УДК 681.3

В.М. Грищенко

ПІДХІД ДО СТВОРЕННЯ WEB-ЗАСТОСУВАНЬ НА КОМПОНЕНТНІЙ ОСНОВІ

Розглядається підхід до розробки Web-застосувань на основі моделей та методів компонентного програмування. У рамках підходу визначаються базові концепції, типова архітектура таких застосувань, типові структури та об'єкти даних, типова схема обробки запитів та генерації відповідей. Наведено опис практичної реалізації Web-застосування на базі розроблених концепцій, архітектури та схеми обробки даних.

Вступ

Особливістю сучасних тенденцій в програмуванні та розробці програмних систем є орієнтація на мережу Інтернет та її окремі технології. Спектр застосування відповідних рішень досить широкий. Це і класичні застосування, найбільш відомим з яких є інформаційні системи на основі Web-сайтів [1], і альтернативна реалізація клієнт-серверних систем зі спрощеним клієнтом ("тонким" клієнтом) [2], і Web-сервіси, які дозволяють будувати інтегровані системи на основі як нових компонентів, так і з застосуванням старих систем [3], та ін.

Такий широкий спектр рішень та їх значна розповсюдженість обумовлюються значною кількістю переваг Інтернет-рішень, найбільш важливими з яких є наступні:

Типовість рішень. Інтернет-рішення створюються на основі типових архітектур, протоколів, механізмів побудови та взаємодії. Наприклад, типовою архітектурою Web-застосувань є клієнт-серверна архітектура. У якості клієнта застосовується Інтернет-браузер, який підтримує стандартизовані протоколи та інтерфейси. Сервером є Web-сервер, який також функціонує на основі типових протоколів та механізмів.

Мобільність рішень. Інтернет-рішення мають високий рівень мобільності. Наприклад, такі Інтернет-протоколи, як TCP/IP, HTTP, реалізовані для самої мережі Інтернет, для корпоративних мереж, для локальних обчислювальних мереж і навіть для окремого автономного комп'ютера. Тому рішення на основі цих протоколів без великих зусиль адаптується до різних мережних середовищ.

Властивість щодо маштабування рішень. Така властивість обумовлена самою еволюцією мережі Інтернет, її протоколів та типізацією методів маштабування. Тому Інтернет-рішення без значних зусиль адаптуються до розподілених систем з широкими діапазонами значень кількості кінцевих користувачів та навантажень на окремі ділянки системи.

Менша вартість створення та застосування, ніж для інших типів рішень. Зменшення вартості Інтернет-рішень досягається за рахунок кількох факторів, найбільш важливими з яких є:

– застосування браузерів, що спрощує розробку й експлуатацію клієнтських місць і інтерфейсів кінцевого користувача;

– застосування існуючих програмних компонентів, значна кількість яких безкоштовні і тому можуть використовуватись без закупівлі спеціальних ліцензій;

– створення нових Web-застосувань без зміни існуючої інфраструктури (мережне середо-

вище, Web-сервери, браузери та ін.);

– спрощення процедур розгортання та інсталяції (ці процедури мають великий рівень стандартизації та уніфікації).

При розробці Інтернет-рішень застосовується широкий спектр підходів, методів та засобів, які вже знайшли своє відображення у великій кількості публікацій, наприклад [4–6]. В цілому, відповідні технології, засоби, мови програмування мають багато можливостей щодо розробки Web-застосувань. Проте, як правило, ці можливості загалом стосуються забезпечення зручності та збільшення ефективності роботи розробників Інтернет-систем. Тому впровадження сучасних методологій програмування в розробку Web-застосувань з метою досягнення більш якісних характеристик процесів та продуктів є актуальною задачею сьогодення.

Однією з сучасних методологій є компонентне програмування. Цей підхід забезпечує багато переваг при створенні програмних систем і, зокрема, Web-застосувань. Ідеї компонентного програмування в тій чи іншій мірі застосовуються у багатьох розробках, проте вони у більшості випадках носять обмежений характер. На відміну від інших робіт ця стаття послідовно та комплексно аналізує різні аспекти та процеси створення Web-застосувань на компонентній основі, що забезпечує цілісність підходу, що пропонується.

Під використанням компонентної методології [7, 8] розуміється застосування апарату компонентного проектування та програмування щодо різних аспектів побудови Web-орієнтованих систем. Методологія включає і певні об'єктно-орієнтовані концепції та методи. Тому, по суті, вона носить об'єктно-компонентний характер.

Концепції компонентного підходу щодо розробки Web-застосувань

Основними концепціями, що визначають сутність застосування компонентного підходу і значним чином впливають на особливості процесів проектування та розробки, є такі:

Ієрархічна властивість компонентного підходу. Ця концепція означає, що різні аспекти, характеристики, елементи, які пов'язані зі створенням Web-застосування, мають ієрархічну впорядкованість у процесі їх деталізації. Наприклад, на початку проектування Web-застосування подається як певна сукупність компонентів — компоненти першого рівня ієрархії. Кожен з них може мати власну структуру і у подальшому подаватись як сукупність більш деталізованих компонентів другого рівня. При цьому відповідні компоненти першого рівня відіграють роль контейнерів. Така процедура може повторюватись кілька разів із збереженням ієрархічної залежності.

