Міністерство освіти й науки України
Донбаська державна машинобудівна академія

МЕТОДИЧНІ ВКАЗІВКИ

до самостійної роботи

по дисципліні

«ТЕХНОЛОГІЯ ПРОГРАМУВАННЯ СКЛАДНИХ СИСТЕМ»

(для студентів спеціальності 151
денної і заочної форми навчання)

Затверджено на засіданні
методичної ради ФАМІТ
Протокол № від

Краматорськ 2018

УДК 004.415.2

Методичні вказівки до самостійної роботи по дисципліні ”Технологія програмування складних систем” (для студентів спеціальності 151 денної і заочної форми навчання) /

З метою покращення розуміння складної методології процесу моделювання програмного забезпечення систем мовою моделювання UML розглядається спрощений варіант цієї методології – процес ICONIX. В методичних вказівках висвітлюються чотири основних етапи проектування об'єктно-орієнтованих моделей програмного забезпечення систем – моделювання предметної області й прецедентів, аналіз придатності моделі, розробка діаграм послідовності й діаграм класів. Висвітлюються характерні помилки студентів на всіх етапах моделювання. Наведені вправи по аналізу помилок та тематика індивідуальних завдань.

1 ОРГАНІЗАЦІЯ САМОСТІЙНОЇ РОБОТИ (ЗАГАЛЬНІ ПОЛОЖЕННЯ)

1.1 Ціль самостійної роботи

Ціль самостійної роботи – придбати вміння самостійно створювати моделі програм складних систем із застосуванням мови моделювання UML.

При цьому особлива увага повинна бути приділена пошуку відповідей на фундаментальні питання про систему:

- хто ставиться до користувачів системи (акторів) і що вони повинні зробити;
- що являють собою об'єкти «реального миру» (предметної області) і які між ними асоціації;
- які об'єкти беруть участь у кожному прецеденті;
- як об'єкти взаємодіють один з одним у кожному прецеденті;
- як урахувати обмеження, що накладаються режимом реального часу;
- як буде виглядати система на найнижчому рівні.

Відповіді на ці питання, особливо на перші чотири, потрібні при розробці будь-якої системи з використанням UML [1]. *Відсутність відповіді на кожне із сформульованих вище питань може створити необґрунтований ризик у створенні якісного програмного продукту.*

У багатьох випадках одержати ці відповіді можна за допомогою мінімальної підмножини UML. Така мінімальна підмножина визначена в процесі ICONIX. Освоєння методології процесу ICONIX, у якому представлена базова нотація UML, дозволить полегшити освоєння більше складних CASE-технологій програмування систем, які вивчаються в даній дисципліні і в яких використовується більш повна нотація UML.

Процес ICONIX демонструється класичним прикладом розробки програмної системи, наведеним Розенбергом Д. і Скоттом К. [2].

1.2 Зміст самостійної роботи (основні етапи процесу ICONIX)

На рисунку 1.1 представлена укрупнена схема процесу ICONIX. Він складається із ***двох частин***: зверху зображена ***динамічна*** модель, що описує **поводження**, а знизу – ***статична***, що описує **структуру**.

Шлях, по якому повинен йти розроблювач, можна представити як дорогу, позначену *віхами.* Наприкінці цієї дороги перебуває якісний програмний продукт. Основні етапи й пов'язані з ними віхи процесу ICONIX полягають у наступному.

Етап 1. Аналіз вимог:

1. Виявлення об'єктів предметної області, а також визначення відносин узагальнення й агрегування між ними.
2. Створення приблизного прототипу системи.
3. Виявлення прецедентів.
4. Групування прецедентів, подання груп пакетами.
5. Розподіл функціональних вимог між прецедентами й об'єктами.

Цей етап закінчується віхою 1 – розробкою вимог до програмної системи (рецензуванням вимог).

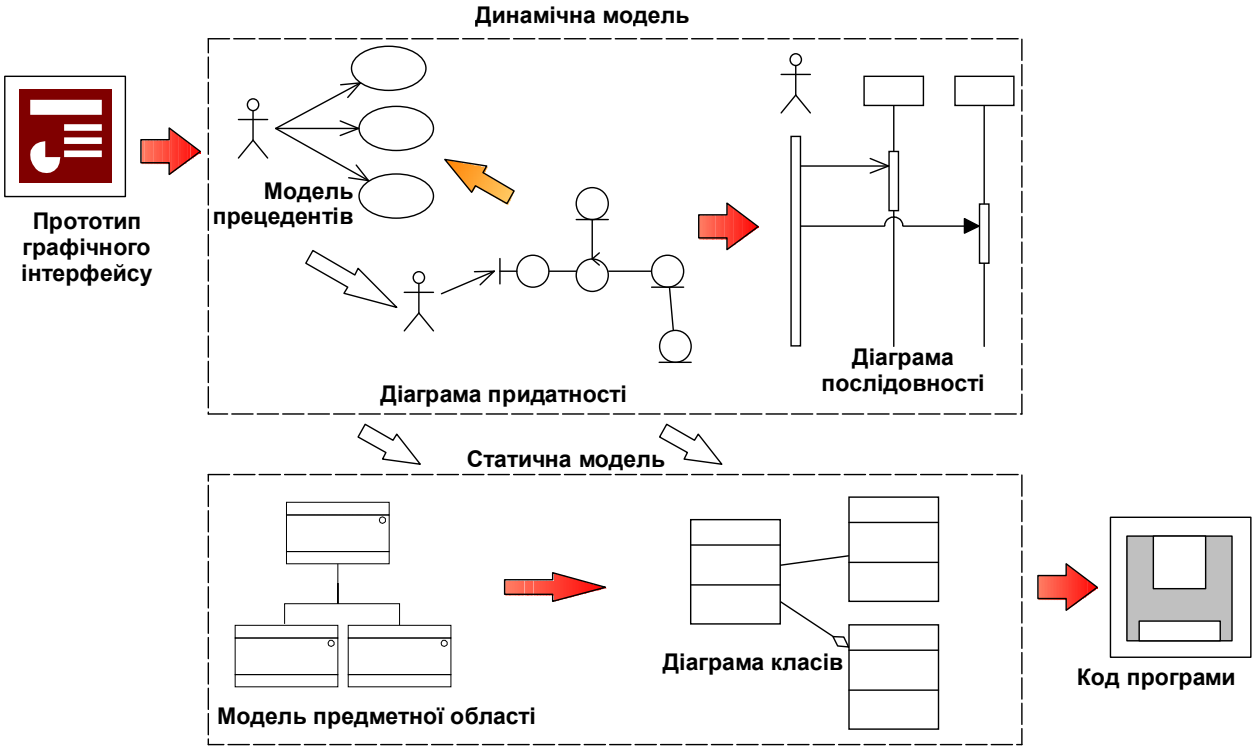

Рисунок 1.7 – Процес ICONIX у моделюванні із застосуванням UML

Етап 2. Попереднє проектування:

1. Складання текстового опису прецедентів – головної й альтернативної послідовностей.
2. Виконання аналізу придатності для кожного прецеденту.
3. Розробка й модифікація діаграми класів.

Цей етап закінчується віхою 2 – розробкою ескізного проекту (рецензуванням попереднього варіанту проекту).

Етап 3. Проектування:

1. Уточнення і розподіл поводження об'єктів при реалізації кожного прецеденту.
2. Розробка діаграми послідовності з урахуванням реального часу.
3. Завершення розробки статичної моделі із внесенням у неї детальної проектної інформації.

Цей етап завершується віхою 3 – рецензуванням остаточного проекту.

Етап 4. Реалізація:

1. Розробка (при необхідності) додаткових діаграм, наприклад, компонентів або розгортання.
2. Генерація коду.
3. Тестування програми.

Цей завершальний етап (віха 4) дозволяє отримати оформлений програмний продукт.

Самостійна робота передбачає розробку реальної предметної області, тобто *перші три етапи*. Як приклад використовується загальновідома студентам денного й заочного навчання предметна область за назвою «Книжковий Internet-магазин». У цій предметній області сконцентровані особливості, властиві багатьом проектам сучасного миру, пронизаного “Всесвітньою павутиною”. Використання цього прикладу [2] дозволяє краще зрозуміти всі етапи процесу моделювання.

Вимоги до програмного забезпечення Internet-магазина сформульовані в такий спосіб.

- прийом замовлень через Internet;
- захист рахунків за допомогою пароля;
- засоби пошуку в головному каталозі;
- застосування різних методів пошуку, у тому числі по автору, за назвою, по ISBN і по ключових словах;
- безпека клієнтів при оплаті по кредитній картці;
- безпечні засоби платежу по перерахуванню;
- підтримку спеціальних видів рахунків, для яких заздалегідь отриманий дозвіл плати за перерахуванням;
- електронний обмін даними із системою доставки товарів;
- електронний обмін даними із системою керування запасами;
- рецензування книг;
- формування рейтингу книг на основі відгуків користувачів.

1.3 Організація самостійної роботи

Кожний студент одержує завдання для самостійної роботи – предметну область, для якої необхідно розробити моделі та скелет програмного коду. Зразковий перелік можливих завдань наведений у додатку А. При виконанні завдання варто вивчити лекційні матеріали та технічну літературу, а також виконати вправи, наведені в даних методичних вказівках.

При розробці діаграм слід застосовувати доступні CASE-засоби, наприклад, Microsoft Visio 2007, Pacestar UML та інші.

Підсумковим результатом практичних занять є звіт по моделюванню програмного забезпечення для заданої предметної області.

2 МОДЕЛЮВАННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

Ціль етапу – освоєння навичок моделювання предметної області, що являє собою основу ***статичної моделі*** програмної системи.

2.1 Методика моделювання предметної області

Побудова моделі предметної області починається з ***виявлення абстракцій***, що існують у реальному світі, тобто тих основних концептуальних об'єктів, які зустрічаються в системі. При проектуванні об'єктно-орієнтованого програмного забезпечення варто структурувати програму так, щоб у центрі виявилися об'єкти із простору завдання.

Модель предметної області повинна являти собою словник термінів (об'єктів і класів), необхідних для моделювання.

У ході виявлення об'єктів із предметної області необхідно встановити, які зв'язки існують між ними. Особливий інтерес представляють два види відносин:

- ***узагальнення*** (відношення між підкласом і суперкласом);
- ***агрегування*** (відношення між цілим і частиною).

Між класами можуть існувати й інші відносини (асоціації), але ці два винятково важливі. В основу статичної моделі повинні бути покладені діаграми класів, що відображають модель предметної області.

Клас в UML – це місце для розміщення атрибутів (даних про об'єкти) і операцій (функцій, виконуваних об'єктами). Однак при моделюванні предметної області у визначенні атрибутів і операцій немає необхідності – ці процедури варто виконати пізніше. Тут же необхідно сконцентрувати увагу на виявленні власно об'єктів і відносин між ними.

Однієї з головних цілей побудови програми на основі абстракцій з реального миру є можливість повторного використання того, або іншого програмного модуля. Ця можливість створюється саме на етапі, коли класи формуються. Від вибору класів, тобто від якості абстрагування залежить, яке буде спостережуване поводження об'єкта. Реалізація цього поводження забезпечується внутрішнім пристроєм (проведеною інкапсуляцією), однак цей пристрій не повинен містити надмірностей, що ускладнюють поводження, або не повинен бути занадто простим, обмежуючим поводження.

При моделюванні предметної області (частини статичної моделі) варто рухатися зсередини назовні. Це означає, що спочатку в системі виділяються ключові об'єкти, а потім, вивчаючи, з якими ще об'єктами вони взаємодіють, до ключових об'єктів додаються інші з убутним ступенем значимості, поки не буде створена повна картина взаємодії об'єктів.

На відміну від процесу моделювання предметної області в динамічній частині системи, тобто при виявленні прецедентів, варто застосовувати рух зовні усередину, поки не буде зрозуміло, за допомогою чого реалізується кожний елемент необхідного поводження.

Таким чином, рухаючись ***одночасно*** в обох напрямках, необхідно забезпечити стикування статичної й динамічної частин системи посередині, не залишивши розриву.

Отже, моделювання предметної області – це погляд на систему зсередини назовні. На рисунку 2.1 показано, яке місце в загальній картині процесу ICONIX займає моделювання предметної області.

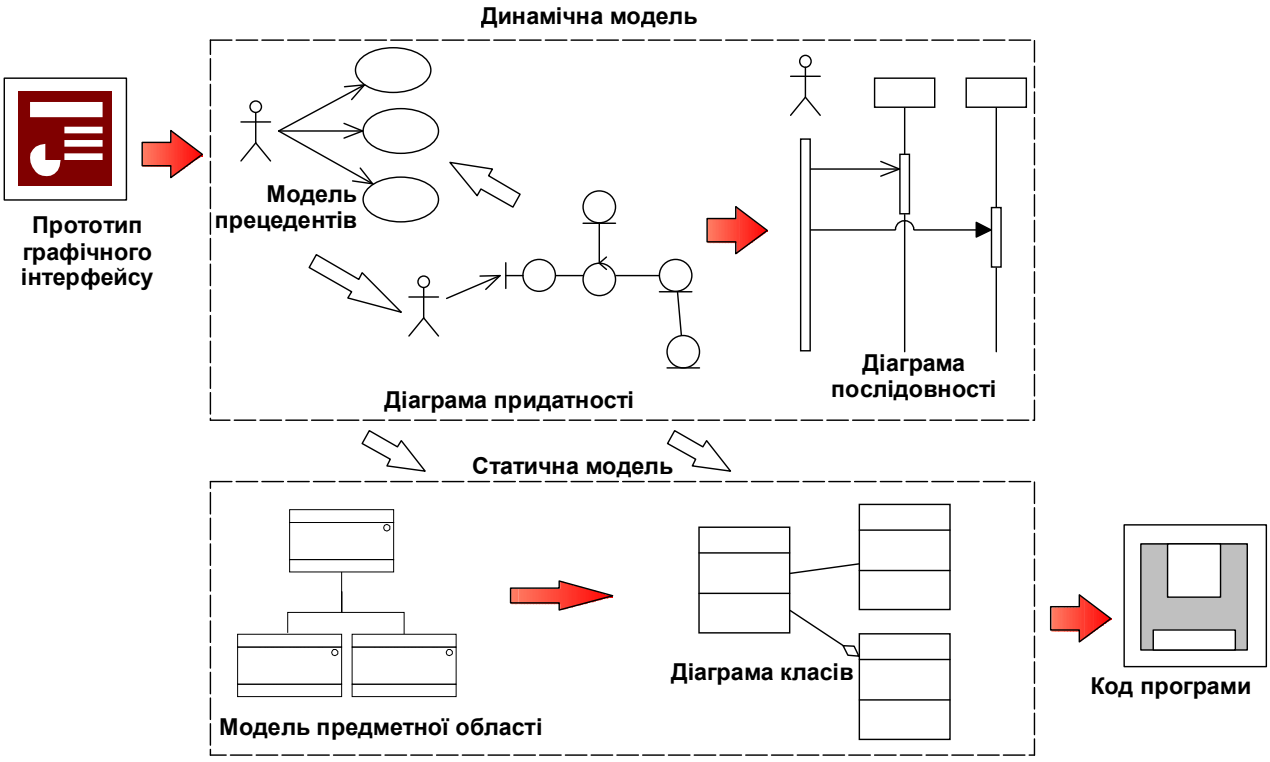

Рисунок 2.1 – Моделювання предметної області в процесі ICONIX

Перше, що потрібно зробити при побудові статичної моделі системи, – це знайти класи, які адекватно відбивають абстракції предметної області.

