УДК 004.89:378.4(477-25) НаУКМА

ГЛИБОВЕЦЬ М.М., ІВАЩЕНКО С.А., КРУСЬ О.О., *Національний університет "Києво-Могилянська академія"*

Глибовець Микола Миколайович – к.т.н., доцент, декан факультету комп'ютерних наук НаУКМА.

РОЗРОБКА СИСТЕМИ УПРАВЛІННЯ НАВЧАЛЬНОГО ЗАКЛАДУ НА ПРИКЛАДІ НаУКМА

В роботі описано прототип типової автоматизованої системи управління навчальним закладом (АСУНЗ), що забезпечує виконання стандартних функцій керування навчальним процесом та допоміжними структурними підрозділами. Однією з задач досліджень була розробка архітектурного рішення і реалізація платформи, які б забезпечили можливість ефективного виконання довільної функціональності згідно з вимогами, що будуть розроблятися під час поетапної автоматизації навчального закладу.

The prototype of a typical automated educational establishment control system is described in this work. The system provides with a set of standard functions for controlling educational process and auxiliary structural subdivisions. One of the tasks of the research was to develop the architecture and to realize the platform, which would provide with the possibility of effective implementation of arbitrary functionality according to the requirements, which will be made during stage-by-stage automation of educational establishment.

Мотивація досліджень

Метою науково-дослідної роботи було створення прототипу типової автоматизованої системи управління навчальним закладом (АСУНЗ), що забезпечує виконання стандартних функцій керування навчальним процесом та допоміжними структурними підрозділами: створення внутрішньої інформаційної мережі університету (Intranet), інформаційних порталів факультетів та основних структурних підрозділів університету з забезпеченням прозорого видаленого доступу до інформаційно-довідкових та навчальних матеріалів засобами Інтернету, порталів навчально-методичного відділу та відділу кадрів з забезпеченням "дружнього" інтерфейсу, що має можливість віддаленого доступу, який забезпечить зручне ведення облікової інформації, складання розкладів занять, розрахунки навантаження викладачів, запис студентів на вибіркові курси, ведення та облік оцінок студентів за всі види навчальних робіт, документообіг, пошукові послуги тощо.

Проблема є актуальною, оскільки її розв'язання забезпечить: значну економію часу на управлінські рішення, чіткість керування, інформованість керівництва, реальне підґрунтя для прийняття рішень (з можливістю використання автоматизованої експертної системи по виробленню таких рішень).

Однією з задач досліджень була розробка архітектурного рішення і реалізація платформи, які б забезпечили можливість ефективного виконання довільної

функціональності згідно з вимогами, що будуть розроблятися під час поетапної автоматизації навчального закладу.

Вибір технології

Вибір архітектури та технології розробки системи базувався на їх можливості забезпечити гнучку розробку системи протягом великого періоду часу багатьма незалежними розробниками. Обрана компонентно-орієнтована технологія забезпечує розбиття системи на незалежні модулі та гнучке зв'язування модулів в одну систему. Більш того, при необхідності внесення змін до будь-якого модуля у майбутньому, кожен компонент може бути замінений на модифіковану версію без перекомпіляції всього коду. Такий підхід не тільки покращує параметри підтримки коду, пришвидшує розробку системи, організує вихідний код, а й надає можливість дуже чітко протестувати системні модулі, що підвищує стабільність всієї системи.

Основним підходом при побудові компонентної структури є так звана концепція Inversion of Control (IoC) [1] Основна ідея IoC характеризується висловом: «Не клич мене, я покличу тебе». Тобто відповідальність за виклик будь-якого компонента в системі лягає не на самі компоненти, а на каркас, в якому вони поєднані. Таким чином, компоненти не повинні «знати» про інші компоненти для своєї роботи, а також, як їх знайти та викликати. Каркас, що реалізує IoC контейнер, сам визначає, який компонент потребує якого і надає його під час виконання.