Розповсюдженість ідей компонентного програмування на всі етапи життєвого циклу. Це означає, що вимоги, процеси, процедури, обмеження кожного етапу розробки подаються як відповідні елементи з орієнтацією на компонентну методологію. Наприклад, етап формування вимог до Web-застосування повинен включати вимоги, які властиві будь-яким компонентним програмам. Це, зокрема, наступні:

– Web-застосування будується на основі клієнт-серверної архітектури;

– Web-сервер та Web-клієнти повинні бути подані як компоненти і відповідати моделі компоненту;

– функціональність Web-

застосування визначається інтерфейсами його компонентів;

– взаємодія між компонентами визначається лише на основі інтерфейсів.

Адаптація компонентного підходу до різних типів елементів Web-застосувань. Ця концепція вимагає, щоб усі елементи застосування були подані як компоненти чи об'єкти. Наприклад, програмний елемент (відповідно до моделі компонента) повинен мати унікальне ім'я, визначений інтерфейс та не менше однієї реалізації [9]. Дані для Web-застосування подаються як об'єкти даних з реалізацією відповідного інтерфейсу.

Адаптація компонентного підходу до різних схем та механізмів реалізації Web-застосувань. Наприклад, стандартною схемою реалізації механізму звернення до функцій компонента є метод віддаленого виклику процедур. Необхідною умовою цього методу є обов'язкове визначення імені функції у певному компоненті та множини вхідних та вихідних параметрів [8].

Для Web-застосувань цей метод трансформується у наступну схему. Необхідний для звертання компонент визначається на основі його URL [1], тобто сукупністю, яка складається з імені домену Інтернету порту Web-серверу, контекстного маршруту та імені серверного компонента, що приймає та виконує обробку запитів від клієнта. Список вхідних параметрів є множиною CGI-параметрів [1], кожен з яких є пара: ім'я параметра і його значення. Вихідним параметром є HTML-сторінка, яка відображає результати запиту.

З наведених концепцій існує декілька важливих висновків.

Висновок 1. Внутрішня структура серверної частини Web-застосування повинна бути інкапсульована для клієнтів. Зокрема, це стосується і структури каталогів Web-серверу. Внаслідок цього при формуванні посилань на окремі HTML-сторінки чи інші компоненти обробки запитів безпосереднє застосовування маршрутів з послідовності серверних каталогів є порушенням компонентних умов.

Висновок 2. Функція серверного компонента, яка виконується для обробки запиту та генерації відповіді, визначається на основі URL (тобто тільки іменем компонента) або URL та сукупністю параметрів запиту клієнта і їх значеннями. Цей висновок є наслідком інтерфейсних вимог та реалізації механізму віддаленого виклику процедур для Web-застосувань. Взаємодія з серверними компонентами повинна відбуватись за допомогою їх інтерфейсів. Інтерфейс між клієнтом та сервером (CGI-інтерфейс) визначає тільки ім'я компонента, але не його функції. Тому повинен існувати механізм, який на основі аналізу сукупності параметрів та їх значень, визначає конкретну функцію компонента. У частковому випадку кожна функція може бути реалізована як окремий компонент. Тоді між компонентами та функціями існує однозначна залежність, яка не порушується будь-якими сукупностями параметрів та їх значеннями.

Висновок 3. Функціональність клієнта визначається тільки HTML-сторінкою, яка є відповіддю на попередній запит. Зокрема, це означає, що його функціональність може динамічно змінюватись як для одного й того ж Web-застосування, так і для довільної кількості.

Розробка узагальненої архітектури Web-застосування

Узагальнена архітектура побудована з використанням згаданих вище концепцій. Вона відображає множину моделей, методів, засобів, які є базовими при побудові Web-застосувань, а також місце кожної складової у цій архітектурі. Сама архітектура подається як впорядкована сукупність логічних рівнів окремо для клієнта та сервера. Кожен з рівнів підтримується певними технологіями, мовами програмування, засобами та ін., які забезпечують реалізацію цього рівня для конкретного Web-застосування. Узагальнена архітектура Web-застосування зображена на рис. 1.

Web-застосування (у першому наближенні) подається як множина компонентів першого рівня ієрархії — клієнтів та серверів (у загальному випадку серверів може бути кілька, кожен з яких може мати функціональні компоненті, що входять до складу одного Web-застосування). У подальшій деталізації клієнт проектується як сукупність браузера та елементів, які функціонують у його середовищі. Браузер є основою клієнту і одночасно розглядається як контейнер, який забезпечує управління та функціонування відповідних компонентів другого рівня ієрархії. Ці компоненти також розглядаються як структурні елементи, які поділяються на менші компоненти. Такий процес деталізації виконується за кілька кроків.

