презентация. Една първа стъпка в тази посока са специализираните административни интерфейси за редактиране на съдържанието. Основно предимство е тяхната тясна връзка с конкретното приложение. Недостатъците са два – неприложимост за друг сайт и дълго време за разработка. Целият този път на развитие води до необходимостта от универсален метод за управление на съдържанието. Основни

изисквания към него са: универсалност и леснота за работа на администратора на съдържанието.

- **Обзор на основните концепции в разработката**

Разглеждането на програмната реализация на повечето уеб сайтове ни води до прякото наблюдение, че страниците в него могат да бъдат разделени на една, две, рядко повече групи. Всяка група използва еднакви начало и край. Между тях се разполага динамично генерираното съдържание. Разликите в HTML кода за началото и края рядко са съществени в рамките на дадена група. Те обикновено са свързани с отразяването на избраната страница в навигацията, извеждане на заглавие или подзаглавие и други подобни. Съдържанието от своя страна е поредица от повтарящи се елементи с еднакво форматиране. Например една дипломна работа, публикувана в уеб е списък от следните елементи: *Параграф, Голямо Заглавие, Параграф, Параграф, Параграф, Заглавие, Подзаглавие, Параграф, Параграф, Подзаглавие, Параграф, Заглавие, Подзаглавие, Параграф* и т.н.

Не е трудно на база тези наблюдения да създадем един елементарен модел на шаблон за представяне съдържанието на група от подобни страници. Ако приемем, че имаме HTML кода на съдържанието, то шаблонът се състои от начало и край. Конкатенирайки началото, съдържанието и края получаваме кода на страницата. Към всеки елемент от съдържанието можем да приложим същият подход. Така група страници могат да бъдат форматираны, ако имаме основния шаблон и списъка с шаблоните за типовете елементи на съдържанието. В нашия пример с

публикуването на дипломни работи в уеб това са: *Голямо Заглавие, Заглавие, Подзаглавие, Параграф*. Съдържанието за всяка страница (в примера дипломна работа) можем да представим като списък от двойки тип и текст.

Разбира се, този опростен модел не е приложим в практиката и служи само като база на нашите разсъждения. Първият сериозен недостатък е невъзможността да представим елементи от съдържанието, състоящи се от повече от един компонент – например хипервръзка (състои се от текст и URL) или номерирани заглавия и подзаглавия с различно форматиране за текста и номерацията. Втори недостатък е, че в случаите с промяна в началото и края на страницата (например отразяване в навигацията коя е текущата страница) броят групи подобни страници става равен на броя на страниците. Често в практиката се наблюдава разполагането на един и същи блок от информация във всички групи страници. Трети недостатък на елементарния модел е невъзможността да се представят подобни блокове информация по удобен за редактиране начин. В елементарния модел подобни блокове биха влезли в кода на началото и края. За тях не би било възможно прилагането на подобна стратегия за разделяне на данните от форматирането.

Всичко това ни води до създаването на модел, който решава поставените проблеми. Първият се решава, като разширим представянето на всеки елемент до тип на елемента и списък от текстове. Съответно разширяваме шаблона до списък от $n+1$ елемента, ако броят текстове е n . На практика работата със списъци от списъци от елементи не е удобна. Това налага въвеждането на

метаезик и представянето както на елементите, така и на съответните шаблони като текст. Същата концепция прилагаме и към основния шаблон за всяка страница – вместо начало и край използваме един текст, в който с елемент на метаезика указваме къде трябва да бъде поставено съдържанието. За решаването на втория проблем, отново използваме метаезик, с който описваме разликите в шаблона в рамките на група страници. Третият проблем решаваме като въвеждаме в модела възможността за включване на страници една в друга. Така описанието на една страница става чрез елементарния модел, а за допълнителните възможности използваме метаезик или даже цели блокове, които от своя страна са самостоятелни страници. Този нов, разширен модел ни позволява да опишем всяка страница, като използването на допълнителни възможности и по-сложни операции зависи единствено от сложността на страницата – колкото е по-проста структурата на страницата, толкова е по-просто и представянето и. Като цяло разделянето на сайта на групи подобни страници, определянето на общите блокове и отделянето на съдържанието от форматирането ни позволява след като веднъж е създаден шаблона лесно и бързо да създаваме нови страници или да редактираме съдържанието на съществуващите.

- **Благодарности**

Искам да изкажа своите благодарности на катедра Компютърна Информатика и лично на ръководителя на катедрата доц. Димитър Добрев за оказаната помощ и съдействие през всичките години на обучението ми в тази катедра. Искам да благодаря и на научния ми ръководител доц. Владимир Димитров за полезните дискусии,

постоянната подкрепа и напътствия. Благодаря също и на моите колеги и приятели Петър Петров, Иван Йосифов и Иван Попиванов за ценните съвети и помощ.

Глава 1 – Описание на модела

- **Основни компоненти**

Структурирането на модела за представяне на съдържанието пряко отразява както структурата на уеб сайтовете, така и разделянето на форматирането от съдържанието.

Много често в практиката се налага съдържанието на един сайт да е на повече от един език. Това води до необходимост от вградена в модела поддръжка на многоезично съдържание. Това е основен недостатък на WYSIWYG HTML редакторите – управлението на съдържание на различни езици е оставено изцяло на потребителя и това често води до недобре структуриран резултат, липса на синхронизация в структурата или външния изглед между различните езици.

Все по-често в практиката (и особено в бизнес презентационните сайтове) се наблюдава преобладаващо графично съдържание. То се налага с цел постигане на по-добър външен вид, пряка връзка с фирмения имидж на собственика на сайта и/или връзка с рекламни материали. Като графични изображения предимно се представят рядко променящи съдържанието си части от сайта, докато често променящите се, се представят като обикновен текст. Многоезичността и факта, че като графични изображения се представят и текстове ни налага да обърнем специално внимание на графичните елементи в модела на представяне.

Освен текст и графични елементи в сайтовете се използват и файлове – било като части от самия сайт или информация за сваляне от потребителя на сайта.

Разбира се, не можем да пропуснем шаблоните за форматиране на съдържанието, както и блоковете (страници), които съдържат самото съдържание.

Разглеждаме следните основни елементи на модела:

- Език
- Графично изображение
- Файл
- Шаблон (основен и елементи)
- Страница (блок)

• Език

За да представим поддържаните езици в даден сайт, използваме списък от двубуквени кодове (например кодовете на държави по ISO). Тези кодове са просто етикети и могат да бъдат избрани от архитекта на системата в зависимост от конкретното приложение. Например в сайт със съдържание на български и английски езици, кодовете могат да бъдат *bg* и *en*. Наричаме кодовете идентификатори на езика. Идентификаторите трябва да са уникални за всеки различен език в системата.

• Графични изображения

Графичните изображения могат да се разделят на два вида – езиково неутрални и езиково зависими. Всяко графично изображение в системата има уникално име. Ако графичното изображение е езиково зависимо, то има възможност за задаване на различен файл за всеки език. Допуска се задаването и на езиково неутрално изображение, което би улеснило добавянето на нов език, без за него да има дефинирани всички езиково зависими изображения.

В представянето изображенията са множество от уникални имена. Зад всяко име стои списък от файлове за всеки език плюс езиково неутрално изображение.

В метаезика на системата връзката с дадено изображение се указва с неговото уникално име. Системата сама определя на база на текущия език и наличието на файл за този език и езиково независим файл, кой от тях да бъде използван. Налице е следното ограничение – езиково неутралния файл е задължителен за всички изображения, които нямат файл за всеки език.

- **Файлове**

Представянето на файловете е същото както и на графичните изображения, но не използва връзка с езика. Файловете в системата са езиково неутрални.

При необходимост може да бъде въведен в системата езиково зависим файл, но това трябва да се реализира с подходящо именуване и използване на метаезика при извикването или

свързването към файла. Най-естественият начин за това е включването в името на файла на идентификатора на езика.

- **Шаблони**

Шаблоните са основна градивна единица в модела на системата. Те се разделят на два основни вида – основен и множество от шаблони за всички различни елементи на дадения клас страници (или блокове). Всички те се дефинират за всеки дефиниран в системата език. В шаблона дефинираме различните типове елементи от класа блокове (страници), които използват този шаблон.

Всеки шаблон в системата притежава уникален текстов идентификатор. Към него свързваме множество от двойки идентификатор на език – основен шаблон за всички дефинирани езици и множество от уникални в рамките на шаблона текстови идентификатори на различните типове елементи от шаблона. Към всеки елемент от шаблона свързваме множество от двойки идентификатор на език – шаблон за форматиране на съответния елемент на дадения език.

Обработката на тагове от метаезика се различава при шаблоните на елементите. Там са дефинирани два допълнителни типа тагове – за разделяне на съдържанието на елементи (*Splitter*) и за вграждане на елементите от съдържанието в текста на шаблона (*CTags*). Тагът за разделяне единствено има смисъл, когато е използван в съдържанието на елемент от блок (страница), таговете за вграждане – в дефиницията на шаблон за елемент.

След конкатенацията на резултата от обработката на даден блок, резултата трябва да се вгради в основния шаблон. Дефиниран е трети специален таг, който указва точно това – къде да бъде вграден този текст в основния шаблон. Този специален таг (Content) има смисъл единствено в дефиницията на основен елемент от шаблона.

Всички тагове от метаезика, които са дошли от съдържанието на страницата или от елемента на шаблона или от основния шаблон остават необработени на този етап. Това изискване към реализацията ни е необходимо, за да можем да дефинираме тагове, които използват текущия блок, език или страница като контекст на своята логика на обработка. Така можем да определим реда, в който се обработват таговете. Разбира се, при реализацията може да се вземат различни решения – обработка на таговете по нива или разделяне на таговете на такива с незабавна или отложена обработка, в зависимост от това, от кой контекст зависи тяхната логика. Практиката показва, че реализация, която обработва незабавно единствено таговете, които не могат да бъдат обработени отложено, а на една последна стъпка обработва наведнъж всички останали, дава най-висока производителност.

