УДК 681.3

Чичкань С. М.

МОДЕЛЬ АВТОМАТИЗОВАНОЇ СИСТЕМИ ПЕРЕТВОРЕННЯ ДОКУМЕНТІВ НА ОБ'ЄКТИ ІНФОРМАЦІЙНИХ СИСТЕМ

У статті розглянуто концепцію структурування змісту документа за допомогою метатегів, що дає змогу автоматизувати процес розпізнавання структурно значущих частин документа та реалізувати різноманітні системи автоматизованої обробки і використання документальних ресурсів.

Вступ

Гіпертекстові технології призначені для розмітки текстового документа особливими мітками (тегами) з метою виділення різноманітних структурних одиниць тексту. Поширені стандарти HTML, PS, PDF, RTF, XML [1-5] дають змогу відображати на екрані або на папері після друку високоякісний форматований текст із включенням нетекстових об'єктів. На жаль, існуючі системи тегів призначені майже виключно для форматування, тобто, відкривши гіпертекстовий документ, ми знаємо, який відступ робити на початку абзацу, яким шрифтом друкувати текст, але нічого не можемо сказати про зміст самого документа. Такі інформаційно значущі елементи, якавтор, назва, час написання та предметна галузь, у кращому випадку описані як атрибути документа й розміщені в його заголовку. У результаті ідеально форматований документ залишається поза контекстом уваги потенційного читача, оскільки читач не може нічого дізнатися про зміст, не прочитавши сам документ.

1. Життєвий цикл документа

На відміну від паперового, електронний документ проходить значно довший життєвий цикл. На першому етапі створення автори тексту часто використовують інші електронні документи як першоджерела та шаблони. На другому етапі життєвого циклу електронний документ розміщують для доступу за допомогою різних технологій і, на відміну від паперового, тривалий час повторно редагують. Можливість публікації в електронному середовищі зумовила появу таких форм роботи з документом, як динамічне оновлення змісту, колективне редагування, легалізація електронними методами, внесення до баз даних, одночасне використання в кількох інформаційних системах тощо. На третьому етапі документ доступний для потенційних читачів, а його зміст асимілюється в інформаційних системах і повторно використовується. Згідно з методологією побудови експертних систем має бути і четвертий етап — виявлення прихованої «структурної» інформації, яка дає змогу класифікувати документ відповідно до галузі знання та контексту і включити його в загальнозначущі бази знань для повторного використання разом з іншими однотипними документами.

На кожному з етапів життєвого циклу, окрім першого, відбувається утилізація вартості, прихованої у змісті документа. Зовнішнє оформлення при цьому відіграє другорядну роль. Невипадково основними напрямами розвитку інформаційних систем на сьогодні стали технології побудови електронних бібліотек, пошукові системи та банки документів.

Стандартні методи забезпечення наповнення .банків та електронних бібліотек основані на ручному перегляді документів і їх класифікації експертами [10] або на технологіях автоматичного аналізу змісту документів на основі наборів ключових слів [11]. Наприклад, у рекомендованому для впровадження стандарті підтримки дистанційної освіти IMS (Instructional Management Systems) [6] визначені кінцевий формат даних у вигляді схем XML, файлів DTD та семантика, що використовується для іменування різних об'єктів, але практично нічого не сказано про те, як досягнути такого рівня структурування. Унаслідок цього у процесі реалізації цілісний навчальний курс, який до того існував у вигляді зв'язного, структурованого підручника, перетворюється на набір окремих фрагментів, розпорошених між різними частинами загальної інформаційної системи — провайдера контенту, підтримки тестів, чатів, дискусій, збірників запитань. З погляду автора, який по-

винен підтримувати та оновлювати курс, бажаний зворотний результат – збереження вихідного документа у цілісному вигляді. Існуючі технології обробки документів перекладають попередню підготовку, аналіз структури та підтримку актуальності на автора, експертів або операторів. Можливим компромісом між негнучкими інформаційними системами та трудомісткою ручною обробкою документів може бути вбудована розмітка вихідних документів, яка б забезпечувала можливість автоматизованої обробки з однієї сторони та була достатньо простою для автора документа.

