МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

МЕТОДИЧНІ ВКАЗІВКИ

до виконання лабораторних робіт з дисципліни

«Архітектура та проектування програмного забезпечення»

за розділом

«Мова моделювання UML як засіб аналізу та проектування програмних систем»

для студентів з напряму підготовки 6.050103 «Програмна інженерія»

Харків 2017

МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

МЕТОДИЧНІ ВКАЗІВКИ

до виконання лабораторних робіт з дисципліни
«Архітектура та проектування програмного забезпечення»
за розділом
«Мова моделювання UML як засіб аналізу та проектування програмних систем»
для студентів з напряму підготовки 6.050103 «Програмна інженерія»

Затверджено
редакційно-видавничою
радою університету,
протокол № 01 від 22.06.2017

Харків
НТУ «ХПІ»
2017

Методичні вказівки до виконання лабораторних робіт з дисципліни «Архітектура та проектування програмного забезпечення» за розділом «Мова моделювання UML як засіб аналізу та проектування програмних систем» / уклад. Ткачук М.В, Сокол В.Є., Гамзаєв Р.О. та інш. – Харків.: НТУ “ХПІ”, 2017. – 60 с.

Укладачі : М.В. Ткачук, В.Є. Сокол, Р.О. Гамзаєв, І.О. Мартінкус, К.А. Нагорний, О.С. Науменко, С.В. Бронін.

Рецензент проф. Кучук Г.А.

Кафедра програмної інженерії та інформаційних технологій управління

ЗМІСТ

ВСТУП

У сучасній програмній інженерії широко застосовуються різноманітні моделі з метою аналізу вимог та синтезу відповідних проектних рішень при створенні програмних систем (ПС) різного призначення. Моделювання процесів і артефактів життєвого циклу (ЖЦ), створення та супровід ПС є складним та трудомістким завданням, для ефективного вирішення якого необхідно застосовувати відповідні мовні та інструментальні засоби, які мають автоматизувати як сам процес побудови відповідних моделей: архітектурних, інформаційних, компонентних та ін., так і потім забезпечити можливість генерації вихідного коду для їх програмної реалізації.

Одним з найбільш поширених та розвинених візуальних засобів моделювання всіх основних фаз ЖЦ ПС є уніфікована мова моделювання UML (Unified Modeling Language). Вона підтримує об'єктно-орієнтовану парадигму (object-oriented programming) розробки програмного забезпечення (ПЗ) і є одним з суттєвих компонентів уніфікованого процесу розробки ПЗ RUP (Rational Unified Process).

Для автоматизації процесів моделювання ПС із використанням UML існує багато відповідних CASE-засобів, одним з яких є середовище Visual Paradigm (VP), архітектура і основні функціональні можливості якого розглянуто в лабораторній роботі 1.

Застосовуючи інструментарій VP, системні аналітики та розробники ПЗ можуть послідовно створювати моделі відповідної ПС на концептуальному рівні. Ці питання розглянуто в лабораторній роботі 2, потім на логічному рівні - у лабораторних роботах 3-4, і на фізичному рівні опису компонентів ПЗ - у лабораторній роботі 5.

Знання та вміння, отримані при виконанні цього лабораторного практикуму, можуть бути використані студентами таких напрямів підготовки як 121 – «Інженерія програмного забезпечення» та 122 – «Інформаційні управляючі системи та технології».

ЛАБОРАТОРНА РОБОТА 1

ВИВЧЕННЯ АРХІТЕКТУРИ, ВІЗУАЛЬНИХ ІНТЕРФЕЙСІВ ТА ІНСТРУМЕНТАЛЬНИХ ЗАСОБІВ CASE-СИСТЕМИ VISUAL PARADIGM

Мета – вивчити такі запитання:

- призначення системи, склад та перелік її основних компонентів;
- вивчення основних понять: пакети, класи, відношення (зв'язки) компоненти, стереотипи, представлення програмної системи як сукупності UML-діаграм;
- інтерфейси та можливості пакета UML Diagrams.

1.1. Загальні відомості

Призначення системи, склад та перелік її основних компонентів

Visual Paradigm для UML – це інструмент для проектування програмного забезпечення (ПЗ) будь-якої складності, який підтримує UML 2, SysML і Business Process Modeling Notation (BPMN) від Object Management Group (OMG). Дистрибутив пакета Visual Paradigm можна вільно отримати на офіційному Web-сайті розробника [1].

На цій сторінці необхідно обрати продукт Visual Paradigm for UML (VP-UML), а саме безкоштовну версію (community edition), завантажити її та пройти процедуру реєстрації для отримання ліцензійного ключа. Після інсталяції необхідно імпортувати ліцензійний ключ до системи.

VP-UML надає розробнику ПЗ засоби для аналізу і моделювання програмної системи (ПС), що розробляється. Крім підтримки моделювання різних аспектів функціонування ПС, він забезпечує генерацію коду, а також побудову звітів та деякі інші можливості. Однією з корисних особливостей VP-UML є функція зворотного інжинірингу проекту (reverse code engineering), тобто генерування UML-діаграми з існуючого програмного коду, але ці можливості є доступними лише в повній, комерційній версії системи.

До основних компонентів інтерфейсу CASE-засобу VP-UML можна віднести такі (рис. 1.1): 1) головне меню; 2) панель UML – вона, окрім стандартних функцій (створення, відкриття, друку проекту тощо), надає можливість вибрати тип нової ді-

аграми; 3) вікно документації – це область для ведення документації проекту; 4) браузер проекту – містить дерево проекту: наявні діаграми та пакети; 5) робоча область – у ній безпосередньо будуються діаграми; 6) панель інструментів діаграми – це набір візуальних засобів для побудови окремих діаграм; 7) вікно виводу повідомлень – область, де відображуються необхідні системні повідомлення.

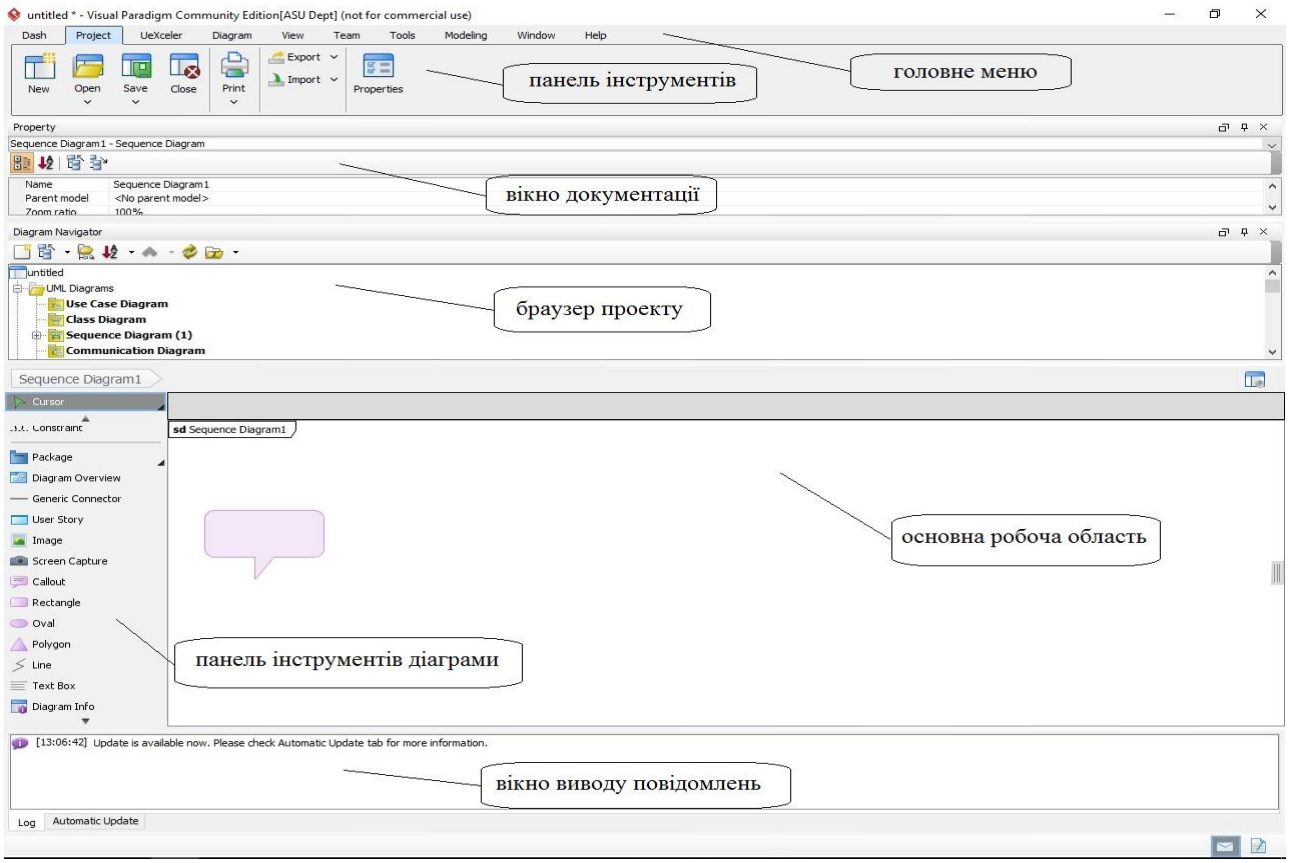


Рисунок 1.1 – Загальний вигляд інтерфейсу користувача CASE-засобу VP-UML

Вивчення основних понять uml-діаграм: пакети, класи, відношення (зв'язки), компоненти, стереотипи

У цьому циклі лабораторних робіт використовується нотація UML-діаграм версії UML 2.0 [2].

Конструктивні блоки UML

Сутності (entity) є основою всіх типів UML-моделей. Прив'язку сутностей одну до одної забезпечують *відношення* (relationship), а діаграми групують сутності у певні набори (конфігурації). У UML подані чотири типи сутностей: *структурні* (struc-

tural), *поведінкові* (behavioral), сутності, що угруповують інші сутності (grouping entity), та анотації (annotation). Графічне зображення окремих типів сутностей, прийняте в UML, наводиться нижче.

Єдиним представником сутності, що угруповує інші сутності, є *пакет* (package). Пакет – це механізм загального призначення для організації елементів у вигляді єдиної групи сутностей. Структурні, поведінкові та навіть інші що групують сутності, можуть бути поміщені всередині певного пакета.

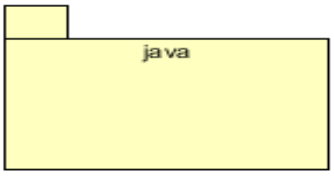


Рисунок 1.2 – Позначення сутності – пакета

Анотації є пояснювальною та коментованою частиною UML. Єдиним її типом є *примітка* (note). Примітка з'єднується пунктирною лінією із сутністю, до якої вона належить.

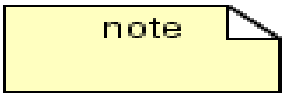


Рисунок 1.3 – Позначення примітки

Клас (class) у мові UML використовується для визначення множини об'єктів, які мають однакову структуру, поведінку та відношення з об'єктами з інших класів. Графічно клас зображується у вигляді прямокутника, який додатково може бути розділений горизонтальними лініями на розділи або секції (рис. 1.4). У цих розділах указуються ім'я класа, атрибути (змінні) та операції (методи).

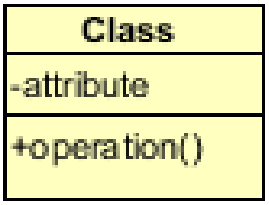


Рисунок 1.4 – Позначення класу

Атрибут (attribute) – у другій зверху секції прямокутника класу записуються його атрибути (attributes) або властивості. У мові UML прийнята визначена стандартизація запису атрибутів класу, яка підпорядковується деяким синтаксичним правилам. Кожному атрибуту класу відповідає окремий рядок тексту, який складається із квантора видимості атрибута, імені атрибута, його кратності, типу значень атрибута й можливо, його початкового значення:

<квантор видимості><ім'я атрибута>[кратність]:

<тип атрибута> = <початкове значення>{строка-властивість}.

Квантор видимості може набувати одне із трьох можливих значень та, відповідно, відображається за допомогою спеціальних символів:

1. Символ «+» означає атрибут з областю видимості типу загальнодоступний (public). Атрибут із цією областю видимості доступний або видний із будь-якого іншого класу пакета, в якому визначена діаграма.
2. Символ «#» означає атрибут з областю видимості типу захищений (protected). Атрибут із цією областю видимості недоступний або невидимий для всіх класів, за виключенням підкласів цього класу.
3. Знак «-» означає атрибут з областю видимості типу закритий (private). Атрибут із цією областю видимості недосяжний або невидимий для всіх класів без виключення.

Метод (method) – у третій зверху секції прямокутника класу записуються операції або методи класу. Метод являє собою деякий сервіс, що кожний екземпляр класу надає за умови виклику цього методу. Сукупність операцій характеризує функціональний аспект поведінки класу. Запис операцій класу у мові UML також стандартизований та підпорядкований певним синтаксичним правилам. При цьому кожній

операції класу відповідає окремий рядок, який складається із квантора видимості операції, імені операції, типу даних, значення, яких повертає операція і, можливо, строка-властивостей цієї операції, а саме:

<квантор видимості>< ім'я атрибута>(список параметрів):

<вираз типу повертального значення>{строка-властивість}.

Квантор видимості, як і у випадку атрибутів класу, може приймати одне із трьох можливих значень та, відповідно, відображається за допомогою спеціального символу. Символ «+» визначає операцію із областю видимості типу загальнодоступний (public). Символ «#» визначає операцію із областю видимості типу захищений (protected). І, нарешті, символ «-» використовується для визначення операції з областю видимості типу закритий (private).

Крім внутрішньої побудови або структури класів, на відповідній діаграмі вказуються різноманітні *відношення* або *зв'язки* (relationship) між класами. При цьому сукупність типів таких відношень в UML визначена семантикою типів цих відношень. Базовими відношеннями або зв'язками у мові UML є:

1) залежності (dependency relationship);
2) асоціації (association relationship);
3) узагальнення (generalization relationship);
4) агрегації (aggregation relationship);
5) композиції (composition relationship) як окремий випадок агрегації.

Відношення залежності у загальному випадку вказує на деяке семантичне відношення між двома елементами моделі або між двома множинами таких елементів, яке не є відношенням асоціації, узагальнення або агрегації. Воно стосується тільки самих елементів моделі і не вимагає множина окремих прикладів для пояснення свого сенсу. Відношення залежності використовується у такій ситуації, коли деяка зміна одного елемента моделі може потребувати зміни іншого залежного від нього елемента моделі.

Відношення залежності графічно зображується пунктирною лінією між відповідними елементами із стрілкою на одному з її кінців («->» або «<-»). На діаграмі класів це відношення зв'язує окремі класи між собою, при цьому стрілка направлена від класу-клієнта залежності до незалежного класу або класу-джерела (рис. 1.5). На

ньому зображено два класи: клас_А і клас_Б, при цьому клас_Б є джерелом деякої залежності, а клас_А – клієнтом цієї залежності.

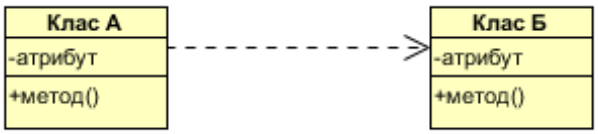


Рисунок 1.5 – Позначення відношення залежності

Відношення асоціації відповідає наявності деякої функціональної залежності між класами. Це відношення позначається суцільною лінією з додатковими спеціальними символами, які характеризують окремі властивості конкретної асоціації. Як додаткові спеціальні символи можуть використовуватися імена асоціації, а також імена і кратність класів-ролей асоціації. Ім'я асоціації є необов'язковим елементом її позначення. Якщо воно задане, то записується із заголовної (великої) букви поряд з лінією відповідної асоціації. Найбільш простий випадок цього відношення – це бінарна асоціація.

Нехай у деякій UML-моделі є клас «А» (співробітник) і клас «Б» (компанія). Тоді прикладом відношення асоціації між ними буде твердження «Розробник працює в Компанії» і відповідна діаграма класів показана на рис. 1.6.

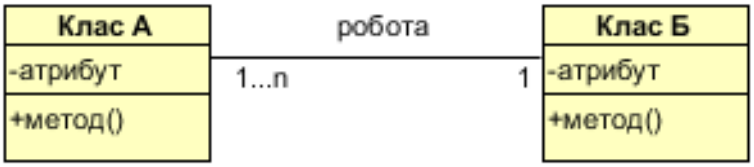


Рисунок 1.6 – Позначення відношення асоціації

Відношення агрегації має місце між декількома класами в тому випадку, якщо один з класів є деякою сутністю, що включає як складові частини певні інші сутності.