Кращими джерелами для виявлення класів є:

- високорівневий опис завдання;
- низькорівневі вимоги до системи;
- експертна оцінка завдання.

Практично варто виписати максимально можливе число речень із цих джерел, а потім в цих реченнях обвести або підкреслити всі іменники й іменні групи. У такий спосіб можна знайти майже всі важливі доменні об'єкти (класи). У міру уточнення цього переліку повинне відбуватися наступне:

- іменники й іменні групи можуть стати об'єктами або атрибутами;
- дієслова й дієслівні групи можуть стати операціями й асоціаціями;
- родовий відмінок показує, що іменник повинен бути атрибутом, а не об'єктом.

Далі варто видалити зі списку класів непотрібні (надлишкові або несуттєві) і непридатні (занадто розпливчасті) елементи.

При побудові діаграм класів можна також прийняти попередні рішення про узагальнення (відносинах виду «є»). Етап моделювання предметної області – це саме той момент, коли варто також прийняти рішення про агрегування класів.

У підсумку модель предметної області, доповнена асоціаціями (статичними відносинами між парами класів), повинна адекватно описувати статичні аспекти завдання, що не залежать від часу, і в цьому нагадувати діаграми сутність-зв'язок, яка застосовується в моделюванні бази даних.

2.2 Розповсюджені помилки при моделюванні предметної області

При моделюванні предметної області студенти допускають наступні помилки:

1. Відразу починають призначати кратності асоціаціям.

При моделюванні предметної області не слід звертати увагу на кратності, тому що це забирає час і може стати причиною «аналітичного паралічу».

2. Занадто велику увагу приділяють аналізу іменників і дієслів, забуваючи про все інше.

При складанні великого словника є ризик спуститися на надто низький рівень абстракції.

3. Включають у класи операції, не вивчивши, як треба, прецеденти й діаграми послідовності.

Не варто приділяти занадто багато уваги визначенню операцій на етапі моделювання предметної області. У цей момент ще мало інформації для прийняття обґрунтованих рішень.

4. При аналізі асоціації виду «є частиною» зазнають труднощів, що використовувати - агрегування або композицію.

На стадії моделювання предметної області краще говорити просто про агрегування. Більш точний вибір варто відкласти до етап детального проектування.

5. Студенти дають класам малозрозумілі імена, наприклад, cPortMgr.Intf *замість* PortfolioManager.

Чим очевидніше імена класів, тим простіше взаєморозуміння учасників процесу розробки програми. Про акроніми й інші види скорочень можна подумати на етапі реалізації.

6. Студенти відразу починають користуватися програмними конструкціями, наприклад, відносинами дружності й параметризованими класами.

Ці конструкції, що запозичені з мови C++, мають відношення до простору рішення, а не до простору завдання. При моделюванні предметної області варто сконцентрувати увагу на просторі ***завдання***.

2.3 Вправи

Знайдіть і виправте помилки у наведених нижче прикладах.

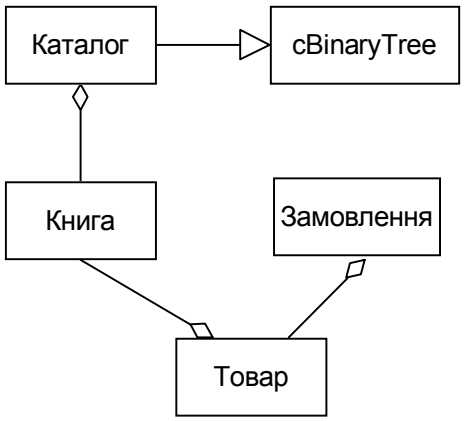

Рисунок 2.2 – Вправа 1

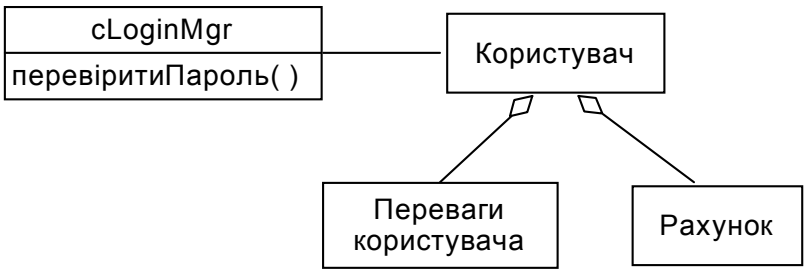

Рисунок 2.3 – Вправа 2

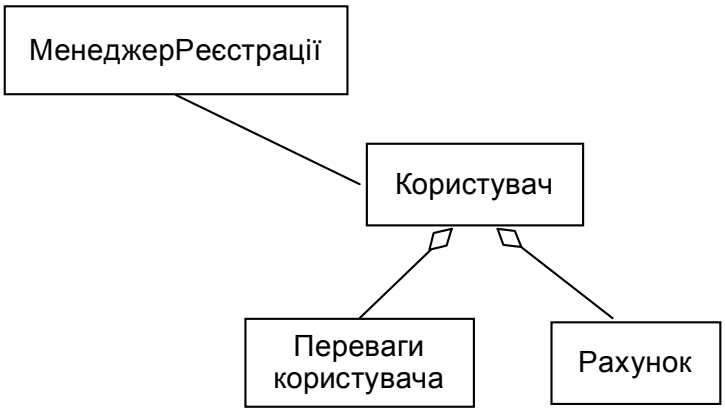

Рисунок 2.4 – Вправа 3

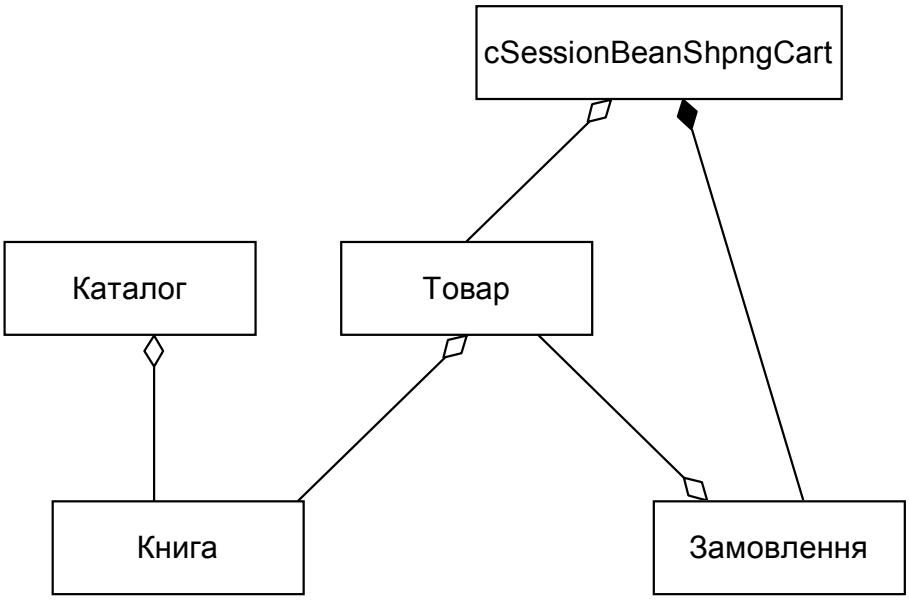

Рисунок 2.5 – Вправа 4

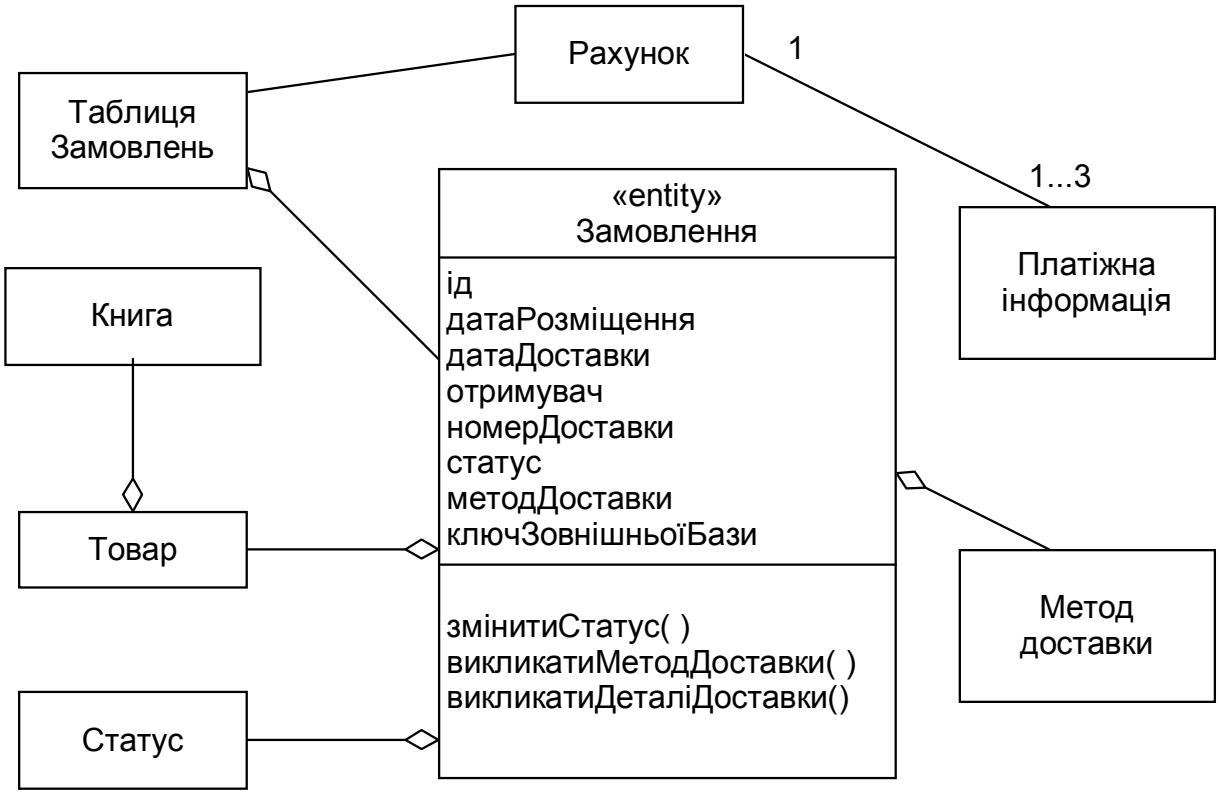

Рисунок 2.6 – Вправа 5

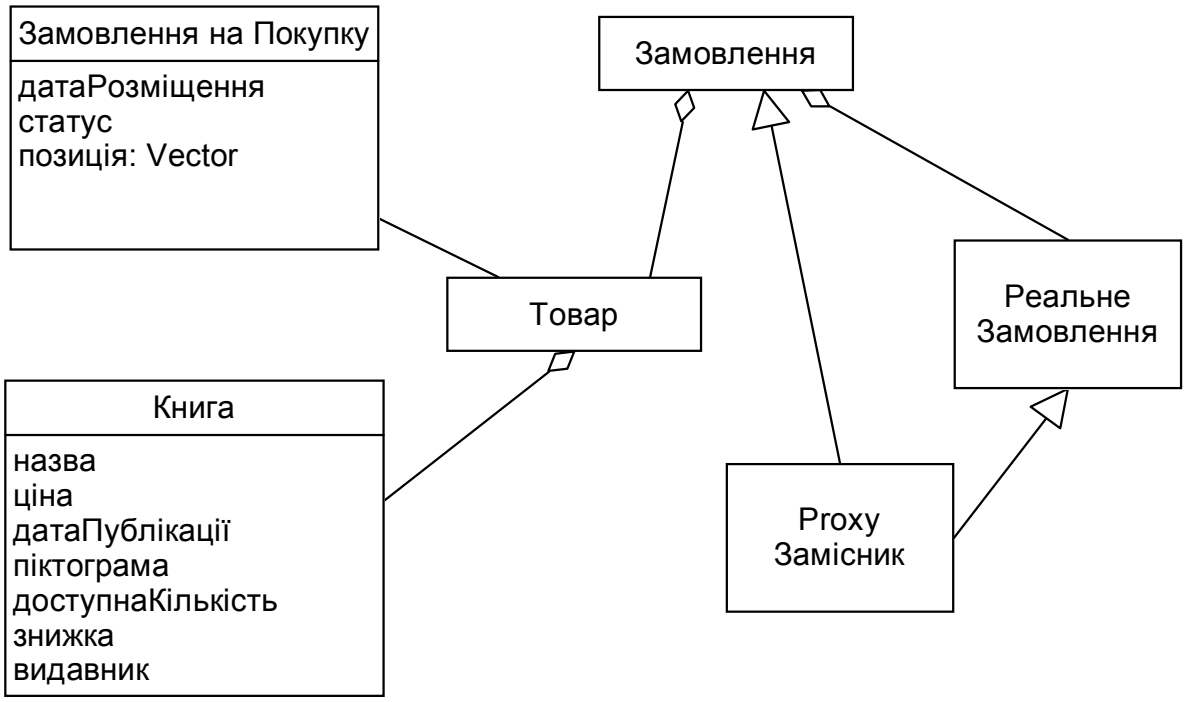

Рисунок 2.7 – Вправа 6

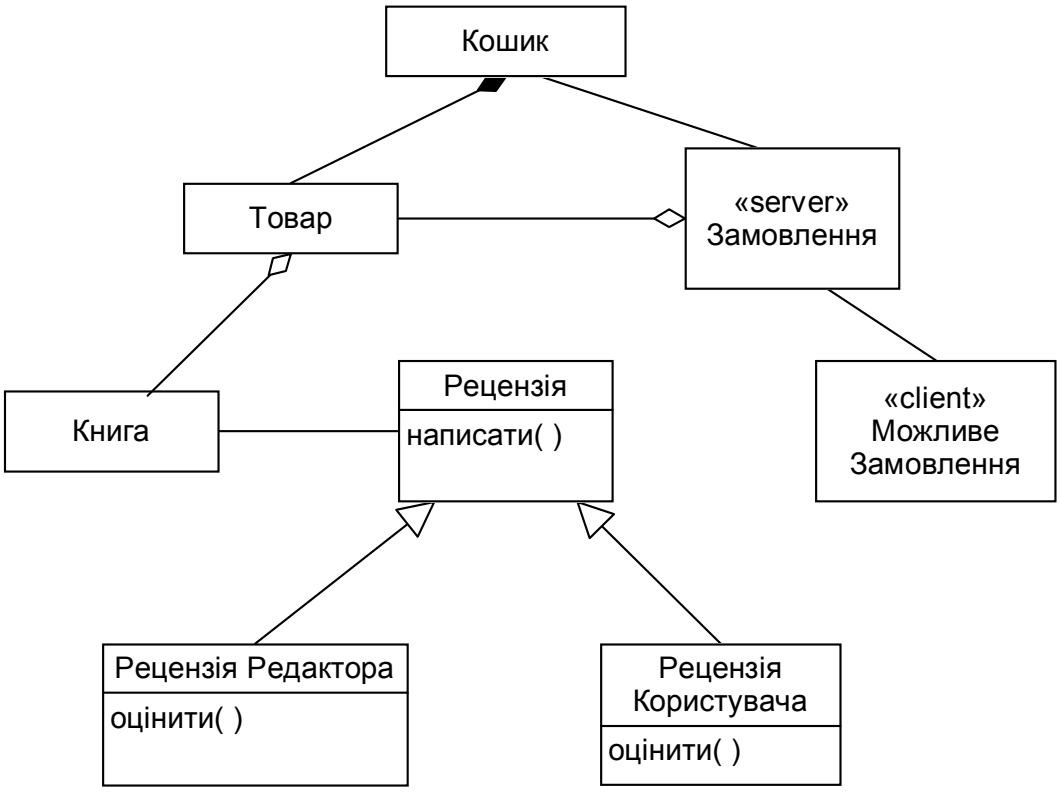

Рисунок 2.8 – Вправа 7

2.4 Модель предметної області

Зразок повної моделі предметної області для книжкового Internet-магазина наведений на рисунку 2.9.

