УДК 681.3

ФОРМУВАННЯ ЛОГІКИ ВНУТРІШНЬОЇ МІЖРІВНЕВОЇ ВЗАЄМОДІЇ В БАГАТОЛАНКОВІЙ СИСТЕМІ

Ю.В. Бойко, С.Д. Погорілий, О.В. Коваленко

Київський національний університет імені Тараса Шевченка,
01033, Київ, вул. Володимирська, 64, б.
Тел.: (044) 206 2615, (044) 526 0522, (044) 526 1214.
e-mail: kovalenko.oleksandr@gmail.com, sdp@rpd.univ.kiev.ua, boyko@univ.kiev.ua

Розглянуто інноваційну методологію побудови розподіленої багатоланкової системи через організацію зв'язування рівнів та проаналізовано її архітектурні особливості. Запропоновано використання дескрипторів для зв'язування даних з елементами багатопрофільного інтерфейсу користувача, опис якого робиться мовою XML. Викладено рекомендації щодо впровадження створеної системи на платформі .Net. Виконано аналіз переваг методу.

Innovative methodology of the distributed multi-tiered system development was studied by functional decomposition. Architectural features were researched. Descriptors using for data binding with multi-profile user interface elements was suggested. XML language was used for the interface descriptions. Recommendations for the developed system implementation on the .Net platform were presented. Technique advantages of such approach over existing one were analyzed.

Вступ

Сучасне становище в галузі формування нової інфраструктури виробництва, господарства, торгівлі в Україні вимагає створення потужних засобів автоматизації, використання комп'ютерних технологій, необхідного програмного забезпечення. Наприклад, мережа супермаркетів потребує відмовостійкої розподіленої системи. Вона, з одного боку, має працювати з торговим обладнанням, а з іншого, підтримувати значну кількість профілів користувачів, які виконують різні функції. До того ж, мають існувати можливості централізованого формування оперативної управлінської та бухгалтерської звітності. Зазначимо, що актуальність реалізації такого класу систем є вкрай важливою в Україні.

Створення систем такого класу потребує функціональної декомпозиції загального проекту на етапі практичної реалізації. У зв'язку з цим, метою роботи є проектування класів для організації ефективної взаємодії всередині багатоланкової системи.

Актуальність проблеми формування логіки внутрішньої міжрівневої взаємодії продиктована задачами, що виникають при розробці програмного забезпечення, в якому будується взаємодія застосування з джерелом даних (СКБД), та присутній графічний рівень клієнтського застосування.

У застосуванні слід організувати двостороннє зв'язування даних [1]: дані, що находять з СКБД мають взаємодіяти з елементами інтерфейсу, а інформацію, яку отримано з елементів інтерфейсу, слід передавати для збереження до СКБД.

При не оптимізованому підході для реалізації задачі в код доводиться вводити сукупність однотипних фрагментів – процедурних засобів синхронізації [2]. Як наслідок, виникає небезпека дублювання коду. Сам код стає важко підтримувати та вносити до нього зміни, втрачається його структурованість та впорядкованість.

Для вирішення задач використовують:

- бібліотеки доступу до даних з підтримкою прозорої зміни джерела даних [3]. Проблема інтерпретування вирішується лише частково;
- сторонні об'єктні реляційні моделі (Object Relation Model, ORM) створюють низку об'єктів системи з даними в реляційні моделі. Прикладом такої бібліотеки є Hibernate для мови J2SE/J2EE [4]. Суттєвим їх недоліком є те, що вони не враховують специфіку роботи системи й, як наслідок, призводять до виникнення додаткових проблем з моделюванням прикладної задачі, через недостатню виразність моделей або проблеми швидкодії;
- в багатьох реалізаціях окремо взята ORM модель добре працює для роботи з одиничними об'єктами та погано працює, коли потрібно працювати з великими колекціями об'єктів. Ще одною проблемою може бути те, що ORM прив'язується лише до реальних таблиць, або до представлень бази даних, і не може бути використана із збереженими процедурами, які дозволятимуть зменшити об'єми даних, що передаються мережею, та час виконання операцій обробки. Як приклад невдалої моделі ORM можна навести EntityBeans [5], як частину специфікації EJB архітектури програмного комплексу [6]. Її використання є доцільним при формуванні простих зв'язків, а при реалізації складних випадків на наборах - доводиться застосовувати повністю ручне кодування й спостерігати проблеми швидкодії.