Це відношення має фундаментальне значення для опису структури складних систем, оскільки застосовується для представлення системних взаємозв'язків типу «частина-ціле» (part of). Розкриваючи внутрішню структуру системи, відношення агрегації показує, з яких компонентів складається система і як вони зв'язані між собою. З погляду моделі окремі частини системи можуть виступати як у вигляді елементів, так і у вигляді підсистем, які, у свою чергу, теж можуть утворювати складні компоненти або підсистеми. Це відношення за своєю суттю описує декомпозицію або розбиття складної системи на простіші складові частини, які також можуть бути піддані декомпозиції, якщо в цьому виникне необхідність у подальшому. Наприклад: розподіл комп'ютера на складові частини – системний блок, монітор, клавіатура, мишка (рис. 1.7). В UML 2.0 визначено дві форми відношення «частина-ціле»: загальна (агрегація) та приватна (композиція).

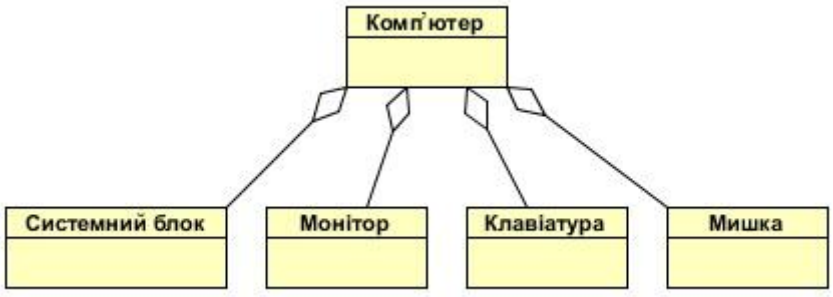


Рисунок 1.7 – Позначення відношення агрегації

Композиція (composition) – це окремий випадок агрегації, що характеризується двома додатковими обмеженнями. Складова частина може належати не більш ніж одному агрегату. Таким чином, у композиції малося на увазі, що частини належать цілому, тому видалення об'єкта-агрегата автоматично викликає видалення всіх його складових, якщо він утворює їхню композицію. Для позначення композиції використовується невеликий зафарбований ромбик, який ставиться поряд з класом-агрегатом (для агрегації, що не є композицією, використовується не зафарбований ромбик).

Наприклад: компанія складається з відділень, які, у свою чергу, складаються з відділів. Компанія побічно є композицією відділів (рис. 1.8).

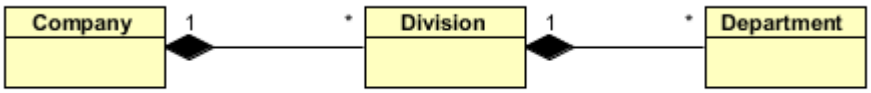


Рисунок 1.8 – Позначення відношення композиції

Відношення узагальнення є таксономічнім відношенням між більш загальним елементом (батьком або предком) і більш спеціальним елементом (дочірнім або нащадком). Це відношення може використовуватися для представлення взаємозв'язків між пакетами, класами, варіантами використання і іншими елементами мови UML. На діаграмі класів це відношення описує ієрархічну побудову деяких класів і наслідування їх властивостей (через відповідні атрибути) та поведінки (за рахунок методів). При цьому передбачається, що клас-нащадок володіє всіма властивостями і поведінкою класу-предка, а також має свої власні властивості і поведінку, які відсутні у класу-предка. На діаграмах відношення узагальнення позначається суцільною лінією з трикутною стрілкою на одному з кінців (рис. 1.9). Стрілка вказує на узагальнюючий клас (клас-предок або суперклас), а її відсутність – на спеціалізований клас (клас-нащадок або підклас).

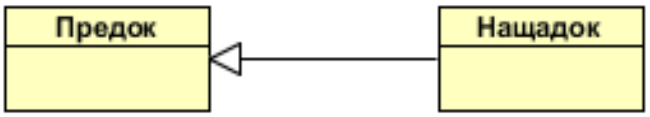


Рисунок 1.9 – Позначення відношення узагальнення (наслідування)

Компонент (component) – спеціальний термін, який застосовується в мові UML для представлення окремої сутності на фізичному рівні моделювання ПС. Компонент реалізує деякий набір інтерфейсів і використовується для загального позначення варіантів можливої програмної реалізації моделі, наприклад, це: бінарний код, що виконується (*.exe – файл, java-аплет і інш.), бібліотека, що динамічно підк-

лючається (*.ddl – файл), програмний скрипт, написаний деякою мовою опису сценаріїв (*.asp, *.php – файли) та ін. Для графічного представлення компонента у UML 2.0 використовується спеціальний символ – прямокутник, що містить у собі праворуч дрібніший прямокутник зі вставленими у нього двома ще дрібнішими прямокутниками (рис. 1.10). Всередині цього прямокутника записується ім'я компонента і, можливо, деяка додаткова інформація.

Рисунок 1.10 – Позначення компонента (фізичної сутності)

Стереотип (stereotype) – це механізм утворення нового типу (виду) класів в мета-моделі UML. На рис. 1.11 зображено існуючий список наявних стереотипів певного класу.

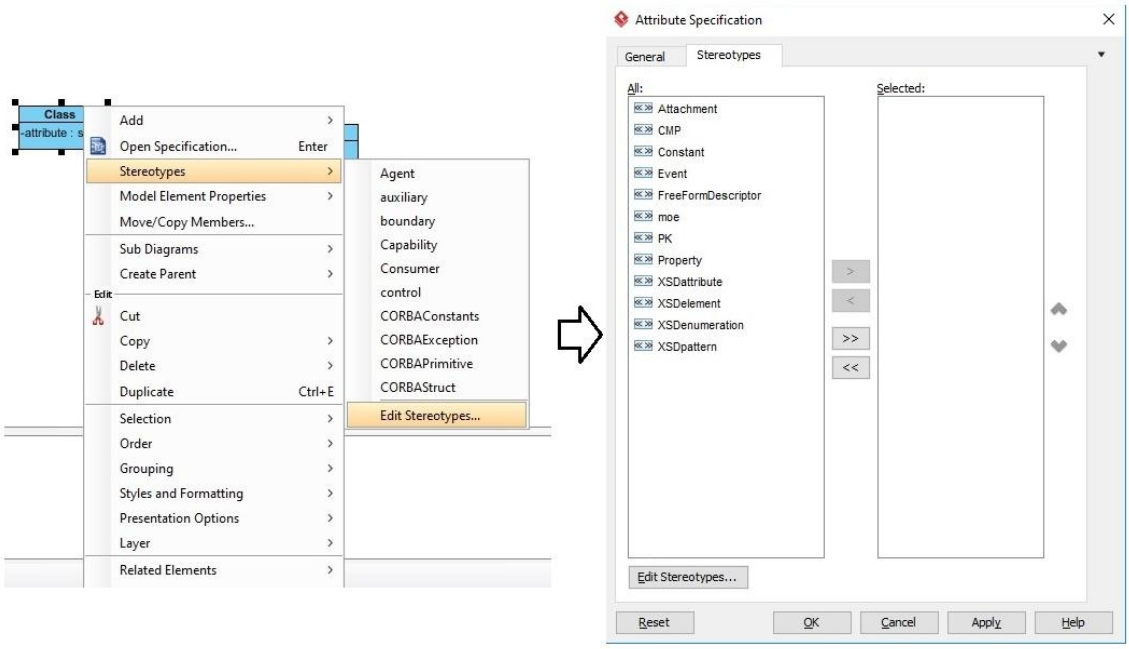


Рисунок 1.11 – Приклад списку наявних стереотипів

Для того щоб додати потрібний стереотип у список, необхідно натиснути правою кнопкою миші на клас. Далі на вкладці *Stereotypes* натиснути на кнопку *Edit Stereotypes* та у вікні, що з'явилося, натиснути на кнопку *Add*. Потім необхідно ввести назву нового стереотипу, також є можливість встановлення додаткових параметрів (таких, як тип шрифту, логотип і та ін.). Для додавання натискаємо *OK*. На рис. 1.12 наведений приклад компонента *Server Machine* з доданим стереотипом *Intel*.

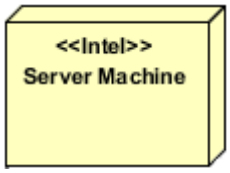


Рисунок 1.12 – Позначення компонента, що має стереотип

У середовищі VP / UML існує 13 типів діаграм у межах пакета UML Diagrams, а саме:

1. UseCase diagram (діаграма прецедентів).
2. Class diagram (діаграма класів).
3. Sequence diagram (діаграма послідовностей).
4. Communication diagram (діаграма комунікації).
5. State machine diagram (діаграма кінцевого автомату).
6. Activity diagram (діаграма активності).
7. Component diagram (діаграма компонентів).
8. Deployment diagram (діаграма розгортання).
9. Package diagram (діаграма пакетів).
10. Object diagram (діаграма об'єктів).
11. Composite structure diagram (діаграма композитної структури).
12. Timing diagram (часова діаграма).
13. Interaction overview diagram (діаграма огляду взаємодії).

Усі ці діаграми у VP / UML розбиті на 4 різних групи (пакети), що знаходяться на вкладці UML головного меню системи (рис. 1.13):

- моделювання прецедентів (Use Case Modeling);
- структурне моделювання (Structural Modeling);
- моделювання поведінки (Behavioral Modeling);
- моделювання архітектури (Architectural Modeling).

За допомогою цих діаграм можливо розробляти моделі ПЗ на концептуальному, логічному та фізичному рівнях проектування відповідної ПС.

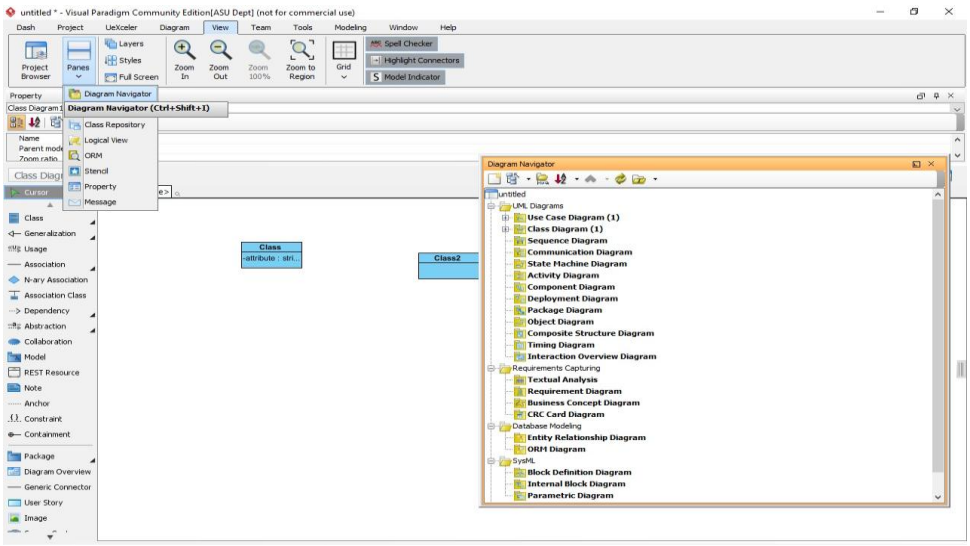


Рисунок 1.13 – Основні групи (пакети) UML-діаграм

Пакет Use Case Modeling

Цей пакет може містити одну або сукупність діаграм варіантів використання (або прецедентів – Use Case diagram), які використовуються для концептуального рівня проектування ПС. Приклад інтерфейсу цього пакета показано на рис. 1.14. На панелі інструментів цього пакета присутні специфічні для цього типу діаграм візуальні інструменти: *Актори, Варіанти використання, Асоціації* тощо.

Пакет Structural Modeling

Цей пакет може містити одну діаграму або сукупність діаграм логічного рівня проектування ПС, які забезпечують моделювання її структурних (тобто статичних) аспектів (рис. 1.15).

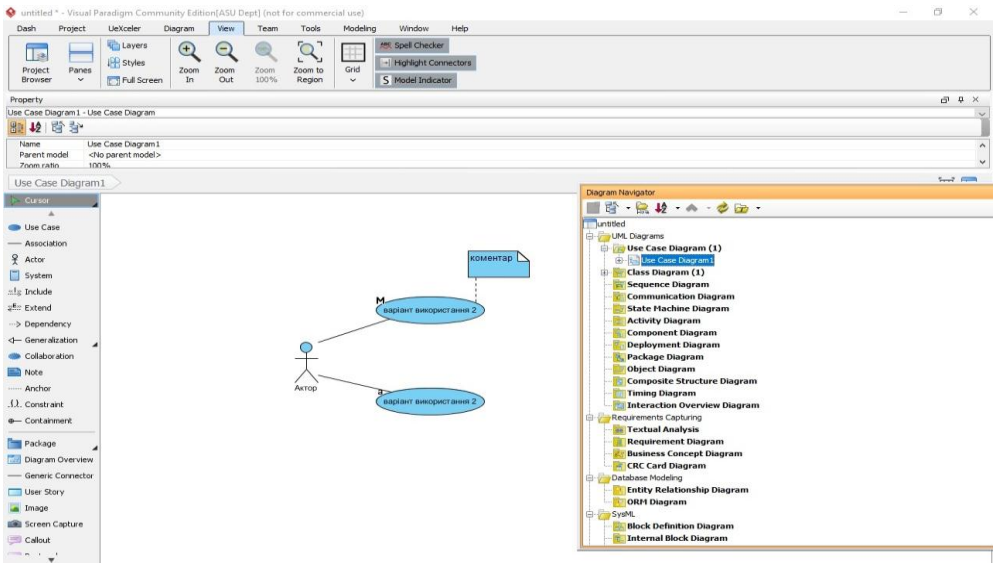


Рисунок 1.14 – Приклад інтерфейсу в пакеті «Use Case Diagram»

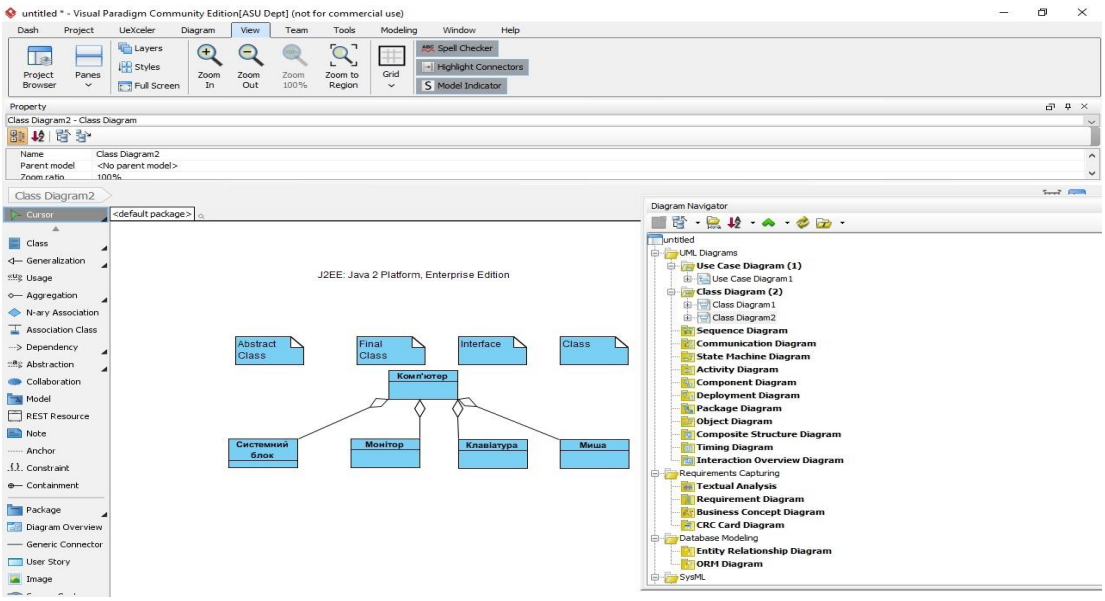


Рисунок 1.15 – Приклад інтерфейсу в пакеті ”Structural Modeling”

Це такі діаграми як

– діаграма класів (Class diagram);

– діаграма композитної структури (Composite structure diagram),

діаграма об’єктів (Object diagram). Щоб додати до логічного рівня нову діаграму, потрібно на відповідному пакеті натиснути праву кнопку мишки і у меню, яке

з’явиться, обрати опцію New Diagram та обрати потрібну діаграму. Ці діаграми будуть більш детально розглянуті у лабораторних роботах 2–5.

Пакет Behavioral Modeling

Діаграми цього пакета описують динамічні аспекти програмної системи, що моделюється: процес функціонування елементів системи, включаючи їхні методи та взаємодію між ними, а також процес зміни станів окремих елементів і системи у цілому. До нього входять такі діаграми: Sequence diagram (діаграма послідовності), Communication diagram (діаграма комунікації), Activity diagram (діаграма активності), State Machine diagram (діаграма кінцевого автомату), Timing diagram (діаграма часу), Interaction Overview diagram (діаграма обзору взаємодії) (рис. 1.16).

