

**Міністерство освіти і науки України
Донбаська державна машинобудівна академія**

О.Ю.Мельников

**ОБ'ЄКТНО-ОРІЄНТОВАНИЙ АНАЛІЗ
І ПРОЕКТУВАННЯ
ІНФОРМАЦІЙНИХ СИСТЕМ**

НАВЧАЛЬНИЙ ПОСІБНИК

для студентів спеціальностей
«Економічна кібернетика»
та

«Інтелектуальні системи прийняття рішень»

Краматорськ 2007

УДК 004:681
ББК 32.973-01
М48

Рецензенти:

В.К.ГАЛІЩИН, д-р екон. наук, професор, завідувачий кафедрою Київського національного економічного університету

М.М.ЛЕПА, д-р. екон. наук, професор, директор Науково-дослідного центру інформаційних технологій Інституту економіки промисловості Національної академії наук України

К.А.ЮРЧЕНКО, канд. техн. наук, доцент, доцент кафедри Дніпропетровського національного університету

Рекомендовано
Міністерством освіти і науки України
Лист №1.4-Г-1752 від 19.10.2007

Мельников О. Ю.

М48 Об'єктно-орієнтований аналіз і проектування інформаційних систем: Навчальний посібник для студентів спеціальностей «Економічна кібернетика» і «Інтелектуальні системи прийняття рішень». – Краматорськ: ДДМА, 2007. – 176 с.
ISBN 978-966-379-204-0

Навчальний посібник містить основні теоретичні відомості про об'єктно-орієнтований аналіз і проектування систем на уніфікованій мові моделювання UML, порядок роботи у середовищі IBM Rational Rose і приклади створення моделей систем у цьому середовищі.

Рекомендується студентам спеціальностей 7.050102 «Економічна кібернетика» і 7.080404 «Інтелектуальні системи прийняття рішень» як навчальний посібник при вивченні відповідних модулів курсів «Об'єктно-орієнтоване програмування», «Проектування інформаційних систем», «Системний аналіз і проектування систем обробки інформації» і «Технологія програмування і створення програмних продуктів», а також як довідник під час курсового і дипломного проектування.

УДК 004:681
ББК 32.973-01

ISBN 978-966-379-204-0

© О.Ю.Мельников, 2007
© ДДМА, 2007

ЗМІСТ

Вступ.....	5
1 Основні поняття об'єктно-орієнтованого підходу.....	6
1.1 Об'єктна модель.....	6
1.2 Класи і об'єкти.....	10
1.3 Класифікація.....	14
2 Уніфікована мова моделювання UML як засіб проектування програмних систем і бізнес-процесів.....	16
2.1 Передісторія, основи і структура UML.....	17
2.2 Діаграма концептуального моделювання – діаграма варіантів використання (use case diagram).....	22
2.3 Діаграми логічного моделювання.....	29
2.3.1 Діаграма класів (class diagram).....	29
2.3.2 Діаграма кооперації (collaboration diagram).....	36
2.3.3 Діаграма послідовності (sequence diagram).....	42
2.3.4 Діаграма станів (statechart diagram).....	46
2.3.5 Діаграма діяльності (activity diagram).....	52
2.4 Діаграми фізичного моделювання.....	57
2.4.1 Діаграма компонентів (component diagram).....	57
2.4.2 Діаграма розгортання (deployment diagram).....	60
3 Проектування програмних систем з використанням CASE-засобу IBM Rational Rose.....	63
3.1 Загальна характеристика інструментарію IBM Rational Rose.....	64
3.2 Приклад розробки моделі інформаційної системи у середовищі IBM Rational Rose.....	66
3.3 Генерація коду спроектованої моделі в середовищі програмування.....	96
4 Приклади проектування інформаційних систем.....	108
4.1 Інформаційна система для функціонування кадрового агентства.....	108
4.2 Інформаційна система для автоматизованого складання розкла-	

ду занять у вищому навчальному закладі.....	114
4.3 Інформаційна система для спеціалізованого торгівельного підприємства.....	123
4.4 Інформаційна система для невеликої страхової компанії.....	130
4.5 Інформаційна система для забезпечення функціонування фінансового відділу підприємства.....	139
4.6 Інформаційна система для розрахунку собівартості металопродукції.....	145
4.7 Інформаційна система для обліку і контролю готової продукції..	155
4.8 Інформаційна система для маркетингових досліджень і аналізу надійності.....	163
Список літератури.....	172

ВСТУП

Складність сучасного програмного забезпечення і висока вартість його розробки і супроводу викликали виникнення нового – об'єктно-орієнтованого – підходу до створення інформаційних систем. При цьому через участь у розробці цих систем десятків і сотень різних фахівців виникла необхідність побудови попередньої моделі системи до початку написання відповідного програмного коду. Основна вимога до такої моделі – бути зрозумілою як замовнику програмної системи, так і всім фахівцям-розробникам (системним аналітикам, програмістам і т.п.).

У даний час у світі поширені три нотації візуального моделювання: IDEF (Icam DEFinition), ARIS (Architecture of Integrated Information Systems) і UML (Unified Modelling Language). Перші дві застосовуються найчастіше при моделюванні бізнес-процесів. Розгляду третього, який став, по суті, світовим стандартом розробки програмного забезпечення, і присвячений даний навчальний посібник.

Посібник складається з чотирьох частин – двох розділів теоретичної спрямованості і двох – практичної. Спочатку в конспективній формі висловлюються концепції об'єктно-орієнтованого підходу до аналізу і проектування – поняття і визначення об'єктної моделі, класів і об'єктів і їх можливих класифікацій (за докладнішою інформацією слід звертатися до «класичної» праці Граді Буча [1]). Потім детально описується мова проектування програмних систем UML (базовим підручником прийнята праця Олександра Леоненкова [2], яка, на відміну від ряду перекладних видань [3-7], максимально зручно структурована і припускає шлях «від простого до складного»).

Третій розділ присвячений опису CASE-засобу проектування програмних систем «IBM Rational Rose», четвертий містить приклади спроектованих систем (усі наведені моделі розроблені студентами спеціальності «Економічна кібернетика» у межах курсового і дипломного проектування під керівництвом або з участю автора посібника).

1 ОСНОВНІ ПОНЯТТЯ

ОБ'ЄКТНО-ОРІЄНТОВАНОГО ПІДХОДУ

Усі методи проектування програмних систем можна об'єднати у три групи:

- структурне проектування зверху вниз, в основі якого лежить алгоритмічна декомпозиція, і яке не може забезпечити створення складних систем і неефективне в об'єктно-орієнтованих мовах;
- метод потоків даних, коли система розглядається як перетворювач вхідних потоків у вихідні;
- об'єктно-орієнтоване проектування.

Об'єктно-орієнтоване програмування (ООР – Object-oriented programming) – це методологія програмування, заснована на поданні програми у вигляді сукупності об'єктів, кожний з яких є екземпляром певного класу, а класи утворюють ієрархію спадкоємства. Слід розрізняти «об'єктні» («об'єктно-базовані») і «об'єктно-орієнтовані» мови програмування.

Об'єктно-орієнтоване проектування (OOD – Object-oriented design) – це методологія проектування, сполучаючи у собі процес об'єктної декомпозиції і прийоми подання логічної і фізичної, а також статичної і динамічної моделей проектованої системи.

Об'єктно-орієнтований аналіз (ООА – Object-oriented analysis) – це методологія, при якій вимоги до системи сприймаються з погляду класів і об'єктів, виявлених у наочній області.

У результаті об'єктно-орієнтованого аналізу формуються моделі, на яких ґрунтується об'єктно-орієнтоване проектування, що створює фундамент для остаточної реалізації системи з використанням методології об'єктно-орієнтованого програмування. Усі три методології базуються на поняттях *об'єктної моделі*.

1.1 Об'єктна модель

Об'єктна модель є концептуальною базою об'єктно-орієнтованого стилю. Вона складається з чотирьох головних елементів (абстрагування, інкапсуляція, модуль, ієрархія) і трьох додаткових (типізація, паралелізм, збереженність), які корисні, але не обов'язкові.

Абстрагування виділяє істотні характеристики деякого об'єкта, відрізняючи його від всіх інших видів об'єктів, і, таким чином, чітко визначає його концептуальні межі з погляду спостерігача. Головний принцип абстрагування – принцип якнайменшого здивування, згідно з яким абстракція повинна охоплювати всю поведінку об'єкта, але не привносити сюрпризів або побічних ефектів, що лежать поза її сферою застосовності.

Вибір правильного набору абстракцій – найголовніша задача.

Абстракції можна об'єднати у 4 групи.

Абстракція суті: об'єкт є корисною моделлю якоїсь суті в наочній області.

Абстракція поведінки: об'єкт складається з узагальненої безлічі операцій.

Абстракція віртуальної машини: об'єкт групує операції, які або разом використовуються вищим рівнем управління, або самі використовують деякий набір операцій нижчого рівня.

Довільна абстракція: об'єкт включає набір операцій, що не мають один з одним нічого спільного.

Приклади абстракцій в гідропонному господарстві: теплиця, температура, вогкість, освітлення, кислотність, датчики, культури, план вирощування.

Інкапсуляція – це процес відокремлення один від одного елементів об'єкта, визначаючих його будову і поведінку; служить для того, щоб ізолювати контрактні зобов'язання абстракції від їх реалізації.

Абстрагування та інкапсуляція доповнюють один одного: перше направлене на спостережувану поведінку об'єкта, а друга займається його внутрішнім устроєм. *Інтерфейс* відображає зовнішню поведінку об'єкта, описуючи абстракцію поведінки всіх об'єктів даного класу; внутрішня *реалізація* описує подання цієї абстракції і механізми досягнення бажаної поведінки об'єкта.

(Слід звернути увагу: «Інкапсуляція не рятує від дурості; вона захищає від помилок, але не від шахрайства» – Б. Страуструп).

Приклад: простий нагрівник має розташування (конструктор), вмикач, вимикач і перевірку стану. Ускладнений нагрівник може бути пов'язаний з датчиком температури і планом вирощування.

Модульна – це властивість системи розкладатися на внутрішньо зв'язні, але слабо зв'язані між собою модулі. Логічне продовження інкапсуляції, її «практична проекція».

Ієрархія – це впорядкування абстракцій, розташування їх за рівнями. Розрізняють два види ієрархічних структур: структура класів («is-a») і структура об'єктів («part of»).

Структура класів є ієрархією спадкоємства типу «узагальнення-спеціалізація», в якій підклас (нащадок) є спеціалізованим окремим випадком свого суперкласу (предка). Приклад: «токарь є окремий випадок робітника», «датчик температури є окремий випадок датчика».

Структура об'єктів є ієрархією типу «агрегація», в якій один об'єкт міститься усередині іншого. Приклад: «вікно містить смуги прокрутки, меню, рядок стану», «управляючий елемент в теплиці містить датчики температури, вологості, кислотності й освітлення».

Типізація – це спосіб захиститися від використання об'єктів одного класу замість іншого, або, принаймні, управляти таким використанням. Типізація примушує виражати абстракції так, щоб мова програмування підтримала дотримання ухвалених проектних рішень. Розрізняють сильну і слабку типізацію.

Паралелізм – це властивість, що відрізняє активні об'єкти від пасивних. Припускає паралельну (незалежну) роботу ряду активних об'єктів. При проектуванні треба вживати заходи, які гарантують, що об'єкт не буде розтерзаний на частини декількома незалежними процесами.

Збереженість – здатність об'єкта існувати у часі, переживаючи процес, що його породив, і (або) в просторі, переміщаючись зі свого первинного адресного простору.

Спектр збереженості об'єктів охоплює:

- проміжні результати обчислення виразів;
- локальні змінні в підпрограмах;
- власні (глобальні) змінні та динамічно створювані дані;
- дані, що зберігаються між сеансами виконання програми;
- дані, що зберігаються при переході на нову версію програми;
- дані, які взагалі переживають програму.

Традиційно, першими трьома рівнями займаються мови програмування, а останніми – бази даних. Можливі змішані рішення: програмісти

розробляють спеціальні схеми для збереження об'єктів у період між запусками програми, а конструктори баз даних переінакшують свою технологію під короткоживучі об'єкти.

Існують так звані об'єктно-орієнтовані бази даних, які зберігають не тільки дані, але і класи. Ця технологія не прижилася, на практиці вважають за краще створювати об'єктно-орієнтовану оболонку для реляційних баз даних.

Переваги об'єктної моделі

1 Дозволяє повною мірою використовувати можливості об'єктних і об'єктно-орієнтованих мов програмування.

2 Підвищує рівень уніфікації розробки і придатність для повторного використання не тільки програм, але і проектів. Використання попередніх розробок дає вигреш у вартості та часі.

3 Спрощує процес внесення змін через наявність стабільних проміжних описів. Це дає можливість не переробляти систему цілком навіть у разі істотних змін початкових вимог.

4 Зменшує вірогідність допущення різноманітних помилок.

5 Орієнтована на людське сприйняття світу.

Деякі факти з історії

1969 – Кей (Kay) у своїй дисертації запропонував ідею об'єктного підходу;

1979 – Джонс (Jones) ввів концепцію об'єктного підходу;

1981 – Буч (Booch) запропонував методи OOD;

1988 – Шлеєр і Меллор (Shlaer and Mellor) запропонували методи OOA;

1988 – Страуструп (Stroustrup) опублікував методичну статтю про OOP;

1991 – Румбах (Rumbaugh) об'єднав OOA і OOD.

Організації, що відповідають за стандарти за об'єктною технологією: Object Management Group (OMG) і комітет ANSI X3J7.

1.2 Класи і об'єкти

Одними їх базових понять об'єктної моделі є поняття класів і об'єктів.

Об'єкт моделює частину навколишньої дійсності і, таким чином, існує в часі та просторі. Об'єкт володіє станом, поведінкою й ідентичністю; структура і поведінка схожих об'єктів визначає загальний для них клас; терміни «екземпляр класу» і «об'єкт» взаємозамінні.

Стан об'єкта характеризується переліком (звично статичним) всіх властивостей даного об'єкта і поточними (звично динамічними) значеннями кожної з цих властивостей. Приклад: автомат з продажу напоїв.

Поведінка визначає сукупність дій і реакцій об'єкта; виражається в термінах стану об'єкта і передачі повідомлень. Іншими словами, поведінка об'єкта – це спостережувана і перевіряльна ззовні діяльність. Стан об'єкта є сумарним результатом його поведінки.

Операцією називається певна дія одного об'єкта на інший з метою викликання відповідної реакції; це послуга, яку клас може надати своїм клієнтам. Типовий клієнт може скоювати операції 5 видів:

Модифікатор – операція зміни стану об'єкта.

Селектор – операція, що прочитує стан об'єкта, але не змінюючи його.

Ітератор – операція, що дозволяє організувати доступ до всіх частин об'єкта у точній послідовності.

Конструктор – операція створення об'єкта і/або його ініціалізації.

Деструкція – операція очищення і/або руйнування об'єкта.

Сукупність всіх методів і вільних процедур, що відносяться до конкретного об'єкта, утворює *протокол* цього об'єкта.

Об'єкти можуть бути активними і пасивними. Активний об'єкт може проявляти свою поведінку без дії з боку інших об'єктів. Пасивний об'єкт, навпаки, може змінювати свій стан тільки під впливом інших об'єктів. Таким чином, активні об'єкти системи – джерела управляючих дій.

Ідентичність – це така властивість об'єкта, яка відрізняє його від всіх інших об'єктів. Слід уміти відрізнити ім'я об'єкта від самого об'єкта. Якщо абстракція є зборами властивостей без власної поведінки, це не об'єкт, а структура.

Система реалізується тільки у процесі взаємодії об'єктів. Відносини між об'єктами можуть бути двох типів: зв'язок і агрегація.

Зв'язок – це специфічне зіставлення, через яке клієнт запрошує послугу у об'єкта-сервера або через яке один об'єкт знаходить шлях до іншого.

Беручи участь у зв'язку, об'єкт може виконувати одну з трьох ролей:

актор: може впливати на інші об'єкти, але сам ніколи не піддається дії інших об'єктів (виключно активний об'єкт);

сервер: може тільки піддаватися дії з боку інших об'єктів, але сам ніколи не виступає в ролі впливаючого об'єкта (виключно пасивний об'єкт);

агент: може виступати як в активній, так і в пасивній ролі.

На рисунку 1 зображені чотири об'єкти, кожний з яких характеризує описані вище ролі.

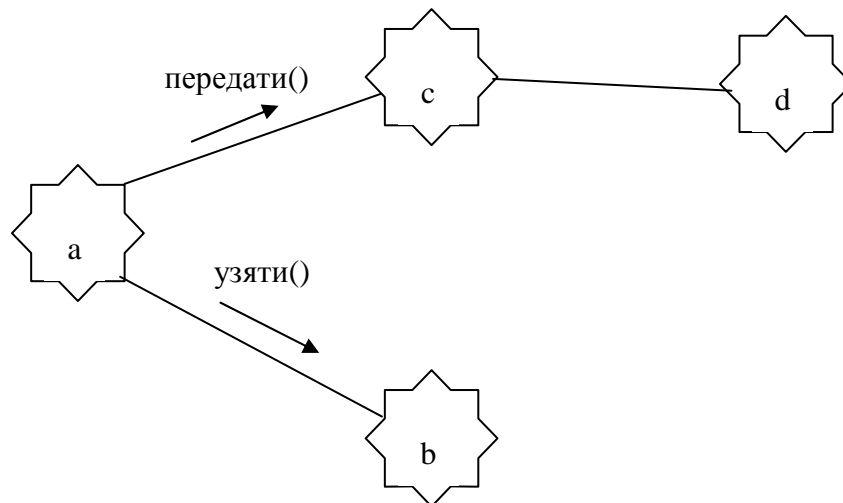


Рисунок 1 – Взаємодіючі об'єкти

Агрегація – спеціалізований окремий випадок асоціації; описує відносини цілого і частини, що приводять до відповідної ієрархії об'єктів.

Агрегація може означати фізичне входження одного об'єкта в інший (автомобіль і двигун), але не обов'язково (акціонер і акції). Приклад агрегації з фізичним входженням поданий на рисунку 2.

Клас – це якась безліч об'єктів, які мають загальну структуру і загальну поведінку. Тоді як об'єкт позначає конкретну суть, клас визначає лише абстракцію істотного в об'єкті. Будь-який конкретний об'єкт є просто екземпляром класу.

Підтримуються шість видів відносин між класами.



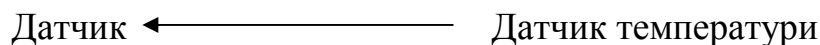
Рисунок 2 – Агрегація

Асоціація – смисловий зв'язок (за умовчанням – двостороння), що фіксує учасників, їх ролі і потужність відношення (1:1, 1:*, *:*)).



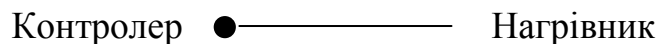
Спадкоємство – таке відношення між класами, коли один клас повторює структуру і поведінку іншого або інших класів. Встановлює відношення загального і приватного. Класи, екземпляри яких створюються, називаються конкретними, не створюються – абстрактними. Висновки: у будь-якого класу може бути два види клієнтів – екземпляри (об'єкти) і підкласи (спадкоємці).

Зображається у вигляді стрілки, направленої від нащадка до предка:



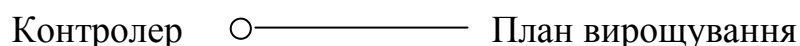
Агрегація між класами має той же сенс, що і у разі об'єктів. Розрізняють агрегацію за *значенням* (фізичне включення) і за *посиланням*. Якщо є сумніви у виборі між спадкоємством і агрегацією, краще вибрати другу.

Зображається у вигляді лінії із зафарбованим колом:



Відношення **використання** між класами відповідає зв'язку між їх екземплярами (клієнт-серверні відносини).

Зображається у вигляді лінії з порожнім колом:



Інстанціювання припускає використання класів, що параметризуються, коли як формальний параметр класу передається показник на який-небудь клас, а при зверненні до його екземпляра формальний параметр за-

міщається фактичним. Є продовженням (або окремим випадком) відношення використання.

Зображається у вигляді прямокутника-врізки у правому верхньому кутку класу.

Метаклас – це клас, екземпляри якого є класами.

Зображається у вигляді замальованного класу.

На етапі аналізу і ранніх стадіях проектування розв'язуються дві основні задачі:

- виявлення основних класів і об'єктів, що становлять словник наочної області;
- побудова структур, що забезпечують взаємодію об'єктів, при якому виконуються вимоги задачі.

Для оцінки якості класів і об'єктів, що виділяються у системі, можна запропонувати наступні п'ять критеріїв: зачеплення, зв'язність, достатність, повнота і примітивність.

Зачеплення визначає ступінь глибини зв'язків між окремими модулями (чим менше, тим краще). Приклад неправильного підходу: джерело живлення стереосистеми розміщене в одній з колонок.

Зв'язність – це ступінь взаємодії між елементами окремого модуля (класу, об'єкта), характеристика його насиченості: «Клас Dog буде функціональний зв'язковим, якщо він описує поведінку собаки, всього собаки, і нічого, окрім собаки».

Достатність – наявність у класі або модулі всього необхідного для реалізації логічної і ефективної поведінки.

Під **повнотою** мається на увазі наявність в інтерфейсній частині класу всіх характеристик абстракції; інтерфейс повинен гарантувати все для взаємодії з користувачами.

Примітивними є тільки такі операції, які вимагають доступу до внутрішньої реалізації абстракції

1.3 Класифікація

Класифікація – це засіб впорядкування знань. Історично відомі тільки *три* підходи: класична категоризація, концептуальна кластеризація, теорія прототипів.

При *класичному підході* всі речі, що володіють даною властивістю або сукупністю властивостей, формують деяку категорію. Причому наявність цих властивостей є необхідною і достатньою умовою, що визначає категорію.

Проте природні категорії нечітко відокремлені один від одного. Більшість птахів літає, але не всі. Стілець може бути дерев'яним, металевим або пластмасовим, а кількість ніжок необов'язково дорівнює чотирьом.

Концептуальна кластеризація – сучасний варіант класичного підходу. Тут спочатку формуються концептуальні описи класів («кластерів об'єктів»), а потім ми класифікуємо суть відповідно до їх описів. Це саме поняття, а не ознака або властивість: «Якщо пісня скоріше про любов, ніж про щось інше, то ми поміщаємо її до категорії *любовна пісня*, хоча ступінь любовності навряд чи можна зміряти».

Концептуальну кластеризацію можна пов'язати з теорією нечітких (багатозначних) множин, в якій об'єкт може належати до декількох категорій одночасно з різним ступенем точності.

Проте існують деякі абстракції, які не мають ні чітких властивостей, ні чіткого визначення. *Теорія прототипів* визначає клас одним об'єктом-прототипом; новий об'єкт відноситься до класу за умови, що він наділений істотною схожістю з прототипом. Приклад: ігри.

На практиці ми спочатку ідентифікуємо класи і об'єкти за властивостями, важливими у даній ситуації, тобто прагнемо зазначити і відібрати структури і типи поведінки за допомогою словника наочної області. Якщо таким шляхом не вдалося побудувати нормальної структури класів, ми пробуємо концептуальний підхід: надаємо увагу поведінці об'єктів. Нарешті, пробуємо зазначити прототипи і асоціювати з ними об'єкти. Ці три способи класифікації складають теоретичну основу об'єктно-орієнтованого аналізу.

Межі між стадіями аналізу і проектування розмиті, проте під час аналізу ми моделюємо проблему, *знаходячи* класи і об'єкти, які складають

словник наочної області, тоді як на стадії проектування ми *винаходимо* абстракції і механізми, що забезпечують необхідну поведінку.

Існує *сім* перевірених практикою підходів до аналізу об'єктно-орієнтованих систем.

Класичні підходи – спираються на класичну категоризацію.

Шлаєр і Меллор: помітні предмети (датчики), ролі (робітник, студент), події (запит, переривання), взаємодія (зустріч, перетин).

Рос (БД): люди (виконують певні функції), місця (області, пов'язані з людьми або предметами), предмети (помітний матеріальний об'єкт), організації (формально організована сукупність людей і предметів, яка має певну мету і не залежить від окремих індивідуумів), концепції (принципи і ідеї), події.

Коад і Йордан: структури (відносини «ціле-частина» і «загальне-приватне»), інші системи (зовнішні), пристрої, події, ролі, місця, організаційні одиниці.

При **аналізі поведінки** саме динамічна поведінка розглядається як першоджерело класів і об'єктів. Ініціатори певної поведінки і його учасники пізнаються як об'єкти і стають відповідальними за певні ролі. Приклади: введення/висновок, запит.

Аналіз наочної області – спроба зазначити ті об'єкти, операції і зв'язки, які експерти даної області вважають найважливішими.

Етапи: побудова скелетної моделі наочної області; вивчення існуючих у даній області систем; визначення схожості і відмінностей між системами; уточнення загальної моделі для пристосування до потреб конкретної системи.

Приклад: наочна область – бухгалтерські звіти; визначаються абстракції і механізми, обслуговуючі всі види звітів; мета початкової задачі – створення системи звітів.

Експерт – майбутній користувач системи (бухгалтер, диспетчер).

Окремо всі вищеперелічені підходи сильно залежать від індивідуальних здібностей і досвіду аналітика. **Аналіз варіантів** передбачає впорядковане послідовне їх використання. Заснований на переліку і ретельному опрацюванні сценаріїв роботи системи.

Метод CRC-карток вдало реалізує на практиці попередній підхід (Class / Responsibilities / Collaborators – Клас / Відповідальності / Учасни-

ки). Зверху на картці пишеться назва класу, знизу в лівій половині – за що він відповідає, в правій – з ким він співробітничает. Прохід за сценарієм приводить до дописування нових пунктів в існуючих картках або до створення нових. Розташування карток може показати потік повідомлень між об'єктами та ієрархію класів.

Неформальний опис – радикальна альтернатива класичному аналізу (Аббот). Треба описати задачу або її частину на звичній (людський) мові, а потім підкреслити іменники і дієслова. Іменники – кандидати на роль класів, а дієслова можуть стати іменами операцій. Метод можна автоматизувати. Використовується в Токійському технологічному інституті та в Fujitsu.

Друга альтернатива класичній техніці – використання **структурного аналізу** як основи об'єктно-орієнтованого проектування. Після його проведення ми вже маємо модель системи, описану діаграмами потоків даних; виходячи з цього, можна починати визначення класів і об'єктів будь-якими іншими способами.

2 УНІФІКОВАНА МОВА МОДЕЛЮВАННЯ UML ЯК ЗАСІБ ПРОЕКТУВАННЯ ПРОГРАМНИХ СИСТЕМ І БІЗНЕС-ПРОЦЕСІВ

Уніфікована мова моделювання UML (Unified Modeling Language) призначена для опису, візуалізації і документування об'єктно-орієнтованих програмних систем і бізнес-процесів з орієнтацією на їх подальшу реалізацію у вигляді програмного забезпечення.

Використовується у наступних областях:

- інформаційні системи підприємств;
- банківська і фінансова діяльність;
- телекомунікації;
- транспорт;
- авіація і космонавтика;
- торгівля;
- розподілені Web-системи.

Ми розглядатимемо UML як засіб проектування і документування при розробці інформаційних систем (як у вигляді нових програмних про-

дуктів, так і у вигляді комп'ютерного забезпечення бізнес-процесів) для промислових і торгових підприємств, а також банківських і освітніх установ.

2.1 Передісторія, етапи розвитку і загальна структура UML

Можна зазначити методологічні і математичні основи UML. Крім того, UML базується на діаграмах структурного системного аналізу.

Методологічні основи UML

- методологія процедурно-орієнтованого програмування;
- методологія об'єктно-орієнтованого програмування;
- методологія об'єктно-орієнтованого аналізу і проектування;
- методологія системного аналізу і системного моделювання

Математичні основи UML

- теорія множин (діаграми Венна);
- теорія графів;
- семантичні мережі

Діаграми структурного системного аналізу

- діаграми «сутність-зв'язок» (ERD – Entity-Relationship Diagrams);
- діаграми функціонального моделювання (SADT – Structured Analysis and Design Technique);
- діаграми потоків даних (DFD – Data Flow Diagrams)

Окремі мови об'єктно-орієнтованого моделювання стали з'являтися до кінця 1970-х років; до 1994-му року їх кількість досягла 50. Кожен розробник вважав свій метод кращим; ухвалення окремих методик (IDEF0) як стандартних не змогло змінити ситуацію, що склалася.

У 1994-у році найпоширенішими методами стають:

- метод Граді Буча (Grady Booch) – Booch'93;
- метод Джеймса Румбаха (James Rumbaugh) – Object Modeling Technique (OMT-2);
- метод Айвара Джекобсона (Ivar Jacobson) – Object-Oriented Software Engineering (OOSE).

У жовтні 1994 року Г.Буч і Дж.Румбах з Rational Software почали роботу з *уніфікації* своїх методів, пізніше до них приєднався А.Джекобсон з

Objectory AB (Швеція). Робота була направлена на отримання нового методу, який би містив усе краще з попередніх, а саме:

- дозволяв моделювати не тільки програмне забезпечення, але і ширші класи систем і бізнес-додатків, з використанням об'єктно-орієнтованих понять;
- явним чином забезпечував взаємозв'язок між базовими поняттями для моделей концептуального і фізичного рівнів;
- забезпечував масштабованість моделей, що у край необхідне для складних багатоцільових систем;
- був зрозумілий аналітикам і програмістам, а також підтримувався спеціальними інструментальними засобами, реалізованими на різних комп'ютерних платформах.

Одержаний результат був названий UML-0.8 і опублікований у жовтні 1995 року, після чого був переданий практично всім компаніям – розробникам програмного забезпечення для зауважень і доповнень. Версія UML-1.0 з'явилася на світ на початку 1997 року; у березні 2003 року була випущена версія UML-1.5.

Хоча у даний час в країнах СНД використовуються три нотації візуального моделювання: IDEF (Icam DEFinition), ARIS (Architecture of Integrated Information Systems) і UML, саме остання поступово стає стандартом при розробці інформаційних систем. Ряд середовищ розробки додатків і візуального програмування – MS Visual Studio.NET, Borland Delphi 2005 і т.п. – не тільки підтримують нотацію UML як засіб моделювання, але і дозволяють одержати здійснимі програми на основі розробленої моделі.

Основні компоненти UML

Мова UML спирається на деякий набір базових принципів, вживаних при моделюванні складних систем, а саме:

- *принцип абстрагування*, приписуючий включати до моделі тільки ті аспекти проекрованої системи, які мають безпосереднє відношення до виконання системою своїх функцій;
- *принцип багатомодельності*, який затверджує, що ніяка єдина модель не може з достатнім ступенем адекватності описувати різні аспекти складної системи;

- *принцип ієрархічної побудови*, приписуючий розглядати процес побудови моделі на різних рівнях абстрагування або деталізації у рамках фіксованих уявлень.

Таким чином, процес ООАП можна подати як порівневий спуск від найзагальніших моделей і уявлень концептуального рівня до більш приватних і детальних уявлень логічного і фізичного рівнів.

Загальна схема взаємозв'язків моделей і уявлень складної системи під час об'єктно-орієнтованого аналізу і проектування подана на рисунку 3.



Рисунок 3 – Схема взаємозв'язків моделей і уявлень складної *системи*

Формальний опис мови UML ґрунтується на деякій загальній ієрархічній структурі модельних уявлень, що складається з чотирьох рівнів:

- мета-метамодель;
- метамодель системи;
- модель;
- об'єкти користувача.

Докладніше про кожен рівень наведено в [1-7].

У межах мови UML усі уявлення про модель системи фіксуються у вигляді спеціальних графічних конструкцій, що одержали назву діаграм. Передбачені наступні види діаграм (рис.4):

- 1 Діаграма варіантів використання (use case diagram)
- 2 Діаграма класів (class diagram)

- 3 Діаграми поведінки (behavior diagrams):
 - 3.1 Діаграма станів (statechart diagram);
 - 3.2 Діаграма діяльності (activity diagram)
 - 3.3 Діаграми взаємодії (interaction diagrams):
 - 3.3.1 Діаграма послідовності (sequence diagram);
 - 3.3.2 Діаграма кооперації (collaboration diagram)
- 4 Діаграми реалізації (implementation diagrams):
 - 4.1 Діаграма компонентів (component diagram);
 - 4.2 Діаграма розгортання (deployment diagram).

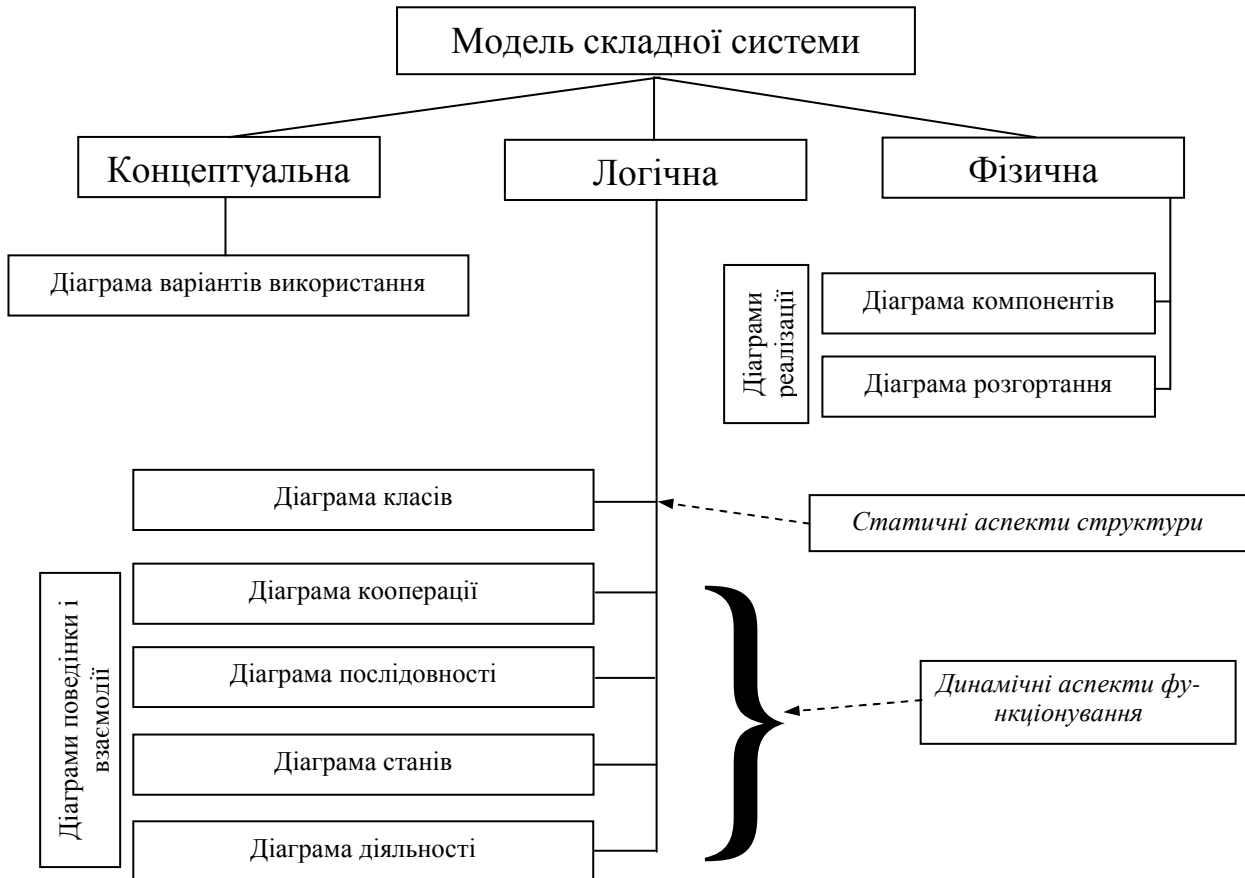


Рисунок 4 – Модель складної системи у нотації UML

Кожна з діаграм по-своєму описує (конкретизує) систему, причому загальна модель може містити лише ті діаграми, які достатньо адекватно характеризують проєктовану систему.

Особливості зображення діаграм

Більшість перелічених діаграм є, по суті, графами спеціального вигляду, що складаються з вершин у формі геометричних фігур, які зв'язані між собою ребрами або дугами. Як правило, ні розміри, ні розташування

(за винятком діаграми послідовностей) елементів діаграм не мають принципового значення.

Для всіх діаграм існує три типи візуальних графічних позначень:

- геометричні фігури на площини, що виконують роль вершин графів (*графічні примітиви*);
- графічні взаємозв'язки, що зображуються різними лініями;
- спеціальні символи (графічні або текстові), що зображаються поблизу інших елементів і називаються *специфікаціями*.

При зображенні діаграм слід дотримуватися наступних семи рекомендацій:

- 1 Кожна діаграма повинна служити закінченим представленням відповідного фрагмента модельованої наочної області.
- 2 Усі сутності на кожній діаграмі повинні бути одного концептуального рівня (для складних моделей слід використовувати принципи послідовного уточнення або деталізації).
- 3 Уся інформація про сутність повинна бути явно подана на діаграмі (не слід у всьому покладатися на «значення за умовчанням»).
- 4 Діаграми не повинні містити суперечливої інформації.
- 5 Діаграми не повинні бути переобтяжені текстовою інформацією.
- 6 Кількість типів діаграм для конкретної моделі не є точно фіксованою: наприклад, модель може не містити діаграму розгортання, якщо передбачається виключно локальна робота системи.
- 7 Модель системи повинна містити тільки ті елементи, які визначені в нотації мови UML.

2.2 Діаграма концептуального моделювання – діаграма варіантів використання (use case diagram)

Діаграма варіантів використання (іноді її називають діаграмою прецедентів) описує функціональне призначення системи або, іншими словами, те, що система робитиме під час свого функціонування. Ця діаграма є початковим концептуальним уявленням або концептуальною моделлю системи під час її проектування і розробки.

Розробка діаграми варіантів використання переслідує такі цілі:

- визначити загальні межі та контекст модельованої наочної області на початкових етапах проектування системи;
- сформулювати загальні вимоги до функціональної поведінки проєктованої системи;
- розробити початкову концептуальну модель системи для її подальшої деталізації у формі логічних і фізичних моделей;
- підготувати початкову документацію для взаємодії розробників системи з її замовниками і користувачами.

Суть даної діаграми полягає у наступному: проєктована система зображується у формі так званих варіантів використання, з якими взаємодіють деяка зовнішня суть, або актори. При цьому *актором*, або *дійовою особою*, називається будь-який об'єкт, суб'єкт або система, що взаємодіє з модельованою системою ззовні. У свою чергу, варіант використання служить для опису сервісів, які система надає актору. Іншими словами, кожен варіант використання визначає деякий набір дій, створений системою при діалозі з актором.

Таким чином, базовими елементами діаграми варіантів використання є: власне варіант використання (прецедент), актор і примітки.

Варіант використання (use case) є специфікацією загальних особливостей поведінки або функціонування модельованої системи без розгляду внутрішньої структури цієї системи. Позначається на діаграмі еліпсом, усередині (або нижче) якого міститься його коротка назва або ім'я у формі дієслова із словами пояснень; при цьому текст обов'язково повинен починатися із заголовної букви (рис.5).

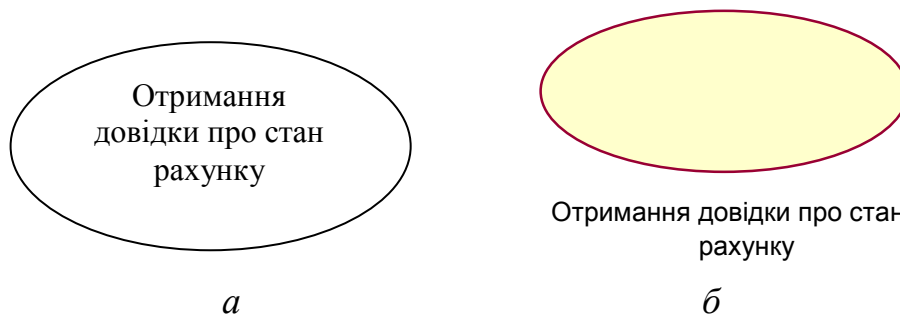


Рисунок 5 – Графічне позначення варіанта використання у початковій нотації (а) або в середовищі Rational Rose (б)

Актор (actor) є будь-якою зовнішньою у відношенні до модельованої системи суттю, яка взаємодіє з системою і використовує її функціональні можливості для досягнення певної мети або рішення приватних задач. Стандартним графічним позначенням актора на діаграмах є фігурка «людини», під якою записується конкретне ім'я актора (рис. 6). Як правило, під ім'ям маєтись на увазі посада (якщо актор – людина), але ніяк не ім'я власне (Вася Іванов) або назва конкретного пристрою (Маршрутизатор Cisco 3640).



Рисунок 6 – Графічне позначення актора

Актори взаємодіють з системою за допомогою передачі і прийому повідомлень від варіантів використання. Повідомлення є запитом актором сервісу від системи і отримання цього сервісу. Ця взаємодія може бути виражена за допомогою асоціацій між окремими акторами і варіантами використання або класами. Окрім цього, з акторами можуть бути пов'язані інтерфейси, які визначають, яким чином інші елементи моделі взаємодіють з цими акторами.

Інтерфейс (interface) служить для специфікації параметрів моделі, які видимі ззовні без вказівки їх внутрішньої структури. У мові UML інтерфейс є класифікатором і характеризує тільки обмежену частину поведінки

модельованої суті. Стосовно діаграм варіантів використання, інтерфейси визначають сукупність операцій, які забезпечують необхідний набір сервісів або функціональності для акторів. Інтерфейси не можуть містити ні атрибутів, ні станів, ні направлених асоціацій. Вони містять тільки операції без вказівки особливостей їх реалізації. Формально інтерфейс еквівалентний абстрактному класу без атрибутів і методів з наявністю тільки абстрактних операцій. На діаграмі варіантів використання інтерфейс зображається у вигляді маленького кола, поряд з яким записується його ім'я. Як ім'я може бути іменник, який характеризує відповідну інформацію або сервіс (рис. 7).



Рисунок 7 – Графічне позначення інтерфейсу

Примітка (note) призначена для включення до моделі довільної текстової інформації, що має безпосереднє відношення до контексту проекту, що розробляється. Графічно зображається прямокутником з «заломленим» куточком (рис.8), сполученим з елементом діаграми пунктирною лінією. Якщо примітка має ключове слово <<constraint>>, то є обмеженням, що накладається на елемент.

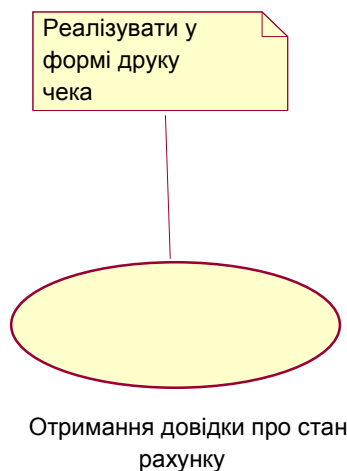


Рисунок 8 – Приклад примітки

Між компонентами діаграми варіантів використання можуть існувати різні відносини, які описують взаємодію екземплярів одних акторів і варіантів використання з екземплярами інших акторів і варіантів.

У мові UML є декілька стандартних видів відносин:

- відношення асоціації (association relationship);
- відношення включення (include relationship);
- відношення розширення (extend relationship);
- відношення узагальнення (generalization relationship).

Відношення асоціації визначає особливості взаємодії акторів і варіантів використання і позначається відрізком суцільної лінії (рис. 9).



Рисунок 9 – Приклад відношення асоціації

Відношення включення між двома варіантами використання указує, що деяка задана поведінка для одного варіанта використання включається як складовий компонент у послідовність поведінки іншого варіанта використання. Відношення включення, направлене від варіанта використання А до варіанта використання В, указує, що кожен екземпляр варіанта А включає функціональні властивості, задані для варіанта В. Ці властивості спеціалізують поведінку відповідного варіанта А на даній діаграмі. Графічно дане відношення позначається пунктирною лінією із стрілкою (варіант відношення залежності), направленою від базового варіанта використання до того, що включається, і позначається ключовим словом <<include>> (рис. 10). Один варіант використання може бути включений в низку інших варіантів, а також включати інші варіанти.

Відношення розширення визначає взаємозв'язок базового варіанта використання з деяким іншим варіантом використання, функціональна поведінка якого задіюється не завжди, а тільки при виконанні деяких додаткових умов.

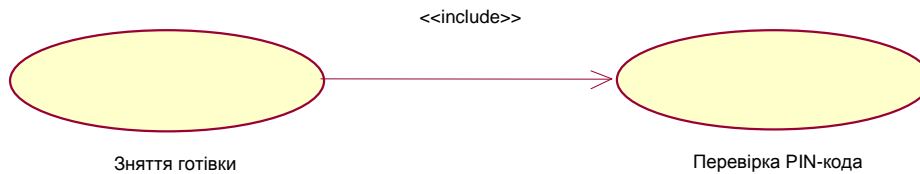


Рисунок 10 – Приклад відношення включення

Графічно позначається пунктирною лінією із стрілкою (варіант відношення залежності), направленою від того варіанта використання, який є розширенням для початкового варіанта використання, і позначається ключовим словом `<<extend>>` (рис. 11).

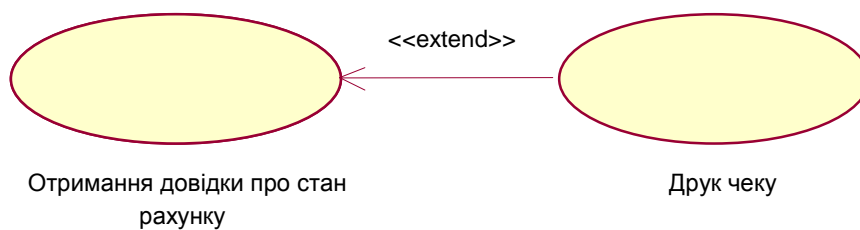


Рисунок 11 – Приклад відношення розширення

Відношення узагальнення служить для вказівки того факту, що деякий елемент А може бути узагальнений до елементу В. Таким чином, А буде спеціалізацією В. При цьому В називається предком або батьком у відношенні до А, а А – нащадком у відношенні до В. Графічно дане відношення позначається суцільною лінією із стрілкою у формі не зафарбованого трикутника, яка указує на батьківський елемент. Слід зазначити, що даний вид відносин застосовується як між варіантами використання (рис. 12), так і між акторами (рис. 13).



Рисунок 12 – Приклад відношення узагальнення між варіантами використання

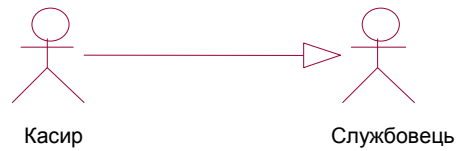


Рисунок 13 – Приклад відношення узагальнення між акторами

У [2] подано приклад моделі системи роботи банкомату, яку ми також візьмемо як базовий приклад. Дана система має два актори – Клієнт і Банк, причому головним є Клієнт, оскільки ініціює роботу. Базові варіанти використання – «Зняття готівки з картки» і «Отримання інформації про стан рахунку». Додаткові сервіси – «Перевірка PIN-коду» (використовується завжди, тому зв'язано відношенням включення) і «Друк чека» (використовується при бажанні Клієнта побачити стан рахунку не тільки на екрані, але і в друкарському вигляді, тому зв'язано відношенням розширення). Одержана діаграма зображена на рисунку 14.

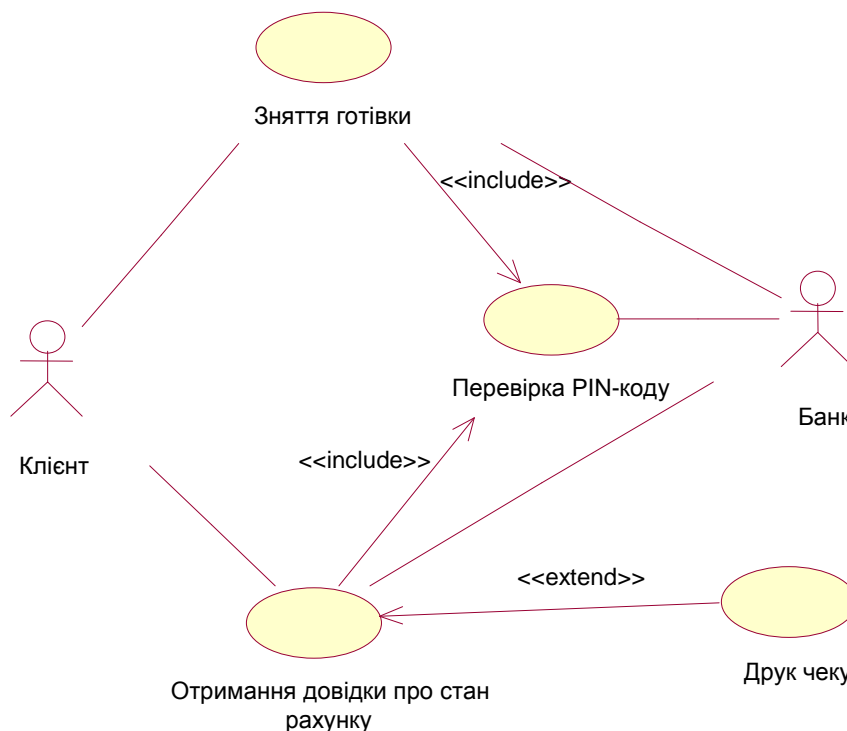


Рисунок 14 – Діаграма варіантів використання для моделі банкомата

Проста інформаційна система – програма для обробки якої-небудь інформації – може мати двох акторів (звичного користувача і адміністратора) і два базові варіанти використання («Введення і модифікація даних» і «Обробка даних»). Крім того, можливі додаткові сервіси – «Перевірка імені і пароля» (використовується постійно для відмінності простого користувача від адміністратора, тому зв'язано відношенням включення) і «Формування звіту» (використовується користувачем при необхідності, тому зв'язано відношенням розширення). Одержана діаграма зображена на рисунку 15.

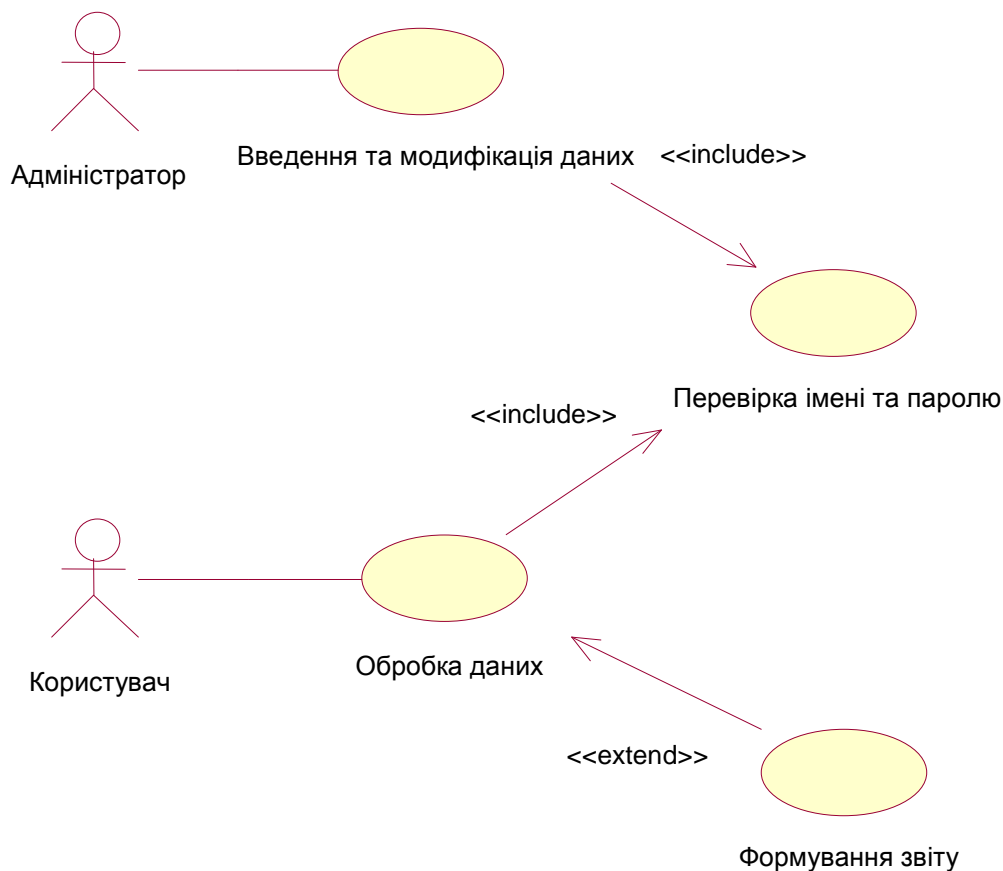


Рисунок 15 – Діаграма варіантів використання для моделі простої інформаційної системи

2.3 Діаграми логічного моделювання

Наступні п'ять діаграм – класів, кооперації, послідовності, станів і діяльності – описують логічний рівень моделі (як статичні аспекти структури, так і динамічні аспекти функціонування). Основні елементи цих діаграм – класи, об'єкти і відносини між ними.

2.3.1 Діаграма класів (*class diagram*)

Центральне місце в методології ООАП посідає розробка логічної моделі системи у вигляді діаграми класів.

Діаграма класів служить для представлення статичної структури моделі системи в термінології класів об'єктно-орієнтованого програмування. Діаграма класів може відображати, зокрема, різні взаємозв'язки між окремою суттю наочної області, такими, як об'єкти і підсистеми, а також описує їх внутрішню структуру і типи відносин. На даній діаграмі не указується інформація про тимчасові аспекти функціонування системи.

Базовими елементами діаграми класів є самі класи (з своїми атрибутами і операціями) і відносини між ними.

Клас – абстрактний опис або представлення властивостей безлічі об'єктів, які володіють однаковою структурою, поведінкою і відносинами з об'єктами з інших класів – графічно зображається у вигляді прямокутника, який додатково може бути розділений горизонтальними лініями на розділи або секції, в яких можуть указуватися ім'я класу, атрибути (змінні) і операції (методи) (рис. 16).

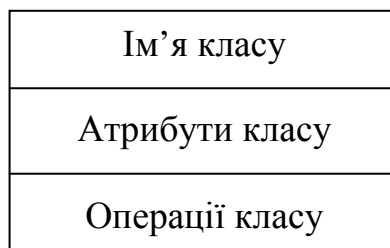


Рисунок 16 – Зображення класу

Як правило, якщо навіть якась секція виявляється порожньою, в позначенні класу вона все одно показується. Різні приклади зображення класів подані на рисунку 17.

Коло	Графічний об'єкт	База даних
xs, ys: Точка r: Число		data: Дані
	показати() приховати()	відкрити() відобразити() зачинити()

Рисунок 17 – Приклади зображень різних класів

Ім'я класу повинне бути унікальним у межах пакету (у пакеті може міститися декілька діаграм). Записується у центрі секції напівжирним шрифтом і повинне починатися із заголовної букви. Якщо клас не може мати екземплярів (об'єктів), тобто є абстрактним, його ім'я виділяється курсивом.

При позначенні імен класу рекомендується використовувати іменники, записані без пропусків. Необхідно пам'ятати, що імена класів утворюють словник наочної області при ООАП.

Атрибут класу служить для подання окремої властивості або ознаки, яка є загальною для всіх об'єктів даного класу. Запис атрибуту підкоряється синтаксичним правилам:

<квантор видимості> <ім'я атрибута> [кратність]:
 <тип атрибута> = <ісходне значення> {рядок-властивість}

Квантор видимості (visibility) може набувати одного з чотирьох значень:

- «+» – public – доступність без обмежень;
- «#» – protected – доступність тільки для даного класу і його нащадків;
- «-» – private – доступність тільки для даного класу;
- «~» – package – доступність тільки у межах даного пакету.

Ім'я атрибуту повинне починатися з рядкової (малої) букви і не може містити пропусків.

Кратність характеризує загальну кількість атрибутів даного типу, що входять до складу класу (за умовчанням дорівнює 1), наприклад:

[0..1] – або такого атрибуту немає, або він є;

[0..*] – або такого атрибуту немає, або їх скільки завгодно;

[1..5] – таких атрибутів може бути від 1 до 5.

Тип атрибуту визначається типом даних, наприклад: «колір: Color» або «ім'яСпівробітника [1..2]: String».

Початкове значення служить для завдання деякого початкового значення атрибуту у момент створення екземпляра класу, наприклад: «колір:Color=(255,0,0)» або «ім'яСпівробітника[1..2]:String=«Іван Іванович»».

Рядок-властивість служить для вказівки додаткових властивостей атрибуту, наприклад: «заробітняПлата:Гроші=500{frozen}» – фіксована сума.

Операція класу – це деякий сервіс, який надає кожен екземпляр (об'єкт) класу на вимогу своїх клієнтів (інших об'єктів, у тому числі і екземплярів даного класу). Сукупність операцій характеризує функціональний аспект поведінки всіх об'єктів даного класу.

Формат запису операції наступний:

<квантор видимості> <ім'я операції> (список параметрів):
<повертаєме значення> {рядок-властивість}

Наприклад:

+намалювати (форма: Багатокутник = Прямокутник)

–змінитиРахунок (номерРахунка: Integer): Гроші

Докладніше про операції див. в [1-7].

Між класами можуть бути наступні види відносин:

- відношення асоціації (association relationship);
- відношення узагальнення (generalization relationship).
- відношення агрегації (aggregation relationship);
- відношення композиції (composition relationship);
- відношення залежності (dependency relationship).

Відношення асоціації відповідає наявності довільного взаємозв'язку між класами і може бути як ненаправленою (рис. 18), так і направленою (рис. 19). Якщо зв'язані два класи один з одним, асоціація називається *бінарною*, якщо клас зв'язаний сам з собою – *рефлексії*. Асоціація може мати ім'я і містити кратності класів-ролей асоціації.

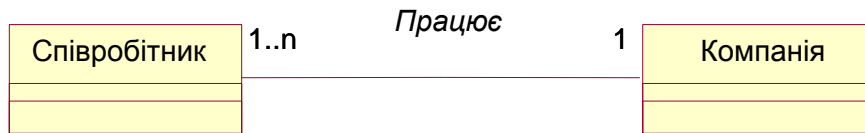


Рисунок 18 – Приклад ненаправленої бінарної асоціації



Рисунок 19 – Приклад направленої бінарної асоціації

Відношення узагальнення є відношенням між предком і нащадком (рис. 20).

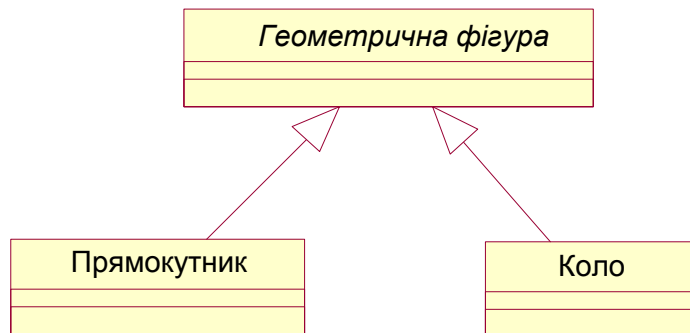


Рисунок 20 – Приклад відношення узагальнення

До стрілки-узагальнення можуть бути додані наступні обмеження:

- {complete} – означає, що в даному відношенні визначені всі класи-нащадки, й інших у даного класу-предка бути не може;
- {incomplete} – означає, що, навпаки, вказані не всі класи-нащадки даного класу-предка;

{disjoint} – означає, класи-нащадки не можуть містити об'єктів, що одночасно є екземплярами двох і більш класів;

{overlapping} – означає, що, навпаки, окремі екземпляри класів-нащадків можуть належати одночасно декільком класам.

Відношення агрегації має місце між класами в тому випадку, якщо один з класів є деякою суттю, яка включає як складові частини іншу суть (ієрархія вигляду «частина – ціле»). Зображається у вигляді відрізка лінії з не зафарбованим ромбом з боку класу-контейнера (рис. 21).

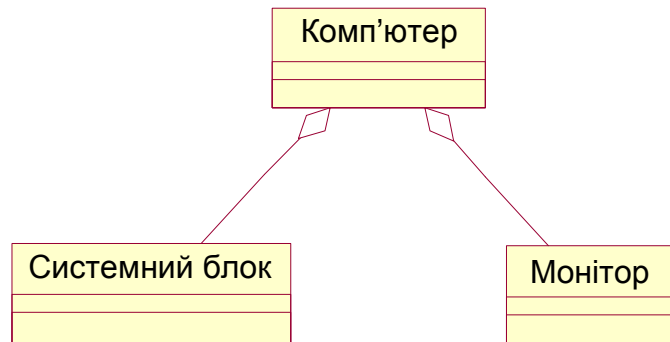


Рисунок 21 – Приклад відношення агрегації

Відношення композиції є окремим випадком агрегації, при якому складові частини не можуть існувати у відриві від цілого. Зображається у вигляді відрізка лінії із зафарбованим ромбом з боку класу-контейнера (рис. 22).

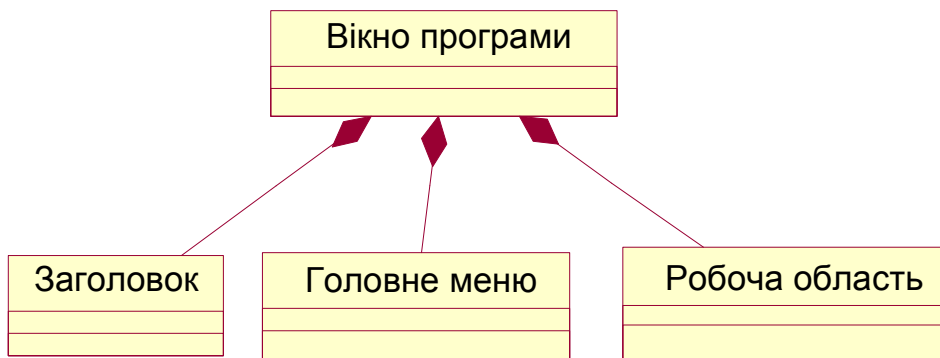


Рисунок 22 – Приклад відношення композиції

Відношення залежності використовується у разі наявності особливих відносин між елементами, які не можна точно віднести ні до одного з перелічених вище. Зображається у вигляді пунктирної лінії із стрілкою.

Інтерфейс є особливим випадком класу, у якого є тільки операції і відсутні атрибути. Для зображення інтерфейсу використовується стереотип <<interface>> або спеціальний графічний символ – коло.

На рисунку 23 зображена діаграма класів системи управління банкоматом з [2]. У ряду класів вказані їх стереотипи: «Пристрій читання», «Екран банкомату», «Принтер банкомату» і «Пристрій видачі готівки» – <<boundary>>, «Контроллер банку» – <<interface>>, «Контроллер банкомату» – <<control>>.

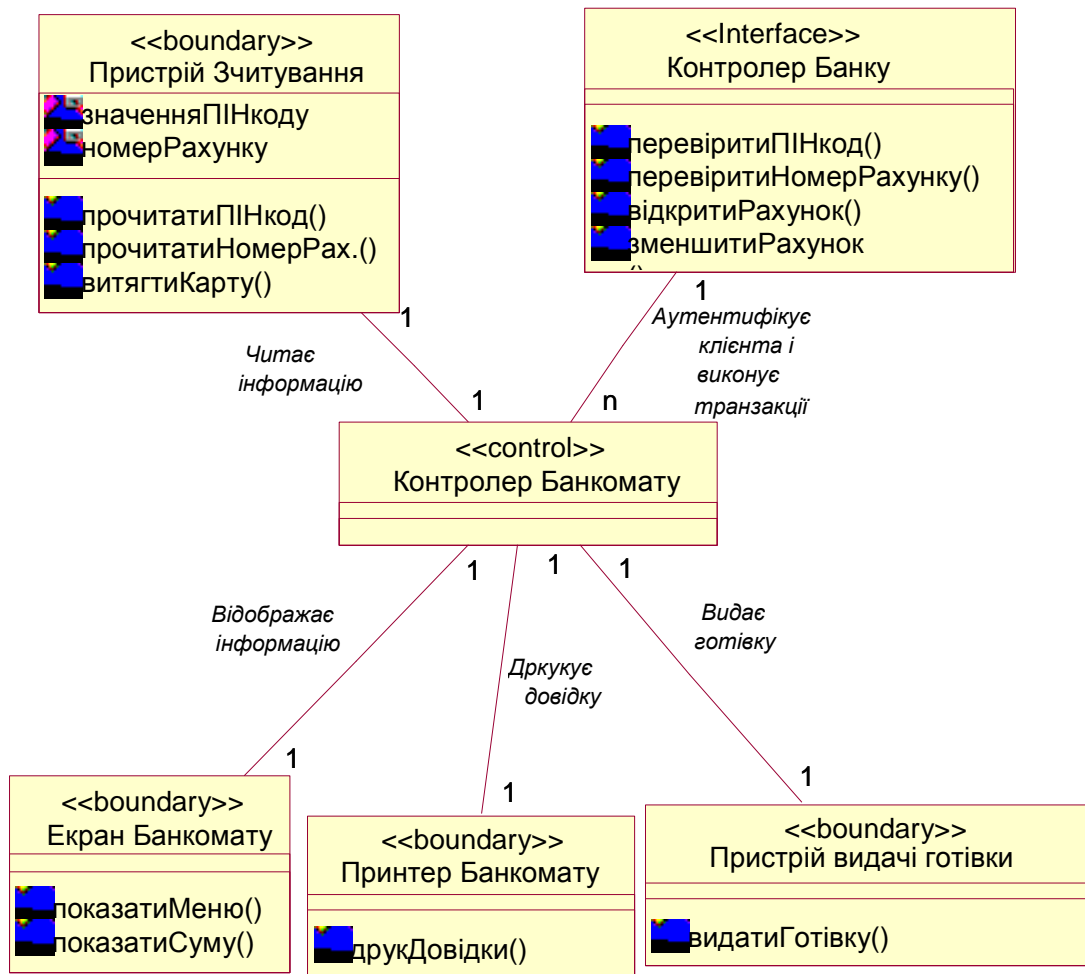


Рисунок 23 – Діаграма класів

Досить часто класи на діаграмі містять дуже багато атрибутів і операцій, внаслідок чого її перегляд стає скрутним. У таких випадках її поділяють на дві частини – власне діаграму класів (або діаграму асоціацій класів) з опущеними секціями атрибутів і операцій (рис. 24) і діаграму конкретизації класів (рис. 25).

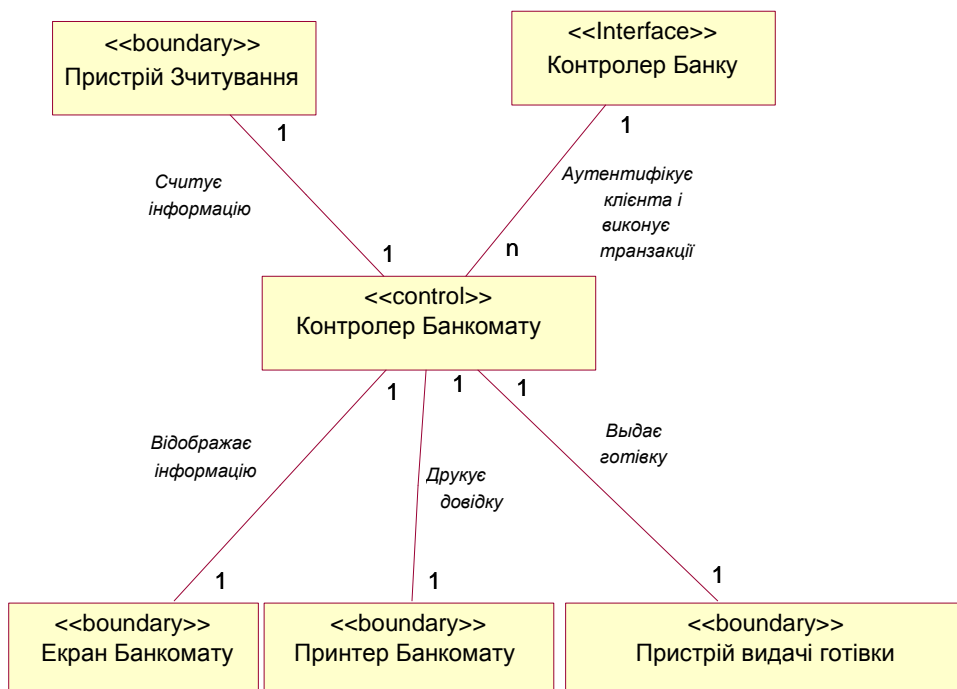


Рисунок 24 – Спрощена діаграма класів

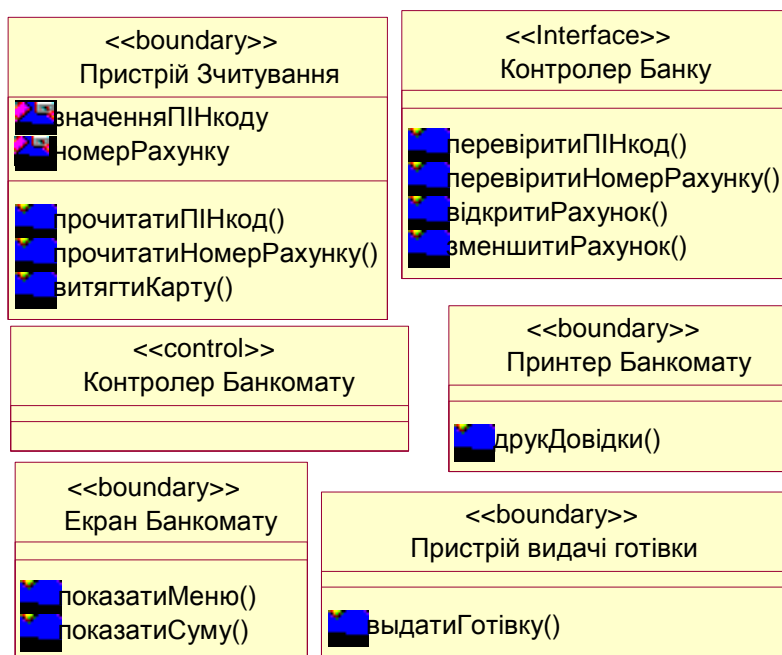


Рисунок 25 – Діаграма конкретизації класів

На рисунку 26 зображена діаграма класів простої інформаційної системи. Тут є два класи акторів – «Користувач» і «Адміністратор», управляючий (<<control>>) клас «Програма» і два класи для роботи з даними – «База даних» і «Звіт».