З урахуванням недоліків існуючих технологій виникає необхідність створення універсального комплексного підходу. В роботі розв'язання задачі проведено з допомогою методу функціональної декомпозиції, шляхом розбиття системи на рівні за функціональним призначенням [7].

Декомпозиція

До більшості архітектур програмних комплексів можна застосувати поділ на логічні рівні, зокрема з використанням шаблону проектування «Модель-Представлення-Контролер» (Model-View-Controller, MVC).

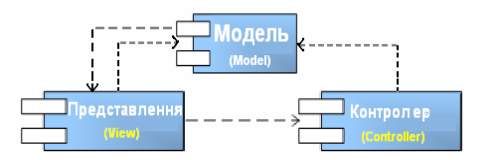

Рис. 1. MVC модель

- Model – модель. Рівень. що підтримує роботу з даними;
- View – вид або представлення. Рівень, що відповідальний за формування візуального представлення даних та інтерфейс користувача;
- Controller – контролер. Рівень, що відповідає за функціональну та бізнес-логіку, взаємозв'язок моделі та представлення.

Задача зводиться до організації взаємодії між рівнями View та Controller за допомогою допоміжного рівня Model (рис. 1). Вона має відповідати таким вимогам:

- бути універсальною (підтримувати різні типи даних);
- бути простою в застосуванні (прив'язка елементів до даних має бути простою і не вимагати написання допоміжного коду);
- підтримувати різні за обсягом набори даних;
- створювати можливість для віддаленої роботи з даними (мається на увазі технологія, при якій користувач завантажує дані з сервера, модифікує та здійснює певні операції з ними локально, а потім відсилає зміни на сервер).

Рис. 2 віддзеркалює розбиття на рівні та підрівні і їх взаємодію, випливаючи з задач обробки даних та їх передачі між рівнями. Приклад наведено для системи автоматизації на основі архітектури «Товстого клієнта» (Thick Client) [3, 8]. На відміну від архітектури «Тонкого клієнта» (Thin Client), у даній моделі клієнтська частина навантажена логікою і самостійно виконує частину обробки даних. Коли на робочих станціях доводиться працювати з нетиповим обладнанням, мати потужні засоби для обробки й представлення інформації та використовувати базу даних для її зберігання, то можна дійти висновку, що для таких задач оптимальна модель «Товстого клієнта».

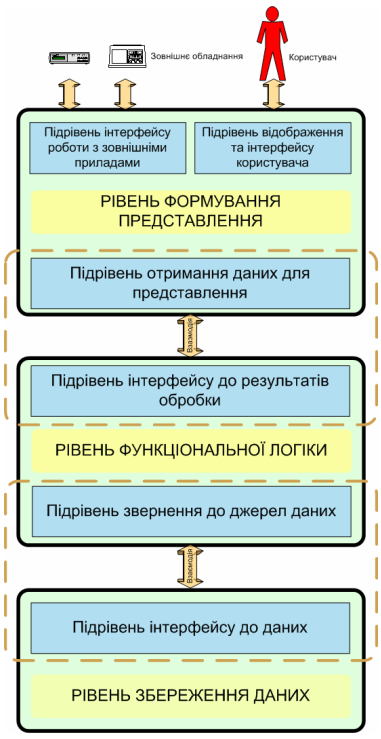

Рис. 2. Загальна схема взаємодії архітектурних ланок системи

Побудова контролеру прикладної логіки, орієнтованого на доступ до даних

Для роботи з даними розроблюються класи оболонки, що братимуть на себе задачі по завантаженню даних із бази, збереженню змін у базі та обробку помилок. Такі об'єкти називаються об'єктами-сутностями. Маніпулювання об'єктами-сутностями може відбуватися з кожним об'єктом окремо, або відразу з цілим набором. Через це вводиться два типи об'єктів оболонок: об'єкт-атомарна сутність та об'єкт-колекція сутностей.