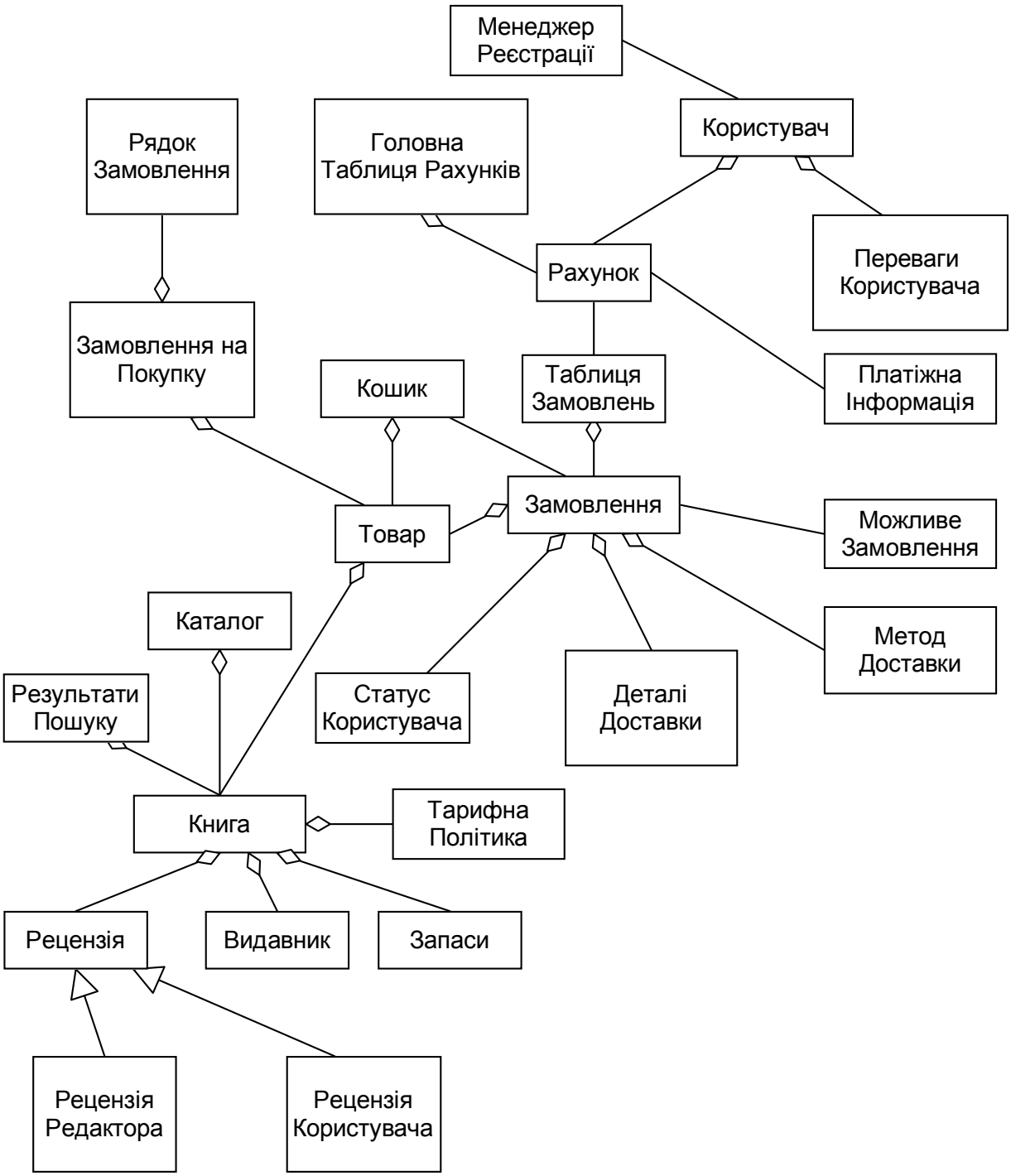

Рисунок 2.9 – Модель предметної області для книжкового Internet-магазина

3 МОДЕЛЮВАННЯ ПРЕЦЕДЕНТІВ

Ціль етапу – освоєння методики й придбання навичок побудови діаграми прецедентів.

3.1 Методика моделюванню прецедентів

Робота системи залежить від того, як до неї звертаються й чого хочуть домогтися. Зазвичай відповіді на питання “Що хочуть робити користувачі?” пов'язані із графічним інтерфейсом користувача.

На рисунку 3.1 показане місце моделювання прецедентів у загальній картині процесу ICONIX.

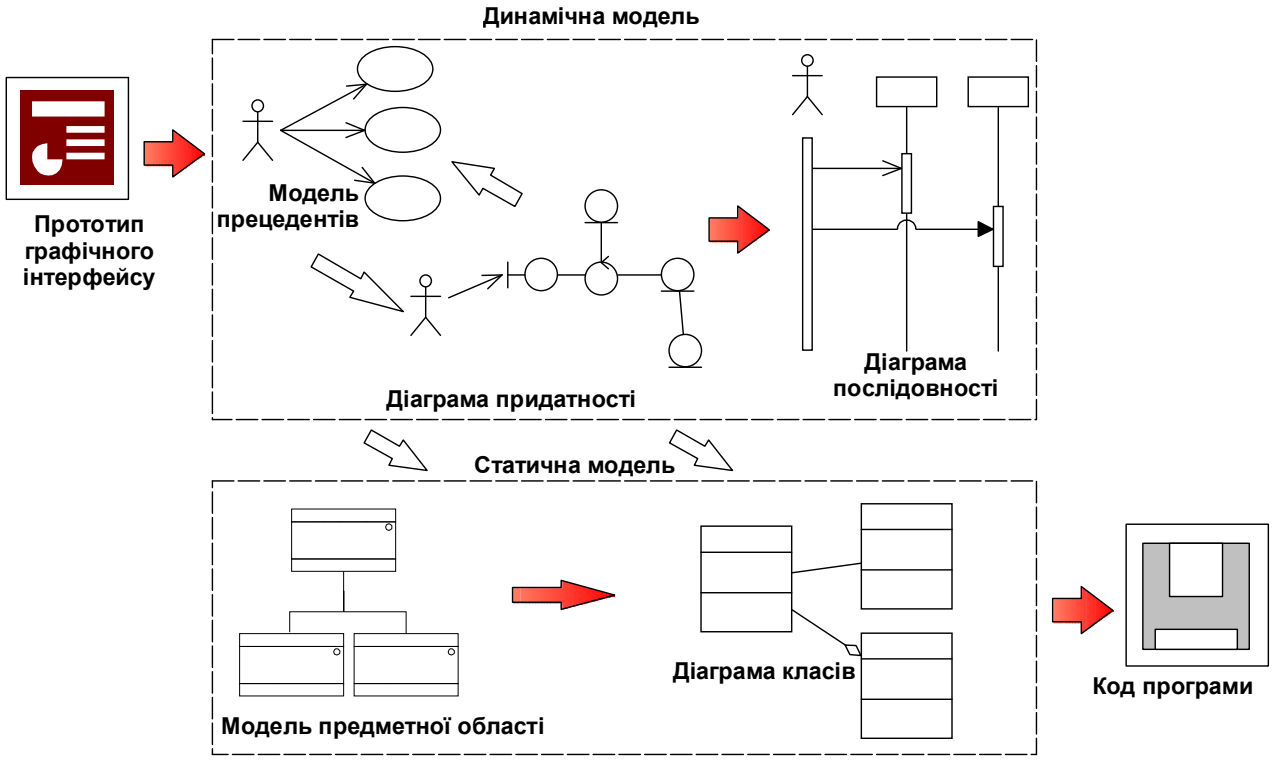

Рисунок 3.1 – Місце моделювання прецедентів у процесі ICONIX

Для визначення прецедентів потрібно використовувати прототип графічного інтерфейсу, а робота над моделлю прецедентів виконується паралельно з моделюванням предметної області. Таким чином, з моделі прецедентів безпосередньо випливає динамічна частина об'єктної моделі.

З рисунка 3.1 також видно, що потрібно постійно уточнювати статичну модель на основі аналізу прецедентів. Те ж саме стосується діаграм придатності й послідовності. Статична модель розвивається в міру розгляду все нових сценаріїв. Саме так відбувається еволюція від грубої моделі предметної області до детальної статичної моделі рівня проектування. Цей

підхід цілком і повністю ґрунтується на прецедентах – вони визначають і архітектуру, і проект програмної системи.

Підхід до побудови динамічної моделі треба описати як «зовні усередину». Варто почати з обстеження користувачів, які перебувають поза системою, і по цьому шляху розкрити деталі поводження програми. На цій основі буде створена структура програми, що підтримує необхідне поводження.

Просування спрямоване усередину, одним його кроком є розгляд одного конкретного сценарію. Оскільки прецеденти – це базові одиниці декомпозиції, то все інше жорстко підкоряється цьому руху.

Щоб вирішити завдання побудови прецедентів для нової системи, необхідно із самого початку ідентифікувати якнайбільше прецедентів, а потім скласти й надалі постійно уточнювати їхній словесний опис. По ходу справи будуть виявлятися нові прецеденти й загальні для них фрагменти.

При виявленні прецедентів рекомендується зв'язувати їх з майбутнім документом – "Керівництвом користувача". Зв'язок між кожним прецедентом і розділом керівництва повинен бути очевидним. Основа на прецедентах означає: «Спочатку напишіть посібник користувача, а потім код». Якщо переробляється успадкована система, то можна просто взяти за основу посібник користувача й вносити в нього необхідні зміни.

Текст прецеденту варто формулювати в ясній і короткій формі. Кожна пропозиція повинне мати структуру дієслово-іменник, а актори й потенційні доменні об'єкти повинні бути відразу видні. У міру виявлення нових об'єктів або уточнення їхнього поводження потрібно відразу обновляти модель предметної області. Важливо також уважно стежити за можливими альтернативними послідовностями дій для кожного прецеденту.

Рекомендується наступна послідовність моделювання прецедентів:

1. *Створити шаблон прецеденту,* у якому передбачити розділи «Головна послідовність» і «Альтернативні послідовності». Інші розділи не потрібні, вони будуть тільки відволікати увагу.

2. *Задати собі питання: «Що повинне відбуватися?».* З відповіді на нього починається головна послідовність.

3. *Задати собі питання: «Що ще може відбуватися?».* Важливо врахувати все, що може зробити користувач. Від чіткості визначення поводження залежить якість роботи системи.

Існують спеціальні механізми для вичленовування фрагментів, загальних для *деякого* набору прецедентів, наприклад, відносини ***включення*** (include) і ***розширення*** (extend), наявні в UML.

Для виділення логічних ділянок робіт, які можна розподілити між різними групами співробітників прецеденти рекомендується поєднувати в

пакети. Варто дотримуватися правила: *кожний пакет повинен відповідати розділу посібника користувача.*

Переходити до подальших етапів процесу розробки треба після того, як досягнуті наступні цілі моделювання прецедентів:

- прецеденти описують всю необхідну функціональність системи;
- для кожного прецеденту чітко й коротко описана головна послідовність дій, а також всі альтернативні послідовності;
- виділені (за допомогою деякого механізму) сценарії, загальні для декількох прецедентів.

3.2 Типові помилки при моделюванні прецедентів

1. Замість словесного сценарію використання студенти пишуть функціональні вимоги.

Варто чітко відрізняти опис порядку використання (поводження) від вимог до системи. Вимоги зазвичай формулюються з розрахунку на те, ***що система повинна робити***. Сценарій же описує дії, що здійснюються користувачем, і реакцію системи на ці дії. В остаточному підсумку, потрібно, щоб текст прецеденту служив специфікацією поводження системи для описуваного сценарію й розташовувався в лівій частині діаграми послідовності. Повинне бути відразу видно, *як* система, що представлена у вигляді сукупності об'єктів і повідомлень, реалізує *необхідне поводження*, описане за допомогою тексту прецеденту.

2. Описуються атрибути й методи, а не порядок використання.

Тексти прецедентів не повинні зайво докладно описувати деталі подання. Варто також утримуватися від посилань на поля в екранних формах. У тексті прецеденту не варто ні іменувати, ні описувати методи (операції), оскільки вони говорять про те, *як* система робить щось, а не про те, *що* вона повинна робити.

3. Прецеденти записуються занадто лаконічно.

При написанні текстів прецедентів, багатослівність більш переважна, ніж лаконічність. При переході до аналізу придатності й моделюванню взаємодій прийдеться відбити всі деталі операцій користувача, тому цілком логічно хоча б частину цих деталей із самого початку описати в прецедентах. Прецеденти будуть бути основою посібника користувача, тому краще включити багато деталей, чим щось випустити з уваги.

4. Прецеденти повністю абстрагуються від інтерфейсу користувача.

Один з фундаментальних принципів, заснований на прецедентах, полягає в тому, що при проектуванні системи розроблювачі відштовхуються від її сприйняття користувачами. Тому не можна ігнорувати дії, які

користувачі виконують за допомогою екрана й клавіатури, тобто особливості інтерфейсу.

5. Не привласнюються явні імена граничним об'єктам.

Граничними називаються об'єкти, з якими взаємодіють актори. До їхнього числа ставляться екранні форми, діалогові вікна й меню. Тому граничні об'єкти варто йменувати в текстах прецедентів явно. Крім того, поводження таких об'єктів буде досліджено на етапі аналізу придатності й присвоєння їм імен на ранніх стадіях дозволить уникнути неоднозначності надалі.

6. Прецеденти описуються не з погляду користувача.