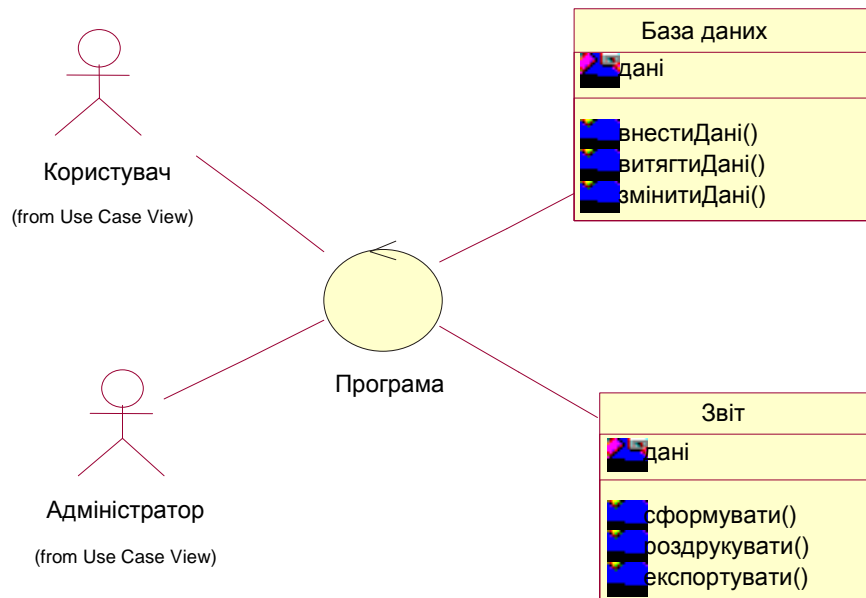


Рисунок 26 – Діаграма класів для моделі простої інформаційної системи

2.3.2 Діаграма кооперації (collaboration diagram)

Для моделювання взаємодії об'єктів у мові UML використовуються так звані діаграми взаємодії, до яких відносяться діаграма кооперації і діаграма послідовності.

Особливості передачі та прийому повідомлень у контексті статичної структури моделі відображає діаграма кооперації. Тут поведінка системи описується на рівні окремих об'єктів, які обмінюються між собою повідомленнями, щоб досягти потрібної мети або реалізувати деякий варіант використання. З погляду аналітика або архітектора системи важливо подати у проекті структурні зв'язки окремих об'єктів між собою. Таке подання структури моделі як сукупності взаємодіючих об'єктів і забезпечує дана діаграма. Слід розуміти, що одна і та ж сукупність об'єктів може брати участь у реалізації різних кооперацій – при цьому, залежно від даної кооперації, можуть змінюватися як зв'язки між окремими об'єктами, так і потік повідомлень між ними.

Ключові поняття діаграми кооперації – об'єкти (екземпляри класів), зв'язки (екземпляри асоціацій) і повідомлення. Кооперація може бути подана на двох рівнях:

- на рівні специфікації – показує ролі класифікаторів і ролі асоціацій у даній взаємодії;
- на рівні прикладів (екземплярів) – указує об'єкти і зв'язки, створюючи окремі ролі в кооперації.

При моделюванні інформаційних систем найчастіше використовується другий випадок, тому він і буде розглянутий. Для розгляду діаграм рівня специфікації треба звернутися до роботи [2].

Об'єкт (object) є окремим екземпляром класу, який створюється на етапі реалізації моделі (виконання програми). Він може мати власне ім'я і конкретні значення атрибутів. Відповідно, зображається у вигляді прямокутника, що іноді (при необхідності вказівки значень атрибутів) складається з двох секцій (див. рис. 27).

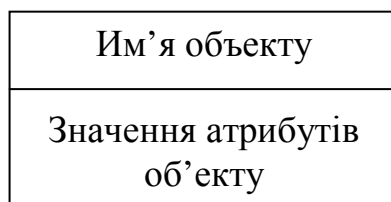


Рисунок 27 – Зображення об'єкта

Ім'я об'єкта в загальному випадку має наступний формат:

<власне ім'я>/<ім'я ролі класифікатора>:<ім'я класифікатора>

В окремих випадках власне ім'я об'єкта може бути відсутнім – такий об'єкт називається *анонімним* (двокрапка перед ім'ям класу повинна залишитися обов'язково). Якщо відсутнє ім'я класу, то об'єкт називається *сиротою*.

Різні приклади зображення об'єктів подані на рисунку 28.

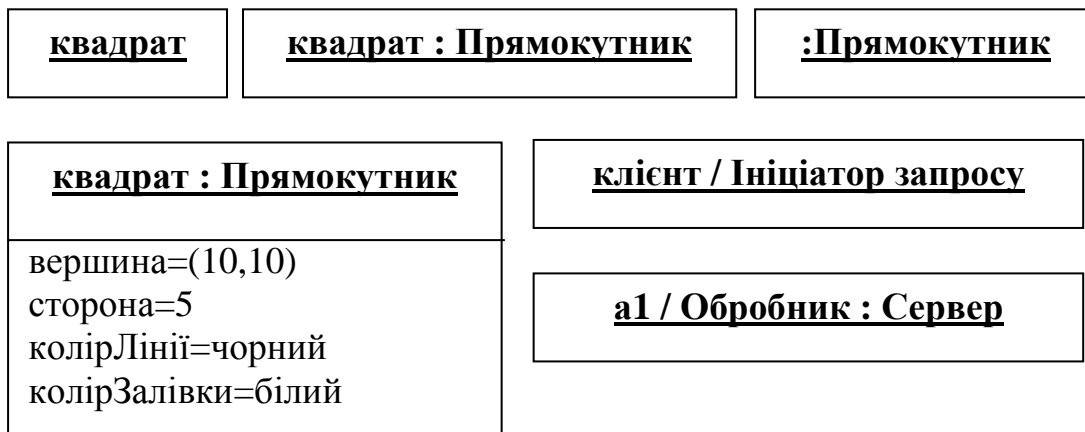


Рисунок 28 – Приклади зображень різних об'єктів

Іноді необхідно окремо зобразити на діаграмі безліч об'єктів, які можуть бути утворені на основі одного класу. Таке зображення у вигляді «накладених» прямокутників називається *мультіоб'єктом*. При цьому може бути явно вказане відношення агрегації (композиції) між мультіоб'єктом і окремим об'єктом з його множини (рис. 29).

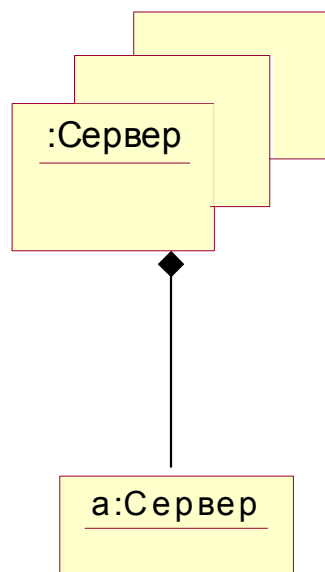


Рисунок 29 – Приклад зображення мультіоб'єкта

Усі об'єкти поділяються на дві категорії: пасивні й активні. *Пасивний об'єкт* (passive object) оперує тільки даними і не може ініціювати діяльність з управління іншими об'єктами. *Активний об'єкт* (active object) має свій власний потік управління (процес) і може ініціювати діяльність з управління іншими об'єктами. Як правило, активні об'єкти на діаграмі коо-

перації позначаються або потовщенням меж прямокутника, або явною вказівкою ключового слова { active }.

Як ілюстрація вищевикладеного на рисунку 30 наведений приклад діаграми кооперації для виклику функції друку з текстового редактора [2].

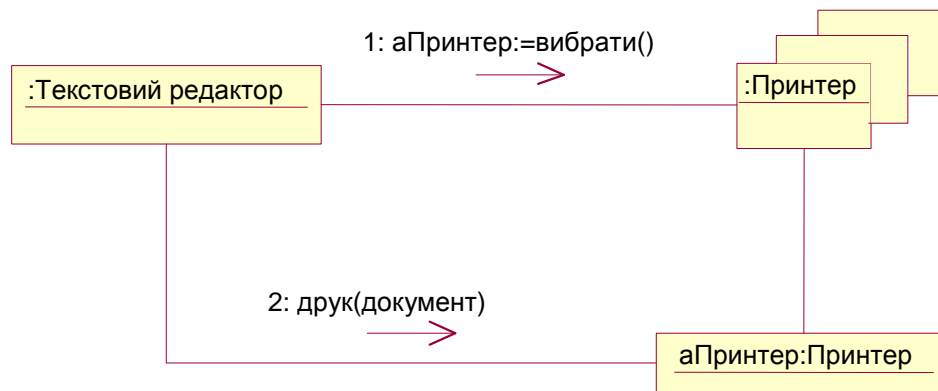


Рисунок 30 – Приклад діаграми кооперації

Складовий об'єкт (composite object) є екземпляром класу-композиту, який зв'язаний відношенням композиції зі своїми частинами (див. приклад на рис. 31).



Рисунок 31 – Складовий об'єкт «Вікно»

Зв'язок (link) є екземпляром асоціації і може мати місце між двома і більш об'єктами. Зображається відрізком прямої лінії, на кінцях якого можуть бути явно вказані імена ролей відповідної асоціації.

Повідомлення (message) показує комунікацію між двома об'єктами, один з яких передає іншому деяку інформацію, при цьому передбачається, що після отримання повідомлення другим об'єктом настане деяка дія. По-

повідомлення зображаються стрілками поряд з відповідним зв'язком, напрям стрілки указує на одержувача повідомлення (див. рис. 30).

Стрілки повідомлень можуть бути трьох типів:

- суцільна лінія з трикутною закрашеною стрілкою (—►) позначає виклик процедури (операції) або передачу потоку управління, при цьому об'єкт, що передає повідомлення, чекає закінчення деяких дій, викликаних цим повідомленням (*синхронні повідомлення*);

- суцільна лінія з V-подібною стрілкою (—>) позначає *асинхронне повідомлення*, тобто об'єкт, що передає повідомлення, продовжує свою діяльність, не чекаючи відповіді;

- пунктирна лінія з V-подібною стрілкою (--->) позначає повернення з виклику процедури; як правило, передбачаються за умовчанням і на діаграмах зображаються рідко.

Повідомлення має наступний формат:

<Попередні повідомлення> <Вираз> <Значення:=ім'я> <(Список аргументів)>

Значення вказівки *попередніх повідомлень* полягає у тому, що дане повідомлення не може бути передано, поки не будуть передані адресатам перелічені повідомлення:

2,4 / 5: передати (дані)

Вираз може бути умовою, за якою здійснюватиметься передача повідомлення:

6 [(хс > 0) & (хс < 800)]: намалюватиКоло (хс, 100)

Значення припускає, що очікується повернення деяких результатів виконання операції:

7: аПринтер:=обратиПринтер()

На рисунках 32 і 33 наведені приклади діаграм кооперації – для моделі системи управління банкоматом (з [2] – прецедент «Зняття готівки») і

для моделі простої інформаційної системи (прецедент «Робота користувача: формування звіту»). Стереотипи зв'язків (<<Local>>, <<Global>> і т. п.) показані літерами на кінцях.

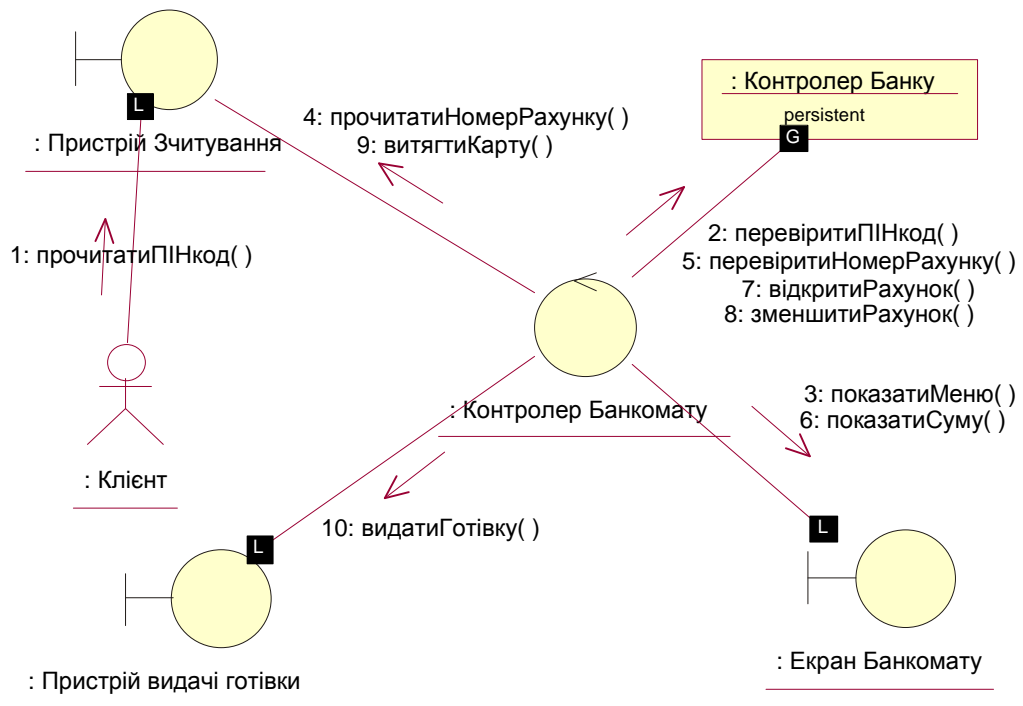


Рисунок 32 – Діаграма кооперації для моделі системи управління банкоматом

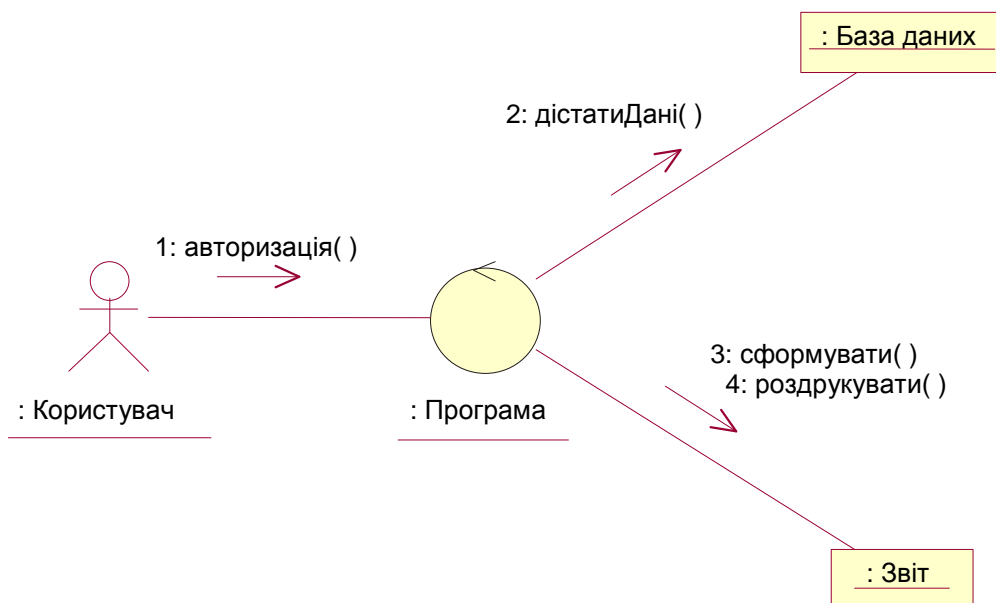


Рисунок 33 – Діаграма кооперації для моделі простої інформаційної системи

2.3.3 Діаграма послідовності (*sequence diagram*)

Діаграма послідовності відображає тимчасові особливості передачі та прийому повідомлень між об'єктами. З її допомогою можна описати повний контекст взаємодій як своєрідний «графік життя» всієї сукупності об'єктів, що взаємодіють між собою для реалізації варіанта використання програмної системи, досягнення бізнес-мети або виконання якої-небудь задачі.

Кожен об'єкт на цій діаграмі зображається у вигляді прямокутника, як і на діаграмі кооперації, проте розташовуються ці прямокутники послідовно зліва направо, причому крайнім зліва зображається об'єкт – ініціатор взаємодії (як правило, це актор). Порядок розташування об'єктів визначається виключно міркуваннями зручності.

З кожного об'єкта «витікає» вертикальна пунктирна лінія – його *лінія життя* (object lifeline). Для статичних об'єктів ця лінія триває до самого низу діаграми, для динамічних – до спеціального символу знищення об'єкта.

Явне виділення активності об'єкта наголошується *фокусом управління* (focus of control), який зображається у вигляді витягнутого вузького прямокутника, «нанизаного» на лінію життя. Кожен об'єкт за час існування може одержувати фокус управління скільки завгодно раз. Фокус управління актора, як правило, існує в системі постійно, відзначаючи характерну активність такого об'єкта.

Повідомлення на діаграмі послідовності мають такий же сенс і майже такий же опис, як і на діаграмі кооперацій (тут номер повідомлення опускається, послідовність визначається зліва направо і зверху вниз). Якщо об'єкт посилає повідомлення самому собі, воно називається *рефлексією*. Якщо в результаті повідомлення рефлексії створюється новий процес, то він зображається у вигляді *рекурсивного* (вкладеного) фокусу управління.

Для зображення галуження біля кожної гілки в квадратних дужках повинна бути вказане відповідна умова у формі булевого виразу.

Усі перелічені вище елементи діаграми послідовності подані на рисунку 34.

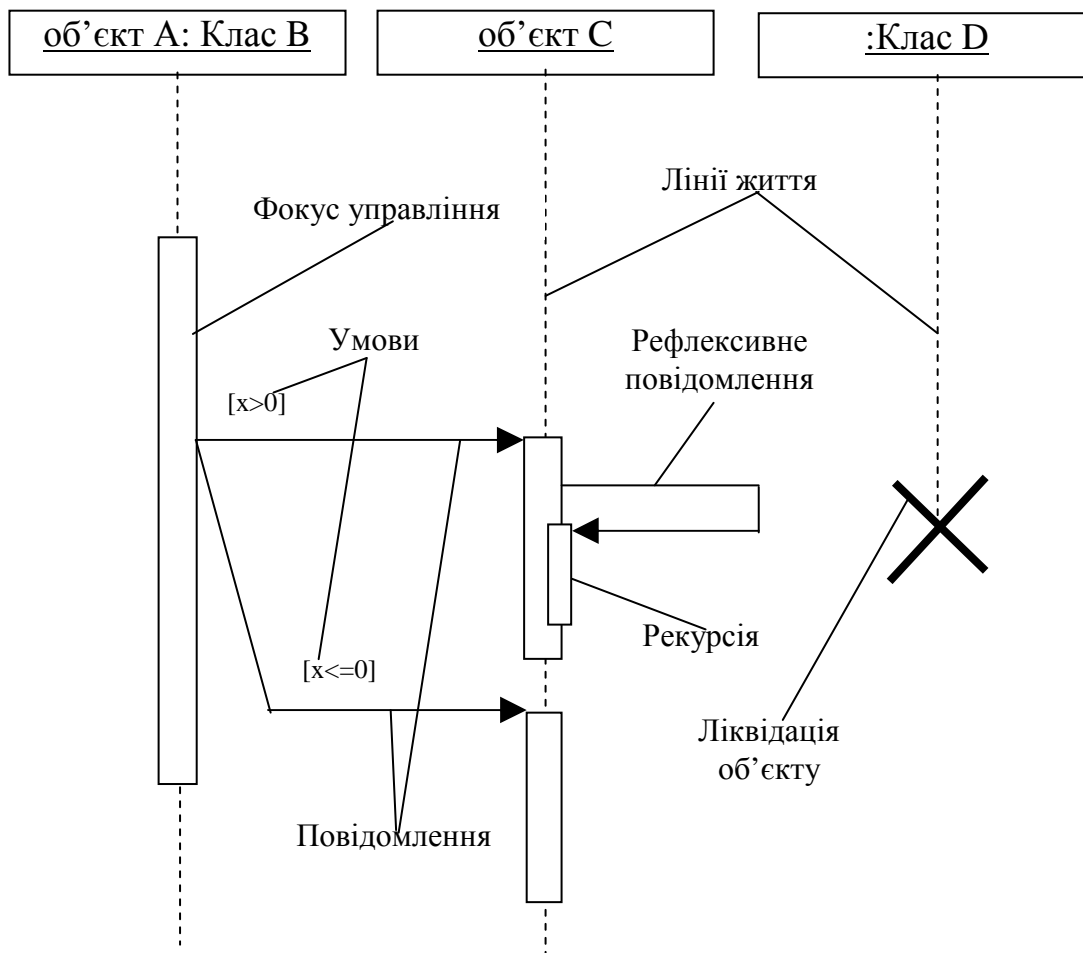


Рисунок 34 – Елементи діаграми послідовності

На рисунках 35 і 36 зображені діаграми послідовності для моделі системи управління банкоматом з [2] і для моделі простої інформаційної системи.

Слід зазначити, що Rational Rose дозволяє не будувати діаграму послідовності «з нуля», а одержати її автоматично з діаграми кооперації.

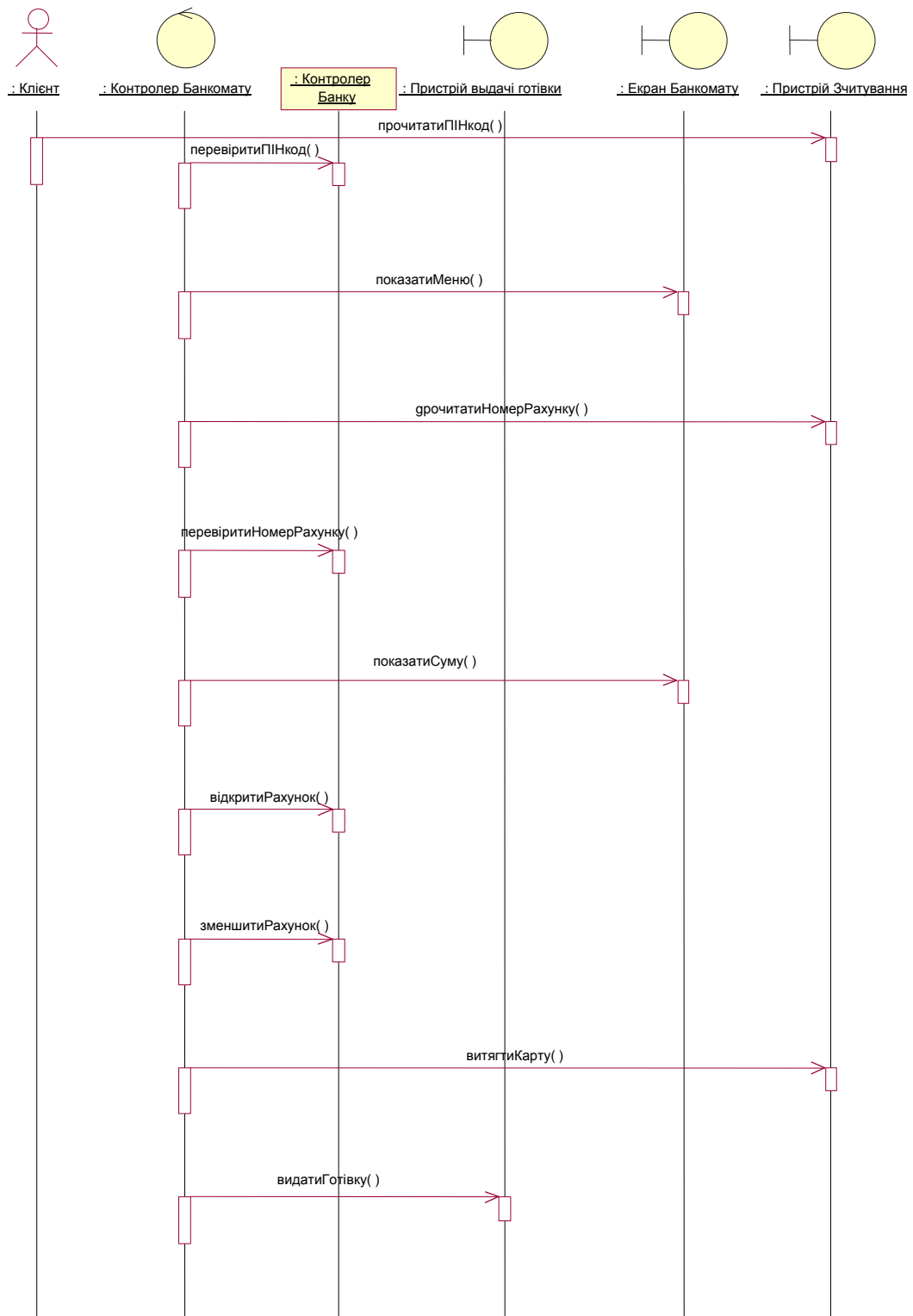


Рисунок 35 – Діаграма послідовності для моделі системи управління банкоматом

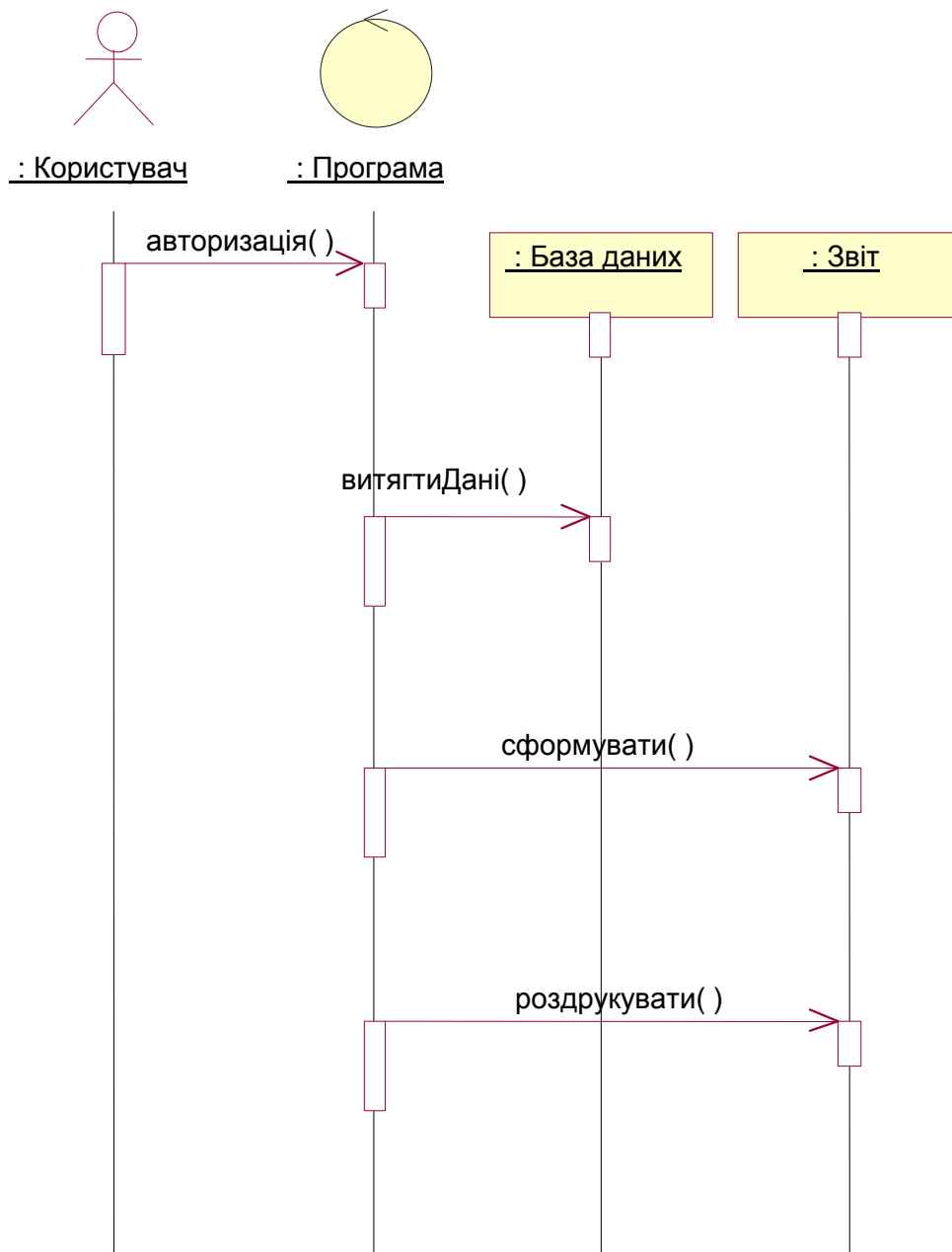


Рисунок 36 – Діаграма послідовності для моделі простої інформаційної системи

2.3.4 Діаграма станів (*statechart diagram*)

Для більшості складних систем зображення динамічної взаємодії елементів моделі у вигляді діаграм кооперації і послідовності виявляється недостатнім.

На відміну від своїх попередниць, діаграма станів описує процес зміни станів системи при реалізації всіх варіантів використання. При цьому такі зміни можуть бути викликані діями з боку інших елементів або ззовні системи.

Головне призначення даної діаграми – описати можливі послідовності станів і переходів, які в сукупності характеризують поведінку модельованої системи (або якоїсь підсистеми) протягом всього її життєвого циклу. За своєю суттю діаграма станів є графом спеціального вигляду, який служить для подання деякого кінцевого автомата. Основні поняття – стан, перехід і умови.

Під **станом** розуміється абстрактний метаклас, використовуваний для моделювання окремої ситуації, протягом якої виконується певна умова. На діаграмі зображається прямокутником із заокругленими вершинами, який може бути розділений горизонтальною лінією (рис. 37).

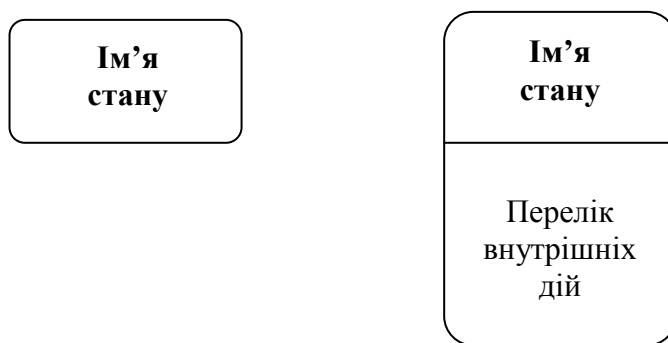


Рисунок 37 – Зображення станів

Ім'я стану є закінченою пропозицією і записується з прописної літери. У виняткових випадках ім'я може бути відсутнім – такий стан називається *анонімним*.

Список внутрішніх дій стану визначає діяльність системи при її знаходженні в даному стані. Кожна така дія записується окремим рядком у наступному форматі:

мітка дії / вираз дії

Мітка дії указує на обставини або умови, за якими виконуватиметься ця дія, і є або одним з чотирьох службових слів, або ідентифікатором події, що запускає дію (т.з. *внутрішній перехід*). Стандартні мітки дії наведені нижче:

entry – указує на вхідну дію, тобто дія, яка повинна бути виконана у момент входу в стан;

exit – указує на вихідну дію, тобто дія, яка повинна бути виконана у момент виходу із стану;

do – визначає деяку діяльність (do activity), яка виконуватиметься протягом всього часу знаходження системи у даному стані або до виконання спеціальної умови;

include – звертається до кінцевого підавтомата, ім'я якого указується замість виразу дії.

Приклад стану із списком внутрішніх дій – введення пароля – наведений на рисунку 38 [2].

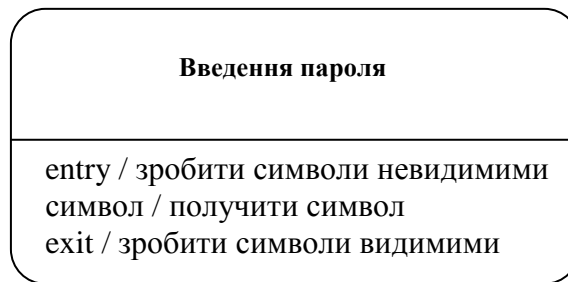


Рисунок 38 – Приклад стану із списком внутрішніх дій

Початковий стан (initial state) є окремим випадком стану, який не містить ніяких внутрішніх дій, і в якому система (об'єкт) знаходиться у початковому моменті. Графічно зображається у вигляді зафарбованого кола (рис. 39, а).



Рисунок 39 – Початковий і кінцевий стани

Кінцевий стан (final state) є окремим випадком стану, який також не містить ніяких внутрішніх дій і в якому модельована система (об'єкт) зна-

ходиться після завершення роботи. Графічно зображається у вигляді зафарбованого кола, поміщеного у інше коло (рис. 39, б).

Простий перехід (simple transition) є відношенням між двома послідовними станами, яке указує на факт зміни одного стану (*джерела*) іншим (*цільовим*). Перехід здійснюється при настанні події – завершенні “діяльності” в початковому стані (*перехід нетрігера*) або надходженні повідомлення (*перехід тригера*), а також при задоволенні так званої *сторожової умови*. Зображається суцільною лінією із стрілкою, направленою від початкового стану в цільовий, рядком тексту, що супроводжується. Якщо початковий і цільовий стани співпадають, перехід називають *переходом в себе* і зображають петлею із стрілкою. *Сторожова умова* (guard condition) записується в квадратних дужках після (або замість) рядка-події і є деяким булевим виразом, тобто повинне приймати одне з двох значень – «істина» або «брехня».

Складовий стан (composite state) полягає декількох вкладених *підстанів* (substate). Залежно від взаємного розташування підстанів можуть бути послідовними (sequential substates) або паралельними.

На рис. 40 показаний складовий стан з двома вкладеними послідовними підстанами на прикладі роботи телефонного апарату (прецедент «дозвон до абонента») [2], на рис. 41 – складовий стан з двома вкладеними паралельними підстанами.

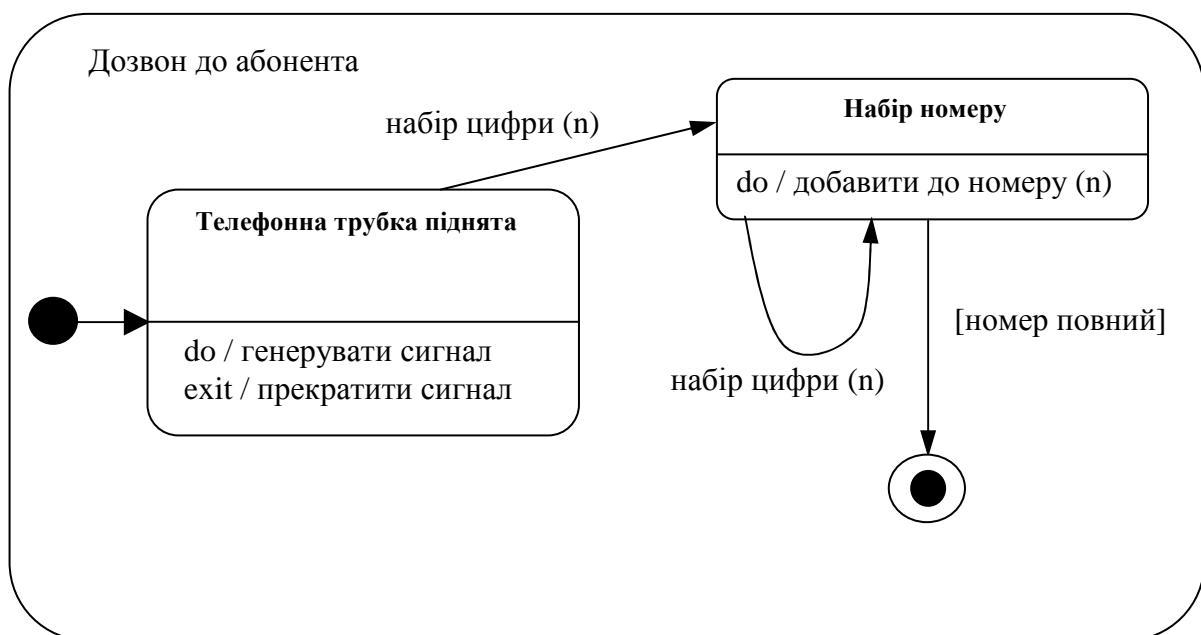


Рисунок 40 – Діаграма станів дозвону до абонента

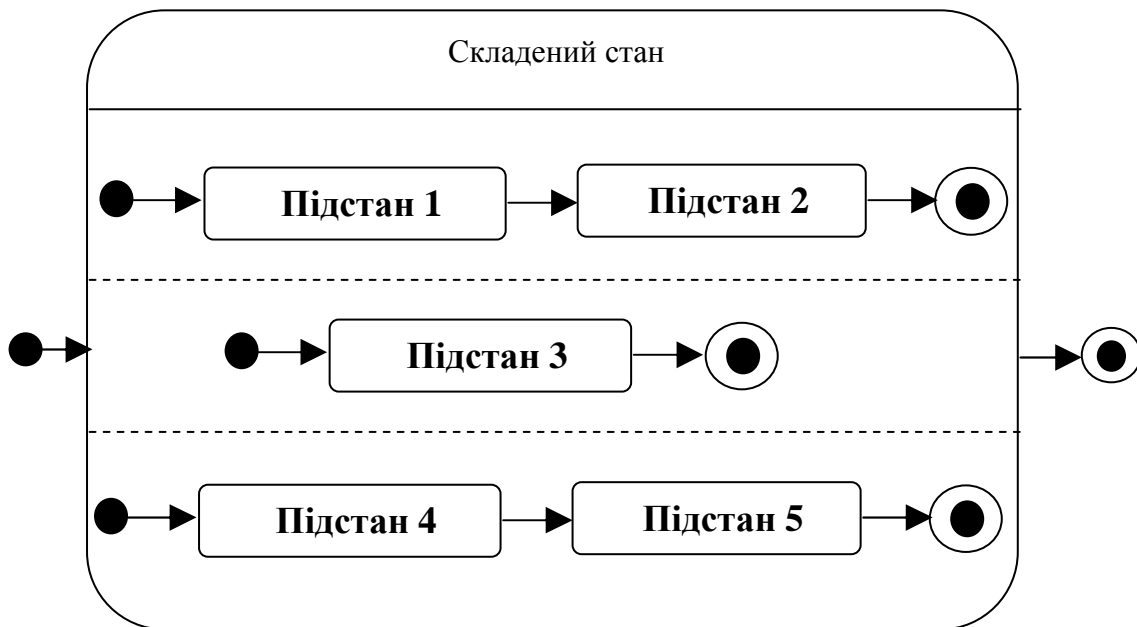


Рисунок 41 – Складений стан з паралельними підстанами

Якщо при виході з підстану необхідно його «запам'ятати» для подальшого повернення, використовуються так звані *історичні стани* (history state). *Неглибокий історичний стан* (shallow history state) зображається літерою Н (рис. 42, а) і визначає перший підстан при вході в складовий стан (при першому вході воно автоматично замінить початковий стан). *Глибокий історичний стан* (deep shallow history state) зображається символом Н* і служить для запам'ятовування всіх вкладених підстанів (рис. 42, б).

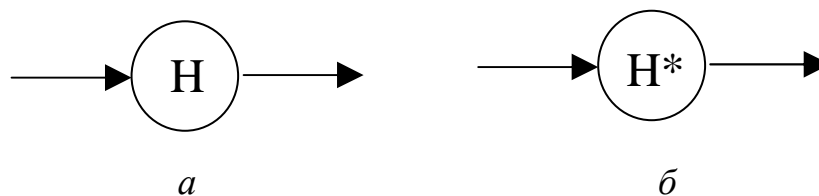


Рисунок 42 – Недавнє (неглибоке) і давнє (глибоке) історичні стани

Якщо потрібно явно показати, що деякий перехід може мати декілька початкових станів або декілька цільових станів, він називається *паралельним переходом* і зображається вертикальною межею. На рисунку 43 показані два різновиди паралельного переходу – *злиття* (join) і *розділення* (fork).

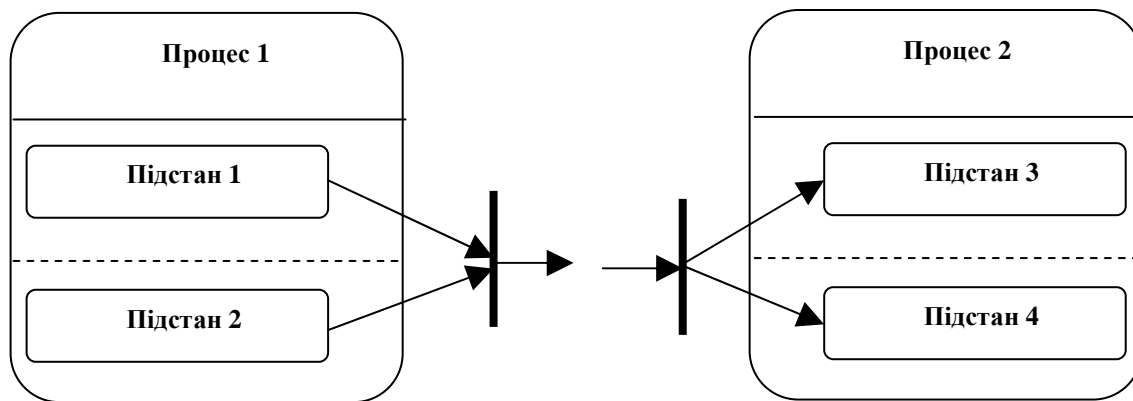


Рисунок 43 – Паралельні переходи вигляду «злиття» (а)
і «розділення» (б)

Синхронізуючий стан (synch state) використовується спільно з переходом-злиттям або переходом-розділенням для явної вказівки впливу подій в одному підавтоматі на інший підавтомат і позначається невеликим колом з «зірочкою» всередині.

На рисунку 44 наведений приклад використання синхронізуючих станів при моделюванні будівництва будинку [2], на рисунку 45 зображена діаграма станів для моделі системи управління банкоматом, а на рисунку 46 – можливий варіант діаграми станів для роботи простої інформаційної системи.

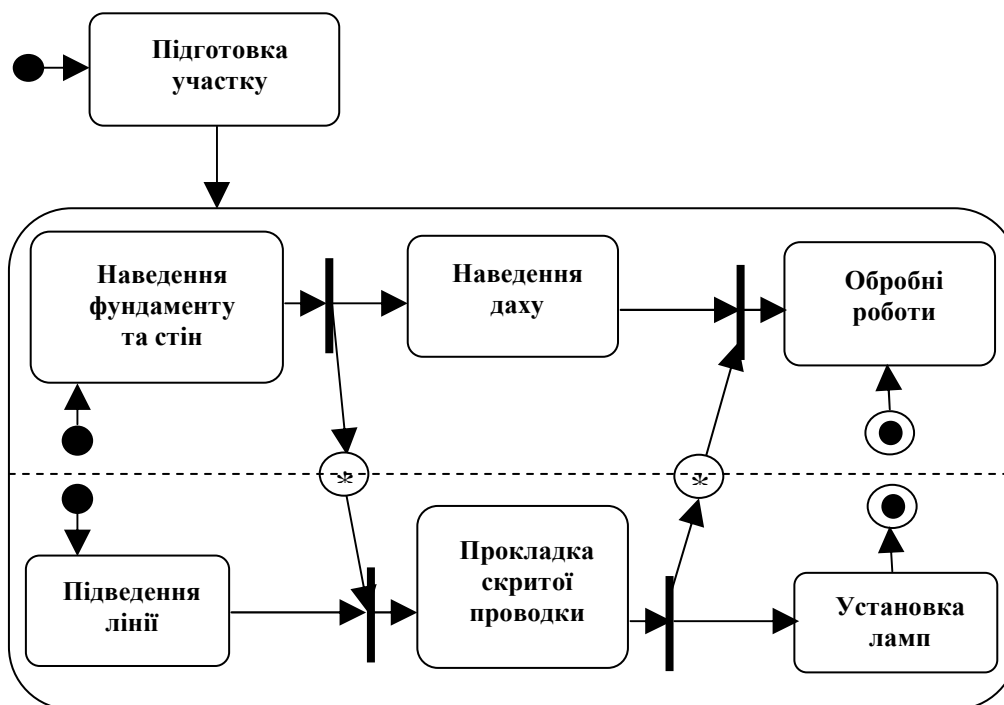


Рисунок 44 – Діаграма станів моделювання будівництва будинку

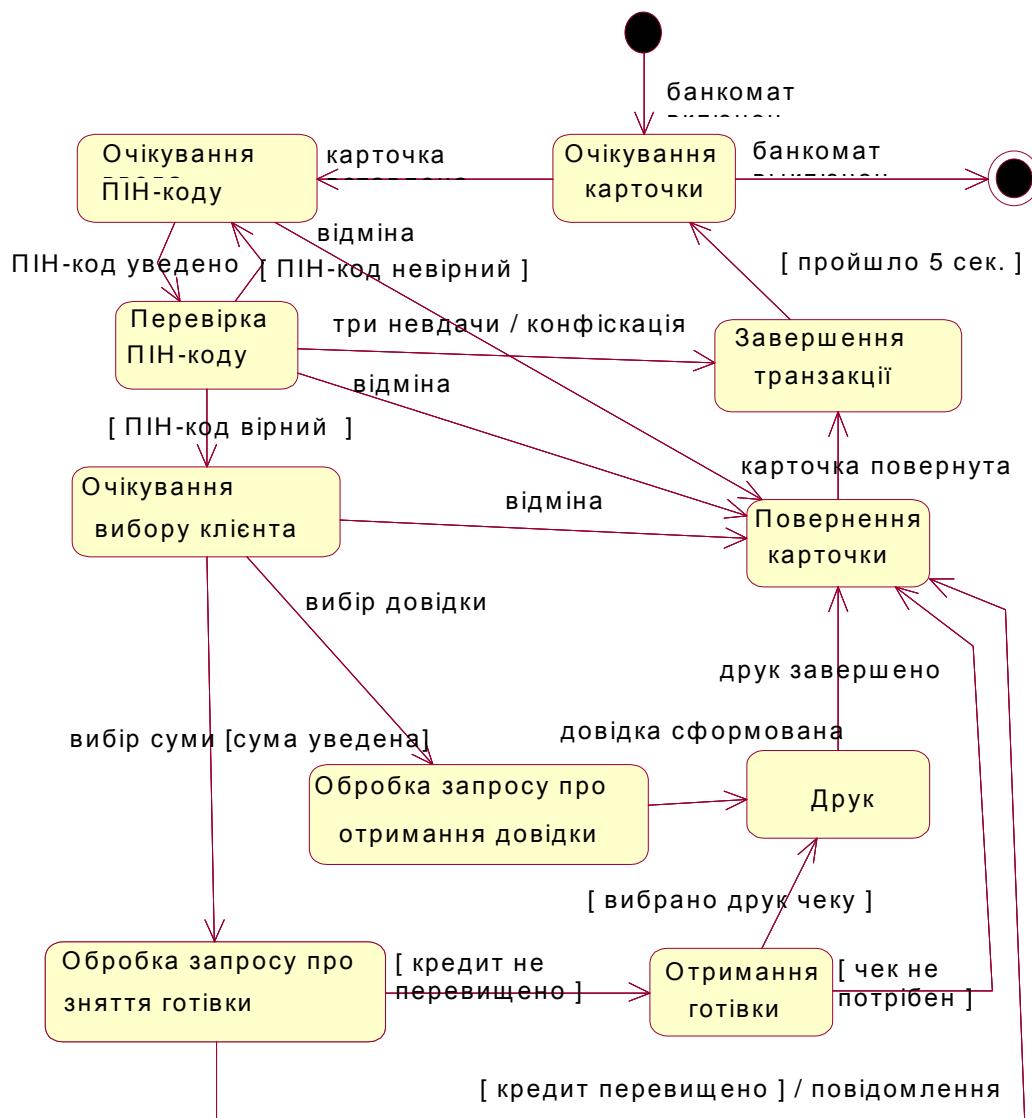


Рисунок 45 – Діаграма станів для моделі системи управління банкоматом

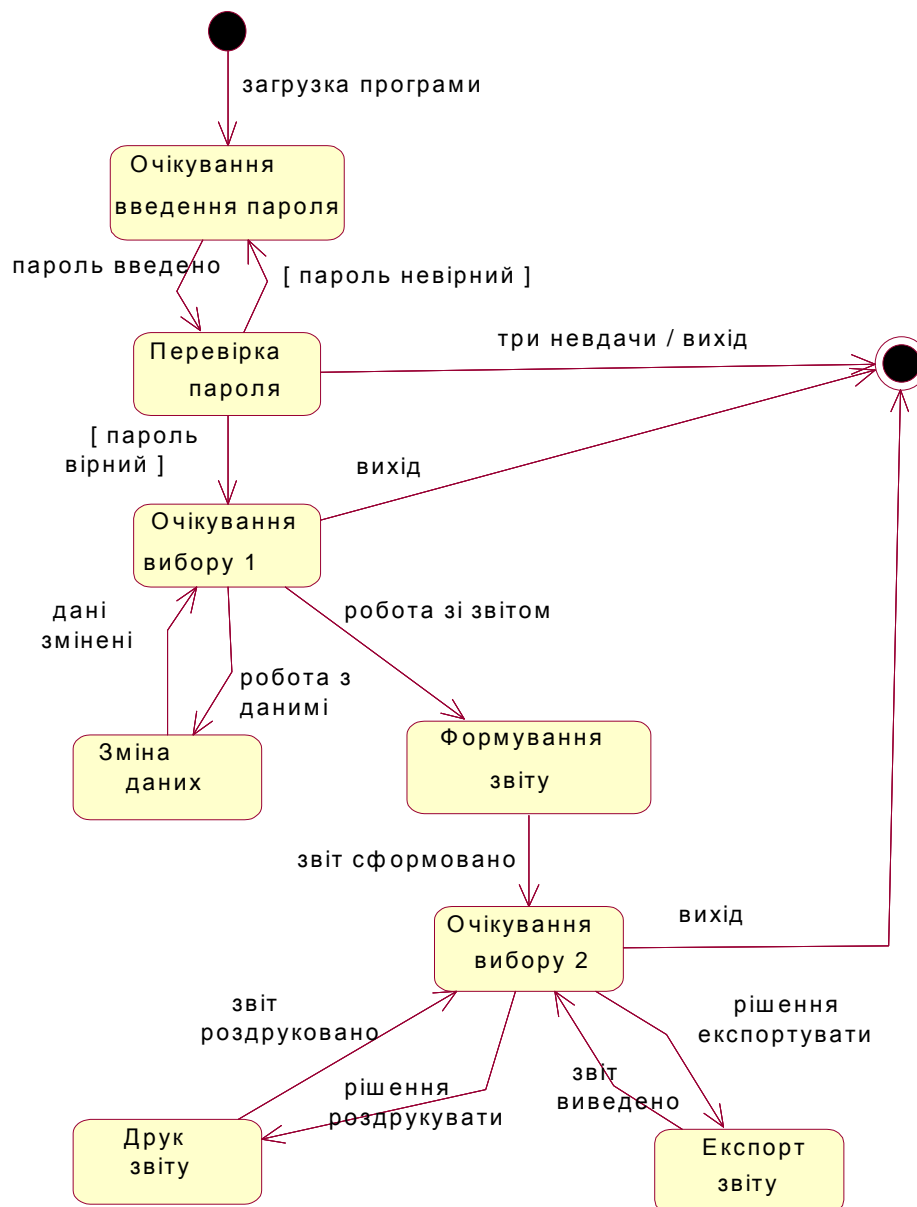


Рисунок 46 – Діаграма станів для моделі простої інформаційної системи

2.3.5 Діаграма діяльності (activity diagram)

Із збільшенням складності системи посилюється важливість точного дотримання послідовності виконуваних дій. Для моделювання процесу виконання операцій в мові UML використовуються діаграма діяльності, яка за своєю суттю є окремим випадком діаграми станів.

Діяльність (activity) є деякою сукупністю окремих обчислень, які можуть приводити до деякого результату або дії.

Стан дії (action state) є спеціальним випадком стану з деякою вхідною дією і, як мінімум, одним переходом, що виходить із стану, який неявно припускає, що вхідна дія вже завершилася. Стан дії не може мати внутрішніх переходів і підстанів, оскільки є елементарним – моделює один крок виконання алгоритму.

Графічно стан дії зображається прямокутником з сферичними сторонами, усередині якого записується ім'я стану-дії у формі виразу-дії (action-expression), як це показано на рисунку 47.

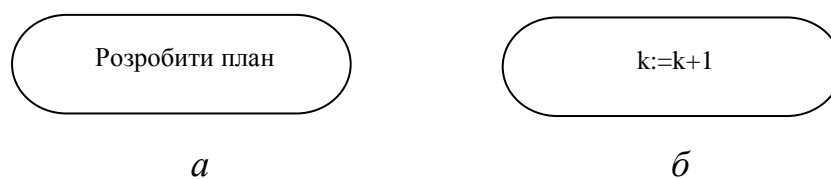


Рисунок 47 – Дія вигляду «проста діяльність» (*а*)
і «вираз» (*б*)

Переходи на діаграмі діяльності можуть бути тільки нетрігерами, тобто не припускаючими ніяких інших подій, окрім завершення попередньої діяльності.

Галуження зображається символом *рішення* (decision) – невеликого ромба без тексту. Об'єднання альтернативних гілок здійснюється за допомогою такого ж ромба, але званого вже *з'єднанням* (merge).

При моделюванні бізнес-процесів корисно використовувати спеціальну конструкцію під назвою **доріжки** (swimlane), яка дозволяє показати взаємозв'язок підрозділів. На рисунку 48 подано фрагмент діаграми діяльності для торгової компанії, а саме – прийом замовлення на товар і його відпустку зі складу. Тут передбачаються три доріжки: «Відділ прийому і оформлення замовлення», «Відділ продажів» і «Склад». Зрозуміло, що при моделюванні реальної бізнес-діяльності доріжок і дій буде значно більше.

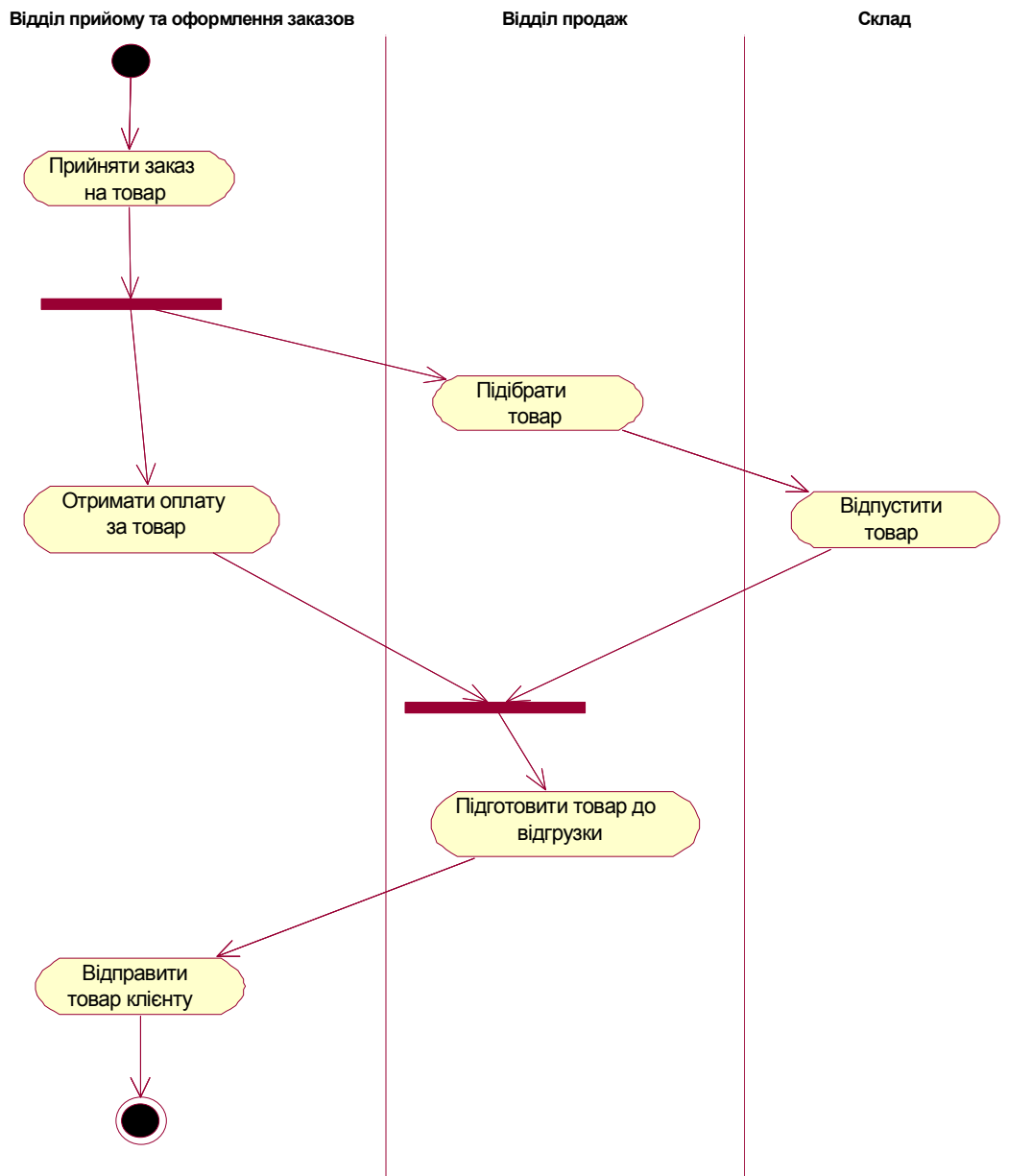


Рисунок 48 – Діаграма діяльності для торгової компанії

На рисунку 49 побудована діаграма діяльності для нашого крізного прикладу – процесу функціонування системи управління банкоматом [2], а на рисунку 50 – діаграма діяльності для простої інформаційної системи.

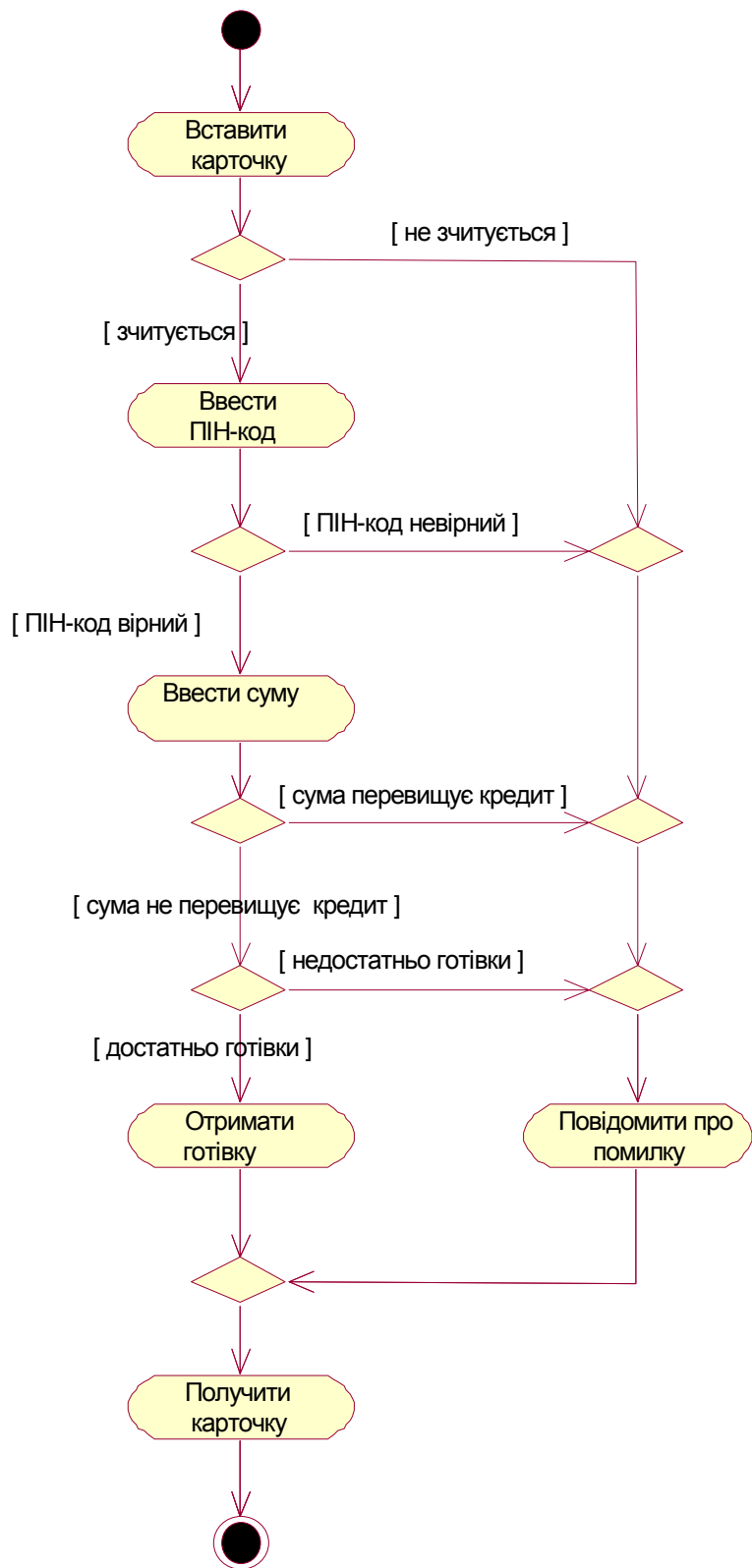


Рисунок 49 – Діаграма діяльності для процесу функціонування системи управління банкоматом

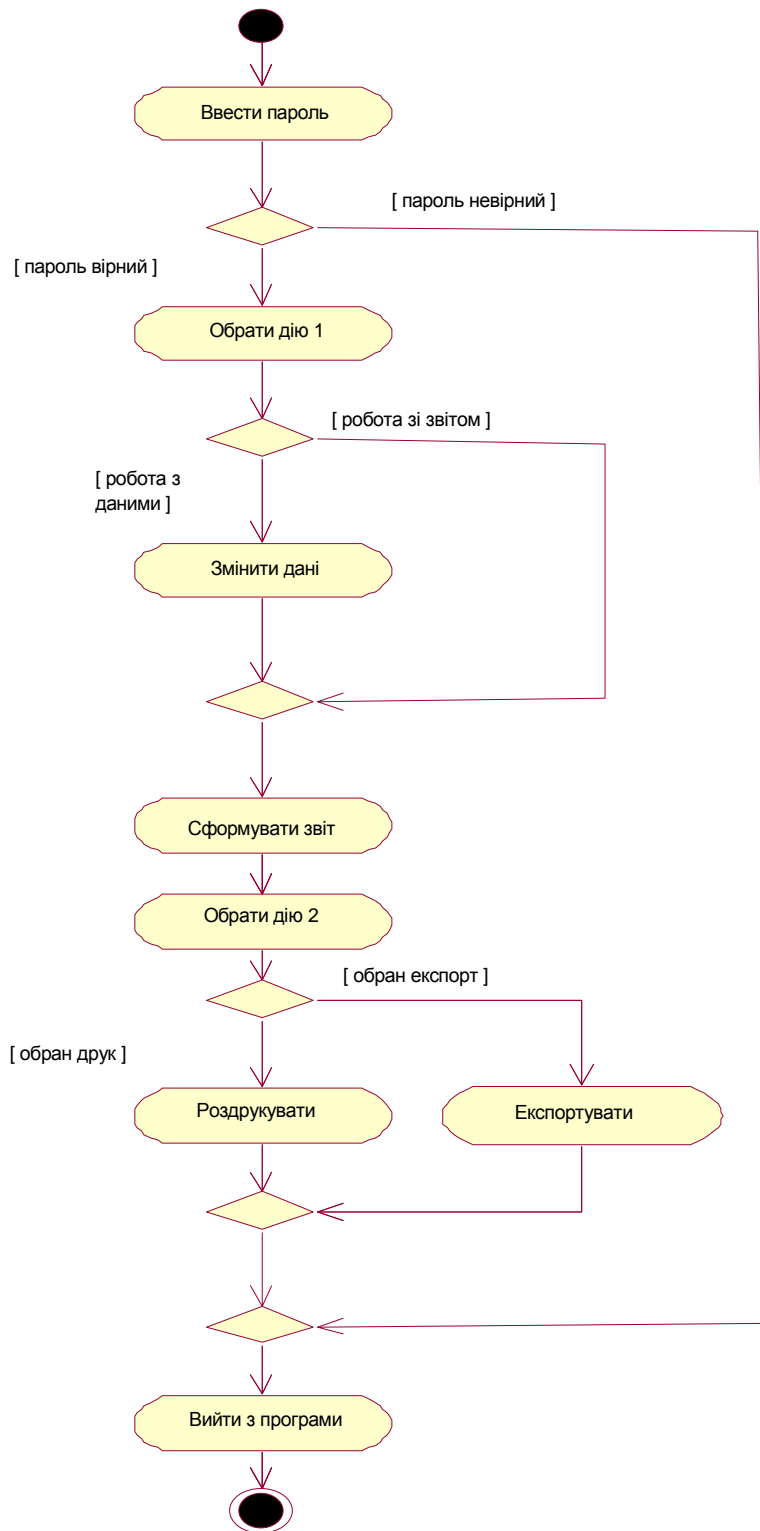


Рисунок 50 – Діаграма діяльності для простої інформаційної системи

2.4 Діаграми фізичного моделювання

Для створення конкретної фізичної системи необхідно деяким чином реалізувати всі елементи логічного уявлення в конкретну матеріальну суть. Моделювання такого перетворення описують так звані *діаграми реалізації* (implementation diagram), до яких відносяться *діаграма компонентів* і *діаграма розгортання*.

2.4.1 Діаграма компонентів (component diagram)

Діаграма компонентів дозволяє визначити архітектуру системи, що розробляється, встановивши залежності між програмними компонентами, в ролі яких може виступати початковий і виконуваний код (як правило, компонент відповідає файлу операційної системи). Основними графічними елементами цієї діаграми є компоненти, інтерфейси і залежності між ними.

Компонент (component) є деякою фізичною суттю і може реалізовувати деякий набір інтерфейсів. Графічно зображається прямокутником з ім'ям і зі вставленими зліва двома прямокутниками трохи менше (рис. 51).