Для реалізації кожної з них необхідно розробити відповідний абстрактний клас та створювати реальні об'єкти-сутності як їх нащадки. При реалізації завантаження, збереження та видалення даних використовуються уніфіковані методи, які генерують запити до джерела даних, базуючись на використанні первинних ключів (primary key).

Для зв'язування поля, чи властивості об'єкта-сутності з відповідним полем бази даних (що повертається з бази та відсилається до бази за допомогою SQL-запитів), слід зазначити дане поле за допомогою дескриптору (такими в .Net є атрибути: «Attribute»), що встановлюється для поля або властивості класу.

Ієрархія класів у системі

Ієрархію класів системи зображено з погляду функціональної взаємодії у вигляді UML діаграми [9].

Систему розбито на 7 зв'язаних між собою основних компонентів:

- MainController;
- FormController;
- BussinessLogics;
- DataViewManager;
- View;
- DataEntities;
- DLLMamager.

Зазначимо, які базові класи будемо використовувати, як вони будуть розподілені за даними компонентами, та як вони будуть взаємодіяти. Крім того, зазначимо, які шаблони проектування варто для них використати.

MainController

- XFormsManager – власне реалізує головний контролер, будується за шаблоном Singleton;
- XApplication – допоміжний клас, що зберігає загальну системну інформацію, будується за шаблоном Singleton;
- XMessage – використовується для передачі повідомлень між окремими FormController'ами.

FormController

- XFormController – реалізує FormController.

BussinessLogics

- XActionController – базовий клас для реалізації прикладної логіки, при побудові використовується шаблон Template.

DataViewManager

- XEntitiesManager – використовується для реєстрації об'єктів-сутностей та надає засоби для синхронізації їх з вмістом елементів керування.

View

- XFormBuilder – будує форму на базі опису її у вигляді XML-документа, забезпечує кешування об'єктів форм для підвищення швидкодії, будується за шаблоном Singleton та Builder;
- XForm – реалізує кожну конкретну форму;
- XConfigCache – забезпечує кешування конфігураційних файлів.

DataEntities

- XEntity – базовий клас для атомарних об'єктів-сутностей;
- XEntityCollection – базовий для колекцій об'єктів-сутностей;
- XCachedEntityCollection – базовий для дуже великих колекцій об'єктів-сутностей.

DLLManager

- XDLLManager – забезпечує керування оновленням програмної системи, також використовується для ідентифікації типів.

Діаграму класів, що демонструє взаємодію між вищеописаними класами, показано на рис. 3. Варто відзначити, що дана діаграма є результатом не лише початкового проектування, а і реалізації системи, під час якої шляхом рефакторінгу віднаходилися найкращі зв'язки між елементами системи.

Організація опису багатопрофільного інтерфейсу користувача

При розробці складних систем, орієнтованих на підтримку великої кількості користувачів, однією з перших постає проблема адаптивності інтерфейсу кінцевого користувача та багатопрофільності застосування [7]. Складність інтерфейсу та вимоги щодо його функціональності в корпоративних системах неможливо задовольнити засобами HTML та споріднених технологій. Доводиться користуватись засобами, що надаються більш традиційною моделлю «Товстого клієнта». Ускладнення виникають через необхідність різним групам користувачів виконувати різні задачі.

Реалізацію опису багатовіконного інтерфейсу користувача виконаємо з допомогою технології XML. Опис кожного вікна будемо здійснювати в окремому файлі.