Існуючі системи гіпертекстової розмітки не обмежені винятково форматуванням тексту, як в PDF, або структуруванням тексту з метою подальшого автоматизованого розпізнавання, як у XML. У більшості гіпертекстових стандартів рекомендовано ігнорувати невідомі теги і, отже, відкритий шлях для визначення додаткових тегів, які б виділяли фрагменти тексту, значущі у контексті, відмінному від форматування або структурування. Саме таким чином текст XML приховано в документах HTML під тегом <?, що дає змогу правильно працювати процесорам HTML, які не розрізняють XML. Тож якщо тип документа допускає гіпертекстову розмітку, безпосередньо в текст можна вставити відповідні теги, які не будуть видимі у документі, але можуть бути засобом розпізнавання окремих смислових частин та побудови відповідного семантичного дерева.

2. Визначення структури документа

У науковій та технічній сферах визначення змістової структури документа давно стало нормою та увійшло до численних стандартів документообігу. У нашій праці ми зробили спробу проектувати вимоги до оформлення змісту документів у відповідні структури, придатні для автоматизованої обробки та створення засобів автоматизовного аналізу контенту.

Змістовні частини документа зазвичай мають прості властивості чіткого розмежування однієї частини від іншої, що не виключає включення однієї частини в текст іншої, а також переходу від попередньої частини до наступної, що відповідає звичайному способу читання тексту зверху вниз. Таким чином, зміст тексту можна виразити у вигляді контекстно вільної граматики, що включає виключно вкладені дужкові вирази, списки таких виразів і фрагментарні включення виразів у тексті батьківського виразу.

Таке визначення елементів змісту документа дає змогу обробляти зміст стандартними засобами гіпертекстових технологій, якщо відповідні змістовні елементи виразити у термінах відповідних тегів. Наприклад, у форматі RTF [4] допускаються вкладені вирази (періоди) та одиночні теги – «слова». Окрім того, частину значущої інформації, що стосується всього документа, можна зберігати або у стандартних періодах («автор», «оператор» тощо), або у визначених користувачем. У документі XML природнішим є визначення схем [5], що відповідають окремим структурним елементам, і заповнення змісту тегів відповідними фрагментами тексту (враховуючи вкладені теги XML). Завдяки уніфікованості різних гіпертекстових мов розмітки кінцева програма-парсер генерується на базі вихідних визначень структури, сформульованих або у вигляді, дуже близькому до формул Бекуса – Наура, або безпосередньо засобами розмітки. Другий спосіб більше відповідає ідеології XML, а перший є класичним методом побудови синтаксичних аналізаторів за допомогою Lex/Yacc [7] або інших програм (див., наприклад, огляд [8]). У нашій праці ми розглядаємо саме комбінацію Lex/Yacc завдяки поширеності, універсалізму та простоті опису вхідних виразів.

Ми пропонуємо схему реалізації аналізатора, або обробника розмітки (рис. 1).

Рис. 1. Схема реалізації аналізатора, або обробника розмітки

Наявність окремої стадії генерації вхідних файлів Lex/Yacc необхідна, оскільки в різних мовах розмітки використовуються синтаксично різні теги і методи їх декорування.

Метою побудови подібного обробника може бути як пошук окремих фрагментів текстово-

го документа, що відповідають заданому елементу структури, так і автоматичне або ручне розпізнавання тексту з метою побудови і збереження структури, що відповідає першій та другій умовним фазам життєвого циклу документа. Для підтримки третьої та четвертої фаз потрібен той самий аналізатор тексту, але з іншою функціональністю – визначення інтерфейсів для збереження посилань на відповідні місця текстових документів та індексації множини попередньо структурованих документів відповідно до вибраної підмножини структур. Наприклад, визначивши такий елемент структури, як тест, у базі документів можна легко знайти всі документи, що містять тести, і створити базу тестових запитань у вигляді посилань до конкретних місць документів. У таких галузях, як дистанційна освіта, виникає ще один цікавий варіант синтаксичного аналізатора – програми, що дає змогу вільно оперувати з деревом структури документа. До таких операцій належать переміщення окремих частин дерева, зв'язування частин різних документів, включення частин з одного документа до іншого. В теорії онтологій [9] подібні операції здійснюють без зміни змісту самого документа на рівні оперування структурами, а доцільність самих операцій відповідає рівневі знань про зміст, тобто можливим способам його інтерпретації.