З погляду користувача прецедент найбільше ефективно формулюється із застосуванням *дієслів теперішнього часу в дійсній заставі*, наприклад, “обновити”, “доставити”. Інженери давно використовують пасивну заставу (“виводиться”, “будується”), але прецеденти повинні описувати дії користувача й реакцію на них системи, а такого роду текст найкраще звучить у дійсній заставі.

7. Описуються тільки дії користувача, а реакція системи ігнорується.

Текст прецеденту повинен відображати як подію, так і відгук на неї, наприклад: «Система робить те-те, коли користувач робить те-те». Прецедент повинен описувати багато чого з того, що відбувається «під капотом» у відповідь на дії користувача – створення нових об'єктів, контроль даних, що вводяться, або вивід повідомлень про помилки. У тексті прецеденту розглядаються обидві сторони діалогу користувача із системою, а *поводження програми – це те, що потрібно виявити, – перебуває на системній стороні цього діалогу.* Тому при ігноруванні реакції системи ігнорується поводження програми.

8. Опускаються описи альтернативних послідовностей дій.

Головні потоки зазвичай простіше виявити й описати. Але це не означає, що роботу з альтернативними потоками можна відкласти до етапу детального проектування. Досвід показує, що, коли залишаються нерозкритими важливі альтернативні послідовності, виникають проблеми з доведенням проекту.

3.3 Вправи

В наведених нижче текстах прецедентів треба виправити наявні три-чотири помилки.

Вправа1. [*із прецеденту Відкрити Рахунок*]

Головна послідовність. Клієнт уводить необхідну інформацію. Система перевіряє її коректність і створює новий об'єкт "Рахунок".

Альтернативна послідовність. Якщо уведені некоректні дані, система виводить відповідне повідомлення про помилку.

Вправа2. [*із прецеденту Пошук по Автору*]

Головна послідовність. Користувач відправляє запит. Система виводить сторінку, що містить результати пошуку.

Вправа3. [*із прецеденту Реєструвати*]

Головна послідовність. Клієнт уводить свій код і пароль, після чого натискає кнопку "Зареєструватися". Система виводить "Початкову Сторінку".

Вправа4. [*із прецеденту Змінити Вміст Кошика*]

Головна послідовність. На "Сторінці Кошика" клієнт змінює кількість товару в кошику й натискає кнопку "Обновити". Потім клієнт натискає кнопку "Продовжую Купувати".

Вправа5. [*із прецеденту Скасувати Замовлення*]

Головна послідовність. Система виводить інформацію про замовлення на "Сторінці Скасування Замовлення", у тому числі вміст замовлення й адресу доставки. Клієнт натискає кнопку "Підтвердити Скасування".

Вправа 6. [*із прецеденту Доставити Замовлення*]

Головна послідовність. Службовець закінчує впакування замовлення, записує його номер і відправляє через відповідного постачальника.

3.4 Зразок діаграми прецедентів для книжкового Internet-магазина

Зразок повної діаграми прецедентів для книжкового Internet-магазина показаний на рисунку 3.2.

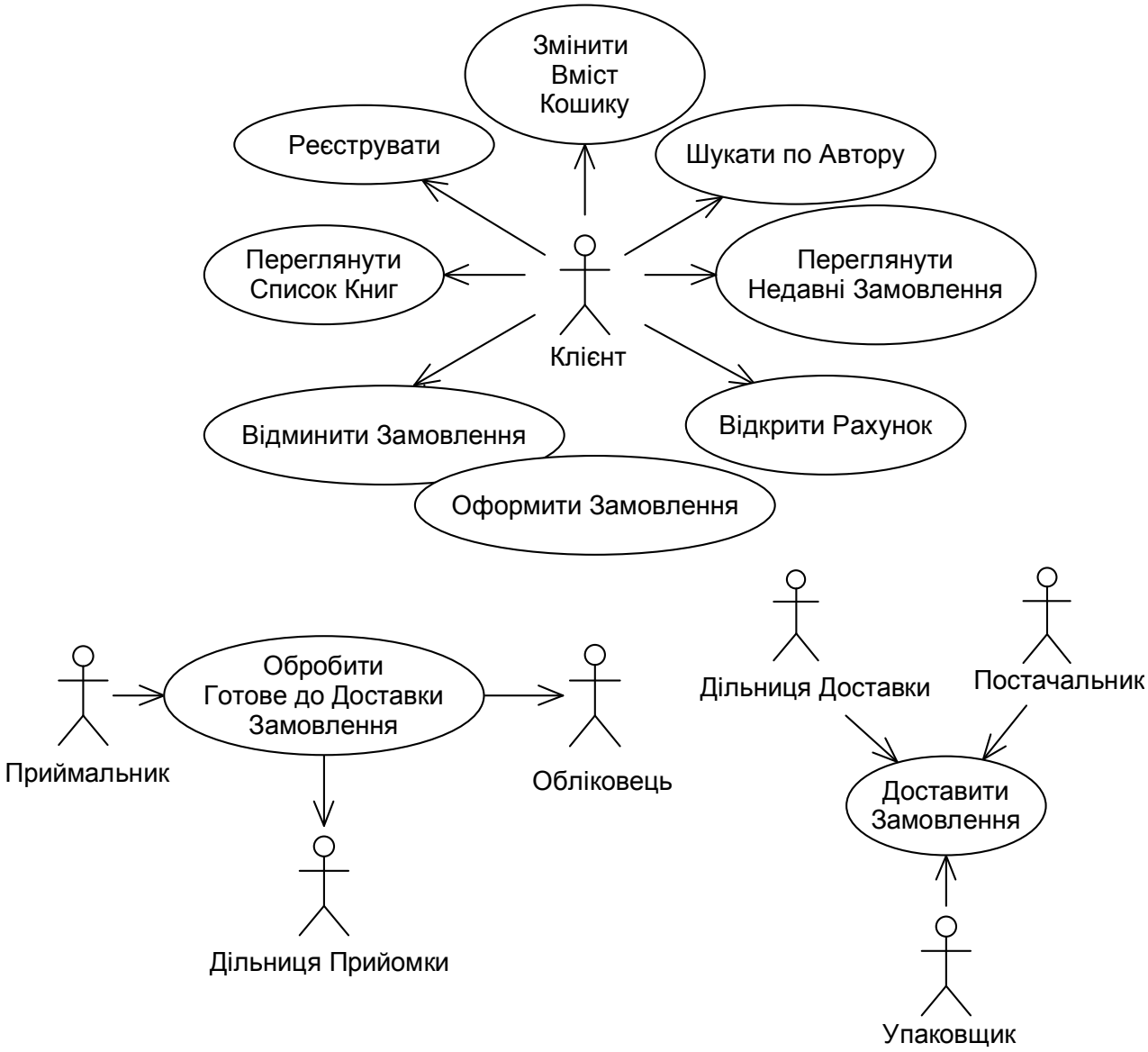

Рисунок 3.2 – Діаграма прецедентів для книжкового Internet-магазина

4 АНАЛІЗ ПРИДАТНОСТІ МОДЕЛІ

Ціль етапу – освоєння прийомів, що застосовуються в аналізі придатності моделі, для забезпечення вимог до системи.

4.1 Значення й сутність аналізу придатності

У процесі ICONIX аналіз придатності служить сполучною ланкою між аналізом *(що)* і проектуванням *(як)*. Схема цього зв'язку представлена на рисунку 4.1.

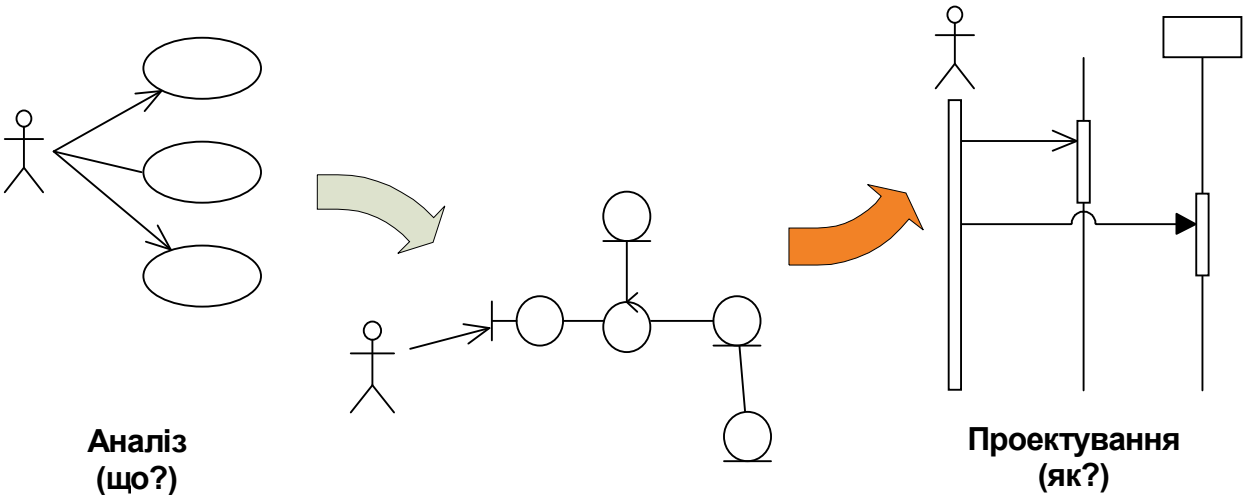

Рисунок 4.1 – Зв'язок між аналізом і проектуванням

Для забезпечення зв'язку між статичною й динамічною моделями необхідно відповістити на запитання: ***"Які об'єкти потрібні для кожного прецеденту?"***

Діаграма придатності нагадує діаграму кооперації з UML – на ній зображені об'єкти, що беруть участь у сценарії, і способи їхньої взаємодії. Оскільки аналіз придатності не входить у ядро UML, то він вимагає деяких стереотипів, що дозволяють зв'язувати нестандартні піктограми з будь-якими символами. При аналізі придатності в ролі таких стереотипів виступають піктограми класів (рис. 4.2).

Рисунок 4.2 – Зображення об'єктів на діаграмах придатності

Застосовувані на етапі аналізу придатності стереотипи об'єктів характеризуються наступними особливостями:

- ***граничні об'єкти*** – це такі об'єкти, з якими безпосередньо взаємодіють актори (користувачі) у розроблювальній системі. До них ставляться екранні форми, діалогові вікна й меню;

- ***сутнісні об'єкти*** часто відображаються на таблиці бази даних і файли з інформацією, яка повинна «пережити» час виконання прецеденту. Окремі сутності, наприклад, результати пошуку, є тимчасовими й зникають, коли прецедент завершується;

- ***керуючі об'єкти*** (контролери) містять у собі логікові додатки, тобто з'єднують користувача зі збереженими даними. Саме в них інкапсулюються бізнес-правила й стратегії. Керуючі об'єкти дозволяють *локалізувати зміни*, не зачіпаючи інтерфейсу користувача й схеми бази даних. Зрідка (приблизно в 20% випадків) контролери виявляються реальними об'єктами в проекті, але зазвичай вони виконують функції *контейнерів.*

Процес розробки програмного забезпечення починається з рівня вимог, на якому аналізується тільки те, *що* потрібно користувачам. При цьому не розглядаються деталі реалізації. Потім кут зору змінюється й вся увага зосереджується винятково на проектуванні. Перехід від погляду на світ з позиції «що робити» до погляду з позиції «як робити» – це одна із самих важких проблем при розробці програмного забезпечення. Саме для рішення цього завдання й призначений аналіз придатності.

На рисунку 4.3 показане місце аналізу придатності в загальній картині процесу ICONIX.

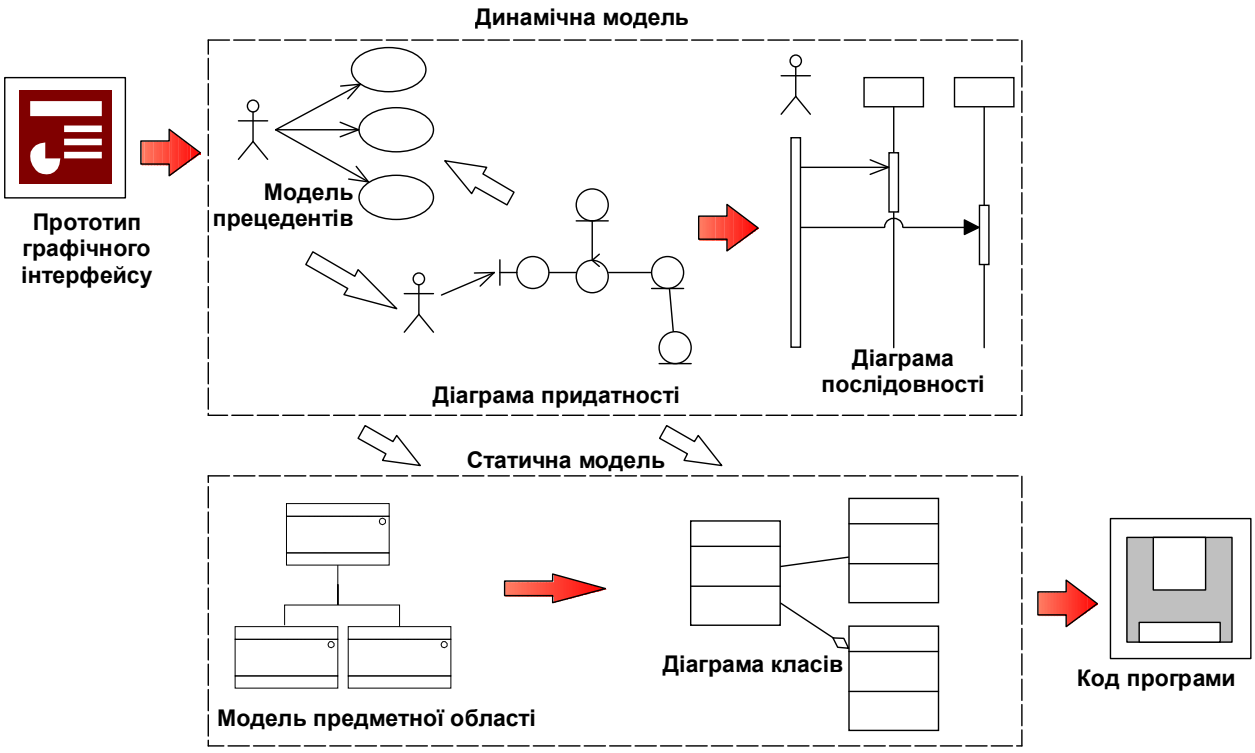

Рисунок 4.3 – Аналіз придатності – уточнення моделі предметної області

4.2 Основні елементи й методика аналізу придатності

Аналіз придатності повинен забезпечити:

- *контроль відсутності тривіальних помилок* (аналіз придатності допомагає впевнитися, що тексти прецедентів коректні, або уточнити текст – якщо на етапі аналізу прецедент формулювався з погляду посібника користувача, то на етапі проектування він стає описом порядку застосування *в контексті об'єктної моделі);*
- *перевірку повноти й правильності опису прецедентів* (перевірка створює впевненість, що в прецедентах описані всі необхідні альтернативні послідовності);
- *виявлення об'єктів, пропущених на етапі моделювання предметної області;*
- *виявлення невідповідностей у найменуваннях об'єктів*.