У файл «main.xml» винесемо опис головного вікна застосування. В ньому будуть розміщені посилання на дочірні файли з описом інших форм та вікон.

```
<screens>
  <screen id="Screen1" title="Довідник №1" maximize="false" width="748" height="465"
     file="reference.xml"/>
  <screen id="Screen2" title="Групи товару" maximize="false" width="340" height="130"
     file="good_group.xml preload="4" />
  …
</screens>
```

Опис елементів інтерфейсу неможливий без попередньої стандартизації. По-перше, встановлюються характеристики, що визначають елементи зовнішнього вигляду: положення, розмір тощо.

По-друге, вирішується проблема синхронізації елементів керування та стану об'єктів-сутностей предметної області. Для цього реалізуються можливості:

- зазначення в конфігураційних файлах об'єкта його властивостей, які потрібно зв'язувати з вмістом даного елемента керування;
- процедура синхронізації даних у напрямку від елементів керування до об'єктів даних перед виконанням будь-якої операції (реакція на дії користувача);
- процедура синхронізації даних у напрямку від об'єктів-даних до елементів керування після виконання будь-якої операції (реакції на подію у інтерфейсі користувача);
- реєструвати у прикладній логіці об'єкти-даних, які є доступними для синхронізації з елементами керування.

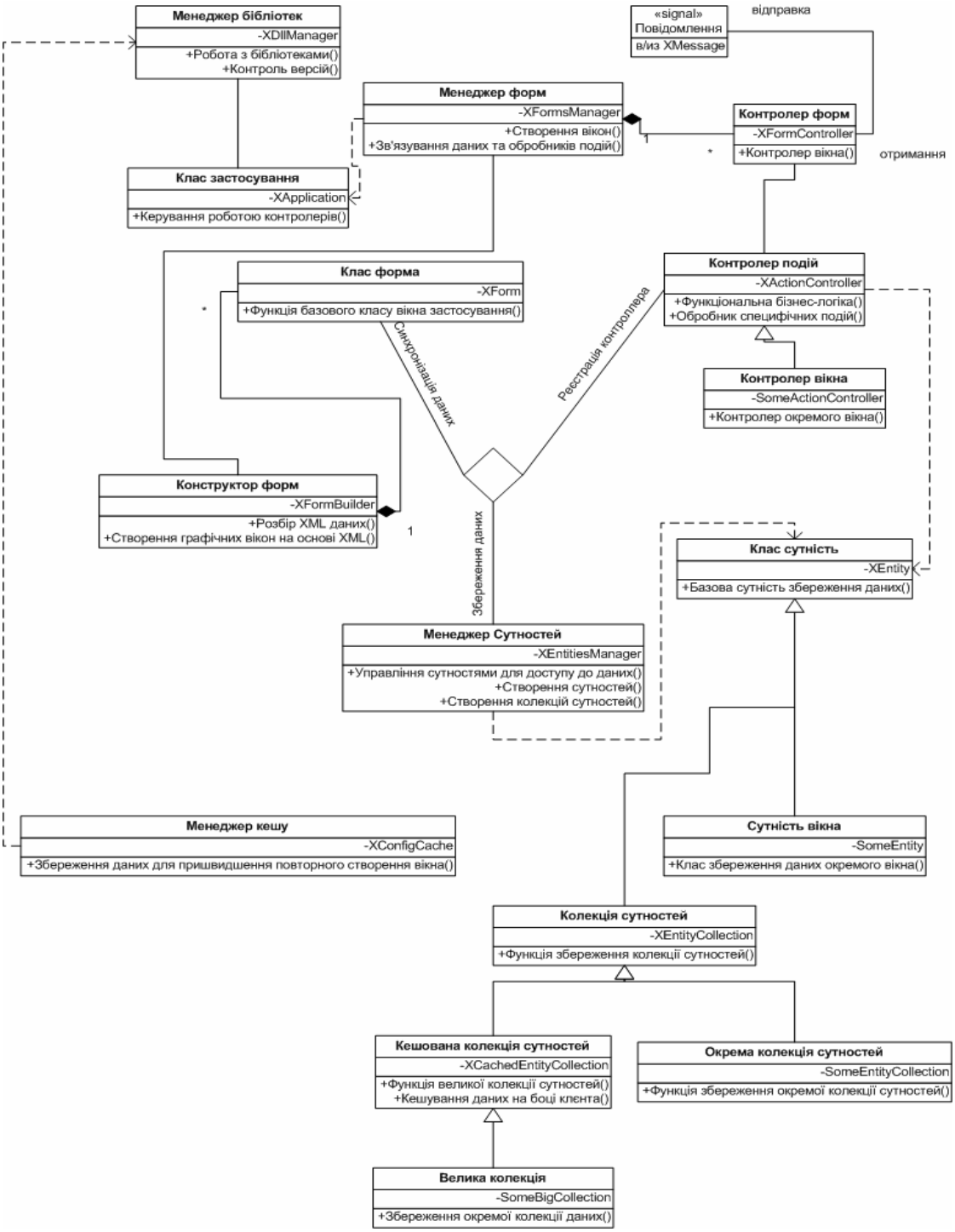

Рис. 3. Ієрархія функціонального відношення класів у системі

Всі перелічені властивості описуються в xml файлі за допомогою значень тегів елементів. Зв'язок елемента керування з об'єктом чи його властивістю зазначається у вигляді:

- **Object.Property –** якщо необхідно зв'язатися з полем об'єкта (Field) або властивістю об'єкта (Property);