- **Страници (блокове)**

Първо нека ясно дефинираме разликата между страница и блок. Страницата е блок, който се представя в браузъра на потребителя на уеб сайта. Страницата може да включва други блокове в своя код, както може и да играе ролята на блок при генерирането на друга страница. Елементарен пример от практиката може да бъде сайт, който съдържа недовършени (незапълнени със съдържание) страници с еднакво съобщение. Ако веднъж вече сме реализирали една такава страница, тя може да бъде включена като блок от всички останали с помощта на метаезика. Важно е да се отбележи, че проверката на валидността на генерирания HTML код е оставена изцяло на администратора на шаблоните. Полученият код не би бил валиден, ако в страница с поставени в шаблона начални и крайни HTML тагове се включи подобна на нея. Обратно – ако шаблона за страницата съдържа само тага `Content`, трябва включеният блок да съдържа тези тагове.

Всеки блок в системата се идентифицира с уникално име. За разлика от шаблоните и техните елементи тук имаме по-строги изисквания наложени от HTTP (Hyper Text Transfer Protocol) протокола, защото ще използваме идентификаторите на страници като част от адресирането на сайта (URL – Uniform Resource Locator). Разбира се, същото ограничение важи и за идентификаторите на езика, но тъй като те са дефинирани като двубуквени кодове, то те вече отговарят на това изискване.

Към всеки идентификатор на блок свързваме идентификатор на шаблон, набор от флагове – дали блока се използва като страница,

дали блока трябва да бъде намиран при търсене като самостоятелна страница и други, множество от тройки идентификатор на език, заглавие на страницата на съответния език и списък от двойки идентификатор на елемент и съдържание. Трябва да се отбележи, че идентификаторите на елементите от съдържанието са уникални само в рамките на даден шаблон. При създаването на даден блок се допуска използването на елементи само от избрания шаблон.

- **Метаезик - дефиниция**

Метаезика в системата е базиран като идеология на HTML, но основната му цел е простота, удобство за работа и пълна интеграция със системата за управление на съдържанието. За да се различават таговете на метаезика от HTML таговете, е приета минимална промяна – добавени са допълнителни символи за бързо отделяне на двата типа тагове.

В метаезика са въведени тагове за опростяване и автоматизиране на много често срещани задачи при разработката на уеб сайтове. Поради преобладаването на предимно графично ориентирани уеб сайтове, особено внимание е обърнато на обработката на графични изображения. Създаването на графична навигация, при която всяка хипервръзка се представя чрез графично изображение е максимално развита до степен на генериране от системата на JavaScript за обработка на събития от мишката при клиента. Създаването на галерия от графични изображения много често в практиката се прави като страница с малки изображения, които са хипервръзки. След избиране на малкото изображение се

автоматизира отварянето на прозорец с голямото графично изображение, със съответния размер. Това отново се реализира, като системата генерира JavaScript. Цялата тази автоматизация много ускорява разработването на уеб сайт в системата, както и неговата поддръжка в последствие. Администраторът на съдържанието е освободен от подробности от типа на размерите на изображенията, съвместимостта с различни видове браузъри, правилна реакция на резултатния HTML код, при условие, че браузъра на клиента не поддържа или е със забранен JavaScript.

Вграждането на блокове е реализирано чрез тагове на метаязика, които указват кой блок, на кой език да се форматира с даден шаблон и да се замести на мястото на тага. Цялата тази система позволява рекурсивно включване на блокове, като в програмния код са предвидени специални мерки за предотвратяване на безкрайна рекурсия.

Обработката на останалите тагове от езика се свежда до изпълняване на посоченото действие, като се вземат предвид контекста на извикването (име на страница, име на блок, текущ език) и параметрите в тага. Резултата от обработката на тага се замества на негово място. Допустимо е в реализацията с цел опростяване на обработката общите части да бъдат развити като тагове от метаязика. Тяхната обработка също се извършва рекурсивно до изчерпване на таговете от метаязика. Осигуряването на крайност на тази рекурсивна обработка става при реализацията на обработката на таговете. Правилото е да се присвои ниво на сложност на всеки таг или група от тагове, и когато се налага един таг да бъде развит с помощта на друг, да се

съблюдава да се използват само тагове от по-ниско ниво. Това ограничение не е сериозно, защото касае само реализацията, но не и общата работа на модела или класа решавани задачи, тъй като е възможно обработката на всеки таг да бъде развита в програмен код. Така обема на програмния код ще нарастне, но лесно може да се провери, че на всяка итерация броя необработени тагове ще намалява. Ако на дадена итерация не бъде обработен нито един таг, това означава, че междинният резултат не съдържа повече тагове от метаязика и обработката е приключила.

Метаязика ще дефинираме, използвайки Бекус-Наурови форми:

Splitter : "<>"

CTags : "<?text?>"
| "<?url?>"
| "<?tag_ " Number "?>"

Number : Digit
| Number Digit

Digit : 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Content : "<?content?>"

Tag : "<?" TagC "?>"

TagC : Page
| Lang
| Title
| Condpage

- | Condgpage
- | Link
- | File
- | Imgsrc
- | Imgwidth
- | Imgheight
- | Imgparams
- | Picture
- | Ipopup
- | Ipopup_script
- | Jscript

Page : "page" PageC

PageC :

- | "=" *pageid*
- | "=" *pageid* "," *langid*
- | "=" *pageid* "," *langid* "," *templateid*

Lang : "lang"

Title : "title"

Condpage : "condpage=" CondpageC

Condgpage : "condgpage=" CondpageC

CondpageC : *pageid* "," *text_yes* "," *text_no*

Link : "link=" *pageid*

File : "file=" *filename*

Imgsrc : "imgsrc=" *imagename*

Imgheight : "imgheight=" *imagename*

Imgwidth : "imgwidth=" *imagename*

Imgparams : "imgparams=" *imagename*

Picture : "picture=" PictureC

PictureC : *imagename*
| *imagename* "," *url*
| "!" *pageid* "," *imagename* "," *imagename* "," *url*
| *jsname* "," *imagename* "," *imagename* "," *url*
| "!" *pageid* "," *jsname* ","
| *imagename* "," *imagename* "," *imagename* "," *url*

Ipopup : "ipopup=" *imagename* "," *imagename*

Ipopup_script : "ipopup_script=" *imagename*

Jscript : "jscript"

• **Метаезик - семантика**

- специални тагове

Семантиката на специалните тагове – разделителя и таговете за вграждане разгледахме при описанието на шаблоните. Трябва да се отбележи, че таговете "<?text?>" и "<?url?>" са синоними на

"<?arg_1?>" и "<?arg_2?>" съответно. Помощния таг "<?javascript?>" има много близка семантика до "<?content?>" по отношение на това къде е валиден и кога се извършва неговата обработка. На негово място системата поставя генерирания JavaScript по време на събирането на текста на всички блокове, от които се състои страницата.

- елементарни тагове

Таговете "<?lang?>", "<?page?>", "<?title?>", "<?link=...?>", и "<?file=...?>" се обработват чрез елементарно заместване. Първият дава идентификатора на текущия език. Втория – идентификатора на текущия блок. Третия – дефинираното при страниците заглавие за текущия език. Последните два предоставят удобен начин да се изгради хипервръзка до страница или файл в системата, без да е необходимо познаване на вътрешната и структура. Автоматично се генерира URL до зададената страница или файл.

- условни тагове

"<?condpage=...?>" дава възможност да се избира между два текста в зависимост от идентификатора на текущия блок. Ако е необходимо да се направи същото нещо, но в контекста на текущата страница, вътре в блок се използва тага "<?condpage=...?>".

- помощни тагове за работа с графични изображения

Таговете "`<?imgsrc?>`", "`<?imgwidth?>`", "`<?imgheight?>`", и "`<?imgparams?>`" дават съответно пътя, ширината в пиксели, височината в пиксели и параметрите на HTML тага `img` за дадено графично изображение. Пътят може да се променя в зависимост от текущия език, а ако изображението не е дефинирано за всички езици то може да се променя и до изображението по подразбиране. Тага "`<?ipopup_script=imagename?>`" се заменя с JavaScript-а, който отваря в нов прозорец зададеното графично изображение.

- тагове за работа с графични изображения

"`<?picture=imagename?>`" – заменя се с HTML таг за включване на зададеното графично изображение

"`<?picture=imagename,url?>`" – заменя се с HTML таг за включване на зададеното графично изображение, което е хипервръзка към зададеното URL

"`<?picture=jsname,imagename,imagename,url?>`" – заменя се с HTML таг за включване на зададеното графично изображение, което е хипервръзка към зададеното URL. При преминаване с курсора на мишката над изображението, то се заменя с второто зададено изображение. Този ефект е много често използван за имитиране на бутони или поставяне на акцент върху текущо избран елемент от графичната навигация. *Jsname* е уникално име, използвано в генерирания JavaScript. Изнесено е на това ниво, а не се генерира автоматично от системата, за да може да се позволи по-голяма гъвкавост и свобода при необходимост от надграждане на вградените възможности.

"<?picture=!*pageid*,*jsname*,*imagename*,*imagename*,*imagename*,*url*?>" – този таг се заменя се с обикновено изображение (използва се третото от тага), ако текуща страница е *pageid*. В противен случай е еквивалентно на предишния таг с първото и второто изображения. Приложението е за директно генериране на графична навигация, в която всеко изображение има три състояния – нормално, избрано и забранено. Забраненото се налага, когато изображението е хипервръзка към дадена страница, а тя в момента е активна. Няма реално приложение на хипервръзка от една страница към самата нея. Също много често в практиката се налага по някакъв начин в навигацията ясно да се акцентира върху текущата страница – чрез смяна на графичното изображение.

"<?ipopup=*imagename*,*imagename*?>" – този таг се заменя се с HTML кода за първото изображение. То е връзка, която отваря второто изображение в нов прозорец. Генерирания код позволява, ако клиентската система поддържа JavaScript, новият прозорец да бъде оразмерен спрямо отваряното изображение, да бъде използван отново при повторно използване на подобна хипервръзка и други. При липса или забрана на поддръжката на JavaScript, генерирания от системата код отново работи, но без изброените подобрения.