Аналогічним чином серверна частина подається як сукупність Web-серверу (відіграє роль контейнера) та більш деталізованих серверних компонентів. Ці серверні компоненти розглядаються як базовий рівень для реалізації серверної логіки, яка, у свою чергу, також проходить декілька кроків деталізації у відповідності до свого функціонального призначення та місця у процесі обробки запитів клієнтів.

Рис. 1. Узагальнена архітектура Web-застосування

Рівні архітектури клієнта Web-застосування

Розглянемо рівні архітектури клієнта, наведені на рис. 1. У загальному випадку вони відображають послідовність дій на клієнті, що передують безпосередньому відображенню інформації, яка отримана внаслідок запиту до сервера Web-застосування.

Рівень розподіленого середовища. Цей рівень є логічним поданням мережевого середовища, що складається з Інтернет, корпоративних, локальних обчислювальних мереж та ін. На розробку Web-застосувань значним чином впливають протоколи середовища, які забезпечують взаємодію Web-компонентів. Це, зокрема, транспортний протокол TCP/IP, базовий протокол для взаємодії клієнтів та серверів HTTP, спеціальні протоколи взаємодії Web-компонентів на базі протоколу HTTP, як, наприклад, SOAP для взаємодії з Web-сервісами, та ін.

Базовий рівень клієнта. Типовою реалізацією цього рівня є Інтернет-браузери. Браузер визначає середовище клієнта, забезпечує формування та посилання запитів до серверного компонента, отримання та базову обробку відповіді від сервера. Крім цього, браузер забезпечує механізми підтримки реалізації інших рівнів архітектури клієнтів Web-застосувань.

Рівень формування об'єктів даних клієнта. На цьому рівні відбувається створення та формування об'єктів даних для реалізації функціональності клієнта. Найчастіше об'єкти даних подаються як ієрархічно впорядкована множина. Функції побудови та впорядкування виконуються браузером. Кожен об'єкт має певні сукупності атрибутів та властивостей, а також методів, які визначають його функціональність.

Об'єкти цього рівня можуть мати різні типи та природу щодо створення і застосування. Наприклад, до складу цих об'єктів

входять елементи для відтворення візуального інтерфейсу користувача (вікна, списки, кнопки і ін.), забезпечення логічної структури клієнта (фрейми), компоненти ActiveX та Java-аплети, що реалізують додаткову функціональність, яка залежить від типу Web-застосування, та ін. [6].

Функціональний рівень клієнта. Функції цього рівня поділяються на два види. До першого відносяться ті, які виконуються під час формування запиту до сервера застосування на основі даних, введених кінцевим користувачем. Типовими функціями є перевірка синтаксису та, частково, семантики даних (розмір елементів, діапазони числових даних, наявність спеціальних символів та ін.). Функції другого типу забезпечують обробку даних, які отримані у відповідь з боку сервера, для формування візуального подання результатів обробки запиту.

У найбільш загальному випадку певні функції можуть динамічно створювати та формувати нові об'єкти даних, що забезпечує динамічне перетворювання моделі даних клієнтів. Для реалізації функцій цього рівня застосовуються скриптові мови (наприклад, JavaScript) та універсальні мови програмування C, C++, Java та ін. (реалізації компонентів ActiveX та Java-аплетів) [6].

Рівень відображення об'єктів даних. На цьому рівні безпосередньо відбувається формування візуального зображення, яке може бачити користувач клієнта Web-застосування. Відображення будується відповідно до ієрархічній моделі даних згідно з правилами відображення для кожного типу об'єктів даних. Для реалізації цього рівня застосовуються мови розмітки даних (HTML, XML та ін.).

Рівні архітектури серверної частини Web-застосування

Як і для випадку з архітектурою клієнта, рівні архітектури серверної частини в загальному випадку відображають послідовність дій на серверному компоненті, що виконуються для обробки запиту та формування відповіді.

Рівень розподіленого середовища. Призначення та функції цього рівня повністю співпадають з аналогічним рівнем для клієнтів. В найбільш загальному випадку розподілене середовище не впливає на тип компонентів Web-застосування, тобто в певному вузлі середовища може знаходитись компонент, який одночасно є сервером і клієнтом (визначення типу компонента відбувається на більш високих рівнях архітектури).

Базовий рівень Web-сервера. На цьому рівні розташовані Web-сервери та контейнери, які забезпечують керування серверними компонентами застосувань та підтримку їх взаємозв'язків з іншими компонентами середовища (наприклад, компонентами, що реалізують сервіси іменування, пошуку, подій, транзакцій та ін.).

Рівень компонентів Web-сервера. Цей рівень архітектури подається як сукупність компонентів, що забезпечують функціональність серверної частини Web-застосувань щодо обробки запитів з боку клієнтів та формування відповідних результатів. Сучасна архітектура обробки запитів, як правило, заснована на шаблоні проектування MVC (model-view-controller) [10]. Ця архітектура передбачає відділення подання даних від функцій безпосереднього формування результатів обробки запитів. До складу шаблону входять правила та механізми реалізації його складових та їх взаємодії.