- **Object** – якщо необхідно зв'язатися з цілим об'єктом**.**

В описі форми зазначаються також два обов'язкових посилання. По-перше, - це посилання на клас-контролер, що буде нащадком від загального базового класу-контролера й буде відповідати за реалізацію реакції на дії користувача. По-друге, – зазначення основного об'єкта-сутності для даної форми.

Для реалізації багатовіконного інтерфейсу слід додатково забезпечити ізольованість кожного набору: «форма»-«зареєстровані для неї об'єкти-сутності».

Використання технології дескрипторів (Attribute) для платформи .Net [10] надає можливість у простий спосіб прив'язати запити SQL до класів та поля, що ними повертаються, до їх властивостей (Properties).

Такий підхід дозволяє в режимі роботи створити низку рівня бази даних з елементами класу в режимі моделі DataTable. Для кожної з основних операцій зчитування, оновлення та видалення даних (select / update / delete) в атрибутах классу вказується свій запит.

```
[EntityCMD(XCommandType::Load, S"exec mp_get_StaffPerson")]
[EntityCMD(XCommandType::Update, S"exec mp_upd_StaffPerson")]
[EntityCMD(XCommandType::Insert, S"exec mp_add_StaffPerson")]
public __gc class StaffPersonEntity: public StaffPersonEntity{
…
}

[Mandatory("company_id", IsPrimaryKey="true")]
__property Int32 get_CompanyID(void) { return m_iCompanyID; }
__property void set_CompanyID(Int32 value) { m_iCompanyID = value; }
```

З допомогою команди «Load» на основі переданого значення первинного ключа (Primary Key) отримується набір даних поля, якого далі стають основою запитів для команд «Update» та «Delete». Набір параметрів для останніх генерується автоматично.