Аналіз придатності для прецеденту виконується шляхом дослідження його тексту, зображення акторів, граничних і сутнісних об'єктів і контролерів, а також зв'язків між різними елементами на діаграмі. При цьому головна й альтернативна послідовності повинні вміститися *на одній діаграмі*.

Існує чотири основних правила при побудові діаграм придатності:

1. Актори можуть спілкуватися тільки із граничними об'єктами.
2. Граничні об'єкти можуть спілкуватися тільки з контролерами й акторами.
3. Сутнісні об'єкти можуть спілкуватися тільки з контролерами.
4. Контролери можуть спілкуватися із граничними об'єктами, сутнісними об'єктами й іншими контролерами, але не з акторами.

Варто врахувати, що граничні й сутнісні об'єкти позначаються іменниками, а контролери – дієсловами. Іменники не можуть спілкуватися з іншими іменниками, а дієслова можуть спілкуватися як з іменниками, так і з дієсловами.

На рисунку 4.4 показані правила застосування стереотипів при побудові діаграм придатності.

При розробці діаграми придатності необхідно ***прочитати*** опис послідовності дій у тексті прецеденту, ***простежити*** уздовж асоціацій на діаграмі й ***побачити*** точну відповідність між текстом і картинкою. Можливо, прийдеться переписати текст прецеденту, щоб усунути неоднозначність і ввести явні посилання на граничні й сутнісні об'єкти. Написати ідеальний текст прецеденту з першої спроби дуже складно.

Результати аналізу придатності використовуються не тільки для поліпшення тексту прецеденту, але й для уточнення статичної моделі. Нові об'єкти, які виявляються в міру розробки діаграм, і основні атрибути варто включити в діаграми класів.

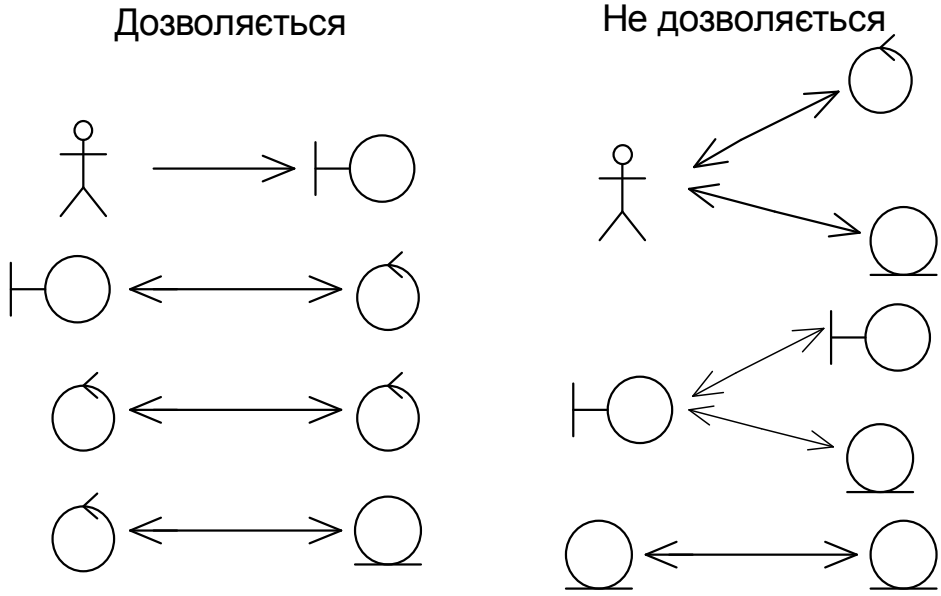

Рисунок 4.4 - Правила застосування стереотипів при побудові діаграм придатності

В якості зразка моделі предметної області на рисунку 4.5 зображена діаграма класів із приклада книжкового Internet-магазина, що включає деякі атрибути класів.

4.3 Типові помилки при аналізі придатності

1. Студенти намагаються виконувати детальне проектування на діаграмах придатності.

Концепція тимчасових діаграм корисна в контексті попереднього проектування, але перестає бути такою, коли справа доходить до детального проектування. Для цього найбільше підходять діаграми послідовності. У ході аналізу придатності потрібно *швидко пройти по всіх сценаріях* і добути із цього максимальну користь для проекту.

2. На діаграмах придатності не відбиваються альтернативні потоки.

Аналізу придатності варто піддати *весь* текст прецеденту, а не тільки головну послідовність. Нерідко самі цікаві аспекти поводження системи проявляються саме в контексті альтернативних послідовностей, так що їхній розгляд відіграє величезну роль при моделюванні.

3. При аналізі придатності не перевіряється погодженість імен класів на діаграмах класів і в текстах прецедентів.

Опис порядку застосування системи в контексті об'єктної моделі – це формула, що допомагає при побудові діаграм послідовності. Іменуючи граничні й сутнісні об'єкти в прецедентах, можна полегшити процес створення діаграм.

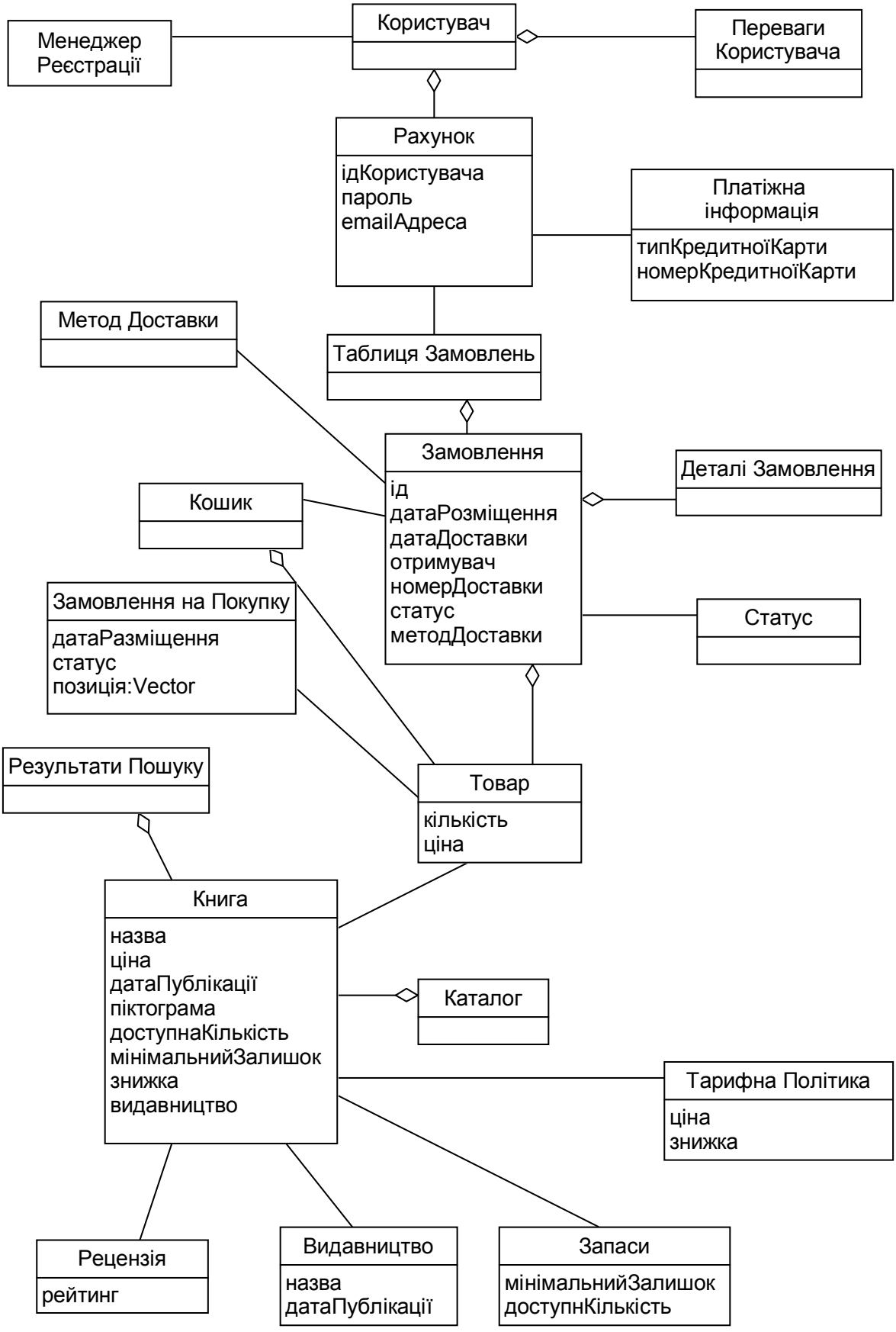

Рисунок 4.5 – Модель предметної області з атрибутами класів для книжкового Internet-магазина

4.4 Вправи

У наведених нижче діаграмах придатності є помилки. Завдання полягає в тім, щоб знайти й виправити ці помилки.

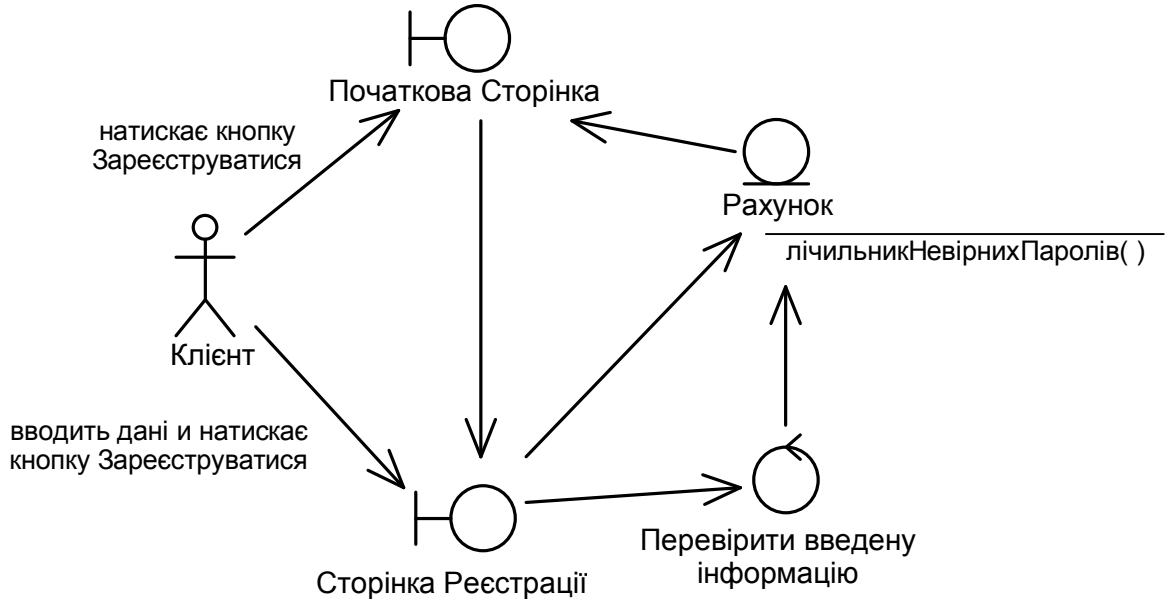

Рисунок 4.6 – Вправа 1 (Реєстрація)

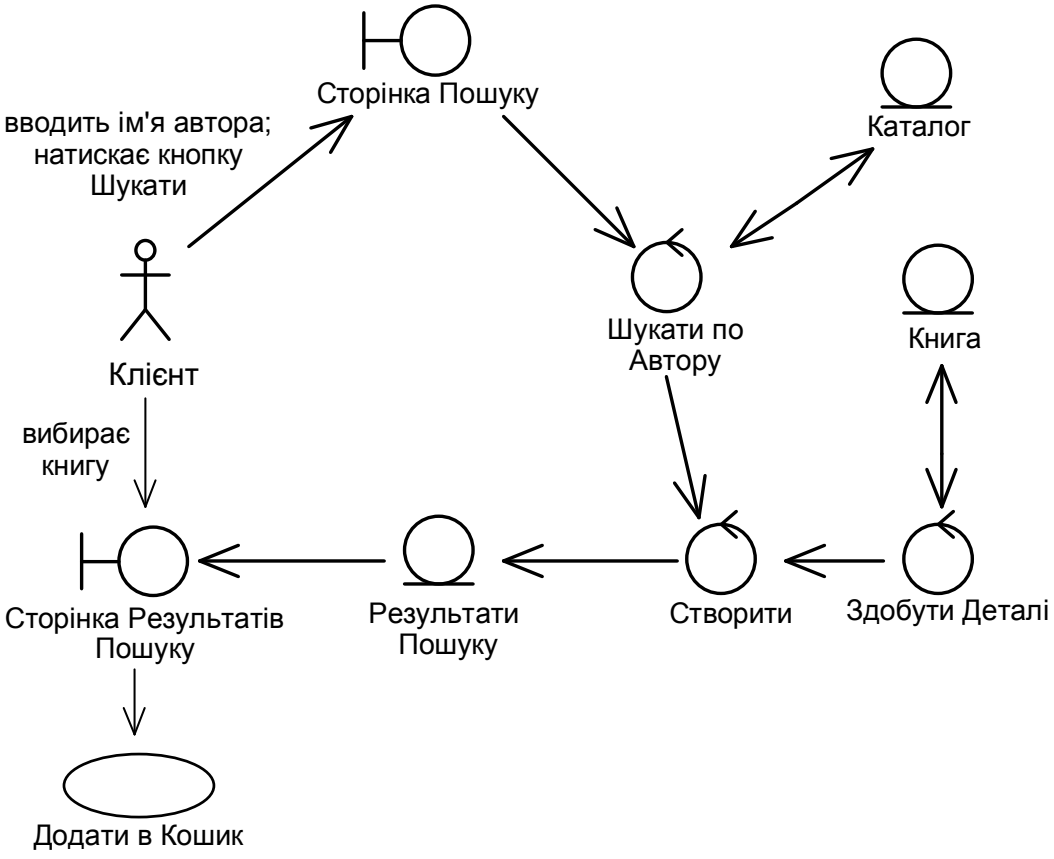

Рисунок 4.7 – Вправа 2 (Пошук по Автору)