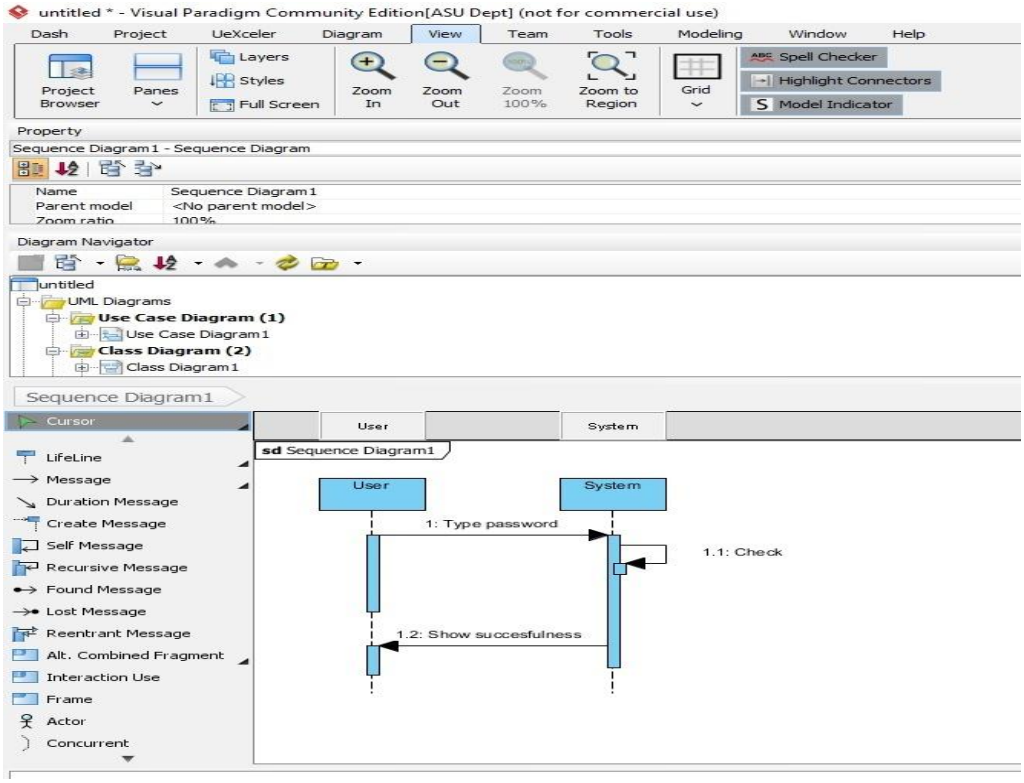


Рисунок 1.16 – Приклад інтерфейсу в пакеті «Behavioral Modeling»

Пакет Architectural Modeling

На цьому рівні містяться такі діаграми: Component diagram (діаграма компонентів), Deployment diagram (діаграма розгортання), Package diagram (діаграма пакетів) апаратного і програмного забезпечення системи на етапі її фізичного проектування.

Для того щоб створити специфікацію будь-якого елемента цих діаграм, треба натиснути правою кнопкою мишки на потрібному елементі діаграми та вибрати пункт Open Specification у списку, або ж нажати лівою кнопкою мишки на елемент та натиснути клавішу Enter (рис.1.17). Пакети Use Case Modeling, Structural Modeling та Architectural Modeling описують виключно статичні аспекти побудови програмної системи, що моделюється, на відміну від пакета Behavioral Modeling, який описує динамічні аспекти цієї ПС.

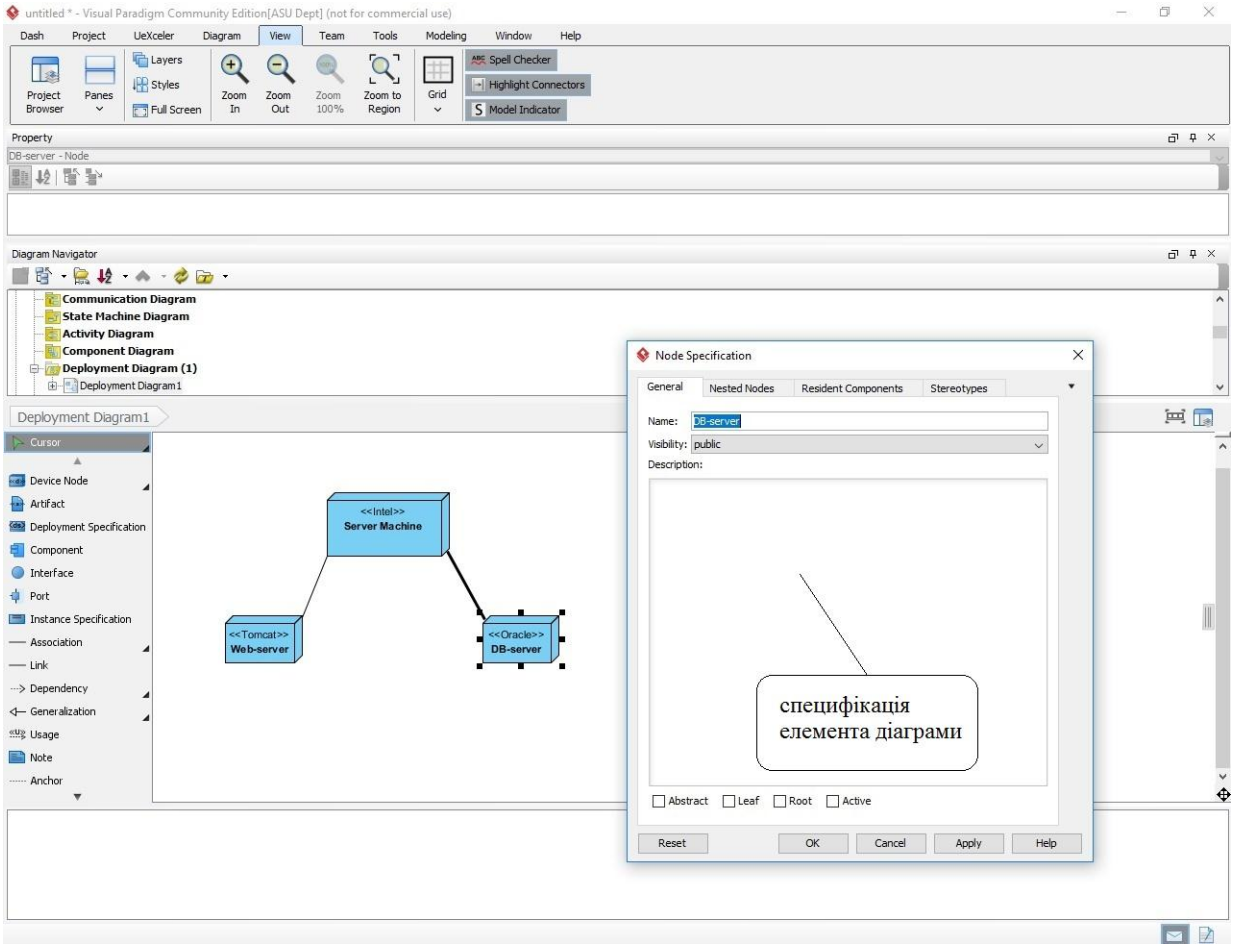


Рисунок 1.17 – Приклад інтерфейсу в пакеті «Architectural Modeling»

1.2. Виконання лабораторної роботи

1) У середовищі системи VP-UML розглянути всі його основні режими та функціональні можливості;

2) створити власну стислу інструкцію для роботи з VP-UML;

3) обрати певну предметну область розробки ПС (за індивідуальним завданням викладача).

1.3. Результати виконання роботи та оформлення звіту

Зміст звіту:

1) титульний аркуш;
2) короткі теоретичні відомості;
3) результати практичного виконання, висновки та рекомендації.

Контрольні запитання

1. Назвіть призначення та основні компоненти архітектури CASE-засобу Visual Paradigm / UML.
2. Назвіть та поясніть, для чого застосовують основні конструктивні елементи (абстракції) UML.
3. Навести типи зв'язків в UML–діаграмі класів і надайте їх стислу характеристику.
4. В чому полягає різниця між відношеннями агрегації та узагальнення? Наведіть конкретні приклади цих типів відношень у деякій ПрО.
5. Що є спільного і у чому полягає різниця між відношеннями агрегації та композиції? Наведіть приклади цих типів відношень.
6. Які основні пакети існують у середовищі Visual Paradigm та для чого використовуються?
7. Які діаграми містяться в пакеті Use Case Modeling?
8. Які діаграми містяться в пакеті Structural Modeling?
9. Які діаграми містяться в пакеті Behavioral Modeling?

10) Які діаграми містяться в пакеті Architectural Modeling?

11) Назвіть існуючі рівні проектування ПЗ та види UML-діаграм, що використовуються для моделювання властивостей ПС на кожному з них.

12) Які пакети (діаграми) застосовуються для моделювання статичних аспектів побудови ПС, і які – для моделювання її динамічних характеристик?

ЛАБОРАТОРНА РОБОТА 2

АНАЛІЗ ВИМОГ ТА РОЗРОБКА UML–ДІАГРАМ КОНЦЕПТУАЛЬНОГО РІВНЯ ПРОЕКТУВАННЯ ПРОГРАМНОЇ СИСТЕМИ

Мета роботи – ознайомитись з методикою побудови UML-діаграм концептуального рівня проектування програмного забезпечення.

2.1. Загальні відомості

На концептуальному рівні проектування ПС відповідно до раніше визначених та задокументованих вимог (наприклад, з використанням методології RUP або Scrum [3]) встановлюються найбільш загальні характеристики системи, що розробляються, до яких належать:

- основні варіанти використання функцій ПС;
  типи її користувачів;
- послідовність взаємодії окремих користувачів і системи або окремих її підсистем (так звані сценарії);
- основні групи програмних компонентів, що реалізують певні функції системи, з рознесенням їх по рівням логічного упорядкування та взаємодії, тобто архітектура ПС.

При об’єктно-орієнтованому підході до проектування ПЗ з використання UML на концептуальному рівні моделювання використовуються такі діаграми [3–4]:

- діаграми варіантів використання (або прецедентів – use case diagram);
- діаграми послідовностей (sequence diagram);
- діаграми пакетів (package diagram).

Як додатковий засіб до діаграм прецедентів, який дозволяє розширити уявлення розробників щодо подальшої програмної реалізації системи, використовують діаграми стійкості (robustness diagram).

Діаграми прецедентів (use case diagram)

Під прецедентом (case) треба розуміти фіксовану послідовність дій між користувачами та ПС або між окремими підсистемами, яка забезпечує досягнення постав-

лених цілей. В обох випадках вони позначаються терміном «актор» (actor). Так, наприклад, процедура авторизації користувача на деякому інтернет-ресурсі є певним прецедентом. Для нього учасниками будуть система авторизації та користувач, а результатом – перевірка наявності в системі реєстраційного запису (account) користувача.

Кожна модель відображає взаємозалежність прецедентів та акторів, яка визначена такими типами:

- асоціація (association);
- включення (include);
- розширення (extend);
- узагальнення (generalization).

Усі ці чотири типа залежностей надані на рис. 2.1.

Відношення *асоціації* є основним типом залежностей між акторами та прецедентами, що вказує на те, що деякий актор відіграє певну роль у взаємодії з системою. На діаграмі асоціацій позначається суцільною прямою лінією і позначається відповідною назвою. На рис. 1 це відношення «realize» між актором «User» та прецедентом «Authorization».

Відношення *включення* дає можливість відобразити те, що деякі функції одного прецеденту А обов'язково мають бути задіяні при використанні іншого прецеденту В. На діаграмі включення позначається пунктирною прямою лінією зі стрілкою на кінці, яка направлена від того прецеденту, що включає в себе деякий інший і має назву «include». На рис. 2.1 це відношення між прецедентом «Authorization» та «Login PW».

Відношення *розширення* дає можливість відобразити те, що при опрацюванні системою деякого прецеденту X є можливим (але не обов'язковим) використання функціональності іншого прецеденту Y. На діаграмі включення позначається пунктирною прямою лінією зі стрілкою на кінці, яка направлена від того прецеденту, що є розширенням та має назву «extend». На рис. 2.1 це відношення між прецедентом «Authorization» та «PW Recovery».

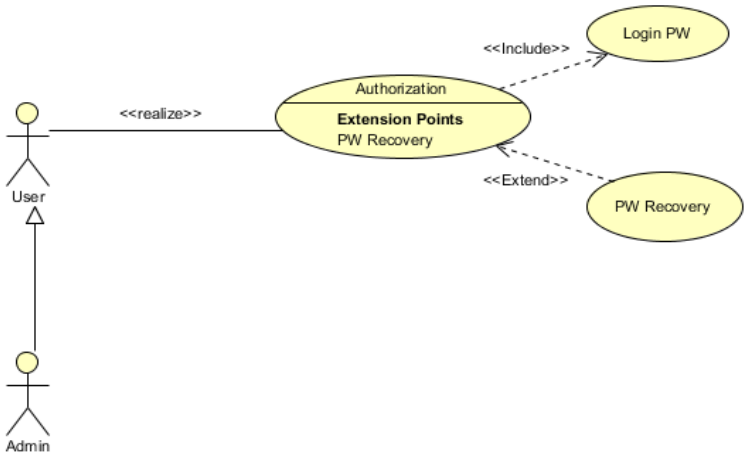


Рисунок 2.1 – Приклад побудови UML-діаграми прецедентів

Відношення *узагальнення* дозволяє показати певну ієрархію акторів та самих прецедентів, тобто позначити, який з них є супер-типом, а який – підтипом певного класу об’єктів. На діаграмі це позначається суцільною прямою лінією з трикутною стрілкою на кінці, яка направлена до прецеденту, що є супер-типом. На рис. 2.1 це відношення між акторами «Admin» та «User».

Діаграма стійкості (robustness diagram)

Прецеденти не можуть бути безпосередньо трансформовані у відповідні програмні об’єкти (класи). Для цього має бути використано певний проміжний рівень опису прецедентів, який дозволяє врахувати деякі особливості наступної програмної реалізації, наприклад, необхідність інтерфейсу користувача або наявність деяких функцій обробки бізнес-логіки. Одним з таких засобів подальшої деталізації моделі прецедентів є *діаграма стійкості* (ці діаграми належать до процесу проектування ПЗ за стандартом ICONIX [3]).

Діаграма стійкості розробляється на базі шаблону (або патерну – pattern) проектування MVC (Model-View-Controller) [4]. Тобто в системі, що розробляється, мають бути наявні три типи програмних об’єктів:

- *Model* – це об’єкти, які моделюють дані предметної області;
- *View* – це об’єкти, які реалізують відображення даних із моделі;
- *Controller* – це об’єкти, які обробляють дані моделі для подальшого їх відображення.

Основна ідея шаблону MVC полягає у відокремленні даних від їх відображення. Таким чином, якщо у процесі розробки виникне потреба у зміненні моделі даних, це ніяк не впливає на відображення (не треба буде нічого змінювати у компонентах View). У діаграмі стійкості є свої окремі позначки для відображення елементів моделі MVC:

- – граничні об’єкти, які слугують для відображення даних (View)
- – об’єкти обробки даних (Controller)
- – об’єкти даних (Model)

На рис. 2.2 побудовано діаграму стійкості для прецеденту реєстрації нового користувача системи, де граничним об’єктом *Registration form* є певна діалогова форма (HTML-сторінка), яка з’являється у вікні браузера на комп’ютері-клієнті системи. Об’єктом-контролером *Add new* може бути, наприклад, програмний скріпт на мові PHP, який виконується на Web-сервері Apache і який за допомогою функцій обробки SQL-запитів працює з об’єктом даних Account – це, у свою чергу, може бути таблиця БД, що знаходиться під керуванням СКБД MySQL.

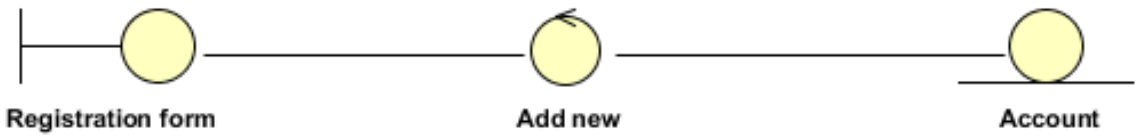


Рисунок 2.2 – Діаграма стійкості для прецеденту реєстрації нового користувача

Декілька прецедентів можуть бути пов’язані на одній діаграмі стійкості (рис. 2.3). Так, об’єкт даних Account використовується для прецеденту входу до системи. Можливим варіантом програмної реалізації є розширення прецеденту реєстрації, додавши до нього контролер *Find* для пошуку вже існуючого реєстраційного запису, тобто за його допомогою при реєстрації система буде перевіряти, чи вже існує реєстраційний запис, який хоче додати користувач.