Процес завантаження застосування при даному підході розбивається на етапи:

- завантаження кореневих бібліотек;
- оновлення бібліотек бізнес-логіки;
- обробка головного конфігураційного файлу (main.xml);

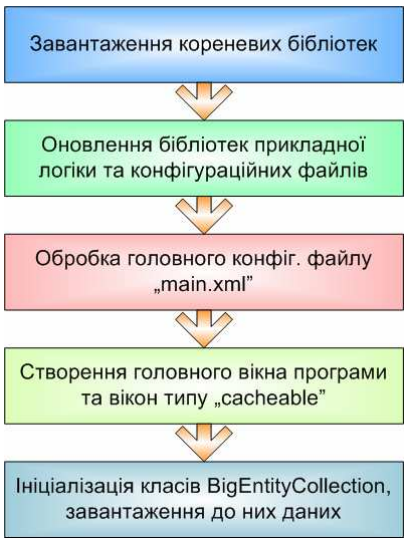

Рис. 4. Ініціалізація застосування

- динамічне створення головного вікна програми та вікон, позначених як «cacheable» (таке позначення примушує основний контролер створювати об'єкт форми при завантаженні застосування і при подальшій роботі гарантує швидке відкриття вікна);
- завантаження основних даних в об'єкти «EntityCollection» (таке завантаження раціонально застосовувати для подальшої роботи з великим масивом записів, до якого необхідно часто звертатись).

Створення окремого вікна об'єкта контролера окремого вікна утворює етапи:

- аналіз конфігураційного файла вікна (визначення об'єкта-контролера та основного об'єкта-сутності «Entity»);
- ініціалізація знайденого в XML описі класу – контролера;
- створення елементів:
  - ініціалізація основних властивостей (розмір, тощо);
  - прив'язування обробників подій (основних та додаткових);
- організація прив'язки властивостей контролера («properties»).

Оновлення клієнтського програмного забезпечення з боку клієнта є найбільш суттєвою проблемою двохрівневої моделі через те, що головна частина логіки, а отже, і програмного коду розміщується на комп'ютері користувача. Отже, при оновленні програмного забезпечення роботи слід виконати на всіх робочих станціях. До того ж, досить часто необхідно підтримувати декілька паралельних версій програми зі спільними частинами, наприклад, для різних підрозділів компанії необхідні різні інтерфейси для роботи з одними й тими самими даними. Так, крім оновлення, виникає ще й проблема керування паралельними версіями.

Для розв'язання проблем оновлення програмного забезпечення існують такі служби як Active Directory [11], Remote Installation Service [12] від компанії Microsoft. Але, в більшості випадків, ці підходи не виправдовують себе у внутрішньо-корпоративних системах саме через необхідність підтримки паралельних версій (особливо в продуктах компанії MicroSoft), або керуванням оновленням з клієнтських комп'ютерів (для OpenSource-сервісів), або через необхідність реалізації досить складних правил розповсюдження для підтримки корпоративної політики керування паралельними версіями.

Для створення оптимальної системи оновлень виконаємо розбиття системи на типи компонентів. Розроблювану систему можна розподілити на три частини, що можуть потребувати оновлення:

- **ядро системи** – це базові бібліотеки, що завантажуються при старті системи і змінюються досить рідко;
- **бізнес-логіка** – може змінюватися досить часто;
- **конфігураційні файли**, що використовуються для формування зовнішнього вигляду і можуть змінюватися досить часто.

Для полегшення керування конфігураційними файлами їх оновлення пропонується виконувати шляхом об'єднання груп конфігураційних файлів у пакет (наприклад, у допоміжну бібліотеку «Sattelite Assemblity» в .Net) відповідно до конкретної ролі користувача.

Для оновлення бібліотек та файлів, що контролюють роботу рівня функціональної логіки, важливе значення має той факт, що після завантаження такої бібліотеки вона може бути без проблем вивантажена, а отже, з'являється можливість завантаження бібліотеки без зупинки системи, перевірки її версії і, в разі необхідності, вивантаження поточної версії бібліотеки і завантаження більш нової.

Змістовність та працездатність ідей були перевірені та підтверджені на практиці. На основі викладених методів побудовано та успішно впроваджено систему комплексної автоматизації, яка задіяла автоматизацію майже тисячі робочих місць.

Впровадження системи

Запропонована в роботі функціональна декомпозиція дозволила створити програмний комплекс для автоматизації великої української мережі гіпермаркетів. Ядро системи реалізовано для операційної системи Windows на платформі Microsoft .Net, мови програмування: C++, C#. Серверна частина: Microsoft SQL Server.