Глава 2 - Реализация

- **Общи сведения**

При реализацията на системата за управление на съдържанието разглеждаме три основни модула – достъп до СУБД, административен интерфейс и публично видима част от сайта. Важно е административния интерфейс да бъде защитен с потребителско име и парола. Допълнително за повишаване на сигурността може да се използват HTTPS (Secure Hyper Text Transfer Protocol) протокол и/или ограничаване по IP (Internet Protocol) адрес.

За да бъде реализацията високо производителна, е необходимо да се използва такава комбинация от Web Server, език за програмиране и СУБД, при която езика е много добре интегриран както с Web Server-а, така и с клиентския интерфейс (API – Application Program Interface) за достъп на СУБД.

- **Избор на програмен език и СУБД**

След задълбочено проучване на възможните комбинации от език за програмиране, Web Server и СУБД от съществуващите в момента технологии разглеждаме:

- Web Server
 - Apache
 - Microsoft IIS

Предимствата на Apache са, че е достъпен свободно, работи под всички съвременни сървърни операционни системи и дава много добра производителност. Много добре се интегрира с програмния език PHP.

Предимствата на Microsoft IIS (Internet Information Server) са комерсиалната поддръжка, перфектната интеграция с операционната система и езиците за програмиране в ASP .NET (Active Server Pages .NET). Лесната администрация на Microsoft IIS го прави предпочитана система за повечето комерсиални решения. Microsoft IIS също може да бъде интегриран с езика за програмиране PHP.

- Език за програмиране
PHP
C#

Свободно достъпният език PHP се интегрира много добре както с Apache, така и с Microsoft IIS. Вградената поддръжка за достъп до СУБД и много добре структурираното API го правят много лесен за бързо изграждане на уеб решения. PHP е интерпретируем език, но това не пречи написаните на него приложения да са много високо производителни.

C# се интегрира перфектно както с Microsoft IIS, така и с ADO .NET (ActiveX Data Objects). Той е компилируем език, но резултата, както в Java е междинен език MSIL (Microsoft Intermediate Language), който при изпълнение се компилира до машинен код от JIT (Just In Time) компилатор. Тази технология дава видимо

забавяне при начално зареждане на приложението и високо бързодействие по време на неговата работа.

- СУБД

- MySQL

- Oracle

- Microsoft SQL Server

От изброените системи за управление на релационни бази от данни Oracle има най-пълни възможности, но за сметка на много усложнена администрация и недобра интеграция с операционната система.

Microsoft SQL Server в последните си версии много се доближава до Oracle по възможности, но негови основни предимства остават удобната администрация и перфектната интеграция с операционната система.

Разглеждаме и една свободна система – MySQL, поради факта, че при администрирането на съдържанието на един посещаван уеб сайт отношението между заявки извличащи информация от СУБД към заявки променящи съдържанието на СУБД зависи предимно от посещаемостта. Ако посещаемостта е висока, то спокойно можем да пренебрегнем заявките променящи базата, защото тогава отношението се движи между 1:1000 и 1:1000000. Липсата на транзакции в MySQL много повишава производителността му при подобно отношение между типовете заявки. При правилно изградени индекси тя дори надвишава Oracle 5 пъти. От друга страна изискванията на системата за управление на съдържанието

към СУБД не са високи по отношение на функционалност. Единствено факторите стабилност или по-добра интеграция с операционната система при Microsoft SQL Server могат да наклонят везните в негова полза.

Като цяло можем да разглеждаме следните комбинации:

1. Apache/PHP/MySQL
2. Apache/PHP/Oracle
3. Microsoft IIS/C#/Microsoft SQL
4. Microsoft IIS/C#/Oracle

Всички те са възможни работещи решения, като избора на платформа вече зависи от спецификата на приложението. Например в организация, която разполага със закупен Oracle (не трябва да пренебрегваме фактора цена, а при Oracle цената е доста висока) бихме се спрели на варинат 2 или 4. В зависимост от това дали разполагаме с Windows или UNIX базирани сървъри избираме между варианти 1-2 и 3-4. За проверка на концепцията и максимална простота на приложеното решение избираме вариант 1, поради това, че в него се използва само свободен софтуер, който е публично достъпен. Разбира се при едно сериозно комерсиално приложение изборът ни ще бъде повлиян от възможностите за поддръжка, стабилността и производителността.

• **Представяне в релационна СУБД**

Представянето на данните на нашия модел в релационна система за управление на бази от данни е естествено, поради тяхната

релационна структура. Връзките между отделните компоненти на системата по един естествен начин ни водят до правилните за модела релации и представяне.

- Език

Представянето на езика в системата е в една таблица. Указва се идентификатора на езика и кой от въведените езици е по подразбиране за системата. Езика по подразбиране се използва за да се зададе основен (първи) език. Идентификатора на език, който служи за връзка с останалите таблици е уникален индекс. Флага за език по подразбиране трябва да е вдигнат точно за един запис. Това се осигурява или от СУБД или от програмната реализация на кода за достъп до СУБД в зависимост от възможностите на конкретната СУБД. В случая на MySQL това ограничение е на ниво програмна реализация.

- Блок (страница)

Блоковете в системата съдържат различни типове информация – езиково зависима и езиково независима. Езиково зависимата се дели на два вида – такава, която е списък от елементи за даден език и такава, която е множество от двойки идентификатор на език и данна. Представяме ги в три таблици – `pages` за езиково независимите данни (идентификатор на блока – `action`, флаг за блок по подразбиране (или начална страница), флаг за отличаване на чист блок от страница, флаг за интегриране на система за търсене и идентификатор на шаблон), `pages_data` за списъците от езиково зависими данни (идентификатор на страница, идентификатор на език, идентификатор на елемент от шаблона,

поле за подредба и самите данни) и `pages_titl` за заглавията на страниците.

- Шаблон

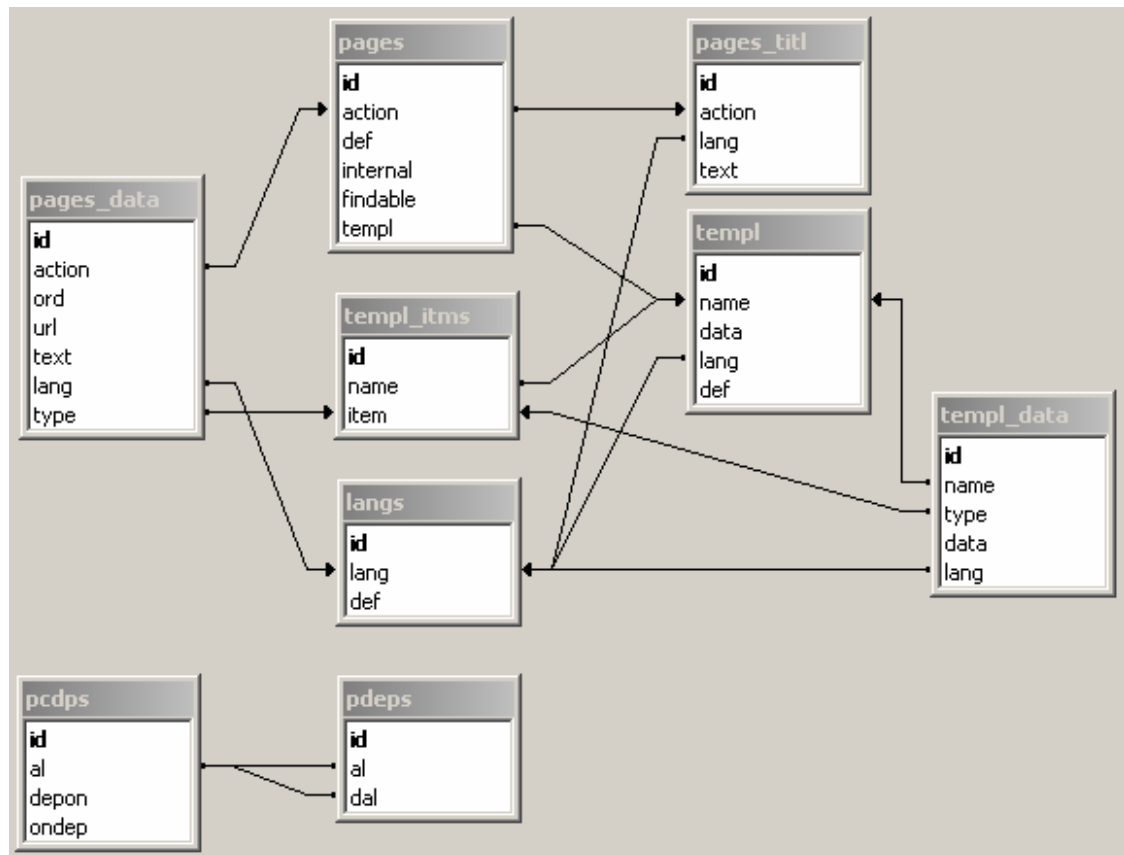
При шаблоните само идентификаторите на елементи са езиково независими. Тях представяме в отделна таблица – `templ_itms`. Останалите елементи на шаблоните представяме в две таблици за основните шаблони и техните елементи – `templ` и `templ_data`.

- Релация включване на блок от друг блок

При оптимизацията на системата за бързодействие ще се наложи във всеки един момент от време да имаме актуална информация за релацията включване на блок от друг блок. Смисъла на релацията е на зависимост – блок1 зависи от (включва, съдържа) блок2. На практика е много полезно да знаем кои блокове съдържат даден блок. В таблиците `pdeps` и `pcdps` съхраняваме тази релация и нейното транзитивно затваряне.

На фиг.1 са представени релациите между таблиците в системата:

Фиг. 1. Схема на релациите



• **Метаезик**

Метаезика в системата е реализиран чрез система за заместване на параметризирани регулярни изрази. В зависимост от тага от метаезика, който се обработва, се прилага или директно заместване на база зададените параметри или се изпълнява програмен код, който изпълнява заявка към базата от данни или прочита и взема параметри от файл на диска (в случая с графичните изображения, размерите им се четат директно от съответния файл) или рекурсивно се обръща към процедурата за обработка на блок в случая на включване на блок от друг блок.