Відповідно до шаблону MVC в найбільш загальному випадку серверні компоненти виконують функції контролера (типовий приклад реалізації — Java-сервлети), функції подання та обробки даних (реалізуються із застосуванням COM, JavaBean, EJB, Corba-об'єктів) [5], функції формування та подання результатів клієнтських запитів (наприклад, ASP, JSP, HTML-сторінки).

Рівень керування обробкою запиту є логічним поданням моделі обробки запитів клієнтів. Відповідно до цієї моделі існують наступні групи функції щодо обробки запитів.

Аналіз запитів. Перевіряються синтаксис та семантика запитів, зокрема:

– наявність та структура певних параметрів запиту;

– типи даних значень параметрів;

– обмеження на значення параметрів;

– тип та вид запиту.

Обробка запиту. Виконується безпосередня обробка даних щодо запиту та формування об'єктів даних для генерації відповіді. До складу цих функцій, зокрема, входять:

– пошук та відбір даних;

– операції над даними;

– внесення змін до баз та файлів даних;

– визначення об'єктів даних для генерації відповіді;

– виконання операцій щодо умов забезпечення цілісності об'єктів даних.

Генерація відповіді. Відбувається формування HTML-сторінки, що є поданням результату обробки запиту. До складу функцій входять:

– обробка інформації відповідно визначеним об'єктам даних для виконання умов їх візуалізації на клієнтській частині;

– формування цілісної HTML-сторінки на основі шаблону відповіді та результатів обробки окремих об'єктів даних;

– посилання сформованої HTML-сторінки відповідному клієнтові.

Рівень реалізації операцій обробки даних. Цей рівень архітектури є логічним рівнем операцій для обробки даних. Відповідно до загальних концепцій побудови компонентної архітектури кожна операція над даними інкапсулюється у відповідному об'єктові даних. Кожен об'єкт даних є типовим елементом, що реалізує виконання типових операцій для певного застосування, і визначається наступними характеристиками:

– вхідними даними;

– множиною операцій над вхідними даними;

– множиною вихідних даних, які пристосовані для обробки об'єктом даних або компонентом вищого рівня в об'єктно-компонентній ієрархії.

Типовими елементами цього рівня є об'єкти даних, які:

– відбирають з баз або файлів даних певну інформацію відповідно критеріям відбору;

– виконують логічну обробку даних;

– генерують фрагменти, які є складовими відповіді для клієнта.

Рівень подання даних. На цьому рівні архітектури знаходяться елементи, які є поданням даних, що застосовуються під час обробки запиту клієнта. Дані мають різну природу та призначення. В найбільш загальному випадку вони поділяються на наступні:

– дані, які є інформаційною основою Web-застосування і призначаються для безпосереднього

(чи з попередньою обробкою) відображення на клієнті;

– дані, які описують множини вхідних даних (включаючи пошук та відбір) для об'єктів даних, які визначені на попередньому рівні архітектури;

– дані, які описують послідовність операцій з множини операцій об'єкта даних для отримання результату відповідно до запиту клієнта.

Визначення структур типових об'єктів даних для Web-застосування

Існує багато способів визначення типових структур даних, кожна з яких має свої особливості та обмеження щодо застосування. У підході, що пропонується, визначаються найбільш характерні типові структури:

– структури об'єктів, які описують інтерфейси між клієнтами та серверними компонентами (типові структури для запитів та відповідей);

– структури об'єктів, що визначають типові операції для генерації серверних відповідей.

Типова структура запиту клієнта. Інтерфейс між клієнтськими та серверними частинами певного Web-застосування визначається сукупністю клієнтських запитів, що відображається у наступному виразі:

$$\mathbf{WebAppInterface} = \{\mathbf{Request}^{p}\}, \quad (1)$$

де $\mathbf{Request}^{p}$ – p-й запит.

Запит формується на основі інтерфейсу CGI. Відповідно до властивостей цього інтерфейсу загальна структура типового клієнтського запиту має наступний вигляд:

$$\mathbf{Request}^{p} = (\mathbf{URL}^{p}, \mathbf{ParamList}^{p}), \quad (2)$$

де $\mathbf{URL}^{p}$ визначає серверний компонент, який приймає запит;

$\mathbf{ParamList}^{p}$ – множина параметрів для цього запиту.

Множина параметрів подається як сукупність пар:

$$\mathbf{ParamList}^{p} = \{(\mathbf{ParamName}^{pq}, \mathbf{ParamValue}^{pq})\}, \quad (3)$$

де $\mathbf{ParamName}^{pq}$ – ім'я q-го параметру p-го запиту;
$\mathbf{ParamValue}^{pq}$ – його значення.

У частковому випадку для деяких запитів параметри можуть бути відсутні і тоді $\mathbf{ParamList}^{p} = \varnothing$.

Типова структура відповіді Web-сервера. Безпосередня серверна відповідь є HTML-сторінкою, процес формування якої залежить від технологій, що застосовуються на сервері. Проте у будь-якому випадку структура відповіді складається з двох частин:

– шаблон відповіді, що визначає загальний вигляд HTML-сторінки і який є постійним об'єктом для цілого Web-застосування або певної підмножини його відповідей (шаблон має типову структуру, постійну розмітку та описує елементи сторінки, які будуть відображатись на клієнті без змін);

– динамічні фрагменти, що генеруються сервером відповідно до типу запиту від клієнта і які вносяться у наперед визначені місця шаблонів як окремі елементи (у загальному випадку для деяких типів запитів фрагменти можуть бути відсутні).