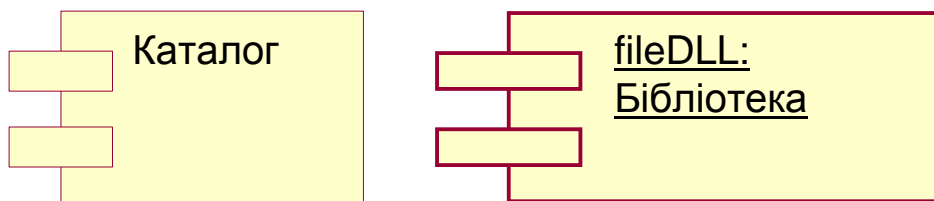


Рисунок 51 – Зображення компонентів

Запис імені компонента залежить від того, уявлений він як тип або як екземпляр. У першому випадку записується ім'я типу із заголовної букви, в другому – у формі «ім'я компонента: ім'я типу».

Як власні імена прийнято використовувати імена виконуваних файлів (EXE), динамічних бібліотек (DLL), web-сторінок (HTML), текстових файлів (TXT або DOC), файлів довідки (HLP), баз даних або файлів з початковими текстами програм (PAS, CPP, JAVA, PL і т.п.). Зовнішній вигляд таких компонентів на діаграмі не визначений нотацією мови UML і залежить від середовища побудови діаграм (CASE-засоби). Представлення деяких компонентів у середовищі IBM Rational Rose показано на рисунку 52.

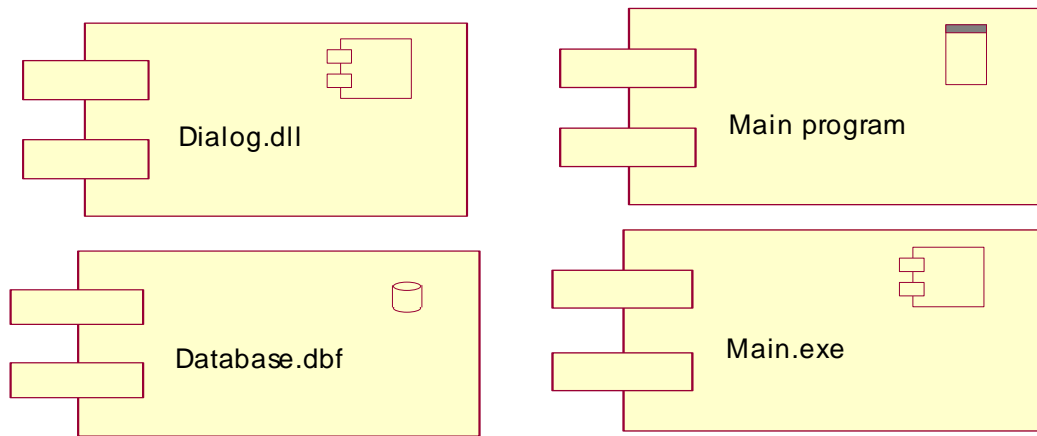


Рисунок 52 – Варіанти зображення компонентів у середовищі IBM RR

Інтерфейс (interface) зображається колом, яке з'єднується з компонентом відрізком лінії без стрілок, або у вигляді прямокутника класу із стереотипом <<interface>> (рис. 53).

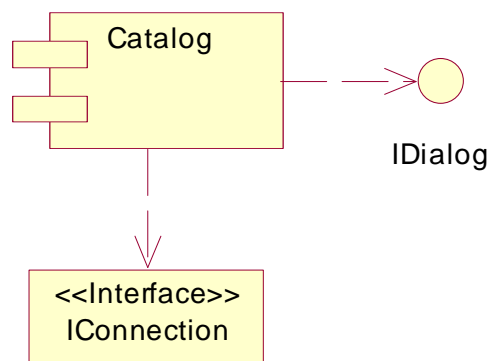


Рисунок 53 – Зображення інтерфейсів

Відношення залежності (dependency) зображається пунктирною лінією із стрілкою, направленою від клієнта (залежного елементу) до джерела (незалежного елементу). На рисунку 53 було показано, що компонент залежить від своїх інтерфейсів.

Докладніше про особливості побудови діаграми компонентів і про додаткові можливості можна прочитати у [2-7]. На рисунку 54 зображена діаграма компонентів системи управління банкоматом [2], на рисунку 55 – один з можливих варіантів діаграми компонентів простої інформаційної системи (передбачається реалізація в середовищі Borland Delphi).

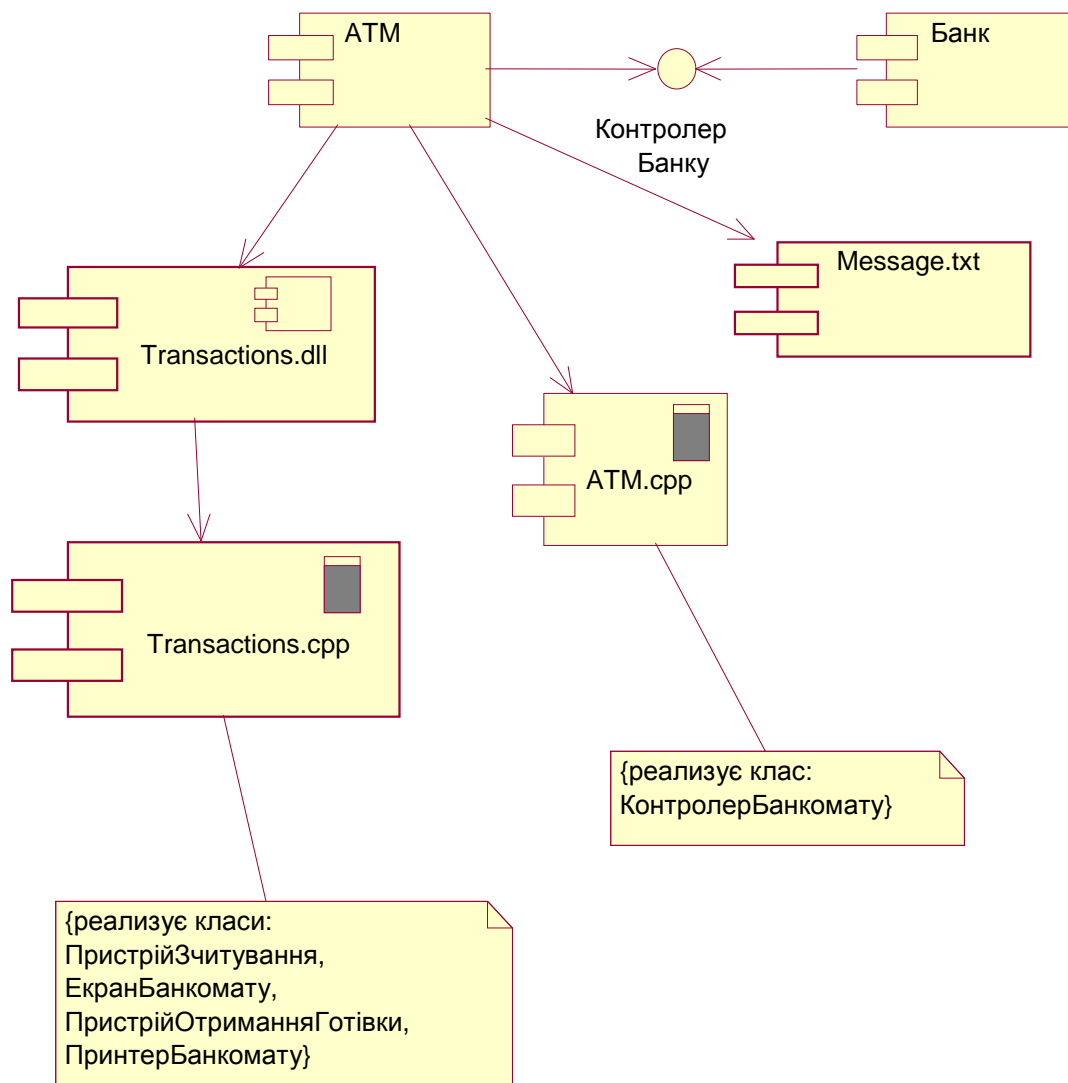


Рисунок 54 – Діаграма компонентів системи управління банкоматом

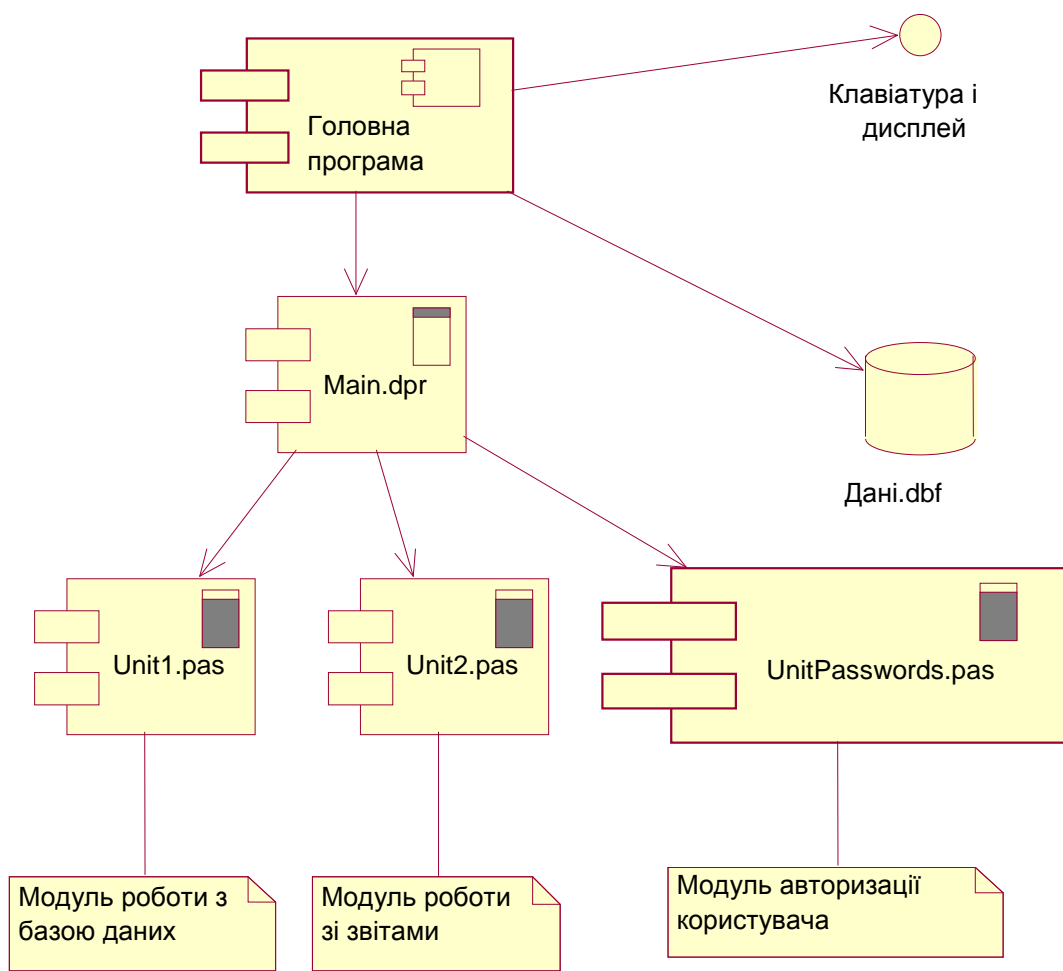


Рисунок 55 – Діаграма компонентів інформаційної системи

2.4.2 Діаграма розгортання (deployment diagram)

Якщо програма, що розробляється, припускає «локальну» роботу на одному комп'ютері, то побудовою діаграми компонентів проектування системи і закінчується. Проте складні програмні системи часто реалізуються у мережевому варіанті, що припускає використання різних обчислювальних платформ і технологій доступу до даних. Для представлення загальної конфігурації і топології розподіленої програмної системи, а також маршрутів передачі інформації між апаратними пристроями застосовується діаграма розгортання.

Основні елементи цієї діаграми – процесори і пристрої (вузли), процеси і зв'язки між ними.

Вузол (node) є деяким фізично існуючим елементом системи, який може володіти обчислювальним ресурсом (один або декілька процесорів, оперативна пам'ять і т. п.). До вузлів відносяться комп'ютери, принтери, модеми, сканери й інші подібні пристрої. Графічно вузол зображається у формі паралелепіпеда з ім'ям, яке може бути як ім'ям типу вузла, так й ім'ям вузла-екземпляра. Ресурсоємний вузол (processor) зображається у формі паралелепіпеда із зафарбованими бічними гранями (рис. 56).

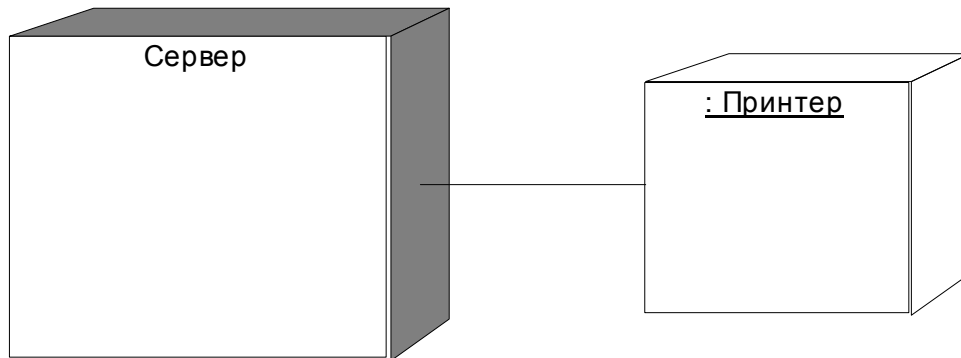


Рисунок 56 – Вузли на діаграмі розгортання

Як відносини між вузлами виступають фізичні з'єднання між ними, а також залежності між вузлами й іншими компонентами. **З'єднання** є різноманітним асоціації і зображаються відрізками прямої лінії без стрілок (див. рис. 56). **Залежності** відображаються відрізками пунктирної лінії із стрілками, направленими від вузла до залежних від нього компонентів.

На рисунку 57 зображена діаграма розгортання системи управління банкоматом [2], на рисунку 58 – простий варіант діаграми розгортання інформаційної системи у разі її клієнт-серверної реалізації.

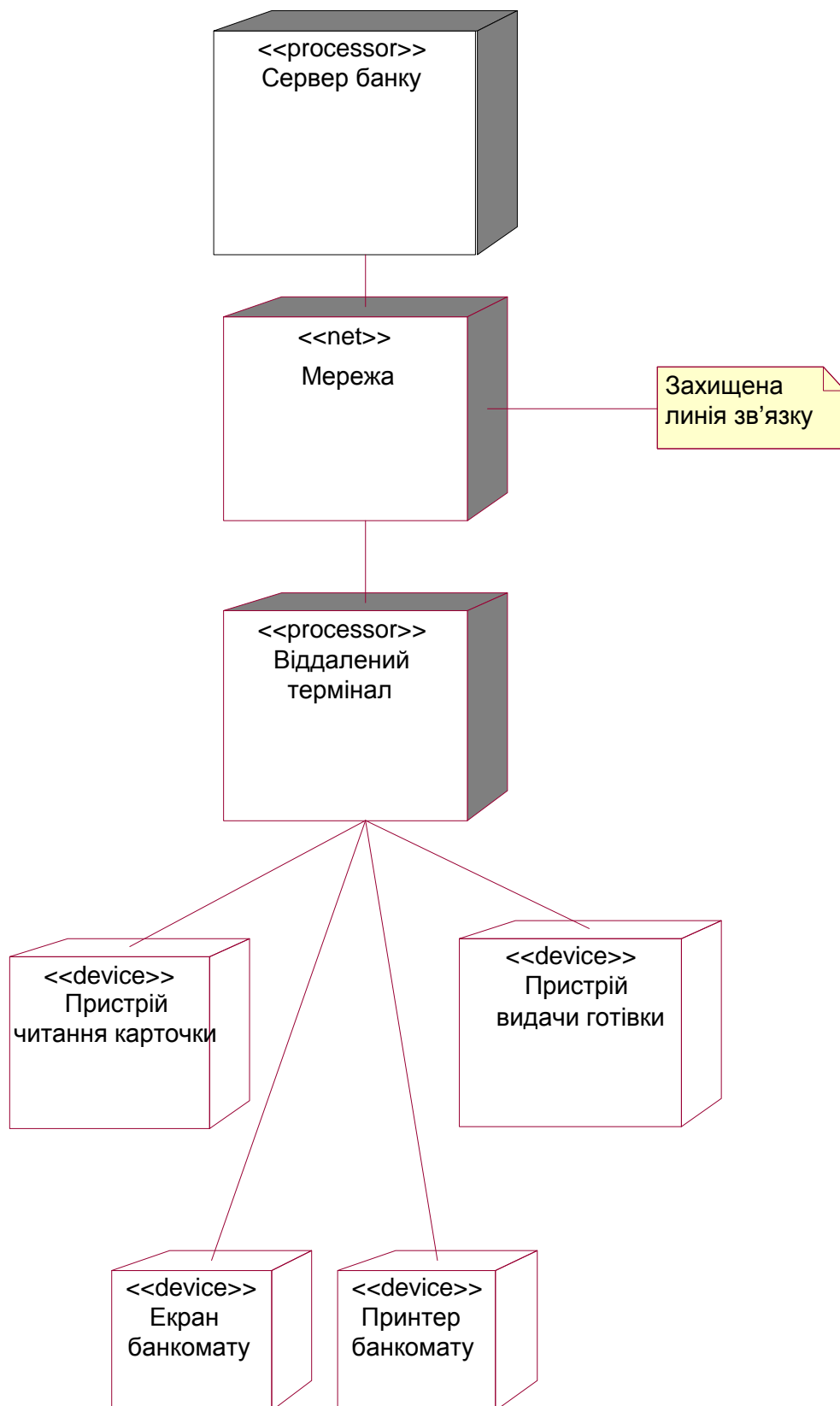


Рисунок 57 – Діаграма розгортання системи управління банкоматом

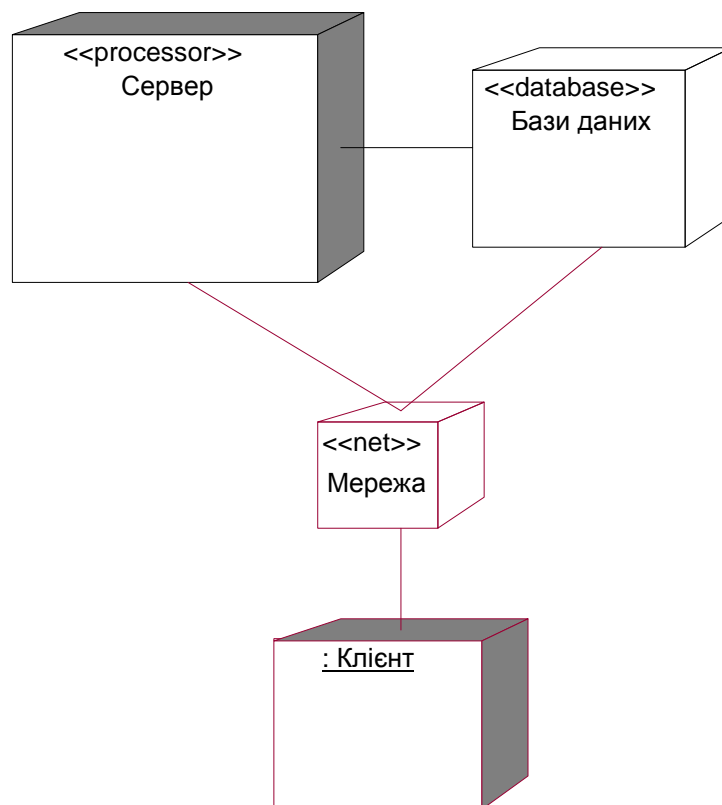


Рисунок 58 – Діаграма розгортання інформаційної системи

3 ПРОЕКТУВАННЯ ПРОГРАМНИХ СИСТЕМ З ВИКОРИСТАННЯМ CASE-ЗАСОБУ IBM RATIONAL ROSE

Проектувати систему на мові UML можна різними способами – зокрема «вручну», тобто зображати діаграми на листі паперу або в середовищі текстового процесора. Зрозуміло, що останній спосіб недоцільний через неможливість швидкого коректування, копіювання і перевірки на наявність помилок; звичайно використовуються так звані CASE-засоби (Computer Aided Software/System Engineering – проектування програм/систем за допомогою комп'ютера).

У даний час існує сім найпоширеніших засобів проектування [8] – IBM Rational Rose, Borland Together, Microsoft Visio, Sparx-Systems Enterprise Architect, GentleWare Poseidon, SmartDraw і Dia. Кожне з них має свою особливість (інтеграція з MS-Office, зручність використання, простота вивчення і т.п.), але фактично стандартом є IBM Rational Rose. Єдиним

недоліком цього могутнього пакету є достатня складність його освоєння. Зменшити негативний вплив цього недоліку покликаний даний розділ навчального посібника.

3.1 Загальна характеристика інструментарію IBM Rational Rose

CASE-засіб IBM Rational Rose дозволяє побудувати всі канонічні UML-діаграми у рамках єдиної моделі, перевірити модель на наявність помилок і здійснити експорт у вигляді кодів програм.

Спроектована модель зберігається у файлі з розширенням MDL, резервні копії – у файлах з розширенням MD~. Одночасно можна працювати тільки з однією моделлю – при завантаженні нової попередня автоматично закривається.

Робота починається з вибору майбутнього середовища реалізації (рис. 59). Якщо середовище поки точно не визначене, рекомендується вибрати «Rational unified process».

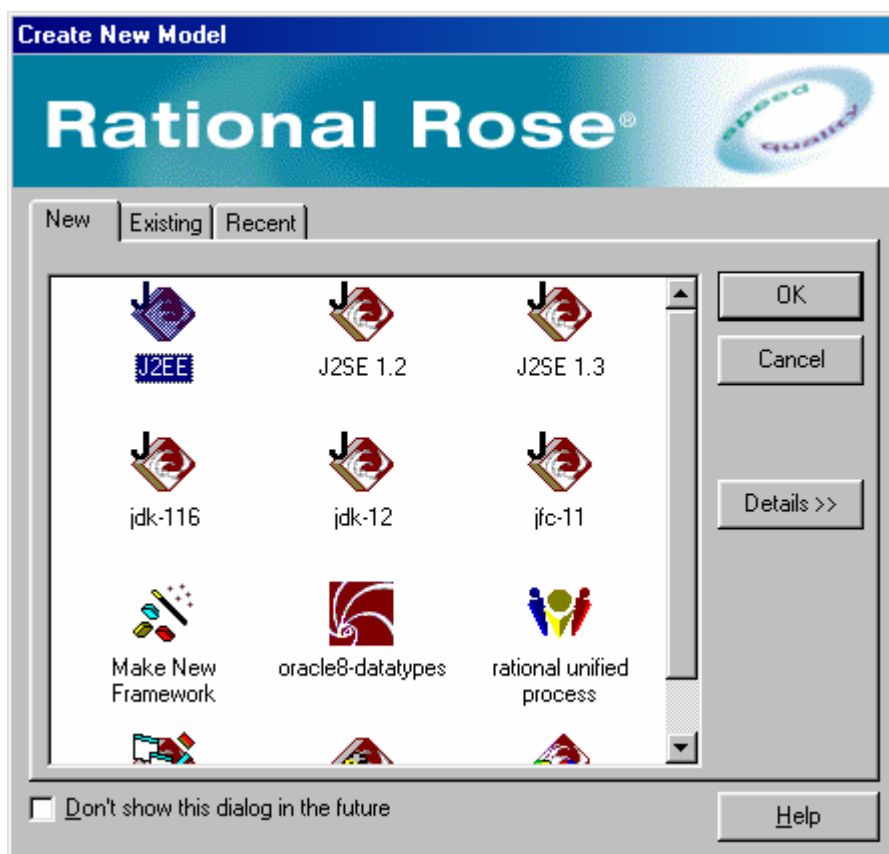


Рисунок 59 – Вікно вибору середовища реалізації

Інтерфейс IBM Rational Rose оформлений за аналогією з інтерфейсами більшості Windows-додатків, тому немає значення зупинятися на пунктах головного меню і докладному переліку змісту панелі інструментів (рис.60).

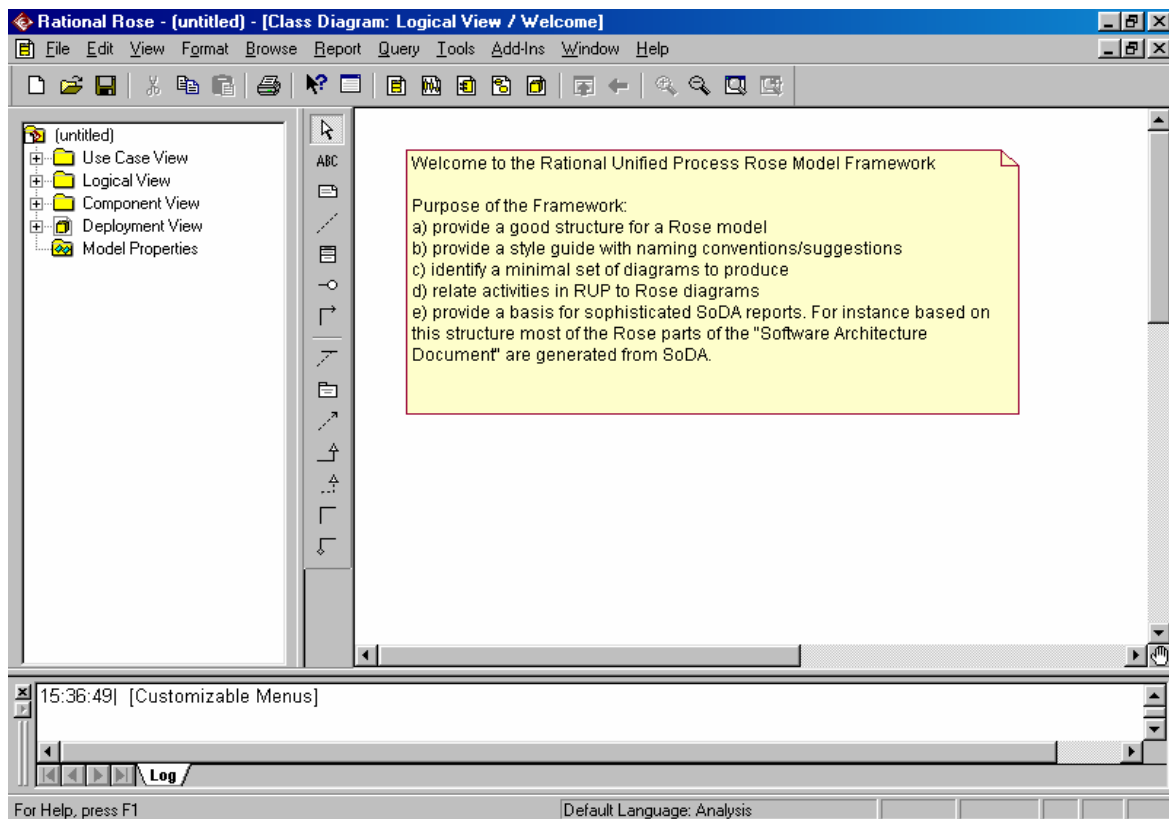


Рисунок 60 – Робочий інтерфейс середовища IBM Rational Rose

У лівій частині екрана розташовується **вікно браузера проекту**, в якому можна бачити проектовану систему у вигляді ієрархічної структури, верхніми рівнями якої є «Концептуальне уявлення» (use case view), «Логічне уявлення» (logical view), «Компонентне уявлення» (component view) і «Подання розгортання» (deployment view).

У правій частині екрана розташовується **вікно діаграми**, де, власне, і відбувається процес проектування. Між вікном браузера і вікном діаграми розташовується **спеціальна панель інструментів**, зміст якої залежить від вибраної діаграми. Склад цієї панелі можна змінювати (пункт Customize контекстного меню).

Внизу екрана знаходиться вікно журналу, куди виводиться службова інформація про виконані дії.

Перемикання між діаграмами здійснюється або натисненням відповідного значка на панелі інструментів, або вибором з головного меню (Browse).

Решта особливостей роботи у середовищі IBM Rational Rose буде зрозуміла надалі при розгляді прикладу розробки моделі простої інформаційної системи.

3.2 Приклад розробки моделі інформаційної системи у середовищі IBM Rational Rose

Згідно з RUP (раціональним уніфікованим процесом) проектування системи повинне починатися з побудови концептуальної моделі – тобто з діаграми варіантів використання.

Виберемо в головному меню пункт «Browse / Use Case Diagram» (або виберемо зліва «Use Case View / Main») – на екрані з'явиться нова діаграма варіантів використання з розміщеними за умовчанням пакетами (рис. 61).

Після видалення пакетів помістимо на діаграму актора (розміщення компонентів здійснюється стандартним прийомом «виділи і клацни на полі»), при цьому відразу введемо його ім'я (у нашому прикладі – Admin) – рис. 62. Аналогічно помістимо другого актора (User).

Наша модель припускає два основні варіанти використання – «Введення і модифікація даних» і «Робота з даними». Примищення варіанта використання відбувається подібно приміщенню актора (рис. 63).

Перший варіант використання пропонується адміністратору, другий – користувачу, тому зв'яжемо їх асоціаціями (рис. 64).

Оскільки передбачається два користувачі (Admin і User), системі необхідно їх заздалегідь ідентифікувати. Для цього помістимо ще один варіант використання «Аутентифікація», пов'язавши його з двома іншими відношенням залежності типу «Включення» (рис. 65).

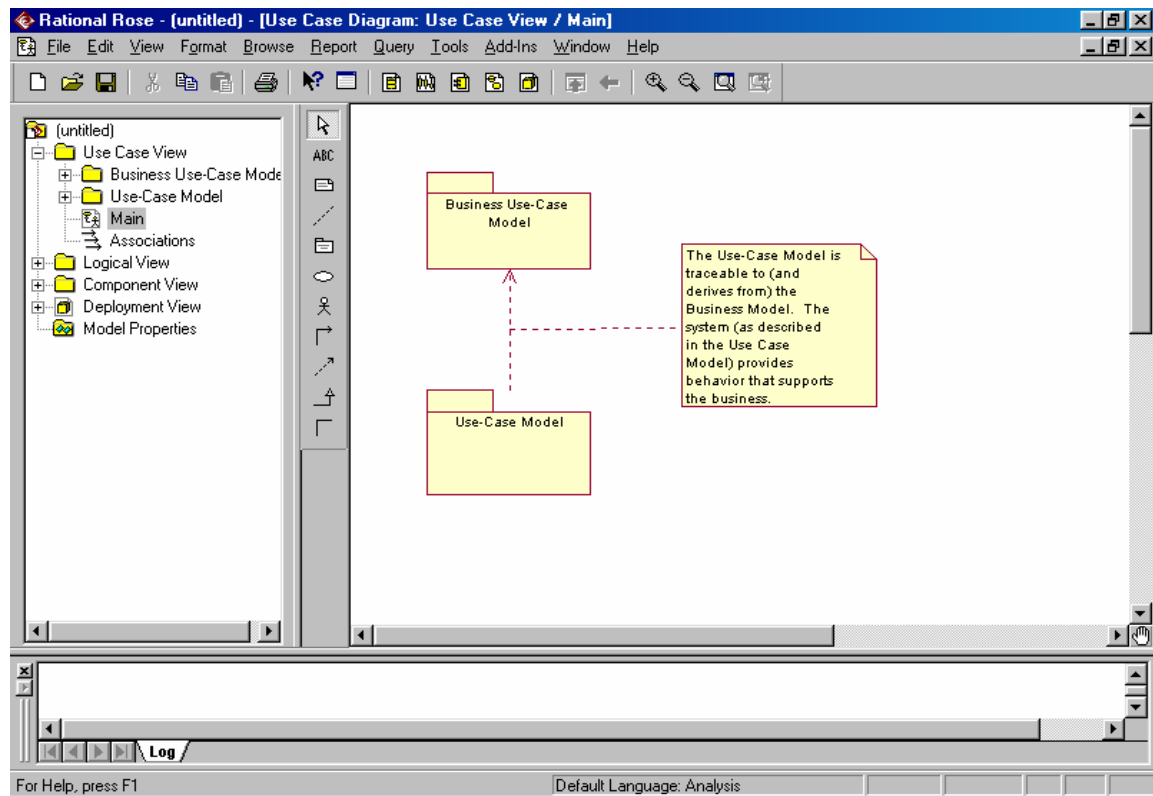


Рисунок 61 – Початок побудови діаграми варіантів використання

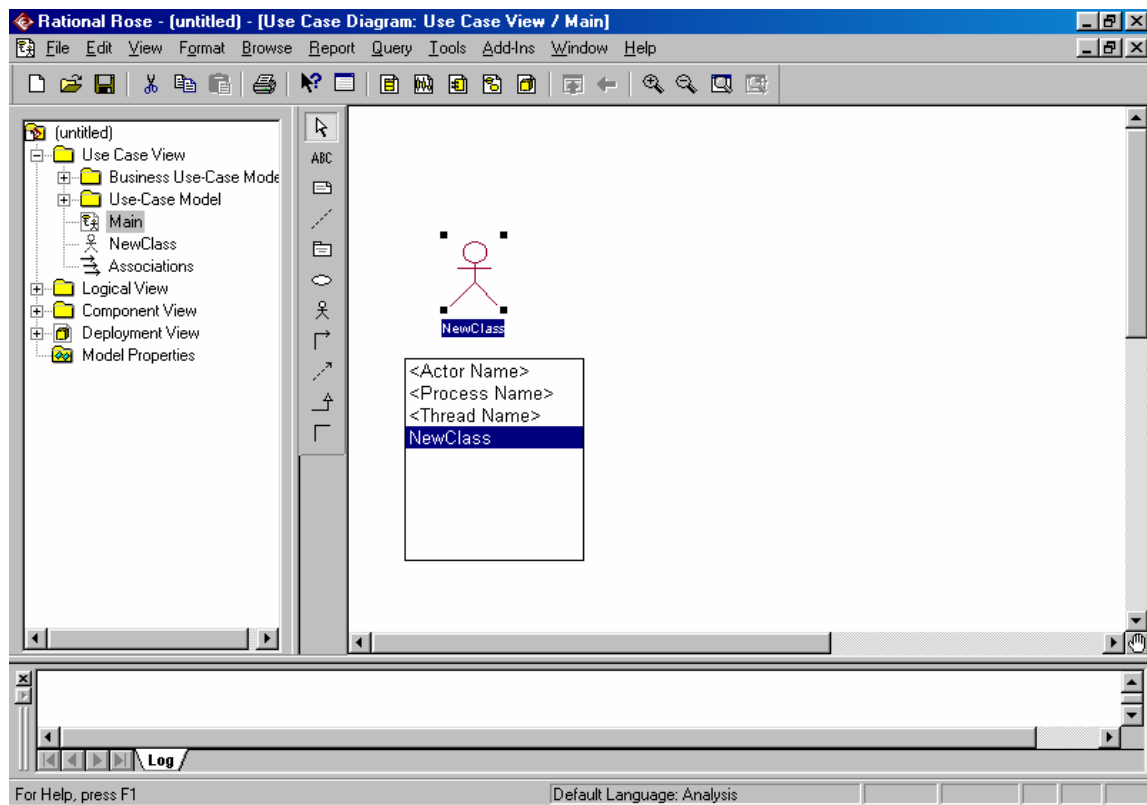


Рисунок 62 – Поміщення актора на діаграму варіантів використання

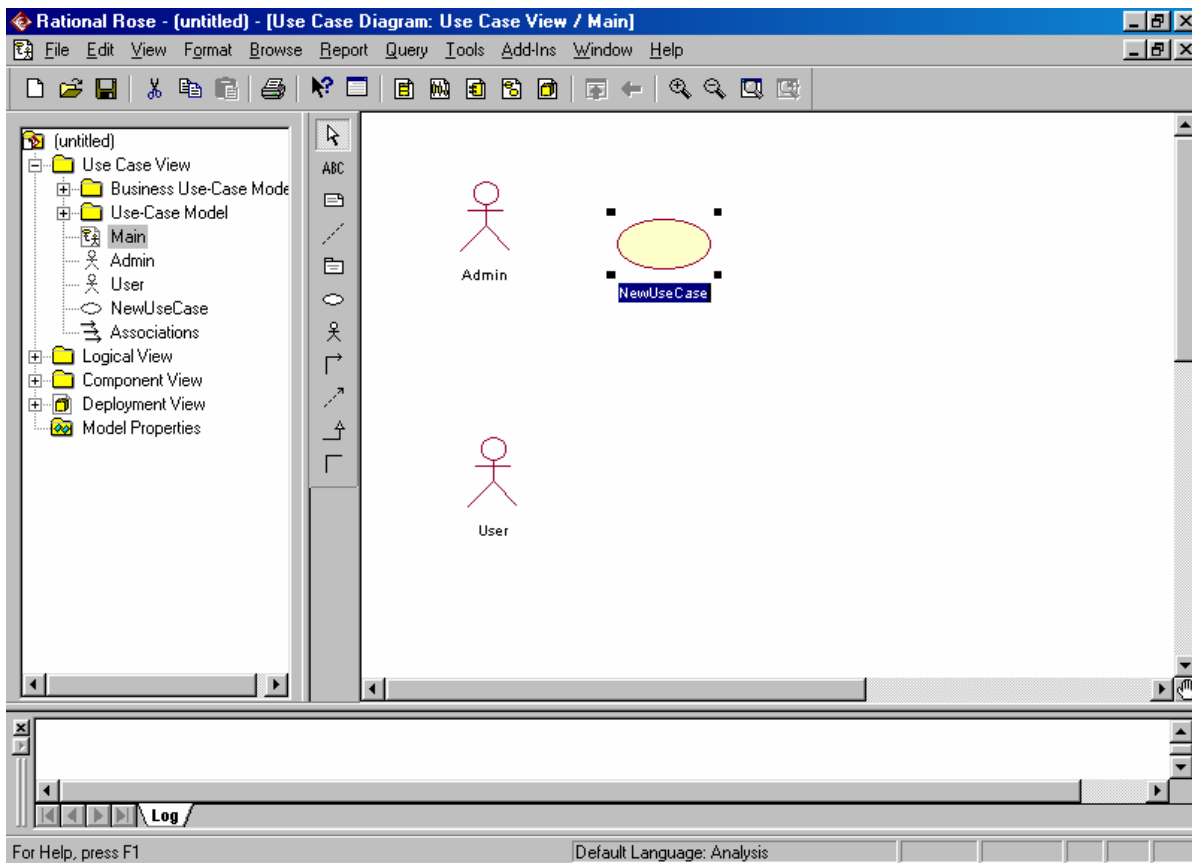


Рисунок 63 – Поміщення варіанту використання на діаграму

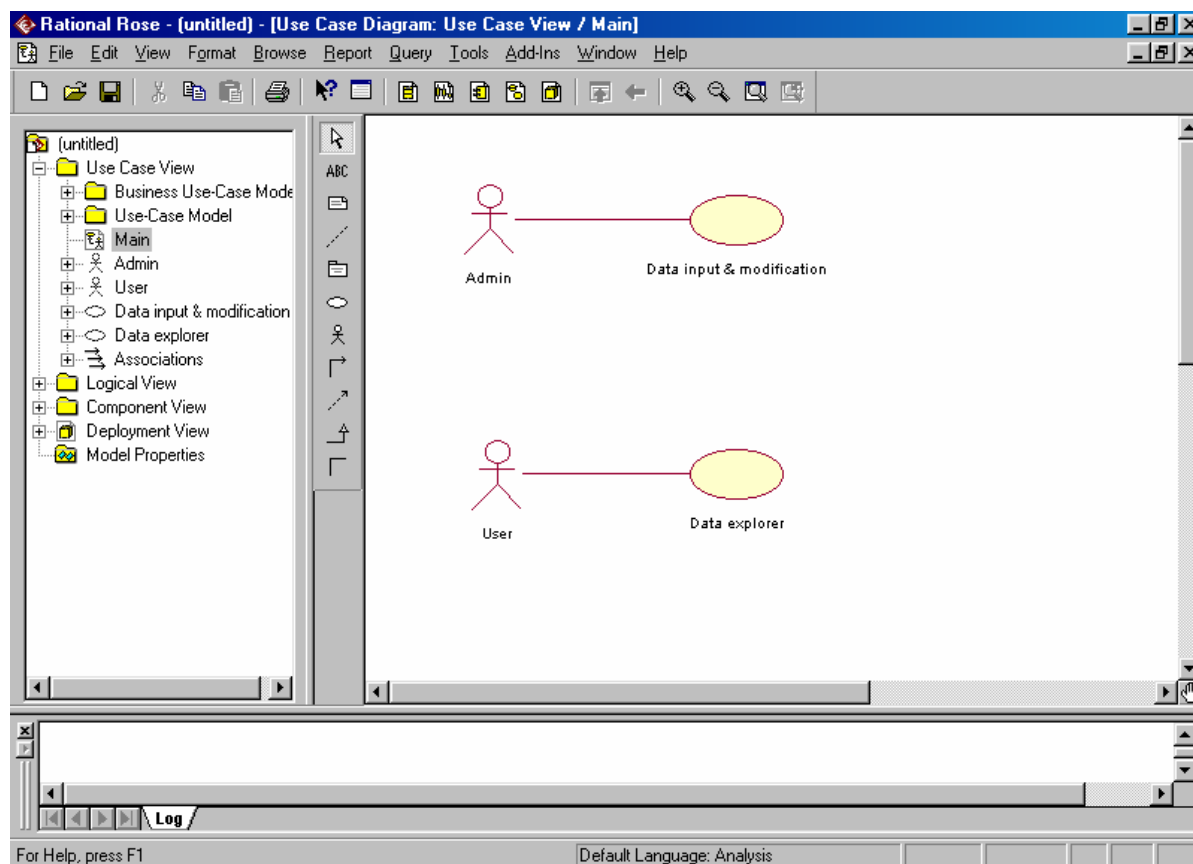


Рисунок 64 – Додавання зв'язків між компонентами

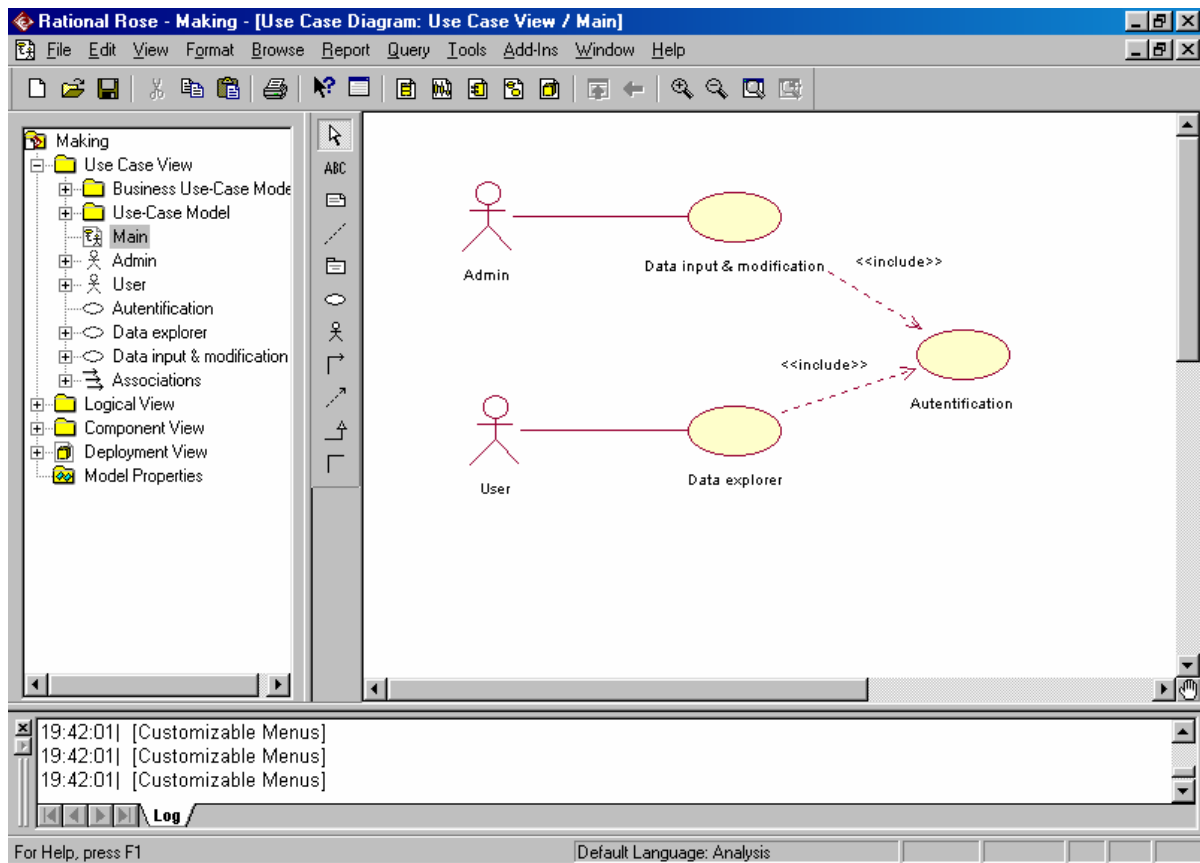


Рисунок 65 – Новий варіант використання

Вид залежності визначається подвійним клацанням миші і вибором «Stereotype» у вікні, що з'явилося (рис. 66).

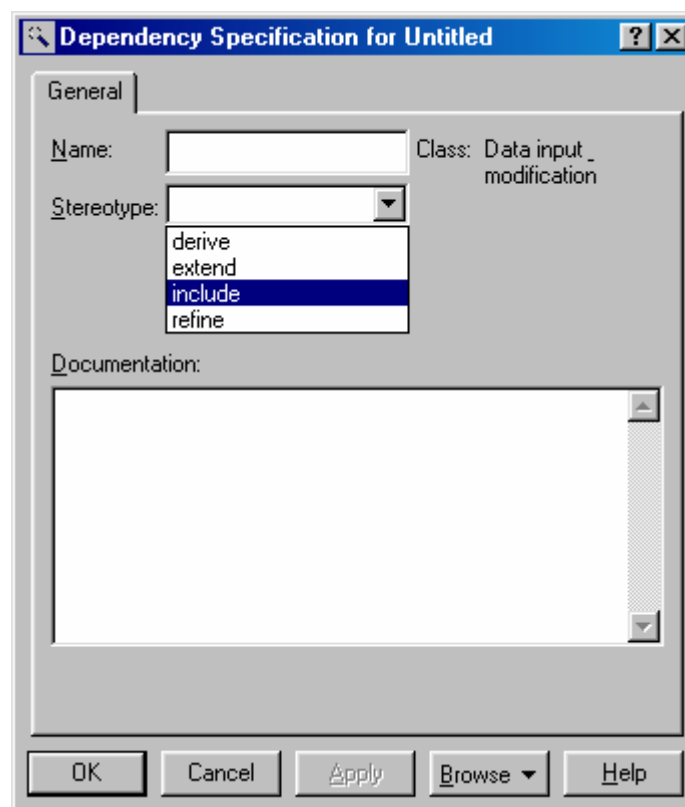


Рисунок 66 – Зміна стереотипу зв'язку

Остання зміна на діаграмі – додавання варіанта використання «Формування звіту» і скріплення його з «Роботою з даними» відношенням залежності типу «Розширення» (рис. 67).

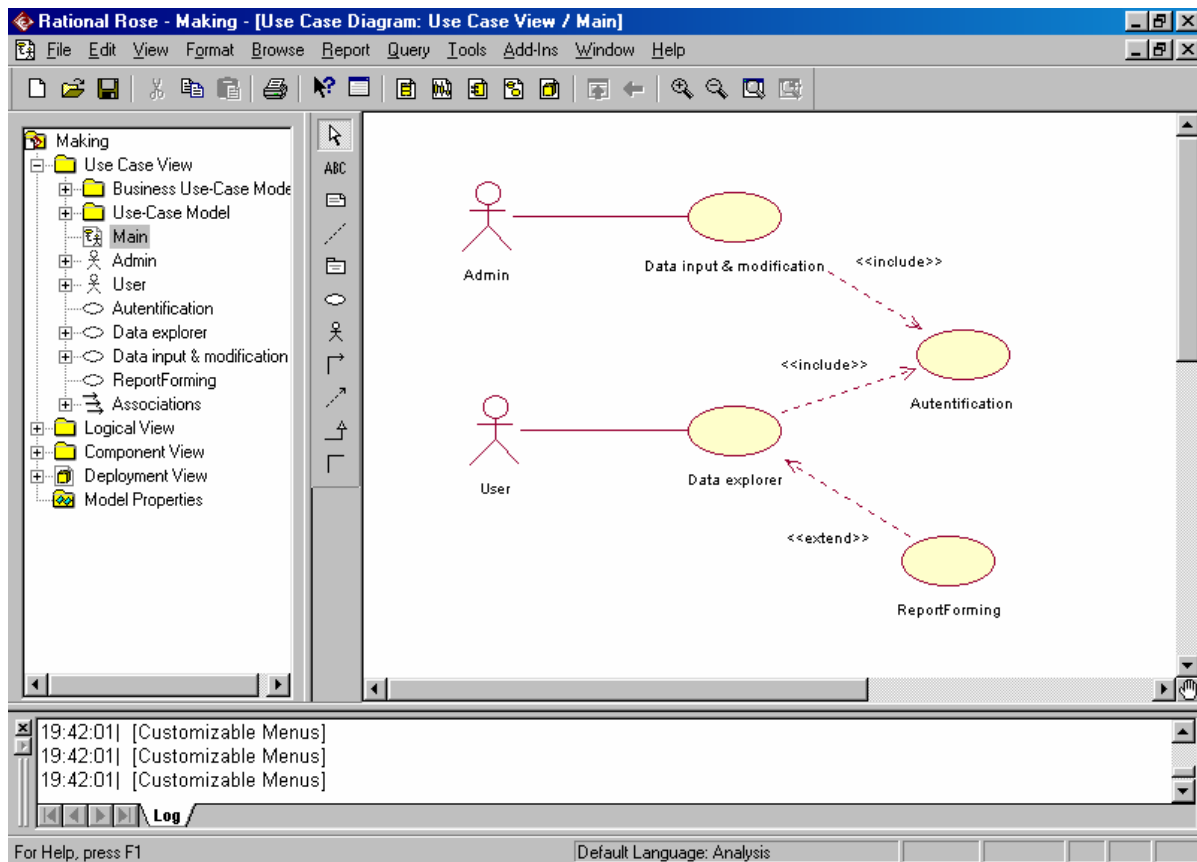


Рисунок 67 – Варіант використання «ReportForming»

При виникненні необхідності видалення елемента з моделі просте виділення і натиснення клавіші «Delete» не приведе до бажаного результату – зникне тільки його зображення. Повне видалення елемента здійснюється у вікні браузера проекту – контекстне меню, пункт «Delete».

Наступний етап – побудова діаграми класів. Виберемо в головному меню пункт «Browse / Class Diagram» (або виберемо зліва «Logical View / Welcome») – на екрані з'явиться нова діаграма. Привласнену за умовчанням назву «Welcome» краще змінити за допомогою контекстного меню (пункт «Rename») на більш відповідне за значенням (рис. 68). Побудова діаграми починається з розміщення нового класу.

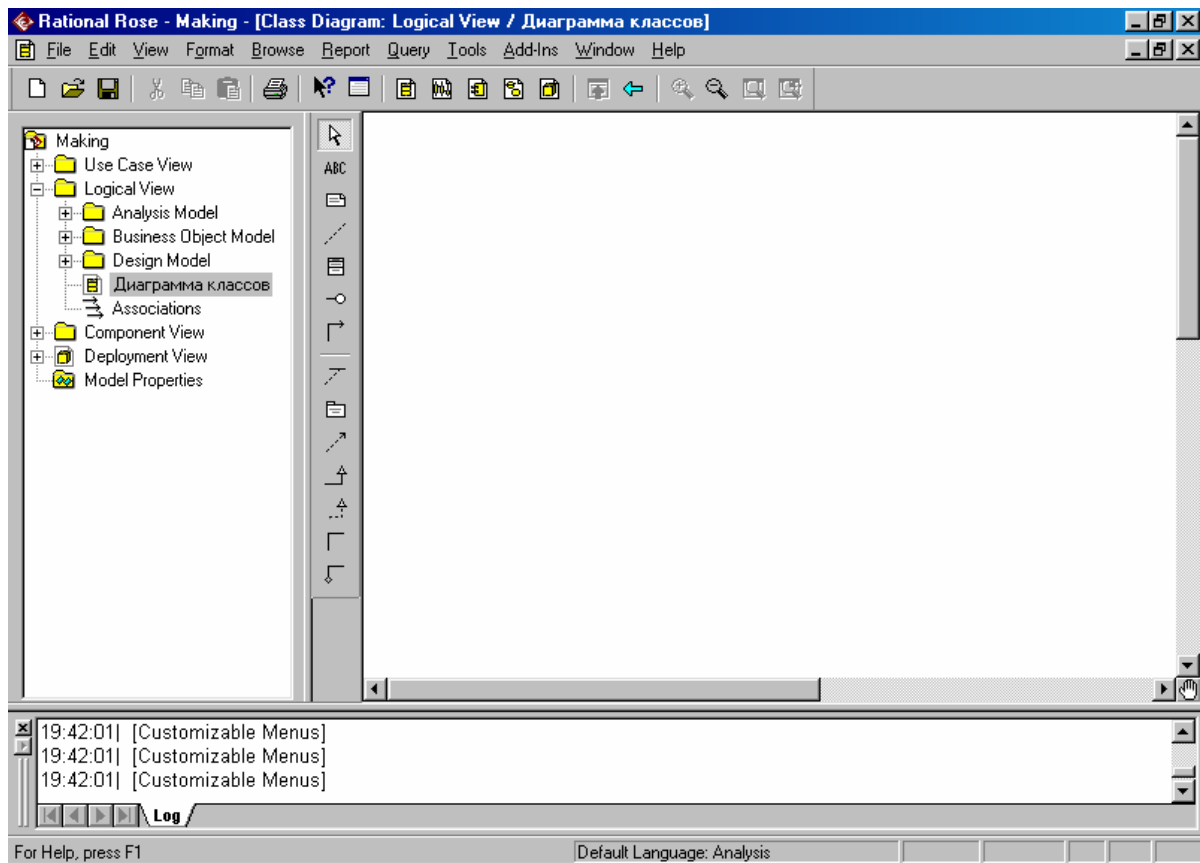


Рисунок 68 – Початок побудови діаграми класів

Нам буде запропонований вибір: ввести ім'я нового класу або скористатися існуючим (актори з діаграми варіантів використання пропонуються як класи). Спочатку введемо клас Admin, основою якого є відповідний актор (рис. 69).

Після створення класів, що описують обидва наші актори, введемо новий клас – програму. У вікні специфікації класу напишемо його ім'я (Program) і виберемо стереотип (control), оскільки клас є керівником (рис. 70). У результаті діаграма набуде вигляду, показаного на рисунку 71. Незручність такого подання управляючого класу позначиться при додаванні атрибутів і операцій, тому за допомогою контекстного меню («Options / Stereotype Display / Label») додамо йому стандартний вигляд.

Додавання атрибутів і операцій класу можна або у вікні специфікації класу («Attributes» і «Operations»), або за допомогою контекстного меню («New attribute» і «New operation»).

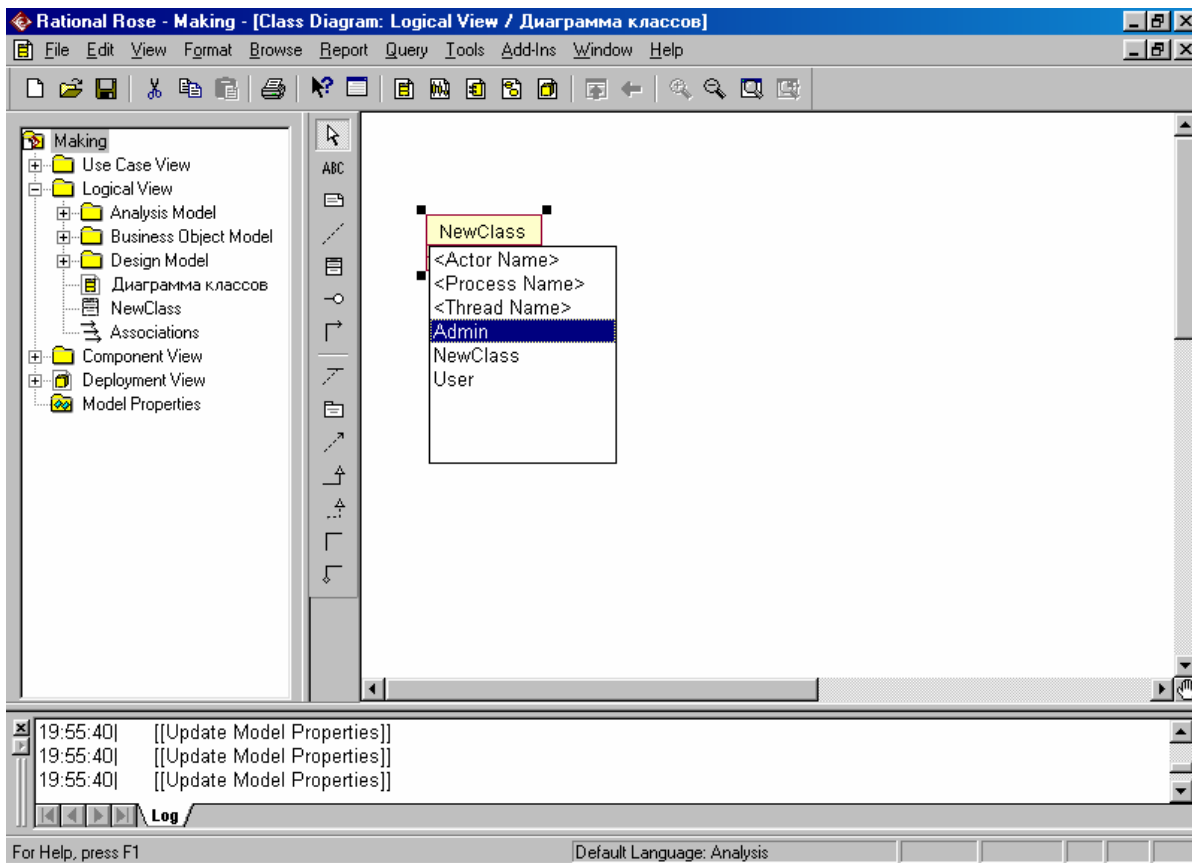


Рисунок 69 – Розміщення нового класу Admin

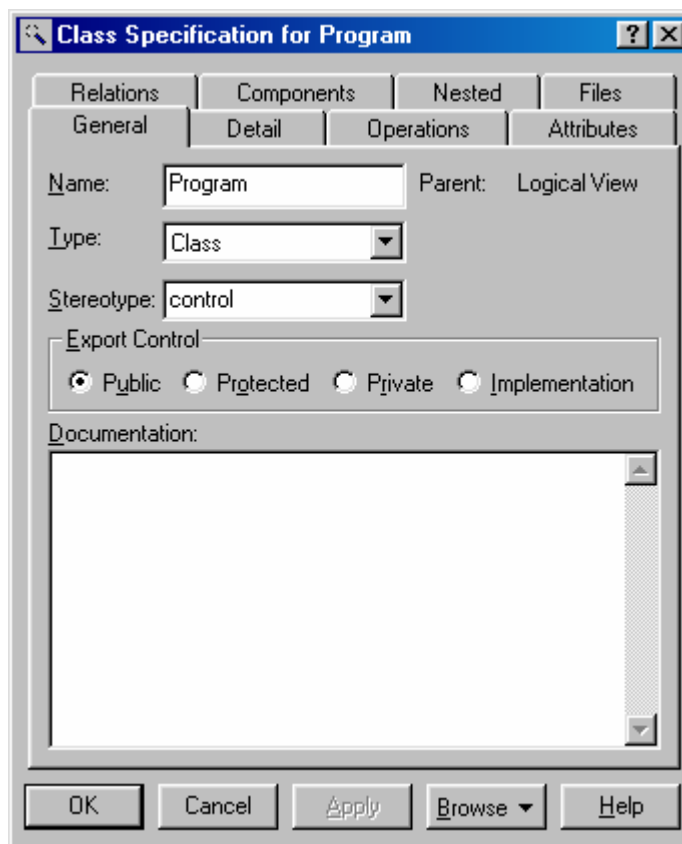


Рисунок 70 – Вікно специфікацій класу

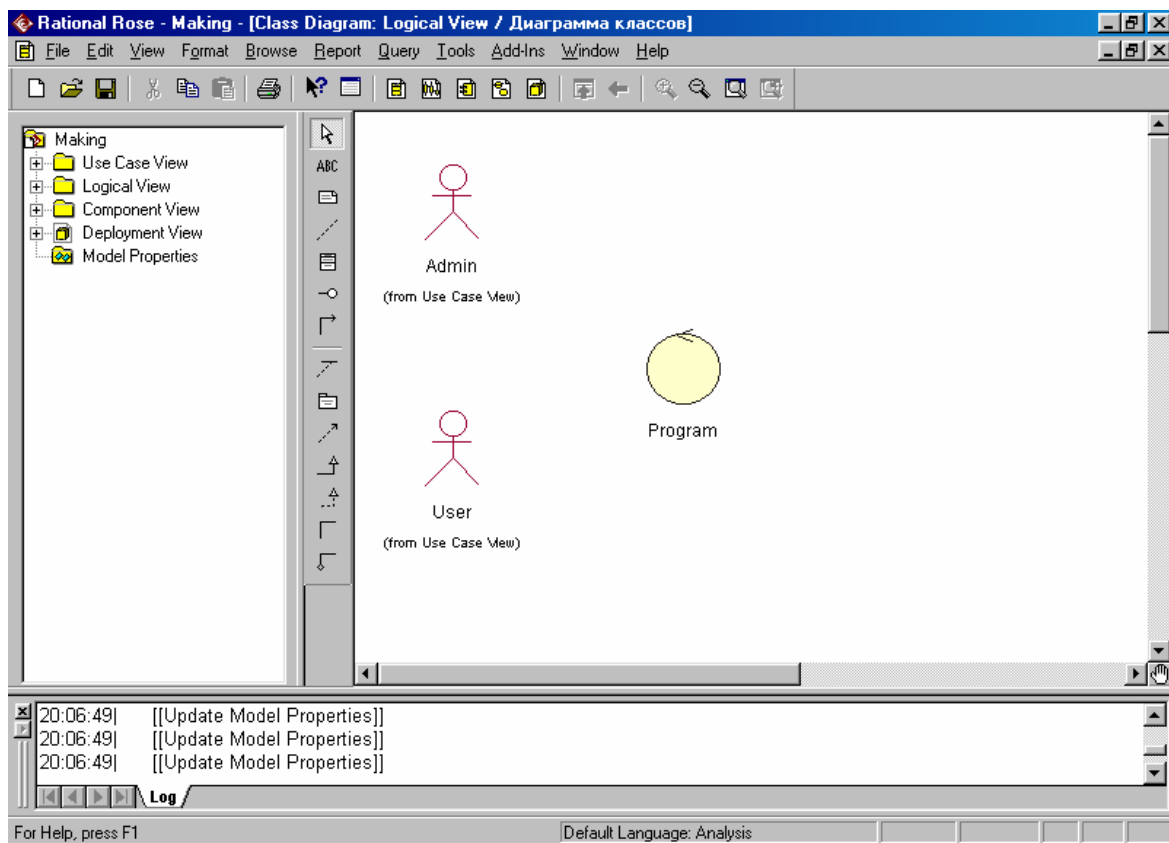


Рисунок 71 – Спеціальне зображення управляючого класу

Введення атрибутів головної програми у нашому прикладі недоцільне, а ось операцію «Авторизація користувача» додати необхідно. Тип значення, що повертається, вкажемо логічним (Boolean), видимість – «доступно для всіх» (public). Результат дій наведений на рисунку 72.

Тепер залишається додати два нові класи – «База даних» і «Звіт» (атрибут – «Дані», операції – «сформувати()», «роздрукувати()» і «експортувати()»), а також зв'язки між ними (рис. 73).

Після закінчення роботи з діаграмою класів починаємо будувати діаграму взаємодії, а саме – діаграму кооперації. Розглянемо тільки один приклад роботи системи – роботу користувача. Спочатку той проходить авторизацію у програмі, потім повинен витягнути інформацію з бази даних, сформувати і роздрукувати звіт.

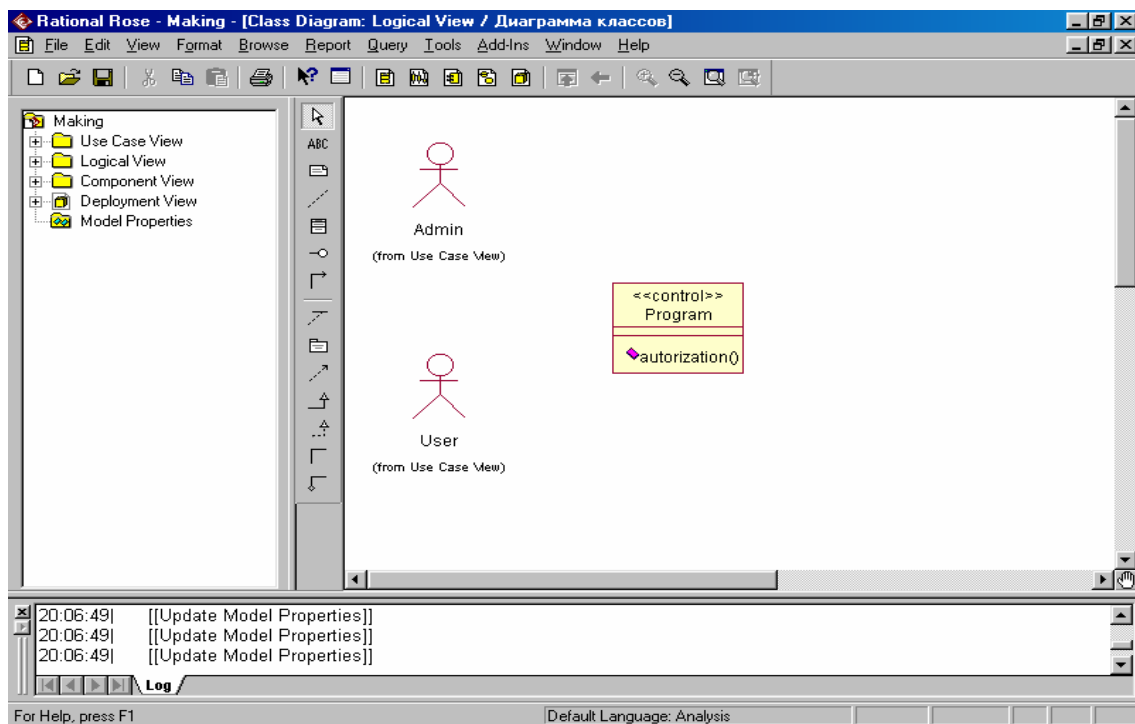


Рисунок 72 – Стандартне зображення управляючого класу з операцією

Створення діаграми відбувається у наступній послідовності: пункт головного меню пункт «Browse / Interaction Diagram», «New / Ok», введення імені («Діаграма кооперації») і вибір типу діаграми («Diagram type: Collaboration») – рис. 74.