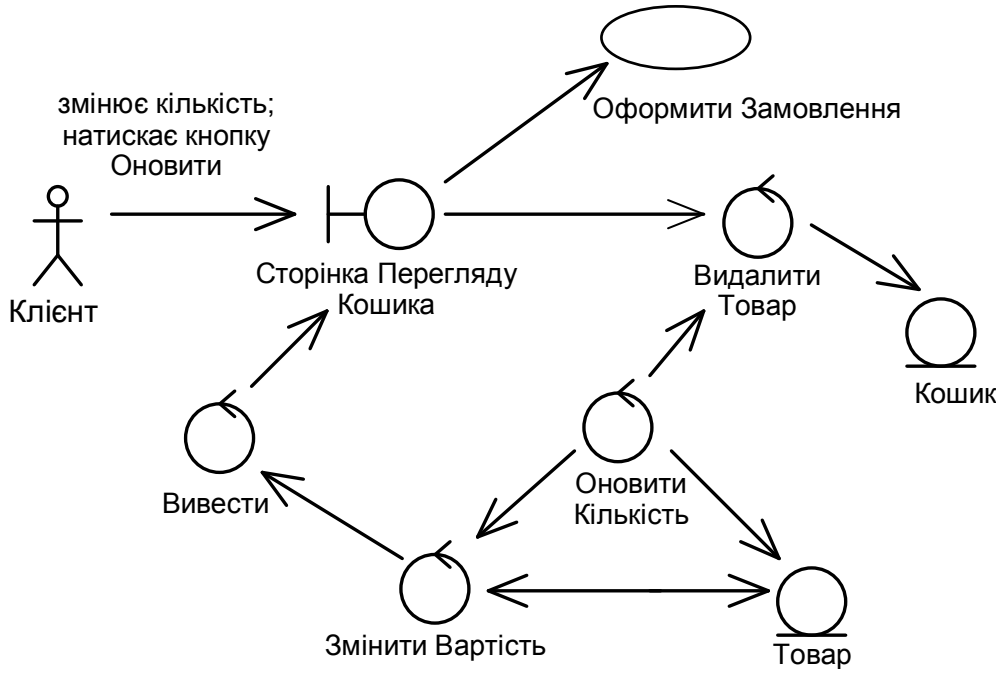

Рисунок 4.8 – Вправа 3 (Уміст Кошика)

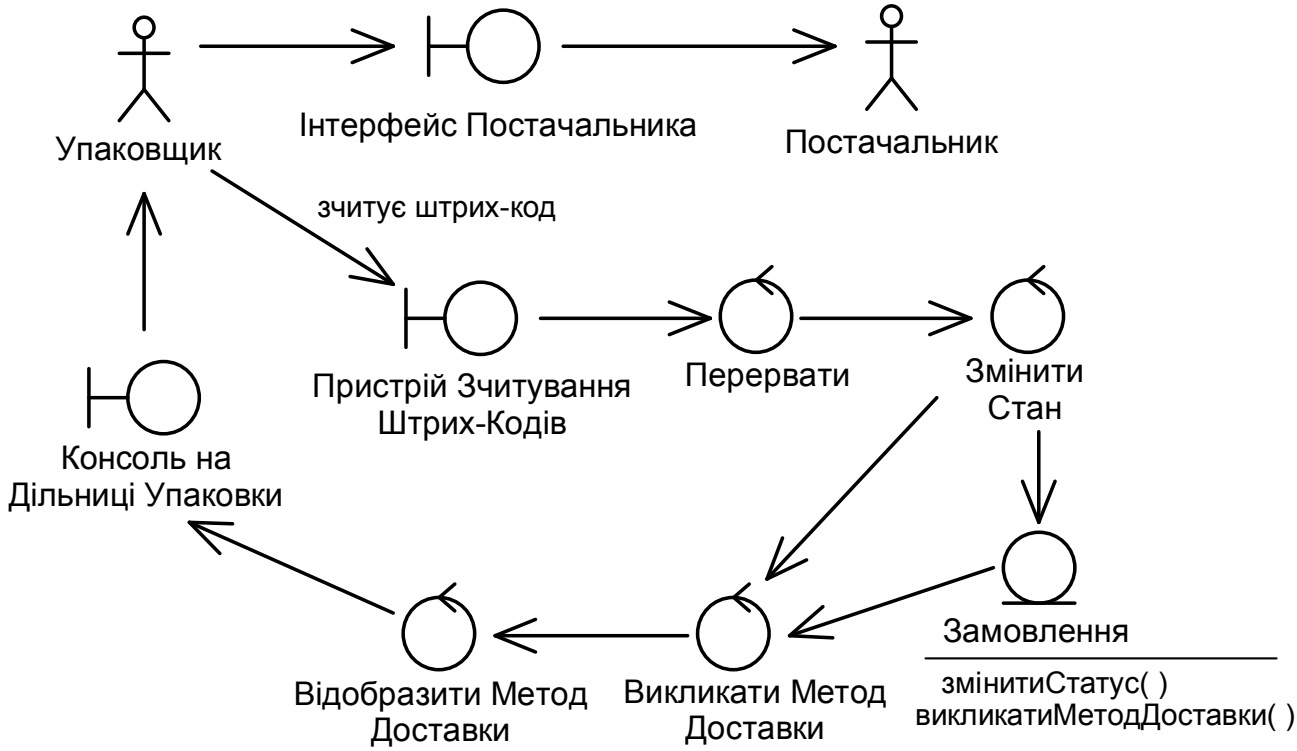

Рисунок 4.9 – Вправа 4 (Доставити Замовлення)

5 РОЗРОБКА ДІАГРАМ ПОСЛІДОВНОСТІ

Ціль етапу – придбати навички розробки діаграм послідовності, що моделюють взаємодію об'єктів.

5.1 Завдання при моделюванні взаємодій

Етап попереднього проектування дозволяє виявити об'єкти. На етапі ж детального проектування здійснюється розподіл функцій програми між цими об'єктами. Основним елементом такого детального проектування є діаграма послідовності – ***динамічна частина об'єктної моделі.***

Виконавши попереднє проектування за допомогою аналізу придатності, необхідно повернутися до сценаріїв і вивчити їх більш детально. Варто переглянути перші неформальні припущення про кооперацію об'єктів і уточнити подання. До цього моменту повинні бути виконані два завдання. *По-перше*, тексти прецедентів повинні бути повними, правильними, деталізованими й недвозначними. *По-друге*, варто виявити більшу частину необхідних об'єктів, принаймні, на концептуальному рівні абстракції.

На рисунку 5.1 показане місце діаграм послідовності в загальному процесі ICONIX.

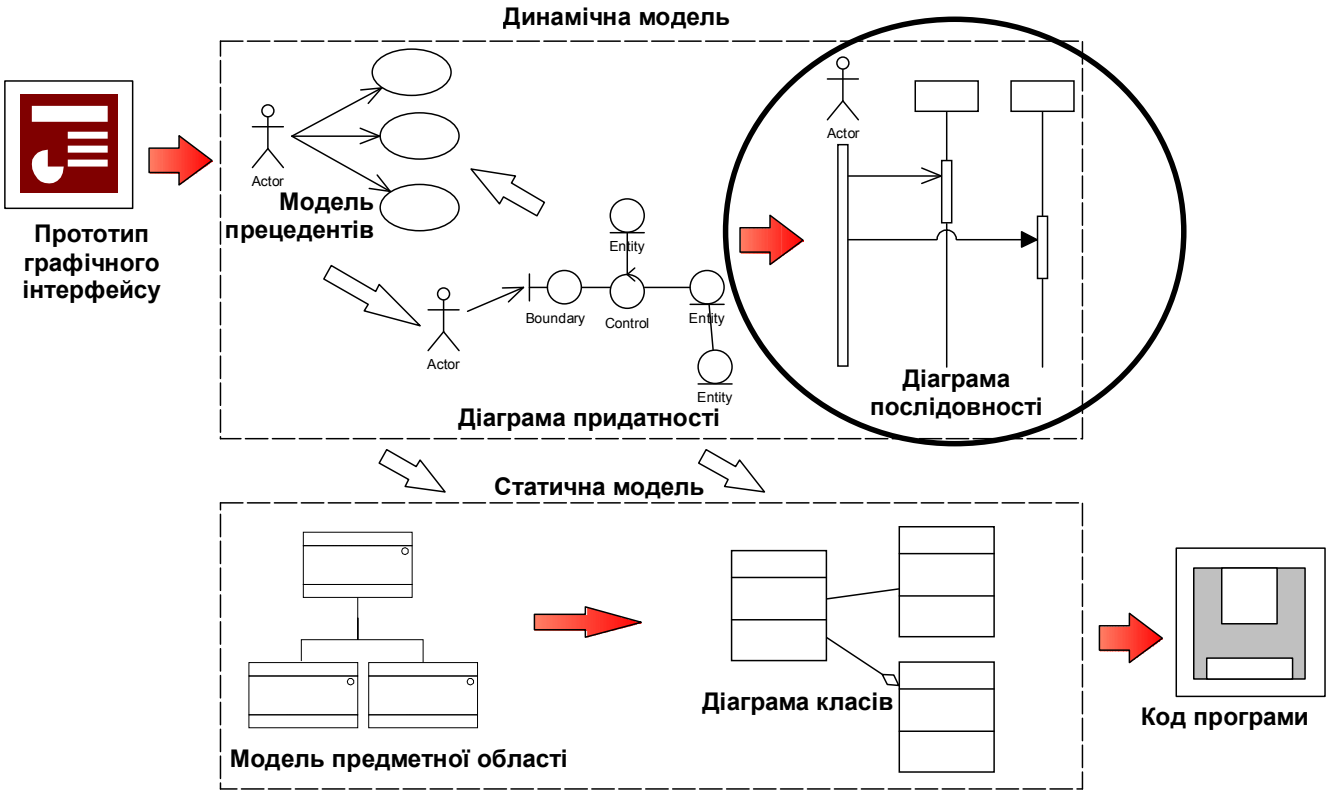

Рисунок 5.1 – Місце діаграми послідовності у процесі ICONIX

При моделюванні взаємодій необхідно вирішити три завдання:

1. Розподілити поводження між граничними, сутнісними й керуючими об'єктами.
2. Детально показати взаємодії між об'єктами, що беруть участь у кожному прецеденті.
3. Закінчити розподіл операцій по класах.

Зображуючи деталі поводження об'єктів на діаграмах послідовності, варто остаточно визначитися з тим, куди віднести всі атрибути й операції. Виконуючи таке *динамічне* моделювання, необхідно модифікувати й розширити *статичну* модель, поліпшуючи розуміння того, як повинна працювати система.

У процесі ICONIX *діаграми послідовності - це основний робочий продукт проектування.* Для кожного прецеденту створюється діаграма, що описує як головну, так і всі альтернативні послідовності дій. У результаті виходить ядро динамічної моделі, у якому дуже докладно визначене поводження системи й те, *як* реалізується це поводження.

Діаграма послідовності складається із чотирьох основних елементів:

1) *тексту послідовності дій у прецеденті,* що записується зверху долілиць по лівій стороні;
2) *об'єктів,* перенесених прямо з діаграми придатності й представлених у вигляді прямокутників, у яких у форматі « об'єкт-клас» записується ім'я або номер екземпляра об'єкта й ім'я класу об'єкта;
3) *повідомлень,* зображуваних стрілками, які спрямовані від одного об'єкта до іншого;
4) *методів (операцій), що* представляються у вигляді прямокутників (фокусів керування). Довжину прямокутника можна використовувати для того, щоб показати *фокус керування* в послідовності: метод володіє керуванням аж до точки, у якій прямокутник кінчається.

5.2 Методика розробки діаграм послідовності

Досвід показує, що на цій стадії розробки проекту виникає багато проблем (особливо якщо був пропущений етап попереднього проектування). У процесі ICONIX застосовується чотири етапи розробки діаграм послідовності.

На цих етапах варто виконати наступні процедури:

1. *Скопіюйте текст прецеденту зі специфікації.* Вставте його в ліве поле сторінки. Текст на діаграмі буде постійно нагадувати про те, чого потрібно домогтися.

2. *Додайте сутнісні об'єкти, представлені на діаграмі придатності.* Кожний з об'єктів – це екземпляр деякого класу, зображеного на діаграмі класів у статичній моделі. У цих об'єктів уже повинна бути проставлена більша частина атрибутів. Багато хто з них будуть виступати як дані, що передаються іншим об'єктам. Пропущені атрибути будуть додані по ходу роботи над діаграмою послідовності.

3. *Додайте граничні об'єкти й акторів* з діаграми придатності. Це буде перший крок до об'єднання двох просторів.

4. *Визначить, які методи й у які класи помістити* - це суть моделювання взаємодій.

Розподіл методів по класах припускає, зокрема, *перетворення контролерів,* зображених на діаграмах придатності, у методи й повідомлення, що реалізують потрібне поводження. У зв'язку із цим рекомендується *використовувати діаграму придатності як контрольний список,* щоб переконатися, що на діаграмах послідовності враховані всі вимоги до поводження системи. Варто врахувати, що один контролер на діаграмі придатності може транслюватися в кілька методів на діаграмі послідовності.

Існує два методи перетворення контролерів – керування на екрані й контролер прецеденту. Якщо при складанні діаграм для всіх прецедентів послідовно використовується тільки один підхід, то можна сказати, що в цьому випадку використовуються паттерни.

Паттерн – це опис взаємодії об'єктів і класів для рішення стандартного завдання проектування. Існує безліч стандартних пат тернів, які застосовуються при моделюванні взаємодій між різними об'єктами. Вони приносять велику користь завдяки ретельній проробці механізмів взаємодії.

Зіставлення діаграм послідовності й придатності дозволить бути впевненим, що проектується те, що потрібно користувачеві.

5.3 Типові помилки при складанні діаграм послідовності

1. Діаграми послідовності складаються не для кожного прецеденту.

Тільки після того, як будуть складені діаграми послідовності всіх прецедентів, можна бути впевненим, що повністю виявлено ролі, які кожний об'єкт може виконувати в системі, а також завдання кожного об'єкта.

2. На діаграмі послідовності немає тексту прецеденту.

Розміщення тексту прецеденту, що відбиває вимоги замовника, на полях діаграм послідовності дозволяє візуально простежити, як перетворюються вимоги в проект. Діаграма повинна відповідати словесному опису конкретного прецеденту.

3. Діаграми послідовності представлені на високому рівні абстракції, тобто не містять деталей.

Діаграми послідовності – це остання зупинка перед початком кодування, тому на них повинні бути зображені всі деталі проекту.

4. Студенти перетворюють діаграми послідовності в блок-схеми замість того, щоб використовувати їх для розподілу поводження між об'єктами.

Діаграма послідовності – це основний інструмент для прийняття рішень про розподіл поводження. Вона потрібна для того, щоб розписати операції класів. Звідси виходить, що не варто позначати стрілки-повідомлення вільним текстом – потрібно зіставити стрілці ім'я повідомлення, що збігається з ім'ям операції класу. Розподіл поводження (прийняття рішень про віднесення операцій до конкретних класів) – це основа процесу моделювання. Рішення, прийняті на цьому етапі роботи, визначають якість проекту в цілому.

5. Студенти помиляються у виборі напрямку стрілок.

Повідомлення, якими обмінюються об'єкти, приводять до виклику операцій класів. На діаграмах придатності напрямок стрілок є несуттєвим, а на діаграмах послідовності ігнорувати це питання вже не можна. Потік керування повинен бути показаний точно. Завжди повинне бути зрозуміло, який об'єкт є визивним, а який – викликуваним.

6. При розподілі поводження не дотримуються базових принципів об'єктно-орієнтованого проектування.