Динамичността на тази реализация и пряката зависимост на броя на операциите от сложността на използваните тагове правят подобна система много гъвкава, с високо бързодействие и много малко време за начално установяване (което е проблем в системите с използване на предварително разпознаване на командите/таговете (parsing) и последваща обработка).

- **Файлове и графични изображения**

Файловете и графичните изображения в системата се реализират като файлове на диска в определена структура от директории. Файловете се записват в поддиректория *f* на уеб видимата част от системата, докато графичните изображения в поддиректория *p*. Така хипервръзката към файла `download.zip` ще бъде `/f/download.zip`. При графичните изображения нещата са малко по-различни – за всеки език се създава поддиректория на *p* с име идентификатора на езика. Така при заявка за графично изображение `myfile.gif` се прави проверка за съществуването на `p/<текущ.език>/myfile.gif` и ако такъв файл не съществува, се използва `p/myfile.gif`.

При качване на езиково независими графични изображения се използва директория *p*, а при езиково зависими в *p* се записва изображението по подразбиране, докато изображенията за различните езици в `p/<идентификатор.на.език>/<име.на.файл>`. Приема се, че езиково зависимите изображения са версии (преводи) на едно и също графично изображение.

- **Оптимизации за бързодействие**

Имайки предвид направените изследвания на отношението между броя на различните типове заявки направено при избора на език и СУБД, можем да оптимизираме на още една стъпка бързодействието на системата. Колкото и бърза да е обработката, колкото и бърза да е използваната система за управление на бази от данни не можем да сравняваме бързодействието на един статичен уеб сайт с един динамично генериран.

По своята природа много малка част от съдържанието на един уеб сайт наистина е динамично. Останалата част се променя само при намесата на администратора на съдържанието. При разработената до този момент система независимо от този факт съдържанието на всяка страница винаги се генерира динамично. Това излишно забавяне може много лесно да бъде избегнато с въвеждането на система за прегенериране в статичен HTML на всички нединамични по природа страници от сайта.

Тази система обаче трябва да позволява както предварително преглеждане на базата преди прегенериране, така и прегенериране на всички блокове зависими от променения блок. Точно тук влиза в употреба транзитивното затваряне на релацията включване на блок от друг блок.

Удобно се оказва да се разработи система за прехващане на заявки към Web Server-а, която да изпълнява програмния код за показване на страница, когато поисканата страница не е прегенирирана на диска, и директно я дава на клиента в обратния

случай. Такава система може много лесно да бъде реализирана както под Microsoft IIS с помощта на ISAPI филтър, така и под Apache, като се използва mod_rewrite. Направените изследвания показват, че включването на подобни системи и в двата случая много слабо намаляват производителността – при тестовата конфигурация за 10000 последователни заявки скоростта при динамично генерирано съдържание е 5 заявки в секунда, 203 при обикновено статично съдържание и 199 при прегенерирано статично съдържание и работеща система за проверка на наличието му.

- **Административен интерфейс**

Административният интерфейс на системата е проектиран с една основна цел – максимално удобство при работа както на администратора на шаблоните за форматиране, така и на администратора на съдържанието.

Функции на административния интерфейс за езиците в системата:

- преглеждане на дефинираните езици
- дефиниране на нов език
- промяна на езика по подразбиране (Фиг. 2)
- изтриване на вече дефиниран език

Функции на интерфейса за администрация на шаблоните в системата:

- преглед на дефинираните шаблони (Фиг. 3)

- добавяне на нов шаблон
- изтриване на съществуващ шаблон
- копиране на един шаблон върху друг
- дефиниране на идентификатори на елементите (Фиг. 4)
- редактиране на елементите на шаблона (Фиг. 5)
- редактиране на основния шаблон (Фиг. 6)

Функции на интерфейса за администрация на страниците (блоковете) в системата:

- преглеждане на дефинираните блокове (Фиг. 7)
- създаване на нов блок
- изтриване на съществуващ блок
- редактиране на блок (Фиг. 8)
- копиране на блок/език към друг блок/език
- предварително преглеждане на резултата от редактиране
- редактиране на заглавието и параметрите на блока

Функции на административния интерфейс за управление на графични изображения:

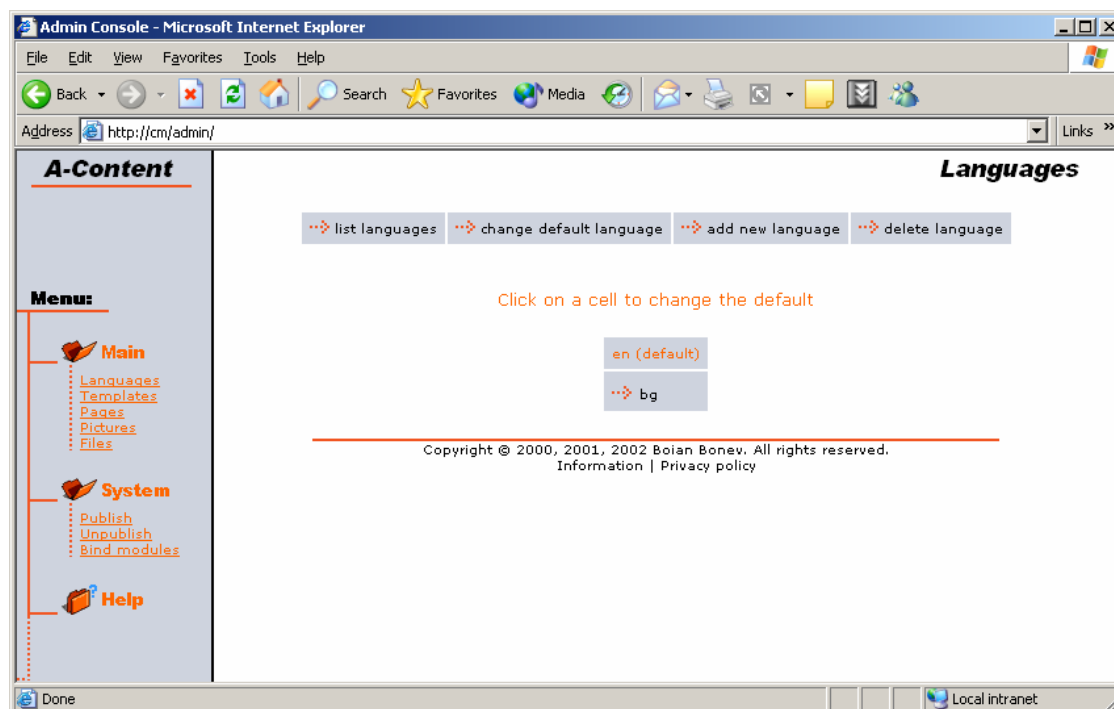
- дефиниране на ново графично изображение (Фиг. 9)
- преглеждане на дефинираните изображения
- изтриване на дефинирано изображение
- промяна на файловете съставлящи дадено графично изображение

Функции на интерфейса за администрация на файлове в системата:

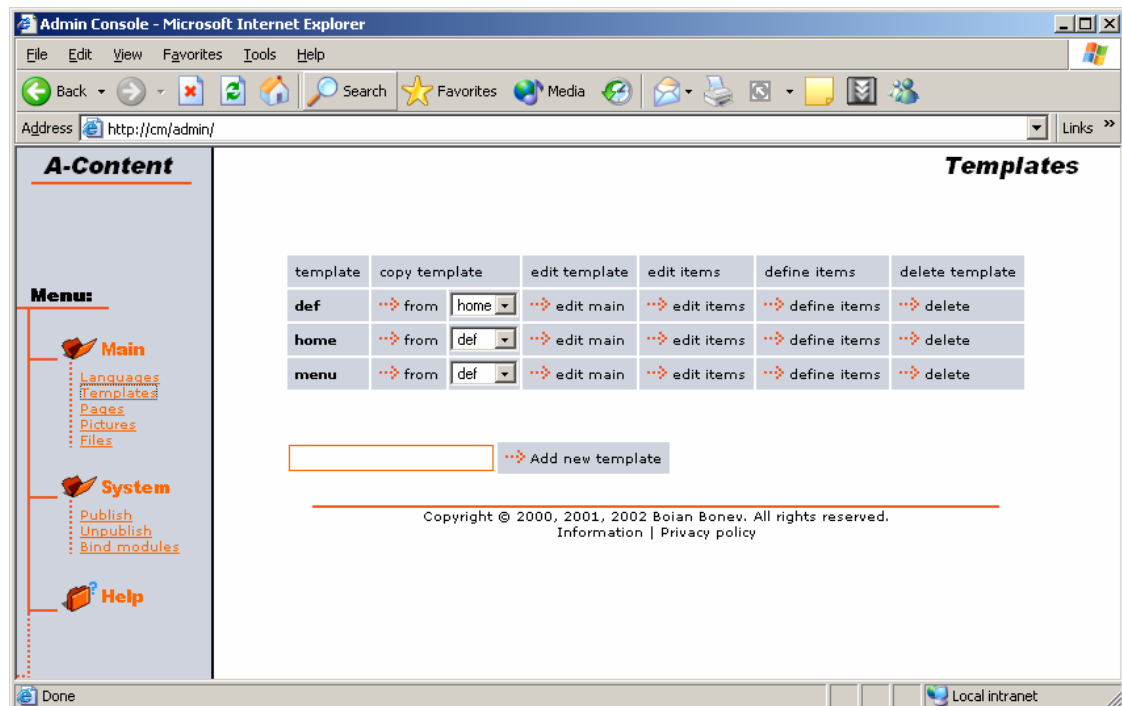
- качване на нов файл (Фиг. 10)
- преглеждане на качените файлове
- изтриване на вече качен файл

Илюстрации на работата на административния интерфейс:

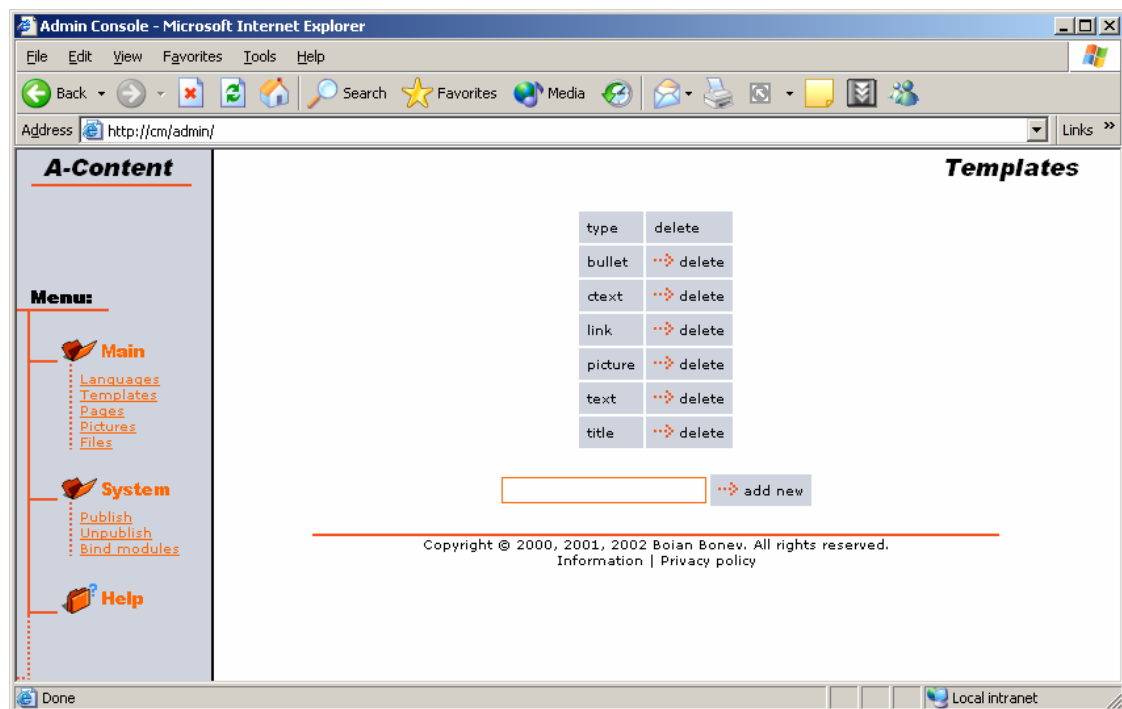
Фиг. 2. – Избор на език по подразбиране



Фиг. 3. – Преглед на дефинираните шаблони



Фиг. 4. – Дефиниране на идентификатори на елементите на шаблон



The screenshot shows a Microsoft Internet Explorer browser window. The title bar reads "Admin Console - Microsoft Internet Explorer". The address bar shows "http://cm/admin/". The main content area displays a web page titled "A-Content" with a "Templates" section. The page contains HTML code for a web template, including meta tags, styles, and body content. The browser's status bar at the bottom shows "Done" and "Local intranet".

Admin Console - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Back Forward Stop Home Search Favorites Media Refresh Print Link