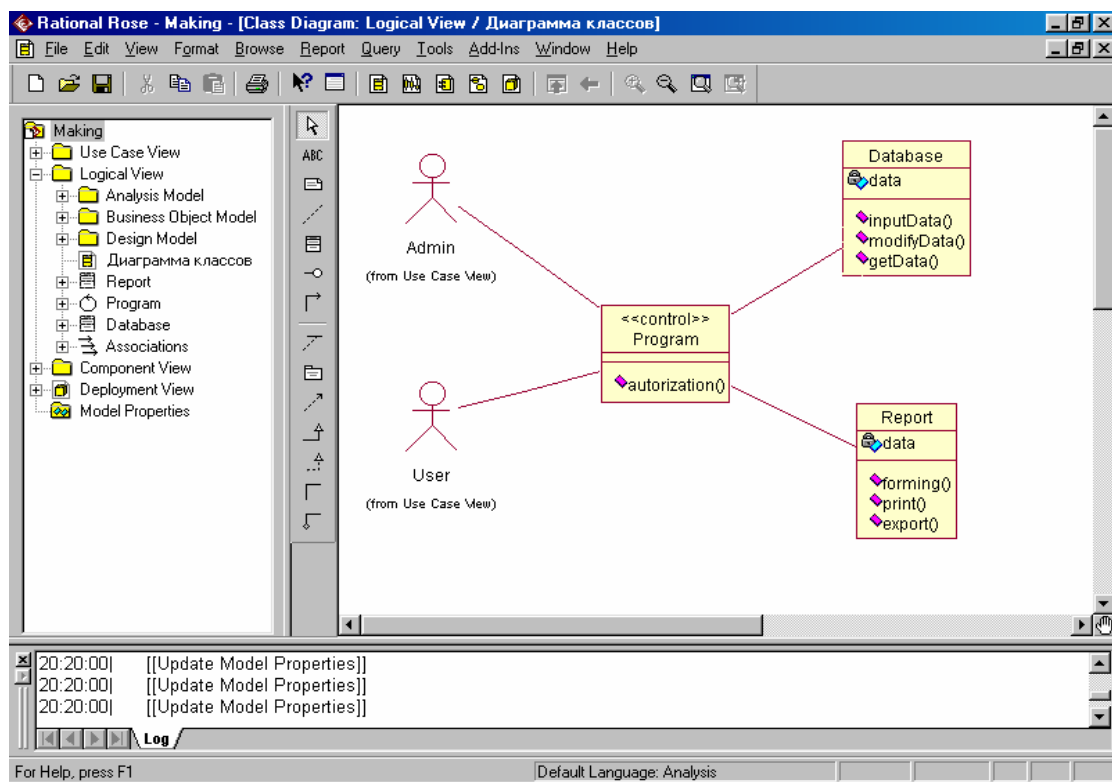


Рисунок 73 – Остаточний вид діаграми класів

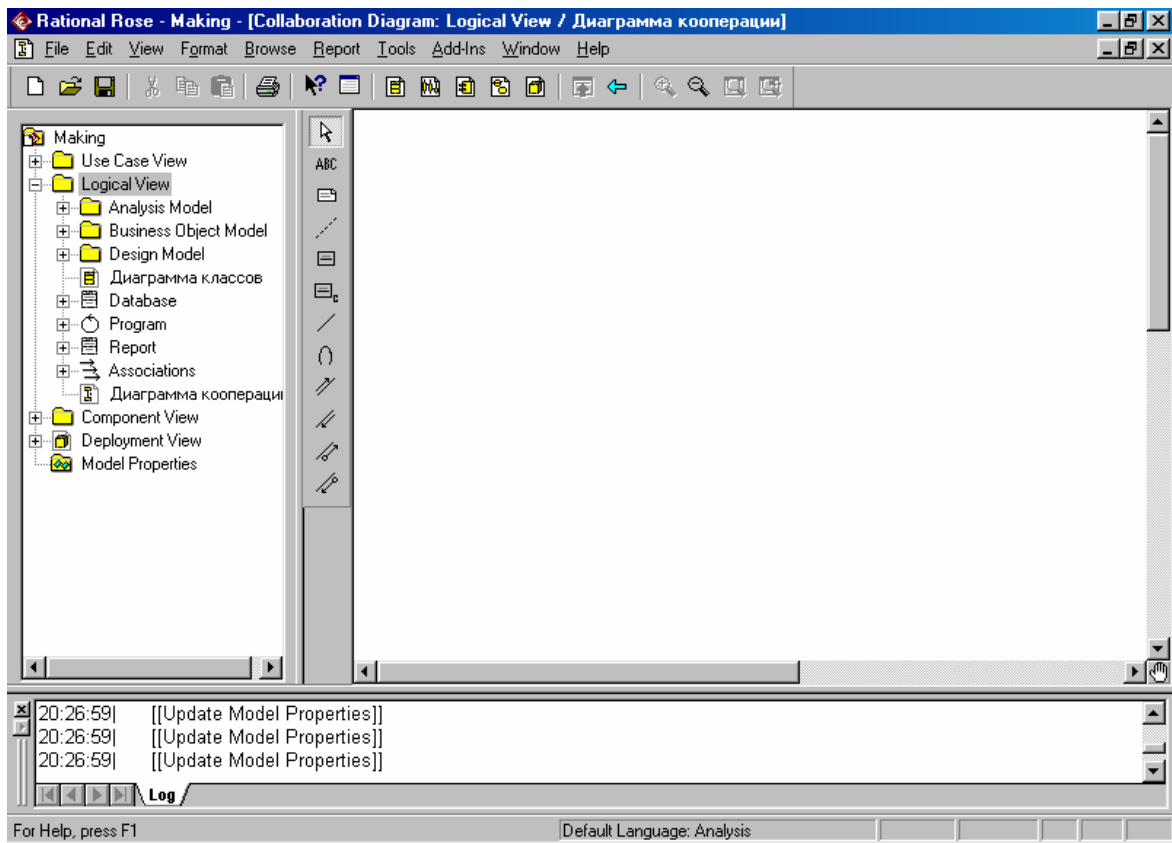


Рисунок 74 – Початковий вид діаграми кооперації

Помістимо у вікно діаграми новий об'єкт і виберемо в його вікні специфікації тип, що цікавить нас, – User (рис. 75). Оскільки власне ім'я користувача нам в даному випадку абсолютно неважливе, рядок «Name» залишимо порожнім (анонімний об'єкт). Аналогічну дію здійснимо у відношенні до об'єкта класу «Програма» (міняти зовнішній вигляд об'єкта тут немає необхідності) і з'єднаємо два об'єкти лінією («Object Link») – рис. 76.

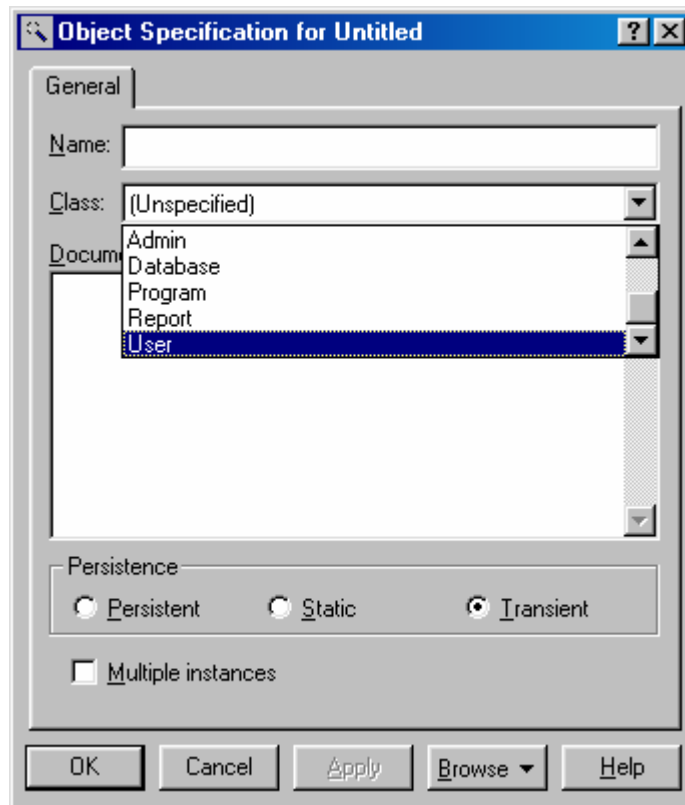


Рисунок 75 – Вікно специфікації об'єкта

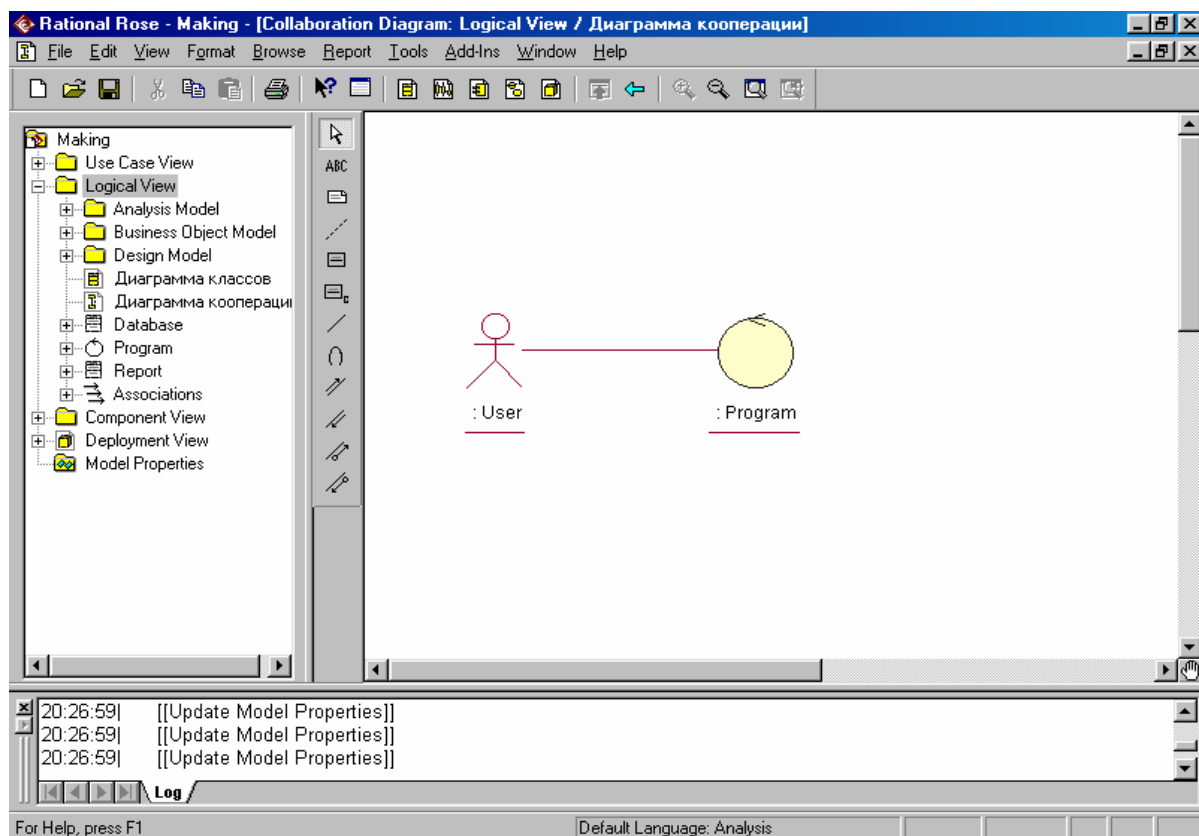


Рисунок 76 – Два об'єкта на діаграмі кооперації

Помістити на лінію зв'язку повідомлення можна двома способами. Перший – викликавши подвійним клацанням миші на лінії вікно її специфікації, у розділі «Messages» за допомогою контекстного меню додати повідомлення (вибрати при цьому пункт «Insert To: Program», тобто явно вказати напрям). При цьому доступні операції будуть показані у вигляді випадаючого меню (рис. 77).

Другий спосіб – виділити на спеціальній панелі інструментів Link message і помістити його на лінію зв'язку. Потім у вікні специфікації цього повідомлення вибрати з випадаючого меню доступних операцій ту, яка цікавить нас. Результат буде ідентичний першому (рис. 78).

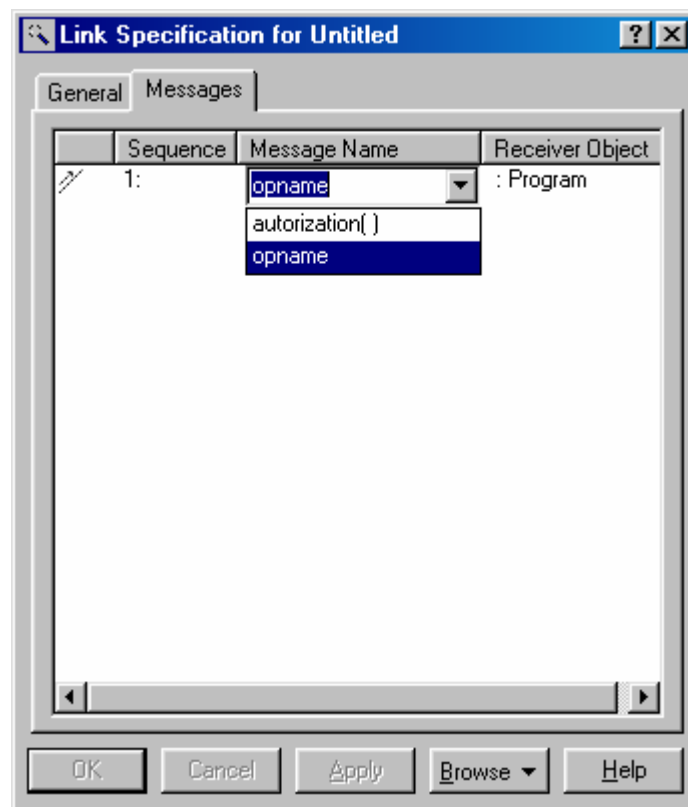


Рисунок 77 – Додавання повідомлення у вікні специфікації зв'язку

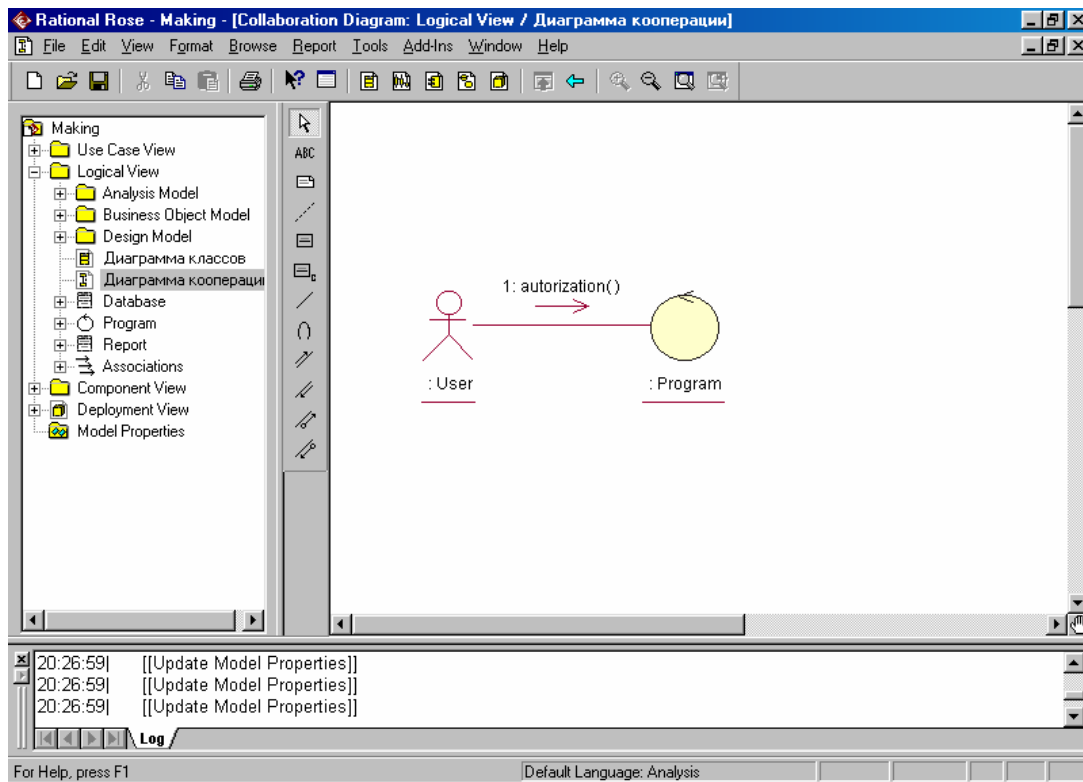


Рисунок 78 – Повідомлення на діаграмі кооператії

Тепер помістимо на діаграму два об'єкти (анонімні об'єкти класів «База даних» і «Звіт»), що залишилися, і пов'яжемо їх з програмою (рис. 79).

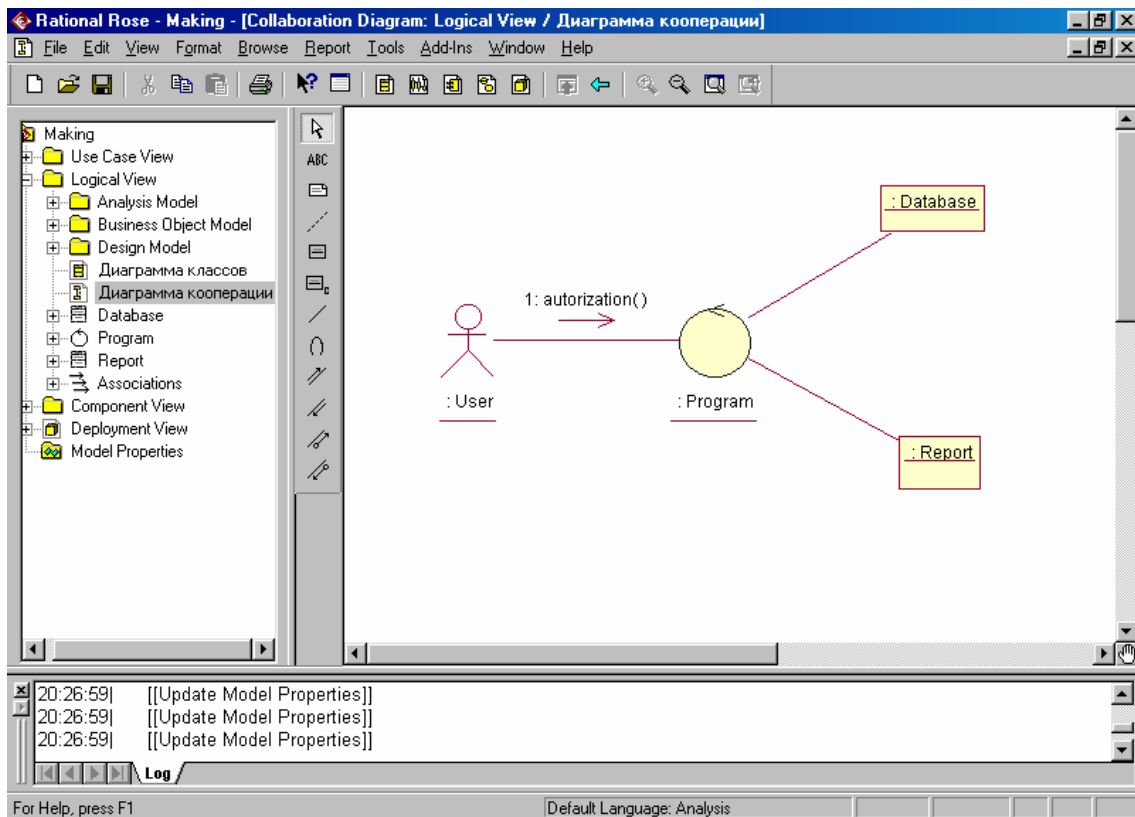


Рисунок 79 – Усі розміщені об'єкти на діаграмі кооператії

Другим повідомленням працюючої системи буде, очевидно, запит від програми до «Бази даних» – виклик операції «витягтиДані()», третім – запит від програми до «Звіту» – виклик операції «сформувати()», четвертим – запит до того ж об’єкта виклику операції «Друк()». Остаточний вид діаграми кооперації поданий на рисунку 80.

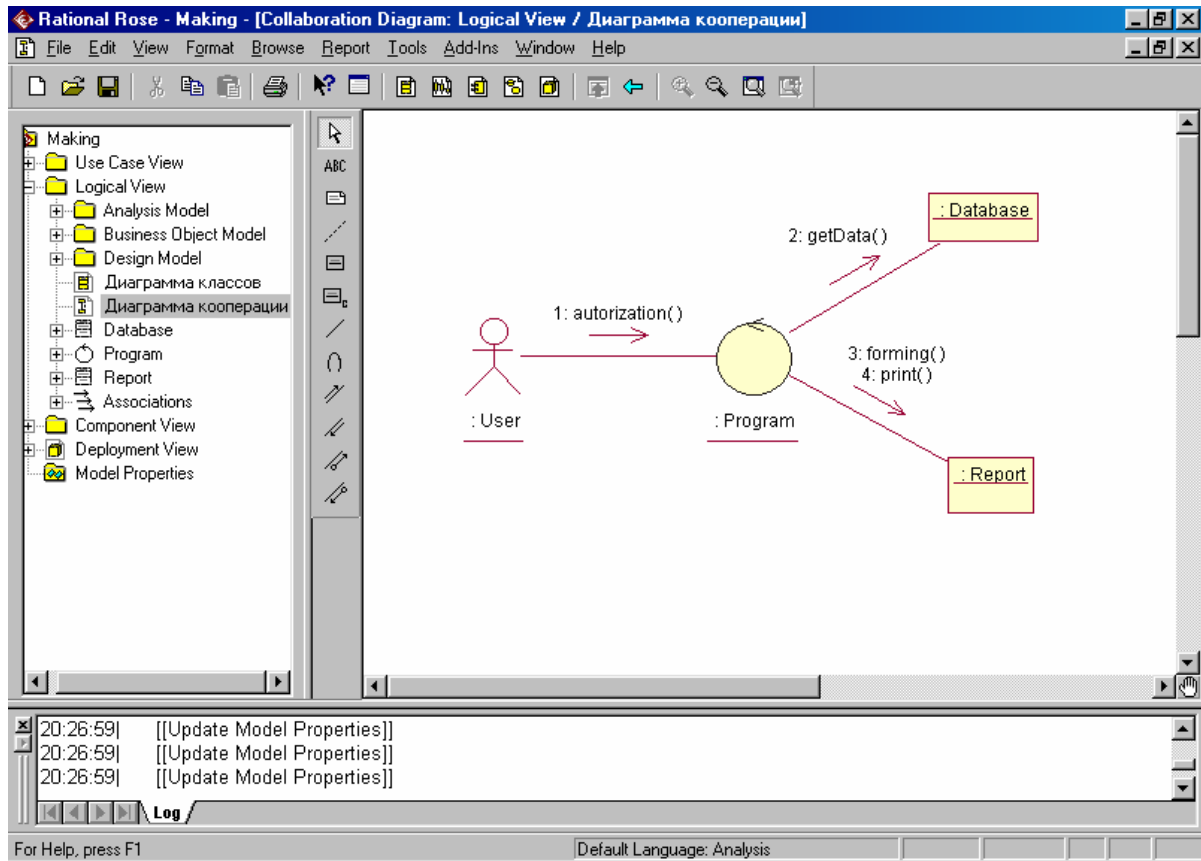


Рисунок 80 – Остаточний вид діаграми кооперації

Діаграма послідовності формується на підставі діаграми кооперації пунктом меню «Browse / Create Sequence Diagram» або просто натисненням клавіші F5. Як видно на рис. 81, у вікні браузера проектів тепер розташувалися дві «Діаграми кооперації», що, по суті, невірно; привласнити діаграмі нове ім'я можна за допомогою контекстного меню (пункт «Rename»).

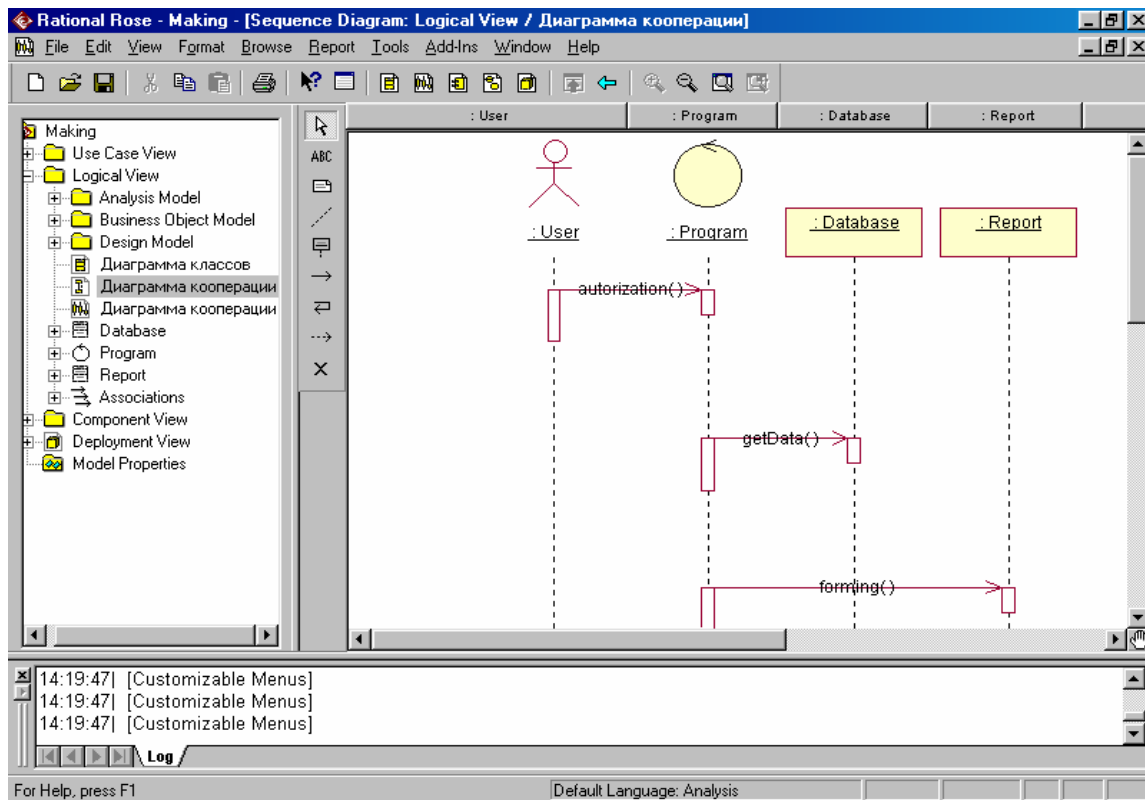


Рисунок 81 – Автоматично сформована діаграма послідовності

Необхідно виконати ще декілька коректувань діаграми послідовності, велика частина яких носить «косметичний» характер. Остаточний вигляд див. на рисунку 82.

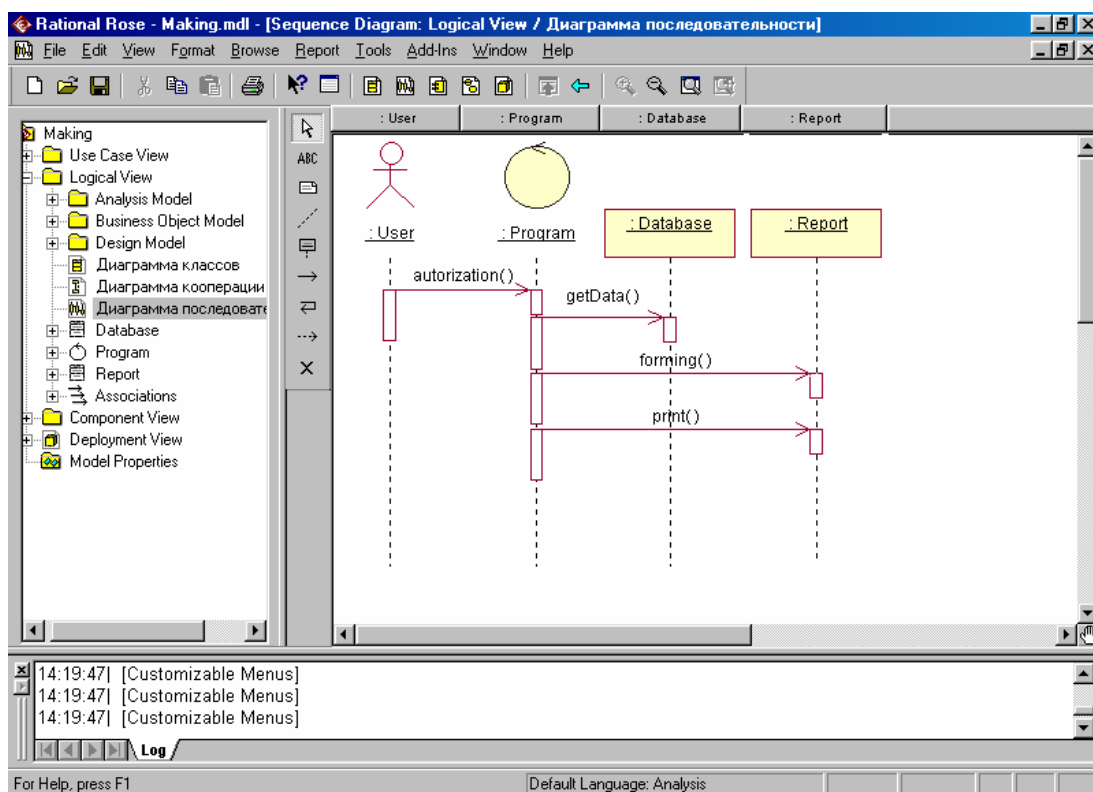


Рисунок 82 – Остаточний вид діаграми послідовності

Діаграма станів створюється пунктом меню «Browse / State Machine Diagram» (або натисненням клавіш Ctrl+T), «New / Ok», потім потрібно ввести ім'я діаграми («Діаграма станів») і її тип («Diagram Type: Statechart»). У браузері проектів нова діаграма розміститься у гілці «Logical View / State-Activity Model» (рис. 83).

Очевидно, що наша система може знаходитися в семи різних станах (не рахуючи початкового і кінцевого): «Очікування введення пароля», «Перевірка пароля», «Вибір даних для звіту», «Формування звіту», «Очікування вибору» (куди направляти звіт), «Друк звіту» і «Експорт звіту».

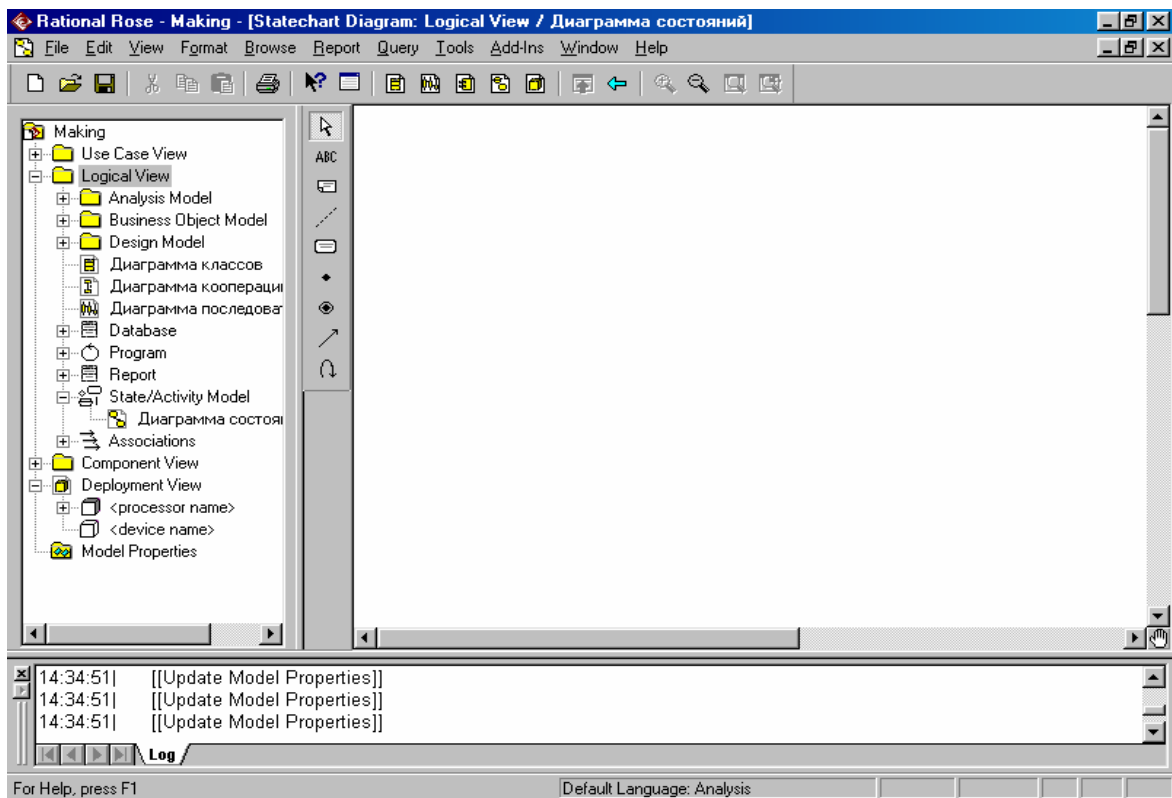


Рисунок 83 – Початковий вид діаграми станів

Помістимо на діаграму початковий (чорний кружок) і перший стани (рис. 84); очевидно, що його слід назвати «Очікування введення пароля» (ім'я вводиться подвійним клацанням миші або за допомогою вікна специфікації стану).

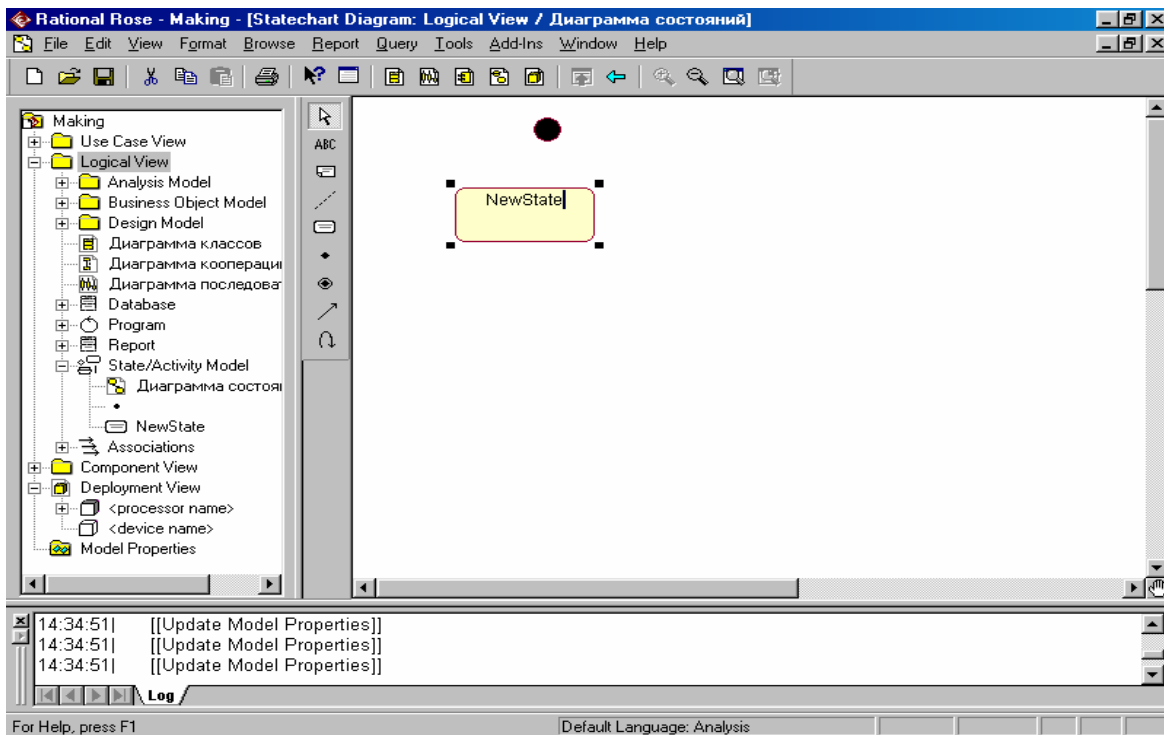


Рисунок 84 – Додавання нового стану

З'єднаємо початковий і перший стан лінією зв'язку (State Transition), у вікні специфікації переходу вкажемо назву події (General / Event) – «Завантаження програми» (рис. 85).

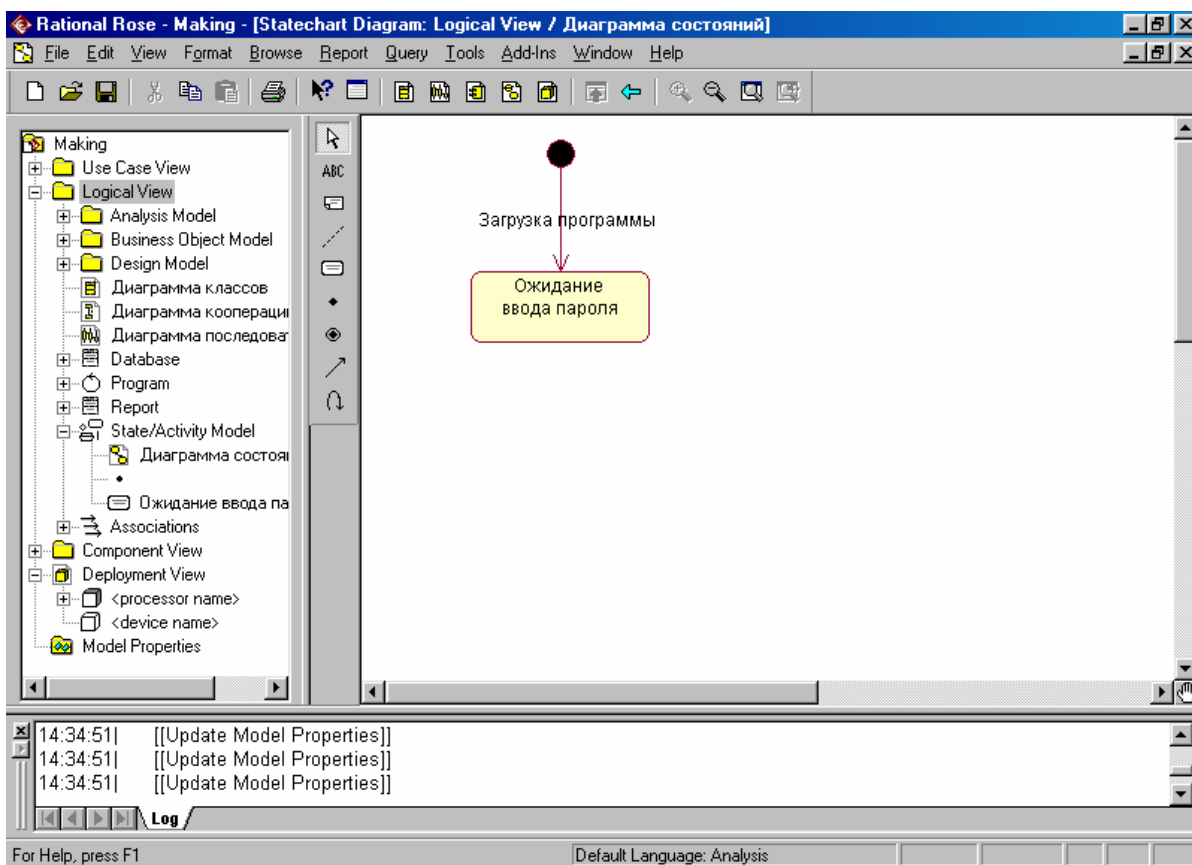


Рисунок 85 – Додавання першого переходу і першої події

Перехід до нового стану – «Перевірка пароля» – може здійснитися при настанні події «Пароль введений» (рис. 86).

Звідси можливі відразу три переходи: якщо введений правильний пароль – на «Вибір даних для звіту», неправильний пароль – повернення до попереднього стану, неправильний пароль вводиться підряд три рази – вихід. Власне подією тут можуть вважатися «Три неправильні спроби», решту умов введемо як сторожові у вікні специфікації переходу (Detail / Guard Condition) (рис. 87).

Перехід, викликаний трьома неправильними спробами, можна конкретизувати у вікні його специфікації: окрім безпосередньої назви, у розділі «Detail / Action» можна явно задати дію («Вихід») (рис. 88).

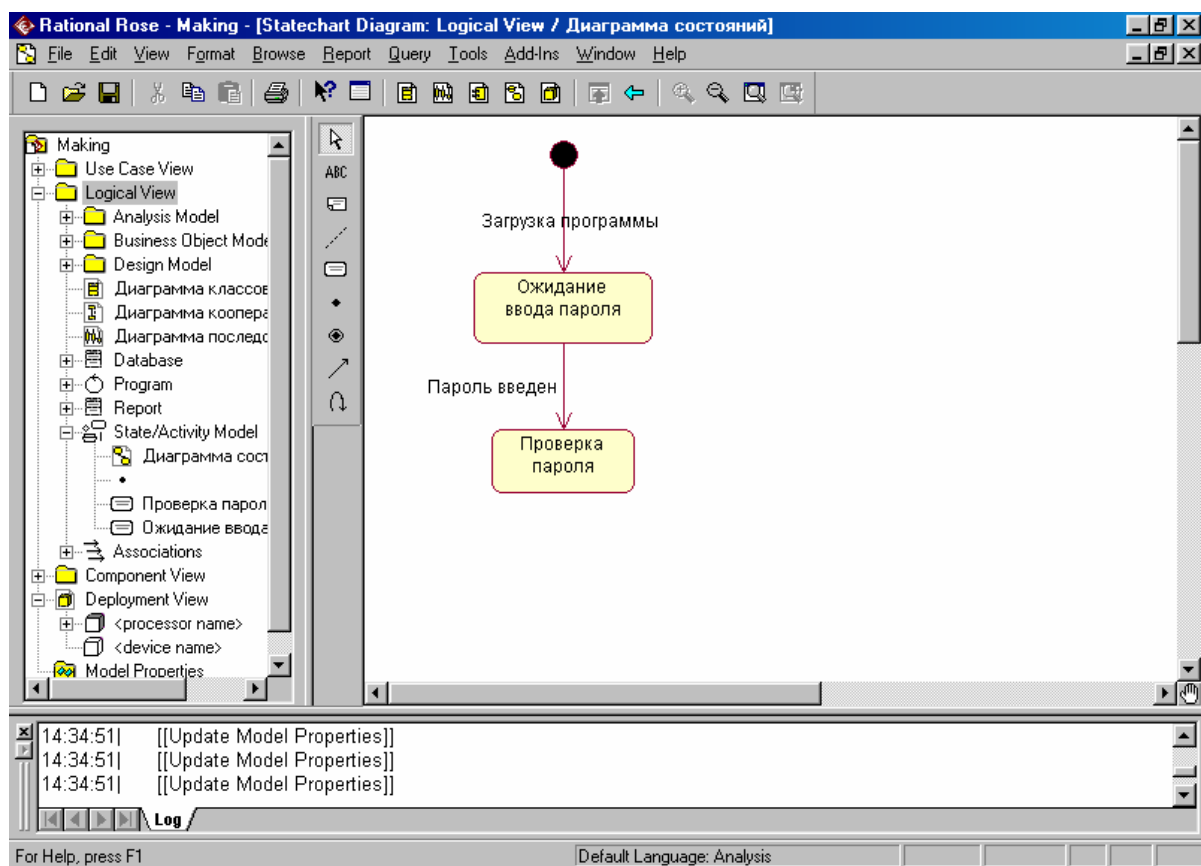


Рисунок 86 – Додавання першого переходу і першої події

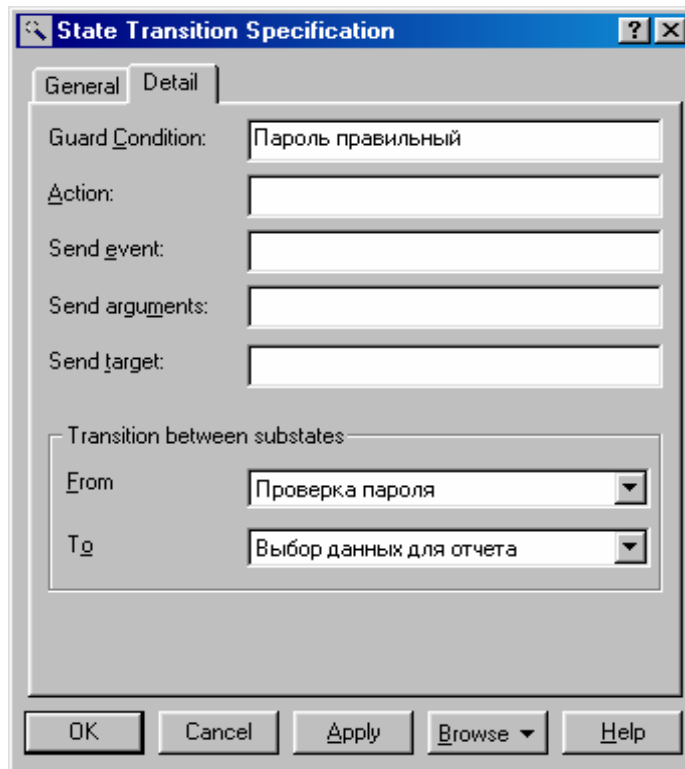


Рисунок 87 – Додавання сторожової умови у вікні специфікації переходу

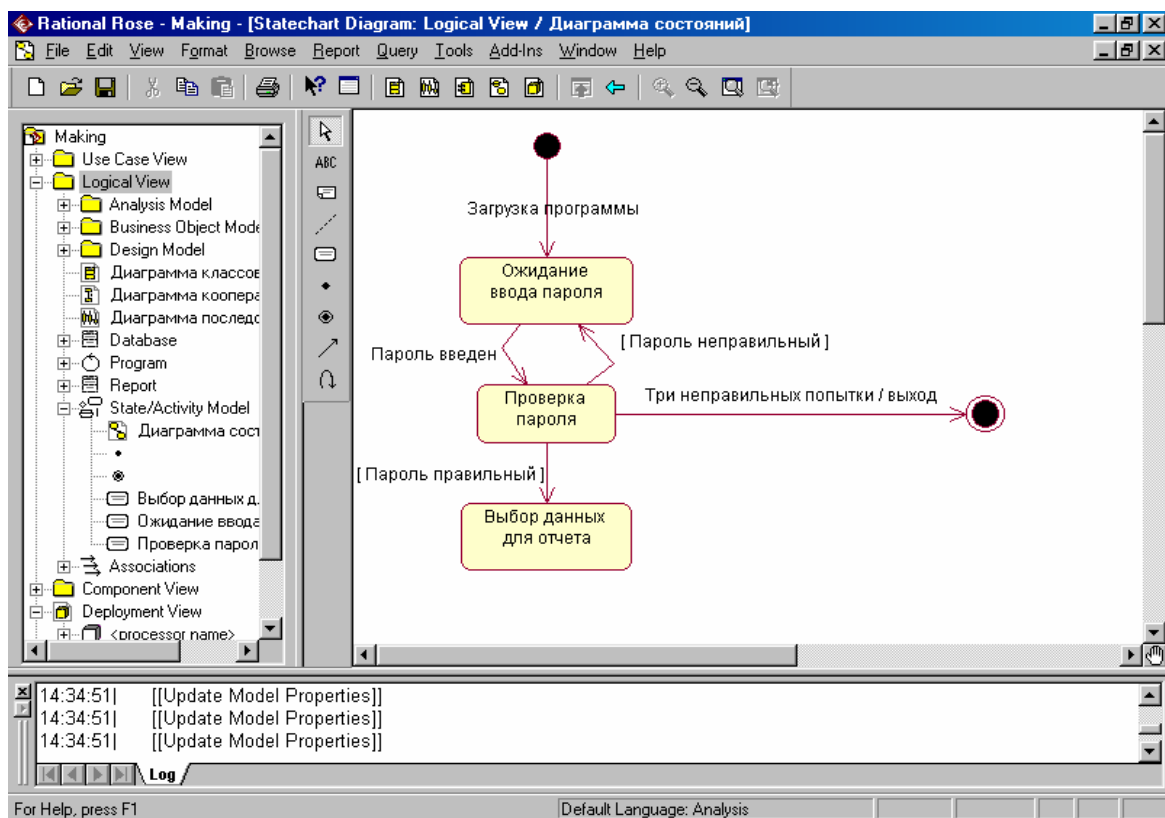


Рисунок 88 – Додавання нових станів і переходів

Залишилося додати чотири, що залишилися, стани і переходи між ними (рис. 89).

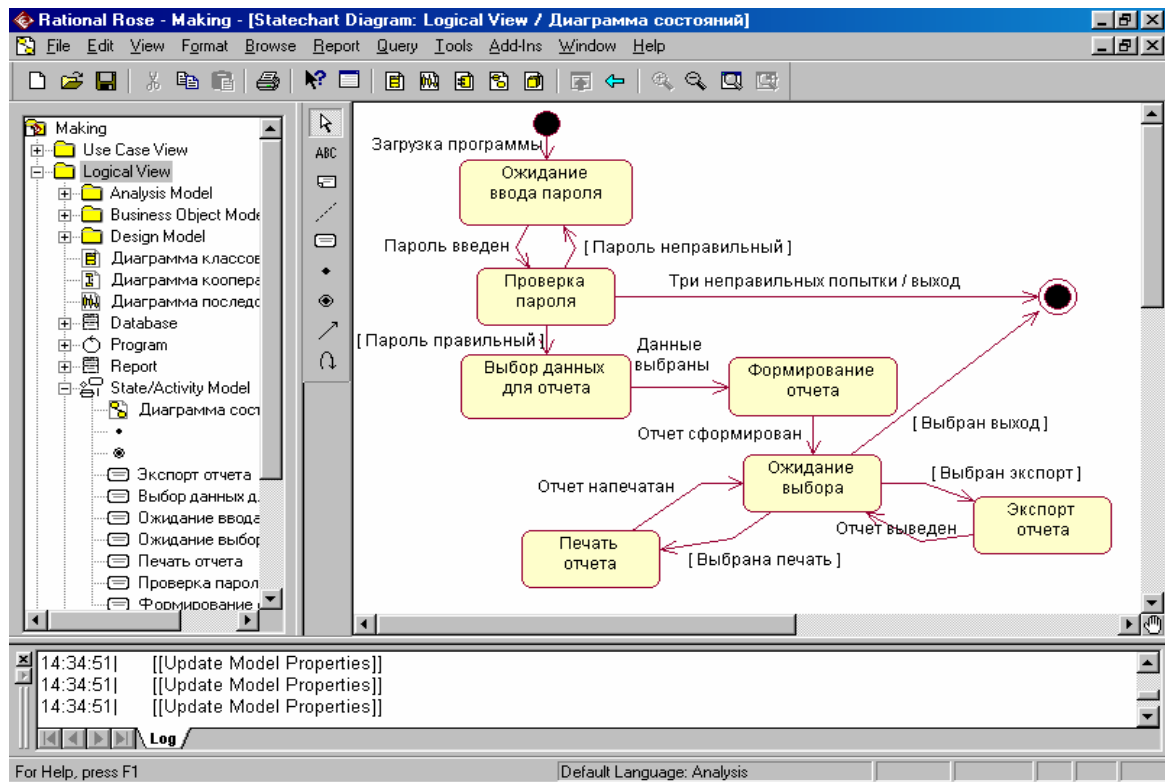


Рисунок 89 – Остаточный вид диаграмми станів

Діаграма діяльності створюється тим же пунктом меню «Browse / State Machine Diagram», що і діаграма стану, але вибирається інший тип («Diagram Type: Statechart»). У браузері проектів нова діаграма також розміститься у гілці «Logical View / State-Activity Model» (рис. 90).

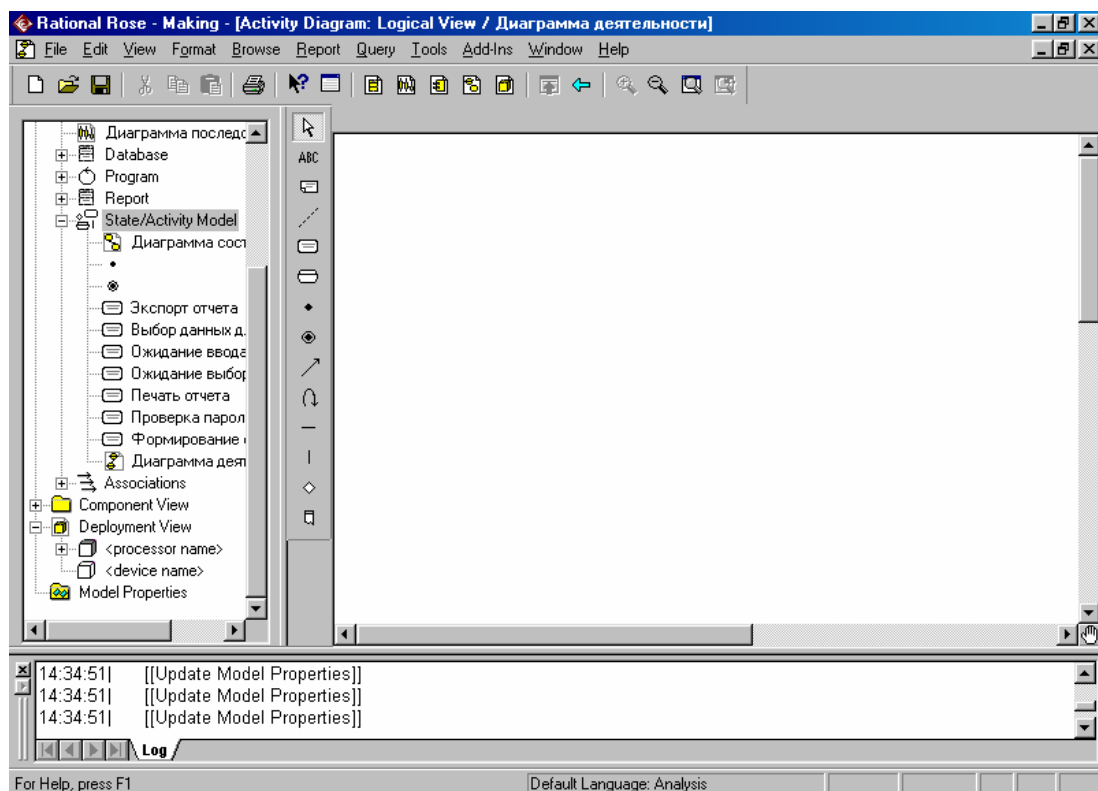


Рисунок 90 – Початковий вид діаграми діяльності

Початковий стан у моделі може бути тільки один, тому спроба взяти його із спеціальної панелі інструментів ні до чого не приведе: початковий стан потрібно знайти у вікні браузера проекту (чорне коло відразу після назви діаграми станів) і «перетягнути» у вікно діаграми.

Логіка побудови діаграми діяльності практично повністю повторює логіку побудови діаграми станів (точно кажучи, у даному прикладі цю діаграму можна було і опустити). Перша дія – введення пароля – вибираємо на спеціальній панелі інструментів (пункт Activity), розміщуємо у вікні діаграми і привласнюємо ім'я (рис. 91).

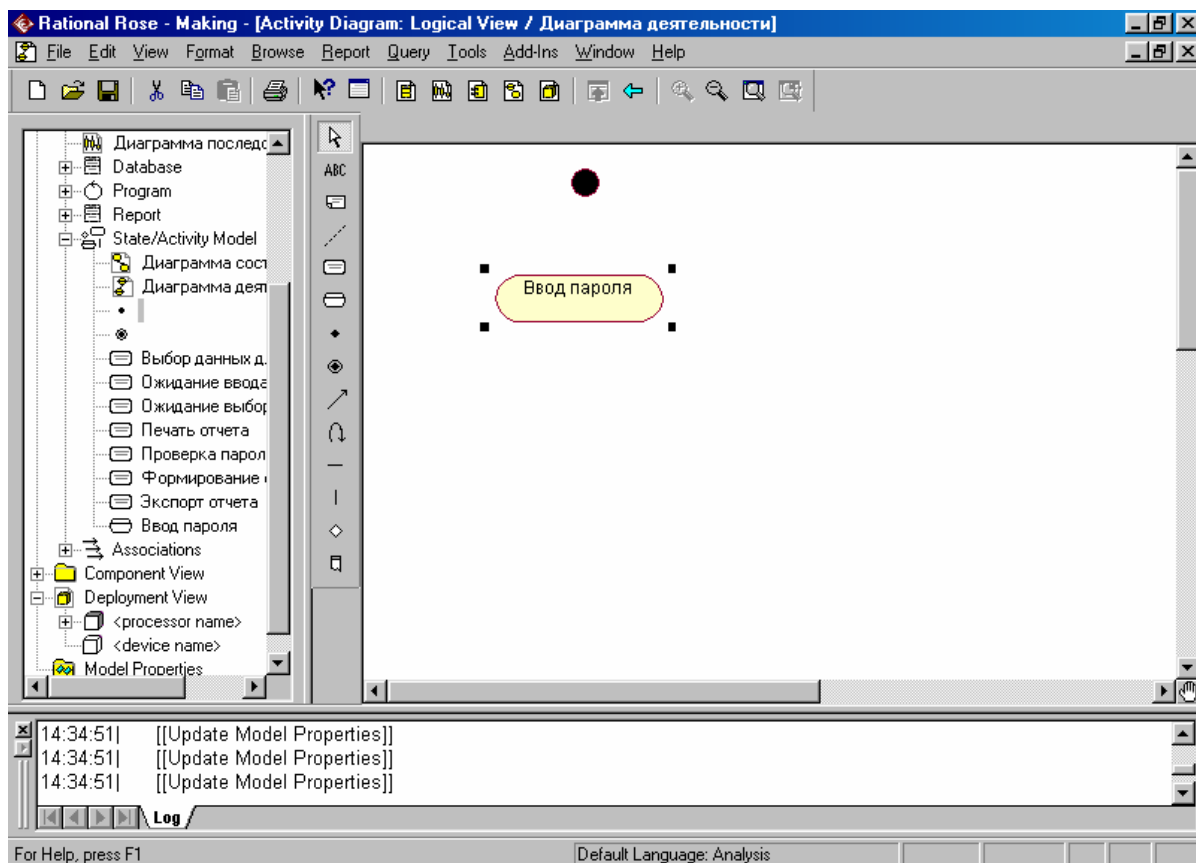


Рисунок 91 – Додавання нової дії

Сполучаємо початковий стан з першою дією; імені події на переході указувати не потрібно. Далі на діаграмі починається галуження: додається знак «Decision», з якого можливі два переходи із сторожовими умовами «Пароль правильний» і «Пароль неправильний». У другому випадку відразу можна додати перехід до дії «Вихід з програми» і закінчення роботи (рис. 92).

У разі правильності пароля додамо дії «Витягання даних для звіту», «Створення звіту» і «Вивід звіту» (розгалуження для розділення випадків

«Друк звіту» і «Експорт звіту» опустимо). Остаточний вид діаграми діяльності поданий на рисунку 93.

Діаграма компонентів створюється пунктом меню «Browse / Component Diagram», у правому вікні з'явиться заготовка нової діаграми (рис. 94). Розташований там за умовчанням пакет «Implementation model» можна видалити (вікно браузера проекту, контекстне меню, пункт «Delete»).

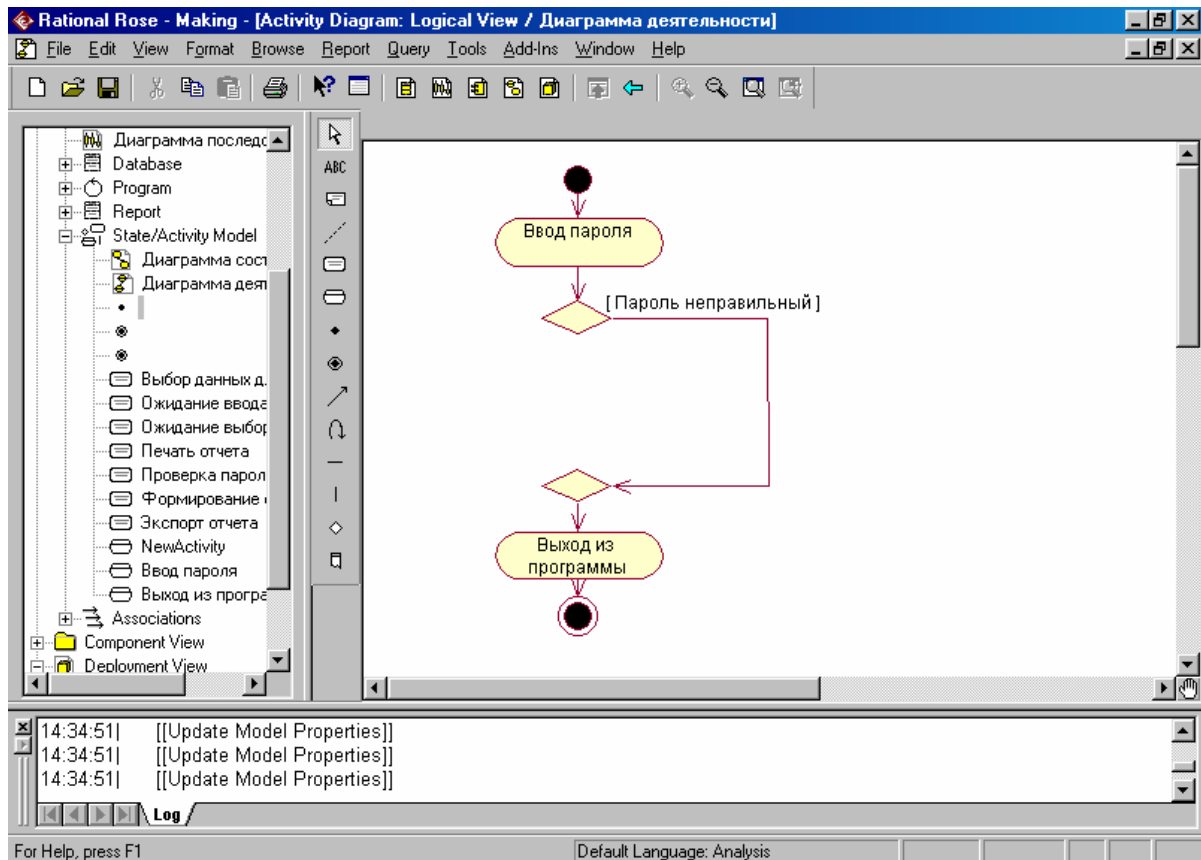


Рисунок 92 – Додавання розгалуження і останніх дій

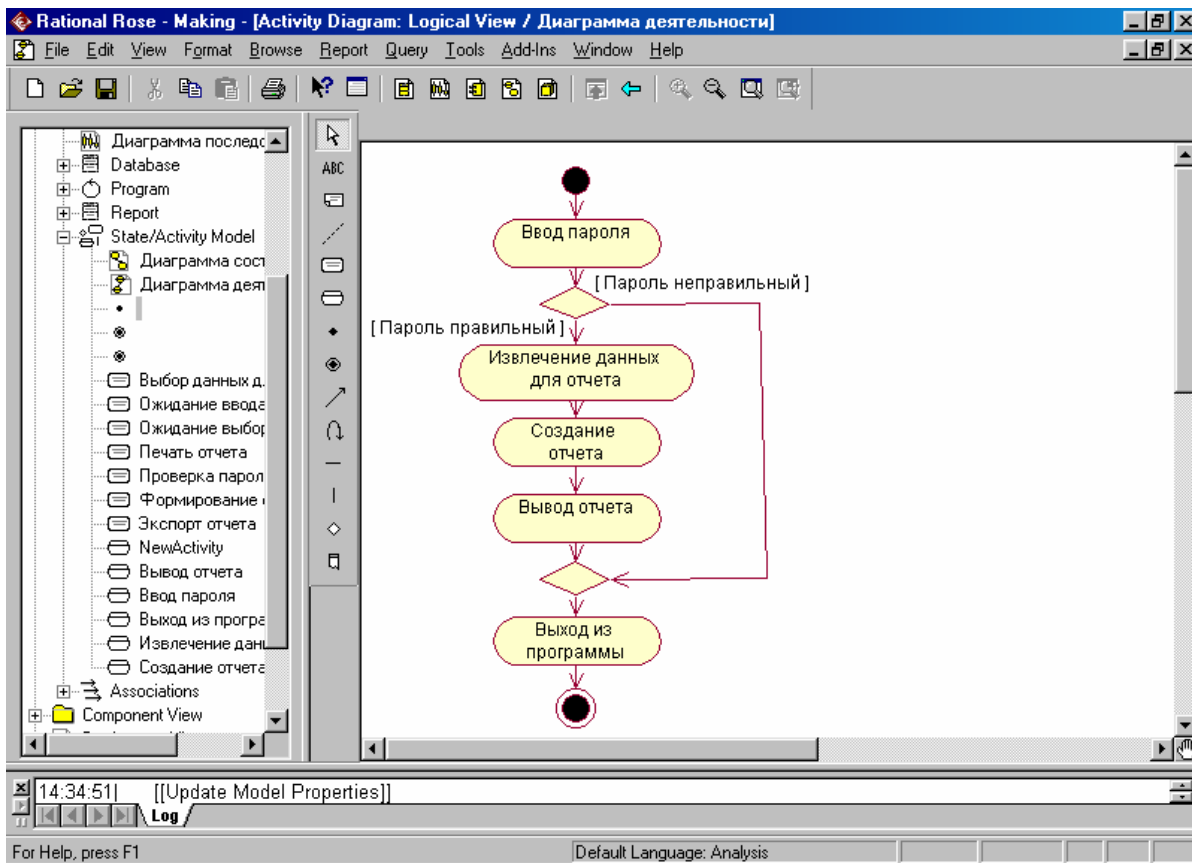


Рисунок 93 – Остаточный вид диаграммы діяльності

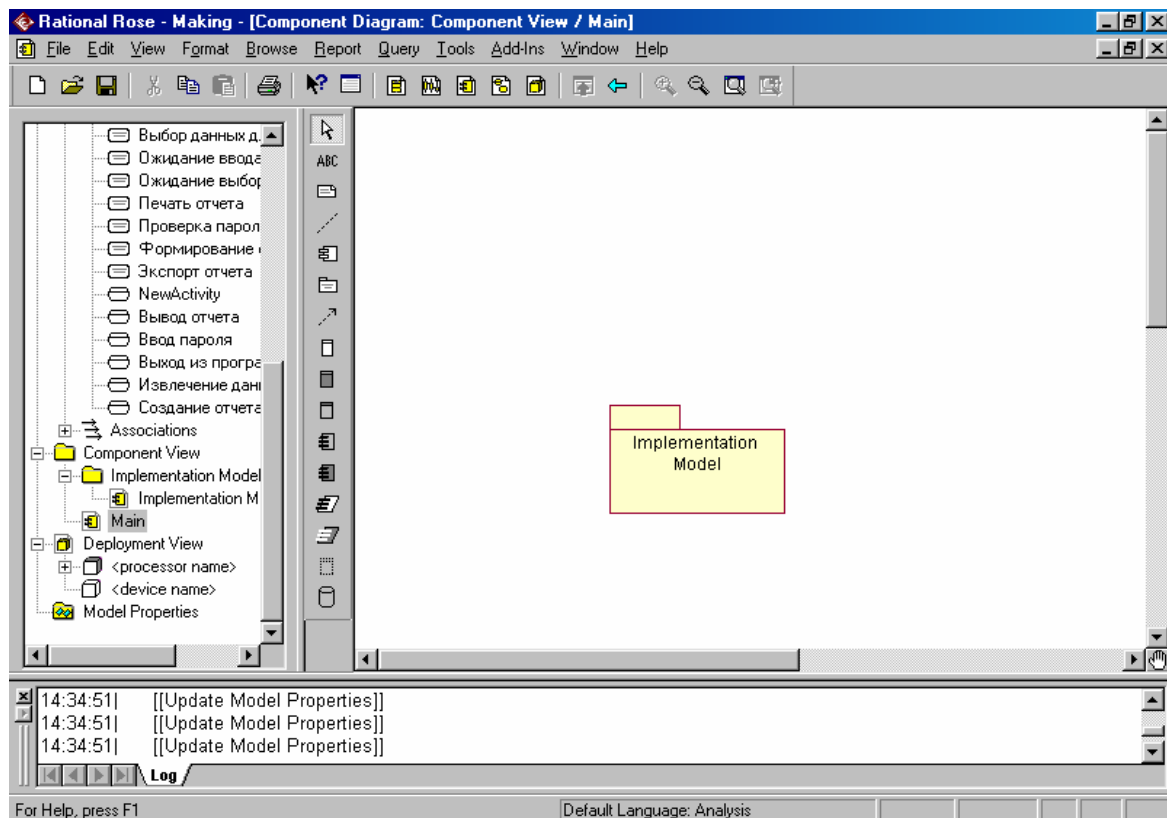


Рисунок 94 – Початковий вид діаграми компонентів

Помістимо на діаграму новий компонент, назвемо його «Головна програма» (рис. 95).