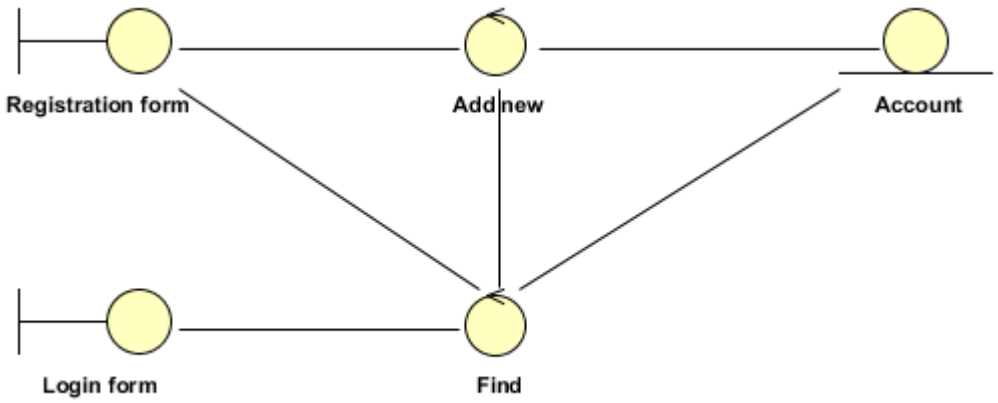


Рисунок 2.3 – Діаграма стійкості для двох прецедентів

Для відображення цих візуальних елементів у VP необхідно вибрати відповідні стереотипи для об'єктів: для View – це boundary, для Controller – це control, для Model – це entity.

Діаграми послідовності (sequence diagram)

Окрім розглянутих вище діаграм прецедентів та діаграм стійкості, для кожного прецеденту може бути побудована відповідна діаграма послідовності. Цей тип діаграм визначає певні сценарії обміну повідомленнями (або викликами відповідних функцій) між різними логічними (архітектурними) компонентами ПС.

Кожна діаграма послідовності розподіляється на декілька частин – за кількістю учасників прецеденту. Наприклад, у прецеденті Login беруть участь два компоненти: користувач (User) та система (System) (рис. 2.4). Ці компоненти позначаються прямокутниками, які знаходяться у верхній частині діаграми.

Кожен такий прямокутник має свою *лінію життя*: це вертикальна лінія, яка направлена донизу від цього компонента та відображає час його активності у системі. Передача повідомлень, що ініціюють певну функцію, позначається за допомогою горизонтальної стрілочки між лініями життя відповідних компонентів (1: Type Password), а результат виконання цієї функції показано пунктирною стрілочкою

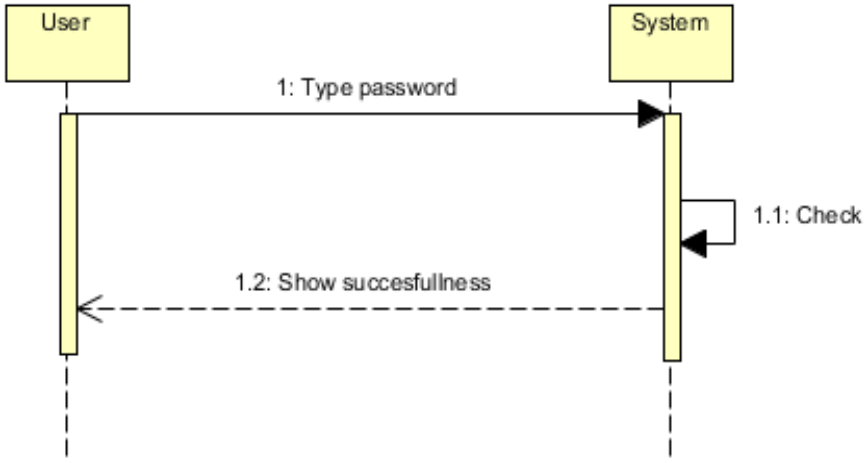


Рисунок 2.4 – Діаграма послідовності для прецеденту Login

повернення управління по цьому сценарію (1.2. Show successfulness).

Відповідно до діаграми послідовності прецеденту Login маємо:

1. Користувач вводить логін та пароль.

2. Система перевіряє правильність вводу.

3. У разі розбіжності введених даних з існуючим реєстраційним записом вивести вікно помилки.

Діаграма пакетів (package diagram)

У процесі об’єктно-орієнтованої розробки ПС проводиться розподіл програмного рішення на окремі підсистеми, що уможливлює розподіл роботи між членами команди розробників і забезпечує подальше багаторазове використання програмного коду. Засобом моделювання в UML, який дає змогу агрегувати окремі компоненти в окремі групи (підсистеми) залежно від їх логічного призначення в системі, є поняття пакета (package).

Кожен пакет володіє всіма елементами, що належать до нього, при цьому сам пакет може входити до складу іншого пакета. Графічно пакет позначається відповідним прямокутником, який має унікальну назву та з’єднується з іншими пакетами за допомогою зв’язку Containment, що у VP позначає відношення вкладеності (рис. 2.5).

На цій діаграмі зображено пакет *Authorization*, до складу якого входять два інші пакети: *Login/PW* та *PW Recovery*.

Крім відношення вкладеності, деякі пакети можуть знаходитися у певній функціональній залежності один від одного, коли компоненти, що знаходяться в одному пакеті, використовуються в іншому. Така залежність позначається на діаграмі пунктирною лінією зі стрілкою на кінці. Для підвищення рівня інформативності діаграм пакетів на них (до речі, як і на інших UML-діаграмах) можливо використовувати коментарі.

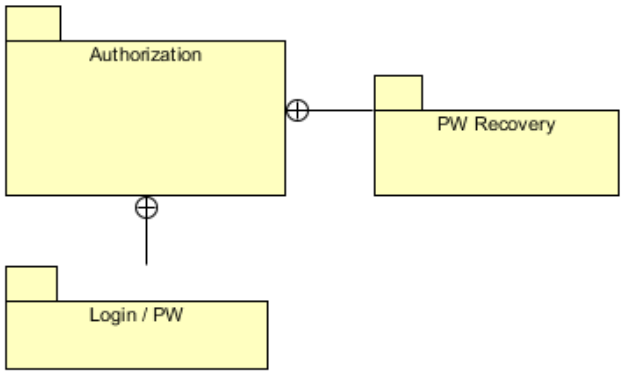


Рисунок 2.5 – UML- діаграма пакетів з відношенням вкладеності

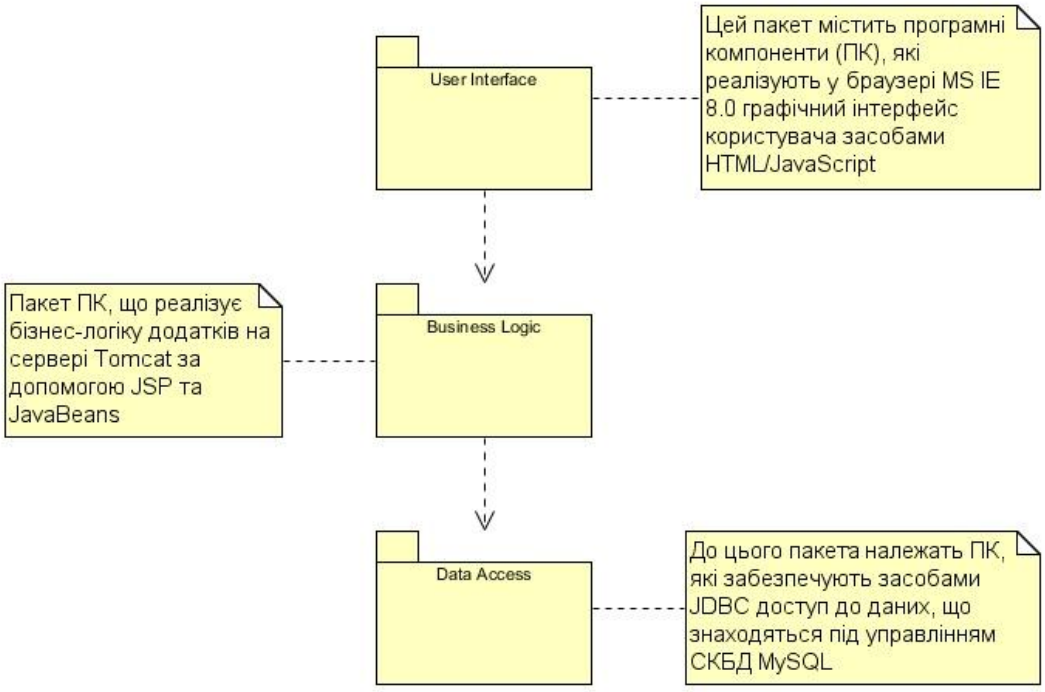


Рисунок 2.6 – Коментована UML-діаграма пакетів з відношенням залежності

Такий коментар позначається спеціальним графічним елементом, який містить відповідний текст. На рису. 2.6 наведено приклад коментованої UML-діаграми паке-

тів з відношення залежності, що відображає типову архітектуру Web-базової програмної системи для роботи з базою даних та сервером додатків.

2.2. Виконання роботи

На основі аналізу та текстової специфікації вимог (див. л/р 2) потрібно:

- побудувати модель прецедентів для наданої предметної області;
- провести функціональний аналіз основних прецедентів та побудувати відповідні діаграми стійкості;
- для основних прецедентів побудувати діаграми послідовності;
- побудувати загальну діаграму пакетів для всієї ПС, що має бути спроектована.

2.3. Висновки до виконаної роботи та оформлення звіту

Зміст звіту:

1) титульний аркуш;
2) короткі теоретичні відомості;
3) результати практичного виконання, висновки та рекомендації.

Контрольні запитання

1. Для чого у процесі проектування ПЗ використовують моделі прецедентів? Що має відобразити така модель ?

2. На підставі якої інформації будується модель прецедентів?

3. Що таке «актор» та «прецедент»? Які типи відношень можуть бути визначені між ними?

4. Який тип відношень між акторами і прецедентами є найбільш поширеним? Навести приклади.

5. У чому полягає спільність і у чому є різниця між відношенням включення та розширення на діаграмі прецедентів? Навести приклади.

6. Для чого використовується відношення узагальнення? Навести приклади.

7. Який сенс у процесі моделювання ПЗ має застосування діаграм стійкості?

8. Назвіть основні графічні елементи, що використовуються при побудові діаграми стійкості. Який шаблон проектування ПЗ вони використовують?

9. Для чого при проектуванні ПЗ використовуються діаграми пакетів? Які типи відношень існують між окремими пакетами?

ЛАБОРАТОРНА РОБОТА 3

РОЗРОБКА UML-ДІАГРАМ ЛОГІЧНОГО РІВНЯ ПРОЕКТУВАННЯ ПРОГРАМНОЇ СИСТЕМИ: МОДЕЛЮВАННЯ СТАТИЧНИХ АСПЕКТІВ

Мета роботи – вивчити методику побудови UML-діаграм логічного рівня проектування ПЗ, які застосовуються для моделювання статичних аспектів побудови ПЗ.

3.1. Загальні відомості

Після виділення прецедентів, побудування діаграми послідовності та діаграми стійкості (див. л/р 2) можна приступати до більш детального проектування системи. На логічному рівні проектування ПЗ визначаються характеристики тих програмних компонентів, які входять до складу архітектури системи. При цьому мають бути розглянуті як *статичні*, так і *динамічні* аспекти цих компонентів. Для моделювання *статичних* (або структурних) аспектів ПЗ застосовуються такі UML-діаграми як [2-4]:

- діаграми класів (class diagram);
- діаграми об’єктів (object diagram).

Для моделювання *динамічних* аспектів ПЗ (тобто взаємодії компонентів) використовуються:

- діаграми станів (state chart diagram);
- діаграми діяльності (activity diagram);
- діаграми кооперації (communication diagram),

(про ці діаграми див. більш детально у л/р 4).

Діаграма класів (class diagram)

Клас (class) у мові UML потрібний для позначення сукупності об’єктів, які мають однакову структуру, поведінку та зв’язки з об’єктами з інших класів. Графічно клас зображується у вигляді прямокутника, який додатково має бути розділений горизонтальними лініями на розділи або секції. У цих розділах можуть вказуватися назва класу, атрибути (властивості) та операції (методи) цього класу.

Обов'язковим елементом позначення класу є його *ім'я* (name), яке має бути унікальним у межах відповідного пакета. На початкових етапах розробки діаграми окремі класи можуть позначатися простим прямокутником з указівкою лише його імені. У міру опрацювання окремих компонентів діаграми класів доповнюються атрибутами та методами.

У другій зверху секції прямокутника класу записуються його *атрибути* (attributes). Кожному атрибуту класу відповідає окремий рядок тексту, який складається з *квантора видимості* атрибуту, *імені* атрибуту, його *кратності*, *типу значень* атрибуту і, можливо, його певного значення за умовчанням (by default).

Квантор видимості може набувати одне з чотирьох можливих значень: *public, protected, private, package* і відображується за допомогою спеціальних символів (їх набір залежить від версії відповідного CASE-засобу, наприклад, Visual Paradigm). Атрибут типу *public* є доступним або видимим з будь-якого іншого класу всієї програмної системи. Атрибут типу *protected* є доступним тільки для всіх класів, які є нащадками цього класу. Атрибут типу *private* є недоступним для всіх інших класів без виключення. І, нарешті, атрибут типу *implementation* є доступним тим класам, що входять до складу пакета (package), в якому знаходиться цей клас.

Операції класу є певними сервісами обробки даних, вони подібно атрибутам також мають унікальні імена та квантори видимості (public, protected, private, implementation). На рис. 3.1 показано фрагмент діалогу в середовищі Visual Paradigm, в якому побудовано клас *User*, що має атрибути *LastName, FirstName* типу *public*, атрибут *Age* – типу *protected*, та атрибут *Status* – типу *private.* Відповідно, цей клас має також такі операції, як *GetStatus()* – типу *protected*, *UpdateStatus()* – типу *private*, та *PrintStatus()* – типу *package.*

Базовими типами відношень або зв'язками між класами в мові UML є: відношення *залежності (dependency relationship),* відношення *асоціації (association relationship),* відношення *узагальнення (generalization relationship)* та відношення агрегації *(aggregation relationship).* Кожен з них має власне графічне позначення на діаграмі класів.

Відношення *залежності* у загальному випадку вказує на деякий семантичний зв'язок між певними класами, який не є, відповідно, жодним з інших типів зв'язків (див. вище).

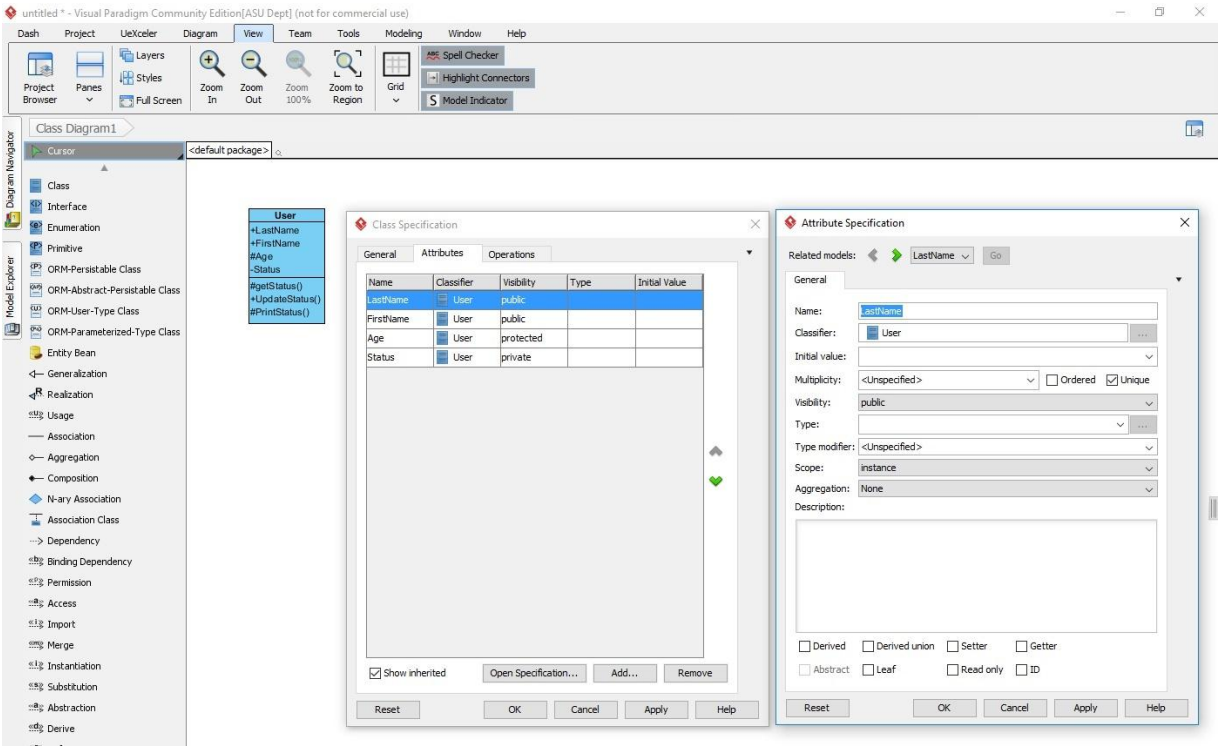


Рисунок 3.1 – Інтерфейс Visual Paradigm для визначення властивостей класу

Наприклад, відношення залежності використовується у такій ситуації, коли деяка зміна одного класу моделі може потребувати зміни певних параметрів (атрибутів і/або операцій) залежного від цього класу. Графічно це позначається пунктирною лінією між відповідними класами зі стрілкою на одному з її кінців, при цьому стрілка направлена від залежного класу, або від клієнта (client) до незалежного класу або класу-постачальника (supplier). На рис. 3.2 показано два класи: клас *User Profile* – це залежний клас-клієнт та клас *Application* – є клас-постачальник у цій залежності.