Повторне використання змісту без його модифікації розглянемо на прикладі підручника у вигляді електронного документа. Його структура цілком відповідає звичайному способу вивчення навчального матеріалу, а саме – читанню підручника від початку до кінця з виконанням включених завдань, вправ, пошуком літературних джерел. Таким чином, дерево змісту документа може слугувати алгоритмом, що визначає послідовність проходження окремих етапів у процесі засвоєння матеріалу в електронному вигляді. Своєю чергою, на рівні онтологій може виявитися доцільнішим інший спосіб послідовності вивчення змісту, який можна отримати шляхом перебудови вихідної лінійної послідовності дій. Якщо онтології вивчення курсу зберігаються окремо від змісту курсів, а самі курси також доступні за уніфікованим механізмом, формальними методами (наприклад, ізоморфного вкладення дерев [12]), можна визначити підмножину курсів, що відповідають заданій онтології. Застосування цієї ідеї засобами XML до еволюції коду детально розглянуто в [13].

3. Базовий набір функцій типового аналізатора та синтаксис вхідних виразів

Відповідно до технології програмування синтаксичних аналізаторів, кожен змістовний фрагмент потрібно описати за такою схемою:

{ім'я фрагмента}::{визначення фрагмента}

Для кожного атомарного фрагмента (тобто такого, який немає сенсу ділити далі) необхідно визначити унікальну константу-ідентифікатор, яку повертатиме кінцевий синтаксичний аналізатор. Таку константу можна визначити і для складних фрагментів, але вона не обов'язково використовуватиметься у кінцевому коді аналізатора. З іншого боку, такий ідентифікатор надзвичайно важливий для побудови аналізатора, орієнтованого на обмін повідомленнями за фактом розпізнавання відповідної структури.

Також потрібно визначити елементарну одиницю тексту, яка містить рівно один структурний елемент, або лінійну послідовність елементів без вкладень. Таке визначення спрощує роботу кінцевого аналізатора в ручному та автоматичному режимі. Ми використовуємо абзац тексту, оскільки в пропонованій праці не розглядаємо деталізації на рівні включених малюнків, формул, таблиць та ін.

Структурою найвищого рівня є контейнер документа – файл, який містить документ, або документальний контейнер Microsoft, який додатково може містити модулі з кодом Visual Basic, шаблони, опис класів та об'єктів OLE, відповідальних за роботу з певним типом документа:

Контейнер_документа :: заголовок_документа? тіло документа return контейнер документа (1)

Знак «?» та інші позначення в цій праці відповідають стандартному синтаксису Lex/Yacc.

Заголовок документа містить загальнозначущу інформацію про нього. Частину цієї інформації можна отримати із заголовків файлів HTML, RTF, PDF, PS, XML, якщо автори документів її там розмістили. Оскільки ім'я автора може бути написане всередині самого документа, необхідно розрізняти відповідні інформаційні поля у заголовку й тілі документа.

Своєю чергою, заголовок може мати вигляд, що відповідає специфікації RTF [4]:

заголовок_документа:: (інфо_рядок)*
return заголовок_документа (2)

де

інфо_рядок :: ім'я_властивості ':' значення_властивості return інфо_рядок (3)

Отже, заголовок документа може містити набір різних структурних елементів, які відрізняються ім'ям. Ми рекомендуємо використовувати набір тегів із відповідними іменами, які визначені для інформаційної групи RTF [4], а саме автора, компанії, оператора, ключових слів, доповнених специфічними для документів інформаційними даними: належністю авторського права, видавцем, електронною адресою базової копії документа та ін. Збір цієї інформації, принаймні частковий, можна реалізувати як у файлах форматів PDF, PS, так і у файлах RTF шляхом пошуку й інтерпретації відповідних тегів.

Структурування самого документа пояснимо на прикладі електронного навчального посібника, який має автора, назву, назву дисципліни, ряд уроків з можливими тестовими запитаннями або вправами наприкінці, а також список літератури і кінцевий тест. Наведемо визначення окремих структурних елементів:

курс – навчальний курс;

урок – елементарна частина навчального курсу. Кожен курс може містити ряд уроків;

вправа – вправа до заданого курсу. Кожен курс і кожен урок може містити вправи;

тест – запитання, задача або завдання для студента. Відповідь на тест дає змогу оцінити рівень засвоєння матеріалу;

вступ – вступ до курсу;

цитата – посилання до першоджерела, рекомендоване джерело;

література – збірник цитат.