Клас повинен відповідати за зв'язану множину функцій. Аналогічне правило існує для об'єктів – об'єкти повинні бути щільними й слабко зв'язаними.

Ще два принципи, яких варто дотримуватися:

1) ***Прагнути забезпечити повторне використання.*** Чим більш загальними є об'єкти й класи, тим вище ймовірність, що їх можна буде використовувати в інших проектах.
2) ***Забезпечувати застосовність.*** Включаючи в клас метод, варто подумати, чи доречний він у цьому класі й чи дійсно завдання, що цей метод вирішує, має відношення до даного класу.

5.4 Вправи

У представлених нижче двох вправах (рис. 5.2-5.3) необхідно знайти й виправити помилки.

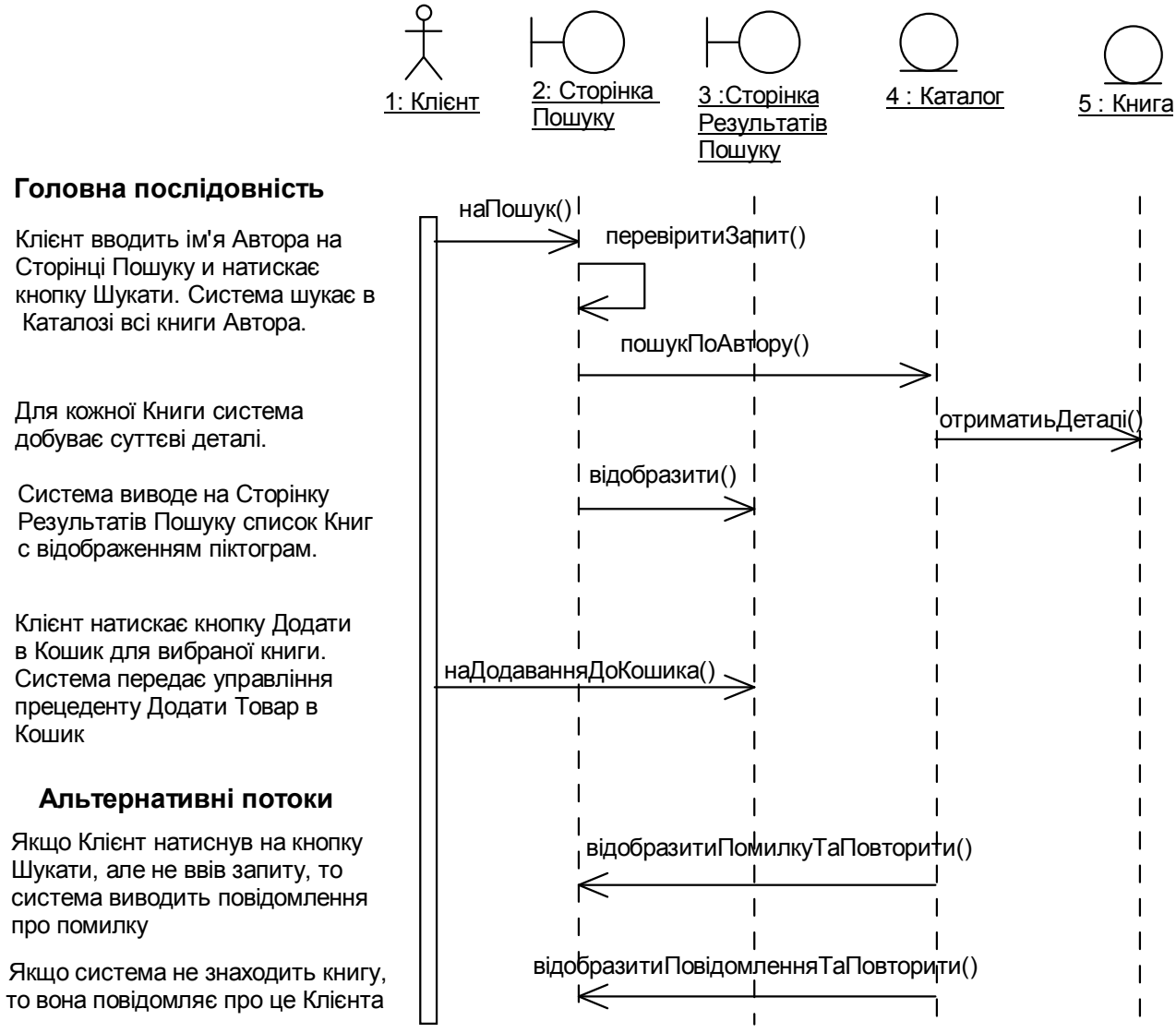

Рисунок 5.2 – Вправа 1 (діаграма послідовності для прецеденту Пошук по Автору)

Запитання: Які альтернативні потоки не показані на діаграмі?

6 РОЗРОБКА ДІАГРАМ КЛАСІВ

Ціль етапу – придбати навички розробки діаграм класів, як основи програмного коду.

6.1 Завдання на етапі розробки діаграм класів

При розробці діаграм класів необхідно забезпечити:

- *зв'язаність* (coupling), що характеризує достаток зв'язків між двома класами (варто прагнути до того, щоб класи були слабко зв'язаними, тобто мало залежними друг від друга);
- *щільність* (cohesion), що характеризує, наскільки сильно залежать друг від друга атрибути й операції одного класу (потрібно прагнути до *високої функціональної щільності)*;
- *достатність* (sufficiency), що означає – клас інкапсулює достатнє число абстракцій;
- *повнота* (completeness), що характеризує те, якою мірою інтерфейс класу покриває всі абстракції, для яких він розроблений (теоретично повний клас може повторно використовуватися в будь-яких контекстах);
- *примітивність* (primitiveness) говорить про те, що операція може бути ефективно реалізована (при наявності доступу) іншими елементами (варто створювати операції як будівельні блоки для інших програм).

6.2 Методика розробки діаграм класів

Клас втілює в собі безліч функціональних обов'язків, які визначають поводження об'єктів цього класу. Обов'язки реалізуються за допомогою операцій, обумовлених у контексті класу. Операція повинна здійснювати всього одну функцію, але у всій її повноті. Кожна операція класу перебуває в розпорядженні всіх його об'єктів.

Повнота операцій забезпечується структурою, тобто реалізацією класу. Структура об'єкта визначається атрибутами класу. Атрибут являє собою визначення порції даних, доступної для об'єкта. Об'єкти класу мають власні значення кожного з атрибутів. Наприклад, до складу класу “Товар” доречно включити атрибути “кількість” і “вартість”. При цьому кожний товар буде мати свою вартість і наявну кількість.

При визначенні атрибутів і операцій варто дотримуватися єдиного стилю. Так, наприклад, ідентифікатор варто позначати двома буквами й починати з малої літери, а назву із декількох слів писати разом, розділяючи великими буквами – “ідКористувача”.

Якщо об'єкт не має потреби в атрибутах і (або) операціях, варто придивитися до визначення класу. Це може свідчити про те, що клас не є самодостатнім і тому підлягає об'єднанню з яким або іншим класом. З іншого боку, надлишок даних або їхня суперечливість також неприпустимі. Варто пам'ятати, що клас являє собою яку-небудь *одну основну сутність*.

Кожна операція повинна йменуватися в термінах того класу, що її реалізує, а не класу, що запитує необхідну функцію. Найменування операції не повинне відображати особливості її реалізації. Наприклад, якщо назвати операцію як “обчислити...”, то подібний ідентифікатор прямо свідчить про те, що застосовується деяке обчислення. Однак у процесі зміни програми необхідність в обчисленні може відпасти. Більше вдалим була б назва операції “одержати...”.

Варто також урахувати, що не всі повідомлення на діаграмах послідовностей перетворюються в операції. Так, деякі повідомлення, спрямовані граничному класу, можуть бути реалізовані у вигляді елемента графічного інтерфейсу користувача, наприклад, у діалоговому вікні.

При створенні атрибутів варто звернути увагу на результати аналізу предметної області. Відтворення атрибутів у діаграмах класів призначено для фіксації структури й поводження класів. При виконанні індивідуальних завдань тип видимості атрибута можна не позначати.

Зв'язок класів, у свою чергу, може мати власну структуру й поводженням, оскільки іноді інформація ставиться до пари об'єктів, а не до кожного окремого об'єкта. Так, наприклад, оцінки студентів повинні належати не тільки конкретному студентові, у якого вони різні по різних дисциплінах, але й зберігатися у відомостях дисциплін, оскільки оцінки студентів навіть по одній дисципліні також різні.

Ім'я класу для більшої наочності й розуміння можна задавати на російській або українській мові, записуючи їхніми символами кирилиці із пробілами. До ім'я класу бажано додати стереотип. Стереотип уписується в секцію ім'я класу в подвійних кутових дужках:

- <<entity>> – сутність;
- <<boundary>> – граничний клас;
- <<interface>> – інтерфейс;
- <<control>> – керування;
- <<utility>> – прикладний клас;
- <<exception>> – виключення.

На рисунках 6.1-6.2 наведені зразки діаграм класів для книжкового Internet-магазина [2]. Варто звернути увагу на те, що класи системи створюють необхідну з точки зору Прецедентів поведінку та забезпечують всіх діючих осіб однозначною інформацією.

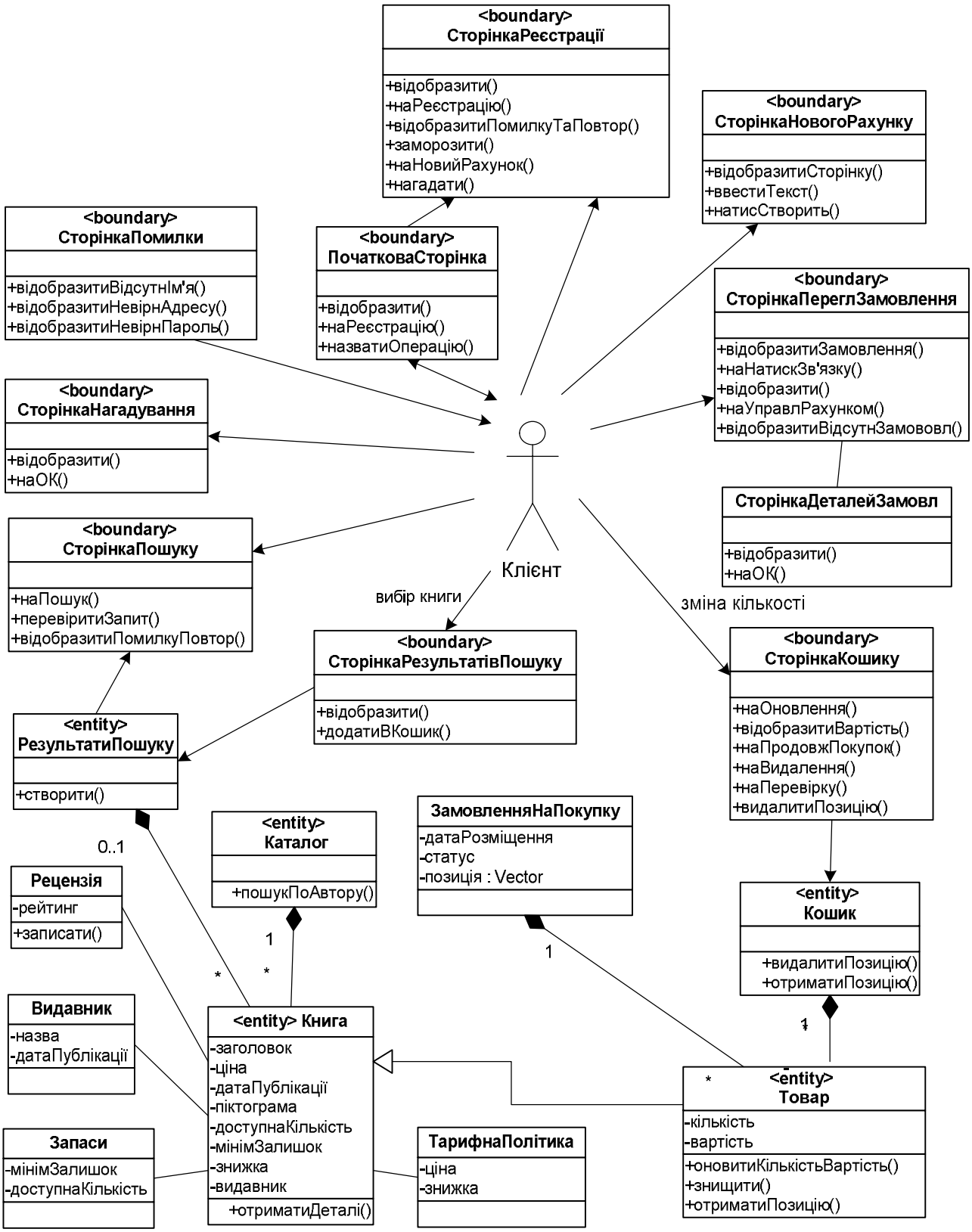

Рисунок 6.1 – Діаграма класів книжкового Internet-магазину з точки зору Прецедентів для Клієнта магазину

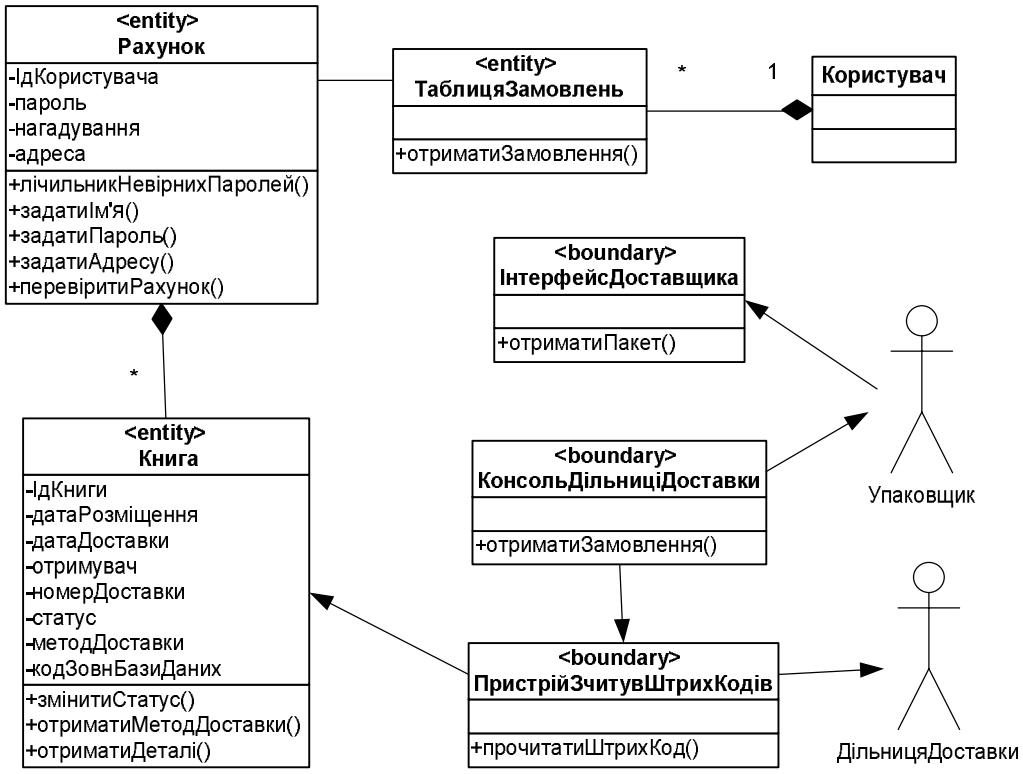

Рисунок 6.3 – Діаграма класів з точки зору Прецедентів для дільниць Упаковки та Доставки

ДОДАТОК А

ПРИБЛИЗНА ТЕМАТИКА ІНДИВІДУАЛЬНИХ ЗАВДАНЬ

1 Торговельний автомат

Потрібно розробити модель програмного забезпечення убудованого процесора універсального торговельного автомата.