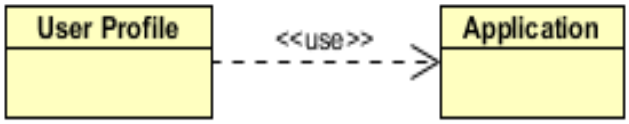


Рисунок 3.2 – Приклад відношення *залежності*

Відношення *асоціації* відповідає наявності деякої структурної залежності між класами. Воно позначається суцільною лінією з додатковими спеціальними символами, які характеризують окремі властивості конкретної асоціації, наприклад назва асоціації, її кратність та напрямок. Приклад асоціації наведено на рис. 3.3, де вона визначена між двома класами: класом *User* і класом *Account*. Вони зв'язані між собою бінарною асоціацією *subscribe*, назва якої вказана поряд з лінією асоціації.

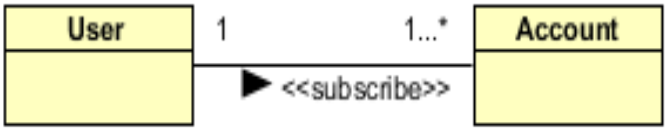


Рисунок 3.3 – Графічне зображення відношення бінарної асоціації

Додатково на діаграмі може бути кратність входження до відношення окремих класів. Вона позначається у вигляді інтервалу цілих чисел, який записується поряд з кінцем відповідної асоціації і може приймати наступні основні значення: ОДИН ДО ОДНОГО – 1:1, ОДИН ДО БАГАТЬОХ – 1:*, БАГАТО ДО БАГАТЬОХ – *:*. Крім того, є можливість вказувати мінімальне та максимальне значення кратності входження, наприклад, позначення кратності входження у вигляді: «0...1» – «1...*» , означає, що можливо один (або жоден) екземпляр класу на лівому боці відповідної асоціації залежить від щонайменше одного, або від декількох екземплярів класу на правому боці цієї ж асоціації.

Відношення *агрегації* має місце між декількома класами у тому випадку, якщо один з класів є таким, що включає в себе як складові частини деякі інші класи. Це відношення має фундаментальне значення для опису структури складних систем, оскільки застосовується для представлення системних взаємозв'язків типа «ціле – частина» («part – of»). Це відношення за своєю суттю описує декомпозицію або розбиття складної системи на простіші складові частини (компоненти), які також можуть бути піддані подальшій декомпозиції, якщо у цьому виникне необхідність. Типовим прикладом агрегації є взаємозв'язок, який має місце між об'єктом «Автомобіль» і такими його компонентами, як «Двигун», «Шасі», «Кузов».

Графічно відношення агрегації зображується суцільною лінією, один з кінців якої є незабарвленим усередині ромбом. Цей ромб указує на той з класів, який є агрегованим класом, а інші класи є його частинами.

Додатково на діаграмі агрегації також може бути вказана кратність входження відповідних класів (рис. 3.4).

Окремим випадком відношення агрегації є *композиція (composition)*, яка застосовується для моделювання класів, коли певні компоненти (частини) завжди знаходяться всередині цілого класу і специфіка взаємозв'язку між ними полягає у тому, що частини не можуть розглядатися у відриві від цілого, тобто зі знищенням цілого знищуються і всі його складові частини.

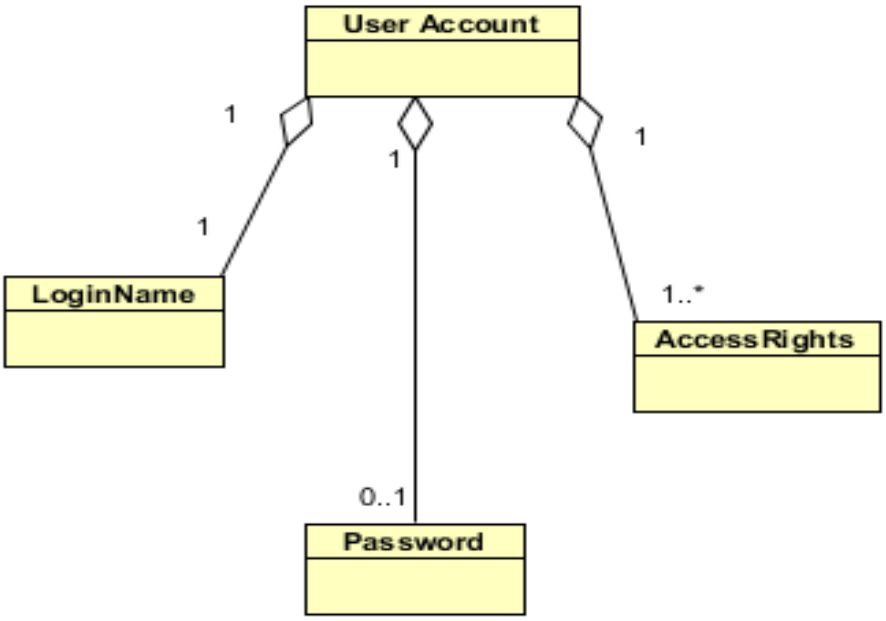


Рисунок 3.4 – Приклад діаграми класів для відношення агрегації

Прикладом цього відношення може бути вікно ділового інтерфейсу користувача певної програми, яке може складатися з заголовка, деяких кнопок управління, двох смуг прокручування та основної робочої області. З логічної точки зору подібне вікно є класом, а його компоненти є як підкласами, так і атрибутами або властивостями вікна. Графічно відношення композиції зображується суцільною лінією, один з кінців якої є *зафарбованим* усередині ромбом. Цей ромб вказує на той з класів, який є класом-композицією (рис. 3.5).

Відношення *узагальнення* є відношенням між деяким загальним класом – предком (parent class), або суперкласом, і більш специфічними класами – нащадком (child class). Це відношення може використовуватися для представлення взаємозв'язків між пакетами, класами, варіантами використання та іншими елементами мови UML.

Щодо діаграми класів відношення узагальнення описує ієрархічну структуру класів і вказує на спадкоємність їх властивостей і поведінки. При цьому передбачається, що клас-нащадок володіє всіма властивостями і поведінкою класу-предка, а також має свої власні властивості і поведінку, які відсутні у класу-предка.

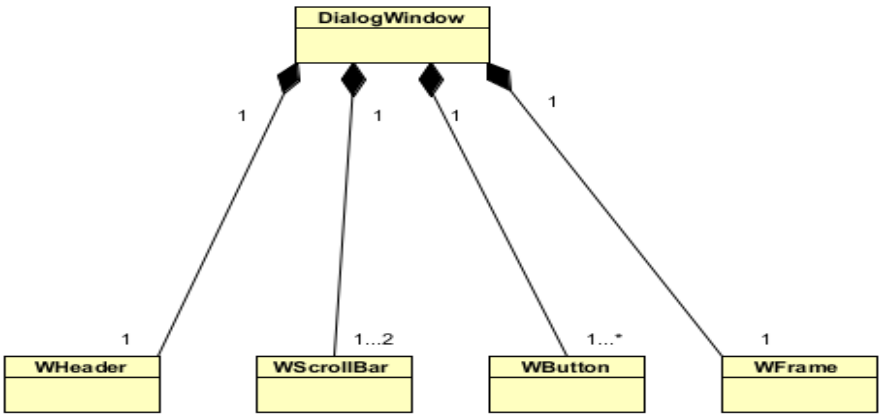


Рисунок 3.5 – Приклад діаграми класів для відношення композиції

На діаграмах відношення узагальнення позначається суцільною лінією з трикутною стрілкою на одному з кінців (рис. 3.6). Стрілка вказує на клас-предок або суперклас, а її відсутність – на більш спеціальний клас (клас-нащадок або підклас).

Діаграми об'єктів

Об'єкт є окремим представником певного класу, і кожен об'єкт має унікальну власну назву та конкретні значення своїх атрибутів. Для позначення об'єктів на діаграмі використовується той же символ, що й для діаграми класів, тільки з позначенням назви об'єкта у форматі *назва об'єкта : назва класу,* а кожен атрибут об'єкту має своє конкретне значення, наприклад, для об'єкту класу *User*, який має назву «Ivanov» та конкретні значення атрибутів LoginName = «Ivan», PW = «1023477892», (рис. 3.7).

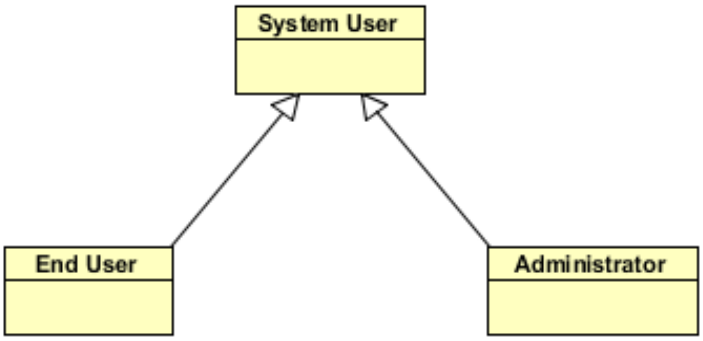


Рисунок 3.6 – Приклад діаграми класів для відношення узагальнення

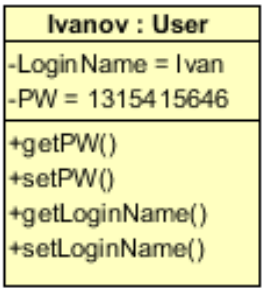


Рисунок 3.7 – Приклад зображення окремого об’єкта на діаграмі

Відношення між об’єктами на діаграмі мають ті ж самі типи, що й для відповідних класів, з тією різницею, що вони завжди позначаються суцільними лініями. Діаграми об’єктів доцільно використовувати для тих випадків у процесі розробки ПЗ, коли конкретні значення даних щодо їх атрибутів та зв’язків дають можливість більш ефективно врахувати їх особливості при подальшій програмній реалізації, тестуванні та супроводу системи. Об’єкти застосовуються також при побудові діаграм комунікації (communication diagram) – див. л/р 4.

3.2. Виконання лабораторної роботи

1. Розробити детальну діаграму класів для обраної предметної області.

2. Для окремих (найбільш важливих) її фрагментів побудувати відповідні діаграми об’єктів.

3.3. Результати виконання роботи та оформлення звіту

Зміст звіту:

1) титульний аркуш;
2) короткі теоретичні відомості;
3) результати практичного виконання, висновки та рекомендації.

Контрольні запитання

1. Які аспекти мають бути враховані при проектуванні програмної системи на логічному рівні? Які типи UML-діаграм застосовуються для цього?

2. Назвіть основі графічні елементи, що мають місце для побудови діаграми класів, поясніть їх призначення.

3. Які типи відношень існують між окремими класами?

4. У чому полягає семантична різниця між відношенням залежності та відношенням асоціації? Наведіть конкретні приклади цих відношень.

5. У чому полягає семантична різниця між відношенням агрегації та відношенням узагальнення? Наведіть конкретні приклади цих відношень.

6. Що означає відношення композиції? Чим воно відрізняється від відношення агрегації?

7. Що таке інтерфейс? Для чого він може бути застосований при розробці діаграми класів?

8. У чому полягає різниця між діаграмою класів та діаграмою об’єктів? Для чого доцільно використання діаграм об’єктів?

ЛАБОРАТОРНА РОБОТА 4

РОЗРОБКА UML–ДІАГРАМ ЛОГІЧНОГО РІВНЯ ПРОЕКТУВАННЯ ПРОГРАМНОЇ СИСТЕМИ: МОДЕЛЮВАННЯ ДИНАМІЧНИХ АСПЕКТІВ

Мета роботи – ознайомитися із методикою побудови UML-діаграм логічного рівня проектування ПЗ, які застосовуються для моделювання динамічних аспектів функціонування ПЗ.

4.1. Загальні відомості

Після побудови діаграми класів (class diagram) та діаграми об’єктів (object diagram) (див. л/р 3), які визначають статичні аспекти побудови відповідних КПР, для моделювання їх динамічних аспектів (тобто для відображення взаємодії компонентів) використовуються такі діаграми UML [3–4]:

- діаграми кінцевого автомату (state machine diagram);
- діаграми діяльності (activity diagram);
- діаграми комунікації (communication diagram);
- діаграми композитної структури (composite structure diagram);
- діаграми обзору взаємодії (interaction overview diagram).

Діаграма кінцевого автомата (state machine diagram)

Діаграма кінцевого автомата описує процес зміни станів одного екземпляра певного класу деякої програмної системи (ПС), тобто моделює всі можливі зміни в стані конкретного програмного об’єкта. При цьому зміна стану об’єкта може бути викликана зовнішніми діями з боку інших об’єктів. Головне призначення цієї діаграми – описати можливі послідовності станів і переходів, які у сукупності характеризують поведінку певного елемента UML-моделі ПС протягом його життєвого циклу.

Для того щоб у середовищі Visual Paradigm перейти до режиму побудови діаграми кінцевого автомата, потрібно на вкладці *UML* вибрати пункт *State Machine Diagram* (рис .4.1).

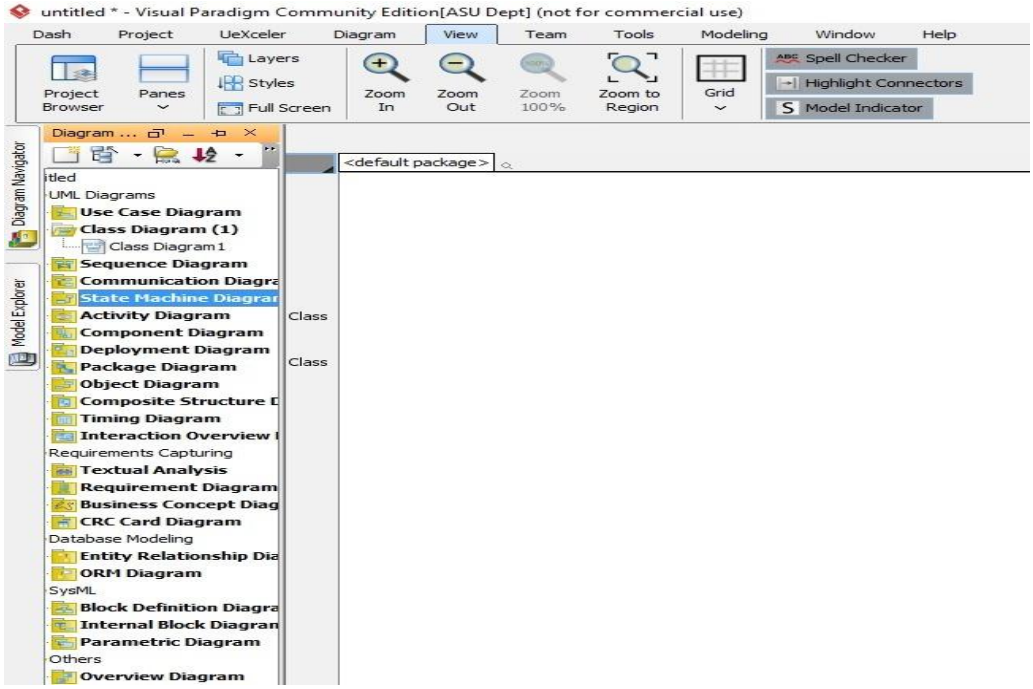


Рисунок 4.1 – Інтерфейс у середовищі Visual Paradigm при побудові діаграми кінцевого автомата

Стан (*state*) об’єкта подається у вигляді набору конкретних значень його атрибутів, при цьому зміна їх окремих значень відображатиме зміну його стану.