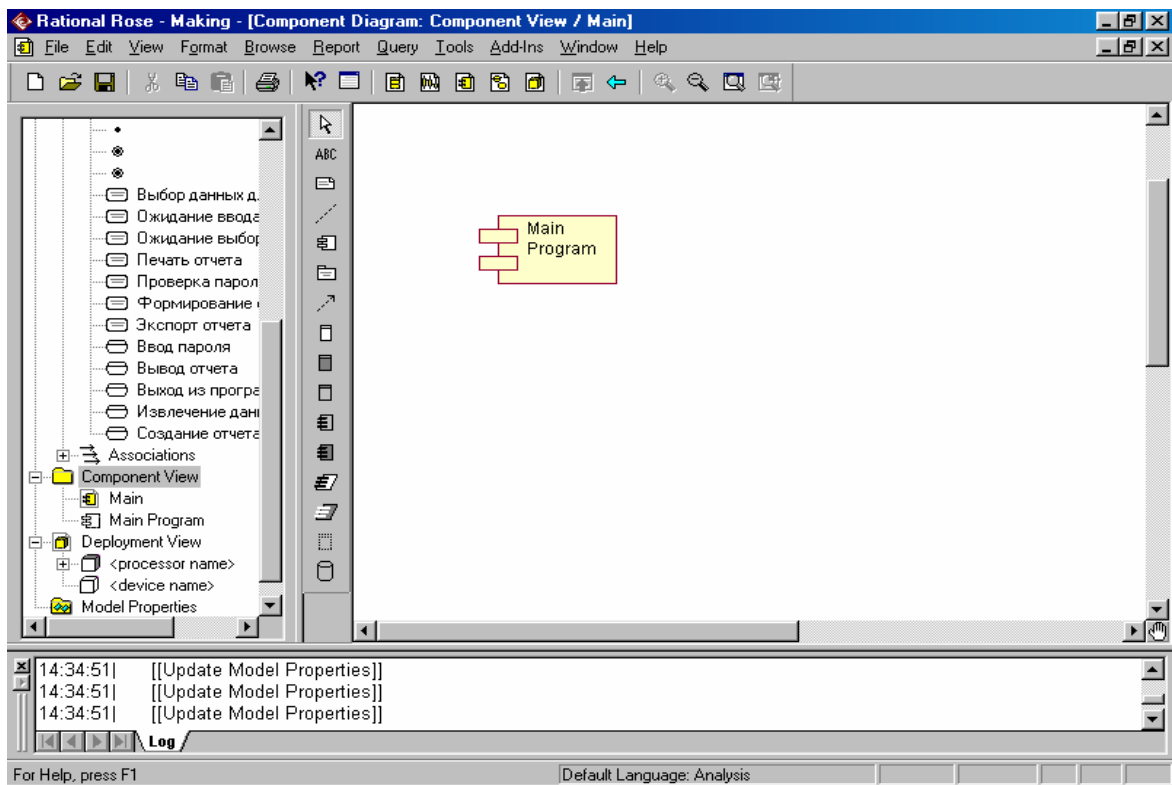


Рисунок 95 – Додавання нового компонента

Тепер можна змінити тип нового компонента: у вікні специфікації виберемо стереотип «EXE» (рис. 96).

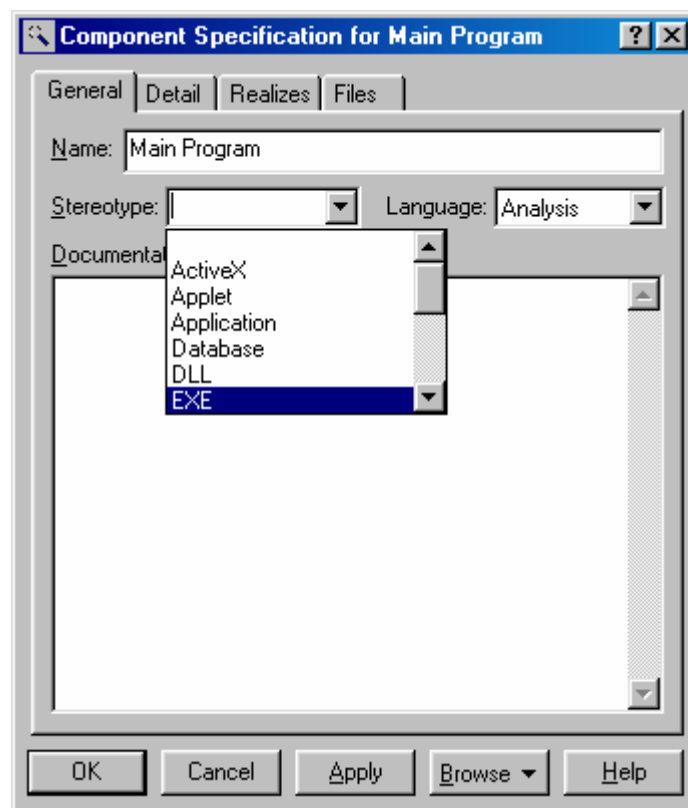


Рисунок 96 – Вікно специфікації компонента

Щоб результат змін був явно видний на діаграмі, виберемо в контекстному меню компонента пункт «Stereotype Display / Decoration» (рис. 97).

До виконуваного файлу віднесемо два файли: з одного боку, це файл Delphi-проекту (DPR), з іншою – база даних. Файла проекту можна привласнити стереотип «Main Program», змінивши його зображення на «Decoration», а базу даних із стереотипом «Database» залишимо в незмінному вигляді. Додамо зв'язку-залежності (dependency) між виконуваним файлом, файлом проекту і базою даних (рис. 98).

Файл проекту буде пов'язаний з трьома файлами – модулями початкових текстів програми (наявність файлів форм і тому подібне мається на увазі) – Unit1.pas, Unit2.pas і Unit3.pas (рис. 99). Стереотип цих файлів можна вибрати довільно – хай буде Subprogram.

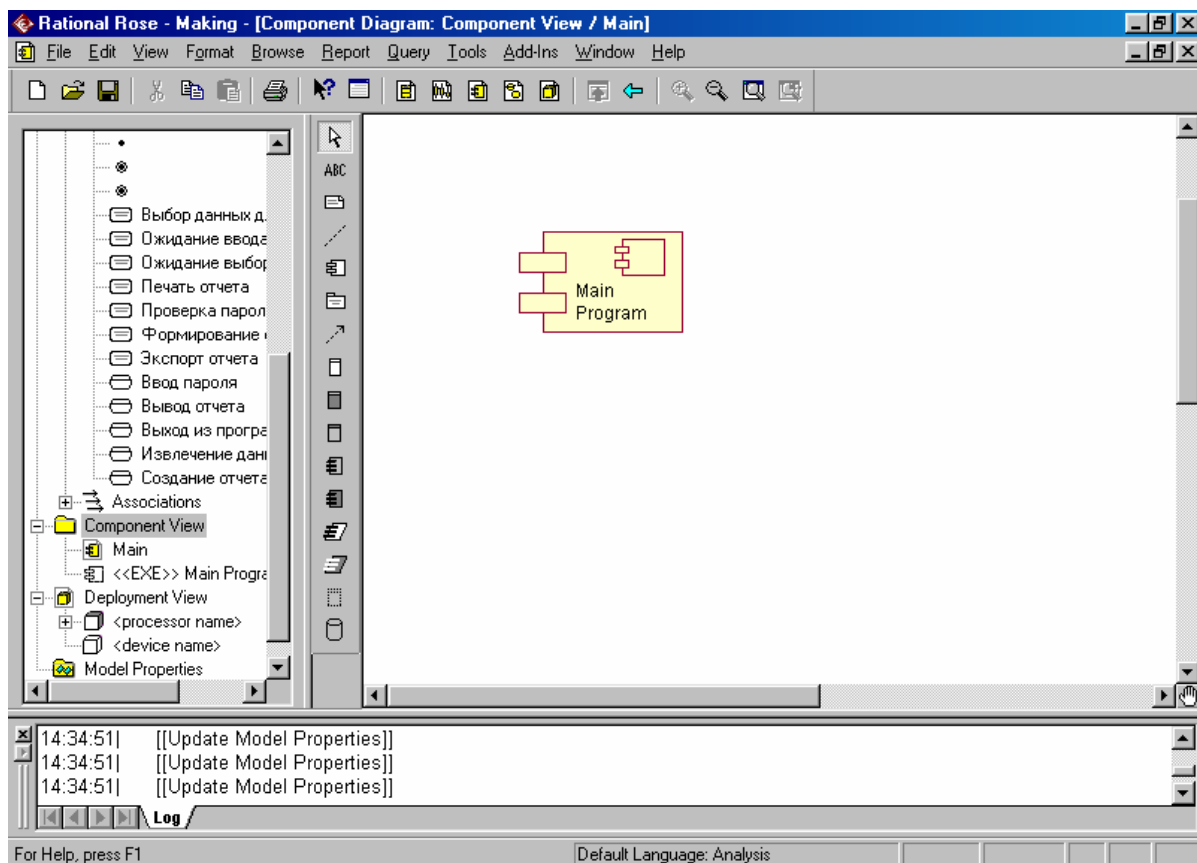


Рисунок 97 – Компонент із стереотипом «виконуваний файл»

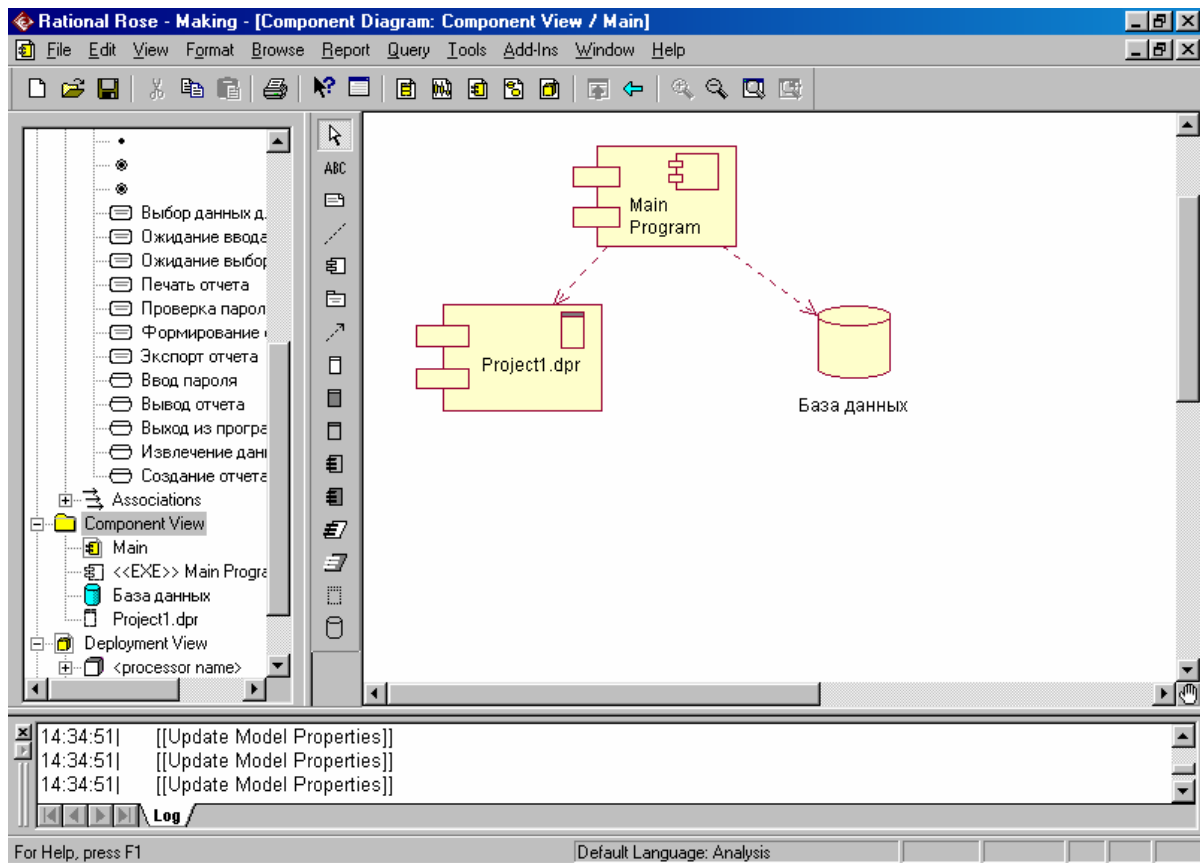


Рисунок 98 – Нові компоненти і зв'язки між ними

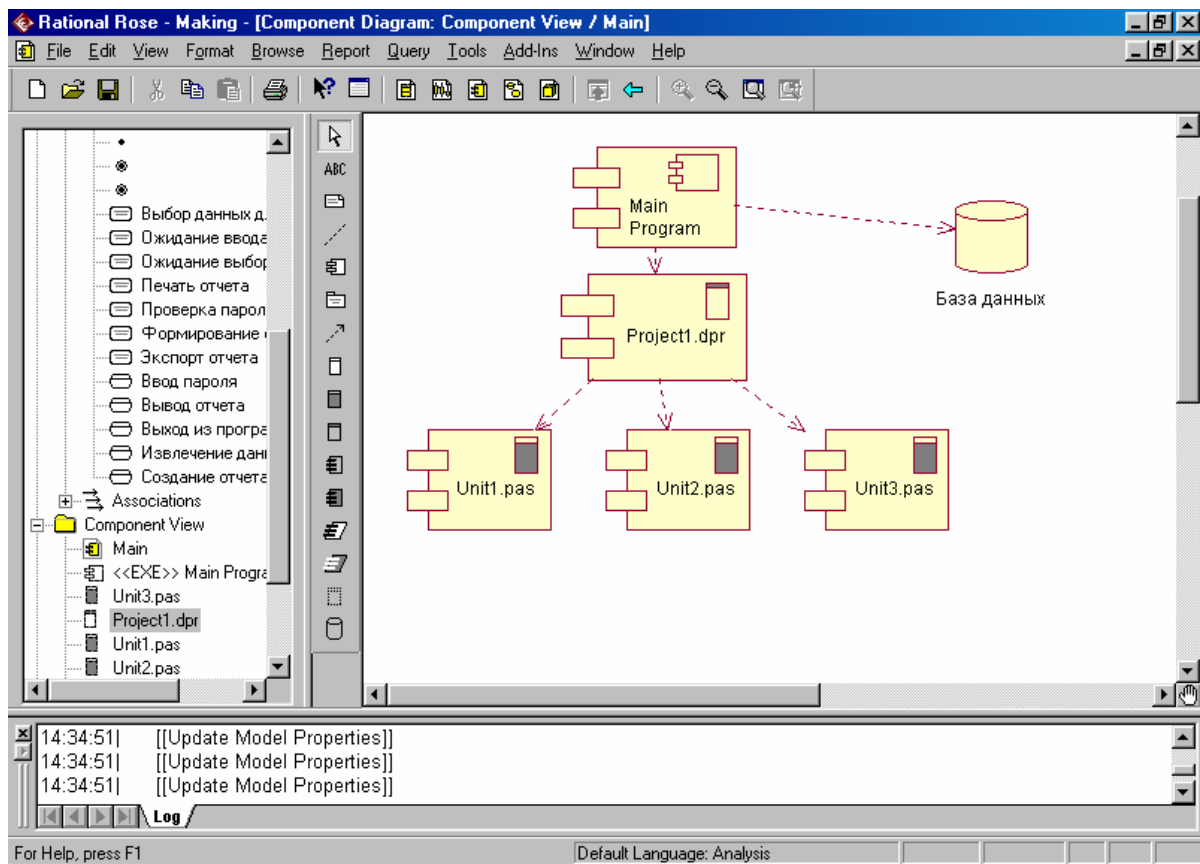


Рисунок 99 – Додавання компонентів з початковим текстом програми

Оскільки незрозуміло, що саме розташовується у кожному з модулів, доцільно додати примітки, в яких уточнити цей момент. Остаточний вид діаграми компонентів поданий на рисунку 100.

Діаграма розгортання створюється пунктом меню «Browse / Deployment Diagram», у правому вікні з'явиться заготовка нової діаграми (рис. 101). Розташований там за умовчанням коментар можна видалити, а два вузли, що знаходяться, – залишити.

Припустимо, що наша система може працювати у клієнт-серверному режимі. З боку серверу будуть розташовані ресурсоємний вузол – комп'ютер з серверною частиною програми, що виконує авторизацію користувача, і бази даних. З боку клієнта розташується будь-який комп'ютер, що має доступ до мережі. У вікні специфікації ресурсоємного вузла («процесора») вкажемо його власне ім'я (наприклад, Сервер) і стереотип – «processor», у вікні специфікації вузла пристрою – ім'я «База даних і відповідний стереотип» (рис. 102).

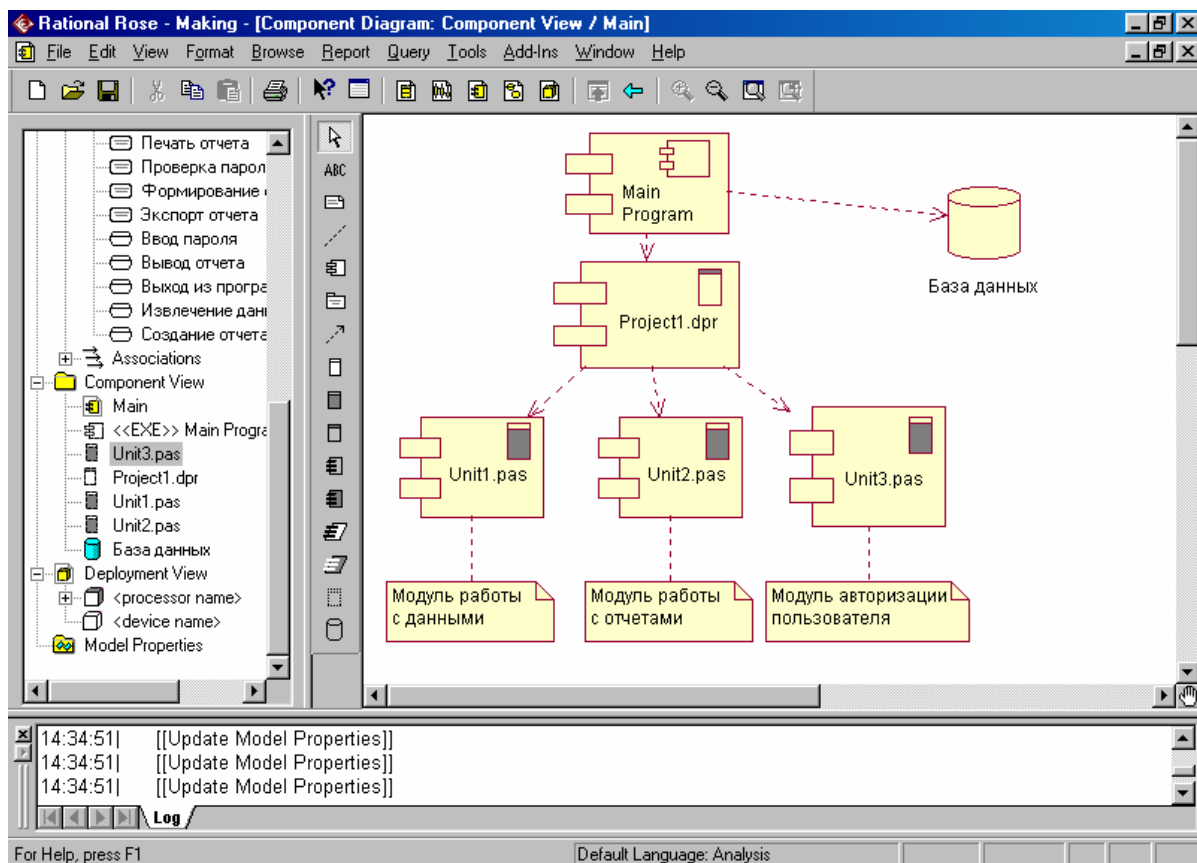


Рисунок 100 – Остаточний вид діаграми компонентів

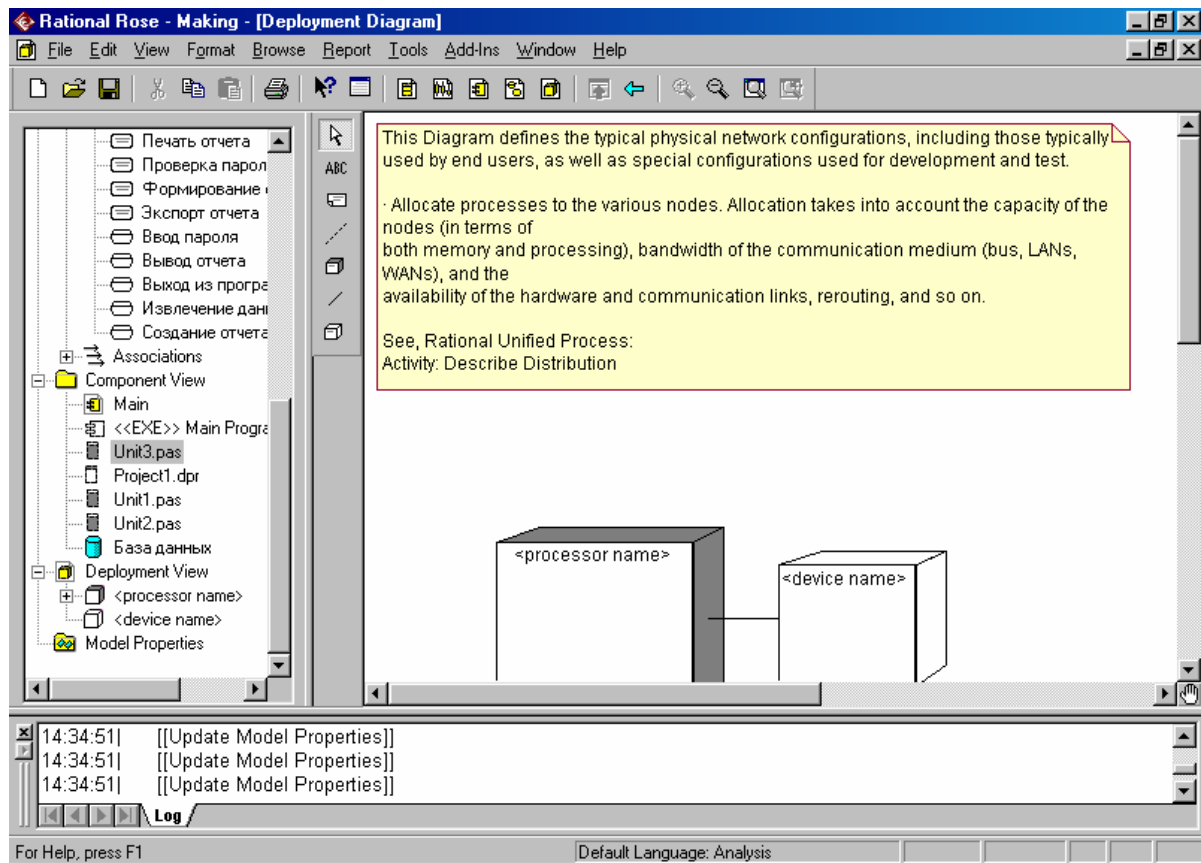


Рисунок 101 – Початковий вид діаграми розгортання

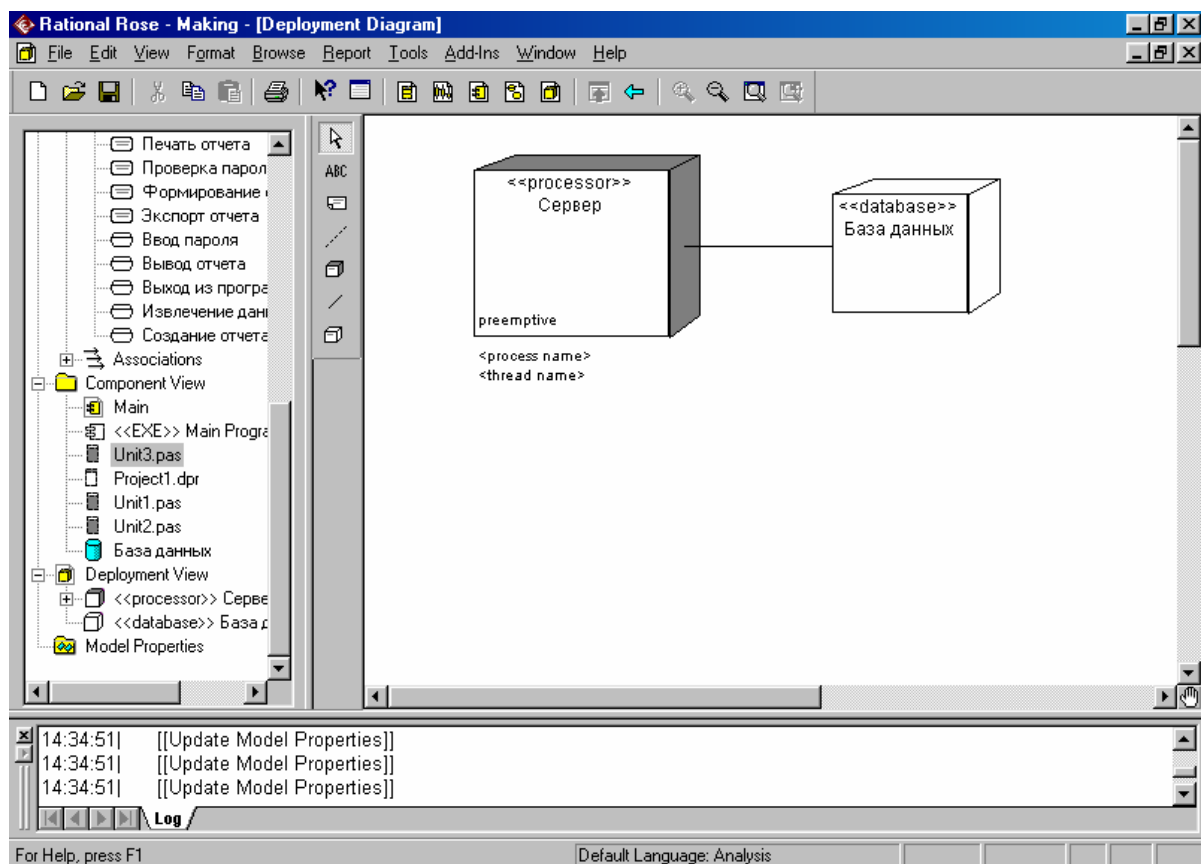


Рисунок 102 – Два вузли на діаграмі розгортання

Непотрібні в нашому випадку рядки з іменами процесів або ниток управління можна прибрати з екрана за допомогою контекстного меню (відміна пунктів «Show Processes» і «Show Scheduling»).

Новий ресурсоємний вузол на діаграмі – клієнт. Зобразимо його у вигляді анонімного екземпляра класу «Клієнт» (рис. 103), причому стереотип «processor» можна не указувати, його ресурсоємність очевидна на вигляд.

Мережа, що є, по суті, проміжним пристроєм між серверною і клієнтською частинами системи, зображена у вигляді звичного вузла (device) із стереотипом «Net» (рис. 104).

Останні дії – розміщення ліній зв'язку (connections). Припустимо при цьому, що доступ до бази даних може здійснюватися як за допомогою серверу, так і безпосередньо від клієнта. Остаточний вид діаграми розгортання поданий на рисунку 105.

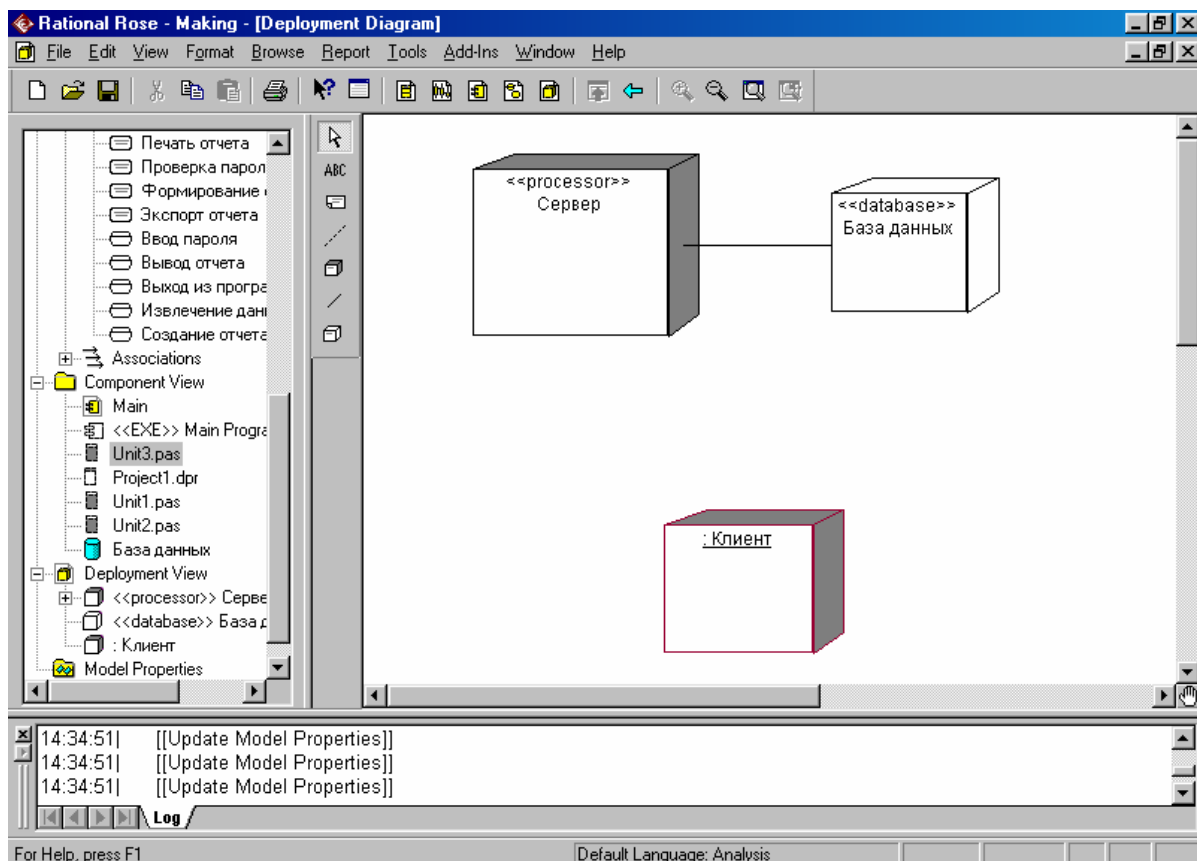


Рисунок 103 – Новий вузол – анонімний екземпляр класу

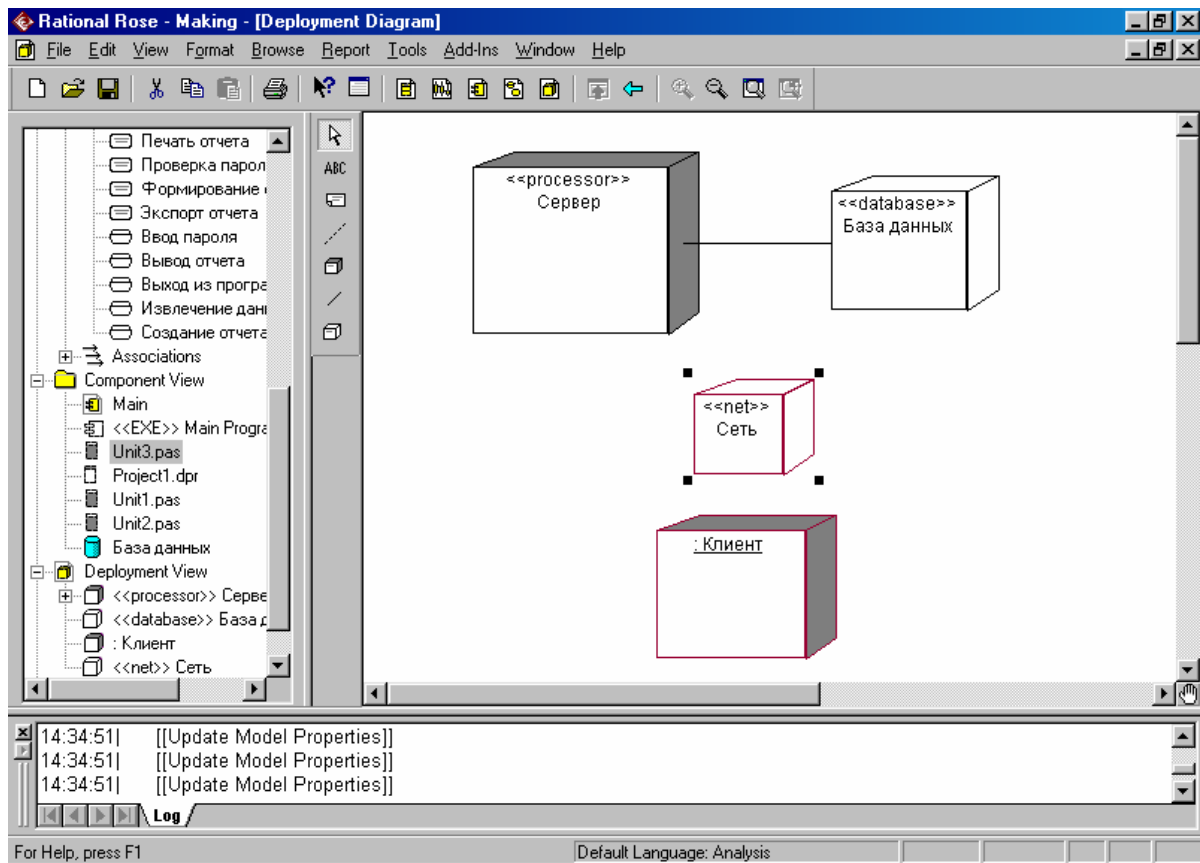


Рисунок 104 – Новый пристрій «Мережа»

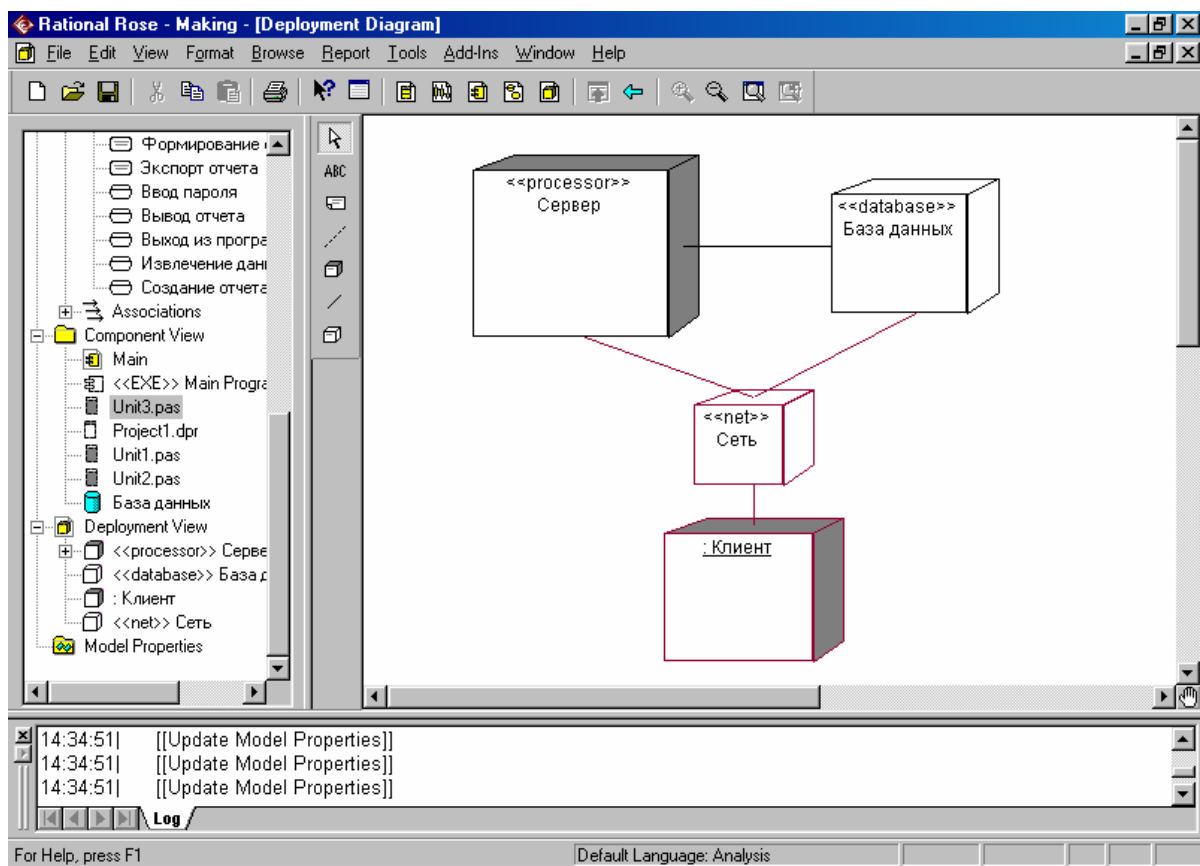


Рисунок 105 – Остаточний вид діаграми розгортання

3.3 Генерація коду спроектованої моделі у середовищі програмування

Створену у середовищі IBM Rational Rose модель можна експортувати в одне з середовищ програмування, підтримуваних даною версією програми. Як правило, таким шляхом створюється каркас майбутньої програмної системи.

Перед генерацією програмного коду потрібно виділити всі компоненти діаграми класів і в головному меню вказати мову програмування («Tools / Options / Notation / Default Language»). Потім у тому ж меню (пункт «Tools») вибрати цю мову і пункт «Code Generation». Результат зберігається на диску у папці «C:\ Program Files\ Rational\ Rose\ <выбранный язык>\ source\».

У нашому випадку при генерації коду на мові C++ в теці «C:\Program Files\Rational\Rose\c++\source\» буде створено 10 файлів, назви і зміст яких наведені нижче (чисельні коментарі, створені системою, видалені).

Admin.h

```
#ifndef Admin_h
#define Admin_h 1
// Program
#include "Program.h"
class Admin
{
public:
    /// Constructors (generated)
    Admin();
    Admin(const Admin &right);
    /// Destructor (generated)
    ~Admin();
    /// Assignment Operation (generated)
    Admin & operator=(const Admin &right);
    /// Equality Operations (generated)
    int operator==(const Admin &right) const;
```



```

    int operator!=(const Admin &right) const;
    /// Get and Set Operations for Associations (generated)
    /// Association: <unnamed>%44B3DEA20104
    /// Role: Admin::<the_Program>%44B3DEA202F8
    const Program * get_the_Program () const;
    void set_the_Program (Program * value);
    // Additional Public Declarations
    /// begin Admin%44B3A68200BE.public preserve=yes
    /// end Admin%44B3A68200BE.public
protected:
    // Additional Protected Declarations
private:
    // Additional Private Declarations
private: /// implementation
    // Data Members for Associations
public: Program { -> RHN}
    Program *the_Program;
    /// end Admin::<the_Program>%44B3DEA202F8.role
    // Additional Implementation Declarations
};
// Class Admin
/// Get and Set Operations for Associations (inline)
inline const Program * Admin::get_the_Program () const
{
    return the_Program;
}
inline void Admin::set_the_Program (Program * value)
{
    the_Program = value;
}
#endif

```

Admin.cpp

```
// Admin
#include "Admin.h"
// Class Admin
Admin::Admin() { }
Admin::Admin(const Admin &right) { }
Admin::~~Admin() { }
Admin & Admin::operator=(const Admin &right) { }
int Admin::operator==(const Admin &right) const { }
int Admin::operator!=(const Admin &right) const { }
// Additional Declarations
```

Database.h

```
#ifndef Database_h
#define Database_h 1
// Program
#include "Program.h"
class Database
{
public:
    /// Constructors (generated)
    Database();
    Database(const Database &right);
    /// Destructor (generated)
    ~Database();
    /// Assignment Operation (generated)
    Database & operator=(const Database &right);
    /// Equality Operations (generated)
    int operator==(const Database &right) const;
    int operator!=(const Database &right) const;
    /// Other Operations (specified)
    void inputData ();
    /// Operation: modifyData%44B3DE2701C2
```

```

void modifyData ();
///  

void getData ();
///  

const Program * get_the_Program () const;
void set_the_Program (Program * value);
// Additional Public Declarations
protected:
// Additional Protected Declarations
private:
///  

const void get_data () const;
void set_data (void value);
// Additional Private Declarations
///  

private: ///  

// Data Members for Class Attributes
void data;
// Data Members for Associations
public: Program { -> RHN}
Program *the_Program;
// Additional Implementation Declarations
};
// Class Database
///  

inline const void Database::get_data () const
{
return data;
}
inline void Database::set_data (void value)
{
data = value;
}

```

```

}
///  

inline const Program * Database::get_the_Program () const
{
    return the_Program;
}
inline void Database::set_the_Program (Program * value)
{
    the_Program = value;
}
#endif

```

Database.cpp

```

// Database
#include "Database.h"
// Class Database
Database::Database() { }
Database::Database(const Database &right) { }
Database::~Database() { }
Database & Database::operator=(const Database &right) { }
int Database::operator==(const Database &right) const { }
int Database::operator!=(const Database &right) const { }
///  

// Other Operations (implementation)
void Database::inputData () { }
void Database::modifyData () { }
void Database::getData () { }
// Additional Declarations

```

Program.h

```

#ifndef Program_h
#define Program_h 1
// User
#include "User.h"

```

```

// Report
#include "Report.h"
// Database
#include "Database.h"
// Admin
#include "Admin.h"
class Program
{
public:
    /// Constructors (generated)
    Program();
    Program(const Program &right);
    /// Destructor (generated)
    ~Program();
    /// Assignment Operation (generated)
    Program & operator=(const Program &right);
    /// Equality Operations (generated)
    int operator==(const Program &right) const;
    int operator!=(const Program &right) const;
    /// Other Operations (specified)
    Boolean authorization ();
    /// Get and Set Operations for Associations (generated)
    const Admin * get_the_Admin () const;
    void set_the_Admin (Admin * value);
    const User * get_the_User () const;
    void set_the_User (User * value);
    const Database * get_the_Database () const;
    void set_the_Database (Database * value);
    const Report * get_the_Report () const;
    void set_the_Report (Report * value);
    // Additional Public Declarations
protected:
    // Additional Protected Declarations

```

```

private:
    // Additional Private Declarations
private: /// implementation
    // Data Members for Associations

    Admin *the_Admin;
    /// Association: <unnamed>%44B3DEA40046
lic: User { -> RHN}
    User *the_User;
public: Database { -> RHN}
    Database *the_Database;
public: Report { -> RHN}
    Report *the_Report;
// Additional Implementation Declarations
};
// Class Program
/// Get and Set Operations for Associations (inline)
inline const Admin * Program::get_the_Admin () const
{
    return the_Admin;
}
inline void Program::set_the_Admin (Admin * value)
{
    the_Admin = value;
}
inline const User * Program::get_the_User () const
{
    return the_User;
}
inline void Program::set_the_User (User * value)
{
    the_User = value;
}

```

```

inline const Database * Program::get_the_Database () const
{
    return the_Database;
}
inline void Program::set_the_Database (Database * value)
{
    the_Database = value;
}
inline const Report * Program::get_the_Report () const
{
    return the_Report;
}
inline void Program::set_the_Report (Report * value)
{
    the_Report = value;
}
#endif

```

Program.cpp

```

// Program
#include "Program.h"
// Class Program
Program::Program() { }
Program::Program(const Program &right) { }
Program::~~Program() { }
Program & Program::operator=(const Program &right) { }
int Program::operator==(const Program &right) const { }
int Program::operator!=(const Program &right) const { }
///## Other Operations (implementation)
Boolean Program::autorization () { }
// Additional Declarations

```

Report.h

```
#ifndef Report_h
#define Report_h 1
// Program
#include "Program.h"
class Report
{
public:
    /// Constructors (generated)
    Report();
    Report(const Report &right);
    /// Destructor (generated)
    ~Report();
    /// Assignment Operation (generated)
    Report & operator=(const Report &right);
    /// Equality Operations (generated)
    int operator==(const Report &right) const;
    int operator!=(const Report &right) const;
    /// Other Operations (specified)
    void forming ();
    void print ();
    void export ();
    /// Get and Set Operations for Associations (generated)
    const Program * get_the_Program () const;
    void set_the_Program (Program * value);
    // Additional Public Declarations
protected:
    // Additional Protected Declarations
private:
    /// Get and Set Operations for Class Attributes (generated)
    const void get_data () const;
    void set_data (void value);
    // Additional Private Declarations
```



```

private: /// implementation
    // Data Members for Class Attributes
    void data;
    // Data Members for Associations
    Program *the_Program;
};
// Class Report
///## Get and Set Operations for Class Attributes (inline)
inline const void Report::get_data () const
{
    return data;
}
inline void Report::set_data (void value)
{
    data = value;
}
///## Get and Set Operations for Associations (inline)
inline const Program * Report::get_the_Program () const
{
    return the_Program;
}
inline void Report::set_the_Program (Program * value)
{
    the_Program = value;
}
#endif

```

Report.cpp

```

// Report
#include "Report.h"
// Class Report
Report::Report() { }
Report::Report(const Report &right) { }

```

```

Report::~~Report() { }
Report & Report::operator=(const Report &right) { }
int Report::operator==(const Report &right) const { }
int Report::operator!=(const Report &right) const { }
///  

void Report::forming () { }
void Report::print () { }
void Report::export () { }
// Additional Declarations

```

User.h

```

#ifndef User_h
#define User_h 1
// Program
#include "Program.h"
class User
{
public:
    ///  

    User();
    User(const User &right);
    ///  

    ~User();
    ///  

    User & operator=(const User &right);
    ///  

    int operator==(const User &right) const;
    int operator!=(const User &right) const;
    ///  

    const Program * get_the_Program () const;
    void set_the_Program (Program * value);
    // Additional Public Declarations
protected:
    // Additional Protected Declarations

```

```

private:
    // Additional Private Declarations
private: /// implementation
    // Data Members for Associations
    Program *the_Program;
    // Additional Implementation Declarations
};
// Class User
///## Get and Set Operations for Associations (inline)

inline const Program * User::get_the_Program () const
{
    return the_Program;
}
inline void User::set_the_Program (Program * value)
{
    the_Program = value;
}
#endif

```

User.cpp

```

// User
#include "User.h"
// Class User
User::User() { }
User::User(const User &right) { }
User::~~User() { }
User & User::operator=(const User &right) { }
int User::operator==(const User &right) const { }
int User::operator!=(const User &right) const { }
// Additional Declarations

```

4 ПРИКЛАДИ ПРОЕКТУВАННЯ ІНФОРМАЦІЙНИХ СИСТЕМ

У цьому розділі подані приклади спроектованих інформаційних систем для вирішення певних економічних і управлінських задач. Усі подані моделі розроблені студентами спеціальності «Економічна кібернетика» у рамках курсового і дипломного проектування під керівництвом або з участю автора посібника. Частина моделей створювалася за допомогою IBM Rational Rose, частина – з використанням інших CASE-засобів або просто «вручну» (у MS-Word); усі моделі наводяться тут в «первозданному» варіанті в цілях ілюстрації різноманіття зображення UML-моделей.

Кожен підрозділ, крім власне моделі на мові UML, містить короткий опис наочної області і проблем та є, по суті, науковою статтею.

4.1 Інформаційна система для функціонування кадрового агентства

За останній час кадровий бізнес у нашій країні перетворився на сферу економіки, що бурхливо розвивається. І якщо раніше всі функції кадрового менеджменту покладалися на відповідні служби усередині підприємств, то зараз широкий спектр послуг з підбору кадрів надають спеціалізовані кадрові агентства. З урахуванням специфіки роботи подібних організацій, де циркулюють великі об'єми інформації, організація роботи з даними тут повинна бути реалізована найефективніше і в найзручнішому для користувачів варіанті.

Аналіз інформаційного забезпечення кадрових агентств показав, що в більшості випадків використовуються стандартні додатки для обробки баз даних, дозволяючи здійснювати введення, зберігання і модифікацію даних, а також пошук у базі даних. У той же час різноманіття всіляких вимог приводить до рішення про необхідність створення спеціалізованої інформаційної системи для функціонування кадрового агентства.

За наслідками аналізу наочної області, що стосується конкретного підприємства (кадрового агентства відділення Краматорська Донецької торговельно-промислової палати), була поставлена задача реалізації наступ-

них оптимізаційних рішень.

По-перше, удосконалення задачі пошуку (він повинен бути двонаправленим і багатопараметричним):

- пошук варіантів, що задовольняють вимогам за вказаною вакансією у таблиці претендентів; умова пошуку може включати обов'язкові вимоги (стать, вік, спеціальність, рівень кваліфікації, рівень освіти, досвід роботи, наявність власного авто, телефону і т.д., причому деякі з цих полів взагалі можуть не включатися до пошуку, як неважливі) і необов'язкові (регіон, місто роботи, заробітна платня і т.п.);

- пошук можливих варіантів працевлаштування для клієнта-претендента у таблиці наявних вакансій (умовою пошуку є дані про конкретного клієнта-претендента, які заносяться із заповнюваних анкет, тобто рівень освіти, спеціальність, кваліфікація, досвід роботи, бажана посада, особисті дані і т.д.);

По-друге, додавання модуля формування статистики:

- облік статистики виконаних замовлень. Після відбору відповідних варіантів до працедавця прямують претенденти на посаду, і до таблиці обліку статистики заносяться дані про те, прийнятий претендент чи ні, тобто закрита вакансія чи ні. Ведення статистики дає можливість складання підсумкової звітності про конкретного клієнта-працедавця (які претенденти були запропоновані і які прийняті) і клієнта-претендента (які вакансії були запропоновані і які прийняті);

- складання звітності за період за задоволеними і незадоволеними заявками з визначенням тенденції і аналізом діяльності.

Застосування уніфікованої мови візуального моделювання UML дозволило усебічно розглянути і подати систему в послідовності від найзагальнішої і абстрактнішої концептуальної моделі до логічної, а потім і до фізичної моделі.

Логічні аспекти статичного подання системи були зображені за допомогою діаграми класів (рис. 106).

Особливості реалізації операцій класів були враховані і розглянуті за допомогою діаграми діяльності (рис. 107).

Функціональні характеристики системи були подані у діаграмі варіантів використання (рис. 108).

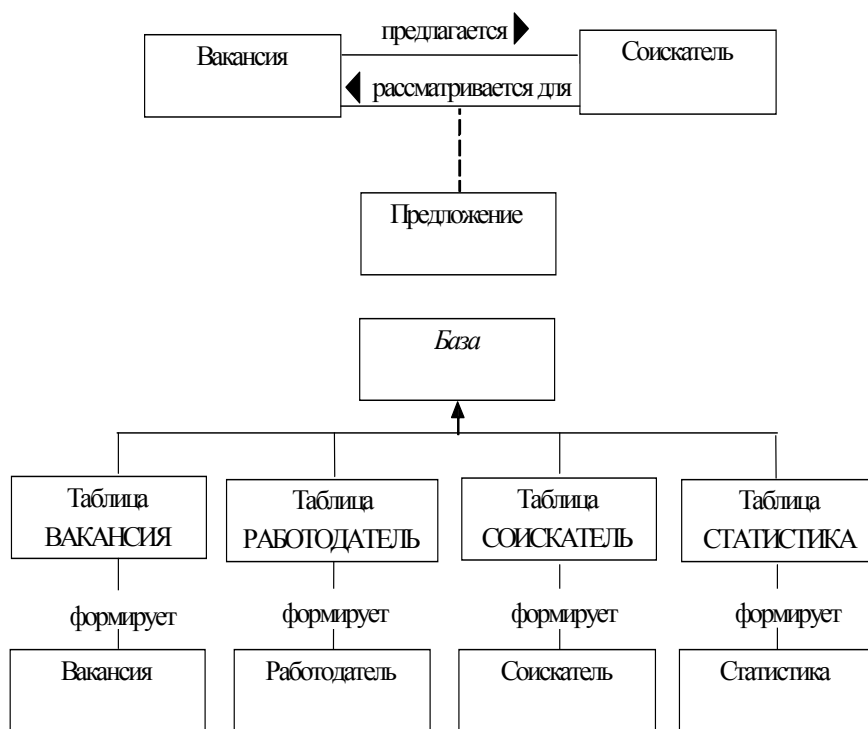
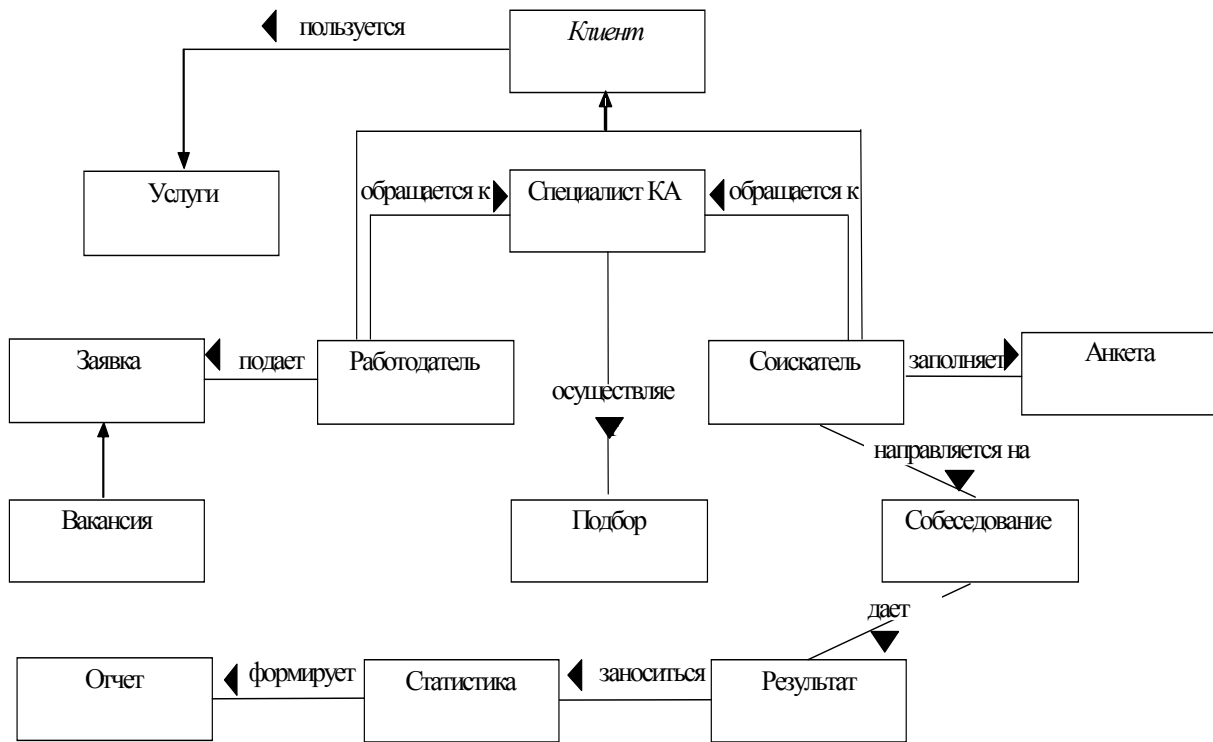


Рисунок 106 – Деталізована діаграма класів

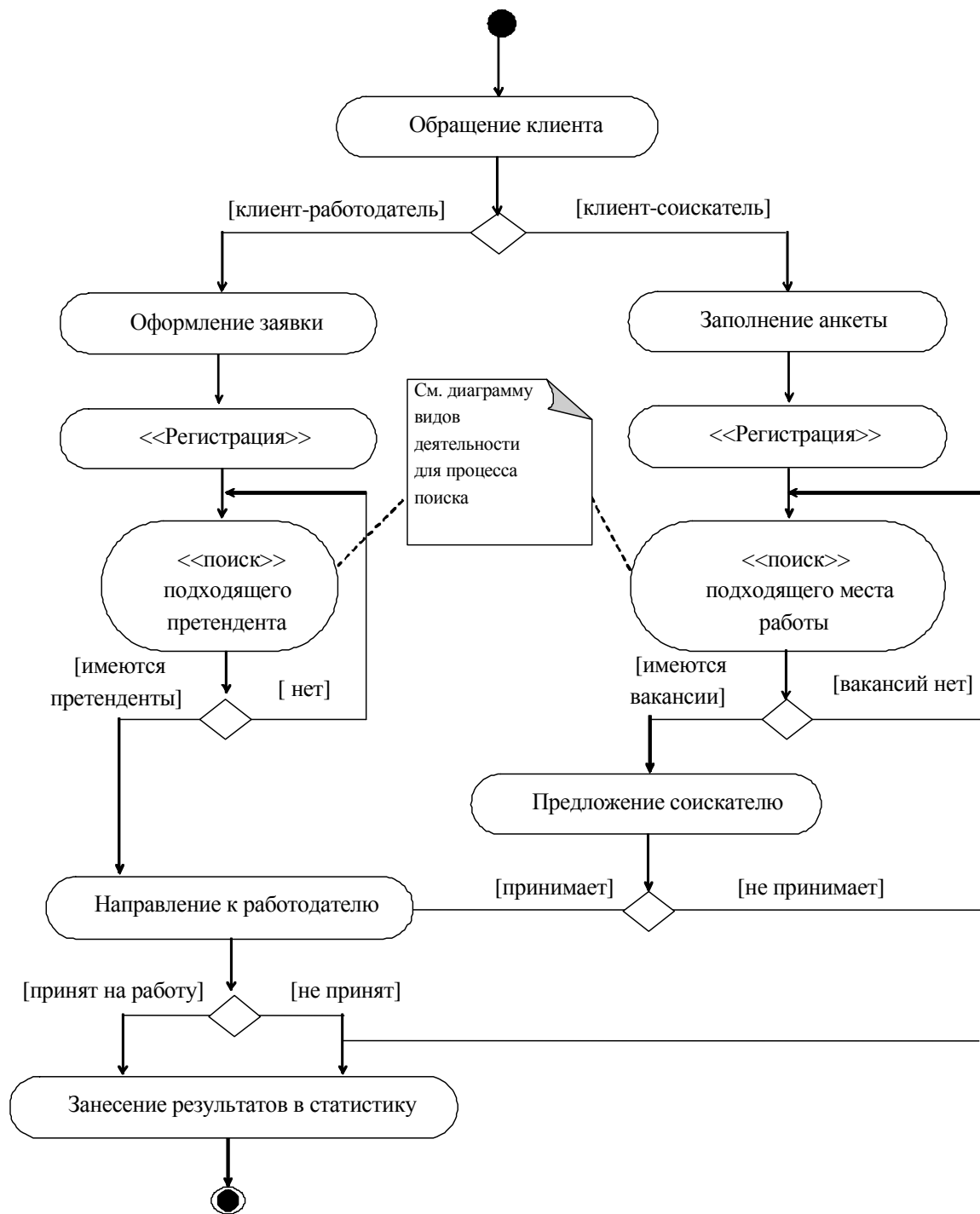


Рисунок 107 – Діаграма діяльності для процесу роботи з клієнтами

Взаємодія елементів системи, що розглядається в інформаційному аспекті їх комунікації, була розглянута за допомогою відповідних діаграм взаємодії: діаграми послідовностей (тимчасові аспекти взаємодії – рис. 109) і діаграми кооперації (структурні особливості взаємодії – рис. 110).

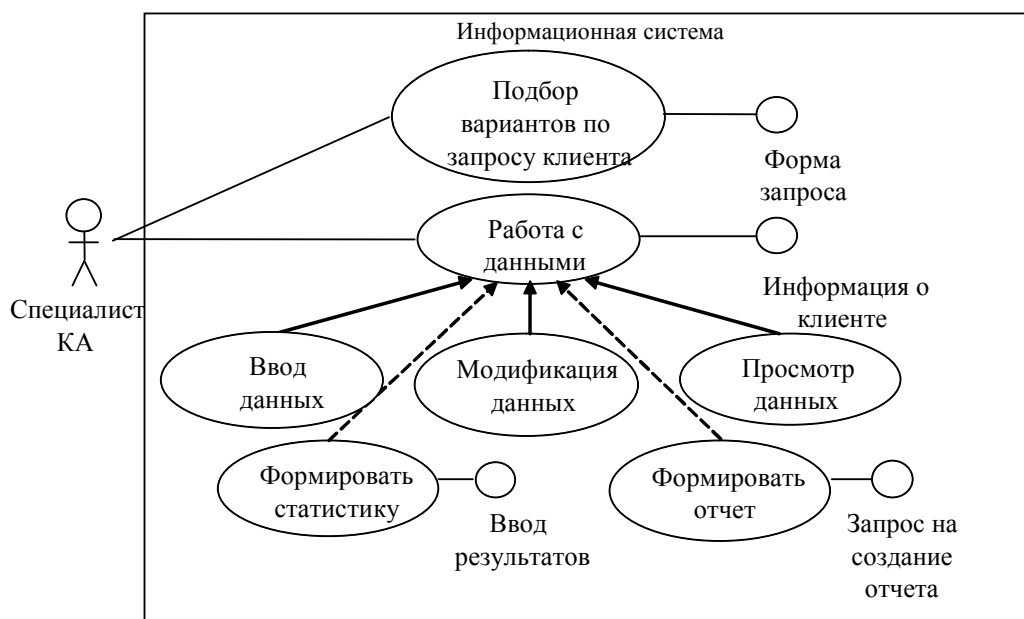


Рисунок 108 – Диаграмма вариантов использования

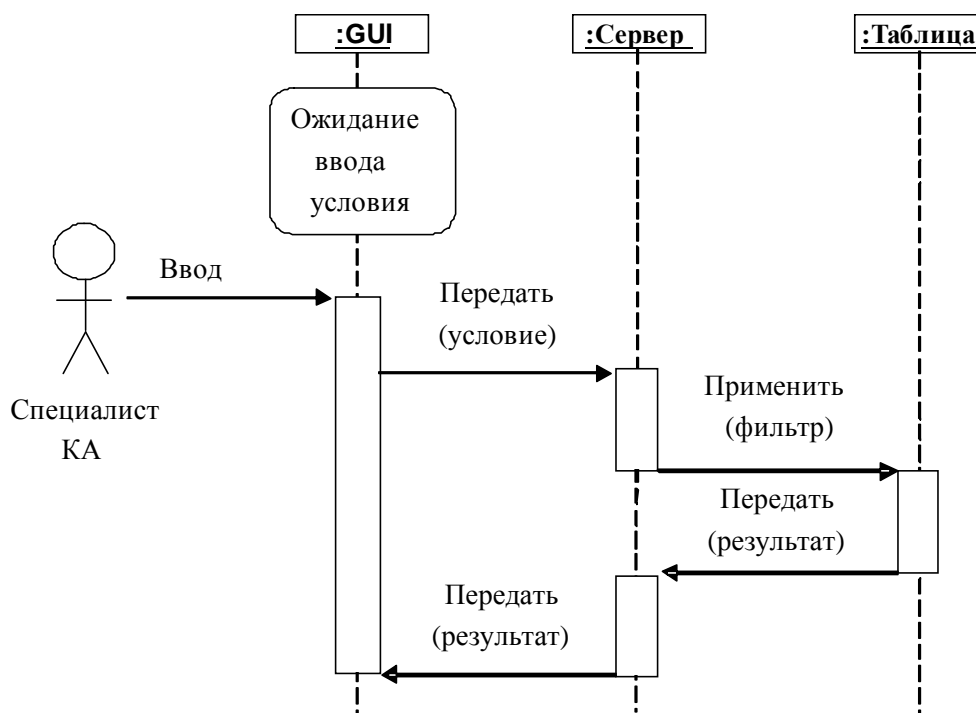


Рисунок 109 – Диаграмма последовательностей для процесса подбора вариантов («Пошук»)

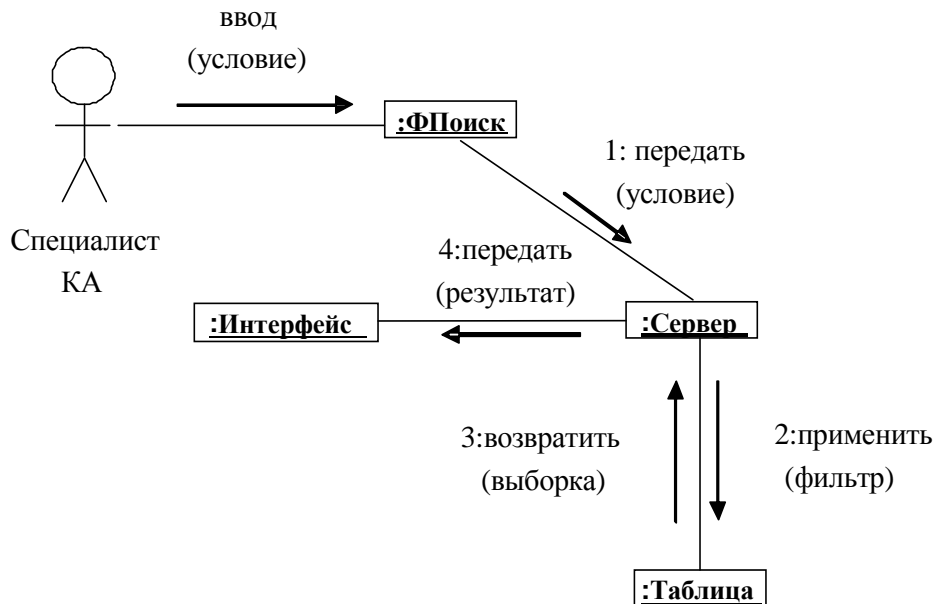


Рисунок 110 – Діаграма кооперацій для процесу підбору варіантів («Пошук»)

Логічне подання системи дозволило зробити аналіз структурних і функціональних відносин між елементами моделі системи. Для створення конкретної фізичної системи, де елементи логічного уявлення мають бути реалізовані в конкретну матеріальну суть, використовувалися діаграми компонентів (рис. 111) і розгортання (рис. 112) з графічним зображенням процесорів, пристроїв, процесів і зв'язків між ними.

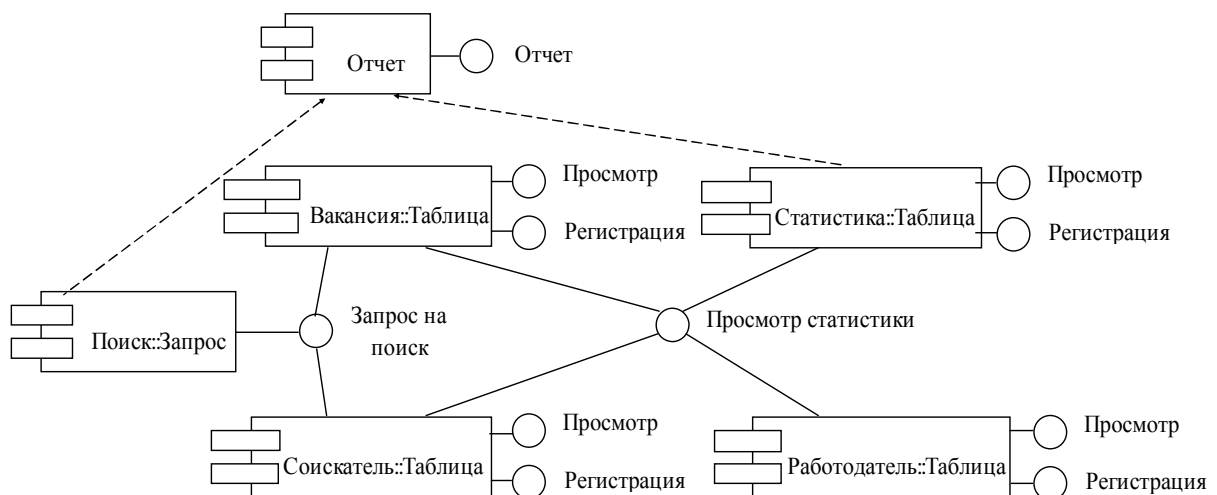


Рисунок 111 – Діаграма компонентів інформаційної системи

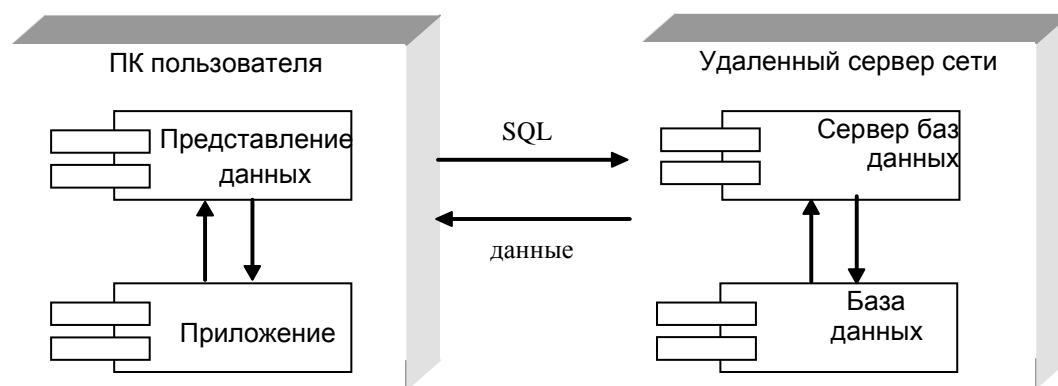


Рисунок 112 – Діаграма розгортання інформаційної системи

Програмна реалізація розробленої моделі була виконана у середовищі Borland Delphi 6. Розроблений програмний продукт на основі побудованої моделі дозволяє підвищити оперативність і якість обслуговування клієнтів за рахунок застосування багатопараметричного двонаправленого пошуку (підбору варіантів). Ведення статистичних даних про результати діяльності кадрового агентства з підбору фахівців і формування відповідних звітів дозволяє багато в чому скоротити об'єми рутинної роботи і в цілому підвищити продуктивність праці працівників кадрового агентства. Результати аналізу ефективності впровадження розробленого програмного продукту показують, що його впровадження дозволить заощадити 2644 грн. на рік за рахунок вивільнення робочого часу фахівця кадрового агентства і в цілому підвищити якість послуг, що надаються [9-13].