Шаблон з відповідними динамічними фрагментами є основою для формування остаточної HTML-сторінки, яка посилається до клієнта. Згідно описаної схеми типова відповідь подається як наступна структура:

$$\mathbf{ResType}_i = (\mathbf{PageLayout}_i, \mathbf{DataObjectResultList}_i), \quad (4)$$

де $PageLayout_i$ визначає шаблон відповіді для i-го запиту;
$DataObjectResultList_i = \{DataObjectResultList_{ik}\}$ — множина фрагментів, кожен з яких є результатом застосування об'єктів даних для генерації відповіді на i-й запит.

Сам фрагмент подається як наступний вираз:

$$DataObjectResultList_{ik} = DataObjectList_{ik}.createFragment_{ik}, \quad (5)$$

де $DataObjectList_{ik}$ — певний елемент з множини об'єктів даних генерації відповіді, що входять до складу компонентів серверної частини Web-застосування;
$createFragment_{ik}$ — типова функція генерації фрагмента для цих компонентів.

Необхідно відмітити, що деякі динамічні фрагменти можуть бути однаковими для різних шаблонів. Це дозволяє спроектувати процес генерації HTML-сторінок таким чином, щоб визначити типові операції обробки запитів для певного Web-застосування або кількох застосувань.

Типова структура об'єкта даних для генерації відповіді. Кожен такий об'єкт даних визначає елементарну функцію обробки певних типів запитів та генерації визначених фрагментів відповіді. Типовим прикладом елементарних функцій може бути пошук інформації у базі даних. Результат обробки клієнтського запиту буде формуватись на основі деякого шаблону та двох фрагментів: один відображає форму з параметрами запиту, а другий — результат пошуку інформації у базі даних. Згідно з цим будуть існувати два об'єкти даних, які генерують вказані фрагменти. Перший об'єкт даних може бути типовим для різних запитів, де є необхідність пошуку інформації у базах даних, а другий може застосовуватись для різних баз даних з аналогічними моделями даних.

Типова структура об'єкта даних для генерації відповіді може бути описана наступним виразом:

$$DataObjectList_{ik} = (ParamList^i, AppData, FuncList_{ik}, ResName_{ik}), \quad (6)$$

де $ParamList^i$ (враховуючи відповідність індексів i та p) описується виразом (3);
$AppData$ — множина даних Web-застосування (бази даних, файли та ін.);
$FuncList_{ik}$ — множина операцій для цього об'єкта даних (зокрема, до цієї множини входить функція $createFragment_{ik}$);
$ResName_{ik}$ — ім'я об'єкта для k-го фрагмента.

Відносно визначення та застосування об'єктів даних для генерації відповідей доцільно зробити наступні зауваження.

У найбільш загальному випадку $AppData$ описує всю множину даних Web-застосування. При практичній реалізації найчастіші випадки, коли для певного об'єкта даних визначається лише деяка підмножина $AppData$ (наприклад, певне відношення бази даних або конкретний файл).

Множини $FuncList_{ik}$ складаються не тільки з функцій генерації фрагментів відповідей. Зокрема, вони можуть включати функції обробки даних, пошуку у базах даних та ін. Тобто перед зверненням до функції $createFragment_{ik}$ може бути виконана послідовність методів, яка передує безпосередній генерації.

Ім'я $ResName_{ik}$ визначається у шаблоні $PageLayout_i$ для k-го фрагмента. Якщо деякий об'єкт даних генерує фрагмент з цим ім'ям, то для цього шаблону у визначене місце HTML-сторінки для відпові-

ді включаються дані з вказаного фрагмента.

Побудова типової схеми обробки запитів клієнтів

Типова схема обробки запитів клієнтів базується на наступних засадах:

1. Забезпечення відповідності наведеній вище архітектурі при розробці Web-застосування дозволить у повній мірі реалізувати основні переваги компонентного підходу, зокрема:

– ієрархічний підхід до розробки компонентів (при деталізації певного компонента за стандартними правилами у ньому реалізується властивості контейнера, а його функціональність поділяється на частини меншого розміру, кожна з яких також за визначеними правилами оформлюється як компонент);

– єдиний механізм керування компонентами, який не залежить від рівня ієрархії, на якому знаходяться компоненти;

– гнучкість компонентної структури (доповнення, модифікація, заміна компонентів проводиться без особливих зусиль, що дозволяє спростити зміну функціональності Web-застосування у період експлуатації та підтримки);

– повторне використання компонентів в інших застосуваннях.

2. Типові структури та об'єкти даних, типові операції та процеси обробки, типові компоненти та інші елементи мають відповідні описи, які складають конфігураційну базу даних (БД) Web-застосування. Ця конфігураційна БД може змінюватись відповідно до розширення та зміни функціональності Web-застосування.