В автоматі є п'ять лотків для зберігання й видачі товарів. Завантаження товарів на лотки здійснюються обслуговуючим персоналом. Автомат стежить за наявністю товару. Якщо який-небудь товар розпроданий, автомат відправляє повідомлення про це на станцію обслуговування й інформує покупців (запалюється червоний індикатор поруч із лотком даного товару).

Автомат приймає до оплати паперові купюри й монети. Спеціальний індикатор показує поточну суму грошей, прийнятих автоматом до оплати. Після уведення грошей клієнт натискає на кнопку видачі товару. Видача товару здійснюється тільки в тому випадку, якщо уведена сума грошей відповідає ціні товару. Товар видається поштучно. При натисканні на кнопку «Повернення» клієнтові вертаються всі прийняті від нього до оплати гроші. Повернення грошей не виконується після видачі товару. Автомат повинен коректно працювати при одночасному натисканні на кнопки видачі товару й повернення грошей.

У спеціальному відділенні автомата, що закривається замком, є «секретна кнопка», що використовується обслуговуючим персоналом для виїмки виторгу. При натисканні на цю кнопку відкривається доступ до ящика із грошима.

Автомат одержує зі станції обслуговування дані про товари й зберігає їх у своїй пам'яті. Дані містять у собі ціну, найменування товару, номер лотка, на якому перебуває товар і кількість товару на лотку.

2 Система автоматизації для пункту прокату відеокасет

Потрібно розробити модель програмної системи для автоматизації роботи пункту прокату відеокасет.

Пункт прокату містить каталог касет, які є в наявності в цей момент часу. Система підтримує роботу каталогу, дозволяючи службовцям прокату додавати нові найменування касет, видаляти старі й редагувати дані про касети.

Клієнт, що звернувся в пункт, вибирає касету по каталозі, вносить заставу й забирає її на певний строк. Строк прокату, вимірюваний у добі, обмовляється при видачі касети. Вартість прокату обчислюється системою виходячи з тарифу за добу й строку прокату. Клієнт повертає касету й

оплачує прокат. Якщо касета не ушкоджена, клієнтові вертається застава. Службовець пункту прокату реєструє здачу касети клієнтові і її повернення в системі. Якщо клієнт ушкодив касету, то касета видаляється з каталогу, а застава залишається в касі прокату. При необхідності службовець може запросити в системи наступні дані:

– чи є в наявності касета з даною назвою;
– коли буде повернута яка-небудь касета з тих, що здано в прокат;
– чи є даний клієнт постійним клієнтом пункту прокату (чи користувався прокатом 5 або більше разів).

Постійним клієнтам надаються знижки, а також від них приймаються заявки на поповнення асортиментів касет. Заявки реєструються в системі. По них готується підсумковий звіт, керуючись яким, пункт прокату обновляє асортимент касет.

3 Пральна машина

Потрібно розробити модель програмного забезпечення убудованого мікропроцесора пральної машини.

Машина призначена для автоматичного прання білизни. Машина містить у собі наступні пристрої: бак для білизни, клапани для забору й зливу води, мотор, пристрій підігріву води, термометр, таймер, дверцята для доступу в бак, кілька ємностей для різних мийних засобів, панель керування із кнопками й індикатором. У пам'яті машини зберігаються 5 програм прання, задані виготовлювачем. Користувачі не можуть вносити в них зміни. Кожна програма визначає температуру води, тривалість прання, використовувані мийні засоби (номер ємності й час подачі), швидкість обертання бака під час прання та під час віджима.

Для використання машини необхідно відкрити дверцята, помістити білизну в бак, помістити мийні засоби в ємності, закрити дверцята, вибрати програму прання й натиснути на кнопку «Пуск». Перед тим як приступити до прання машина відкриває клапан для забору води, набирає необхідну кількість води, після чого закриває клапан. Далі, машина діє по обраній користувачем програмі: підігріває, якщо необхідно, воду до потрібної температури, включає таймер і запускає обертання бака для прання. Мийні засоби подаються в бак по сигналам таймера, що передбачені програмою. По закінченні прання вода зливається із баку й запускається процес віджиму.

Поточний час роботи машини, режим роботи (прання або віджим), номер поточної програми прання виводяться на індикатори. З метою безпеки дверцята бака блокуються до закінчення прання. Машина не сприймає натискань на кнопки, за винятком однієї – користувач має можливість у будь-який момент натиснути на кнопку «Зупинка», щоб примусово зупинити прання й злити воду.

4 Таксофон

Потрібно розробити модель убудованої системи керування роботою таксофона міської телефонної мережі.

Таксофон призначений для надання платних послуг телефонного зв'язку. Він підключений до лінії зв'язку. У ньому є кнопкова панель, дисплей, трубка з убудованим мікрофоном та гучномовцем, приймач карт, пристрій для зчитування телефонних карт, використовуваних для оплати розмови.

У початковому стані трубка таксофона повішена, дисплей погашений, таксофон не реагує на натискання кнопок і які-небудь сигнали з лінії. При знятті трубки таксофон видає на дисплей повідомлення «Вставте карту» і очікує, коли користувач вставить карту в приймач.

Подальше функціонування таксофона здійснюється тільки при вставленій карті. Якщо карту виймають, таксофон вертається до початку й видає повідомлення про необхідність вставити карту. При введенні карти в приймач здійснюється зчитування інформації з карти. Якщо кредит вичерпаний або карта не придатна (не вдається довідатися кредит), то таксофон видає відповідне повідомлення на дисплей таксофона. Якщо карта може бути використана для оплати, то на дисплей видається кількість «одиниць» на карті, і на телефонну станцію (АТС) подається сигнал «Трубка». При одержанні з лінії відповідного сигналу «Тон» таксофон відтворює звуковий тон «Готовий» (довгий гудок, що не припиняється) у трубку. При одержанні сигналу «Зайняте», у трубці відтворюється тон «Зайняте» (короткі гудки).

Після одержання від АТС сигналу «Тон» від користувача приймаються номер викликуваного абонента у відповідному до системи форматі (5 або 6 цифр), інші натискання на кнопки ігноруються. Коли користувач натискає на кнопку із цифрою відповідний їй сигнал «Цифра» передається АТС. Під час набору номера уведені цифри відображаються на дисплеї. У відповідь на набраний номер від АТС приходить або сигнал «Зайняте», або сигнал «Виклик». При одержанні сигналу «Виклик» таксофон відтворює в трубку довгі гудки до того моменту, коли АТС здійснить комутацію й передасть сигнал «Дані». Таксофон відтворює дані, передані із сигналом, у трубку. При одержанні даних із трубки, апарат перетворить їх у сигнал «Дані» і передає їх АТС. Під час розмови на дисплеї ведеться відлік часу й зменшується кредит на телефонній карті – кожні 15 секунд віднімається чверть «одиниці». Обмін даними переривається в наступних випадках:

вичерпаний кредит; карта вийнята із приймача; від АТС прийшов сигнал «Зайняте»; повішена трубка таксофона.

Якщо трубка була повішена, апарат посилає в лінію сигнал «Кінець» і видає на дисплей повідомлення «Вийміть карту». Після видалення карти із приймача таксофон переходить у початковий стан.

5 Банкомат

Потрібно розробити модель програмного забезпечення банкомату. Банкомат – це автомат для видачі готівки по кредитних пластикових картках. У його склад входять наступні пристрої: дисплей, панель керування із кнопками, приймач кредитних карт, сховище грошей і лоток для їхньої видачі, сховище конфіскованих кредитних карт, принтер для печатки довідок.

Банкомат підключений до лінії зв'язку для обміну даних з банківським комп'ютером, що зберігає відомості про рахунки клієнтів.

Обслуговування клієнта починається з моменту введення пластикової картки в банкомат. Після розпізнавання типу пластикової картки, банкомат видає на дисплей запрошення ввести персональний код. Персональний код являє собою чотиризначне число. Потім банкомат перевіряє правильність уведеного коду. Якщо код зазначений невірно, користувачеві надаються ще дві спроби для уведення правильного коду. У випадку повторних невдач карта переміщається в сховище карт, і сеанс обслуговування закінчується. Після уведення правильного коду банкомат пропонує користувачеві вибрати операцію. Клієнт може або зняти готівку з рахунку, або довідатися про залишок на його рахунку.

При знятті готівки з рахунку банкомат пропонує вказати суму (10, 50, 100, 200, 500, 1000 грн). Після вибору клієнтом суми банкомат запитує, чи потрібно друкувати довідку по операції. Потім банкомат надсилає запит на зняття обраної суми центральному комп'ютеру банку. У випадку одержання дозволу на операцію, банкомат перевіряє, чи є необхідна сума в його сховищі грошей. Якщо він може видати гроші, то на дисплей виводиться повідомлення «Вийміть карту». Після видалення картки із приймача, банкомат видає зазначену суму в лоток видачі. Банкомат друкує довідку по зробленій операції, якщо вона була викликана клієнтом.

Якщо клієнт хоче довідатися про залишок на рахунку, то банкомат надсилає запит центральному комп'ютеру банку й виводить суму на дисплей. На вимогу клієнта друкується й видається відповідна довідка.

У спеціальному відділенні банкомату, що закривається замком, є «секретна кнопка», що використовується обслуговуючим персоналом для завантаження грошей. При натисканні на цю кнопку відкривається доступ до сховища грошей і конфіскованих кредитних карт.

6 Холодильник

Потрібно розробити модель програмного забезпечення убудованого процесора холодильника.

Холодильник складається з декількох холодильних камер для зберігання продуктів. У кожній холодильній камері є регулятор температури, мотор, термометр, індикатор, таймер, датчик відкриття двері камери й пристрій для подачі звукових сигналів.

За допомогою терморегулятора встановлюється максимально припустима температура в даній камері. Мотор призначений для підтримки низької температури. Термометр постійно вимірює температуру усередині камери, а індикатор температури, розташований на дверцятах, постійно показує її значення. При підвищенні температури вище межі, обумовленій поточним положенням регулятора, включається мотор. При зниженні температури нижче деякого іншого значення, пов'язаного з першим, мотор відключається. Доступ у камеру здійснюється через дверцята. Якщо двері холодильної камери відкриті протягом занадто довгого часу, подається звуковий сигнал. Звуковий сигнал також подається в будь-яких позаштатних ситуаціях (наприклад, при поломці мотора).

Холодильник веде електронний журнал, у якому відзначаються всі події, що відбуваються:

- зміна положення терморегулятора камери;
- включення й відключення мотора;
- доступ у камеру;
- позаштатні ситуації.

Уміст журналу може бути переданий в комп'ютер, приєднаний до спеціального гнізда холодильника.

7 Система обліку товарів

Потрібно розробити модель системи підтримки замовлення й обліку товарів у магазині.

У магазині для кожного товару фіксується місце зберігання (певна полиця), кількість товару і його постачальник. Система підтримки замовлення й обліку товарів повинна забезпечувати додавання інформації про новий товар, зміну або видалення інформації про наявний товар, зберігання (додавання, зміна й видалення) інформації про постачальників, що включає в себе назву фірми, її адресу й телефон. За допомогою системи складаються замовлення постачальникам. Кожне замовлення може містити кілька позицій, у кожній позиції вказуються найменування товару і його кількість у замовленні. Система обліку на вимогу користувача формує й видає на печатку наступну довідкову інформацію:

– список товарів, що є в наявності;
– список товарів, кількість яких необхідно поповнити;
– список товарів, що поставляються даним постачальником.

8 Бібліотечна система

Потрібно розробити модель системи, що автоматизує діяльність бібліотеки.

Система підтримки керування бібліотекою повинна забезпечувати операції (додавання, видалення й зміна) над даними про читачів. У реєстраційному списку читачів зберігаються наступні відомості: прізвище, ім'я та по батькові читача; номер його читацького квитка й дата видачі квитка. Поряд з реєстраційним списком системою повинен підтримуватися каталог бібліотеки, де зберігається інформація про книги: назва, список авторів, бібліотечний шифр, рік і місце видання, назва видавництва, загальна кількість екземплярів книги в бібліотеці й кількість екземплярів, доступних у сучасний момент.

Система забезпечує додавання, видалення й зміну даних каталогу, а також пошук книг у каталозі на підставі уведеного шифру або назви книги. У системі здійснюється реєстрація взятих і повернутих читачем книг. Про кожну видану книгу зберігається запис про те, кому й коли була видана книга, і коли вона буде повернута. При поверненні книги в записі робиться відповідна позначка, а сам запис не віддаляється із системи.

Система повинна видавати наступну довідкову інформацію:

– які книги були видані за даний проміжок часу;
– які книги були повернуті за даний проміжок часу;
– які книги перебувають у даного читача;
– чи є в наявності деяка книга.

ЛІТЕРАТУРА

Основная литература

1. Буч Г., Рамбо Дж., Джекобсон А. Язык UML. Руководство пользователя: Пер. с англ. – М.: ДМК, 2000.
2. Розенберг Д., Скотт К. Применение объектного моделирования с использованием UML и анализ прецедентов: Пер. с англ. – М.: ДМК Пресс, 2002. – 160 с.

Дополнительная литература

3. Буч Г. Объектно-ориентированное проектирование с примерами применения: Пер. с англ. – М.: Конкорд, 1992. – 519 с.
4. Леоненков А. В. Объектно-ориентированный анализ и проектирование с использованием UML и IBM Rational Rose: Учебное пособие / –М.: Интернет-Университет Информационных Технологий; БИНОМ. Лаборатория знаний, 2006. – 320 с.
5. Орлов С. А. Технология разработки программного обеспечения: Учебник для вузов. 3-е изд. / С. А. Орлов. – СПб.: Питер, 2004. – 527 с.
6. Бенькович Е. С., Колесов Ю. Б., Сениченков Ю. Б. Практическое моделирование динамических систем – СПб.: БХВ-Петербург, 2002. – 464 с.

ЗМІСТ

МЕТОДИЧНІ ВКАЗІВКИ

до самостійної роботи з дисципліни

«ТЕХНОЛОГІЯ ПРОГРАМУВАННЯ СКЛАДНИХ СИСТЕМ»

для студентів спеціальності 7.092501 «Автоматизоване управління технологічними процесами»