Стан на діаграмі зображується прямокутником з вершинами, що є округленими. Цей прямокутник, у свою чергу, може бути розділений горизонтальною лінією на дві секції. Якщо вказана лише одна секція, то у ній записується лише назва стану, інакше в першій з них записується назва стану, а в другій – список деяких внутрішніх дій. Назва стану є рядком тексту, який розкриває змістовний сенс даного стану. Назва завжди записується з великої літери.

У списку внутрішніх дій міститься перелік дій, які виконуються в процесі знаходження об’єкта в цьому стані. Кожна з дій записується у вигляді окремого рядка і має такий формат: «*позначка дії/опис дії*».

Позначка дії вказує на обставини або умови, при яких виконуватиметься відповідна дія. Перелік позначок дії має фіксовані значення, а саме:

• *entry* – ця позначка вказує на дію, яка виконується у момент входу в цей стан (вхідна дія);

• *exit* – ця позначка вказує на дію, яка виконується у момент виходу з цього стану (вихідна дія);

• *do* – ця позначка специфікує діяльність, що виконується, яка виконується протягом усього часу, поки об'єкт знаходиться у цьому стані, або доти, поки не закінчиться обчислення, специфіковане наступним за нею виразом дії;

• *event* – ця позначка визначає дію, яка виконується у момент, коли в ПС наступає певна подія (наприклад, користувач натискає клавішу Escape, тощо).

Початковий стан є окремим випадком стану об'єкта ПС, який не містить жодних внутрішніх дій. Графічно початковий стан у мові UML позначається у вигляді зафарбованого кола, з якого може лише виходити стрілка, відповідна першому переходу ПС у наступний стан.

Кінцевий стан є також окремим випадком стану об'єкта ПС, який також не містить жодних внутрішніх дій. Графічно цей стан позначається у вигляді зафарбованого кола, поміщеного в коло, в яке може лише входити стрілка, відповідна останньому переходу ПС.

Зміст поняття переходу та його позначення на діаграмі

Простий перехід (simple transition) є відношенням між двома послідовними станами, яке вказує на факт зміни одного стану іншим. Перебування об'єкта у першому стані може супроводжуватися виконанням деяких дій, а перехід в другий стан буде можливий після завершення цих дій, а також після задоволення деяких додаткових умов. У цьому випадку говорять, що відповідний перехід *спрацьовує*.

Перехід спрацьовує при настанні деякої події у ПС: це може бути, наприклад, закінчення виконання певної дії (do activity), отримання об'єктом повідомлення та ін. На переході вказується назва події. Крім того, на переході можуть вказуватися дії, що виробляються об'єктом у відповідь на зовнішні події при переході з одного стану в інший. Спрацьовування переходу може залежати не лише від настання деякої події, але і від виконання певної умови, так званої *сторожової умови (guard condition)*. Об'єкт перейде з одного стану в інший у тому випадку, якщо сталася вказана подія і сторожова умова набула значення «true/істина».

Перехід може бути направлений у той же стан, з якого він виходить. У цьому випадку його називають переходом в самого себе, а вихідний і цільовий стани переходу збігаються. При переході в себе об'єкт покидає вихідний стан, а потім знову вхо-

дить в нього. При цьому кожного разу виконуються внутрішні дії, специфіковані позначками *entry* та *exit.*

На рис. 4.2 показано початковий стан деякої ПС та перехід у стан *System start*, що відповідає появі на екрані комп'ютера діалогового вікна (або стартової HTML-сторінки). При вході в цей стан це вікно створюється ПС, при виході з цього стану воно закривається.

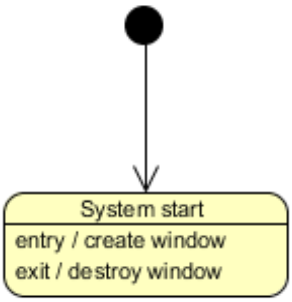


Рисунок 4.2 – Графічне зображення окремого стану з секцією опису внутрішніх дій

На рис. 4.3 наведена діаграма станів для прикладу розробки ПЗ процедури авторизації користувача у ПС (див. л/р 3–4). При вході в стан «System start» створюється відповідне діалогове вікно (при виході воно закривається).

З цього стану є два можливих переходи в інші стани. При виконанні дії *«sign in»* система переходить до стану *«Login and password input»* в якому буде виконано введення даних щодо імені користувача *(LoginName)* та його пароля *(PW).*

При вході у цей стан (позначка дії: *entry/input field initialization*) виконується ініціалізація полів вводу даних (позначка дії *do/get Symbol*) і виконується введення символів у поля вводу, а при отриманні зовнішньої дії натисканням користувачем клавіші «Escape» (позначка дії: *event Escape button / Close window*) вікно закривають.

Із стану *«Login and password input»* існує лише один перехід до стану *«Authorization»,* цей перехід буде виконано після виконання дії *«Input data submit».*

При вході у стан *«Authorization»* буде встановлено з'єднання з базою даних ПС. У цьому стані виконується перевірка імені та пароля користувача (позначка дії *«do /*

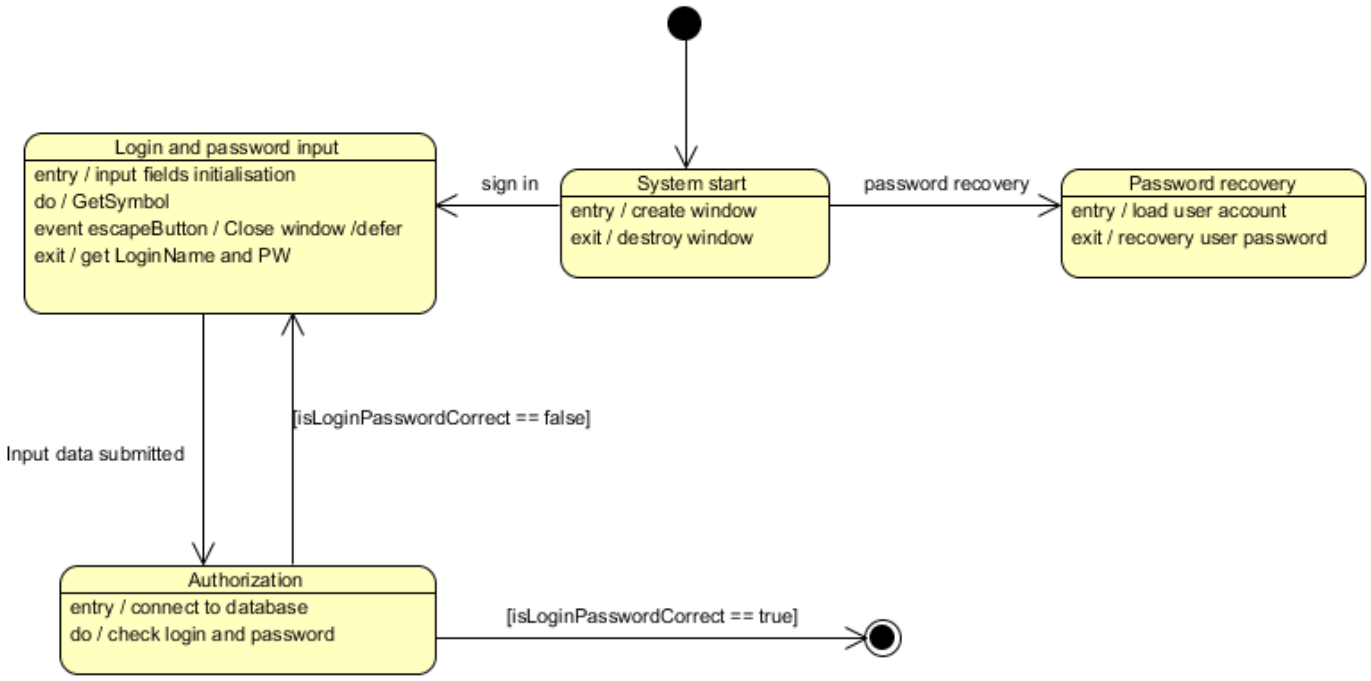


Рисунок 4.3 – Приклад побудови діаграми кінцевого автомата

check login and password»). У разі успішного входу – це позначено сторожовою умовою *[is LoginPasswordCorrect=true]* – система переходить у кінцевий стан, у протилежному випадку – це позначено сторожовою умовою *[is LoginPasswordCorrect=false]* – система має повернутися у стан *«Login and password input»* (рис. 4.3).

Із стану *«System start»* при виконанні дії «*password recovery*» буде виконано перехід до стану *«Password recovery»*. При вході в цей стан буде виконано пошук у БД облікового запису користувача (позначка дії: *entry/load user account*), протягом дії цього стану буде виконано відновлення пароля користувача (позначка дії: *do / recovery user password*).

Діаграми діяльності (activity diagram)

При моделюванні поведінки ПС виникає необхідність не лише представити процес зміни її станів, але і деталізувати особливості алгоритмічної реалізації виконуваних системою операцій. Для моделювання процесу виконання операцій у мові UML використовуються так звані діаграми діяльності. Їх графічна нотація багато в чому схожа на нотацію діаграми станів, оскільки на діаграмах діяльності також присутні позначення станів і переходів. Відмінність полягає в семантиці станів, які ви-

користовуються для визначення саме дій (тобто це так звані *стани дії*), у відсутності на переходах сигнатури опису подій. Кожне перебування на діаграмі діяльності відповідає виконанню деякої елементарної операції, а перехід у наступний стан спрацьовує лише при завершенні цієї операції в попередньому стані.

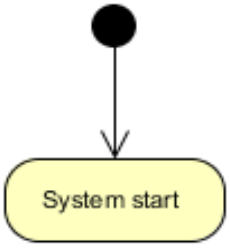
Перехід до режиму побудови діаграми діяльності в середовищі Visual Paradigm виконується так як і для діаграми кінцевого автомата (рис. 4.1).

Стан дії (*action state*) є спеціальним випадком стану з деякою вхідною дією і принаймні одним переходом, що виходить зі стану. Цей перехід неявно передбачає, що вхідна дія вже завершилась. Стан дії не може мати внутрішніх переходів, оскільки воно є елементарним. Звичайне використання стану дії полягає в моделюванні одного кроку виконання алгоритму (процедури) або потоку управління. Графічно стан дії зображується прямокутником, бокові сторони якого замінені округленими дугами (рис. 4.4.). Усередині цієї фігури позначається опис дії (*action-expression*), який має бути унікальним у межах однієї діаграми діяльності. Дія може бути записана розмовною мовою на деякому псевдокоді або мові програмування. Кожна діаграма діяльності повинна мати лише один початковий та один кінцевий стан (вони мають такі ж позначення, як і на діаграмі станів (рис. 4.2–4.3).

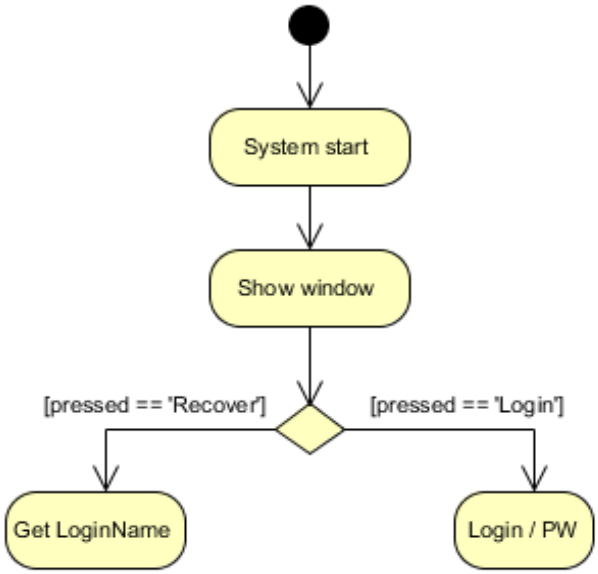
При побудові діаграми діяльності використовуються лише не тригерні переходи, а такі, які спрацьовують відразу після завершення відповідної дії. Цей перехід переводить систему в подальший стан дії відразу, як тільки закінчиться дія в попередньому стані. На діаграмі такий перехід зображується суцільною лінією зі стрілкою. На рис. 4.4, *а* показано окрему дію *«System start»*, яка виконується в ПС безпосередньо після початкового стану.

Якщо зі стану дії виходить єдиний перехід, то він може бути ніяк не помічений. Якщо таких переходів декілька, то спрацювати може лише один з них.

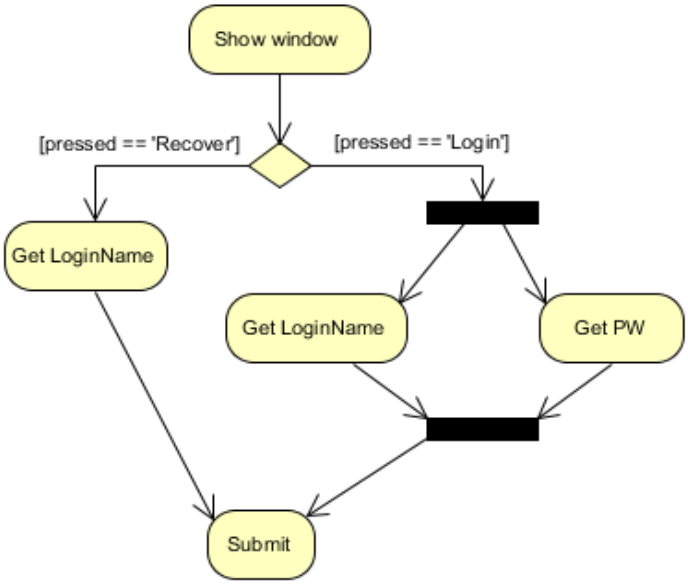
Саме у цьому випадку для кожного з таких переходів має бути явно записана сторожова умова в прямих дужках. При цьому для всіх переходів, що виходять з деякого стану, повинна виконуватися вимога істинності лише одного з них. Подібний випадок зустрічається тоді, коли послідовно виконувана діяльність повинна розділитися на альтернативні гілки залежно від значення деякого проміжного результату.



а



б



в

Рисунок 4.4 – Зображення основних графічних елементів діаграми дії: *а* – окрема дія; *б* – альтернативне розгалуження; *в* – паралельне розділення та злиття (або синхронізація)

Така ситуація отримала назву *альтернативного розгалуження* або просто *розгалуження (decision)*, а для її позначення застосовується спеціальний символ – невеликий ромб, усередині якого немає жодного тексту (рис. 4.4, *б*). У цей ромб може входити лише одна стрілка від того стану дії, після виконання якого потік управління має бути продовжений по одній з гілок, що взаємно виключають. Прийнято вхідну стрілку приєднувати до верхньої або лівої вершини символу розгалуження. Стрілок, що виходять, може бути дві або більше, але для кожної з них явно вказується відповідна сторожова умова у формі логічного виразу.

Один із найбільш значних недоліків звичайних блок-схем або структурних схем алгоритмів пов'язаний з проблемою зображення паралельних гілок окремих обчислень. У мові UML для цієї мети використовується спеціальний символ для розділення і злиття паралельних обчислень або потоків управління в ПС. Це зображується відрізком горизонтальної лінії, товщина якої декілька ширше за основні суцільні лінії діаграми діяльності. При цьому *розділення* (*concurrent fork*) має один вхідний перехід і декілька, що виходять, а *злиття* (*join*) або синхронізація, навпаки, має декілька вхідних переходів і один вихідний (рис.4.4, *в*).

У діаграмах діяльності можливо відобразити той факт, що певні сукупності дій у ПС можуть виконуватися деякими суб'єктами, наприклад, окремі бізнес-процеси є асоційованими з конкретними підрозділами компанії або дії щодо обробки даних розподіляються між клієнтським додатком та сервером.

Для моделювання цих особливостей у мові UML використовується спеціальна конструкція, що отримала назву *доріжки (swimlanes)*. Мається на увазі візуальна аналогія з плавальними доріжками в басейні, якщо дивитися на відповідну діаграму. При цьому всі стани дії на діаграмі діяльності поділяються на окремі групи, які відділяються один від одного вертикальними лініями. Дві сусідні лінії і утворюють доріжку, а група станів між цими лініями виконується окремим суб'єктом: відділом компанії, групою користувачів, програмним додатком тощо. Приклад використання цього механізму відображено на рис. 4.5. На цій діаграмі присутні дві доріжки (*swimlane*). Доріжка *Client* відповідає виконанню клієнта, доріжка *Server* – виконанню сервера. Після старту системи (*System start*) виводиться діалогове вікно (*Show window*). Після цього можуть бути два варіанти виконання системи.