3. Засоби обробки конфігураційної БД та засоби обробки запитів клієнтів відповідно до описів з цієї БД складають програмний каркас (framework) Web-застосування, який одночасно є середовищем для керування іншими компонентами. Наприклад, розширення функціональності Web-застосування відбувається шляхом доповнення конфігураційної БД та розгортанням додаткових компонентів, які підключаються та керуються за допомогою framework.

Конфігураційна БД формується на основі опису типових структур та об'єктів даних, що були розглянуті вище у цій статті. Вона містить наступні структури даних:

Опис типів клієнтських запитів. Кожен тип визначається сукупністю параметрів запиту, їх значеннями та обмеженнями на значення (наприклад, значення складають певний діапазон чи тип даних). Цей опис подається наступними виразами:

$$\boldsymbol{ReqType} = \{\boldsymbol{ReqType_i}\}, \qquad (7)$$

де $\boldsymbol{ReqType_i}$ – i-й тип запиту.

Кожен тип подається як

$$\boldsymbol{ReqType_i} = (\boldsymbol{ReqTypeName_i}, \boldsymbol{ReqParamList_i}\}, \qquad (8)$$

де $\boldsymbol{ReqTypeName_i}$ – ім'я типу запиту;

$\boldsymbol{ReqParamList_i}$ визначає сукупність параметрів, їх значень та обмеження на значення для відповідного типу запиту.

Сукупність параметрів описується виразом:

$$\boldsymbol{ReqParamList_i} = \{(\boldsymbol{ParamName_{ij}}, \boldsymbol{ParamValue_{ij}}, \boldsymbol{Constraints_{ij}})\}, \qquad (9)$$

де $\boldsymbol{ParamName_{ij}}$ – ім'я j-го параметру i-го типу запиту;

$\boldsymbol{ParamValue_{ij}}$ – значення j-го параметру i-го типу запиту;

$Constraints_{ij}$ визначає обмеження на значення для j-го параметру i-го типу запиту.

Необхідно відзначити, що $ParamName_{ij}$, $ParamValue_{ij}$ з виразу (9) та $ParamName^{pq}$, $ParamValue^{pq}$ з виразу (3) можуть співпадати за значеннями, але вони мають різну сутність. Перші є елементами опису типів запитів, а другі — елементами даних з реальних запитів клієнтів. Тому, наприклад, не для кожного $ParamName^{pq}$ може існувати відповідне $ParamName_{ij}$ (такі запити класифікуються як помилкові).

Опис типів відповідей Web-сервера. Цей опис визначається виразами (4) та (5). При цьому $PageLayout_i$ є ідентифікатором, який визначає ім'я відповідного шаблону.

Опис типових об'єктів даних для генерації відповідей. Цей опис розглядається як об'єднання об'єктів даних, що описуються виразом (6). Тобто він має вигляд

$$AppDOList = \{DataObjectList_{ik}\}, \quad (10)$$

де $AppDOList$ — множина об'єктів даних для генерації відповідей певного Web-застосування; $DataObjectList_{ik}$ описується виразом (6).

Опис адаптерів генерації відповідей. Опис зв'язує тип запиту, тип відповіді та компонент, що безпосередньо реалізує процес генерації, у єдину інформаційну структуру, яка визначає певний варіант обробки запиту:

$$AdapterList = \{ReqTypeName_i, PageLayout_i, Adaptert_i\}, \quad (11)$$

де $ReqTypeName_i$, $PageLayout_i$, $Adaptert_i$ — відповідно ім'я типу запиту, шаблон відповіді та адаптер (компонент) реалізації генерації для відповідного випадку.

Типова схема обробки клієнтських запитів складається з наступних кроків. Сервер приймає запит як структуру даних, що описується виразами (2) та (3). Аналізатор запитів на основі виразів (7)-(9) шляхом співставлення з параметрами та їх значеннями зі структури (3) визначає тип запиту $ReqTypeName_i$. Якщо результат співставлення негативний, то сервер вважає, що запит помилковий і формує відповідну HTML-сторінку.

Компонент ініціації обробки запиту на основі виразу (11) вибирає шаблон відповіді $PageLayout_i$ та адаптер генерації $Adaptert_i$ ($ReqTypeName_i$ є ключем пошуку в опису адаптерів генерації). Адаптер на основі значення $PageLayout_i$ виконує пошук в опису типів відповідей, визначений виразом (4), і вибирає відповідний перелік фрагментів

DataObjectResultList$_i$, які повинні бути згенеровані. Відповідно до переліку з ***AppDOList*** вибираються об'єкти даних ***DataObjectList$_{ik}$***. Потім адаптер ініціює послідовність роботи цих об'єктів даних, які формують фрагменти згідно з типом запиту. Після генерації фрагментів адаптер переадресує обробку шаблону ***PageLayout$_i$***, який формує остаточний вид HTML-сторінки для відповіді.