4.2 Інформаційна система для автоматизованого складання розкладу занять у вищому навчальному закладі

Кожен навчальний семестр будь-якого вищого навчального закладу починається зі складання розкладу занять. Як правило, при цьому використовується людський чинник без застосування засобів автоматизації. Проте яким би досвідченим не був диспетчер, цей процес є дуже трудомістким; людина не в змозі справитися з такою кількістю інформації, не допустивши при цьому жодної помилки. Без сумніву, це негативно відображається на процесі навчання.

До теперішнього часу розроблена низка комп'ютерних програм для автоматичного складання розкладу. **CyberMatrix Class Sheduler** може використовуватися як одним викладачем, для складання особистого розкладу, так і диспетчерським відділом для складання розкладу занять всього навчального закладу. У програмі присутні засоби фільтрації і пошуку; її недоліком є наявність тільки англійського інтерфейсу, що однозначно викличе утруднення при роботі. «**Моделедром – Розклад**» як початкові дані використовує список дисциплін, список навчальних груп і навчальне навантаження кожної групи; дозволяє друкувати одержані таблиці розкладів з розбиттям по аркушах. На жаль, демо-версія не дає можливість оцінити рівень програми і доцільність її придбання за чималу ціну. **Розклад ПРО** призначена для складання розкладу занять будь-яких навчальних закладів у ручному і автоматичному режимі; у будь-який момент у розклад можна внести зміни, роздрукувати його або експортувати таблицю в MS-Excel. Програма дозволяє вести списки кабінетів, викладачів, предметів, груп (класів), задавати зв'язки між ними і навантаження за предметами; є можливість задати робочий час індивідуально для викладача і пріоритет предметів у розкладі. Передбачається як ручне складання розкладу (з відстежуванням заданих обмежень), так і автоматичний розрахунок – з подальшим «ручним» коректуванням (оскільки не існує точних алгоритмів складання розкладу, в автоматичному режимі пропонуються лише можливі варіанти автоматичного розрахунку). Є універсальним засобом складання розкладів, проте достатньо складна і незручна в обігу. «**Ректор**» також дозволяє складати розклади для будь-якого навчального закладу і передбачає ручний і автоматичний режими роботи; складений розклад може бути виведений у файл форматів Word 97/2000, Excel 97/2000, HTML 4.0 з розбиттям на сторінки. Проте, не дивлячись на заявлену універсальність, дана програма реально підготовлена для роботи виключно у середніх навчальних закладах: усі терміни («урок», «учні» і т.д.) «жорстко зашиті» у програму і не передбачають модифікації. Таким чином, вивчивши наявні програми, можна зробити висновок, що кількість розроблених програм відносно невелика, а якість їх виконання далека від досконалості. Лідером у цій області є ТОВ «Дігси», що створило програму «Розклад Про» і що представило умовно-безкоштовну (ShareWare) версію.

Логічним висновком є рішення спроектувати власну систему для автоматизованого складання розкладу. Основні вимоги до системи формуються таким чином:

- здійснювати як «ручне» складання розкладу, так і автоматичне з можливістю подальшого «ручного» коректування;
- припускати можливість внесення змін у складений розклад (заміна предметів, викладачів, аудиторій);
- виробляти облік виконання навантаження;
- роздруковувати розклади по курсах, групах, кафедрах й іншу звітність (вивід в Excel).

Передбачається, що система як початкові дані повинна використовувати аудиторний фонд, розподіл навантаження по кафедрах (групах) і розподіл навантаження серед викладачів усередині кафедри, тобто початкова інформація для диспетчерів і для програми ідентична і не зажадає додаткової обробки. При цьому програма повинна бути легко освоєна навіть не знайомим з комп'ютерною технікою персоналом, а складання нових або редагування існуючих розкладів займало б не більш години. Кожен варіант розкладу повинен розглядатися як окремий проект; ввівши одного разу початкові дані, можна буде надалі робити копію проекту і працювати з нею.

Через достатню складність спроектованої інформаційної системи звичний структурний підхід не підходить, оскільки при ньому основою системи є алгоритм – послідовність дій за рішенням задач. Запис процесу складання розкладу у вигляді алгоритму недоцільний, і ми застосуємо об'єктно-орієнтований підхід. Інформаційна система у такому разі буде сукупністю взаємозв'язаних об'єктів.

Розробка інформаційної системи проводиться у три етапи:

- 1 Об'єктно-орієнтований аналіз наочної області: визначення ключових абстракцій, ідентифікація класів і об'єктів.
- 2 Концептуальне, логічне і фізичне моделювання інформаційної системи; побудова відповідних діаграм.
- 3 Програмна реалізація розробленої моделі.

Перший етап включає аналіз дій зі складання розкладу, виділення активних і пасивних класів (об'єктів). Актором (активним об'єктом) тут є диспетчер, пасивними об'єктами – таблиці, що надаються кафедрами згідно з навчальними планами, і аудиторний фонд.

На другому етапі створюється інформаційна модель проектованої системи, для чого використовується уніфікована мова моделювання UML. Спочатку формулюються вимоги до системи, що розробляється, визначаються функції, які вона повинна реалізувати, і задачі, які вона повинна вирішувати. На основі цих даних формується діаграма варіантів використання (рис.113).

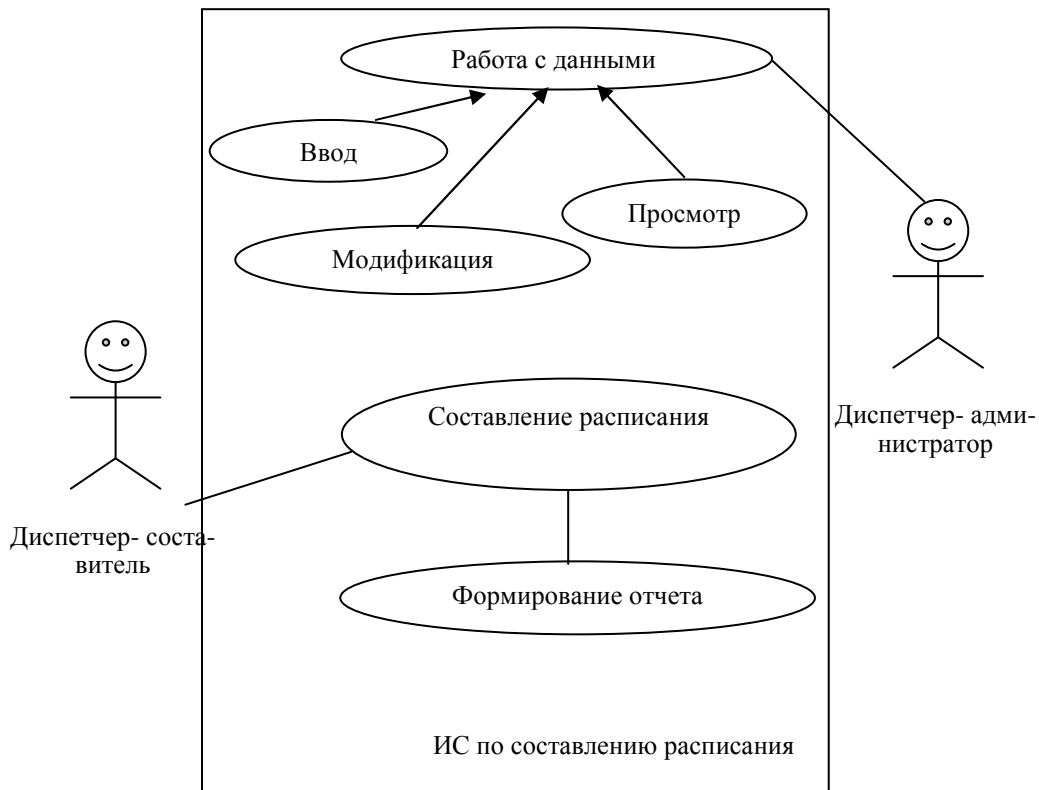


Рисунок 113 – Діаграма варіантів використання

Під час роботи з системою працюватимуть два актори (активних об'єкта), один з яких займатиметься роботою з даними «диспетчер-адміністратор», а другий – власне складанням розкладу і всіляких звітів (диспетчер-укладач). Передбачається наявність таких варіантів використання системи: складання розкладу, формування звіту і робота з даними, яка припускає введення, модифікацію і перегляд даних.

Далі будується структурно-логічна схема системи у вигляді діаграми класів (рис.114).

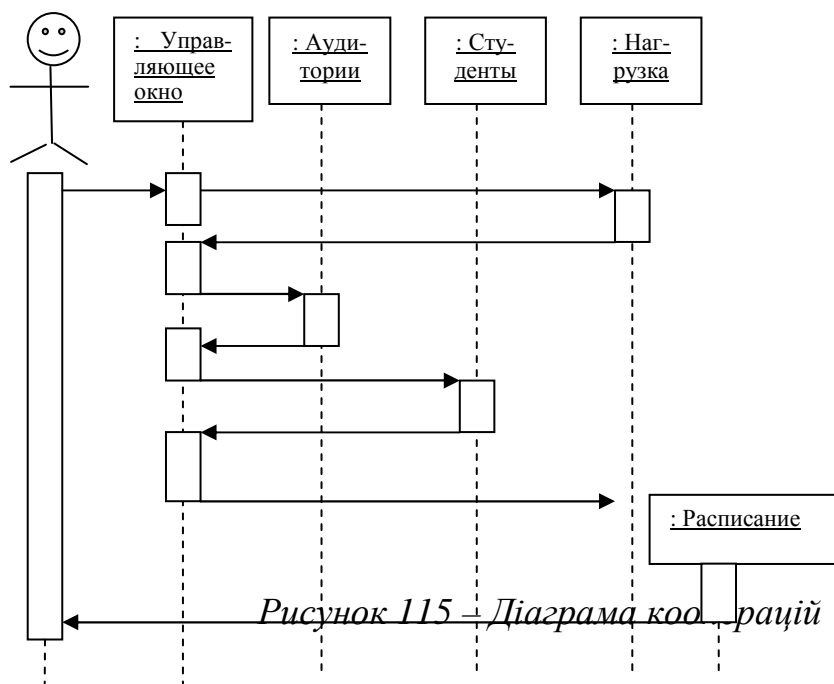


Рисунок 114 – Діаграма класів

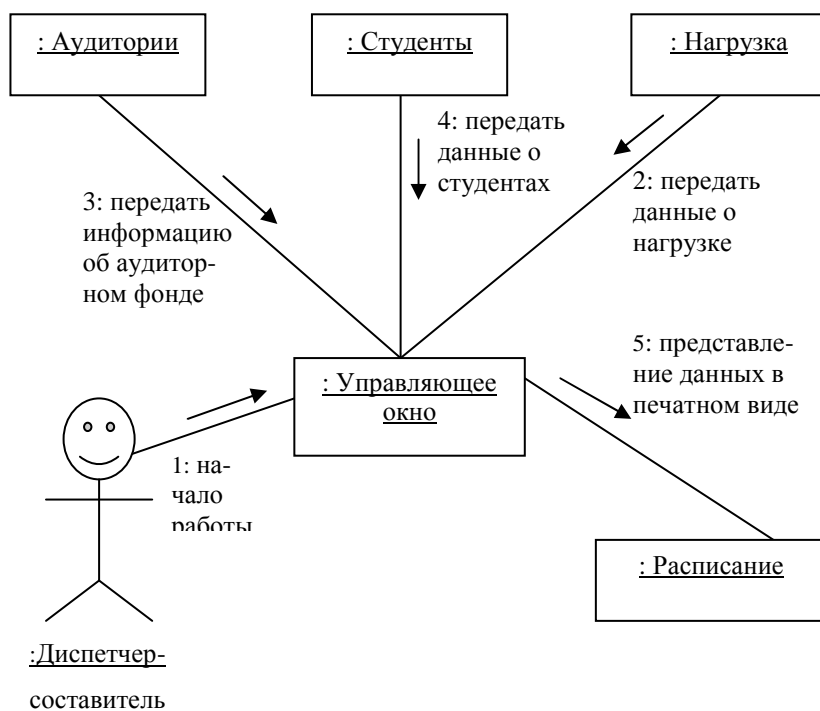
Ця діаграма служить для зображення статичної структури моделі системи, різних взаємозв'язків між окремою суттю наочної області, такими як об'єкти і підсистеми, а також опису їх внутрішньої структури і типів відносин.

Ми припускаємо існування в системі наступних класів: аудиторний фонд, студенти, навантаження, розклад, управляюче вікно і таблиця. Клас «Таблиця» є абстрактним: він не має екземплярів і пов'язаний з класами «Аудиторний фонд», «Студенти» і «Навантаження» відношенням спадкоємства. Стереотип «control» класу «Управляюче вікно» указує на те, що цей клас відповідає за координацію дій інших класів.

Динамічні аспекти поведінки системи (власне процесу складання розкладу) відображаються на діаграмах кооперації і послідовності. На діаграмі кооперації поведінка системи описується на рівні окремих об'єктів, які обмінюються між собою повідомленнями, щоб досягти потрібної мети або реалізувати деякий варіант використання (рис. 115).



За допомогою діаграм послідовності можна описати повний контекст взаємодій як своєрідний часовий графік «життя» всієї сукупності об'єктів, що взаємодіють між собою для реалізації варіанта використання програмної системи, досягнення бізнес-целі або виконання якої-небудь задачі (рис. 116).



На діаграмі видно, що ініціатором взаємодії є диспетчер-укладач, який володіє фокусом управління впродовж всієї роботи системи, постійно контролюючи її діяльність. З об'єктів «Навантаження», «Аудиторний

фонд» і «Студенти» збирається відповідна інформація шляхом переходу фокусу управління від управляючого вікна до відповідного об'єкта і назад. Коли зібрана вся необхідна інформація, створюється об'єкт «Розклад», одержуючий тимчасовий фокус управління.

Діаграма станів описує процес зміни станів системи або її підсистеми при реалізації всіх варіантів використання. При цьому зміна станів окремих елементів системи може бути викликана зовнішніми діями з боку інших елементів або ззовні системи. Саме для опису реакції системи на подібні зовнішні дії і використовуються діаграми станів (рис. 117).

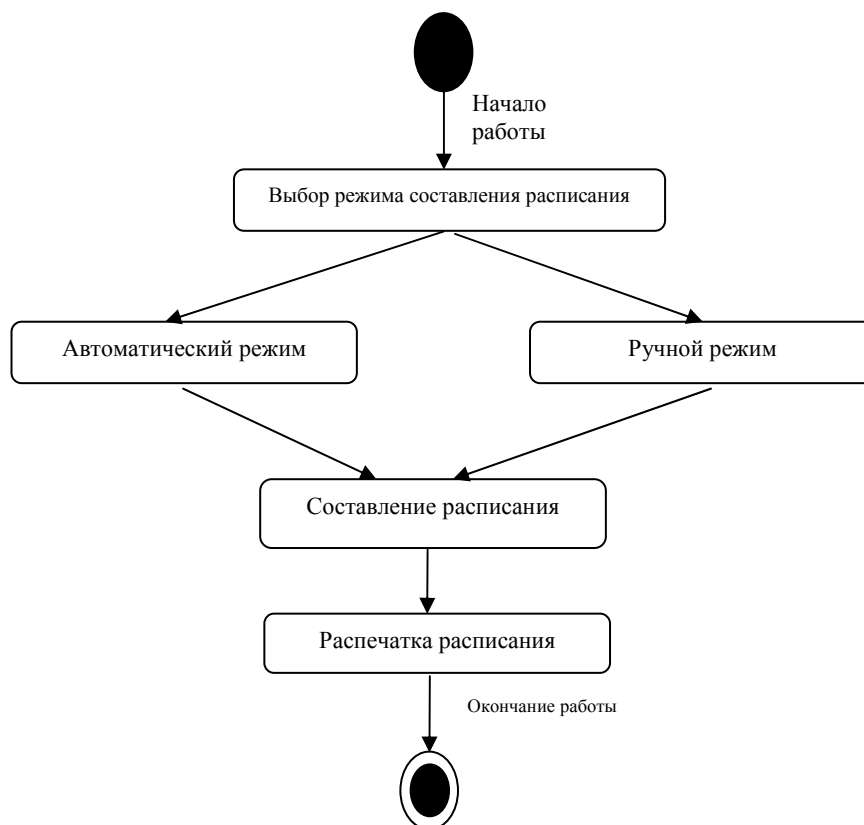
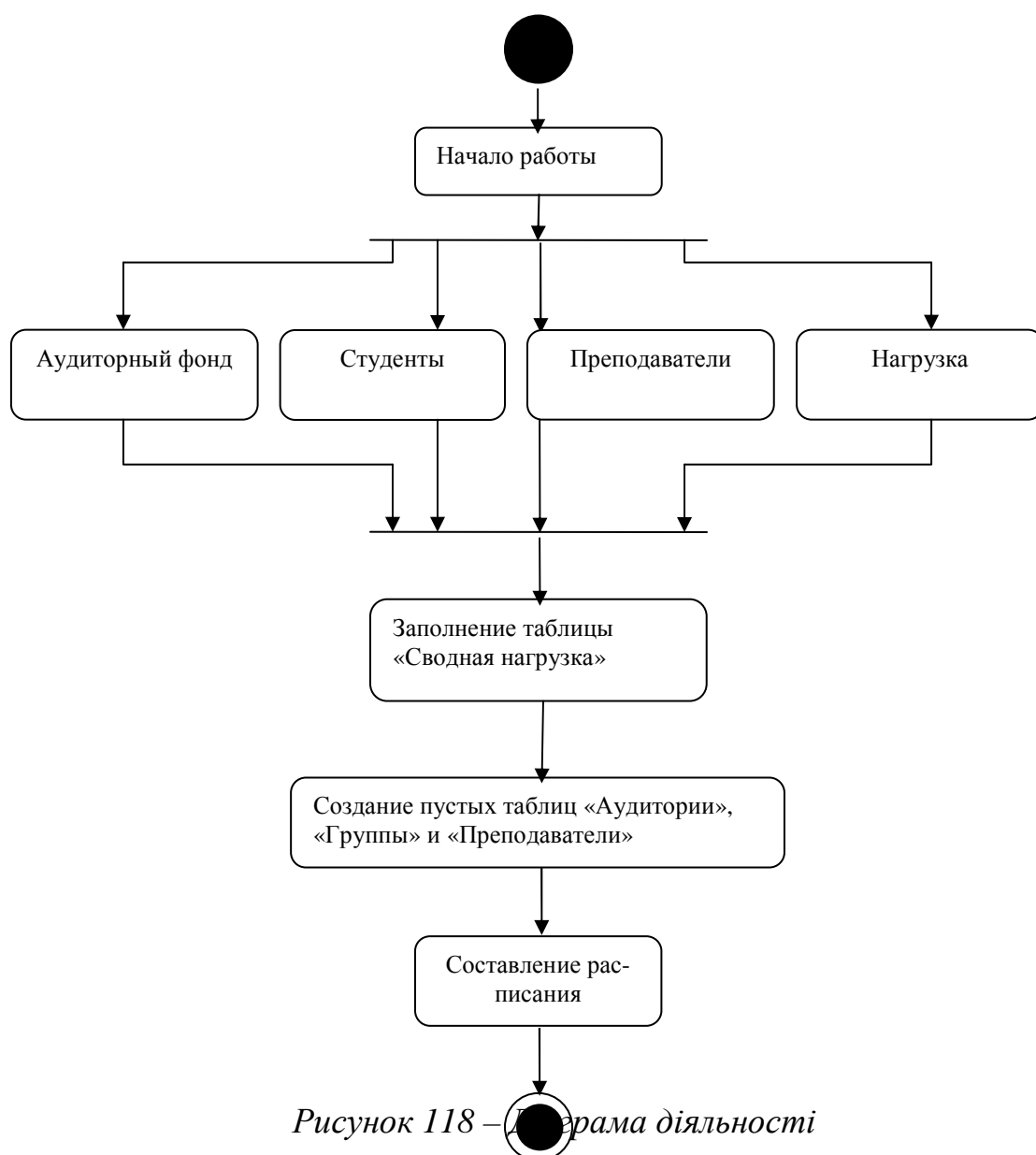


Рисунок 117 – Діаграма станів

Для моделювання процесу виконання операцій у мові UML використовуються так звані діаграми діяльності. Кожен стан на цій діаграмі відповідає виконанню деякої елементарної операції, а перехід у наступний стан спрацьовує тільки при завершенні операції в попередньому стані. Графічно діаграма діяльності зображується у формі графа діяльності, вершинами якого є стани дії, а дугами – переходи від одного стану дії до іншого. Таким чином, діаграми діяльності можна вважати окремим випадком діаграм станів (рис. 118).

Для створення конкретної фізичної системи необхідно деяким чином реалізувати всі елементи логічного уявлення в конкретну матеріальну суть. Для опису такої реальної суті призначений інший аспект модельного уявлення, а саме – фізичне подання моделі. У мові UML для фізичного подання моделей систем використовуються так звані діаграми компонентів і діаграми розгортання, які дозволяють визначити архітектуру системи, що розробляється, встановивши залежності між програмними компонентами (рис. 119).



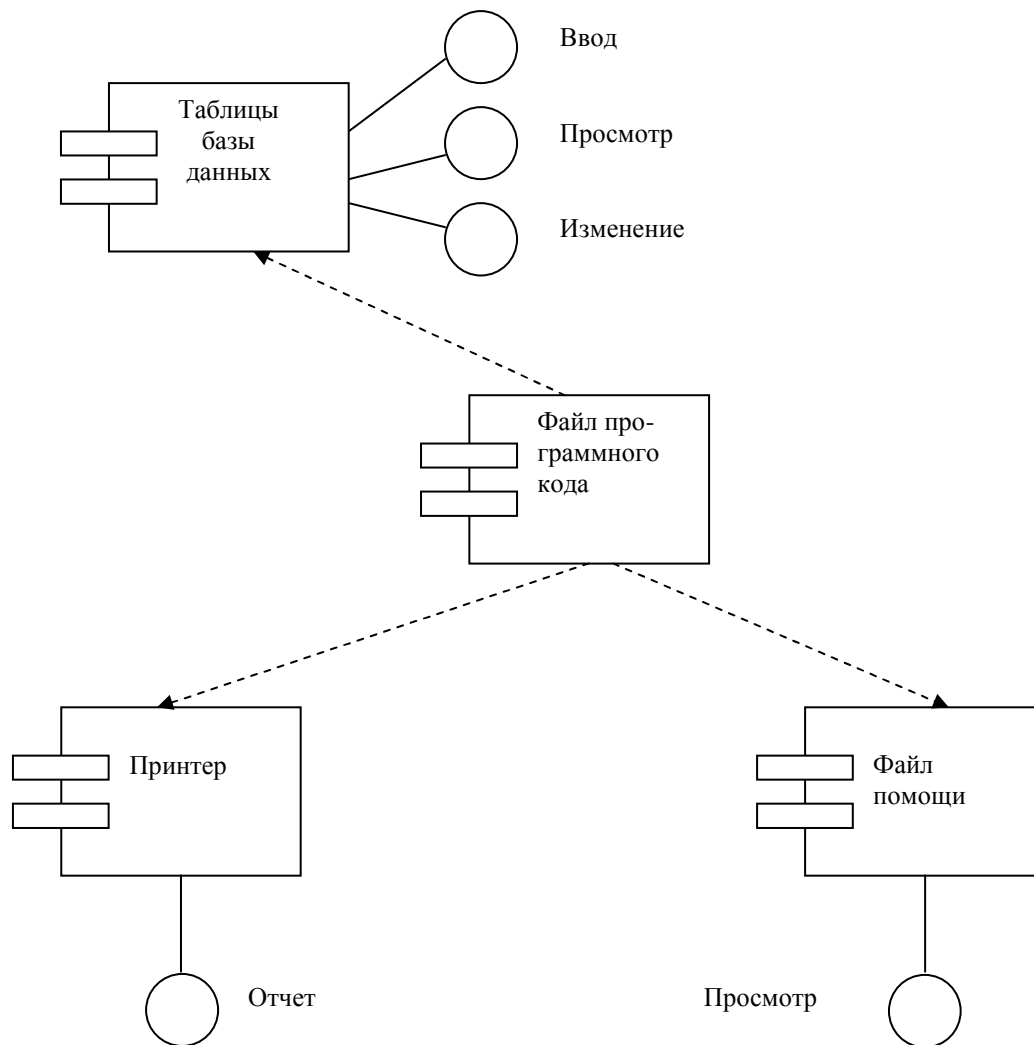


Рисунок 119 – Діаграма компонентів

Спроектowana інформаційна система для автоматизації процесу складання розкладу занять була реалізована в середовищі Borland Delphi [14-19].

Розрахований коефіцієнт економічної ефективності системи виявився 3,43; термін окупності витрат на упровадження інформаційної системи склав 0,29 років.

4.3 Інформаційна система для спеціалізованого торгового підприємства

Робота спеціалізованого торгового підприємства «Магазин-салон з продажу компакт-дисків» припускає облік різновидів компакт-дисків (Soft, Education, Game, Video CD, DVD, MP3, MP4, Audio CD), форматів запису інформації на них (Стандартний, Video CD, DVD, MP3, MP4, Audio CD), категорії інформації, яка може розповсюджуватися на магнітних дисках, і т.п. Крім того, крім власне продажу дисків, здійснюється їх прокат, запис інформації на носії клієнта, а також продаж додаткових товарів (дискет, CD-R, CD-RW, полицок під диски і ін.). Для успішного функціонування такого підприємства необхідне використання інформаційних систем і технологій: за допомогою ефективної інформаційної системи можна значно спростити процеси контролю і управління. Як правило, за відділом продажу компакт-дисків закріплений один комп'ютер, що використовується для обліку наявності компакт-дисків, обліку проведених за день операцій з продажу і перевірки всіх видів дисків на працездатність.

У даний час для обліку руху товару недостатньо використовувати пакет MS Office – необхідна інформаційна система або універсальна програма масового призначення, побудована під конкретне підприємство з конкретним видом товару. Іншими словами, для спеціалізованого підприємства потрібна спеціалізована програма обліку і контролю.

Існують вже розроблені універсальні програми масового призначення, які спрощують облік наявності товару на складі, облік руху товару (поставки, продаж і т.д.). Прикладом може служити система програм «1С:Предприятие». Вона складається з декількох модулів (компонент), які вирішують конкретні чітко поставлені задачі: «1С:Бухгалтерія», «1С:Зарплата і кадри», «1С:Производство» і «1С:Торговля і склад». Останній автоматизує роботу на всіх етапах діяльності підприємства, містить засоби забезпечення збереження і несуперечності інформації, може бути адаптований до будь-яких особливостей обліку на конкретному підприємстві. Проте у пакеті «1С:Торговля і склад» є декілька недоліків:

- Достатньо висока вартість. Розрахована на одного користувача версія програми обійдеться підприємству в \$280; призначена для користувача мережева версія програми – \$480; файл-серверная мережева

- без обмеження кількості користувачів – \$960. Додатково до цієї вартості щомісячні оновлення програми коштуватимуть від \$20 до \$100.
- Складнощі в настройці програми. Для настройки програми «1С:Торговля і склад» під конкретне спеціалізоване підприємство, а також для її подальшого обслуговування необхідне або створення додаткового робочого місця, або донавчання співробітників.

Тому можна зробити висновок, що для обліку руху товару і контролю діяльності підприємства з продажу і прокату компакт-дисків створення нової інформаційної системи зі всіма необхідними функціональними можливостями – найдешевший і якнайменше трудомісткий шлях.

Для створення інформаційної моделі проектованої системи використовується уніфікована мова моделювання UML. Спочатку формулюються вимоги до системи, що розробляється, визначаються функції, які вона повинна реалізувати, і задачі, які вона повинна вирішувати. На основі цих даних формується діаграма варіантів використання (рис. 120).

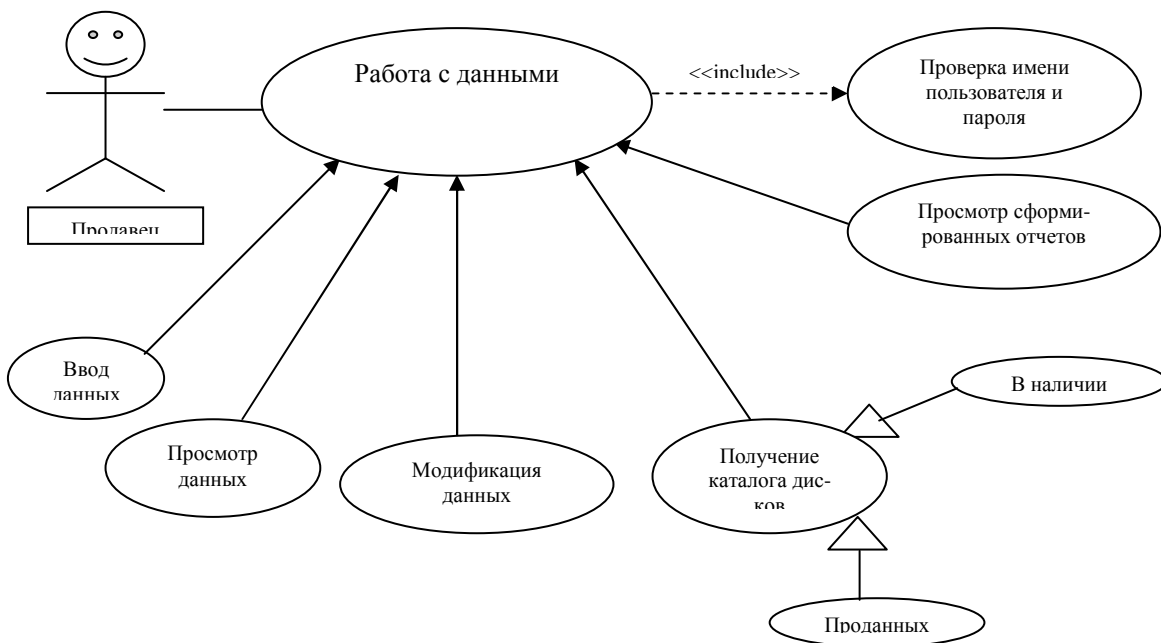


Рисунок 120 – Діаграма варіантів використання

Діаграма варіантів використання описує функціональне призначення системи або, іншими словами, те, що система робитиме під час свого функціонування. Діаграма варіантів використання є початковим концептуаль-

ним уявленням або концептуальною моделлю системи під час її проектування і розробки.

На діаграмі видно, що під час роботи з програмою працюватиме один актор – продавець. Передбачається наявність таких варіантів використання системи, як: введення даних, перегляд даних, модифікація даних, отримання каталога дисків, перегляд сформованих звітів.

Діаграма класів служить для подання статичної структури моделі системи, різних взаємозв'язків між окремою суттю наочної області, такими як об'єкти і підсистеми, а також опису їх внутрішньої структури і типів відносин. Діаграма класів нашої інформаційної системи зображена на рисунку 121. Тут зображені наступні класи: інтерфейс, таблиця, постійні клієнти, каса, постачальники, історія і управляюче вікно. Клас «Таблиця» є абстрактним, який не має екземплярів і об'єктів. Усі операції даного класу характеризуються областю видимості типу `public`, тобто ці операції видні і доступні з будь-якого іншого класу. Стереотип «control» класу «Управляюче вікно» указує на те, що це управляючий клас, який відповідає за координацію дій інших класів. Незафарбований трикутник, направлений до нього від решти класів, указує на наявність відношення ненаправленої бінарної асоціації, тобто наявність довільного відношення або взаємозв'язку між класами. Так, класи «Таблиця», «Інтерфейс», «Постійні клієнти» і «Постачальники» передають дані в клас «Управляюче вікно», а клас «Управляюче вікно» дозволяє подати дані у вигляді звітів про історію проведених операцій і про динаміку величин кас за певний час.

Для розгляду взаємодії об'єктів у контексті статичної структури моделі для зображення структурних особливостей передачі і прийому повідомлень між об'єктами використовується діаграма кооперації. Дана діаграма візуалізує об'єкти (екземпляри класів), зв'язки (екземпляри асоціацій) і повідомлення.

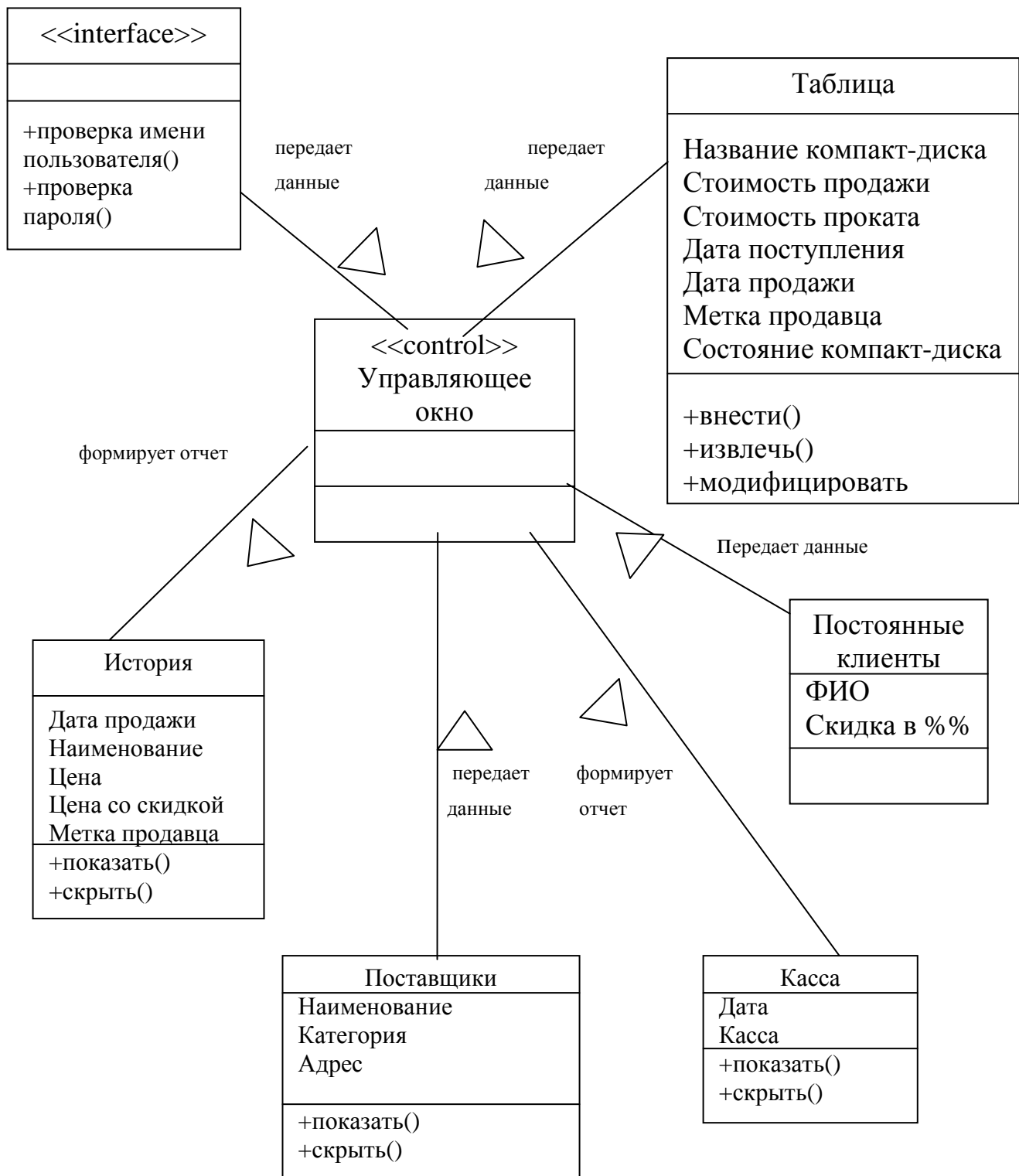


Рисунок 121 – Діаграма класів

Діаграма кооперацій нашої інформаційної системи наведена на рисунку 122.

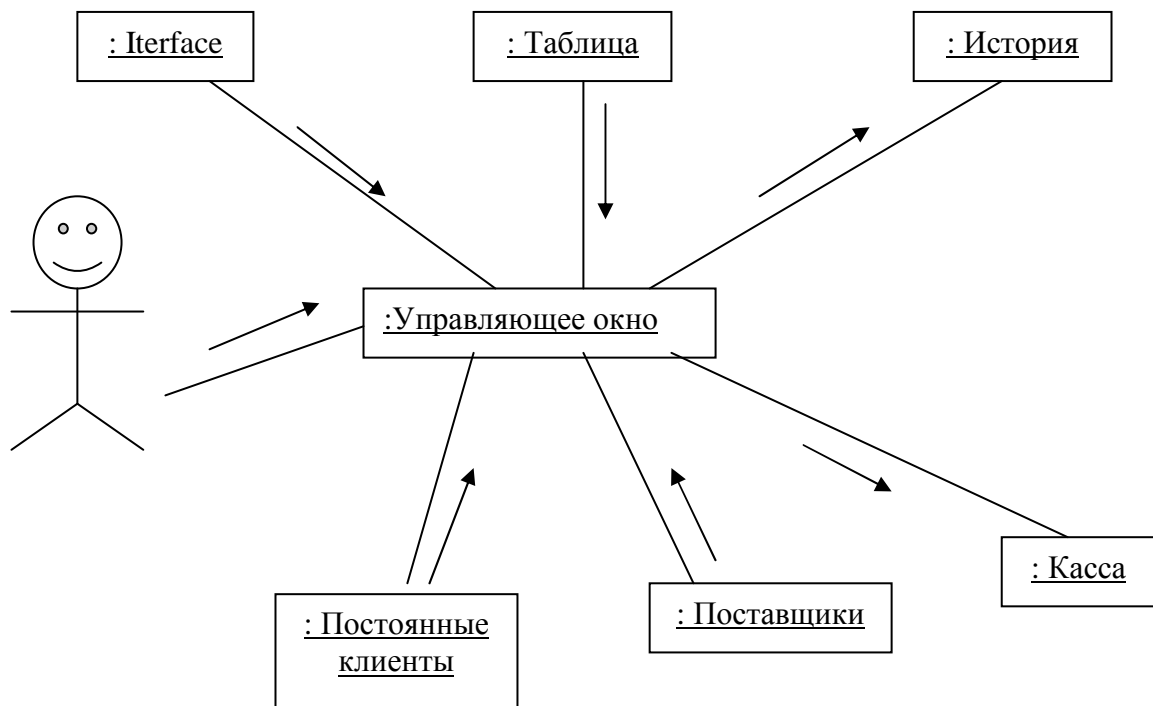


Рисунок 122 – Діаграма кооперації

Тут подані такі класи: таблиця, інтерфейс, історія, каса, постачальники, постійні клієнти. Робота у системі має наступний порядок: продавець починає роботу з управляючим вікном, ввівши своє ім'я і пароль. Далі він ініціює передачу даних про постачальників з класу «Постачальники», постійних клієнтів з класу «Постійні клієнти», формує звіти «Каса» і «Історія», працює безпосередньо з даними таблиці.

Діаграма послідовності відображає часовий аспект поведінки системи (рис. 123). На діаграмі видно, що ініціатором взаємодії є продавець, який володіє фокусом управління впродовж всієї роботи системи, постійно контролюючи її діяльність.

Для моделювання процесу виконання операцій у мові UML використовуються діаграми діяльності. Діяльність є деякою сукупністю окремих обчислень, виконуваних автоматом; при цьому окремі елементарні обчислення можуть приводити до деякого результату або дії.

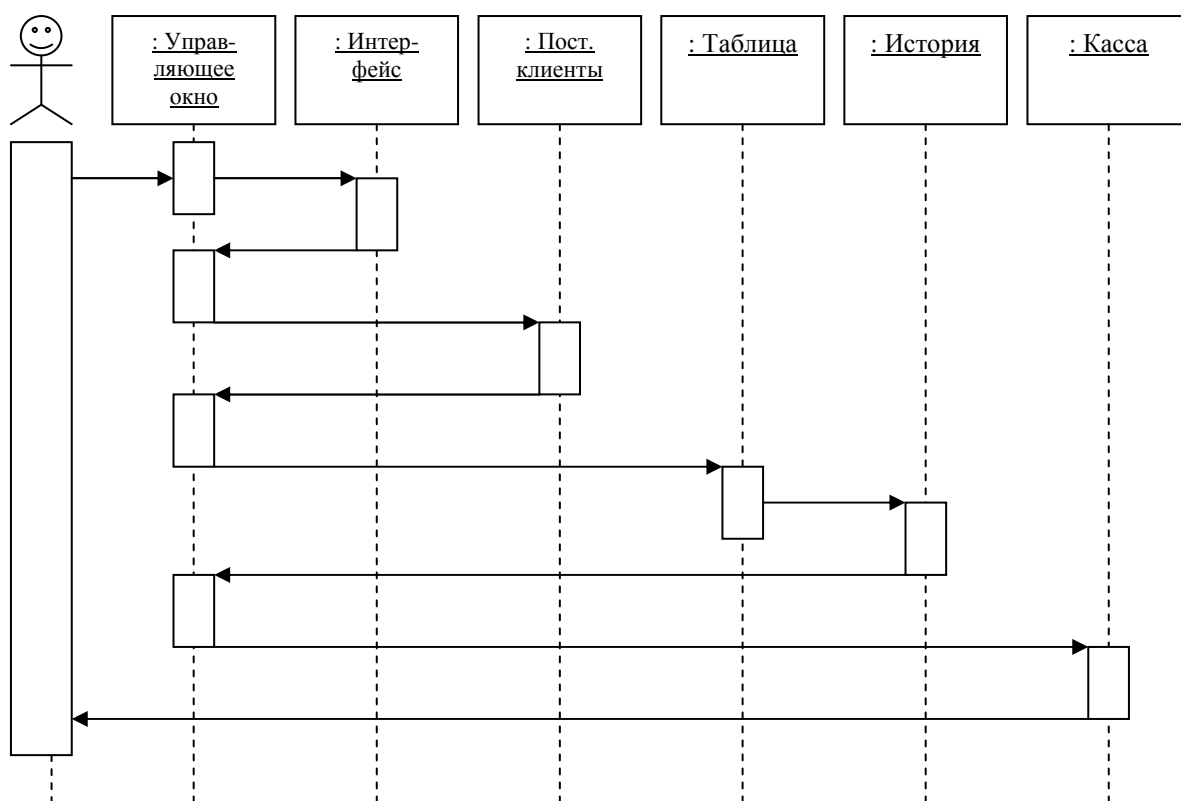


Рисунок 123 – Діаграма послідовності

Діаграма діяльності інформаційної системи зображена на рисунку 124. По ній видно, що на потребу клієнта перевіряється наявність диска (візуально або за допомогою інформаційної системи). Якщо диск є в наявності, і він задовольняє всім вимогам, проводиться перевірка, клієнт постійний чи ні, і вид операції (продаж або прокат). Залежно від одержаної інформації диск списується у продаж або прокат за звичною ціною або за ціною із знижкою.

Для створення конкретної фізичної системи необхідно деяким чином реалізувати всі елементи логічного уявлення в конкретну матеріальну суть. Для опису такої реальної суті призначений інший аспект модельного уявлення, а саме – фізичне подання моделі. У мові UML для фізичного подання моделей систем використовуються так звані діаграми компонентів, які дозволяють визначити архітектуру системи, що розробляється, встановивши залежності між програмними компонентами, в ролі яких може виступати початковий, бінарний і виконуваний код.

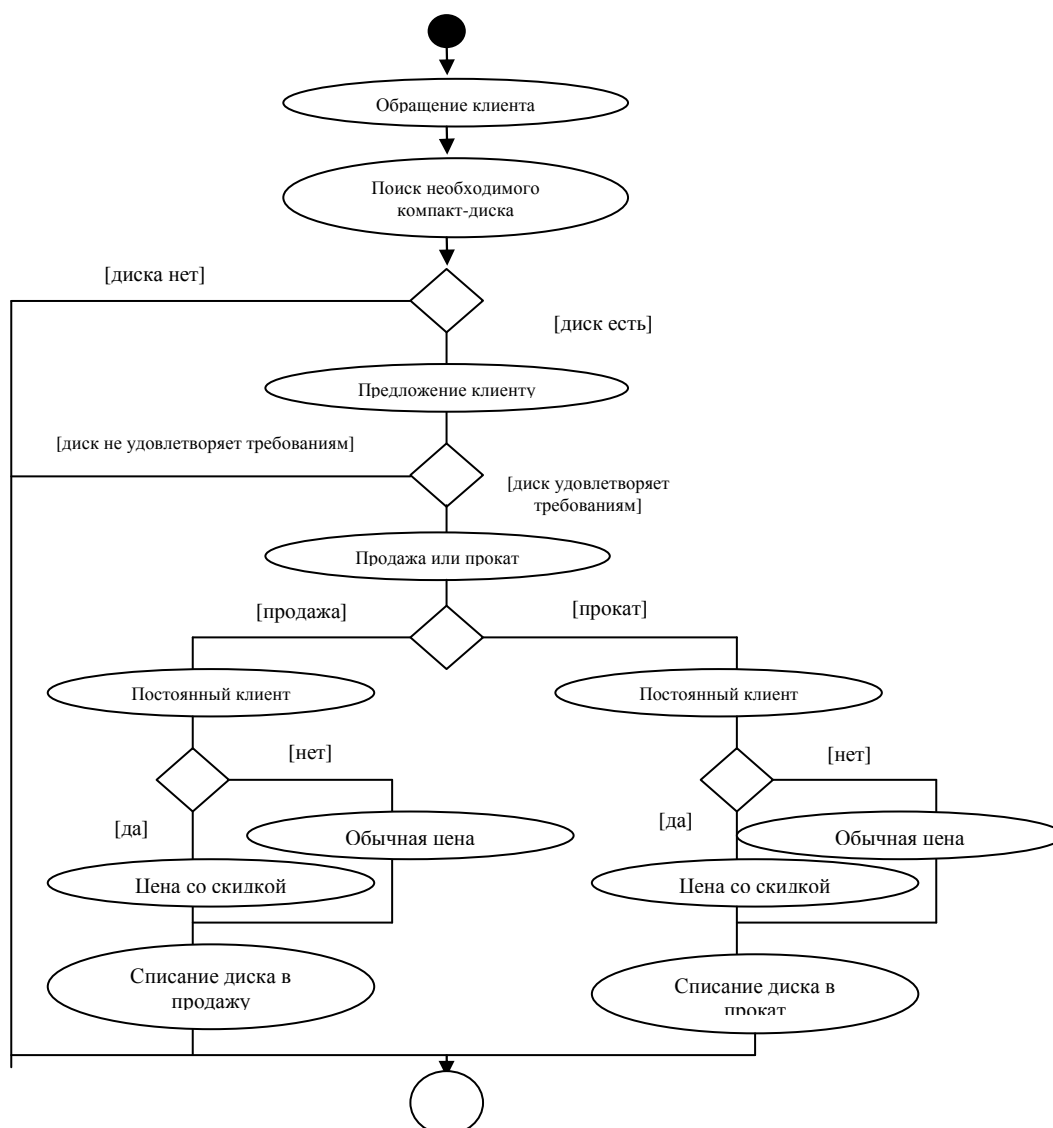


Рисунок 124 – Діаграма діяльності

У багатьох середовищах розробки модуль або компонент відповідає файлу. Діаграма компонентів забезпечує узгоджений перехід від логічного уявлення до конкретної реалізації проекту у формі програмного коду.

Система була реалізована в середовищі програмування Borland Delphi і упроваджена на торговому підприємстві [20-21]. Проведений розрахунок економічної ефективності показав, що упровадження інформаційної системи дозволить заощадити 1035,5 грн. на рік за рахунок вивільнення робочого часу працівників відділу з продажу компакт-дисків. При цьому буде збільшена швидкість обслуговування клієнтів, що відобразиться на величині виручки і, відповідно, і на прибутку.

4.4 Інформаційна система для невеликої страхової компанії

У страховому бізнесі, як і в будь-якій іншій галузі економіки, йде безперервний пошук ефективних технологій і методів підвищення рентабельності. Сучасне ринкове суспільство неможливо собі уявити без страхування як особливого виду економічних відносин. Існує прямий зв'язок між рівнем добробуту суспільства, ступенем розвитку ринкових відносин і рівнем розвитку страхування. У країнах, які є світовими лідерами у сфері соціальних і ринкових відносин (США, Японія, європейські держави і інші), страхування є однією з найстабільніших і динамічніших галузей народного господарства.

Розвиток страхової діяльності на Україні супроводжується низкою проблем. Однією з найважливіших проблем страхування є повільна (порівняно з банківською діяльністю) автоматизація страхування. Ця проблема найгостріше виявляється в отриманні оперативної інформації щодо укладених договорів страхування, що, у свою чергу, не дозволяє повною мірою реагувати на зовнішні дії і ефективно ухвалювати управлінські рішення. Особливо це стосується невеликих страхових компаній, які мало відомі на страховому ринку.

На даний час найпопулярнішими є різні рішення, розроблені на основі сучасних інформаційних технологій. Для страхових компаній ринок сучасних ІТ-технологій пропонує різні за апаратним складом і програмним забезпеченням – від окремих систем, наборів додатків і модулів, до універсальних систем, які забезпечують досить повну автоматизацію всіх виробничих, технологічних і допоміжних процесів.

Так, модульна інформаційна система INSTRAS-4 призначена для комплексної автоматизації обліку в страхових і перестрахових компаніях. Основне призначення INSTRAS-4 – підвищення конкурентоспроможності страхової компанії за рахунок важливого поліпшення керованості. Замикає всі дані і всі бізнес-процеси у страховій компанії в єдиний інформаційний простір, погоджуючи і оптимізуючи їх. Забезпечує комплексну автоматизацію всіх ключових відділів і служб страхової (перестрахової) компанії. Є могутнім інструментом перетворень і вдосконалення роботи компанії, створення і виводу на ринок нових страхових продуктів. Дозволяє органі-

зувати безперебійний обмін інформацією з філіалами, агентствами і точками продажів. Включає інтерфейс до найпопулярніших на українському ринку програм бухгалтерського обліку. Допускає мінімальні витрати на придбання, впровадження, підтримку, супровід і розвиток.

Для використання в страховому маркетингу пропонується ще цілий ряд програмних продуктів. Наприклад, Marketing Expert і БЕСТ Маркетинг – для стратегічного планування, Ластівка і Маркетинг Мікс – для стратегічного і оперативного планування. Програма Marketing Analytic призначена для аналізу, прогнозу, планування і містить у собі елементи CRM. Програма Sales Expert забезпечує управління прямими продажами. Спеціалізовані програмні продукти, призначені для вирішення задач планування, мають й інші вбудовані маркетингові інструменти, використовувані для SWOT-аналізу і ряду інших маркетингових операцій під час обробки даних. У тій чи іншій мірі перелічені вище програмні продукти можна застосовувати з метою прогнозування і розробки сценаріїв розвитку подій на користь рішення маркетингових задач страхової компанії.

Ринок IT-технологій пропонує й інші інформаційні рішення. Наприклад, система Contact Manager є універсальним засобом адміністрування робочого часу і управління системою збуту. Вона дозволяє ефективно контролювати ділову активність співробітників системи продажів на всіх рівнях керівництва, відстежувати історію контактів фахівців агентської мережі – від першого телефонного дзвінка до укладення договору. Contact Manager інтегрований з системою операційного обліку договорів. Можлива інтеграція з технологіями call-центру, бізнес-планування і звітності.

Нова версія інформаційної системи UNICUS EASY ОСАГО v. 1.6. є автоматизованим робочим місцем (АРМ) працівника страхової компанії і дозволяє виконувати наступні операції: повнофункціональний продаж полісів ОСАГО; врегулювання збитків; облік бланків строгої звітності; експорт даних у форматі XML в зовнішні інформаційні бази.

Система ICI (Insurance Company Information System), яка випускається фірмою T-systems, адаптується до різноманітних бізнесів-моделей і може розглядатися як проміжний варіант між індивідуальним і стандартним програмним забезпеченнями. У цій системі комплексної автоматизації, розробленій спеціально для страхових компаній, є такі сучасні функції, як CRM, управління документообігом, ведення статистики й інші. Ця система

легко інтегрується з бек-офісними рішеннями, що вже є в страховій компанії. Програма Connect Insurance, розроблена австрійською компанією, працює за принципом «time to market» (процес від виникнення ідеї про новий страховий продукт до його виходу на ринок) і модернізацією системи продажів страхової компанії без внесення змін в існуючу ІТ архітектуру.

Але треба пам'ятати, що інтегровані інформаційні системи типа Insurance Company використовуються, в основному, великими страховими компаніями, оскільки мають високу ринкову вартість і тому недоступні для дрібних страхових компаній. Для автоматизації своєї діяльності дрібні страхові компанії використовують або системи власних розробок, або недорогі системи, які пропонують обмежений набір можливостей.

Ми пропонуємо інформаційну систему, розраховану на велике коло користувачів, тобто для тих офісних робітників, робота яких безпосередньо пов'язана з базою даної інформаційної системи. Проектована система не виключає використання її будь-якими невеликими страховими компаніями, припускає повну автоматизацію діяльності страхової компанії і підтримує повний цикл процесу обробки інформації за всіма видами страхування.

Розробка інформаційної системи традиційно проводиться у три етапи: об'єктно-орієнтований аналіз наочної області; концептуальне, логічне і фізичне моделювання інформаційної системи; програмна реалізація розробленої моделі.

Перший етап включає аналіз дій, здійснюваних у процесі страхування, виділення активних і пасивних класів (об'єктів). Актором (активним об'єктом) у нашому випадку буде оператор введення даних (у першому наближенні), а пасивними об'єктами – дані за укладеними договорами страхування.

На другому етапі створюється інформаційна модель проектованої системи, для чого використовується уніфікована мова моделювання UML.

Проектування починається з формулювання вимог до системи, що розробляється, визначення функцій, які вона повинна реалізовувати, і задач, які вона повинна вирішувати. На основі цих даних формується діаграма варіантів використання (рис.125). У нашому випадку діаграма варіантів використання («прецедентів») ґрунтується на вимогах, функціях, задачах страхової компанії, які повинна виконувати система, що розробляється.

Згідно з діаграмою прецедент «Робота з даними» є основою для прецедентів: отримання даних; введення даних; модифікація даних; перегляд даних; контроль правильності введених даних.

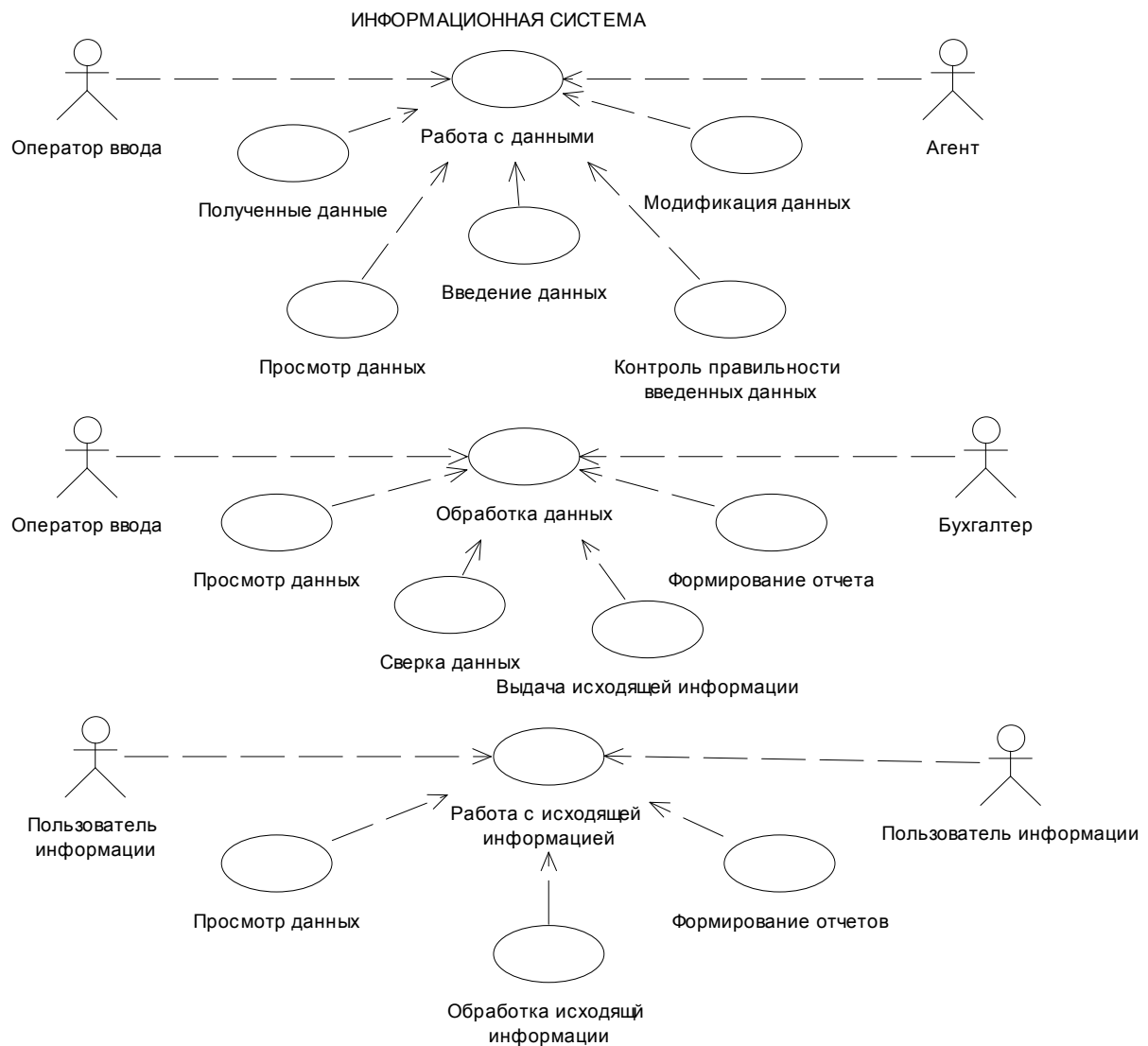


Рисунок 125 – Діаграма варіантів використання

Прецедент «Обработка данных» є основою для прецедентів: перегляд даних; звірка даних; формування звіту; видача початкової інформації.

Прецедент «Робота з початковою інформацією» є основою для таких прецедентів: перегляд даних; обробка початкової інформації; формування звітів.

Далі будується структурна схема системи у вигляді діаграми класів. Діаграми класів є відправною точкою процесу розробки. Діаграми класів допомагають при аналізі наочної області. Вони дозволяють аналітику спіл-

куватися з клієнтом в його термінології і стимулюють процес виявлення важливих деталей у проблемі, яку потрібно вирішити.

Аналіз наочної області процесу функціонування невеликої страхової компанії зображений на рисунку 126.

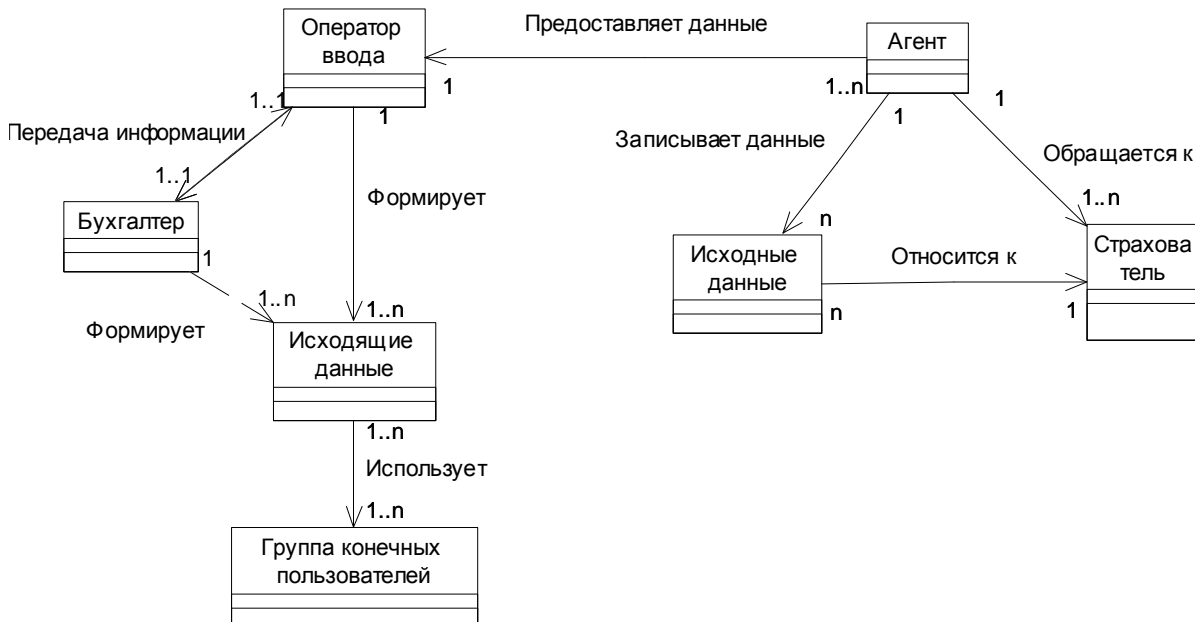


Рисунок 126 - Диаграма асоціацій класів

Клієнт-страхувальник звертається до страхової компанії до клієнта-агента з метою укласти договір страхування. Узгоджуються умови страхування, підписується договір страхування.

Клієнт-агент передає записані початкові дані клієнта-страхувальника (страховий поліс) клієнту-оператору введення, який, у свою чергу, обробляє ці дані і заносить в ПК (збереження в базу). Також клієнту-оператору введення поступають бухгалтерські дані від клієнта-бухгалтера. Після того, як всі дані оброблені і збережені в базі, клієнт-оператор введення формує початкову інформацію (у вигляді звіту певної форми) і передає клієнту-бухгалтеру для звірки. У свою чергу, клієнт-бухгалтер після перегляду і обробки одержаної інформації також формує початкову інформацію (у вигляді звіту певної форми). Сформовані та оброблені дані, як клієнтом-оператором введення, так і клієнтом-бухгалтером передаються клієнту-групі кінцевих користувачів. Вербальний опис процесу роботи страхової компанії з клієнтами дозволяє виділити наступні класи: клієнт-страхувальник, клієнт-агент, клієнт-оператор введення, клієнт-бухгалтер,

клієнт-група кінцевих користувачів, вхідні дані, витікаючі дані. На рисунку 127 подана деталізація класів.

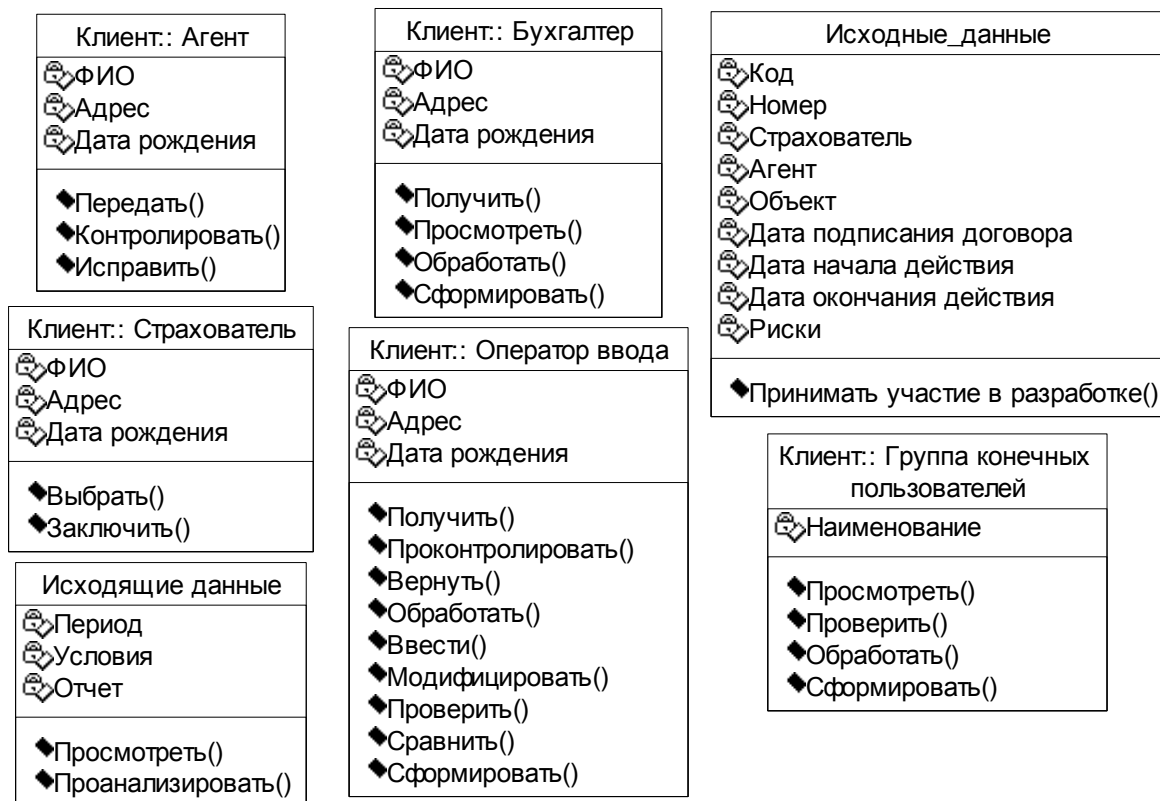


Рисунок 127 – Діаграма деталізації класів

Діаграма послідовностей показує вимірювання часу за взаємодією об'єктів (рис. 128). У нашому випадку фігурка представляє страхувальника (виконавця), який ініціює послідовність, хоча, у принципі, ця фігурка не є частиною діаграми послідовностей. Об'єкт «Оператор введення» одержує повідомлення від агента, після чого відбувається обробка одержаних даних. Під час роботи об'єкт «Оператора введення» одержує асинхронне повідомлення від об'єкта «Бухгалтер» (тобто об'єкт «Бухгалтер» не чекає відповіді від об'єкта «Оператор введення»). Після обробки даних об'єкт «Оператора введення» передає дані об'єкта «Бухгалтер» у вигляді синхронного повідомлення, тобто чекає відповідь від об'єкта «Бухгалтер», а саме йде уточнення з боку об'єкта «Бухгалтер» у вигляді умови: оброблені дані вірні або оброблені дані невірні. У випадку, коли дані невірні, йде доробка, а у випадку, коли дані вірні, відбувається відправлення повідомлення кінцевому користувачу, тобто групі кінцевих користувачів.