При розробці АСУНЗ був використаний каркас Spring Framework [2], який оснований на підході IoC і використовує вставку через спеціальні методи класу – сеттери, через які клас отримує конкретні дані ззовні. Spring реалізує IoC контейнер, який використовує XML файли для зберігання конфігурації кожного компонента (або інші методи зберігання) і за допомогою якої виставляє (injects) необхідні параметри компонентам.

Використаний підхід дозволяє будувати компоненти з чітко визначеним інтерфейсом, які не залежать один від одного, а їх поєднання в одне ціле відбувається за допомогою окремої конфігурації. Слід зазначити, що компоненти не залежать від Spring каркасу, а сам каркас виконує службові функції, наприклад читає конфігураційні файли та виставляє необхідні значення.

Використання Spring каркасу дає такі переваги: надає «легкий» контейнер J2EE застосуванням, використовує зв'язування компонентів за інтерфейсами, не зобов'язує застосування використовувати Spring і залежати від нього, підвищує можливість якісного тестування кінцевого продукту за допомогою unit тестування. Слід зазначити, що Spring не реалізує і не намагається реалізувати конкретні підкаркаси, наприклад роботи з базами даних, забезпечення транзакцій, ведення журналу – він просто надає методи для їх зв'язування, отже не лімітує розробників у використанні найбільш зручних і корисних для проекту технологій.

Архітектура

Основною метафорою при розробці архітектури було розбиття роботи навчального закладу на цикли документообігу. Розроблена архітектура відображає реальні цикли документообігу у навчальному закладі і продовжує їх на проміжках, які можна автоматизувати. Основою для обробки є віртуальні об'єкти моделі даних, які наближено відображають реальні об'єкти. Наближення вибрано таким чином, щоб об'єкти були якомога простішими для системи, забезпечивши збереження їх інформаційних характеристик, які мають бути враховані при автоматизації.

Для можливості роботи з базовими об'єктами системи узгоджено і цілісно, в архітектурі використовується метафора документу. Документ є підсистемою реально існуючої структури даних, в межах якої можуть проводитися зміни. Кожен документ виділяється таким чином, щоб зміни в структурі об'єктів, які входять в його склад, були атомарними, тобто не впливали на всю структуру в цілому. Документ також є об'єктом.

Кожен документ передається між прошарками системи і для кожного прошарку достатньо працювати з документом, щоб забезпечити конкретні кроки будь-якого автоматизованого процесу. Кожному прошарку відома структура даних системи і кожен прошарок може модифікувати документ в межах цієї структури.

Будь-який запит користувача виконується як проведення певної операції над документом. Можливі дії над документом визначаються циклами і підциклами документообігу (workflow cycles). Кожен цикл документообігу виконується в транзакції і трасується, поки не завершиться. Кожен серверний виклик передбачає те, що виклик проходить в якомусь конкретному циклі, який знаходиться в конкретному стані.

Зміни до інформації в системі здійснюються в основному через відповідні документи. Якщо необхідно зробити зміни до системи, наприклад змінити прізвище студента, потрібно відкрити документ, який відповідає реєстраційній картці студента і внести необхідні зміни. Модель захисту чітко відслідковує, які операції користувач може здійснювати над документом і що потрібно для того, щоб документ міг внести зміни в діючу систему. Наприклад, якщо для зміни стану будь-якого бізнес-об'єкту потрібні додаткові санкції, то документ відповідним чином обробляється перед тим, як він змінить стан системи. Система записує зміни, які потім зберігаються у сховищі даних.

Система складається з таких основних прошарків: інтерфейс користувача (показує певну частину стану системи користувачу, обробляє дії користувача і фіксує їх в змінених документах, отримує документи від моделі і передає змінені документи в модель для обробки), модель (формує базову структуру даних, обробляє модифіковані документи згідно з бізнес-правилами, отримує дані від прошарку даних і зберігає дані через нього), прошарок обробки даних (забезпечує правильне і узгоджене зберігання даних та повернення даних на запити)

Документ обробляється одним з трьох можливих способів: створюється, зберігається, відтворюється. Кожна операція реалізується окремо для кожного типу контенту документа. Для обробки кожного документу часто потрібно знати не лише те, що знаходиться в документі (часто документи можуть охоплювати досить великі структури даних, навіть рекурсивно пов'язані в потенційно нескінченні ланцюги структур), а те, що саме з усього контенту документу змінено (або потрібно створити). Для цього існують фільтри, які передаються операціям поруч з документами.