Відповідні визначення Бекуса – Наура мають такий вигляд:

курс ::= вступ? текст | вступ? (назва_уроку? урок вправа* тест*)+ (цитата*) (тест*) return курс (4)

У цьому виразі структурний елемент <текст> використаний як замінник будь-якого тексту включно з форматованим текстом та документами. Список літератури може слідувати курсу у вигляді – література ::= цитата* – і не виділений окремо.

Своєю чергою, окремі сутності можуть потребувати подальшої деталізації, наприклад розбиття цитати на окремі смислові частини:

цитата ::= (автор ',')+ ((назва_статті ',' видання ',' рік ',' місце ',' сторінка) | (назва_книги ',' рік ',' місце ',' кількість_сторінок)) return цитата (5)

Структурні елементи нижчого рівня, що виникають у (5), при спробі опису структури цитати на цьому етапі аналізу можна опустити.

4. Генерація синтаксичних аналізаторів

Як було зазначено вище, на основі визначень можна генерувати різні за призначенням синтаксичні аналізатори.

Аналізатор, призначений для ручного або напівавтоматичного розпізнавання вхідного тексту з подальшою розміткою, працює на базі функції get_token(), що повертає черговий неструктурований фрагмент – черговий абзац тексту. Реалізація функці get_token() повністю залежить від формату вихідного файла. Аналіз текстового фрагмента здійснюється або вручну з послідовним декоруванням відповідними тегами, або напівавтоматично, коли для аналізу фрагмента викликається функція yyparse(), яка будує синтаксичне дерево або повертає код помилки, якщо фрагмент тексту неможливо інтерпретувати у вигляді заданих структурних елементів. Вихідний код для Lex залежатиме від вихідного формату файла. Наприклад, для розбору файла RTF необхідно визначити 2 основні стани – перебування всередині заголовка документа – для аналізу інформації, що міститься у заголовку, та перебування в тілі абзацу – для аналізу змісту абзацу. Розпізнавання останнього стану вимагає виключно розпізнавання тегу /par. Вихідний код Yacc буде уніфікованим для всіх форматів файлів документа, оскільки в результаті роботи сканера Lex до основного циклу Yacc передаються розпізнані лексеми з уніфікованими назвами (кодами) елементів. Якщо результатом роботи Yacc визначити декорування фрагмента тексту відповідними тегами та запис результату в інший файл, то ми отримаємо після компіляції файлів Lex/Yacc програму аналізатора на C, яка переписуватиме вхідний файл RTF (або іншого формату) у вихідний файл RTF, розмічений додатковими тегами. Для файлів RTF ми пропонуємо використовувати як теги закладки (bookmarks), оскільки вони зазвичай не відображаються у тексті, можуть містити будь-яку інформацію, в тому числі назву структурного елемента, а також можуть вкладатись одна в одну.

Якщо результатом роботи Yacc визначити побудову синтаксичного дерева, то ми отримаємо програму-аналізатор на C, яка читатиме попередньо розмічений файл RTF і будуватиме дерево, поміщаючи у вузли позиції розміщення закладок. Оскільки один і той самий структур-

ний елемент може зустрічатися кілька разів, доцільно зберігати унікальний номер елемента. Загальний синтаксис закладок можна знайти у [4]:

<book> <bookstart> | <bookend>

<bookstart> '{*' \bkmkstart (*\bkmkcolf*? & *\bkmkcoll*?) #PCDATA '}'

<bookend> '{*' \bkmkend #PCDATA '}'

З метою зберігання в закладках міток структури достатньо визначити структуру поля #PCDATA:

структурна_мітка :: назва_мітки '; ' номер _мітки. (6)

Якщо окрім назви мітки та її номера зберігати стартову і кінцеву позиції всередині документа, то вираз (6) можна використати для автоматичної генерації опису структури або класу C/C++.