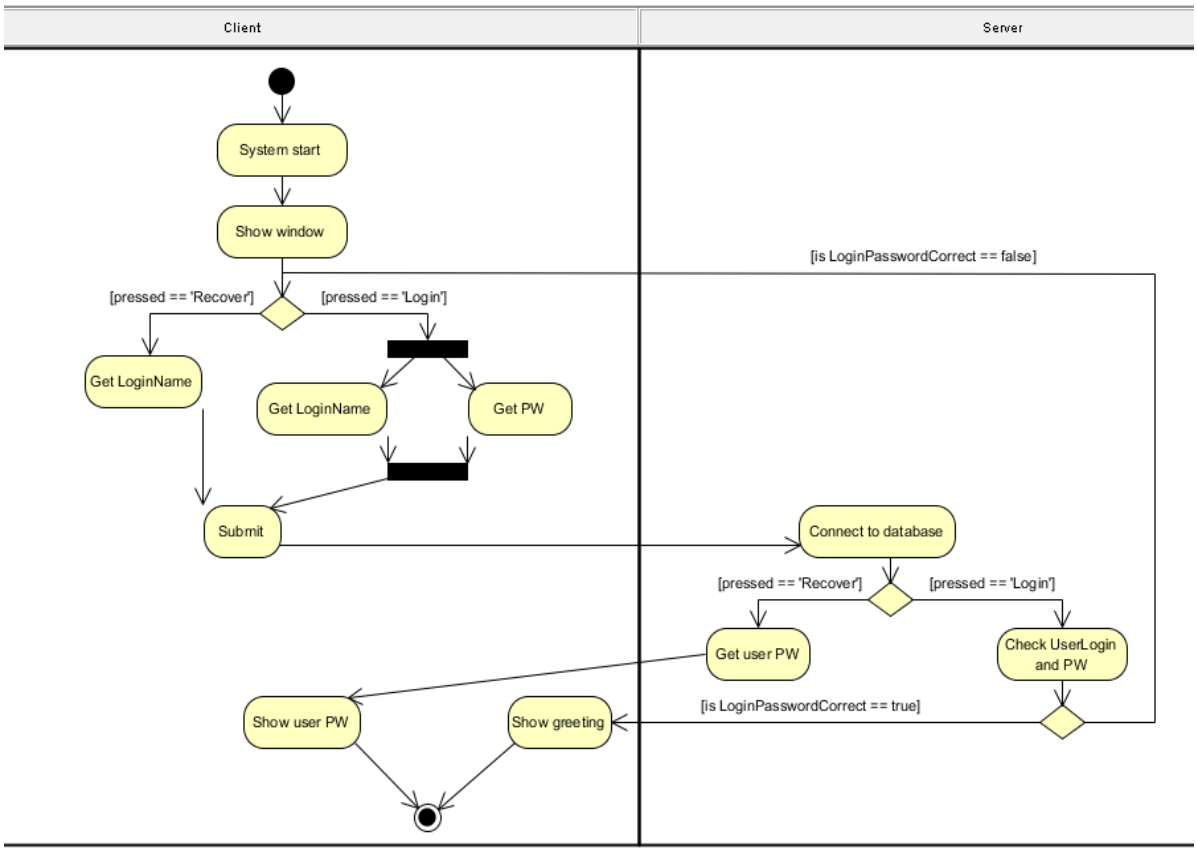


Рисунок 4.5 – Приклад побудови діаграми дії з використанням доріжок

1. Вхід у систему: виконується у випадку виконання умови *pressed == «Login»*. Після цього будуть введені ім’я та пароль користувача *Ger LoginName, Get PW* – ці дії можуть виконуватися у будь-якому порядку, тобто паралельно, – а потім вони синхронізуються при виконанні дії *Submit*. Після цього на сервері виконується підключення до БД системи – дія Connect to database і виконується перевірка імені та пароля: Check Login and PW. У разі успішної перевірки *(«isLoginPasswordCorrect == true»)* буде показано привітання користувачеві *(«Show greeting»)* і система перейде в кінцевий стан, у разі неуспішної перевірки *(«isLoginPasswordCorrect == false»*) виконується перехід до активності *(«Show window»*).

2. Відновлення пароля: виконується у випадку виконання умови *pressed == «Recover»*. Після цього система отримує ім’я користувача *(LoginName)*, на сервері виконується підключення до бази даних та пошук пароля користувача. Після завершення пошуку пароля виконується активність «*Show user PW»* і система переходить до кінцевого стану (рис. 4.5).

Діаграми комунікації (communication diagram)

Для переходу до режиму побудови діаграми комунікації у середовищі Visual Paradigm потрібно на вкладці *UML* вибрати пункт *Communication Diagram* (рис. 4.6).

Головна особливість діаграми комунікації полягає у можливості графічно представити не лише послідовність взаємодії, але і всі структурні стосунки між об'єктами ПС, що беруть участь у цій взаємодії. На цій діаграмі у вигляді прямокутників зображуються об'єкти, що беруть участь у взаємодії, кожен об'єкт містить назву, його клас і, можливо, значення атрибутів. Наприклад, елементарний такий об'єкт, назва якого – *dw,* а тип – *DialogWindow* (рис. 4.6).

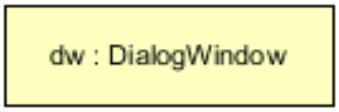


Рисунок 4.6 – Графічне зображення об'єкта на діаграмі комунікації

Далі, як і на діаграмі класів, указуються *асоціації* між об'єктами у вигляді різних сполучних ліній. При цьому можна явно вказати *імена асоціації і ролей*, які відіграють об'єкти у цій асоціації. Додатково зображують динамічні зв'язки – *потоки повідомлень*. Вони подаються у вигляді сполучних ліній між об'єктами, над якими розташовується *стрілка з вказівкою напряму*, *імені повідомлення* і *порядкового номера* в загальній послідовності ініціалізації повідомлень.

Поведінка системи визначається на рівні окремих об'єктів, які обмінюються між собою повідомленнями, аби досягти потрібної мети або реалізувати деякий програмний сервіс. З точки зору аналітика або конструктора важливо представити у проекті системи структурні зв'язки окремих об'єктів між собою. Таке статичне представлення структури системи як сукупності взаємодіючих об'єктів і забезпечує діаграма кооперації.

На відміну від діаграми послідовності, на діаграмі кооперації зображуються лише відношення між об'єктами, що відіграють певні ролі у взаємодії. З іншого боку, на цій діаграмі не вказується час у вигляді окремого виміру. Таким чином, послідов-

ність взаємодій і паралельних потоків може бути визначена на діаграмі кооперації за допомогою порядкових номерів для відповідних асоціацій. (*Примітка: якщо необхідно явно специфікувати взаємозв'язки між об'єктами ПС в реальному часі, то краще це робити на діаграмі послідовності – див. л/р 2*).

На рис. 4.7. наведено варіант побудови діаграми кооперації для розглянутої вище процедури авторизації користувача ПС.

Об’єкт *dw* класу *DialogWindow* посилає повідомлення *setLoginName* та *setPW* об’єкту *user* класу *User*.

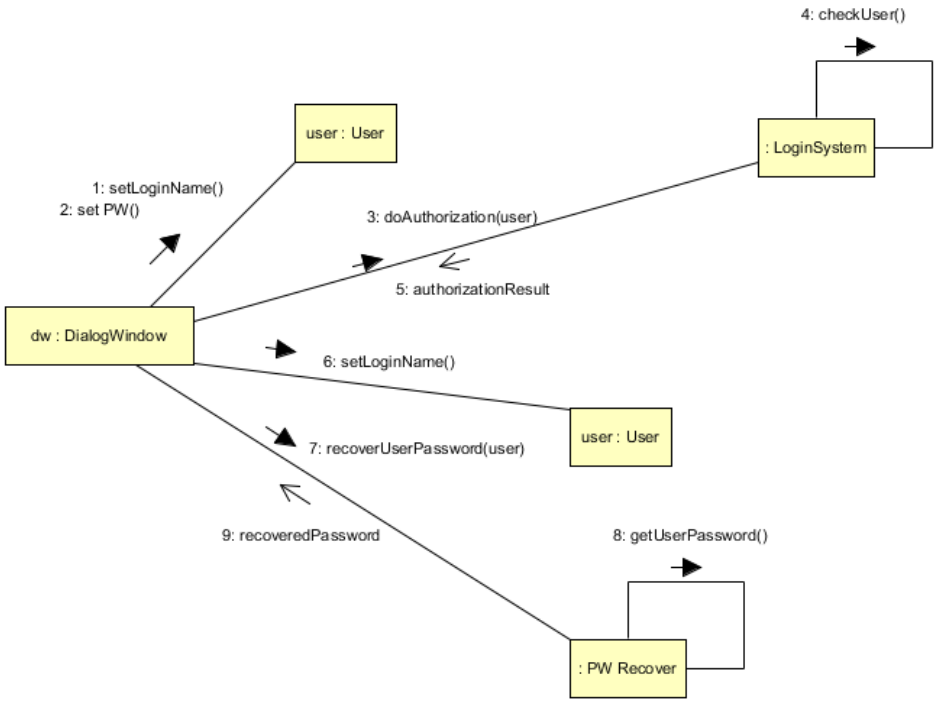


Рисунок 4.7 – Приклад побудови діаграми комунікації

Після чого об’єкт *user* передається системі для проведення перевірки (авторизації) – об’єкт *dw* посилає повідомлення *doAuthorizaton* з параметром *user*. Після чого об’єкт *LoginSystem* виконує метод *checkUser*, результат авторизації передається повідомленням *AuthorizationResult.*

Для відновлення пароля об’єкт *dw* класу *DialogWindow* посилає повідомлення *setLoginName* об’єкту user класу *User*. Після чого об’єкт *user* передається системі для проведення відновлення пароля – об’єкт *dw* посилає повідомлення *recoverUserPassword* з параметром *user*. Далі об’єкт *PWRecover* виконує метод

getUserPassword, результат відновлення пароля передається повідомленням *RecoveredPassword* (рис. 4.7).

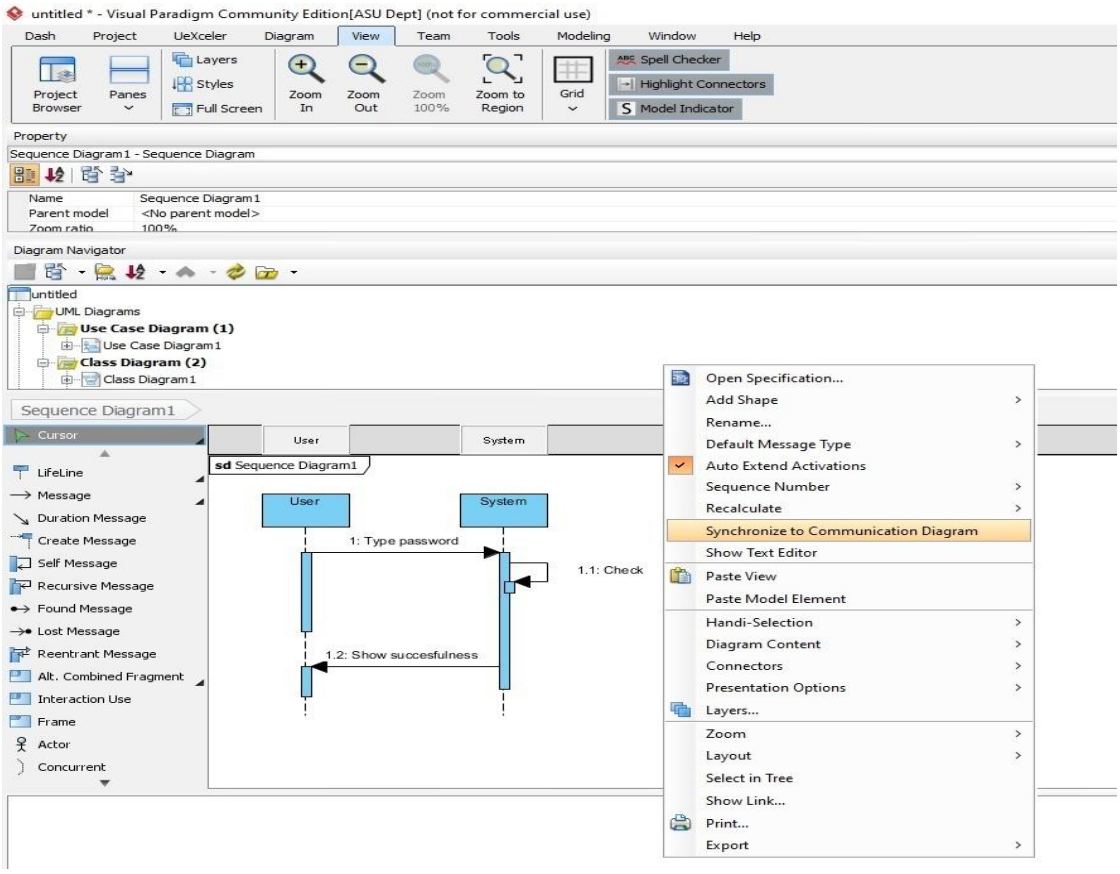


Рисунок 4.8 – Приклад інтерфейсу при автоматичній генерації діаграми комунікації

У VP також є можливість автоматичної генерації діаграм комунікації на основі діаграм послідовностей. Для цього необхідно у полі обраної діаграми послідовності натиснути правою кнопкою миші та вибрати пункт *Synchronize to Communication Diagram* (рис. 4.8).

Діаграми композитної структури (composite structure diagram)

Діаграма композитної (складової) структури – статична структурна діаграма, демонструє внутрішню структуру класів і, по можливості, взаємодію елементів (частин) внутрішньої структури класу. Такі діаграми використовуються спільно з діаграмами класів.

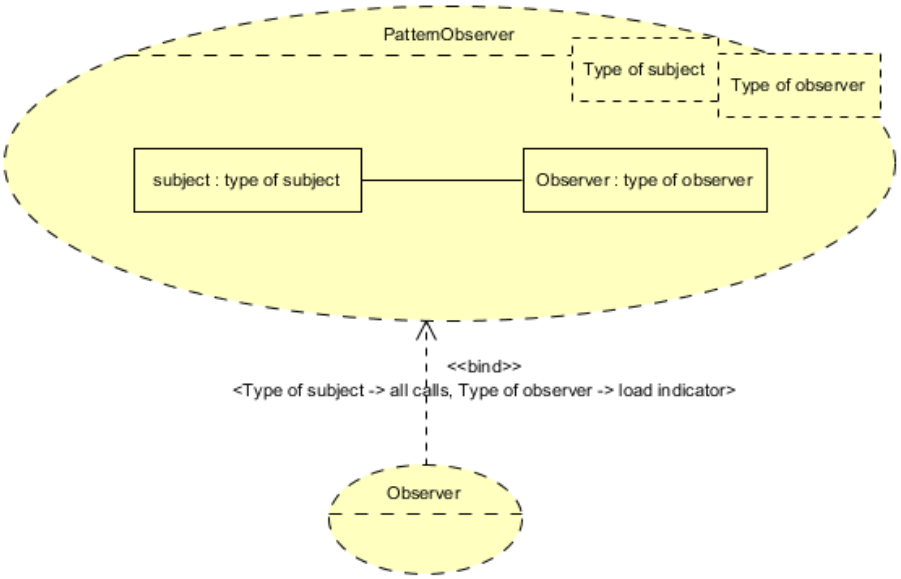


Рисунок 4.9 – Приклад побудови діаграми композитної структури (шаблон кооперації патерна Спостерігач та його зв'язування)

Як елементи опцій на цих діаграмах часто зображується класифікатор – кооперація, який призначається для опису деякої структури елементів або ролей, що виконують спеціалізовані функції та спільно забезпечують бажану функціональність. Ці елементи зображуються у вигляді різних шаблонів (патернів) проектування (рис. 4.9).

Діаграми обзору взаємодії (interaction overview diagram)

Діаграма обзору взаємодії – це комбінація діаграм діяльності та діаграм послідовності. Можна вважати діаграми обзору взаємодії діаграмами діяльності, у яких діяльності замінені невеликими діаграмами послідовності, або ж діаграмами послідовності, що розбиті за допомогою нотації діаграм діяльності для відображення потоку управління. У будь-якому випадку вони являють собою досить незвичайну суміш.

На рис. 4.10 поданий приклад побудови діаграми такого типу для процедури авторизації користувача ПС.