Фільтр є структурою функціональних об'єктів, що накладається на контент (можливий контент) документу для формування обмеженого контенту. При застосуванні в операціях відтворення документа фільтр вказує яку частину контенту з усього доступного для документа контенту очікують інші прошарки для обробки.

Нестандартні архітектурні рішення

Використовується механізм фіксування даних в сталих документах. Окрім динамічної структури даних в базі, контролююча роль може зберегти певну частину даних в окремому документі (зафіксувати), також поряд зі стандартним збереженням, фільтрування за прикладом.

Для заповнення даних в складні ієрархічні структури використовуються ієрархічні форми з аналогічною структурою компонентів (приблизно кожен компонент відповідає певному типу даних в структурі). Основною була вибрана стратегія централізованого

доступу/модифікації даних в структурі. Всі компоненти інтерфейсу забезпечують лише інформацію про шлях до певного параметру в структурі і дозволяють діставати або змінювати параметр за цим шляхом. Базовий компонент повинен обробляти задані шляхи і виконувати операції зі структурою об'єктів. Це дозволяє гнучко працювати зі структурою даних в будь-який спосіб.

В системі ведеться запис виконаних операцій і запис змін даних.

Реалізація

Загальна модель системи зображена на рисунку 1. Основним каркасом в реалізації описаної архітектури є Spring Framework. Для реалізації конкретних прошарків або функціональних частин були застосовані додаткові каркаси.

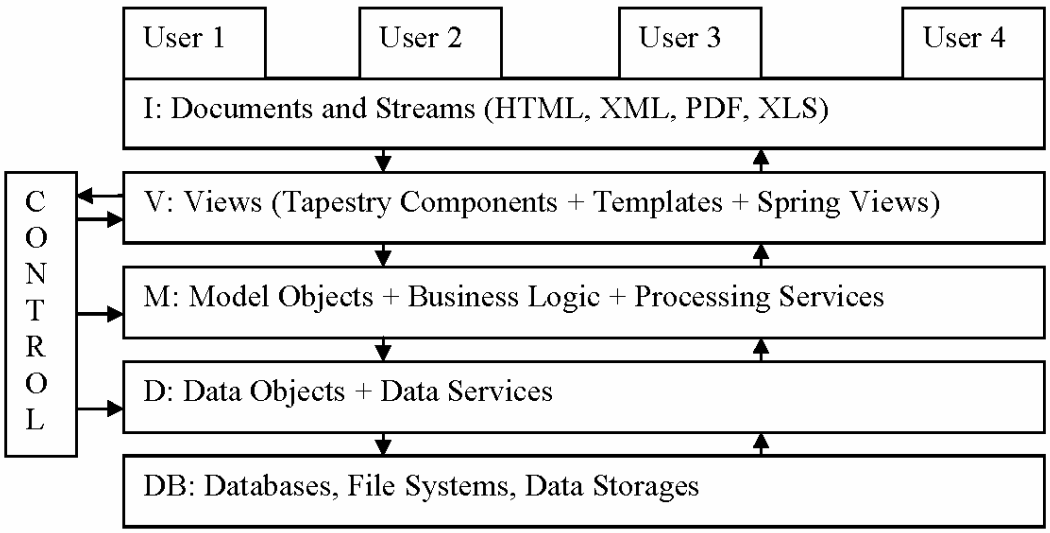

Рис. 1. Загальна модель системи.