Практична реалізація Web-застосування на компонентній основі

Запропонований у цій статті підхід був використаний для проектування та реалізації реального Web-застосування. Це застосування було розроблене з метою створення інформаційної системи вивчення та дослідження структур моделей даних для реляційних баз даних. Множина ***AppData*** складалася:

– з файлів, що описували структуру моделі даних;

– файлів з документацією для моделі даних (ця документація застосовувалась при вивченні предметної області, для отримання довідок про терміни та поняття тощо);

– конфігураційних файлів для опису типових структур об'єктів даних для Web-застосування.

Модель даних реляційної БД описувалась сукупністю XML-файлів, до складу яких входили:

файли опису відношень (таблиць) моделі даних;

– файл опису зв'язків між відношеннями (ключі, індекси);

– файли опису тематичних областей (підмножини відношень з відповідними зв'язками);

– головний файл моделі даних, який об'єднує усі окремі файли описів.

Документація щодо моделі даних містила файли з наступними форматами даних: TXT, HTML, XML, DOC.

Конфігураційні файли, які були подані як XML-файли, включали:

– опис типів та структур клієнтських запитів;

– опис структур серверних відповідей;

– опис об'єктів даних для типових операцій при генерації відповідей;

– безпосередній опис конфігурації та місцезнаходження компонентів та даних Web-застосування, у тому числі перелік адаптерів для генерації відповідей.

У якості шаблонів серверних відповідей застосовувались JSP-сторінки [11], в яких у відповідних місцях існували оператори INCLUDE, що вставляли фрагменти сторінок з файлів, які формувалися об'єктами даних для типових операцій генерації. Множина об'єктів даних для типових операцій генерації містила наступні реалізації:

– генерація фрагментів для отримання інформації про окремі елементи з опису моделі даних (був застосований механізм XSLT-трансформації для XML-файлів з описом [12]);

– формування фрагментів для отримання інформації з файлів документації (інформація подавалась як HTML-код шляхом відповідних трансформацій між різними форматами);

– формування фрагментів з результатами пошукових запитів щодо окремих елементів у XML-файлах з описом моделі даних реляційної БД;

– створення та модифікація додаткових XML-файлів для опису моделі даних;

– створення та відображення графічного подання (діаграм) опису моделі даних та її тематичних областей.

Схематична структура застосування та процесу обробки даних подана на рис. 2.

Клієнти були реалізовані на основі Інтернет-браузерів MS Internet Explorer та Netscape Navigator. Серверний компонент включав Web-сервер (практично без змін були створені два варіанти – для Apache+Tomcat та JBoss), Java-сервлет (як компонент, що приймав та ініціював обробку клієнтських запитів) та компоненти, що виконували функції аналізатора запитів, формування відповідей, генерації фрагментів відповідей та ін.