Сканер Lex повинен мати 2 стани – стан читання заголовка (оскільки інформація в заголовку не структурується і нашою метою є збереження вихідного формату документа) і стан читання тіла документа, в якому сканер має розпізнавати мітки і передавати Yacc назву мітки разом із текстовим фрагментом, що міститься між стартовою і фінішною міткою. Сканер, що будує послідовність дій, які необхідно виконати з окремими фрагментами структури, міститиме сканер структурованого документа, в якому реалізовано послідовний обхід дерева структури з викликом відповідної функції – обробника вузла.

Програма-маніпулятор структурами у документі має містити аналізатор структурованого тесту та стандартний код маніпуляції вузлами дерев [12].

Аналогічний підхід можна застосувати для розмітки документів HTML, однак враховуючи специфіку формату, їх необхідно вставляти парами виду <start tag [tagname]></start tag [tagname]> фрагмент тексту з тегами HTML <stop tag [tagname]></stop tag [tagname]>. Для розмітки документів PDF можна використати вбудовані словники закладок із внутрішнім механізмом навігації по дереву вкладень. Оскільки назви окремих вузлів можуть бути будь-які, для документів PDF достатньо реалізації стандартного механізму розмітки закладками з іменами закладок, що відповідають іменам окремих структурних елементів.

Повторна утилізація структурної інформації можлива після розміщення документів у спільній базі даних. Структуру такої бази може складати одна таблиця, в якій перераховані назви та шлях до файлів документів. Якщо необхідно побудувати базу тестів, то природним підходом буде генерація таблиці з тестами і складеним ключем, що містить назву документа та номер тесту в ньому.

Висновки

Запропонована система додаткової тегової розмітки документів, призначеної для визначення структурно значущих елементів, може бути реалізована у вигляді прихованої розмітки файла гіпертекстового документа у вихідному форматі. У результаті можна отримати зручний та ефективний засіб індексування змісту документа, аналізу його структури та генерації обробників документа, наприклад, з метою автоматичної генерації контенту або управляючого процесу для його обробки.

1. *Эпштейн В. Л.* Введение в гипертекст и гипертекстовые системы.— URL: http://www.ipu.rssi.ru/publ/epstu.htm.
2. PostScript Language, Tutorial and Cookbook, Adobe Systems Inc.— Addison - Wesley, 2003.
3. PDF Reference version 1.4, Adobe Systems Inc.- Addison - Wesley, 2003.
4. Microsoft Office Word 2003 Rich Text Format (RTF) Specification, published April 2004, www.microsoft.com
5. *Megginson David.* Structuring XML Documents.— Prentice-Hall, 1998.
6. IMS Global Consortium, Inc. (2003). 'IMS Content Package Information Model' Retrieved: September 20, 2004: http://www.imsglobal.org/content/packaging/cpvlplp3/imscp_infovlplp3.html.
7. *Aaby Anthony A.* Compiler Construction Using Flex and Bison.— Walla College, cs.wwc.edu, 2004.
8. *Grune Dick, Jacobs Ceriel.* Parsing Technics. A Practical Guide.- Vrije Universiteit, Amsterdam, The Netherlands, 1998.
9. *Chandrasekaran B., Josephson J. R., Benjamins V. R.* Ontologies: What are they? Why do we need them? // IEEE Intelligent Systems and Their Applications, 14(1):20-26, 1999.
10. *Stanchev Peter.* The "Web Technology" Distance Course // Conference on Information Technology in Education, Elizabethtown, PA, September 18, 2004.
11. *Óztiirk Meltem, Tsoukias Alexis, Vincke Philippe.* Preference Modelling *U* DIMACS Technical Report 2003-34, October 30, 2003.
12. *Baxter I.* et. al. Clone Detection Using Abstract Syntax Trees // Proc. International Conference on Software Maintenance, IEEE, 1998.
13. *Soumen Sarkar, Craig Cleaveland.* Code generation using XML-based document transformation.- 2001, published by TheServerSide, J2EE community.

S. M. Chychkan

THE MODEL OF AN AUTOMATED SYSTEM FOR DOCUMENT TRANSFORMATION INTO THE INFORMATION SYSTEM OBJECTS

The concept of automated document structuring has been applied to the document content. The context free grammar expressions have been proposed as abstract schemasfor building different document processing applications and new ways for using the existing document resources.