Одним з основних прошарків є прошарок роботи з даними. При виборі технології реалізації прошарку, велику увагу було приділено незалежності від системи, де дані зберігаються, можливості чітко співставляти бізнес-об'єкти системи з даними і швидкість роботи. На момент вибору технології, найкращою системою підтримки зберігання об'єктів і їх зв'язків, а також пошуку таких об'єктів став Hibernate [3].

Як вже було відмічено, Spring Framework не реалізує конкретні засоби роботи з даними, але дозволяє використовувати будь-яку реалізацію, якою в нашому конкретному випадку є Hibernate. Таким чином, об'єкти, що зберігаються (persistent), поєднуються за допомогою патерну DI, а це дає змогу наприклад конфігурувати такі об'єкти через зовнішні конфігураційні файли, зберігати SQL/HQL запити в зовнішніх конфігурація і т.п. Таким чином об'єкти прошарку даних є повністю незалежними від інших об'єктів, проте легко з ними пов'язуються.

При розробці архітектури, використовувався патерн MVC (Model-View-Control), що дозволило досягнути незалежність від представлення будь-яких даних. Так, одні й ті самі дані можуть бути відображені HTML сторінкою, PDF документом, або SVG діаграмою.

Основною концепцією при побудові інтерфейсу користувача було створення тонкого клієнту – веб застосування. Архітектура підтримує і інші види інтерфейсів (наприклад повноцінний графічний інтерфейс або інтерфейс для мобільних пристроїв) і можуть бути використані поруч зі стандартним веб інтерфейсом у разі необхідності.

При розробці тонкого клієнту був вибраний каркас Tapestry [4], основною відмінністю якого у порівнянні зі схожими розробками (JSP, Velocity) є чітка компонентна структура і використання параметрів JavaBeans для задання конфігурації веб-компонентів. Це дає низку переваг: чітке розділення представлення від коду, всі компоненти в Tapestry можуть повторно використовуватися в будь-якому місці веб застосування; підвищення можливості якісного тестування компонент представлення, що в умовах інших каркасів майже недосяжно; додаткові переваги у захисті, а також можливість роздільної роботи команди програмістів та технічних дизайнерів.

Використання JavaBeans і компонентного підходу, як основного в Tapestry, дуже добре інтегрується в каркас IoC Spring Framework, а отже ми отримуємо додаткові переваги від їх поєднання. Як і з іншими компонентами, веб-компоненти Tapestry є незалежними і конфігуруються окремими конфігураціями, які потім виставляються контейнером Spring.

Використавши Spring в якості основного каркасу системи, що реалізує принцип IoC, ми змогли природно зв'язати додаткові каркаси роботи з даними, а також каркаси представлення. Як видно з рис.1. різні прошарки системи передають інформацію в обох напрямах через відкриті інтерфейси, виділяється чіткий прошарок бізнес-логіки і презентаційний шар ніколи напряму не працює з прошарком даних. Слід зазначити, що всі прошарки даних працюють уніфіковано, так як взаємодія проходить через визначений інтерфейс документу (Document), проте кожен прошарок обробляє тип контенту документа згідно власних зобов'язань та прав.

Така уніфікована робота практично не застосовується в інших подібних системах АСУНЗ. Зазвичай шари систем працюють неуніфіковано, шар інтерфейсу викликає різні методи сервісів, а сервіси викликають потрібні їм методи шару роботи з даними. А такий архітектурний підхід знижує контроль над виконанням операцій, призводить до розсіювання подібних функцій по багатьох класах і збільшує можливість помилок при обробці даних в різних контекстах.

Новизна розробленої архітектури полягає у тому, що поточний контекст задається типом контенту документа. Такий підхід забезпечує виконання важливої вимоги – робота з даними завжди відбувається у відповідності до контексту, в якому вони збираються, обробляються і зберігаються.

Нефункціональні та інші вимоги

При побудові архітектури, а також реалізації системи, велику увагу було приділено нефункціональним вимогам, архітектурні рішення для підтримки яких є ще однією перевагою АСУНЗ від схожих розробок.