Address http://cm/admin/ Links

A-Content

Templates

Lang: en

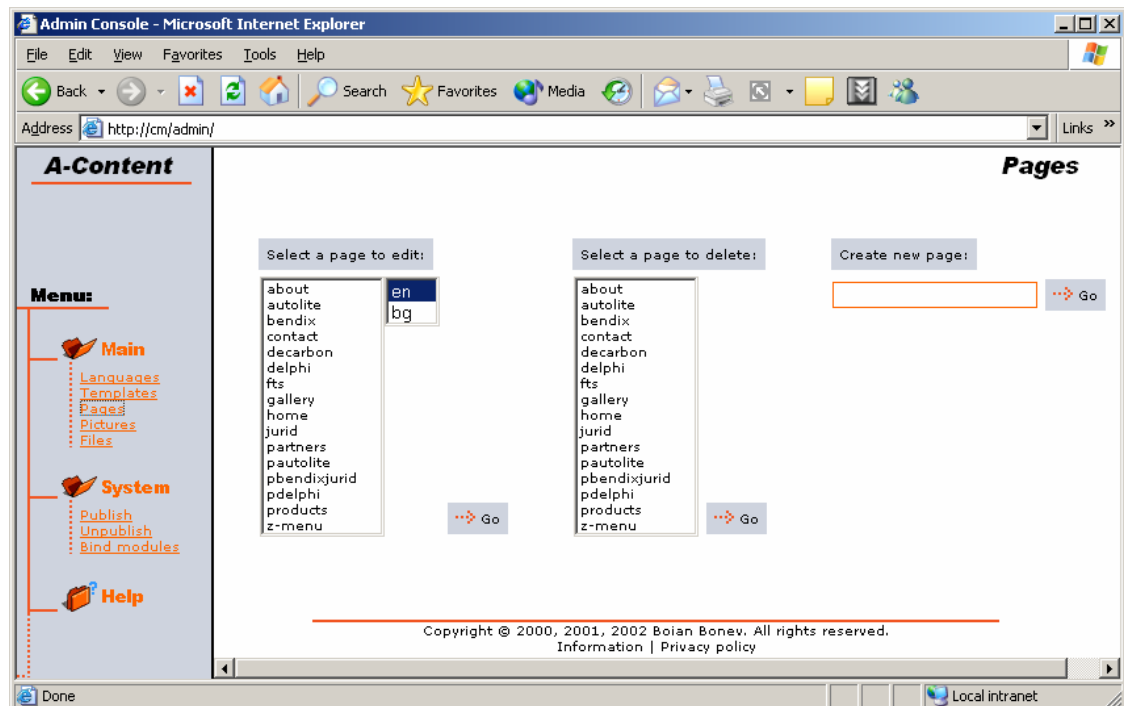
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
<html><head><title>www.kanex.bg</title></head>
<style type="text/css">
<!--
.link{ font-family: Arial; font-size: 10pt; color: #002072; text-decoration: none; font-weight: normal; }
.link: hover{ font-family: Arial; font-size: 10pt; color: #0099cc; text-decoration: none; font-weight: normal; }
.wln { color: ffffff }
a { text-decoration: none }
a: hover { text-decoration: underline }
--></style>
--></style>
<?script?>
<body bgcolor=ffffff link=000000 alink=000000 vlink=000000 marginheight=0 marginwidth=0 topmargin=0 leftmargin=0>
<table border=0 cellpadding=0 cellspacing=0 width=100% height=71>
<tr>
<td valign=middle bgcolor=dd0000><?picture=1.gif,/<?lang?>/home?></td>
<td bgcolor=dd0000></td>
<td valign=bottom bgcolor=dd0000 align=right width=100%><?picture=3.jpg></td>
</tr>
</table>
<table width=100% border=0 cellpadding=0 cellspacing=0>
<tr>
<td align=right background="<?imgsrc=24.gif?>" nowrap><!--?picture=lang,21.gif,22.gif,/bg/<?picture=23.gif?>" height=1 width=100%></td>
<td align=left background="<?imgsrc=24.gif?>" nowrap><br></td>
</tr>
</table>
</body>
</html>
```

Menu:

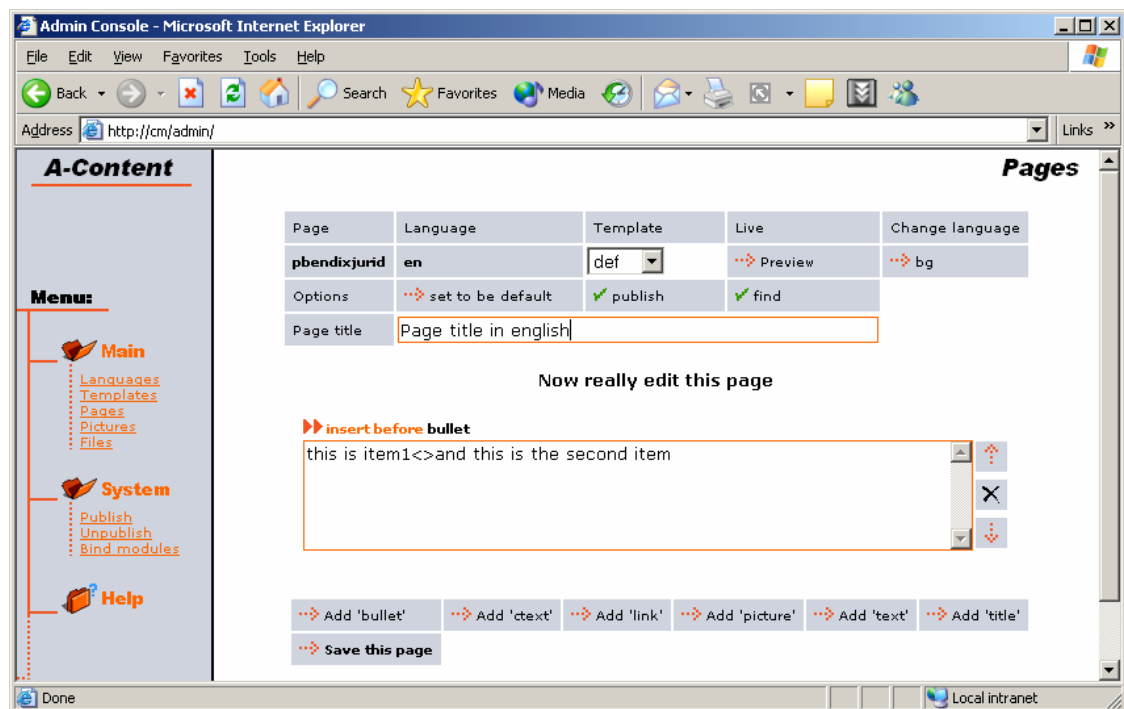
- Main
 - Languages
 - Templates
 - Pages
 - Pictures
 - Files
- System
 - Publish
 - Unpublish
 - Bind modules
- Help

Done Local intranet

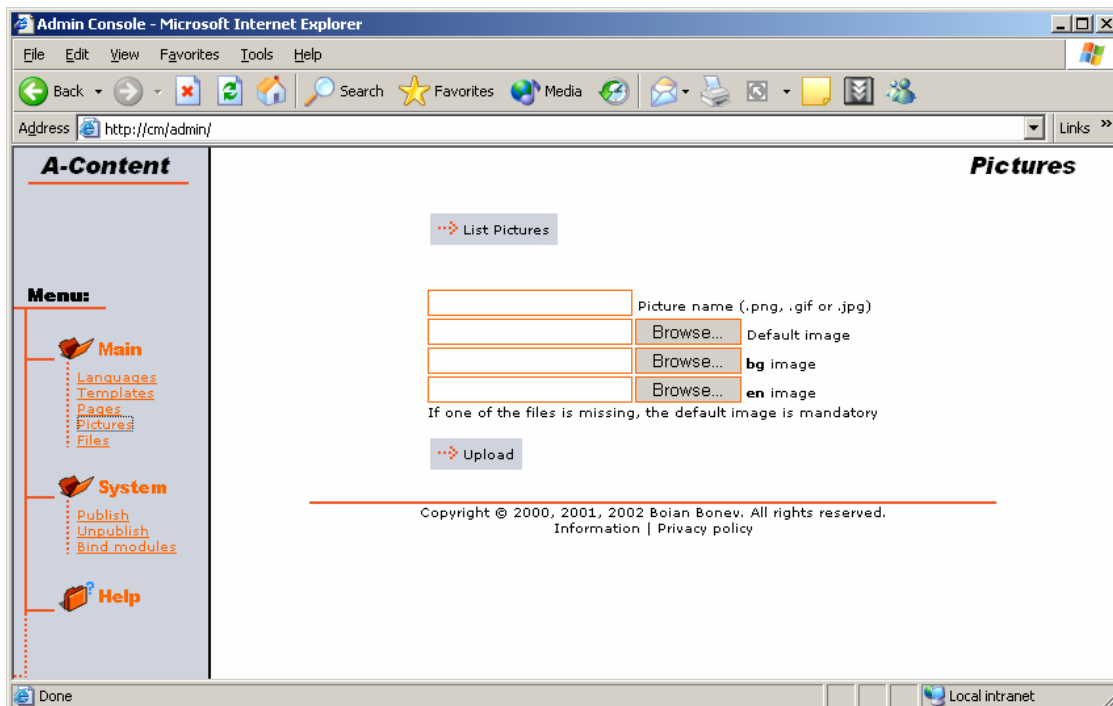
Фиг. 7. – Преглед на дефинираните блокове



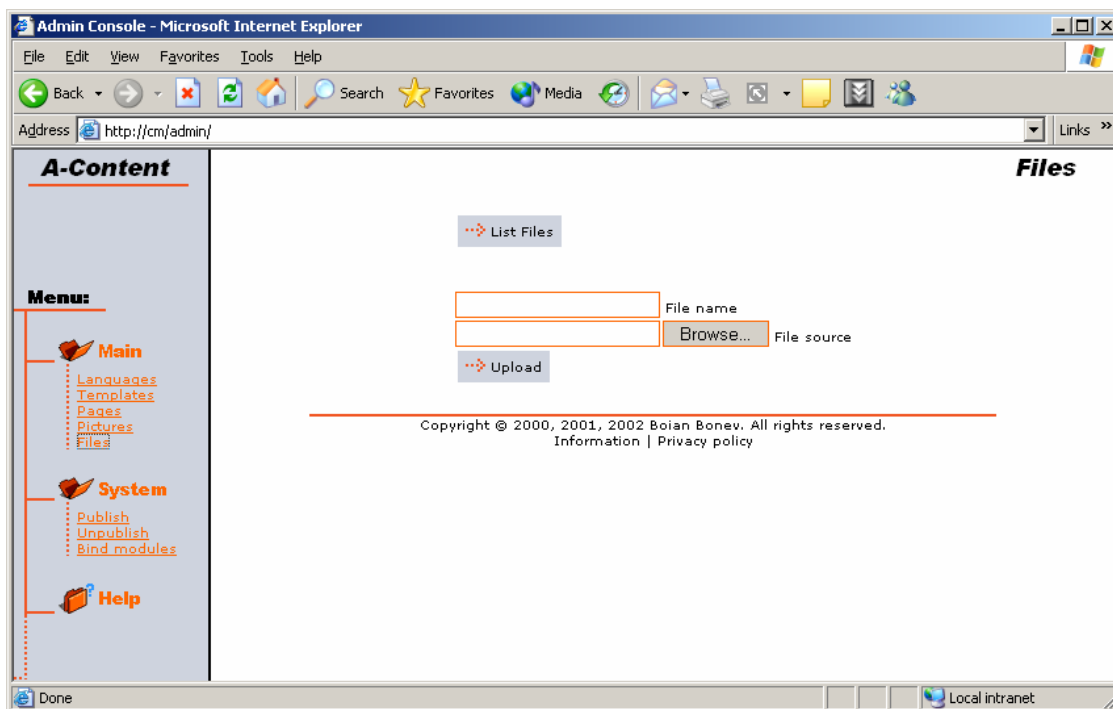
Фиг. 8. – Редакция на блок



Фиг. 9. – Администрация на графични изображения



Фиг. 10. – Администрация на файлове



Глава 3 - Реализация в системата на конкретен интернет сайт

- **Избор на пример от практиката**

За демонстрация на работата на системата е избран реален пример от практиката – презентационната част от сайта на първия български интернет магазин за авточасти – www.kanex.bg. Структурата на сайта е достатъчно проста, за да бъде демонстрацията ясна, но и със сравнително пълен набор от възможности, така че да може да се покаже цялата функционалност на системата.

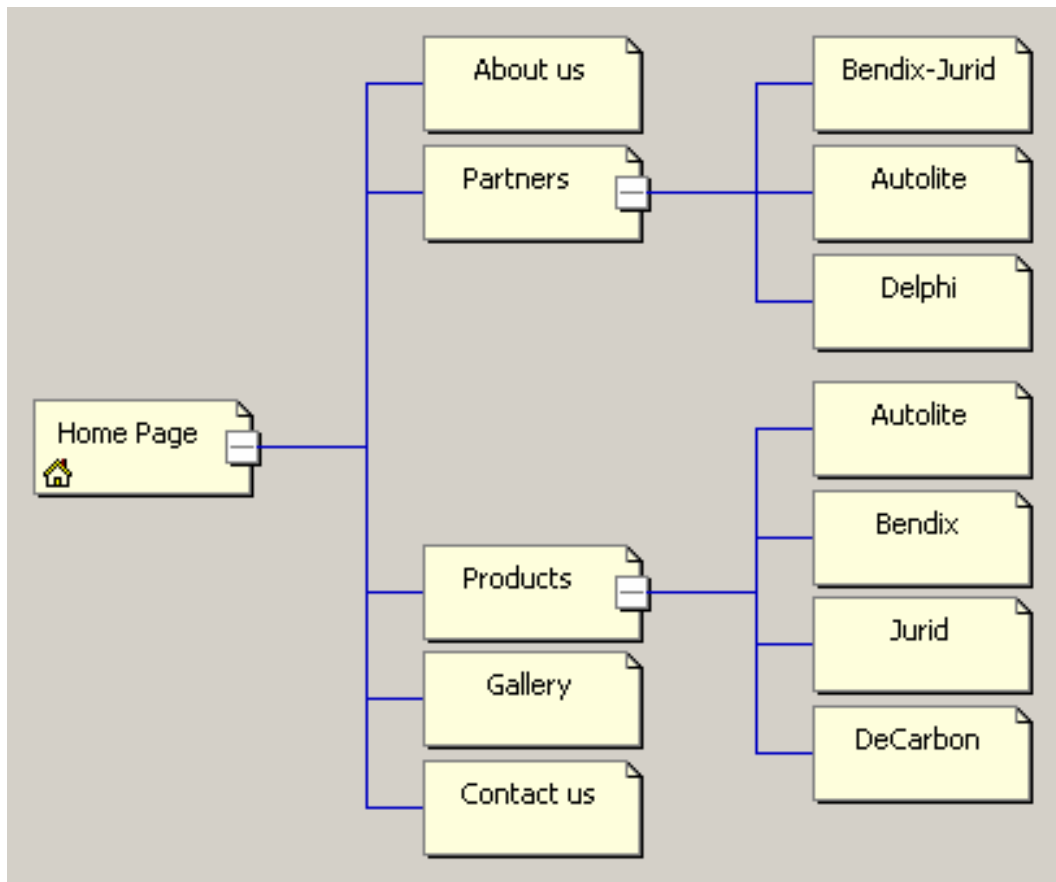
- **Структура на сайта**

Избраният сайт се състои от заглавна страница, страници от първо ниво и подстраници на продукти и партньори (второ ниво).

От маркетингова гледна точка проектантът на сайта е избрал в навигацията да не се показват подстраниците на партньори, а само тези под продукти.

Типовете страници са два – начална и вътрешна. За вътрешните страници изгледа не се променя независимо дали са на първо или второ ниво. Навигацията е текстова и е обща за всички вътрешни страници.

Фиг. 11. – Структура на сайта



Началната страница съдържа връзки към страниците от първо ниво. Тази информация се счита за сравнително постоянна и дизайнера на сайта е взел решение с цел по-добър изглед цялата първа страница да бъде реализирана с графични изображения.

Всички останали страници имат еднакъв изглед. В основния шаблон има два променящи се елемента – заглавието на страницата и хипервръзката за смяна на езика. Прието е, че в обозримо бъдеще няма да се добавят повече езици и тази хипервръзка е реализирана като графично изображение, което се променя при преминаване на мишката над него. Използвана е възможността тези графични изображения да бъдат дефинирани

като езиково зависими и алтернативно да променят текста си при смяна на езика. С помощта на метаязика се генерира хипервръзка, която е към алтернативния език и същата страница. Налице е влагане на тагове

В основния шаблон за английски:

```
<?picture=lang,21.gif,22.gif,/bg/<?page?>?>
```

В основния шаблон за български:

```
<?picture=lang,21.gif,22.gif,/en/<?page?>?>
```

Отново в основния шаблон се вгражда блок с навигационното меню, което е много подобно за всички страници. Разликата е, че текущата страница се изобразява с удебелен шрифт в менюто и не е хипервръзка.

Съдържанието на всички вътрешни страници в сайта е комбинация от два размера текст, ляво подравнени графични изображения, центрирани графични изображения и картинна галерия.

За да се използва общ шаблон за всички тези страници съответните елементи се дефинират като елементи на шаблона както следва (<идентификатор.на.елемент.от.шаблона>: <дефиниция>):

Центриран текст (или графично изображение, ако е зададено с таг на метаязика):

```
ctext: <p align=center><?text?></p>
```

Графично изображение (в данните се задава само името на файла, форматирането е центрирано):

picture: `<p align=center><?picture=?text??></p>`

Параграф текст с първи ред три символа навътре:

text: `<p> <?text??></p>`

Подзаглавие:

Title: `<p> <?text??>

</p>`

- **Реализация на вътрешни страници**

Страниците About us (За нас) и Contact us (За контакти) се състоят само от текстови параграфи. В "за контакти" е използван елемента подзаглавие, за да се даде акцент на представената информация.

Страниците Partners (Партньори) и Products (Продукти) се състоят само от графични изображения на емблемите на фирмите или продуктите, които са хипервръзки към съответната страница от второ ниво. Страницата партньори се състои от следните три елемента (`<идентификатор.на.елемент.от.шаблона>: <данни>`):

picture: `bendixjuridlogo.gif,/<?lang?>/pbendixjurid`

picture: `autolitelogo.gif,/<?lang?>/pautolite`

picture: `delphi2.gif,/<?lang?>/pdelphi`

Страниците от второ ниво се състоят от набор от подзаглавия, параграфи и графични изображения. Примерна комбинация (извадка от страницата bendix на български) е представена на Фиг.

12. Ясно се наблюдава как може да се включва HTML код, но в 99% от случаите се използва 'чист' текст.

Фиг. 12. – Извадка от администрирането на съдържанието на страницата bendix на български

▶▶ insert before text
От продуктовата гама на BENDIX "Канекс" ЕООД предлага:
дискови и барабанни накладки, спирачни дискове и барабани,
спирачна течност, жила, спирачни цилиндри, маркучи, помпи,
сервоусилватели и др.

▶▶ insert before text
Спирачна система:

▶▶ insert before picture
brakesystem.jpg

▶▶ insert before title
Накладки

▶▶ insert before text
Накладките BENDIX са разработени в съответствие с
характеристиките на превозното средство и конфигурацията на
спирачната система, дефинирани от производителя на автомобила.
Асортиментът от фрикционни продукти на Bendix надхвърля 700
позиции и покрива повече от 95 % от европейския пазар. Съставът

- **Реализация на блок меню**

Използва се елементарен основен шаблон:

```
<font face=verdana size=2 color=ffffff><br><?content?><br>
```

За дефиниране на трите вида елементи се използват следните елементи на шаблона – разделител, страница от първо ниво и страница от второ ниво:

```
separator: <br>
```

```
first: &nbsp;<<?condgpage=?url?>,b,a?>  
href="/<?lang?>/<?url?>"  
class=wln><?text?></<?condgpage=?url?>,b,a?>><br>
```

```
second: &nbsp;&nbsp;&nbsp;<<?condgpage=?url?>,b,a?>  
href="/<?lang?>/<?url?>"  
class=wln><?text?></<?condgpage=?url?>,b,a?>><br>
```

Посредством условния таг от метаязика condgpage се прави проверка за използваната страница, независимо от това, че менюто е блок с име z-menu. Ако текущата страница съвпада със зададеното url, се прави промяна на форматирането от хипервръзка към удебелен текст.

- **Реализация на галерия от графични изображения**

Има два подхода в решаването на задачата за представяне на галерия от графични изображения:

- използване на факта, че тага `<?ipopup=imagename,imagename?>` може да се използва за шаблон на елемент (`<?ipopup=<?text?>?>`) и в съдържанието да се въвеждат двете графични изображения разделени със запетая `thumb.gif<>bigpicture.jpg`
- използване на вградената в системата функционалност таг за разделяне. Тогава шаблонът за елемент би изглеждал така:
`<?ipopup=<?tag_1?>,<?tag_2?>?>`, а данните:
`thumb.gif<>bigpicture.jpg`

Разбира се системата позволява директно използване на таг от метаезика или даже HTML код. Не разглеждаме тези варианти поради факта, че те биха се наложили само ако системата не поддържа дадена функционалност.

Заклучение

- **Постигнати резултати**

Разработен е модел за представяне съдържанието на уеб сайт и разделяне на данни от форматиране. Модела е добър компромис между дълбочина на структуриране на данните, простота при представяне на често използвани в практиката концепции и минимални технически изисквания към ролята администратор на съдържанието.

Реализирана е софтуерна система, която имплементирайки този модел проверява неговата практическа приложимост. Изграденият в системата конкретен уеб сайт демонстрира функционалността на системата и ползата от оптимизациите за бързодействие. По време на изграждането му е разгледана методология за приложение на системата както към съществуващи, така и към новоизграждащи се уеб сайтове.

- **Възможности за развитие**

Перспективите пред подобна система са огромни. Винаги ще има често изпълнявани задачи, които не се покриват или не се автоматизират добре от системата. Изреждаме само част от възможните качествени подобрения в реализираната система:

- възможност за интеграция с външни за системата софтуерни модули
- развиване на ролите в системата в йерархична структура и ограничаване достъпа до ресурсите

- добавяне на система за контрол на версиите – запазване на всички работни версии на уеб сайта с възможност за връщане назад или наличие на по-нови версии от публикуваната
- система за разрешаване на конфликти при редакция на един и същи ресурс от повече от един редактор в даден момент (в текущата имплементация втория записал печели, както при работа с поделен файл)
- интегриране на система за уведомяване за събитие и система за одобряване на промени по съдържанието (администратора на съдържанието прави промяна, системата уведомява началника на отдела и той от своя страна взима решение дали направената промяна да се публикува или не)
- дървовидно представяне на блоковете в системата, взимайки предвид релацията включване на блок от блок (в текущата имплементация всички блокове се показват линейно, по азбучен ред). Добавяне на бързи връзки към включваните и включващите блокове от първо ниво при редакция на блок.
- замяна на полетата за редакция на текст в административния модул с ActiveX, Java или .NET клиентски аплети, които поддържат визуално базово форматиране и правят работата с метаязика визуална

Приложение

- **Исходен текст на подсистемата за обработка на метаезика**

```
<? // $Id: cm.php,v 1.29 2001/09/12 01:18:46 bbonev Exp $

include('lib/cmbase.php');
include('lib/cmmod.php');

function cm_rawcontent($l,$t,$a) {
    $SQL="select pages_data.text,pages_data.type,templ_data
.data ".
        "from pages_data left join templ_data on pages_data
.type=templ_data.type and templ_data.name='$t' and templ_da
ta.lang='$l' ".
        "where pages_data.lang='$l' and pages_data.action='
$a' ".
        "order by pages_data.ord";
    if (!($res=mysql_query($SQL)))
        return false;
    $rv=array();
    while ($r=mysql_fetch_array($res,MYSQL_ASSOC)) {
        $rv[]=$r;
    }
    return $rv;
}

function getmicrotime(){
    list($usec, $sec) = explode(" ",microtime());
    return ((float)$usec + (float)$sec);
}
```

```

function cm_content($l,$templ,$content,$a,$todel=false,$recurse=array()) {
    global $cm_path;

    if (count($recurse)==0) {
        $recurse=array("$a?$l?"=>true);
    }
    $toppage=count($recurse)==1;
    // place content and put all together
    $rv=str_replace("<?content?>",$content,$templ);
    // process module calls
    if ($toppage) {
        preg_match_all("|<\?mod=([^\?]+)\?>|U",$rv,$mods,PREG_PATTERN_ORDER);
        for ($i=0;$i<count($mods[0]);$i++) {
            $rv=str_replace($mods[0][$i],cm_mod_call($a,$l,$mods[1][$i]),$rv);
        }
    }
    // process nested tags - page name, lang id
    $rv=str_replace("<?page?>",$a,$rv);
    $rv=str_replace("<?lang?>",$l,$rv);
    $rv=str_replace("<?title?>",cm_page_title($a,$l),$rv);
    // process local page cond tag
    $rv=preg_replace("|<\?condpage=$a,([^\? ,]+),([^\? ,]+)\?>|U", "\\1", $rv);
    $rv=preg_replace("|<\?condpage=([^\? ,]+),([^\? ,]+),([^\? ,]+)\?>|U", "\\3", $rv);
    // process page include tags
    if ($todel&&$todel=='deps') {
        preg_match_all("|<\?page=([^\? ,]+)(,([^\? ,]+)(,([^\? ,]+)))?)?\?>|. ">|U",$rv,$pagesinc,PREG_PATTERN_ORDER);
    }
}

```

```

        $deps=array();
        for ($i=0;$i<count($pagesinc[0]);$i++) {
            $toadd=$pagesinc[1][$i].'?';
            $toadd.=(isset($pagesinc[3][$i]))?$pagesinc[3][
$i]:$1;

            if (!in_array($toadd,$deps))
                $deps[]=$toadd;
        }
        return $deps;
    }
    $rv=preg_replace(
        array(
            "|<\?page=([^\? ,]+), ([^\? ,]+), ([^\? ,]+)\?>|Ue",
            "|<\?page=([^\? ,]+), ([^\? ,]+)\?\".\">|Ue",
            "|<\?page=([^\? ,]+)\?>|Ue",
        ),
        array(
            "cm_page('\2', '\1', '\3', \$recurse);",
            "cm_page('\2', '\1', '', \$recurse);",
            "cm_page(\$1, '\1', '', \$recurse);",
        ),
        $rv);

    // when in recursion do not process anything else but l
    et upper level to do the job
    if (!$stoppage) {
        return $rv;
    }