Особливістю впровадження систем на основі розробленого ядра є відсутність необхідності виконання для кожного елемента користувача зв'язування із елементами класів, змінними в коді, писати процедури для синхронізації даних з базою даних. Всі ці функції виконує загальний контролер, використовуючи прості атрибути з опису вікна в xml-файлі. Створення вікон стало зручним завдяки візуальному конструктору, що в значній мірі схожий з типовими дизайнерами середовищ RAD (Rapid Application Development) [13]. Але основною відмінністю від типового процесу створення є одночасне співставлення елементів інтерфейсу з елементами бази даних. Клас, що відповідає за обробку подій під час роботи користувача з вікном, створюється на етапі виконання в процесі ініціалізації вікна.

Зазначені функціональні особливості дозволяють розробляти потужні системи високого класу за значно швидші терміни, за рахунок виключення необхідності написання дублюючого коду обробників подій, використання базового контролера для роботи з вікнами системи.

Використання нових компонентів можливе шляхом включення в проект нових бібліотек, додавання посилання на них та реєстрації їх в головному XML файлі. Подальше їх використання зводиться до зазначення в XML.

Надійність роботи системи перевірена дворічною експлуатацією системи в промисловому масштабі.

Висновки

Розроблено єдиний комплексний підхід до розв'язування задач, що постають при розробці систем автоматизації на основі клієнт-серверної архітектури. Проведена функціональна декомпозиція дозволила:

- запропонувати підхід до розв'язання задачі взаємодії рівнів усередині багатоланкової системи;
- побудувати функціональне відношення класів;
- створити ядро надійної, масштабованої розподіленої системи;
- розробити технологію опису інтерфейсів, що дозволяє використовувати простий інструментарій для побудови системи, зв'язування рівнів інтерфейсу користувача, логіки та рівня бази даних.

Системи автоматизації, що створені на основі результатів роботи, відрізняються простотою впровадження, короткими термінами розробки (в кілька разів в порівнянні з розробкою на основі стандартних засобів RAD), надійністю роботи.

1. *Kovalenko O.V., Boyko Y.V.* Effective design and development of complex automation systems on distributed multi-tier architecture. Kyiv : Proceedings of the 6th Int. Young Scientists Conference on Applied Physics. Taras Shevchenko National University of Kyiv, 2006.
2. *Стелтинг С., Маассен О.* Применение шаблонов Java. Библиотека профессионала. – М. : Издательский дом "Вильямс", 2002. – 576 с.
3. *Фаулер М.* Архитектура корпоративных программных приложений. – М. : Издательский дом «Вильямс», 2006. – 544 с.
4. *Hibernate*. http://en.wikipedia.org/wiki/Hibernate_(Java).
5. *EntityBean*. http://en.wikipedia.org/wiki/Entity_Bean.
6. *EJB*. http:// java.sun.com/products/ejb.
7. *Погорілий С.Д.* Програмне конструювання. - К. : ВПЦ «Київський університет», 2005. – 438 с.
8. *Таненбаум Е., Ван Стеен М.* Распределенные системы. Принципы и парадигмы. – Спб. : ЗАО Издательский дом «Питер», 2003. – 877 с.
9. *Якобсон А., Буч Г., Рамбо Дж.* Унифицированный процесс разработки программного обеспечения. - Спб.: Питер, 2002. – 496 с.
10. *Рихтер Дж.* Программирование на платформе Microsoft .NET Framework. - М. : Издательско-торговый дом "Русская редакция", 2002. – 512 с.
11. *Active* Directory. http://www.microsoft.com/windowsserver2003/technologies/directory/activedirectory/.
12. *Remote* Installation Services. http://technet.microsoft.com/library/Bb742378.aspx.
13. *Rapid* Application Development. http://en.wikipedia.org/wiki/Rapid_application_development.