В системі реалізована модель захисту, що базується на ролі користувача. Модель захисту функціонально розширює модель захисту платформи Java. Метод аутентифікації, що використовується по замовчанню – логін користувача і пароль, система легко підтримує і інші методи, наприклад використання сертифікатів (Kerberos, X.509). Сесія підтримується за рахунок захищених cookie.

На рівні доступу до бази даних, система захищена від прямого доступу до даних (доступ здійснюється тільки через проміжний шар). Застосовано і не прямий захист – використання журналу внесення змін у систему і протоколювання всіх без винятку операцій в системі.

Побудована на компонентному підході із застосуванням принципу IoC система без принципових обмежень може бути розширена до рівня автоматизації будь-якого навчального закладу, або будь-якої установи, де документообіг відіграє важливу роль, без обмеження функціональності поточних доробок.

Структура бази даних та структура класів повністю відповідає реальним об'єктам світу і не вносить суттєвих обмежень на майбутню розширюваність системи. Основні ймовірні ускладнення системи можуть скоріше привести до додавання нових структур і зміни ролі існуючих структур, аніж до їх повної заміни і повторної розробки.

Підхід до обробки інформації в системі (через цикли документообігу) є стандартним для будь-якої системи документообігу, а тому система пристосована до розширення для підтримки більшості адміністративних функцій.

Застосування гнучкої архітектури дозволяє розподіляти частини системи, будуючи розподілене застосування, що особливо важливо при великій кількості користувачів і інформації, що зберігається в системі. В такому випадку вдається вирішити проблеми піків навантаження, ввести пріоритети в обробці інформації та забезпечити безперервне функціонування системи. Опис вже існуючих інтерфейсів в небінарному форматі, може відкрити доступ до їх використання будь-якій іншій системі, використовуючи наприклад технологію веб сервісів.

Поточну реалізацію системи було протестовано в поточному варіанті на роботи з кількістю інформації, що в 10 раз перевищує заплановану завантаженість. При поточному режимі користування системою, вона може працювати на одному сервері з задовільним часом відгуку.

Використання Spring Framework, чітке виділення інтерфейсів та підхід до уніфікованого інтерфейсу Document дозволяє покрити практично всю критичну функціональність unit-тестами, за допомогою JUnit, або подібних систем. Крім того, використовуючи Tapestry каркас, ми досягли високого рівня можливості тестувати компоненти презентаційного рівня, що в комплексі дає можливість повністю проводити підхід паралельного тестування (test-driven approach) [5] при розробці системи.

Такий підхід не тільки забезпечує стабільність системи в цілому, а й полегшує її розвиток і підтримку у майбутньому.

Висновки і подальші дослідження

В результаті проведення науково-дослідної роботи, була розроблена архітектура, що включає останні досягнення у побудові сучасних застосувань. Для її побудови були використані такі стандартні рішення: використання уніфікованого інтерфейсу для передачі даних між прошарками системи; визначення контексту документа згідно з його вмістом та обов'язками прошарку, що його обробляє. Розширене використання патерну IoC дозволило створити гнучку, стабільну та зручну платформу для розробки АСУНЗ. Прикладом реалізації архітектури стала розробка системи автоматизованого управління Магістеріуму НаУКМА. Наразі, закінчується робота по її перенесенню на весь університет.

Література

1. Martin Fowler, Inversion of Control and the Dependency Injection pattern (http://www.martinfowler.com/articles/injection.html)
2. Spring Framework, (http://www.springframework.org/)
3. Hibernate – Relational Persistence For Idiomatic Java (http://www.hibernate.org/)
4. Tapestry (http://jakarta.apache.org/tapestry/)
5. Scott W. Ambler, Introduction to Test Driven Development (http://www.agiledata.org/essays/tdd.html)
6. Медвідь С.О., Комбінований компетентний паралельний генетичний алгоритм та його застосування для задачі побудови розкладів, "Проблеми програмування", спеціальний випуск, №2-3, 2004 – 261