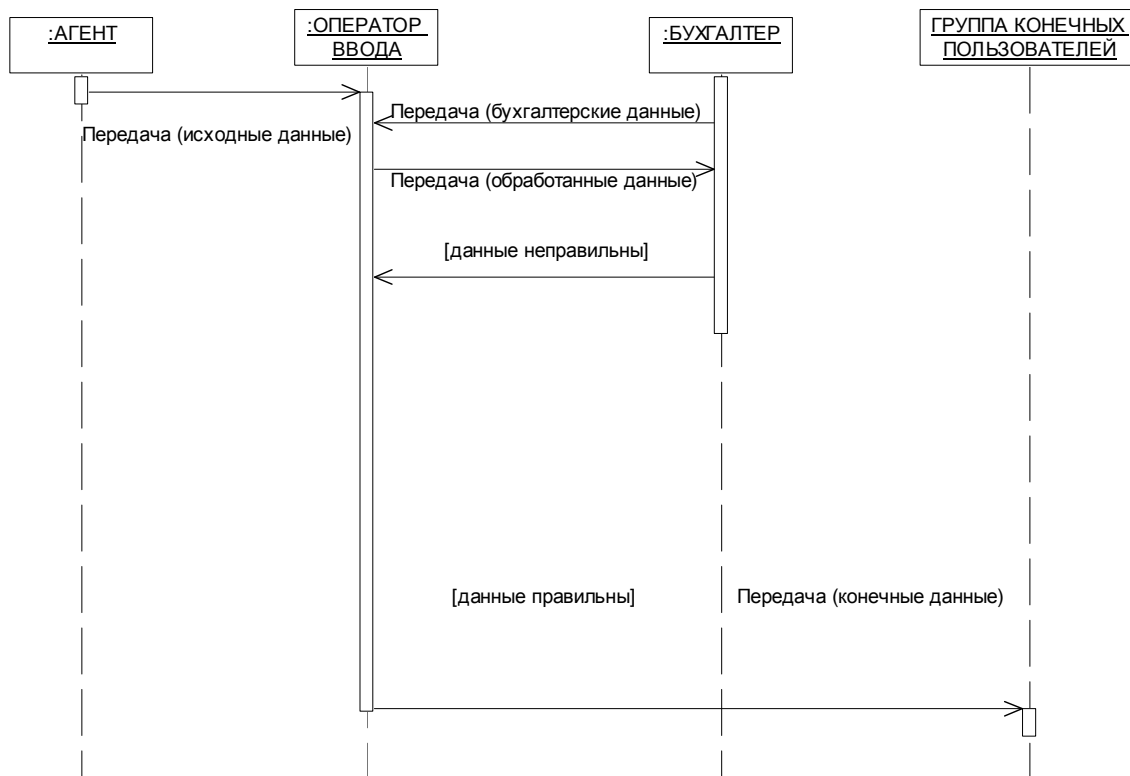


Рисунок 128 – Діаграма послідовності

Діаграма кооперації, так само, як і діаграма послідовностей, відображає взаємодію об'єктів. Але розбіжність цих двох діаграм полягає у тому, що діаграма послідовностей впорядкована відповідно до часу, а діаграма кооперації – відповідно до просторового розташування об'єктів, тобто орієнтована на стан і загальну організацію взаємодіючих об'єктів.

На рисунку 129 показана діаграма кооперації роботи страхової компанії. Діаграма описує послідовність інформації об'єкта, яка поступає у вигляді повідомлення. У нашому випадку деякі повідомлення є дочірніми щодо інших. Вони уявлені з використанням десяткової точки для позначення рівня вкладеності. Так, об'єкт «Оператора введення» може передати оброблені дані об'єкта «Бухгалтер» тільки після того, як одержить бухгалтерські дані від цього ж об'єкта. Оскільки без бухгалтерських даних неможливо сформулювати кінцеві дані. У свою чергу, кінцеві дані об'єкта «Група кінцевих користувачів» від об'єкта «Оператора введення» одержує тільки після передачі витікаючих даних від об'єкта «Бухгалтер».

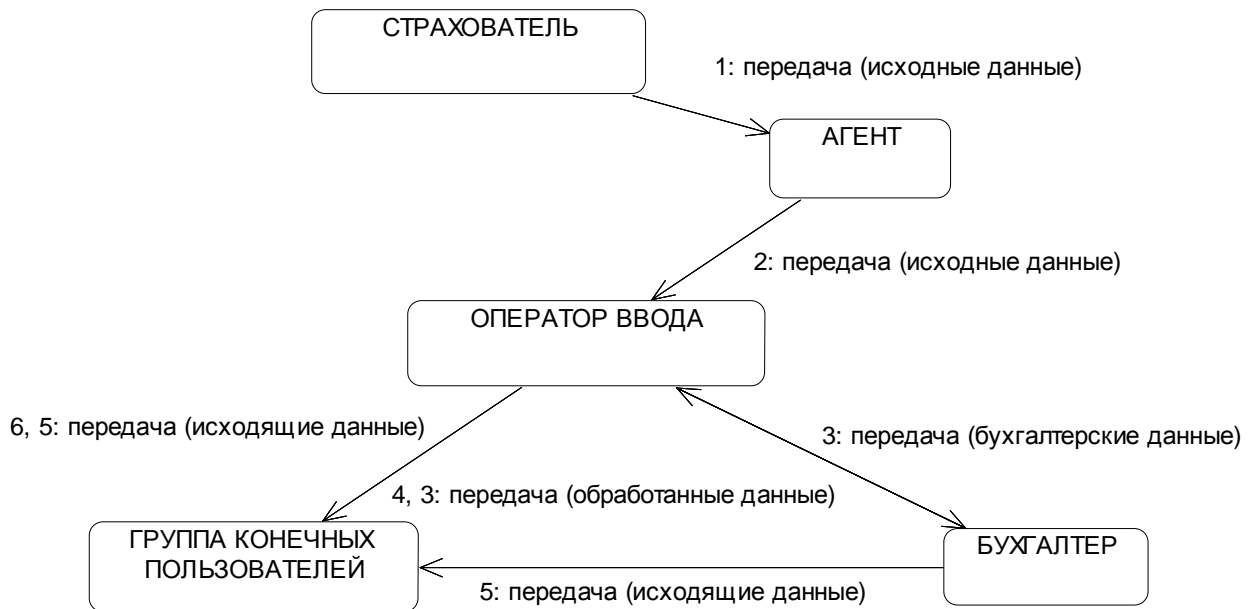


Рисунок 129 – Діаграма кооперації

Діаграма станів відображає, як об'єкти змінюють свій стан у відповідь на події, які відбуваються. При включенні комп'ютера відбувається завантаження. Тому включення комп'ютера є перемикаючою подією, яка приводить до переходу інтерфейсу в стан ініціалізації, а завантаження – дія, яка відбувається під час переходу. Результатом виконання дій у стані ініціалізації є виготовлення перемикаючої події, яка викликає перехід у стан роботи. При клацанні по кнопці завершення роботи здійснюється перехід у стан завершення роботи, і зрештою комп'ютер вимикається. Стан реєстрації нового страхового поліса в базі є підлеглим. Система знаходиться в змозі «Очікування введення даних», поки оператор введення не введе відповідну умову. Далі активізуються стани «Обробка даних» і «Збереження даних». Підсумковий стан – «Візуалізація результатів».

Діаграма діяльності реально описує все, що відбувається під час операції або у процесі. У нашому випадку діаграма видів діяльності розбивається на «плавальні доріжки» – паралельні сегменти, які відповідають ролям (рис. 130).

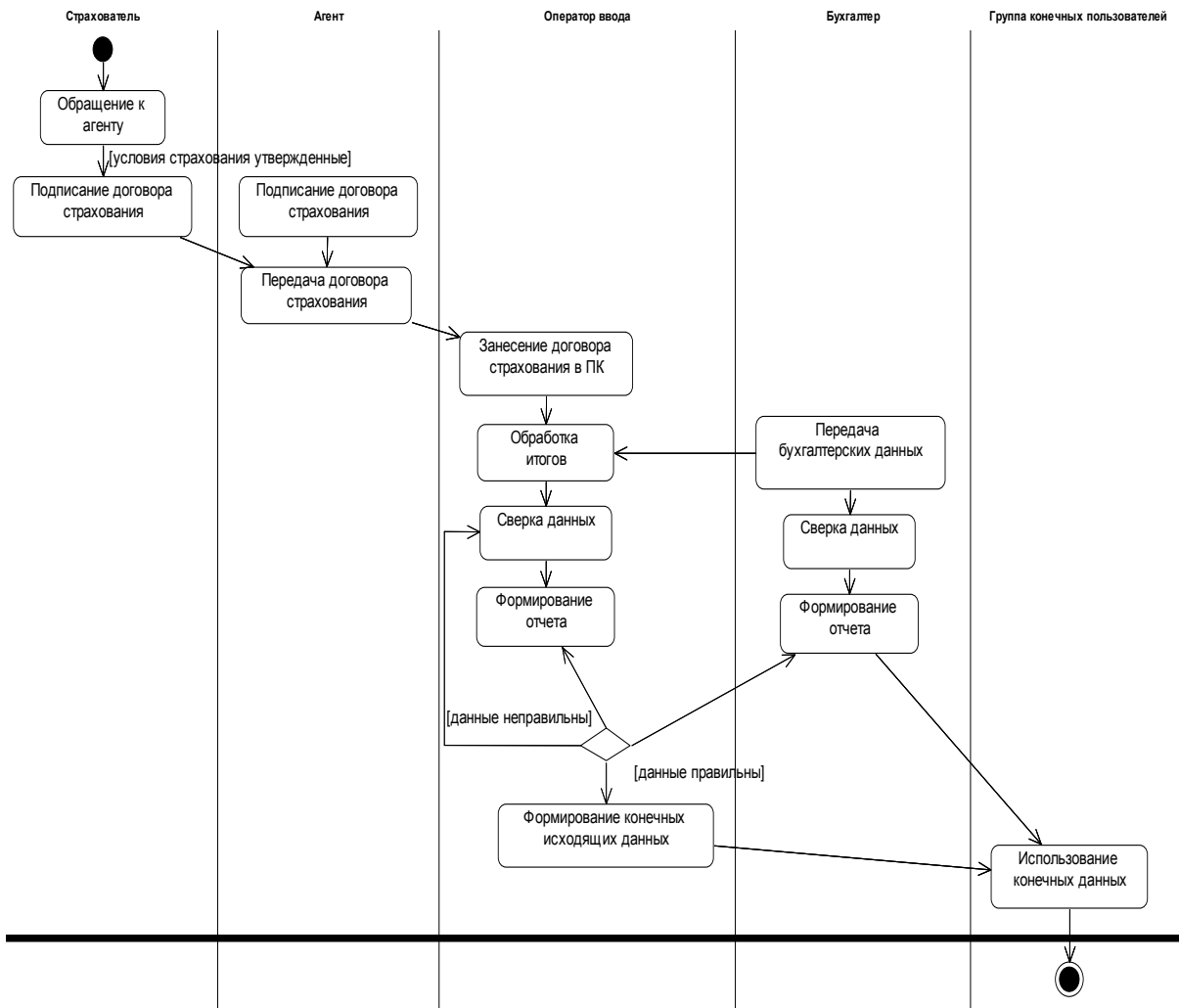


Рисунок 130 – Діаграма діяльності

На діаграмі компонентів у нашому випадку для роботи з даними необхідний компонент «Таблиця», а для подання даних таблиці – інтерфейс «Робота з даними». У свою чергу, робота з даними може містити перегляд даних, реєстрацію нових даних, редагування даних, формування звіту за заданою умовою. Зазначимо наступні інтерфейси: «Перегляд», «Редагування», «Вибір», «Автоматичний підбір» і «Звіт» (рис. 131).

Система була реалізована у середовищі програмування Borland Delphi і упроваджена у відділенні страхової компанії [22-25]. Проведений розрахунок економічної ефективності показав, що термін окупності системи складе 0,15 року.

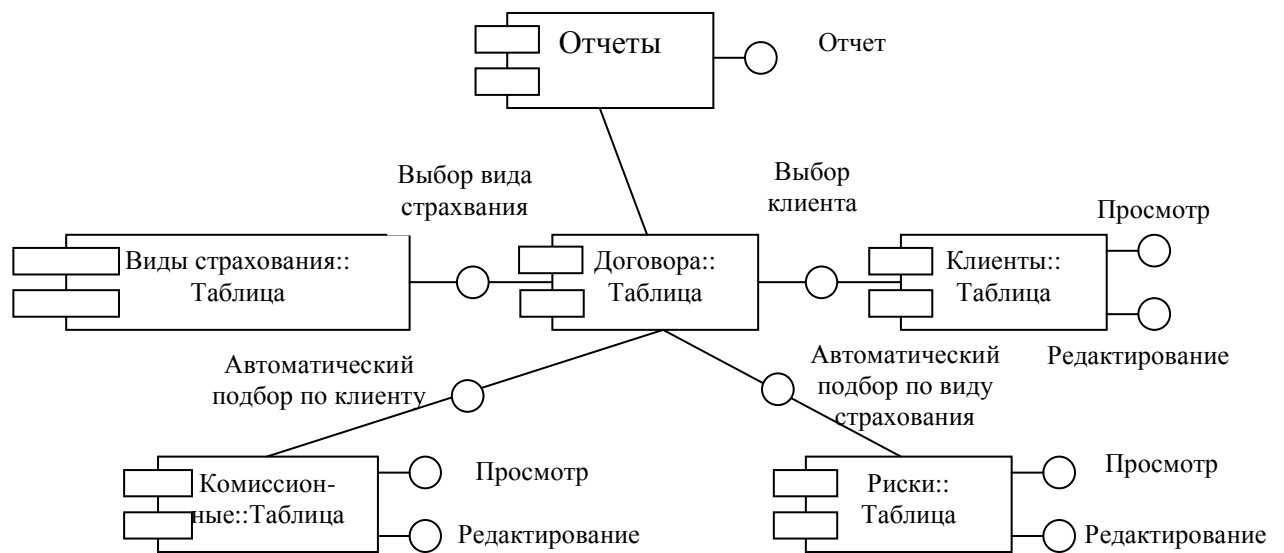


Рисунок 131 – Діаграма компонентів

4.5 Інформаційна система для забезпечення функціонування фінансового відділу підприємства

На чолі фінансового відділу даного підприємства стоїть фінансовий директор, що здійснює управління і контроль над діяльністю служби у цілому. Фінансовий відділ складається з 12 осіб і підрозділяється на наступні сегменти: технічна бухгалтерія, група з податкового обліку, група з управлінського обліку, аналізу і бюджетування. Безпосередній контроль над правильністю відображення господарських операцій і формування фінансової і податкової звітності здійснює внутрішній аудитор. Для ефективного управління витратами і формування бюджету на підприємстві формуються центри фінансової відповідальності.

Фінансовий відділ виконує наступні поставлені задачі: організація фінансової діяльності підприємства, направленої на забезпечення фінансовими ресурсами завдань плану, збереження і ефективного використання основних фондів і оборотних коштів, трудових і фінансових ресурсів підприємства, своєчасності платежів за зобов'язаннями до державного бюджету, постачальникам і установам банків.

У зв'язку з цим йде розбиття функцій фінансового відділу на три сфери: фінансово-кредитного планування, фінансово-оперативної роботи і

сфери контрольно-аналітичної роботи, кожна з яких покладається на одну з трьох груп відділу. Так, група з технічної бухгалтерії виконує 19 функцій зі сфери фінансово-кредитного планування, група з податкового обліку виконує 13 функцій у сфері фінансово-оперативної роботи, група з управлінського обліку, аналізу і бюджетуванню виконує 8 функцій у сфері контрольно-аналітичної роботи.

Першим етапом процесу об'єктно-орієнтованого аналізу і проектування є побудова діаграми варіантів використання (рис. 132).

З діаграми виходить, що користувачами системи будуть три актори – це три групи, на які поділяється фінансовий відділ ЗАТ «ГД» – технічна бухгалтерія, група з податкового обліку і група з управлінського обліку. Розглянемо детально варіанти використання кожного актора для даної системи. Відповідно до діаграми варіантів використання актор «Технічна бухгалтерія» відповідає за роботу з первинною документацією, тобто отримання даних з виробництва, їх детальний розгляд, введення щоденної бухгалтерської документації. Актор «Група з податкового обліку» займатиметься детальною обробкою даних: аналізувати бухгалтерські дані, перевіряти їх правильність, формувати звіти для державних і внутрішніх служб. Актор «Група з управлінського обліку» займається виключно вихідною інформацією, її аналізом, обробкою, підведенням підсумків і формуванням висновків з діяльності відділу в цілому.

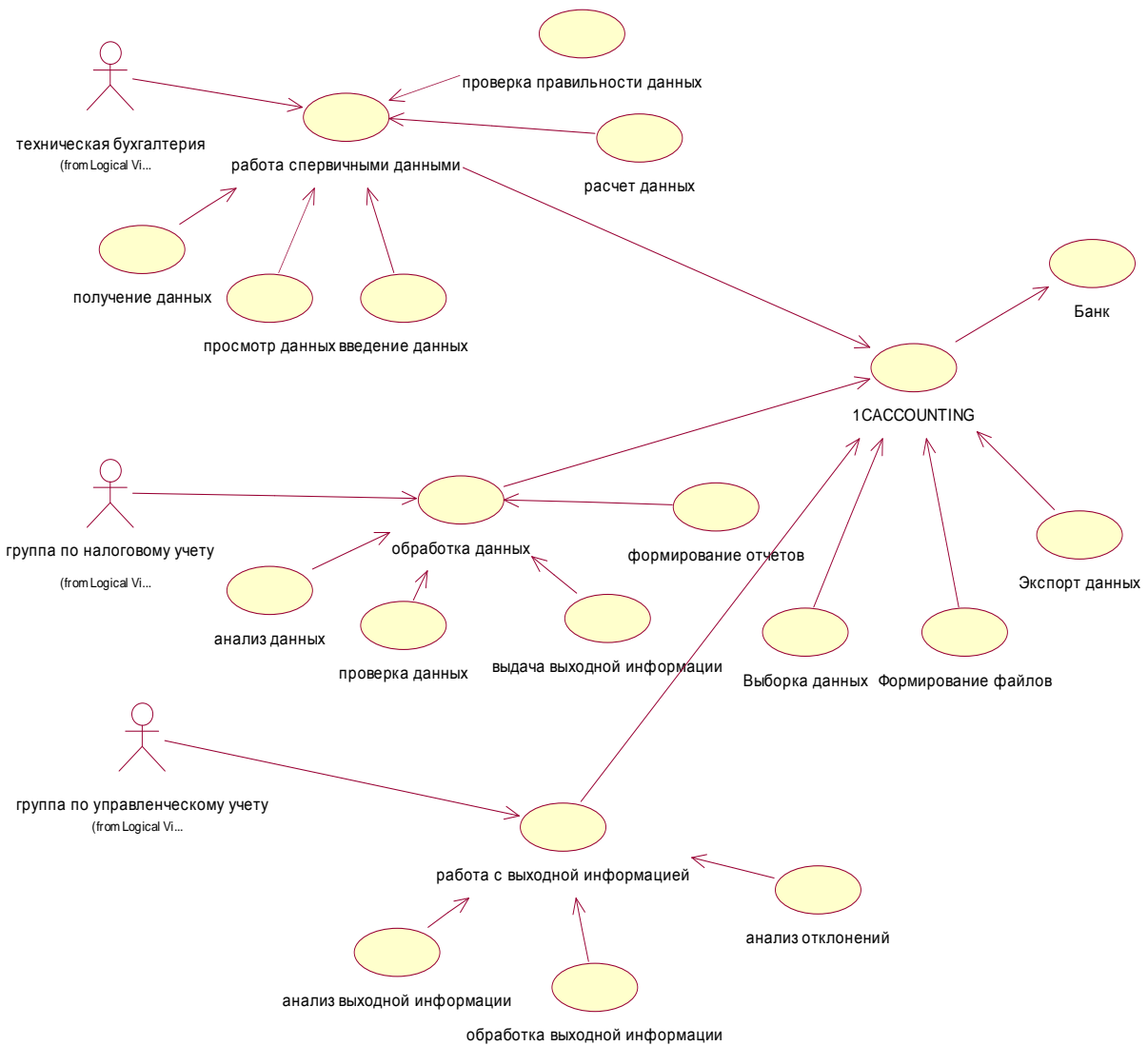


Рисунок 132 – Діаграма варіантів використання

Для подання статичної структури моделі системи у термінології класів об'єктно-орієнтованого програмування служить діаграма класів (рис. 133). На ній зображені наступні класи: первинні дані, технічна бухгалтерія, група з податкового обліку, група з управлінського обліку, вихідна інформація, банк, податкова інспекція, комітет управління. Відношенням залежності вказано, що класи «Технічна бухгалтерія» і «Група з податкового обліку» залежать від класу «Первинні дані». Таким же чином клас «Вихідна інформація» виявляється залежним від класів «Технічна бухгалтерія» і «Групи з податкового обліку», оскільки ці два класи формують звітність для внутрішніх і державних служб обліку і аудиту. Далі з діаграми виходить бінарне відношення асоціації від класу «Вихідна інформація» до класів «Банк», «Податкова інспекція» і до «Групи з управлінського обліку». Між «Групою з управлінського обліку» і «Комітетом управління»

встановлюється відношення залежності, оскільки дана група підзвітна фінансовому директору, генеральному директору, а також акціонерам, які й складають комітет управління.

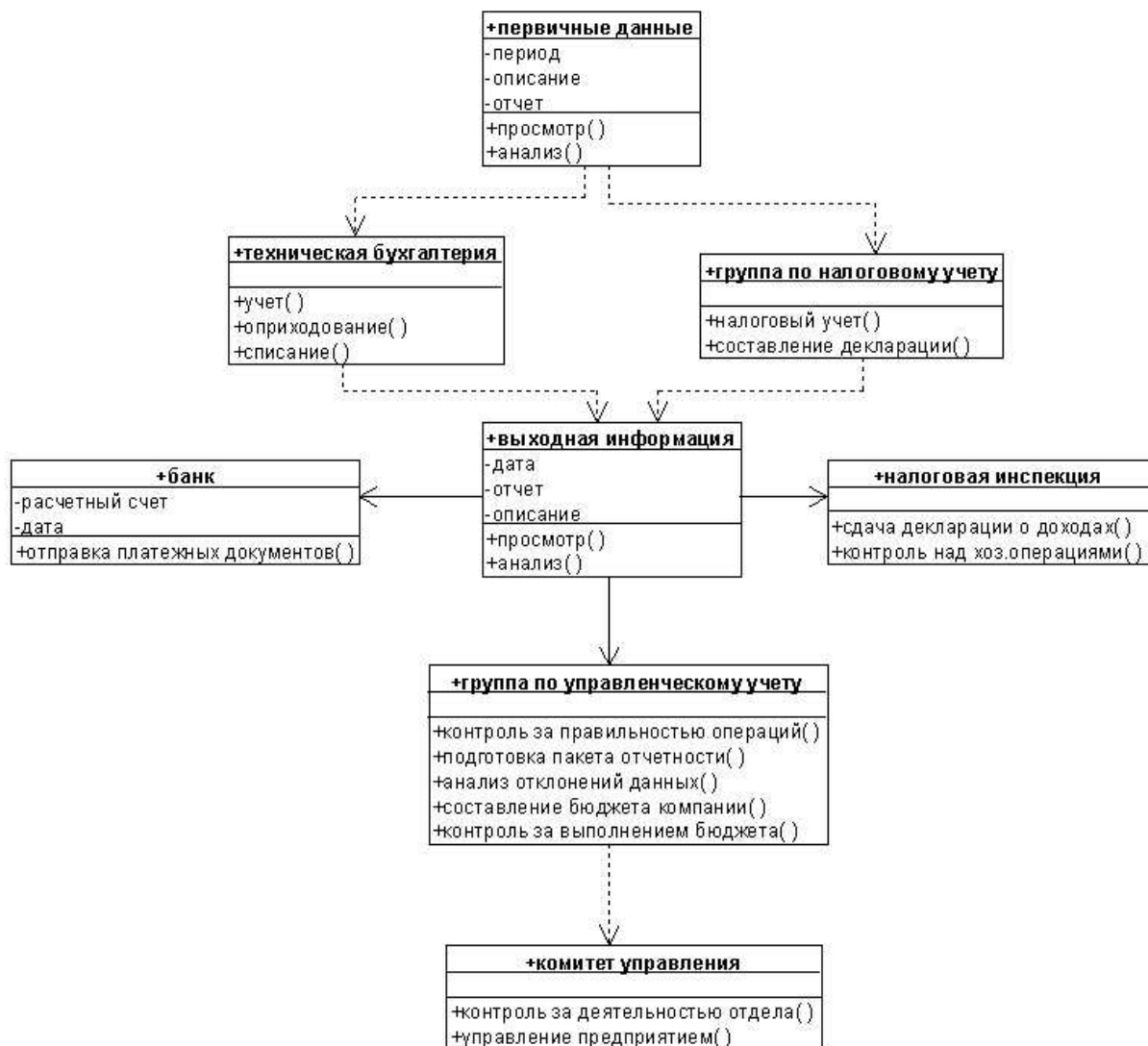


Рисунок 133 – Діаграма класів

Для розгляду взаємодії об'єктів у контексті статичної структури моделі і для подання структурних особливостей передачі та прийому повідомлень між об'єктами використовується діаграма кооперації (рис. 134). Робота у системі здійснюється у наступному порядку: первинні дані, надходячи до фінансового відділу, враховуються технічною бухгалтерією, частина з них обробляється для звіту, інша надходить до групи з податкового обліку, де формується звітність для податкової служби. З технічної бухгалтерії і групи з податкового обліку дані поступають до групи з управлінського обліку, де відбувається підготовка пакету звітності на основі одержаних даних і складається бюджет компанії.

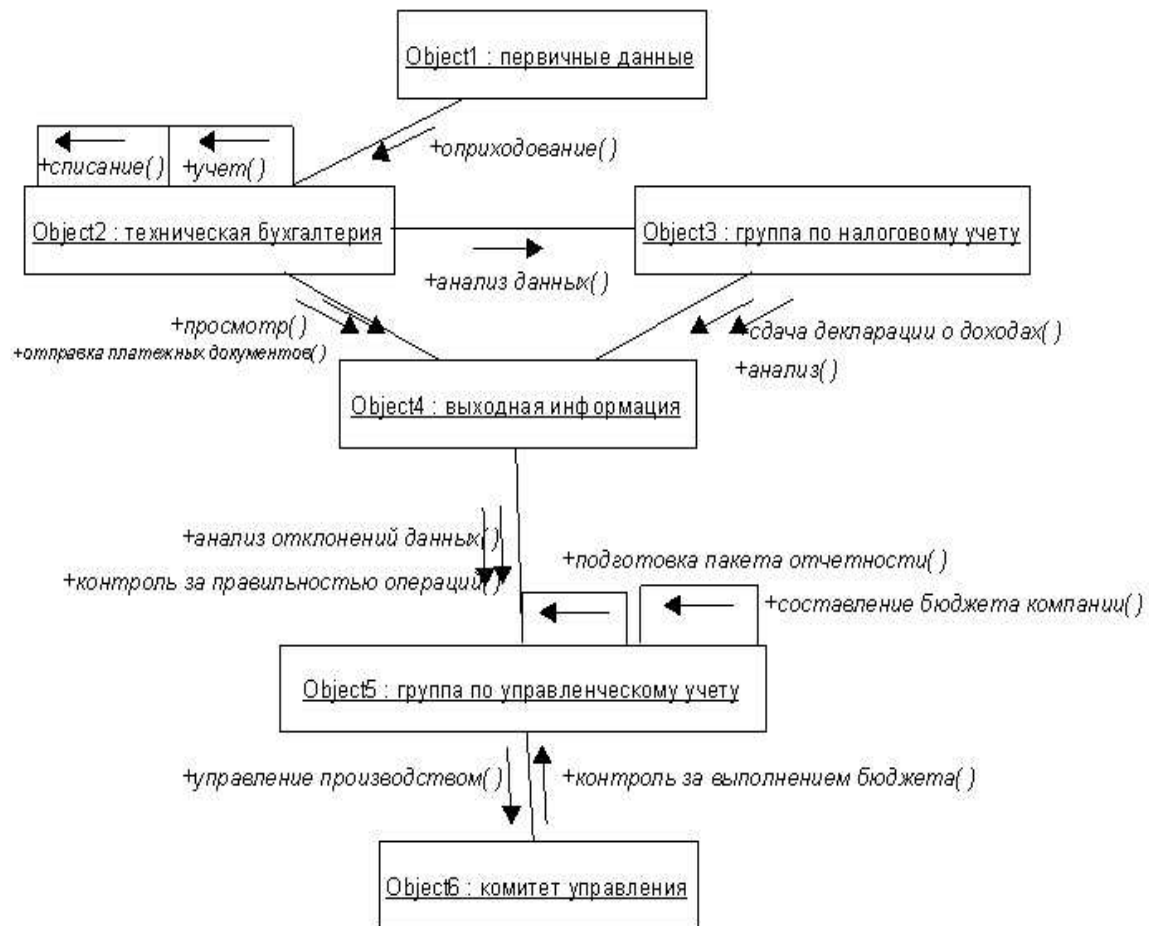


Рисунок 134 – Діаграма кооперації

Клас «Комітет управління» здійснює контроль над виконанням бюджету і вносить зміни в управління виробництвом безпосередньо через зв'язок з класом «Група з управлінського обліку».

Для моделювання взаємодії об'єктів у часі використовується діаграма послідовності (рис. 135). На ній видно, що ініціатором взаємодії є саме виробництво, яке є джерелом первинних даних і яке після їх обробки у фінансовому відділі підпадає під вплив комітету управління.

Діаграма станів описує всі можливі стани одного екземпляра певного класу або всієї системи і можливі послідовності його (її) переходів з одного стану в інше. Діаграма станів роботи всієї системи показана на рис. 136.

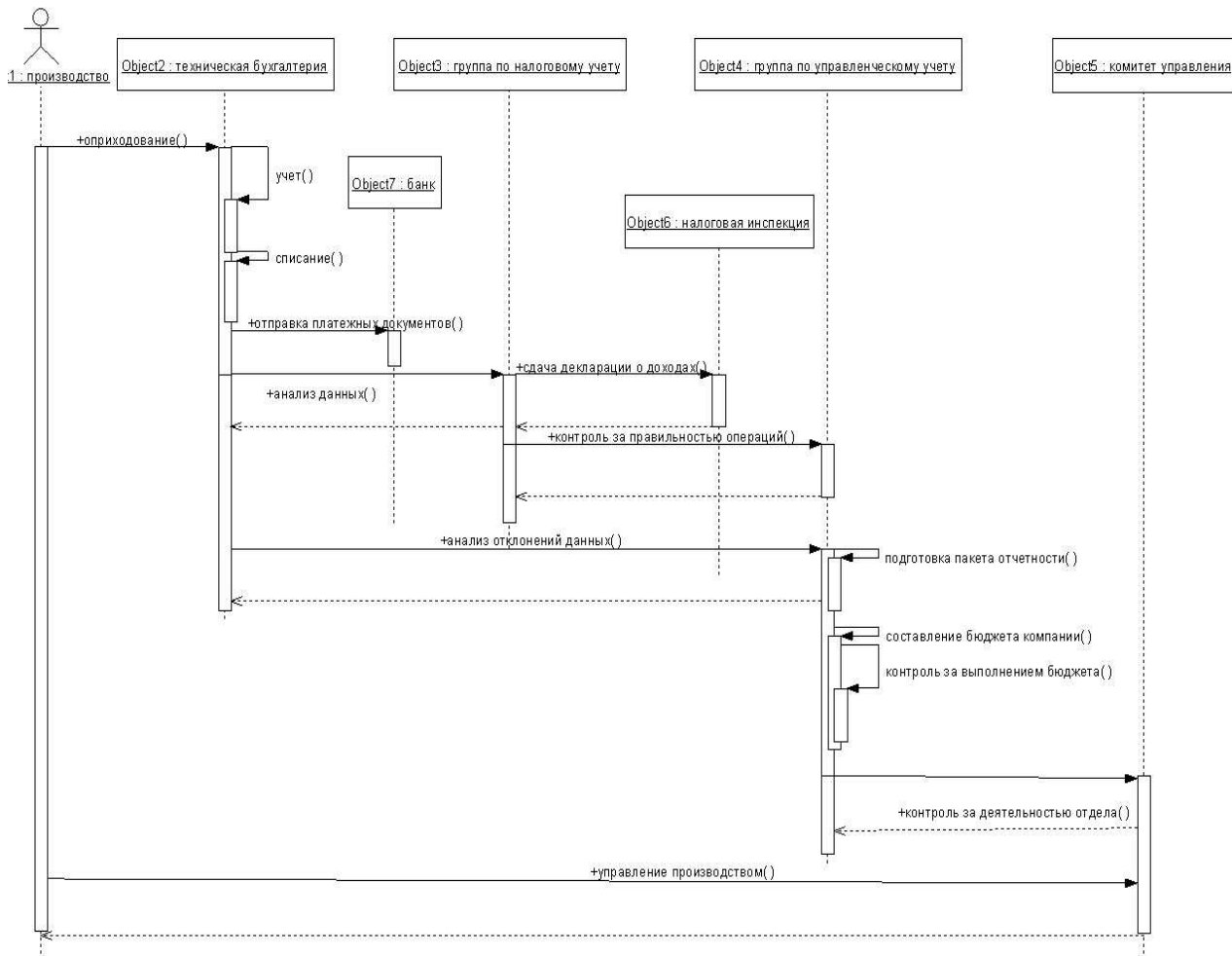


Рисунок 135 – Диаграмма последовательности



Рисунок 136 – Диаграмма станів роботи системи

Для моделювання процесу виконання операцій використовується діаграма діяльності (рис. 137). На ній показано, що після формування початкових даних на виробництві вони детально обробляються у всіх трьох групах фінансового відділу. Після чого оброблені дані надходять до комітету управління, де ведеться повний контроль над діяльністю не тільки відділу, але і всього підприємства в цілому.

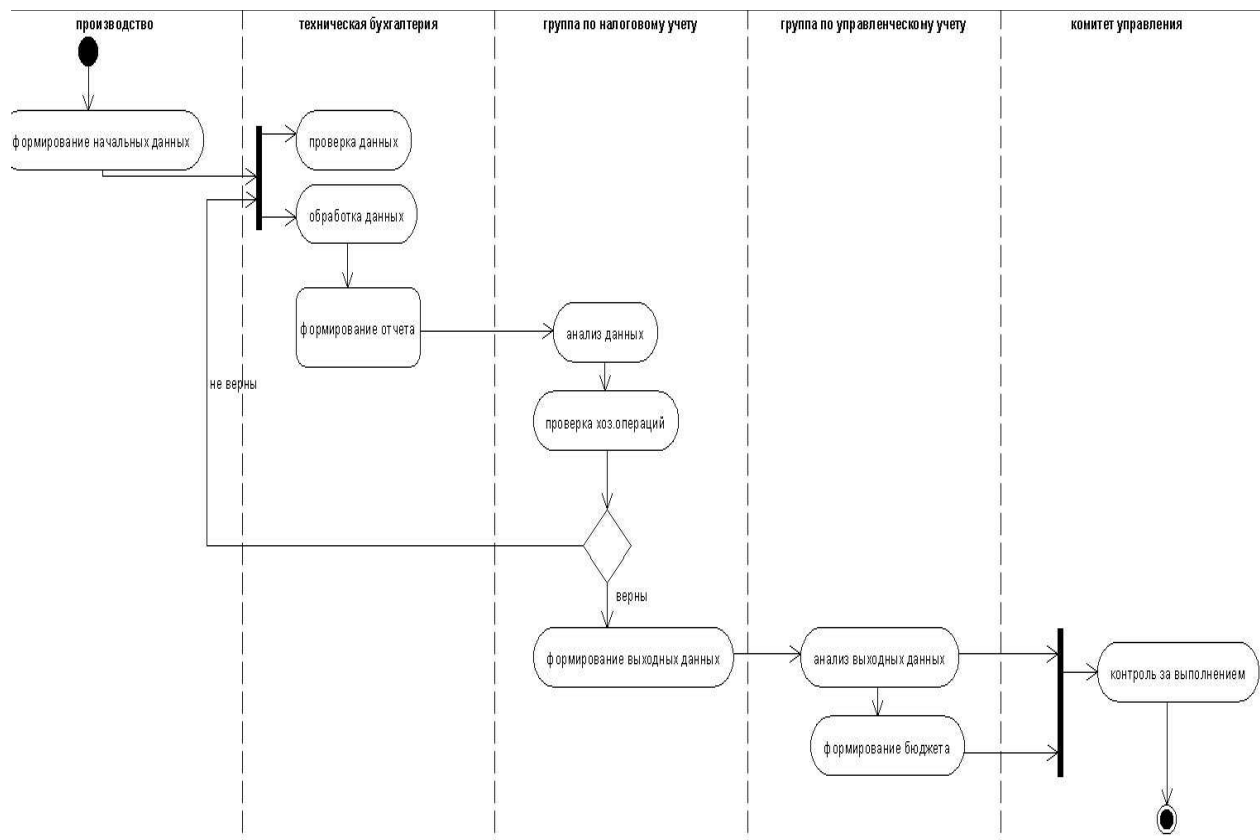


Рисунок 137 – Діаграма діяльності

Задача розробки програмного комплексу не ставилася, тому діаграми реалізації не склалися.

4.6 Інформаційна система для розрахунку собівартості металопродукції

Собівартість продукції залежить від умов виробництва і реалізації продукції. Істотно впливають на рівень витрат роблять техніко-економічні чинники виробництва – зміни у техніці, технології, організації виробництва, у структурі та якості продукції і величині витрат на її виробництво. На будь-якому виробництві важливо правильно спланувати очікувані доходи, врахувати всі витрати фірми і правильно віднести їх на собівартість продукції. Правильно спроектована система розрахунку калькуляції може зробити значний вплив на процес формування собівартості.

Така система повинна мати грамотну структуру, дозволяти оптимізувати інформаційні потоки, відсівати непотрібну інформацію, спрощувати пошук і отримання необхідної інформації. Автоматизація, упроваджувана штучним шляхом у природні інформаційні потоки, буде ефективна тільки

тоді, коли відбудеться успішна інтеграція автоматизованої інформаційної системи до структури підприємства.

На ВАТ «СКМЗ» активно підтримується упровадження автоматизації у виробничі, управлінські, обчислювальні й інформаційно-аналітичні процеси. Підприємством виділяються кошти на дослідження сфер, які підлягають автоматизації. У середині 90-х років була розроблена і упроваджена у планово-економічний відділ інформаційна система для складання калькуляції і розрахунку собівартості металопродукції. Дана програма істотно понизила трудомісткість робіт і собівартість виконання операцій. Проте розвиток комп'ютерних технологій вимагає від будь-якої інформаційної системи удосконалення: перехід з MS-DOS на Windows утруднив функціонування програми, оскільки вона не була призначена для роботи у нових операційних системах. Вона стала видавати помилки, що іноді супроводжуються втратами даних. Отже з'явилася необхідність у створенні новішої і надійнішої версії програми, яка змогла б забезпечити надійність у роботі.

Створення системи починається з проектування її моделі на мові UML. Спочатку необхідно чітко визначити вимоги, яким повинна відповідати система, що розробляється, і які задачі вона повинна вирішувати. Система, що розробляється, повинна:

- а) реалізовувати функції введення, модифікації і перегляду даних;
- б) виконувати основну функцію – розрахунок собівартості;
- в) реалізовувати функцію складання звітності одержаних даних;
- г) надавати простий зручний інтерфейс для роботи з даними;
- д) мати нагоду збереження і редагування даних;
- е) мати захист.

Діаграма варіантів використання подана на рисунку 138.

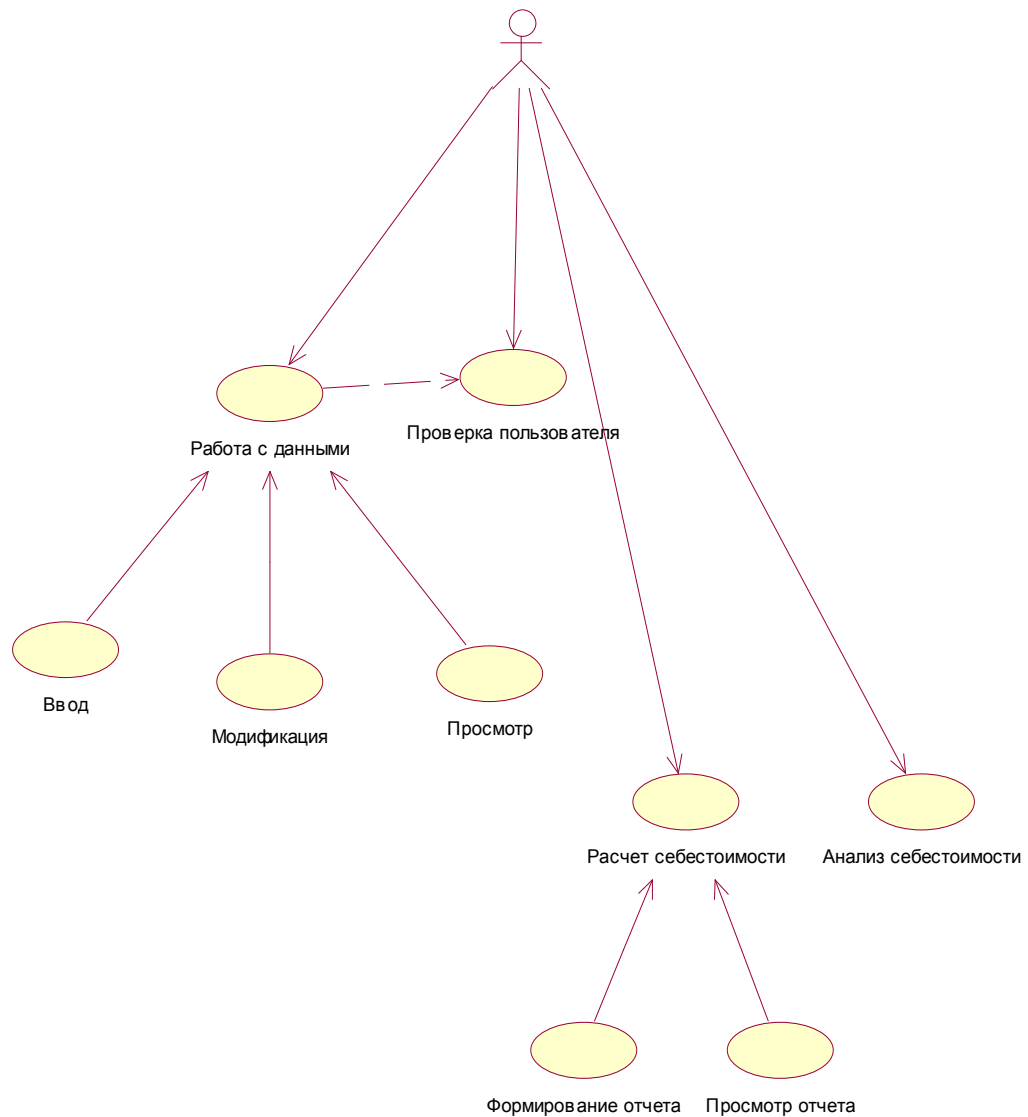


Рисунок 138 – Діаграма варіантів використання

Безпосереднім користувачем програми у даному випадку є фахівець планово-економічного відділу. На діаграмі видно, що передбачається наявність таких варіантів використання системи: введення даних, складання калькуляції розрахунок і аналіз собівартості, експорт одержаних даних. Робота з початковими даними припускає також їх перегляд і модифікацію.

Аналіз структури і функціонування системи проводиться за допомогою діаграм класів і поведінки. Діаграма класів для розрахунку собівартості металопродукції подана на рисунку 139.

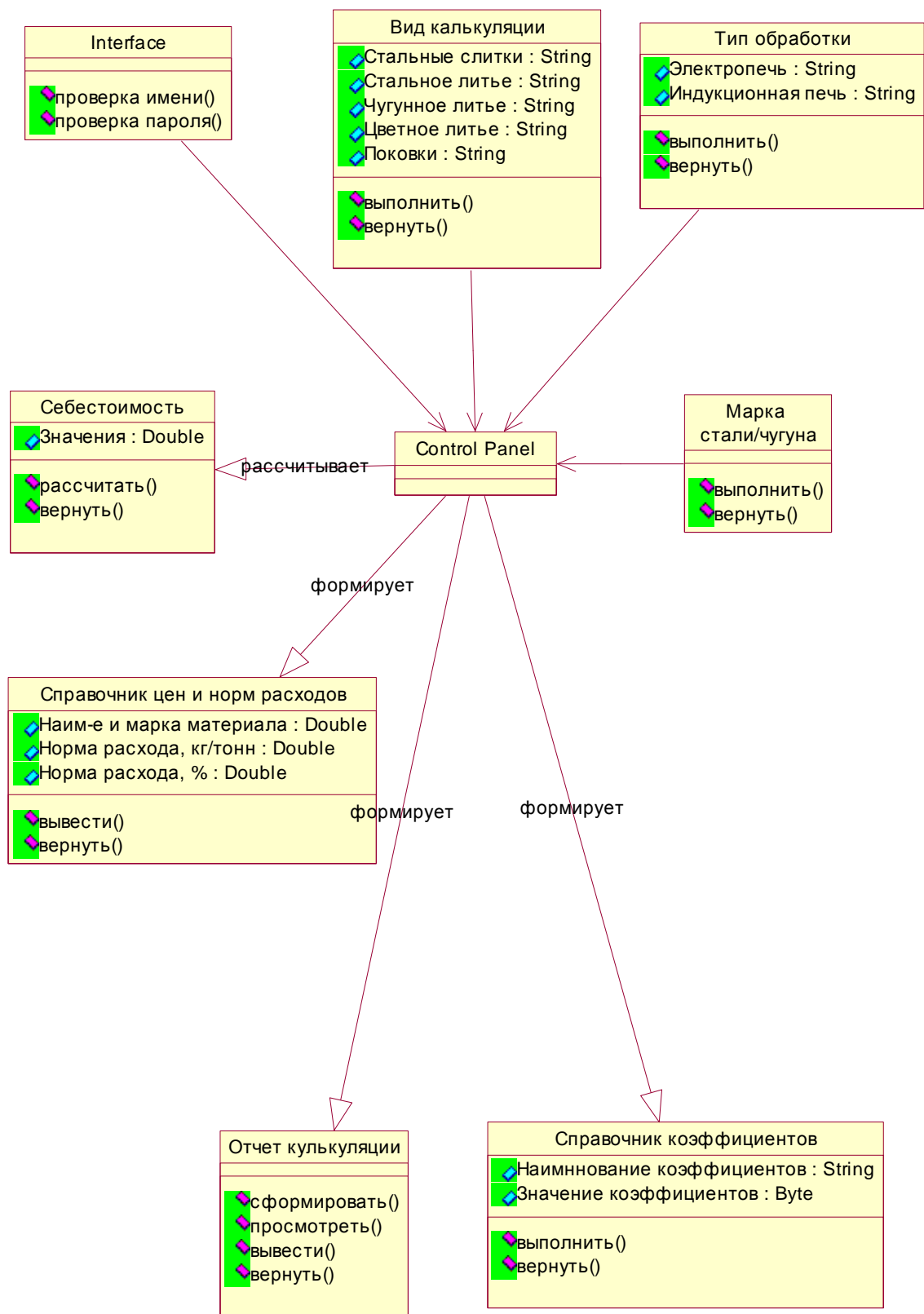


Рисунок 139 – Диаграмма классов

На цій діаграмі зображені наступні класи: interface, вид калькуляції, тип обробки, вигляд марки сталі, довідник коефіцієнтів, звіт калькуляції, довідник цін і норм витрат, собівартість, control panel. Усі операції класів характеризуються областю видимості типу «загальнодоступний» (private), тобто їх видно, і вони доступні з будь-якого іншого класу. Control panel є управляючим класом і відповідає за функціонування інших класів. Стрілки з трикутниками на схемі позначають наявність відносин і бінарного зв'язку між класами. Бінарний зв'язок означає, що взаємодія між класами вільна.

До діаграм поведінки відносяться діаграми кооперацій, послідовностей, стану і діяльності. Діаграма кооперації подана на рисунку 140. На ній подані такі класи: interface, вид калькуляції, обробка, марка сталі/чавуну, довідник коефіцієнтів, довідник цін і норм витрат, звіт про калькуляцію, собівартість. Робота системи має певний порядок. Економіст починає роботу вікном управління, заздалегідь ввівши своє ім'я і пароль. Потім вводить всі необхідні дані для розрахунку собівартості шляхом вибору виду калькуляції, типу обробки, мазкі стали або чавуну. Після, через вікно управління, дає команду на формування довідників коефіцієнтів, цін і норм витрат, створює звіт по калькуляції. Завершальним етапом є розрахунок собівартості металопродукції.

Для моделювання взаємодії об'єктів за часом мовою UML використовуються діаграми послідовності (рис. 141). На діаграмі показано, що ініціатором взаємодії є економіст-користувач, який веде безпосереднє управління і контроль над системою на всьому протязі її роботи. Поступово управління переноситься на вікно управління, яке збирає необхідну інформацію з одних класів (Interface, Вид калькуляції, Обробка, Марка чавуну/сталі) і дає команди виконання іншим класам (Довідник коефіцієнтів, Довідник цін і норм витрат, Звіт калькуляції, Собівартість). У кінці роботи створюються необхідні звіти, і розраховується калькуляція, після чого управління передається економісту.

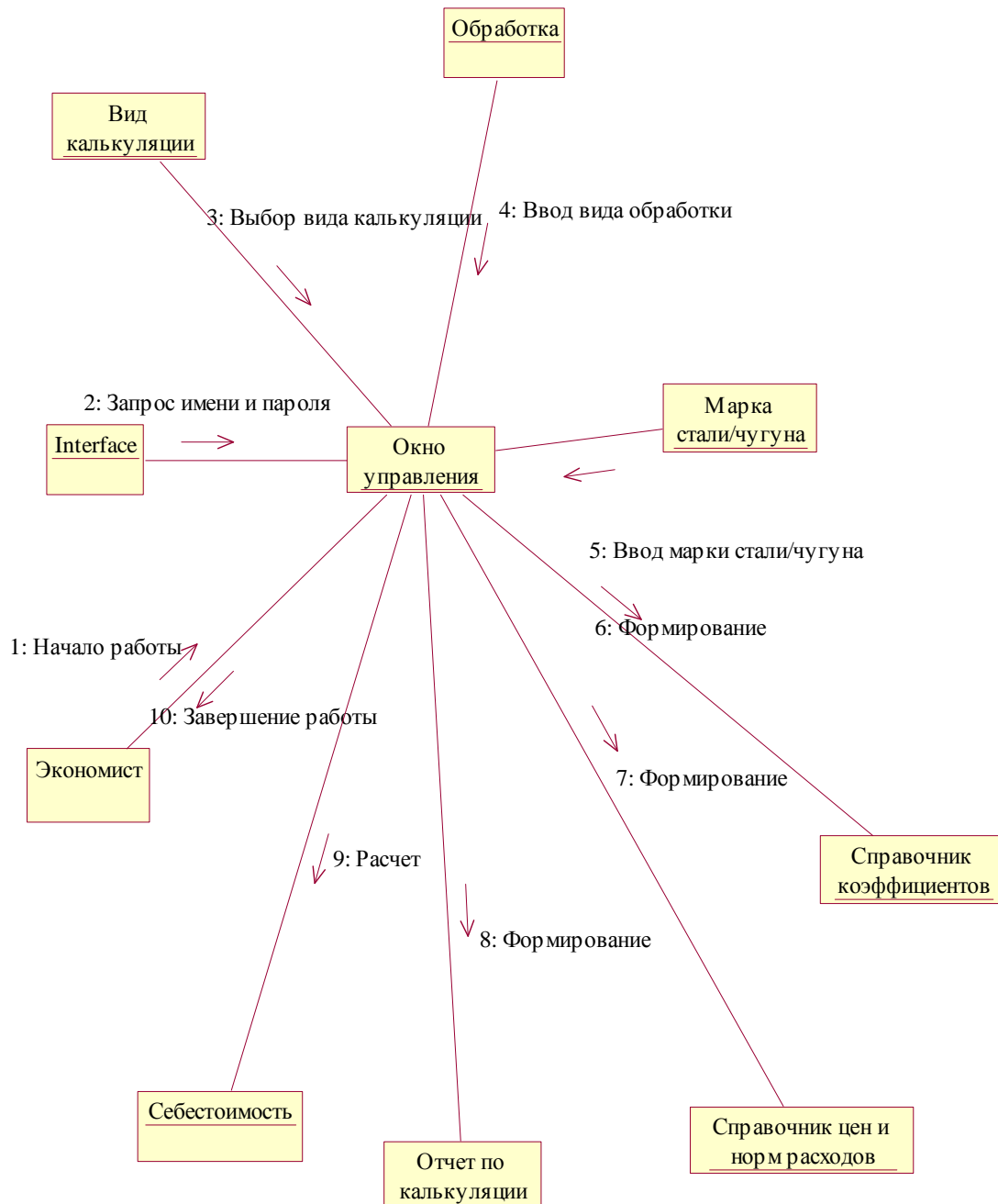


Рисунок 140 – Діаграма кооперації

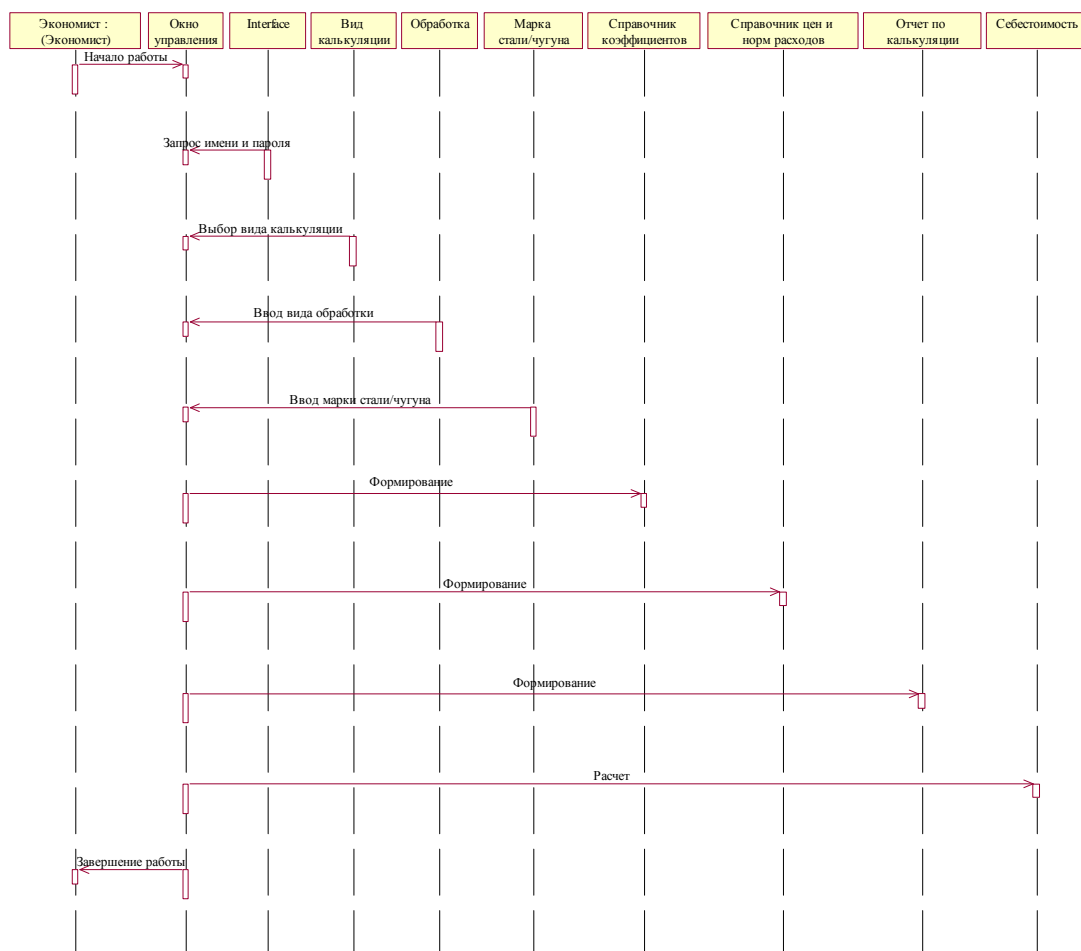


Рисунок 141 – Діаграма послідовності

Функціонування системи характеризують діаграми станів (рис. 142) і діяльності (рис. 143). Роботу системи можна описати так: після обігу економіста-користувача система ідентифікує його шляхом перевірки імені та пароля; якщо перевірка не пройшла успішно, передбачається вихід з системи; у зворотному випадку економіст дістає доступ до головного вікна управління. Подальша робота полягає у виборі виду калькуляції (на головному меню), типу обробки, вибору марки сталі/чавуну. Після цього передбачається сформувати і співвіднести на калькуляцію довідник цін і норм витрат, довідник коефіцієнтів, виконати розрахунок собівартості і потім виконати збереження розрахунків у файл.

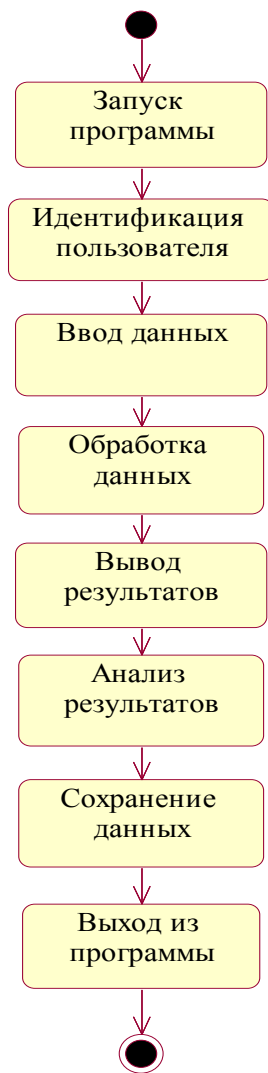


Рисунок 142 – Діаграма станів

Для фізичного подання моделей систем використовуються діаграми компонентів і розгортання.

Діаграма компонентів (рис. 144) показує, що програмна система складається з таких фізичних частин: exe, pas, csv, dbf і xls. Компонента проекту Project1.exe виконує роль основи системи і забезпечує взаємодію всіх компонентів, файли pas містять програмний код, файли dbf – інформаційні дані.

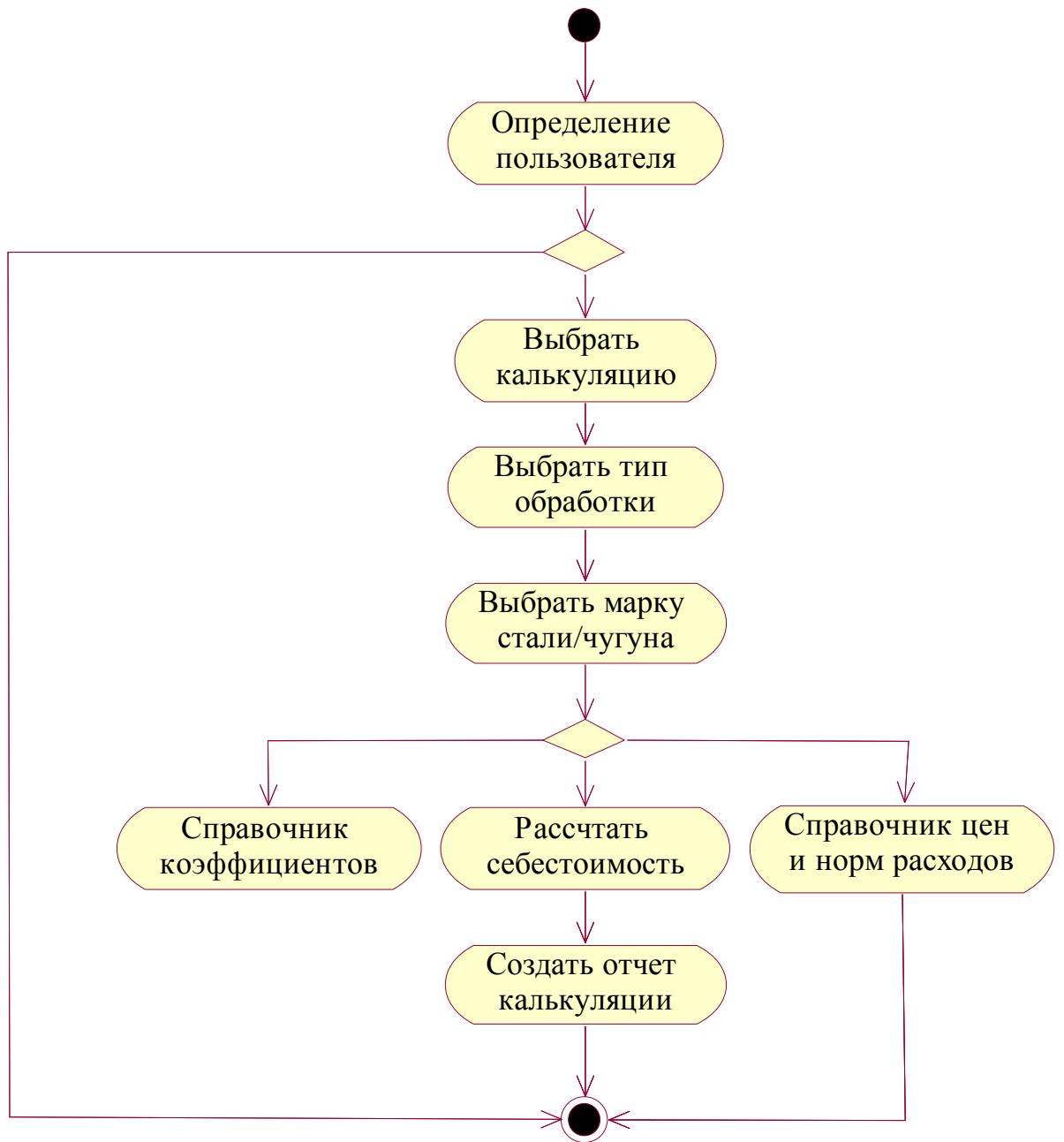


Рисунок 143 – Діаграма діяльності

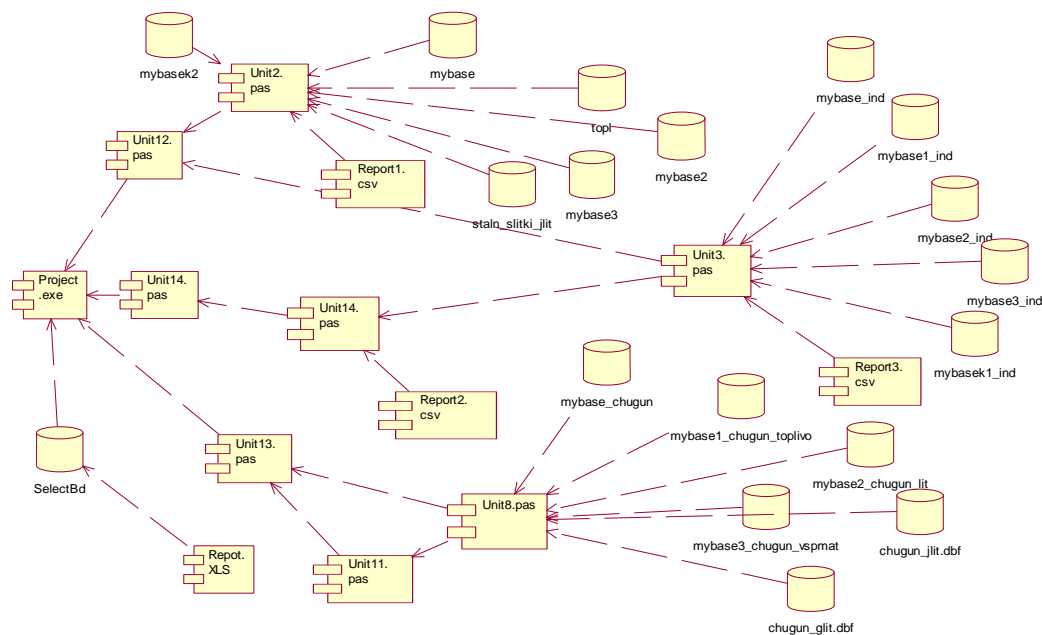


Рисунок 144 – Діаграма компонентів

Діаграма розгортання містить графічні зображення процесорів, пристроїв, процесів і зв'язків між ними. Оскільки в нашому випадку розробляється програма, яка не використовуватиме периферійні пристрої і ресурси (її функціонування обмежене на комп'ютері користувача), тому необхідність в розробці цієї діаграми відсутня.

Створена модель була реалізована в середовищі Borland-Delphi 6.0 (програма «Калькуляція 1.1») і використовується на підприємстві для розрахунку собівартості металопродукції, яку воно виробляє (сталеве литво, сталеві злитки, чавунне литво, кольорове литво, поковки). Програма працює в операційних системах Windows XP/2000/ME, характеризується хорошою продуктивністю і потребою в мінімальних системних вимогах.

Був розрахований річний економічний ефект від упровадження системи: він склав 7711,8 грн. На одну гривну одноразових капіталовкладень величина річного приросту прибутку складає 0,9 грн. Термін окупності склав 1 рік і 1 місяць [26-27].

4.7 Інформаційна система для обліку і контролю готової продукції

Основними задачами обліку готової продукції на підприємстві є: своєчасне оформлення відповідними документами готової продукції, випущеної на виробництві; забезпечення контролю за її збереженням на складах підприємства; своєчасне віддзеркалення операцій з відвантаження і реалізації готової продукції і розрахунків з покупцями; забезпечення контролю за виконанням плану випуску і реалізації продукції. Для підвищення точності розрахунків і зниження трудомісткості такої діяльності її необхідно автоматизувати.

Проблема автоматизації обліку і контролю готової продукції гостро стоїть на багатьох підприємствах України, зокрема на КиАЗ «Авіант». Існує декілька шляхів рішення проблеми: а) проводити розрахунки і складати документацію уручну з використанням засобів Microsoft Office; б) використовувати розроблені раніше програмні продукти; у) використовувати придбані програмні продукти; г) розробляти власні програмні продукти для вирішення поставлених задач.

Проведення розрахунків і складання документації вручну призведуть до великих витрат праці та часу працівників, що виконують облік готової продукції і напівфабрикатів. Використання корпоративної інформаційної системи (наприклад, «1С-предприятие») не може розв'язати проблему: навіть розрахованої на багато користувачів версії «1С» недостатньо для такого підприємства, як КиАЗ «Авіант»; для вирішення проблеми масштабованості доведеться купувати декілька ліцензій, а це призведе до великих матеріальних витрат.

На початку 90-х років на Київському авіаційному заводі «Авіант» співробітниками відділу автоматизованих систем управління виробництвом для вирішення задачі обліку і контролю готової продукції був розроблений програмний продукт, який називається «ПІЗи». У даний час ця програма вимагає значної доробки – переведення на нову обчислювальну платформу, забезпечення розрахованого на багато користувачів мережевого режиму, приведення інтерфейсу до сучасних стандартів і багато що інше. Очевидно, що доцільно розробити нову програмну систему.

Проектування системи починається з побудови концептуальної моделі – діаграми варіантів використання («прецедентів»). Така діаграма для системи обліку і контролю наведена на рисунку 145.

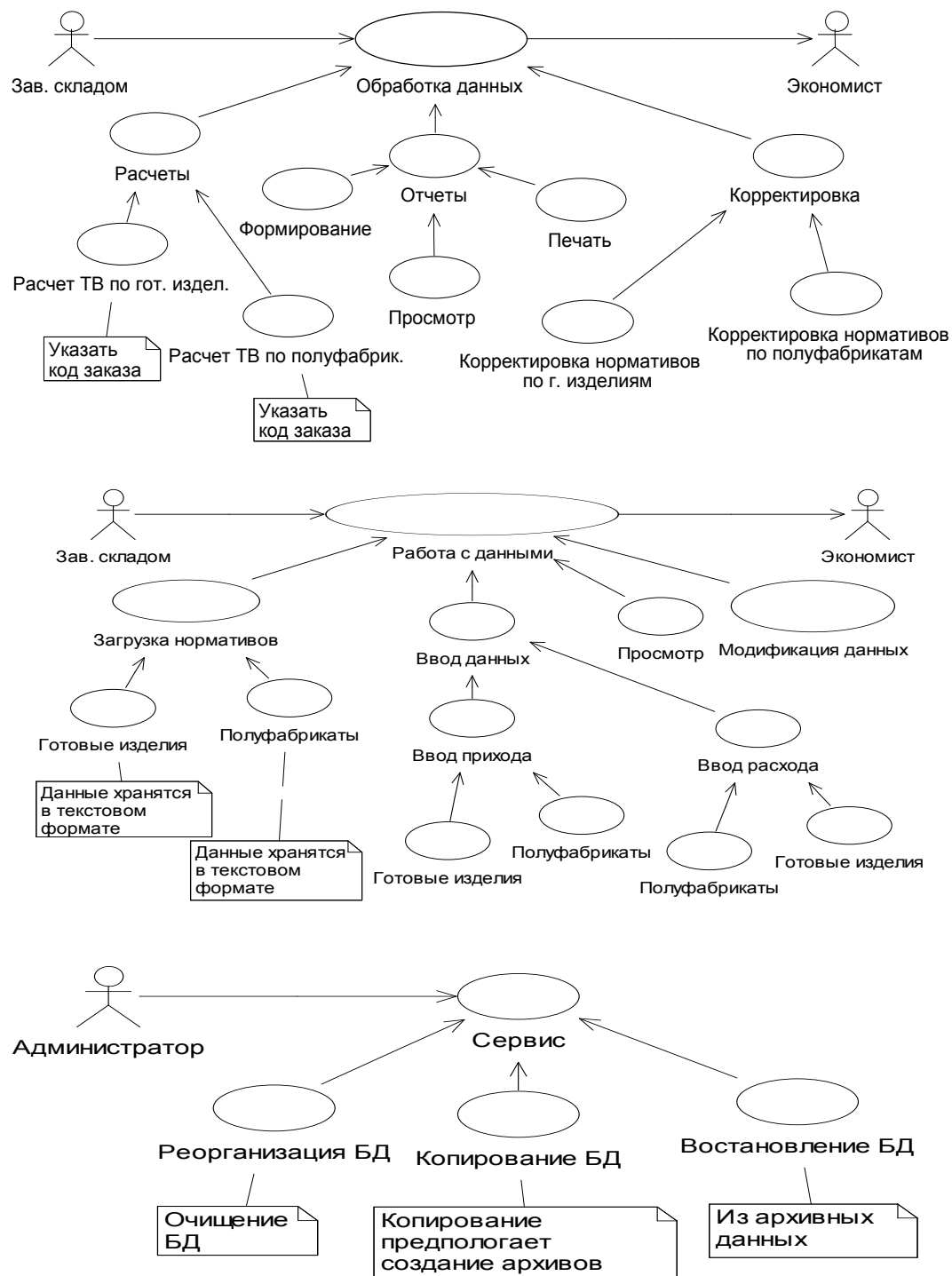


Рисунок 145 – Діаграма варіантів використання

Центральну діаграму логічної моделі системи – діаграму класів – для зручності відображення розподілимо на дві: діаграму асоціацій класів (рис. 146) і діаграму деталізацій класів (рис. 147).