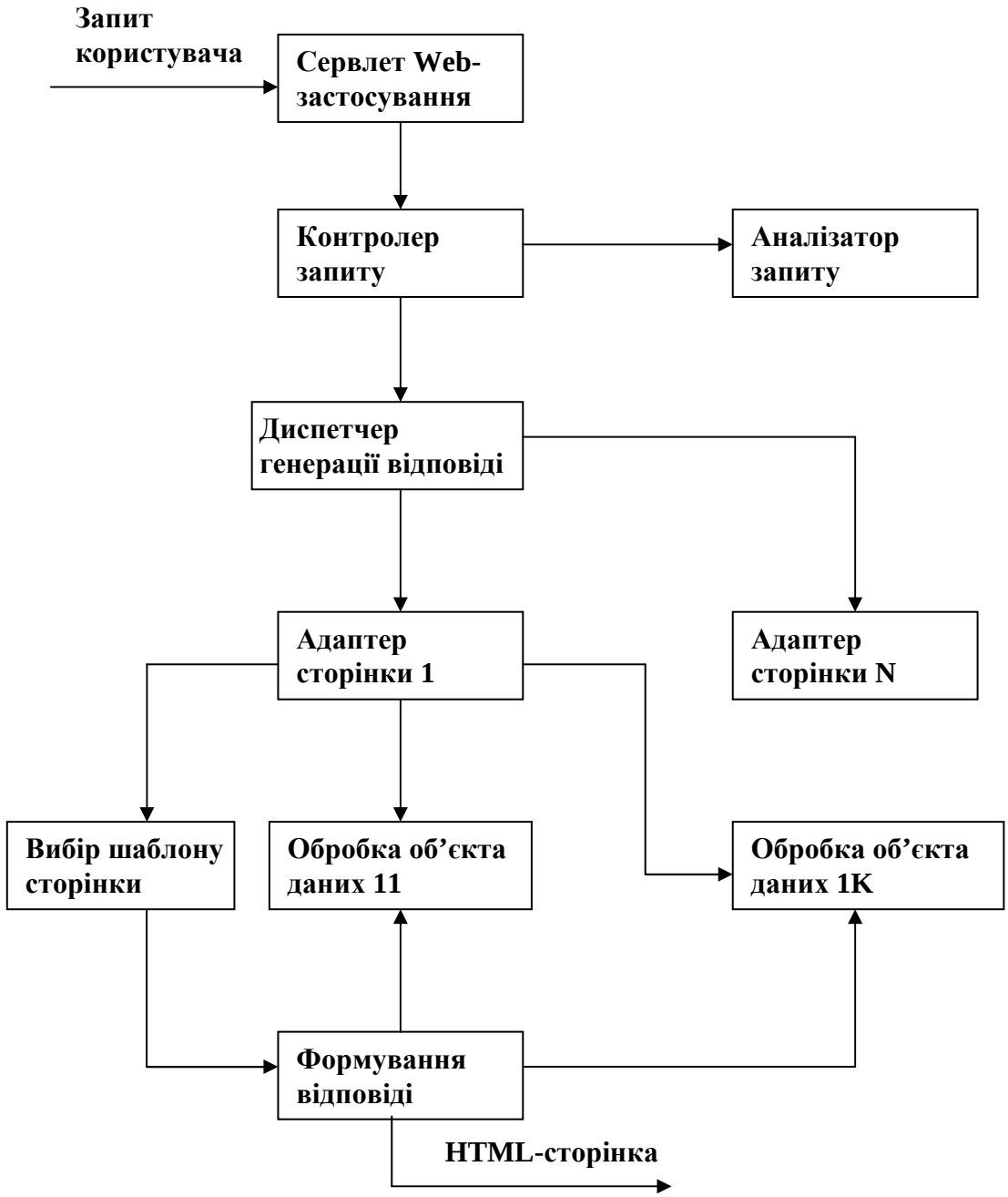

Рис. 2. Схематична структура застосування та процесу обробки даних

Висновки

Проведено аналіз процесів створення Web-застосувань на основі концепцій, моделей та методів компонентного програмування. Головна мета полягала в розробці широкого переліку типових рішень щодо визначення типових структур даних та об'єктів, операцій та функцій, процесів обробки запитів та генерації відповідей. Ці типові рішення отримані внаслідок системного підходу, який послідовно включає розгляд концепцій застосування компонентного підходу, побудову типової архітектури Web-застосувань, визначення типових структур даних для клієнтських запитів та серверних відповідей, проектування об'єктів та компонентів для реалізації типових операцій.

Запропоновані типові рішення склали концептуальну основу нового підходу до проектування та реалізації Web-застосувань.

До головних результатів цього підходу слід віднести:

– створення компонентної моделі для проектування Web-застосувань;

– розробку моделі опису окремих частин та елементів, типових структур даних;

– визначення функціональних властивостей та характеристик типових компонентів Web-застосувань;

– визначення методів та правил побудови структур клієнтів та серверів застосувань на основі механізмів компонентного програмування.

Загалом, новий підхід забезпечує більший рівень впорядкованості процесів розробки Web-застосувань, дозволяє отримати більш якісні характеристики таких систем, спрощує їх підтримку та експлуатацію.

Типові рішення були використані для практичної розробки Web-застосування. Результати цієї розробки підтвердили практичну доцільність та очікувані переваги компонентного підходу, а самий кінцевий програмний продукт був успішно впроваджений.

1. *Секреты* создания интрасетей. — СПб.: Питер, 1998. — 592 с.
2. *Брогден Б., Минник К.* Электронный магазин на Java и XML. — СПб.: Питер, 2002. — 400 с.
3. *Stal M.* Web Services: Beyond Component-based Computing // Communications of the ACM. — Vol. 45, ¹ 10. — October 2002. — PP. 71-76.
4. *Хабибуллин И. Ш.* Создание распределенных приложений на Java 2. — СПб.: БХВ-Петербург, 2002. — 704 с.
5. *Цимбал А.А, Аншина М.Л.* Технологии создания распределенных систем. Для профессионалов. — СПб.: Питер, 2003. — 576 с.
6. *Том Армстронг.* ActiveX: создание Web-приложений. — К.: Изд. группа BHV, 1998. — 592 с.
7. *Грищенко В.Н.* Особливості компонентно-оріентированної розробки програмного забезпечення // Пробл. программирования. — 2001. — № 3-4. — С. 75-92.
8. *Эммерих В.* Конструирование распределенных объектов. Методы и средства программирования интероперабельных объектов в архитектурах OMG/CORBA, Microsoft/COM и Java/RMI. — М.: Мир, 2002. — 510 с.
9. *Грищенко В.Н.* Систематизований підхід до визначення програмних компонентів // Пробл. программирования. — 2001. — ¹ 3-4. — С. 23-30.
10. *Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж.* Приемы объектно-ориентированного проектирования. Паттерны проектирования. — СПб: Питер, 2001. — 368 с.
11. *Холл М.* Сервлеты и JavaServer Pages. — СПб.: Питер, 2001. — 496 с.
12. *Кэй М.* XSLT. Справочник программиста. — Спб.: Символ-Плюс, 2002. — 1016 с.

Отримано ***07.09.04***

Про автора

Грищенко Володимир Миколайович
канд. фіз.-мат. наук

Місце роботи автора:
Інститут програмних систем НАН України,
м. Київ
Тел. (044) 266 3098