    // process global page cond tag
    $rv=preg_replace("|<\?condgpage=\$a, ([^\? ,]+), ([^\? ,]+)\?>
|U", "\1", $rv);

```

```

$rv=preg_replace("|<\?condgpage=([^\?],+),([^\?],+),([^\?],+)\?>|U","\\3",$rv);

// process link tags
$rv=preg_replace("|<\?link=([a-zA-Z0-9_])\?>|U","<a href=\"\\1\">",$rv);
$rv=str_replace("<?/link?>","</a>",$rv);
// process file
if ($todel=='file')
    preg_match_all("|<\?file=([^\?]+)\?>|U",$rv,$filen,PREG_PATTERN_ORDER);
$rv=preg_replace("|<\?file=([^\?]+)\?>|U","<a href=/f/\\1>\\1</a>",$rv);
// process popup pictures
$rv=preg_replace("|<\?ipopup=([^\?],+),([^\?],+)\?>|U","<a href=# onclick=\"<?ipopup_script=\\2?>\"><?picture=\\1?\".\"></a>",$rv);
$rv=preg_replace("|<\?ipopup_script=([^\?]+)\?>|U","window.open('<?imgsrc=\\1?>','ipopup','width='+eval('<?imgwidth=\\1?>+20')+','height='+eval('<?imgheight=\\1?>+20')+','status=no,resizable=no').focus();return false;",$rv);
// process single pictures
$rv=preg_replace("|<\?picture=([^\?],+)\?>|U","<img <?imgparams=\\1?>>",$rv);
// process pictures with conditional link
$rv=preg_replace("|<\?picture=!$a,([^\?],+),([^\?],+),([^\?],+)\?>|U","<img <?imgparams=\\2?>>",$rv);
$rv=preg_replace("|<\?picture=!([^\?],+),([^\?],+),([^\?],+)\?>|U","<?picture=\\2,\\4?>",$rv);
// process pictures with link
$rv=preg_replace("|<\?picture=([^\?],+),([^\?],+)\?>|U","<a href=\"\\2\"><img <?imgparams=\\1?>></a>",$rv);

```

```

    // process pictures with link, mouse over and condition
al link

    $rv=preg_replace("|<\?picture=!$a, ([^?, ]+), ([^?, ]*), ([^
?, ]+), ([^?, ]+), ([^?, ]+)\?>|U", "<img <?imgparams=\\4?>", $rv
);

    $rv=preg_replace("|<\?picture=!([^?, ]+), ([^?, ]+), ([^?, ]
*), ([^?, ]+), ([^?, ]+), ([^?, ]+)\?>|U", "<?picture=\\2, \\3, \\4,
\\6?>", $rv);

    // process special tags

    preg_match_all("|<\?picture=([^?, ]+), ([^?, ]+), ([^?, ]+),
([^?, ]+)\?>|U", $rv, $pics, PREG_PATTERN_ORDER);
    $cnt=count($pics[0]);
    for ($i=0; $i<$cnt; $i++) {
        if ($pics[1][$i][0]=='!')
            $pics[1][$i]=substr($pics[1][$i], 1);
        $jsimg.=$pics[1][$i]. "h=new Image();". $pics[1][$i].
"h.src=\"<?imgsrc=\". $pics[3][$i]. "?>\";";
        $jsimg.=$pics[1][$i]. "l=new Image();". $pics[1][$i].
"l.src=\"<?imgsrc=\". $pics[2][$i]. "?>\";";
    }
    if ($cnt) {
        $jscript='<script type="text/javascript" language="
JavaScript"><!--'. "\n".
        'browserName=navigator.appName; browserVer=parse
Int(navigator.appVersion);'.
        'if( (browserName=="Netscape"&&browserVer>=3) || (
browserName=='.
        '"Microsoft Internet Explorer"&&browserVer>=4))
version="n3";'.
        'else version="n2"; if(version=="n3"){'. $jsimg. '
}''.
        'function h(i){if(version=="n3"){thei=hlfo(i);i

```

```

f(thei!=null)eval(\'thei.src=\'+i+\'h.src\');}}'.
    'function l(i){if(version=="n3"){thei=hlfo(i);i
f(thei!=null)eval(\'thei.src=\'+i+\'l.src\');}}'. "\n".
    'function hlfo(n,d){var p,i,x;if(!d)d=document;
if((p=n.indexOf("?"))>0&&parent.frames.length){'.
    'd=parent.frames[n.substring(p+1)].document;n=n
.substring(0,p);}if(!(x=d[n])&&d.all)x=d.all[n];'.
    'for(i=0;!x&&i<d.forms.length;i++)x=d.forms[i][
n];for(i=0;!x&&d.layers&&i<d.layers.length;i++)'.
    'x=hlfo(n,d.layers[i].document);return x;}//--
></script>'. "\n";
    } else {
        $jscript='';
    }
    // process javascript, if present
    $rv=str_replace("<?jscript?>",$jscript,$rv);
    // process pictures with link and mouse over
    $rv=preg_replace(
        array(
            "|<?picture=!(\[^\?,\]+),(\[^\?,\]*),(\[^\?,\]+),(\[^\?,\]+)
+)\?>|U",
            "|<?picture=(\[^\?,\]+),(\[^\?,\]*),(\[^\?,\]+),(\[^\?,\]+)
+)\?>|U",
        ),
        array(
            "<a href=\"\\4\" onmouseover='h(\"\\1\")' onmou
seout='l(\"\\1\")'><img <?imgparams=\\2?>></a>",
            "<a href=\"\\4\" onmouseover='h(\"\\1\")' onmou
seout='l(\"\\1\")'><img name=\"\\1\" <?imgparams=\\2?>></a>
",
        ),
        $rv);

```

```

// process image parameters -
border=0 width=ww height=hh

preg_match_all("|<\?imgparams=([^\?]+\?)\?>|U",$rv,$pics1,
PREG_PATTERN_ORDER);
preg_match_all("|<\?imgsrc=([^\?]+\?)\?>|U",$rv,$pics2,PREG_PATTERN_ORDER);
preg_match_all("|<\?imgwidth=([^\?]+\?)\?>|U",$rv,$pics3,PREG_PATTERN_ORDER);
preg_match_all("|<\?imgheight=([^\?]+\?)\?>|U",$rv,$pics4,PREG_PATTERN_ORDER);
$pics=array_merge($pics1[1],$pics2[1],$pics3[1],$pics4[1]);
$names=$pics;
$cache=array();
while (list($dummy,$p)=each($names)) {
    if (!$cache[$p]) {
        $cache[$p]=true;
        $fname=$cm_path."p/$l/$p";
        $srcp="/p/$l/$p";
        if (!file_exists($fname)) {
            $fname=$cm_path."p/$p";
            $srcp="/p/$p";
        }
        if (!file_exists($fname)) {
            $pars='border=0';
            $height='';
            $width='';
        } else {
            $tsizes=GetImageSize($fname);
            $pars="border=0 width=${tsizes[0]} height=${tsizes[1]}";

```

```

        $height=$tsizes[1];
        $width=$tsizes[0];
    }
    $pars="src=\"${srcp}\" $pars";
    $rv=preg_replace(
        array(
            "|<\?imgparams=$p\?>|U",
            "|<\?imgsrc=$p\?>|U",
            "|<\?imgsrc=$p\?>|U",
            "|<\?imgwidth=$p\?>|U",
            "|<\?imgheight=$p\?>|U",
        ),
        array(
            $pars,
            $srcp,
            $srcp,
            $width,
            $height,
        ),
        $rv);
    }
}
if ($todel&&$todel!="file") {
    unset($rv);
    return $pics;
} else if ($todel&&$todel=="file") {
    unset($rv);
    if (count($filen[1])>0) {
        return $filen[1];
    }
    return false;
} else

```



```

        return $rv;
    }

function cm_raw2txt($rc) {
    $rv='';
    while (list($k,$v)=each($rc)) {
        $regs=split('<>',$v['text']);
        $vtxt=$regs[0];
        $vurl=$regs[1];
        $tr=str_replace("<?text?>",$vtxt,$v['data']);
        $tr=str_replace("<?url?>",$vurl,$tr);
        for ($i=0;$i<count($regs);$i++) {
            $ii=$i+1;
            $tr=str_replace("<?tag_$ii?>",$regs[$i],$tr);
        }
        $rv.=$tr;
    }
    return $rv;
}

function cm_page($l,$a,$t='', $recurse=array()) {
    global $cm_page_complexity,$cm_page_gentime;

    if (!count($recurse)) {
        $cm_page_complexity=1;
        $benchst=getmicrotime();
    } else {
        $cm_page_complexity++;
    }
    if ($recurse[$l.'?'.$a.'?'.$t])
        return 'Multiple recursive nesting is not allowed';
    $recurse[$l.'?'.$a.'?'.$t]=true;

```

```
$l=cm_cklang($l);  
$a=cm_ckaction($a);  
$t=cm_cktmpl($l,$a,$t);  
$data=cm_gettmpl($l,$t);  
$rc=cm_rawcontent($l,$t,$a);  
$cdata=cm_raw2txt($rc);  
$fc=cm_content($l,$data,$cdata,$a,false,$recurse);  
$cm_page_gentime=getmicrotime()-$benchst;  
return $fc;  
}
```

?>

Литература

1. Deepak Thomas, Wankyu Choi, John Coggeshall, Ken Egervari, et al. *Professional PHP4 Programming*. Wrox Press Inc., 2002.
2. Jeffery Richter. *Applied Microsoft .NET Framework Programming*. Microsoft Press, 2002.
3. Уенди Уилард. *HTML Ръководство на програмиста*. СофтПрес, 2001.
4. Microsoft Press, *Microsoft SQL Server 2000*. Microsoft Press, 2001.
5. Luke Welling, Laura Thomson. *PHP and MySQL Web Development*. Sams, 2001.
6. Paul DuBois, Michael Widenius, *MySQL*. New Riders Publishing, 1999.
7. Sascha Schumann, Harish Rawat, Jesus M. Castagnetto, Deepak T. Veliath. *Professional PHP Programming*. Wrox Press Inc., 1999.