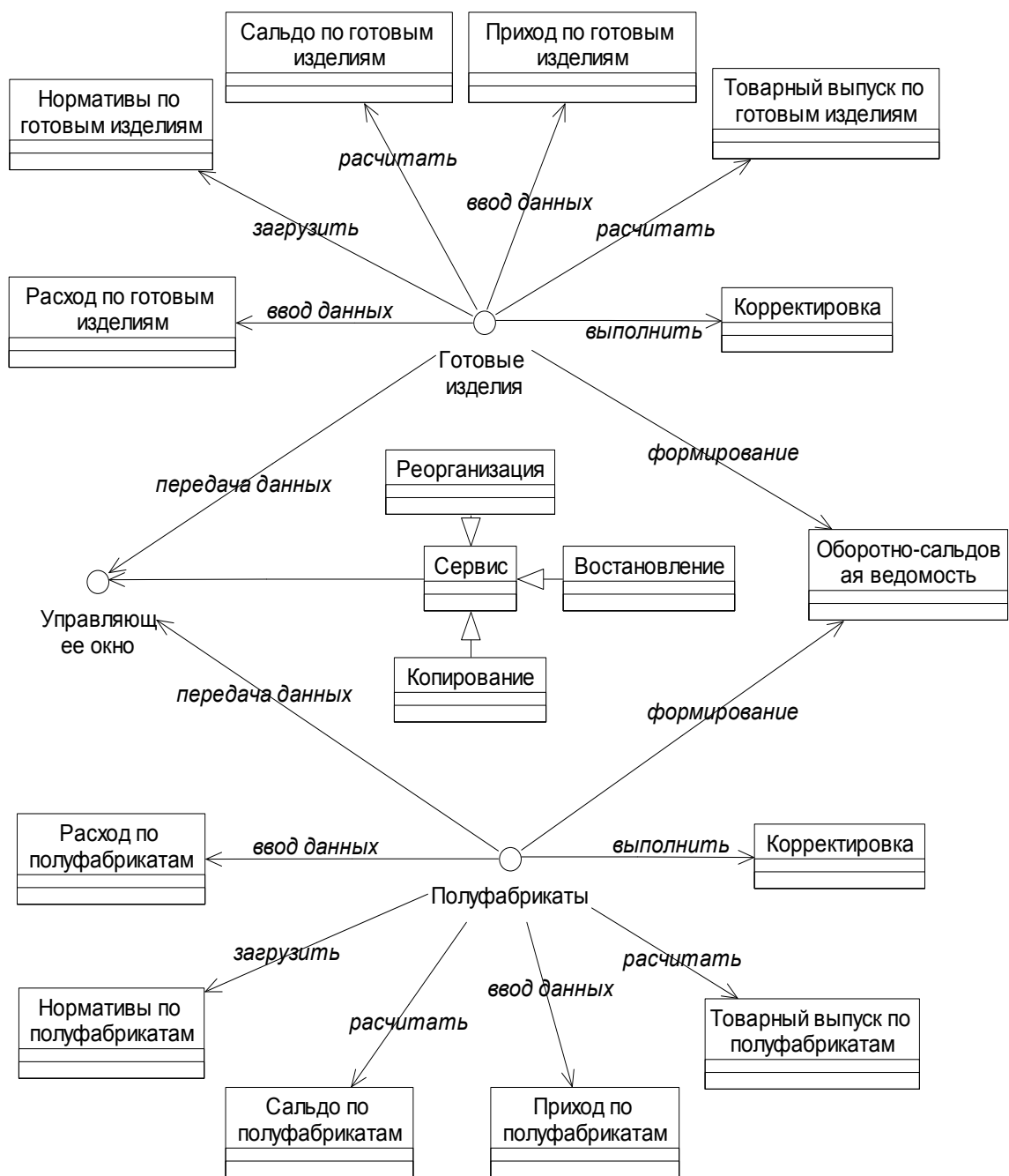


Рисунок 146 – Диаграмма асоціацій класів

Нормативы по готовым изделиям	Расход по готовым изделиям	Товарный выпуск по полуфабрикатам
Год отчетного периода Месяц отчетного периода Балансовый счет Шифр аналитического учета Номер цеха по заказу Количество Цена	Год отчетного периода Месяц отчетного периода Номер документа Цех Балансовый счет Шифр аналитического учета Вид операции Количество Цена Сумма	Год отчетного периода Месяц отчетного периода Балансовый счет Шифр аналитического учета Балансовый счет заказа Цех Шифр материала Количество Цена
+загрузка() +корректировка()	+ввод данных() +просмотр данных()	+расчет() +просмотр() +печать()
Приход по готовым изделиям	Сальдо по полуфабрикатам	Сальдо по готовым изделиям
Год отчетного периода Месяц отчетного периода Номер документа Цех Балансовый счет Шифр аналитического учета Количество Цена	Год отчетного периода Месяц отчетного периода Балансовый счет Шифр аналитического учета Балансовый счет заказа Цех Шифр материала Количество Цена Сумма	Год отчетного периода Месяц отчетного периода Балансовый счет Шифр аналитического учета Балансовый счет заказа Цех Шифр материала Количество Цена Сумма
+ввод данных() +просмотр данных()	+расчет() +просмотр() +печать()	+расчет() +просмотр() +печать()
Приход по готовым изделиям	Расход по полуфабрикатам	Сальдо-оборотная ведомость
Год отчетного периода Месяц отчетного периода Номер документа Цех Балансовый счет Шифр аналитического учета Количество Цена	Год отчетного периода Месяц отчетного периода Номер документа Цех Балансовый счет Шифр аналитического учета Вид операции Количество Цена Сумма	Год отчетного периода Месяц отчетного периода Номер заказа Шифр материала Входящее сальдо Приход Товарный выпуск Исходящее сальдо Количество Цена Сумма
+ввод данных() +просмотр данных()	+ввод данных() +просмотр данных()	+расчет() +просмотр() +печать()
Приход по полуфабрикатам	Товарный выпуск по готовым изделиям	
Год отчетного периода Месяц отчетного периода Номер документа Цех Балансовый счет Шифр аналитического учета Количество Цена	Год отчетного периода Месяц отчетного периода Балансовый счет Шифр аналитического учета Балансовый счет заказа Цех Шифр материала Количество Цена	
+ввод данных() +просмотр данных()	+расчет() +просмотр() +печать()	

Рисунок 147 – Диаграмма детализаций класів

Оскільки взаємодія об'єктів у випадках роботи з готовими виробами і напівфабрикатами декілька розрізняється, діаграму кооперації подамо у двох варіантах – для роботи з інтерфейсами «Готові вироби» (рис. 148) і «Напівфабрикати» (рис. 149).

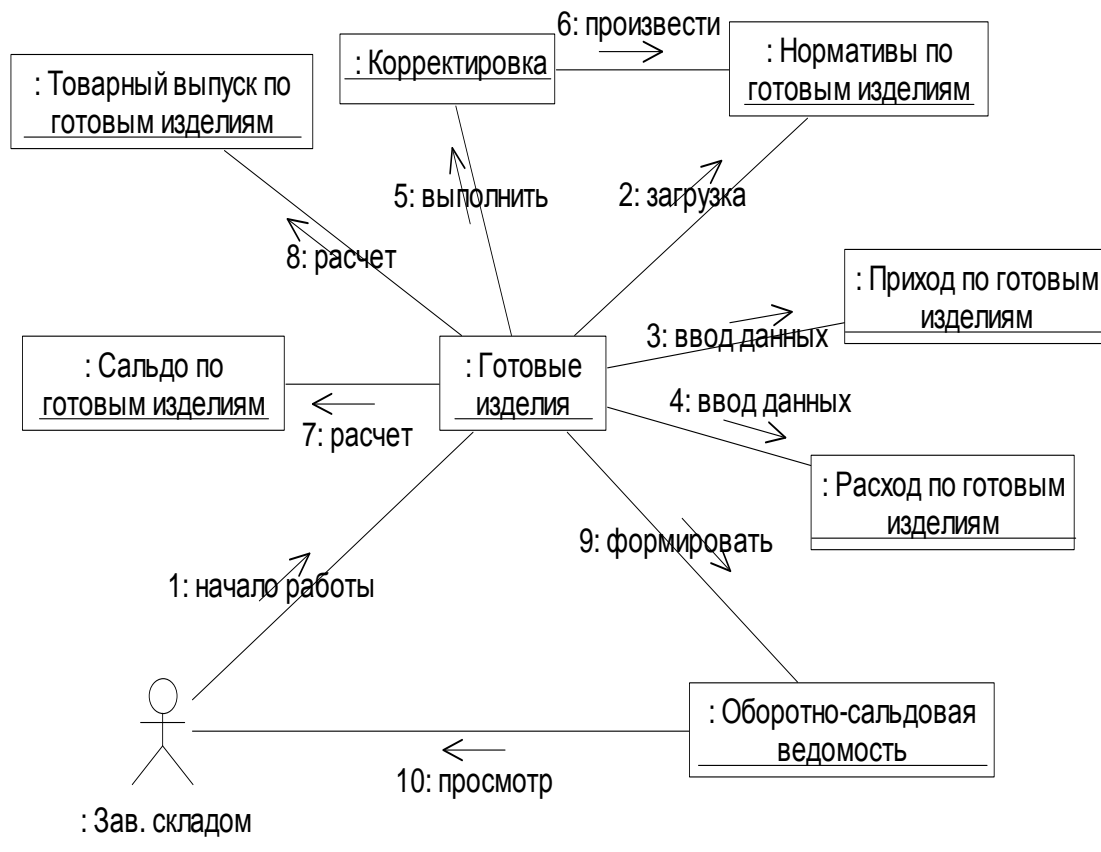


Рисунок 148 – Діаграма кооперації при використуванні інтерфейсу «Готові вироби»

Робота нашої системи нічим не відрізняється від функціонування типової системи, тому діаграма станів не будувалася.

Для моделювання процесу виконання операцій мовою UML використовуються діаграми діяльності; на рисунку 150 показані дії, які відбуваються під час функціонування об'єкта.

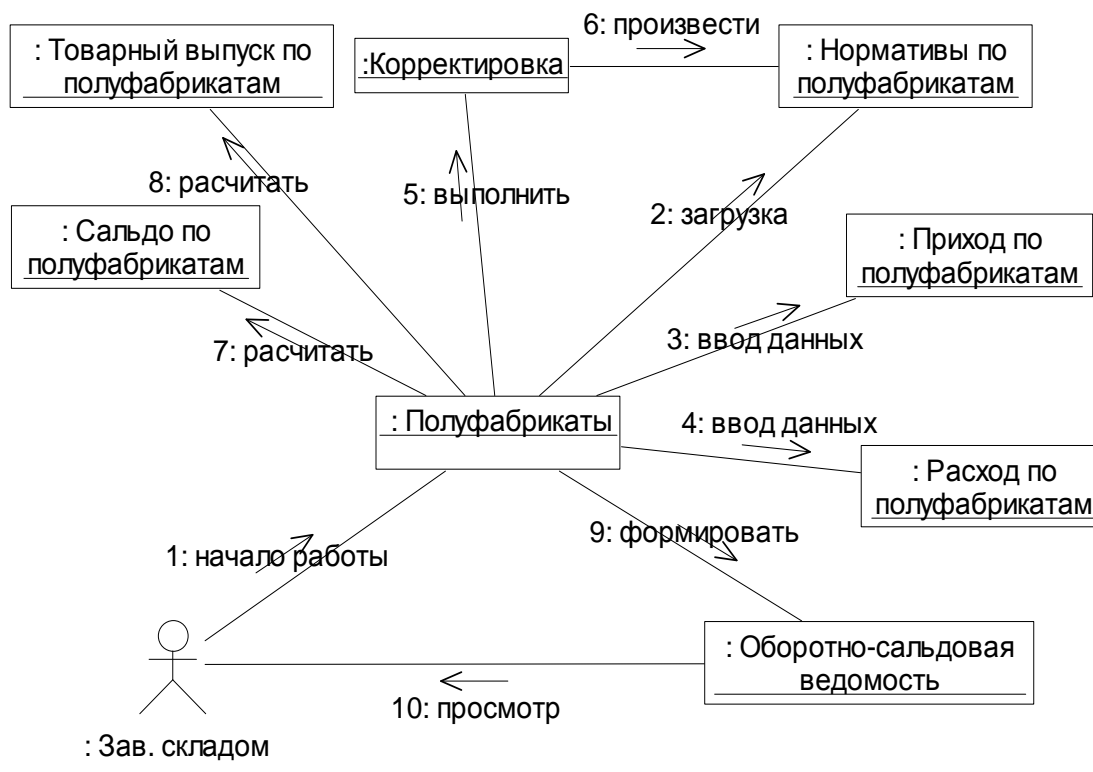


Рисунок 149 – Діаграма кооперації при використуванні інтерфейсу «Напівфабрикати»

Перед побудовою діаграми компонентів виділимо існуючі в системі інтерфейси (інтерфейс є «обличчям» об'єкта для інших об'єктів системи).

Для роботи з даними необхідний компонент «Таблиця», а для представлення даних таблиці – інтерфейс «Робота з даними». У свою чергу, робота з даними може включати перегляд і редагування даних, застосування запитів на вибірку за заданою умовою і формування звіту за заданою умовою. Отже, виділимо наступні інтерфейси: «Введення», «Завантаження», «Перегляд», «Редагування», «Запит на вибірку», «Звіт». Одержана діаграма компонентів зображена на рисунку 151. Компоненти, що мають тип «::Таблиця», на фізичному рівні подані відповідними таблицями бази даних.

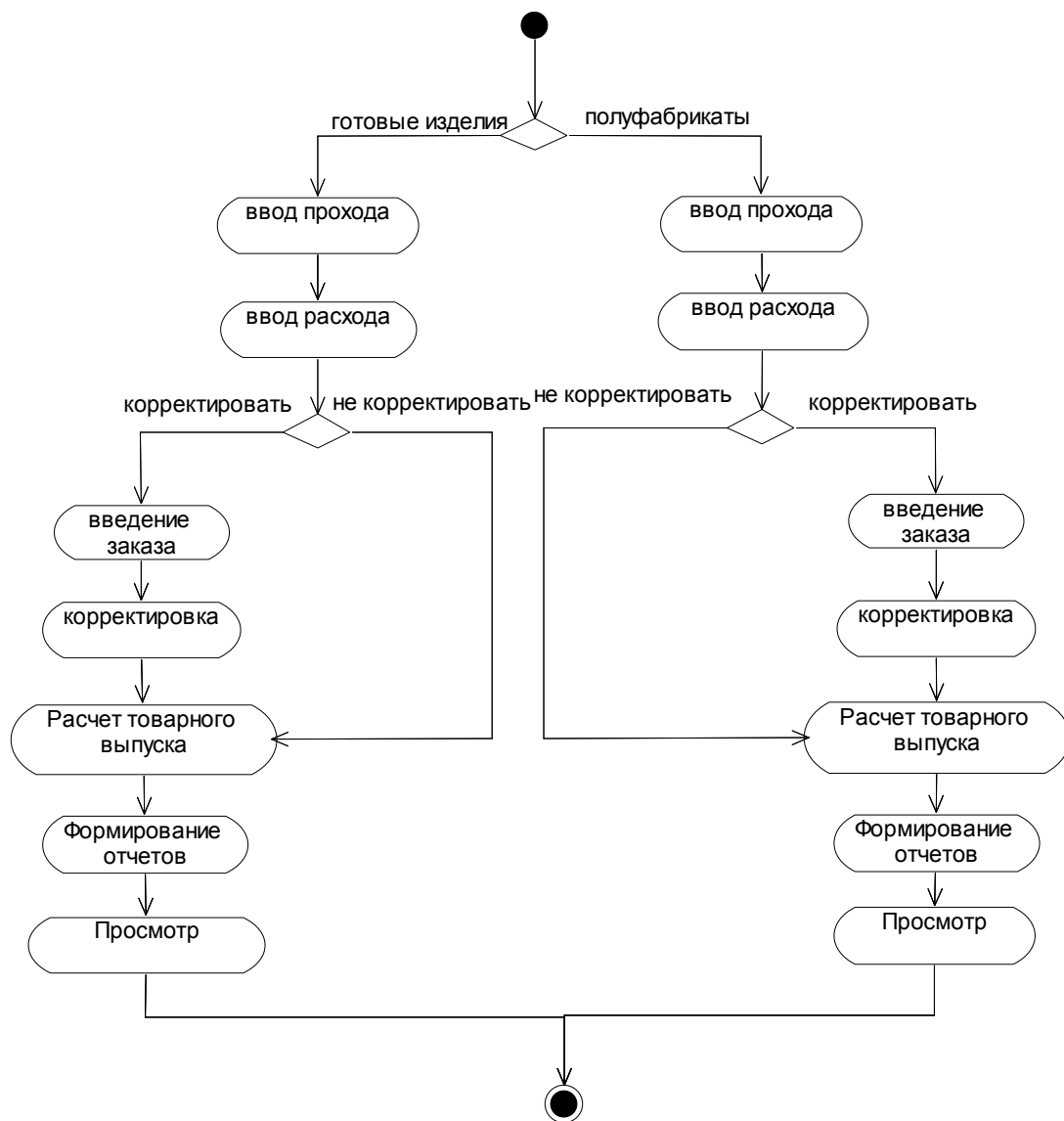


Рисунок 150 – Діаграма діяльності

Для зображення загальної конфігурації і топології розподіленої програмної системи існує діаграма розгортання (розміщення). Проаналізуємо вузли, задіяні в нашій системі.

Проектована програма повинна працювати з базою даних. База даних є видаленою, тобто розміщується на сервері мережі (архітектура проектованої «Клієнт-сервер»). Для отримання даних «клієнт-додаток» формує і посилає запит видаленому серверу. Сервер застосовує умову запиту до даних і мережею відправляє «клієнту-додатку» відповідні дані (рис. 152).

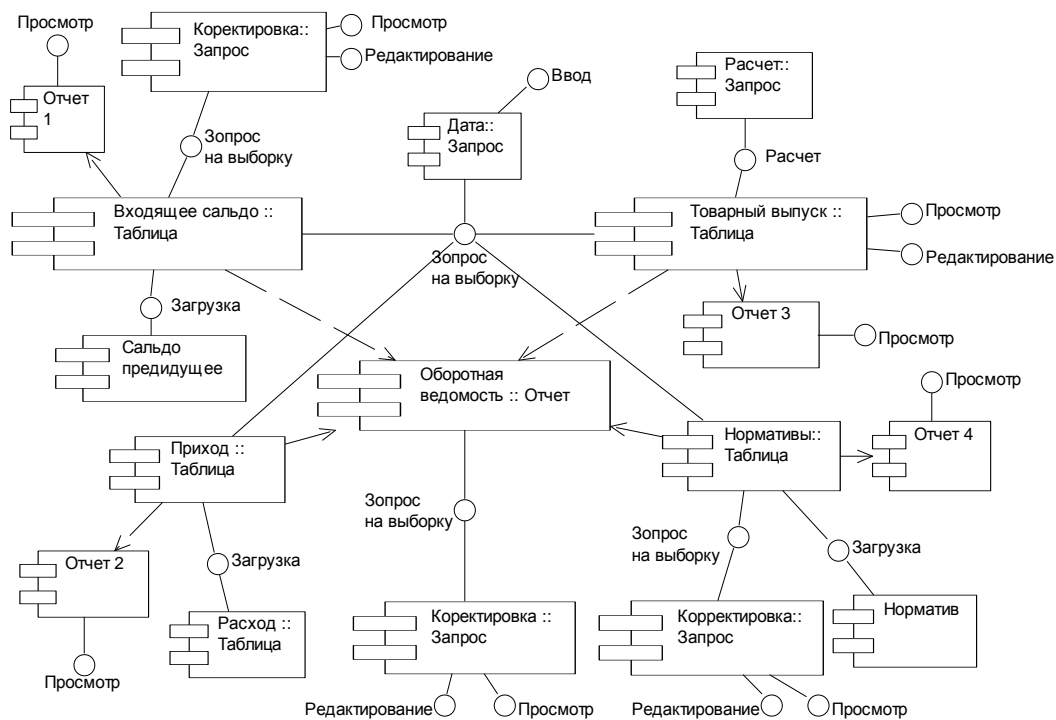


Рисунок 151 – Діаграма компонентів

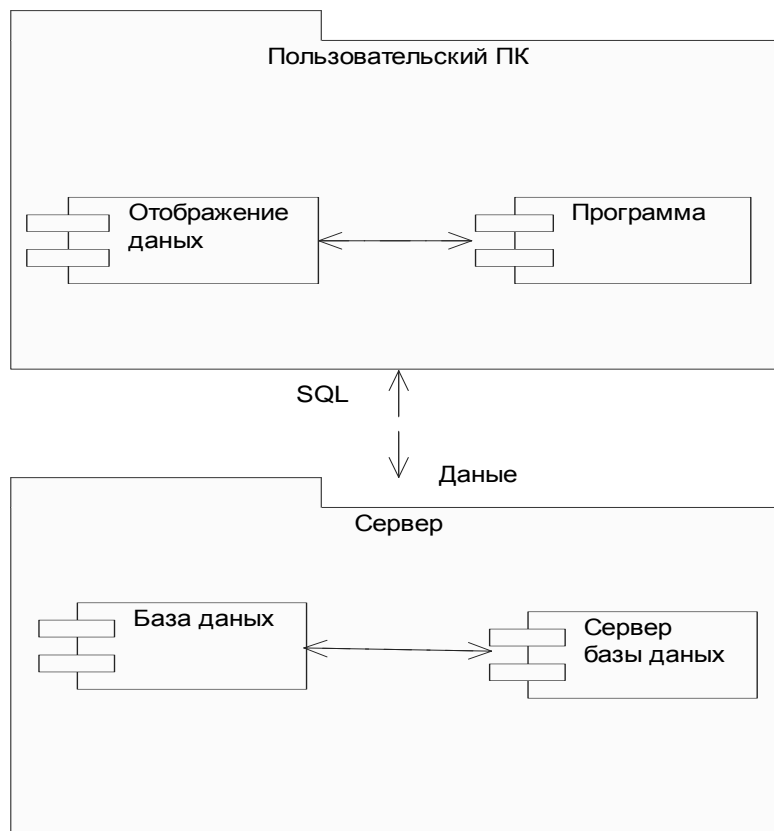


Рисунок 152 – Діаграма розгортання

Спроектowana інформаційна система була реалізована у середовищі Borland Delphi 7 і як програмний продукт «Готові вироби» упроваджена на Київському авіаційному заводі «Авіант» [28].

Був проведений розрахунок економічної ефективності, згідно з яким приріст прибутку на одну гривну одноразових витрат (капіталовкладень) повинен скласти 5,04 гр. Термін окупності капіталовкладень – 2,5 місяці.

4.8 Інформаційна система для маркетингових досліджень і аналізу надійності

Одним з основних етапів створення будь-якого виробу є визначення його області функціонування – ніші даного виробу серед множини йому подібних. Після створення дослідних зразків звичайно виконуються лабораторні, а потім і виробничі випробування. За наслідками випробувань, а також аналізу банку даних експлуатації подібних виробів визначаються статистичні характеристики надійності об'єктів. На підставі їх аналізу робиться висновок про те, наскільки виріб відповідає вимогам надійності, тобто виконується статистична оцінка однієї з найважливіших складових якості виробів.

Для позиціонування технічних виробів і для аналізу ніш різних товарів може бути використана інформаційна система. Наприклад, необхідно визначити, яка розфасовка прального порошку матиме найбільший попит. За допомогою соціального опиту або анкетування визначається набір даних як варіаційний ряд значень і за допомогою інформаційної системи маркетингових досліджень визначається найбільш затребуваний об'єм упаковки.

Таким чином, є необхідність задоволення потреб різних суб'єктів в статистичному аналізі стосовно маркетингових досліджень для позиціонування виробів і оцінки надійності технічних об'єктів. Об'єкт дослідження – варіаційні ряди числових значень характеристик використання виробів, а також варіаційні ряди часів безвідмовної роботи, часів відновлення або наладки. На основі перших проводяться маркетингові дослідження ніш, другі використовуються для статистичного аналізу надійності і контролю якості виробів.

Для проектування інформаційної системи була застосована уніфікована мова створення моделей – UML (Unified Modeling Language). Він допомагає відобразити бачення системи і дає можливість обговорювати його зі всіма зацікавленими особами. Це робиться за допомогою набору позначень і діаграм, причому кожна діаграма виконує свою роль у процесі розробки. Побудова UML-діаграм виконувалася у програмі Rational Rose Enterprise Edition, тому створені діаграми відображають особливості цього інструменту.

Спочатку сформулюємо основні вимоги до системи, і які задачі вона повинна вирішувати:

- 1 Введення різного типу даних, їх редагування, конвертація у відповідний тип (при необхідності) і перегляд. Для характеристик об'єкта можуть застосовуватися як варіаційні ряди, так і послідовності частот і частотей.
- 2 Відповідна обробка введених даних, сортування і т.д.
- 3 Запит у користувача необхідних для розрахунку параметрів.
- 4 Розрахунок характеристик надійності за введеними даними.
- 5 Висновок відповідних графіків і звітів.
- 6 Оскільки інтерфейс програми повинен бути максимально дружній користувачу, інформаційна система повинна мати оболонку, що настроюється користувачем.

Візуальне моделювання в UML є процесом поетапного спуску від найзагальнішої і абстрактнішої концептуальної моделі початкової системи до логічної, а потім і до фізичної моделі відповідної програмної системи. Для досягнення цих цілей спочатку будується модель у формі так званої діаграми варіантів використання (use case diagram), що описує функціональне призначення системи.

Безпосереднім користувачем системи (актором) є «Користувач ІС». Йому доступні наступні варіанти роботи з інформаційною системою: робота з даними (власне самі розрахунки), робота з формою (перегляд звітів і інформації про систему, настройка інтерфейсу програми, обчислення на вбудованому калькуляторі й ін.) і робота з додатковими модулями – Editor

і Converter. Це цілком відособлені програмні модулі, призначені для допоміжної роботи з даними (перегляд даних, їх редагування і конвертація початкових даних у формат, прийнятний інформаційною системою). Для спрощення проектування модель була розбита на 2 модулі (один включає інший). На рисунках 153 і 154 наводяться діаграми цих модулів.

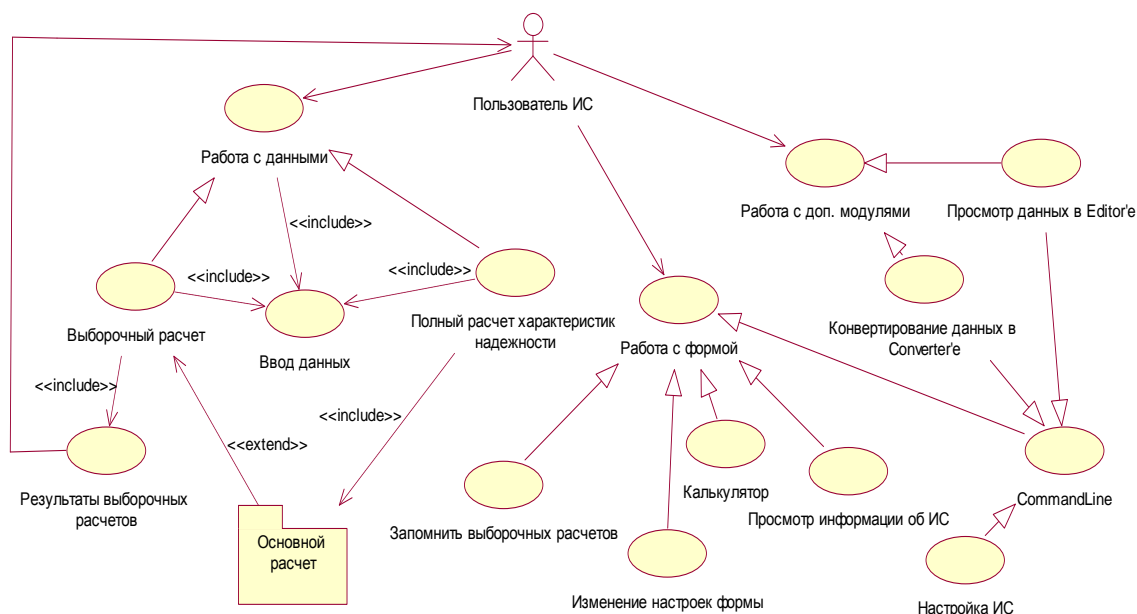


Рисунок 153 – Діаграма варіантів використання (головна програма)

Повна модель складної системи допускає деяку кількість взаємозв'язаних уявлень (views), кожне з яких адекватно відображає деякий аспект поведінки або структури системи. При цьому найзагальнішим поданням складної системи заведено рахувати статичне і динамічне уявлення, які, у свою чергу, можуть підрозділятися на інші, більш приватні уявлення.

До статичного уявлення відноситься віддзеркалення класів інформаційної системи у вигляді діаграми класів. Розробка логічної моделі системи у вигляді діаграми класів посідає центральне місце у процесі проектування інформаційної системи. У даній роботі статична модель розбита на власне діаграму класів (рис. 155) і конкретизацію класів цієї діаграми (рис. 156).

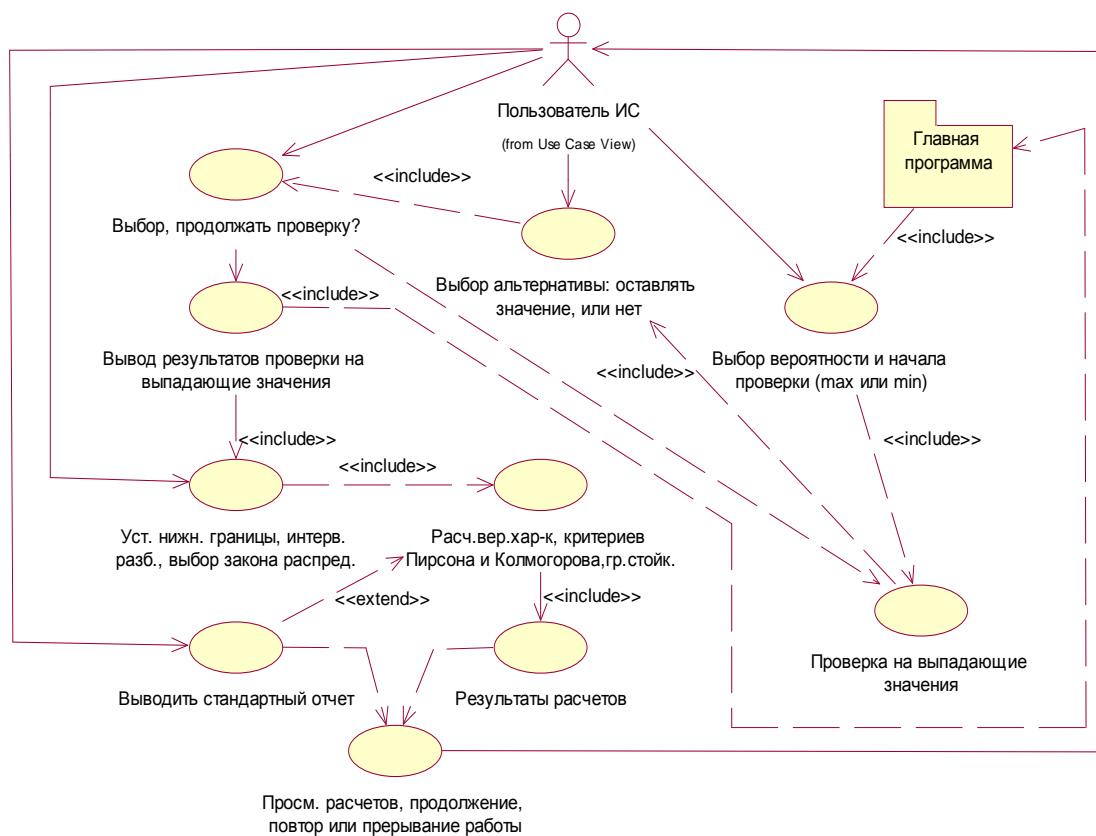


Рисунок 154 – Диаграмма вариантов использования (основной розрахунок)

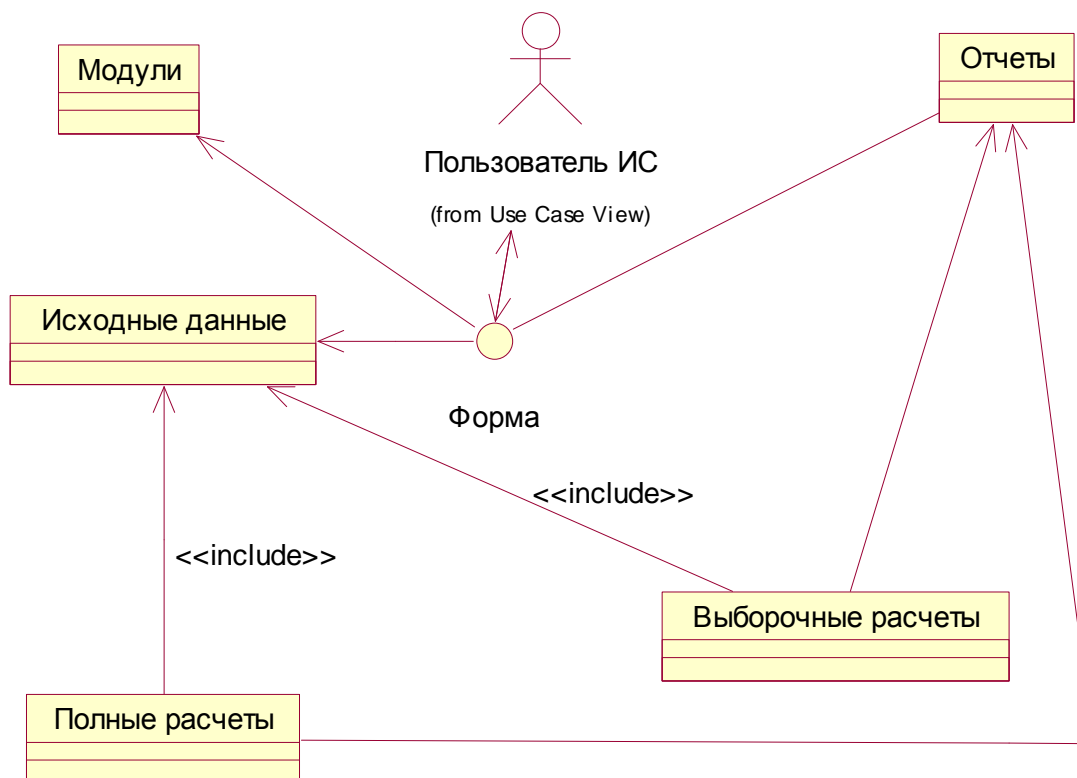


Рисунок 155 – Диаграмма классов

Основними класами є: інтерфейсний клас «Форма»; класи «Модулі», «Звіти», «Початкові дані», «Вибіркові розрахунки», «Повні розрахунки» і клас актора «Користувач ІС». Відносини між класами у діаграмах, побудованих в Rational Rose Enterprise Edition, описуються подібно відносинам у діаграмах варіантів використання.

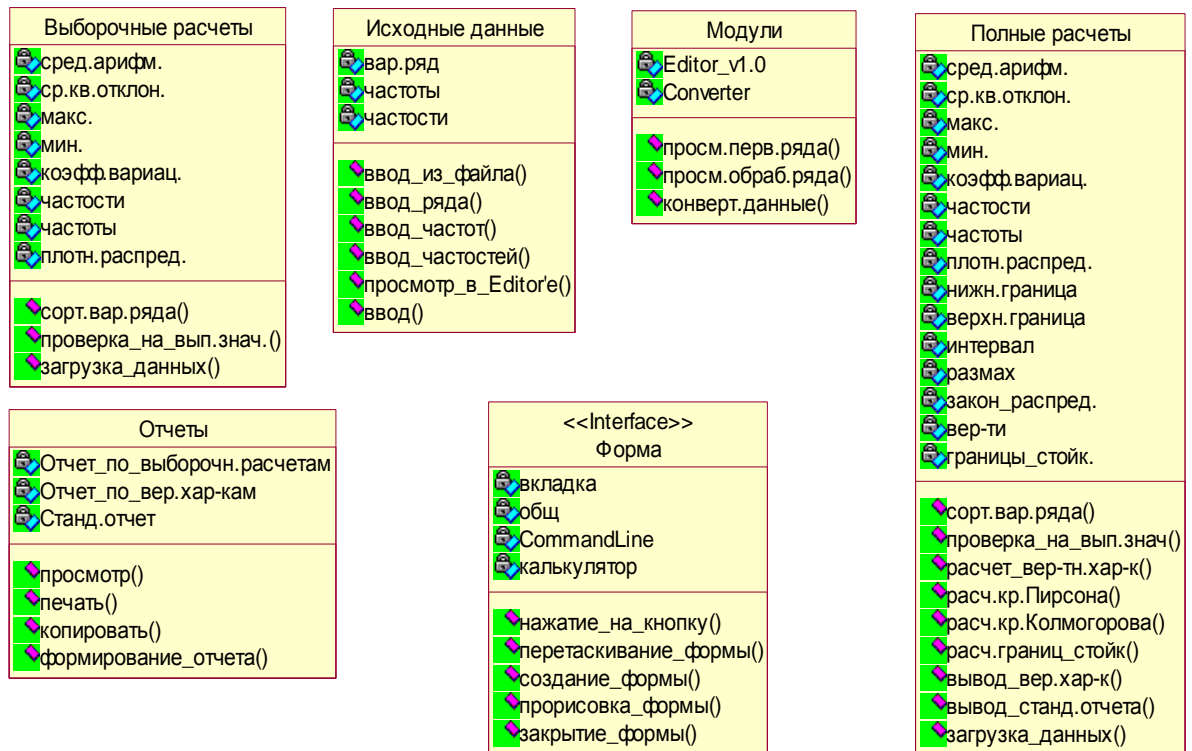


Рисунок 156 – Конкретизація класів

Для віддзеркалення динамічних особливостей (аспектів) проектованої системи застосовуються діаграми поведінки, які подані різними видами діаграм (станів, діяльності та взаємодії). В інтерпретації Rational Rose діаграми станів і діяльності об'єднані єдиною назвою – State Machine Diagram. Проте побудова їх взаємно незалежна.

Діаграма станів (Statechart) призначена для відображення станів об'єктів системи, що мають складну модель поведінки. Стани сполучені між собою відносинами асоціації, над якими, як правило, надписані умови переходу із стану до стану. У проектованій системі діаграма складається з 3 основних станів: введення даних, повний і вибірковий розрахунок. Причому, у введення даних є підстани, а у розрахунків є списки дій (рис. 157).

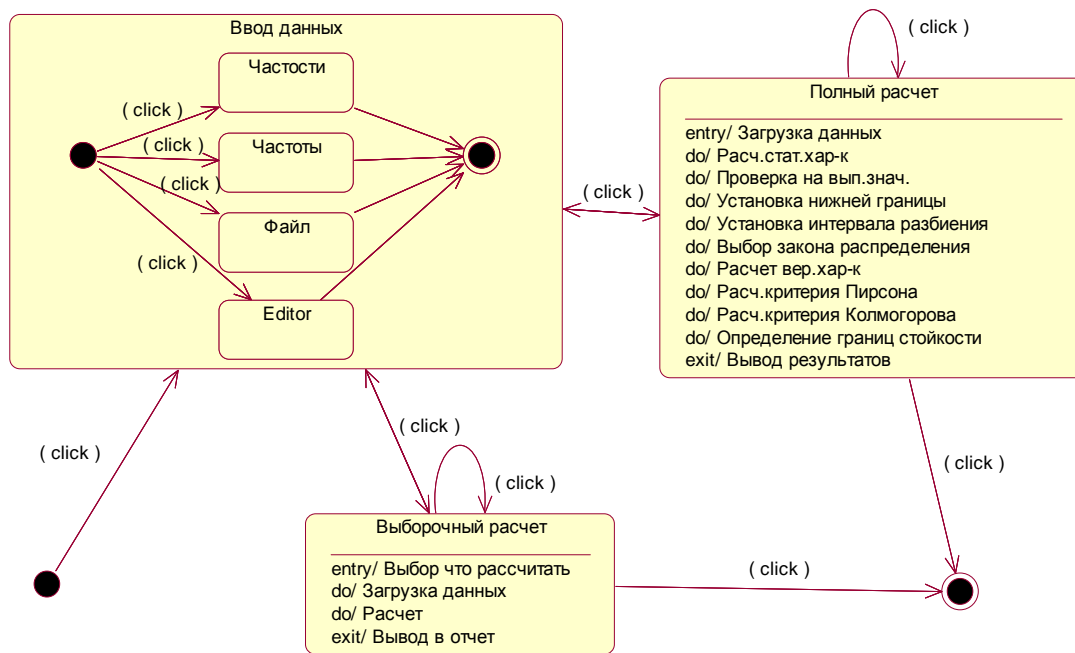


Рисунок 157 – Діаграма станів

Діаграма діяльності – це подальший розвиток діаграми станів. Вона дозволяє показати не тільки послідовність процесів, але і їх галуження і навіть синхронізацію. Для побудови діаграми діяльності проекрованої інформаційної системи були вибрані 4 класи: «Користувач ІС», «Вибіркові розрахунки», «Повні розрахунки», «Звіти» (інші опущені, як що мають допоміжне або супутнє розрахункам значення) – рисунок 158.

Для проекрованої інформаційної системи можна скласти діаграму кооперації, поданої у 3 аспектах: інтерактивна частина (рис. 159) – взаємодія користувача системи з різними формами і звітами, вибіркового розрахунок (рис. 160) і повний розрахунок (рис. 161) – взаємодія користувача з системою у процесі вищезгаданих розрахунків.

Усі описані діаграми описують лише якусь абстракцію, існуючу лише віртуально. Для опису реальнішої і доступнішої для розуміння фізичної форми системи застосовуються діаграми реалізації – діаграма компонентів і діаграма розгортання. Діаграму компонентів проекрованої системи можна розглядати як діаграму проекту системи (рис. 162) і діаграму компонентів реалізованої системи (рис. 163).

Оскільки проектована інформаційна система не припускає мережеву роботу, то діаграма розгортання не будується.

Програмна реалізація системи була виконана в середовищі Borland Delphi 6.0. Проект мав назву StatWorks і включав 12 форм інтерфейсу і більше 6,5 тис. рядків початкового тексту [29]. У даний час програма використовується в навчальному процесі кафедри «Металорізальні верстати та інструменти» ДДМА.

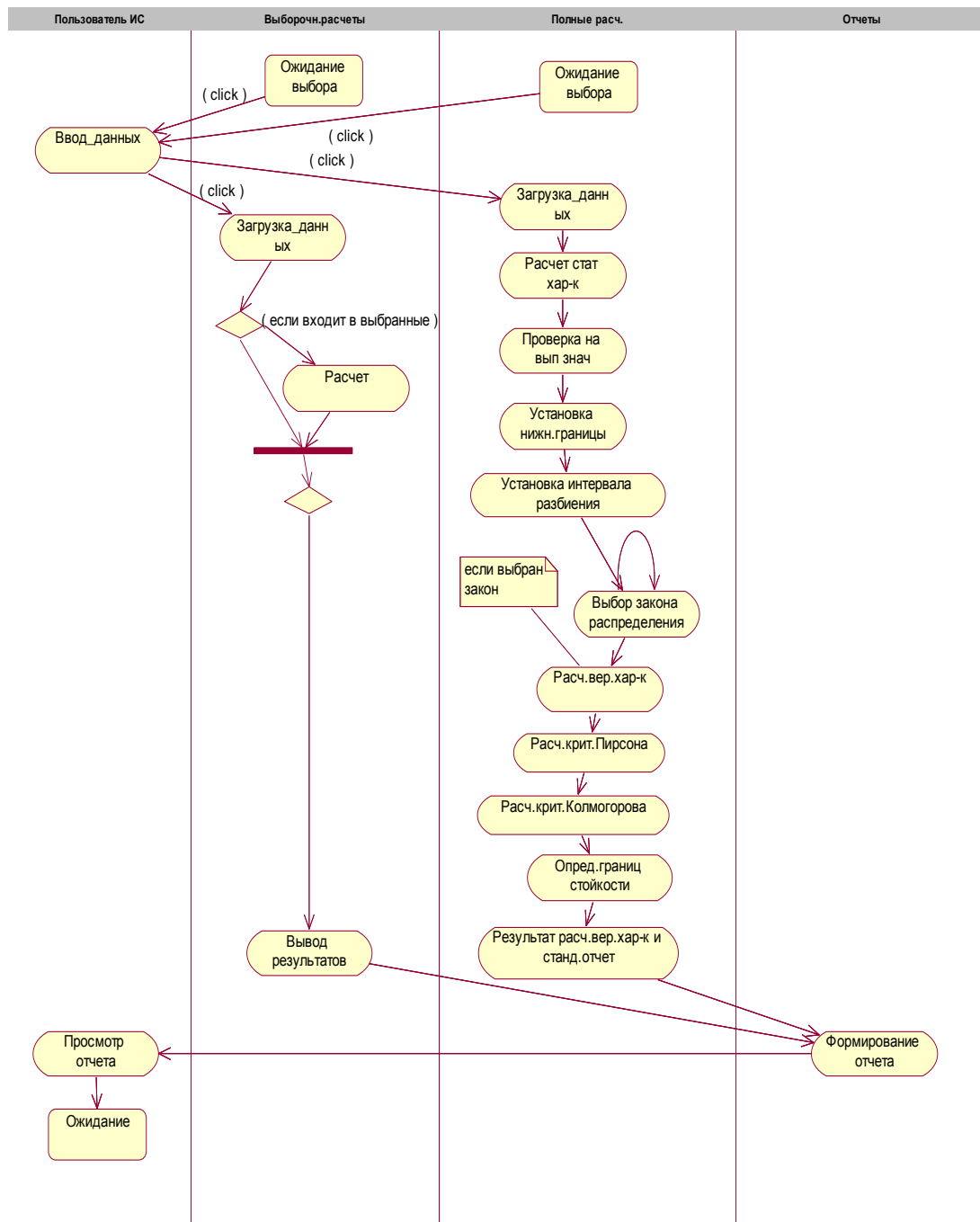


Рисунок 158 – Діаграма діяльності

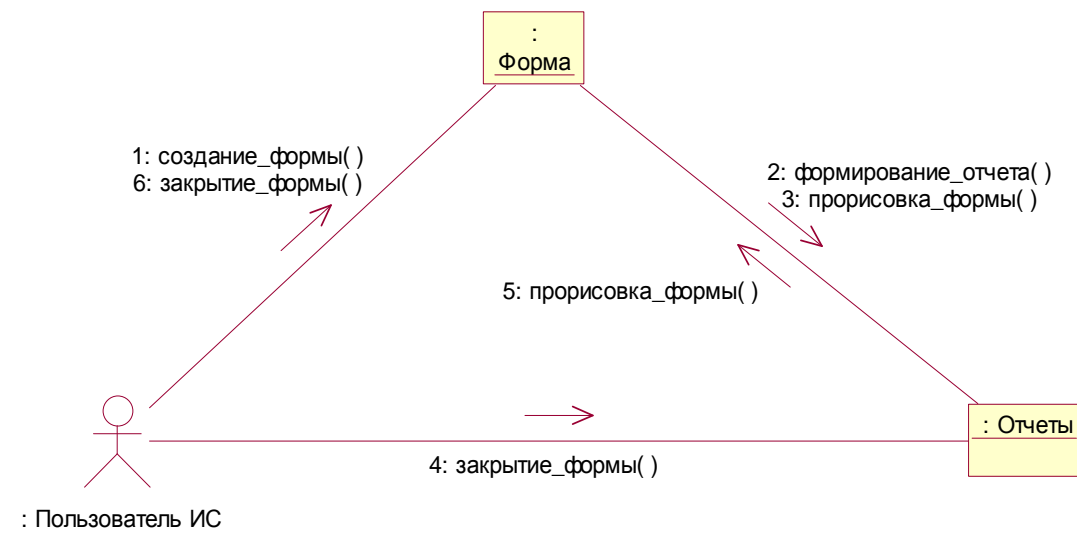


Рисунок 159 – Діаграма кооперації (інтерактивна частина)

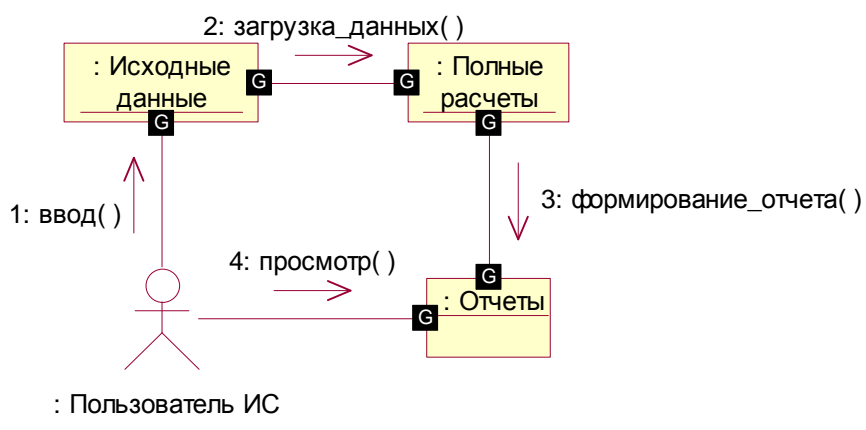


Рисунок 160 – Діаграма кооперації (повний розрахунок)

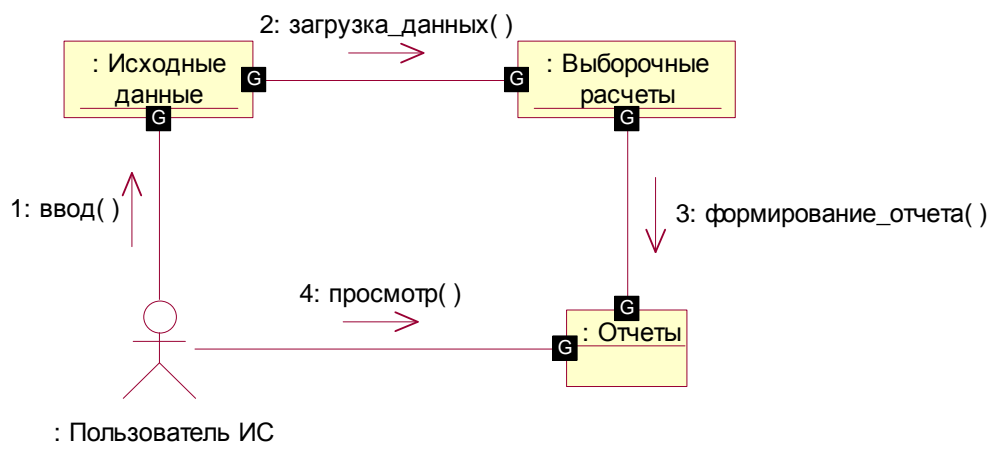


Рисунок 161 – Діаграма кооперації (вибірковий розрахунок)

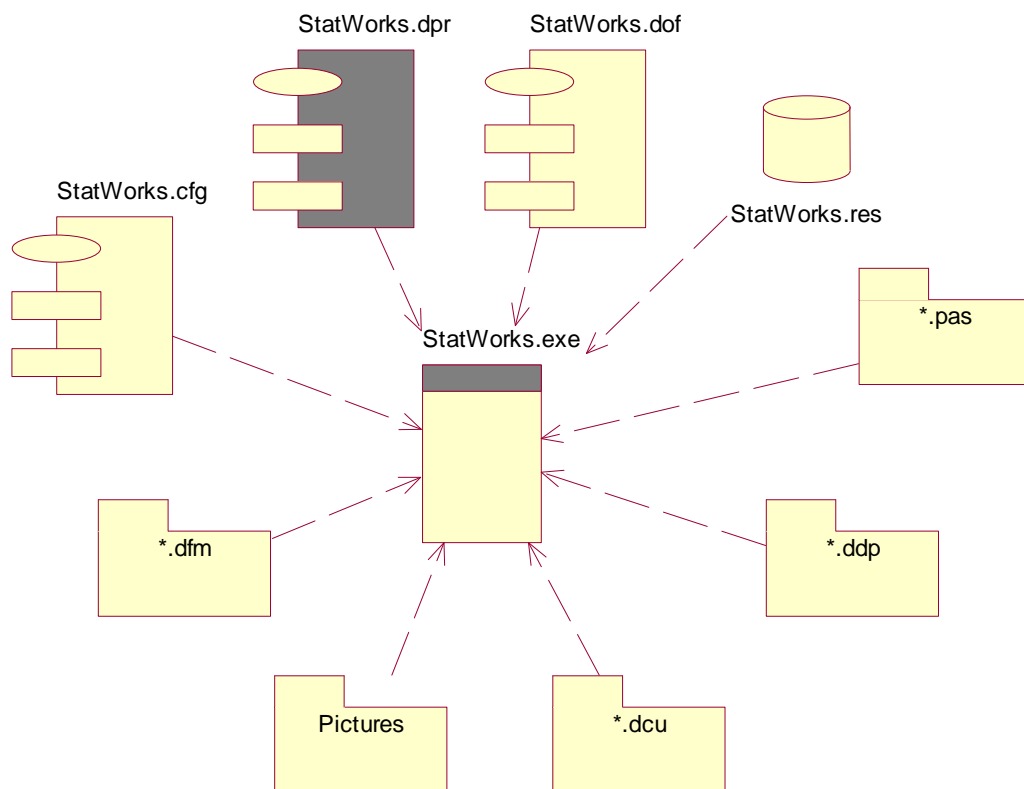


Рисунок 162 – Діаграма компонентів проекту системи

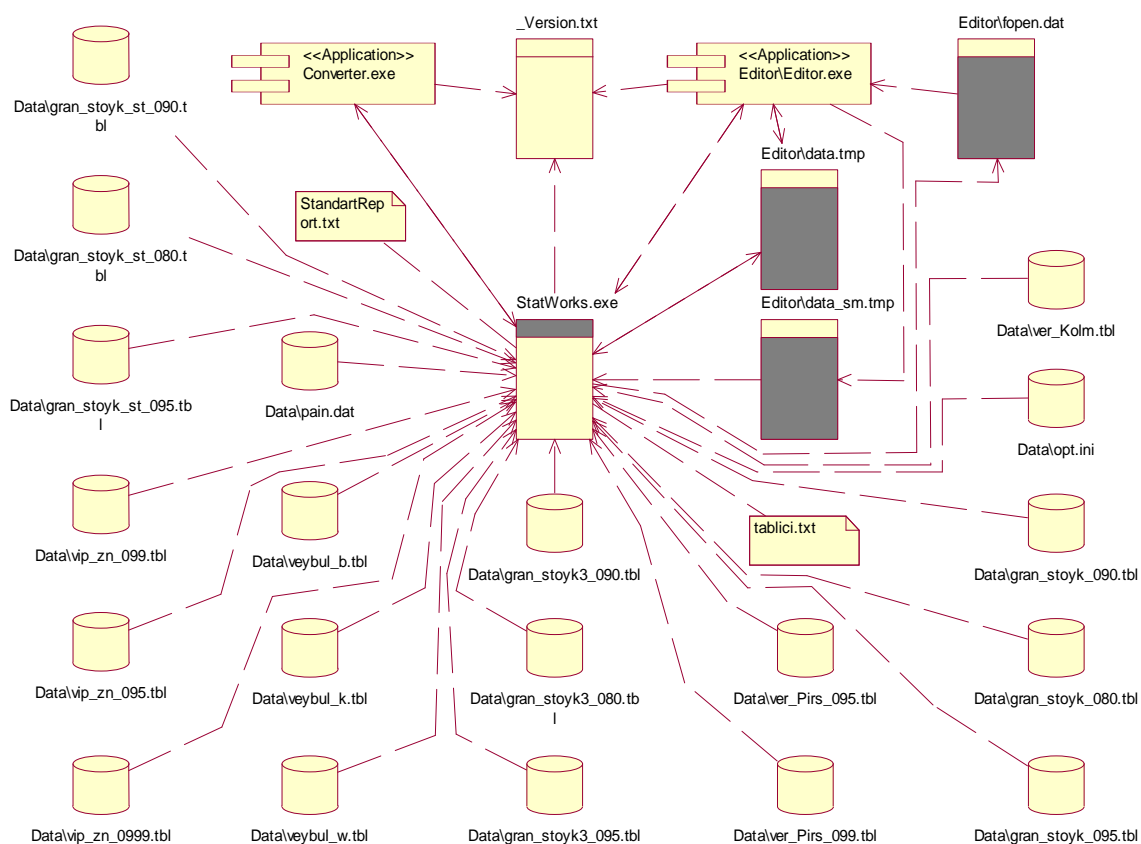


Рисунок 163 – Діаграма компонентів реалізованої системи

СПИСОК ЛІТЕРАТУРИ

- 1 Грейди Буч. Объектно-ориентированный анализ и проектирование с примерами приложений на C++. – 2-е изд. / Пер. с англ. – М.: Издательство Бином; СПб.: Невский диалект, 2001. – 560 с.
- 2 Леоненков А.В. Самоучитель UML. – 2-е изд., перераб. и доп. – СПб.: БХВ-Петербург, 2004. – 432 с.
- 3 Грейди Буч. Язык UML. Руководство пользователя / Грейди Буч, Джеймс Рамбо, Айвар Джекобсон; Пер. с англ. А.А.Слинкин. – 2-е изд., стер. – М.: ДМК Пресс; СПб.: Питер, 2004. – 432 с.
- 4 Т. Кватрани. Rational Rose 2000 и UML. Визуальное моделирование. – СПб.: ДМК-Пресс, 2000. – 176 с.
- 5 UML и Rational Rose / Под ред. У.Боггс, М.Боггс. – М.: Лори, 2001. – 608с.
- 6 UML / Под ред. М. Фауллера, К. Скотта. – М.: Мир, 1999. – 191с.
- 7 Шмюлер Джозеф. Освой самостоятельно UML за 24 часа/ Пер. с англ.. – 2-е издание. – М.: Издательский дом «Вильямс», 2002. – 352 с.
- 8 Бабич А.В. Обзор CASE-средств для построения диаграмм UML // Теорія та методика навчання математики, фізики, інформатики: Збірник наукових праць. Випуск 5: У 3-х томах. – Кривий Ріг: Видавничий відділ НМетАУ, 2005. – Т.3: Теорія та методика навчання інформатики. – С. 8-11.
- 9 Васись Д.В. Информационная система для функционирования кадрового агентства/ Д.В.Васись, А.Ю.Мельников // Сборник докладов 2-го молодежного форума «Информационные технологии в XXI-м веке» (27-28 апреля 2004 г., г. Днепропетровск). – Днепропетровск: ИПК ИнКом-Центра УГХТУ, 2004. – С.31-32.
- 10 Васись Д.В. Інформаційна система для функціонування кадрового агентства/ Д.В.Васись, О.Ю.Мельников // Інформаційні технології в освіті, науці і техніці: Матеріали IV Всеукраїнської конференції молодих науковців ІТОНТ-2004: Черкаси, 28-30 квітня 2004 р. – Черкаси: ЧНУ, 2004. – С.87-89.
- 11 Мельников А.Ю. Информатизация функционирования кадрового агентства/ А.Ю.Мельников, Д.В.Васись// Вісн. Східноукр. нац. ун-ту ім. В. Даля – Луганськ. – 2004. – №11. – С. 230-234.

12 Мельников А.Ю. Проектирование информационной системы для функционирования кадрового агентства/ А.Ю.Мельников, Д.В.Васись // Современные проблемы информатизации в непроизводственной сфере и экономике: Сб. трудов. – Вып. 10 / Под ред. д.т.н., проф. О.Я. Кравца. – Воронеж: Издательство «Научная книга», 2005. – С.14-15.

13 Мельников А.Ю. Объектно-ориентированное проектирование информационной системы для функционирования кадрового агентства/ А.Ю.Мельников, Д.В.Васись // Вісник Донбаської державної машинобудівної академії: Збірник наукових праць. – №1. – 2005. – Краматорськ: ДДМА, 2005. – С. 233-237.

14 Мельников А.Ю. Автоматизация процесса составления расписания занятий в высшем учебном заведении/ А.Ю.Мельников, Н.М.Сусяк // Комп'ютерне моделювання в освіті: Матеріали Всеукраїнського науково-методичного семінару. – 29 березня 2005р. – Кривий Ріг: КДПУ, 2005. – С.52-53.

15 Сусяк Н.М. Проектирование системы для автоматизации составления расписания занятий в вузе/ Н.М.Сусяк, А.Ю.Мельников // Информационные технологии в XXI веке: Сборник докладов и тезисов III-го Молодежного научно-практического форума (Днепропетровск, 27-28 апреля 2005г.) / Под ред. акад. НАНУ В.В. Пилипенко, д.х.н. М.В. Бурмистра, к.ф.-м.н. Н.Ф. Огданского, к.ф.-м.н. Ю.А. Прокопчука. – Днепропетровск: ИПК ИнКомЦентра УГХТУ, 2005. – С. 189-190.

16 Мельников О.Ю. Проектування системи для автоматизованого складання розкладу занять у вищому навчальному закладі/ О.Ю.Мельников, Н.М.Сусяк // Проблеми прийняття рішень в умовах невизначеності: Матеріали міжнародної науково-практичної конференції (м. Бердянськ, 12-17 вересня 2005 року). – Бердянськ, 2005. – С. 70-71.

17 Мельников О.Ю. Проектування системи для автоматизованого складання розкладу занять у вищому навчальному закладі / О.Ю.Мельников, Н.М.Сусяк // Збірник наукових праць Бердянського державного педагогічного університету (Педагогічні науки). – № 3. – Бердянськ: БДПУ, 2005. – С. 40-46.

18 Мельников А.Ю. Автоматизированное составление расписания занятий в высшем учебном заведении/ А.Ю.Мельников, Н.М.Сусяк // Современные проблемы информатизации в информационных системах и те-

лекоммуникациях: Сб. трудов. – Вып. 11 / Под ред. д.т.н., проф. О.Я.Кравца. – Воронеж: Издательство «Научная книга», 2006. – С.344-345.

19 Мельников А.Ю. Система для автоматизированного составления расписания занятий в высшем учебном заведении/ А.Ю.Мельников, Н.М.Сусяк // Открытое и дистанционное образование. – Томск, 2006. – № 2 (22). – С. 52-57.

20 Мельников А.Ю. Разработка информационной системы для специализированного торгового предприятия/ А.Ю.Мельников, А.В.Руденко // Восточно-Европейский журнал передовых технологий. – Харьков, 2005. – №4/2(16). – С. 124-127.

21 Мельников А.Ю. Проектирование информационной системы для специализированного торгового предприятия/ А.Ю.Мельников, А.В.Руденко // Современные проблемы информатизации в информационных системах и телекоммуникациях: Сб. трудов. – Вып. 11 / Под ред. д.т.н., проф. О.Я. Кравца. – Воронеж: Издательство «Научная книга», 2006. – С.37-38.

22 Ольховська О.Л. Проектування інформаційної системи функціонування страхової компанії/ О.Л.Ольховська, О.Ю.Мельников // Информационные технологии в XXI веке: Сборник докладов и тезисов III-го Молодежного научно-практического форума (Днепропетровск, 27-28 апреля 2005г.) / Под ред. акад. НАНУ В.В. Пилипенко, д.х.н. М.В. Бурмистра, к.ф.-м.н. Н.Ф. Огданского, к.ф.-м.н. Ю.А. Прокопчука. – Днепропетровск: ИПК ИнКомЦентра УГХТУ, 2005. – С. 147-148.

23 Мельников А.Ю. Проектирование информационной системы для функционирования страховой компании/ А.Ю.Мельников, О.Л.Ольховская // Современные проблемы информатизации в информационных системах и телекоммуникациях: Сб. трудов. – Вып. 11 / Под ред. д.т.н., проф. О.Я. Кравца. – Воронеж: Издательство «Научная книга», 2006. – С.38-39.

24 Мельников А.Ю. Разработка информационной системы функционирования страховой компании/ А.Ю.Мельников, О.Л.Ольховская // Информационные технологии моделирования и управления: Научно-технический журнал. – Воронеж: Издательство «Научная книга», 2006. – №2 (27). – С.277-283.

25 Мельников О.Ю. Проектування інформаційної системи для невеликої страхової компанії/ О.Ю.Мельников, О.Л.Ольховська // Вісник Наці-

онального технічного університету «Харківський політехнічний інститут»: Збірник наукових праць. Тематичний випуск: Інформатика і моделювання. – Харків: НТУ «ХПІ». – 2005. – № 56. – С. 91-99.

26 Мельников А.Ю. Методы расчета себестоимости металлопродукции/ А.Ю.Мельников, В.Ю.Гуржиев // Сборник тезисов IV Международной научно-практической конференции молодых ученых и специалистов «Интеллект молодых – производству 2005». – Краматорск, 2005. – С. 130-131.

27 Мельников А.Ю. Автоматизация расчета себестоимости/ А.Ю.Мельников, В.Ю.Гуржиев // Важке машинобудування. Проблеми та перспективи розвитку. Матеріали четвертої Міжнародної науково-технічної конференції 5-8 червня 2006 року / Під заг.ред. В.Д.Ковальова. – Краматорськ: ДДМА, 2006. – С.68.

28 Ключков Е.В. Автоматизация учета и контроля производственной деятельности КИГАЗ «Авиант»: учет и контроль готовой продукции // Збірник (частина 2) тез доповідей IX Всеукраїнської науково-практичної конференції студентів, аспірантів та молодих вчених “Технологія-2006” (13-14 квітня 2005 р., м.Северодонецк) / СТІ СХУ ім. В.Даля. – Северодонецьк, 2006. – С. 59.

29 Мельников А.Ю. Разработка информационной системы для маркетинговых исследований и анализа надежности/ А.Ю.Мельников, В.Л.Аносов, Д.Е.Прекрасный // Восточно-Европейский журнал передовых технологий. – Харьков, 2005. – №1 (19). – С. 122-127.

Навчальне видання

МЕЛЬНИКОВ Олександр Юрійович

**ОБ'ЄКТНО-ОРІЄНТОВАНИЙ АНАЛІЗ
І ПРОЕКТУВАННЯ ІНФОРМАЦІЙНИХ СИСТЕМ**

Навчальний посібник

Редактор

І.І.Дьякова

Комп'ютерна верстка

О.П.Ордіна

Вз. 96/2006. Підп. до друку

Формат 60x84/16.

Папір офсетний. Ум. друк. арк.

Обл.-вид. арк.

Тираж прим. Зам. №

Видавець і виготівник

«Донбаська державна машинобудівна академія»

84313, м. Краматорськ, вул. Шкадінова, 72

Свідоцтво про внесення суб'єкта видавничої справи до державного реєстру
серія ДК № 1633 від 24.12.2003