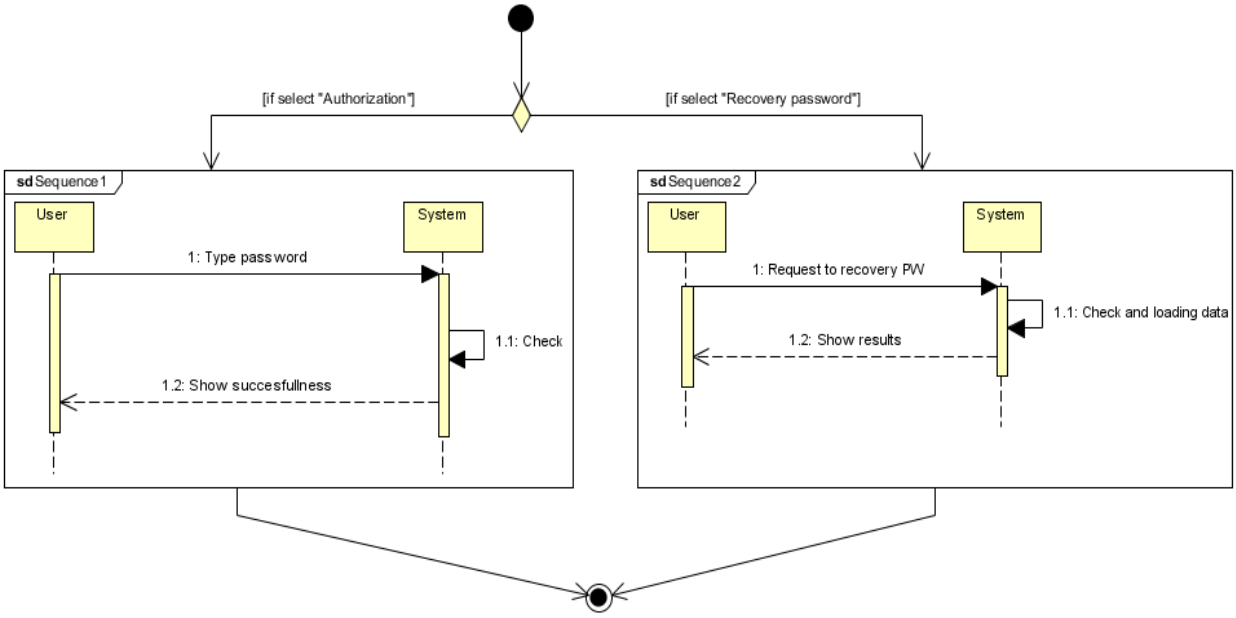


Рисунок 4.10 – Приклад побудови діаграми обзору взаємодії

4.2. Виконання лабораторної роботи

1. Розробити діаграму кінцевого автомату для обраної предметної області.
2. Розробити діаграму діяльності для обраної предметної області.
3. Розробити діаграму комунікації для обраної предметної області.
4. Розробити діаграму композитної структури для обраної предметної області.
5. Розробити діаграму обзору взаємодії для обраної предметної області.

4.3. Висновки до виконаної роботи та оформлення звіту

Зміст звіту:

1) титульний аркуш;
2) короткі теоретичні відомості;
3) результати практичного виконання, висновки та рекомендації.

Контрольні запитання

1. Які типи UML-діаграм застосовуються при проектуванні програмної системи на логічному рівні з метою моделювання динамічних аспектів ПЗ?

2. Для чого застосовуються діаграми кінцевого автомата (ДКА)? Назвіть основі графічні елементи, що мають місце для побудови ДКА, поясніть їх призначення.

3. Які види позначок дій є можливими для ДКА, що вони означають?

4. Що таке сторожова умова, навіщо вона застосовується в ДКА?

5. Для чого застосовуються діаграми діяльності (ДД)? У чому полягає різниця між застосуванням ДКА і ДД?

6. Назвіть основні графічні елементи, що мають місце для побудови ДД, поясніть їх призначення.

7. У чому полягає різниця між станом у ДКА та станом дії у ДД?

8. Що є спільного та в чому полягає різниця між такими елементами ДД як *розгалуження (decision)* та *розділення (concurrent fork)*?

9. Для моделювання яких особливостей виконання ПЗ використовується конструкція *доріжки (swimlines)*?

10. Для чого застосовуються діаграми комунікації (ДК)? Назвіть основні графічні елементи, що мають місце для побудови ДК, поясніть їх призначення.

11. У чому полягає різниця між ДК та діаграмами послідовностей?

12. Яким чином на ДК враховується фактор дії поточного часу?

ЛАБОРАТОРНА РОБОТА 5

РОЗРОБКА UML–ДІАГРАМ ФІЗИЧНОГО РІВНЯ ПРОЕКТУВАННЯ ПРОГРАМНОЇ СИСТЕМИ

Мета роботи – ознайомитись з методикою побудови UML-діаграм фізичного рівня проектування програмної системи (ПС).

5.1. Загальні відомості

Діаграма компонентів

Діаграма компонентів (component diagram) описує особливості фізичного представлення програмної системи. Вона дозволяє визначити архітектуру розроблюваної системи, встановивши залежності між програмними компонентами (вихідним кодом, програмними модулями тощо).

У розробці діаграм компонентів беруть участь як системні аналітики та архітектори, так і програмісти. Для опису фізичних програмних сутностей у мові UML застосовується спеціальний термін: компонент (*component*). Компонент реалізує деякий набір інтерфейсів і слугує для загального позначення елементів фізичного представлення моделі. Для графічного позначення компонента використовується спеціальний символ – прямокутник зі вставленими зліва двома дрібнішими прямокутниками. Усередині великого прямокутника записується ім’я компонента і при необхідності деяка додаткова інформація. Ім’я компонента підпорядковується загальним правилам іменування елементів моделі у мові UML і може складатися з будь-якої кількості букв, цифр та деяких інших символів.

Таким чином, компоненти на UML–діаграмах розгортання являють собою фізичні файли, які можуть бути безпосередньо виконані (наприклад з розширенням exe, dll та ін.) або такі, що використовуються для отримання кінцевої ПС, тобто це файли з вихідними текстами програм (з розширеннями php або cpp для мови C ++), Web-сторінки з розширенням html і файли довідки з розширенням hlp та ін.

На рис. 5.1 показаний приклад простої діаграми компонентів [4].

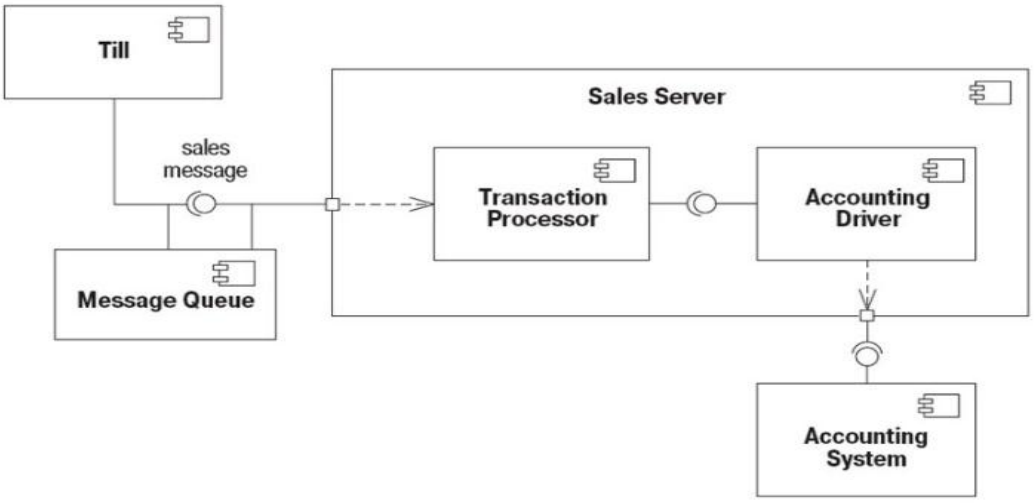


Рисунок 5.1 – Приклад діаграми компонентів [4]

У цьому прикладі компонент Till (Каса) може взаємодіяти з компонентом Sales Server (Сервер Продажу) за допомогою інтерфейсу sales message (повідомлення про продаж). Оскільки мережа ненадійна, то компонент Message Queue (Черга Повідомлень) встановлено так, щоб каса могла спілкуватися з сервером, коли мережа працює, і розмовляти з чергою повідомлень, коли мережа відключена. Тоді черга повідомлень зможе поговорити з сервером, коли мережа знову стане доступною. У результаті чергу повідомлень надає інтерфейс для розмови з касою, і вимагає такий же інтерфейс для розмови з сервером. Сервер розділений на два основних компоненти: Transaction Processor (Процесор транзакцій) реалізує інтерфейс повідомлень, а Accounting Driver (Драйвер рахунків) спілкується з Accounting System (Система обліку рахунків).

Діаграма розгортання

Діаграма розгортання (deployment diagram) призначена для візуалізації елементів і компонентів ПС, які існують лише на етапі її виконання (*runtime*). При цьому подаються тільки компоненти-екземпляри програми, які є виконуваними файлами або динамічними бібліотеками. Ті компоненти, які не використовуються на етапі виконання, на діаграмі розгортання не відображаються. Так, компоненти з вихідними текстами програм можуть бути присутніми тільки на діаграмі компонентів, а на діаграмі розгортання вони не вказуються.

Діаграма розгортання містить графічні зображення процесорів, пристроїв, процесів і зв'язків між ними. На відміну від діаграм логічного подання (напр., діаграми класів) діаграма розгортання є єдиною для ПС в цілому, оскільки повинна повністю відображати особливості її реалізації. Розробка діаграми розгортання, як правило, є останнім етапом специфікації моделі ПС.

При розробці діаграми розгортання переслідують такі цілі:

- визначити розподіл компонентів системи по її фізичних вузлах деякої обчислювальної мережі (network);
- показати фізичні зв'язки між усіма вузлами реалізації ПС на етапі її виконання;
- виявити вузькі місця системи і ре-конфігурувати її топологію для досягнення необхідної продуктивності.

Діаграми розгортання розробляються спільно системними архітекторами, інженерами-системотехніками та програмістами. Вузол (*node*) являє собою деякий фізично існуючий елемент ПС, що володіє певним обчислювальним ресурсом. Як обчислювальний ресурс вузла може розглядатися наявність деякого об’єму електронної або магнітооптичної пам’яті або процесора. В останній версії мови UML поняття вузла розширено і може включати в себе не тільки обчислювальні пристрої, а й інші механічні або електронні пристрої, такі як датчики, принтери, модеми, цифрові камери, сканери і маніпулятори. Графічно на діаграмі розгортання вузол зображується у формі тривимірного куба. Вузол має власне ім’я, яке вказується всередині його графічного символу.

Нижче наведені 2 діаграми розгортання компонентів, які являють собою ПС, яка розроблена з використанням: 1) компонентів технології JEE та 2) мови створення серверних сценаріїв PHP. Обидві ці ПС імплементують функціональність, яка представлена на наступній діаграмі сценаріїв використання (рис. 5.2).

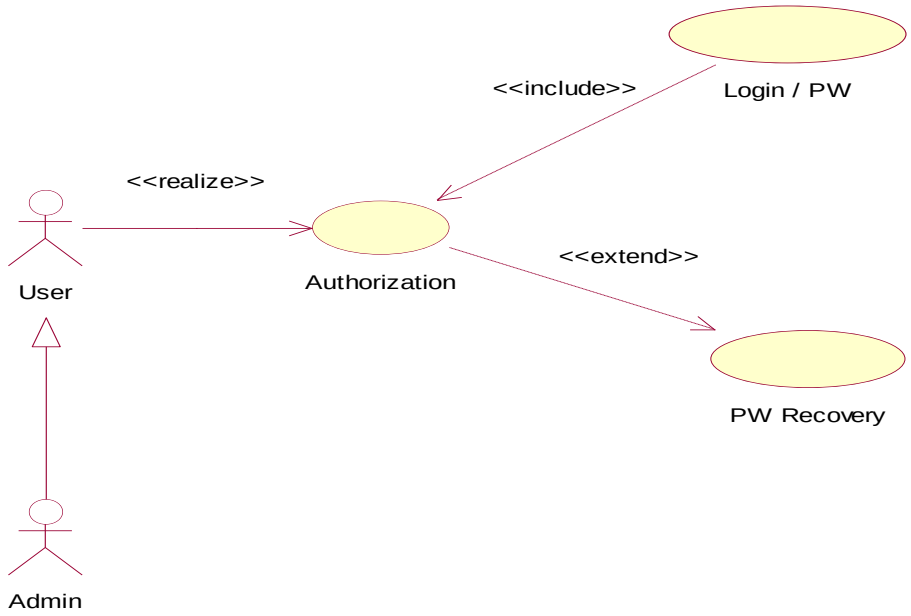


Рисунок 5.2 – Сценарій авторизації користувача у Web-системі

Реалізація на платформі Java

На *вузлі Client* розгорнуто два компоненти (рис. 5.3):

1. Web Browser – програма веб-браузер, що надає користувачеві доступ до системи.
2. JavaScript Framework – каркас javascript, що є необхідним для програмної реалізації певної бізнес-логіки на стороні клієнта.

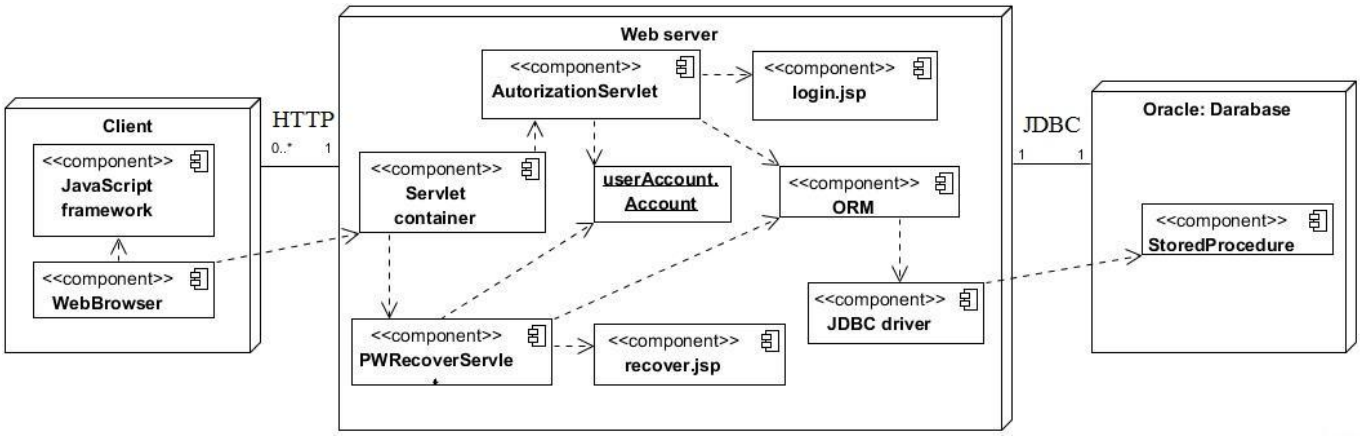


Рисунок 5.3 – Діаграма розгортання для реалізації на Java

Вузол *Web server* – це веб-сервер системи, клієнт взаємодіє з ним за допомогою протоколу HTTP. Розмірність відповідного зв'язку зазначено як «0..* : 1», тобто передбачено можливість одночасної роботи декількох клієнтів із 1 Web-сервером.

На цьому вузлі розгорнуто такі компоненти:

1. Servlet container – це контейнер java-сервлетів, що відповідає за виконання серверної логіки.
2. AuthorizationServlet – сервлет, що реалізує процес аутентифікації у системі.
3. PWRecoverServlet – сервлет, що реалізує процес відновлення пароля користувача.
4. ORM – компонент, що реалізує об'єктно-реляційне відображення (object-relational mapping) при роботі з базою даних системи.
5. login.jsp – це серверна jsp-сторінка, що відображає дані на етапі аутентифікації.
6. Recover.jsp – це jsp-серверна сторінка, що відображає дані на етапі відновлення пароля користувача.
7. userAccount – це програмний об'єкт, який створюється у процесі робот та містить обліковий запис користувача.
8. JDBC Driver – це драйвер, що виконує підключення до бази даних. Цей об'єкт створюється при аутентифікації у системі або при відновленні пароля.

Вузол *Database* – це вузол розміщення бази даних системи, яка реалізована за допомогою конкретної СКБД Oracle, зв'язок з цим вузлом забезпечується за допомогою протоколу JDBC, при цьому розмірність цього зв'язку визначена як «1:1», тобто в системі, що розробляється, передбачено мати 1 Web-сервер і 1 сервер БД. У контейнері сервера БД передбачено також наявність компоненту *StoredProcedure* – це відповідні збережені процедури, що імплементують додаткову бізнес-логіку системи.

Реалізація на платформі PHP

Вузол *Client* має два компоненти (рис. 5.4):

1. Web Browser – програма веб-браузера, що надає користувачеві доступ до системи авторизації;

2. JavaScript Framework – каркас javascript, що необхідний для реалізації додаткової бізнес-логіки на стороні клієнта.

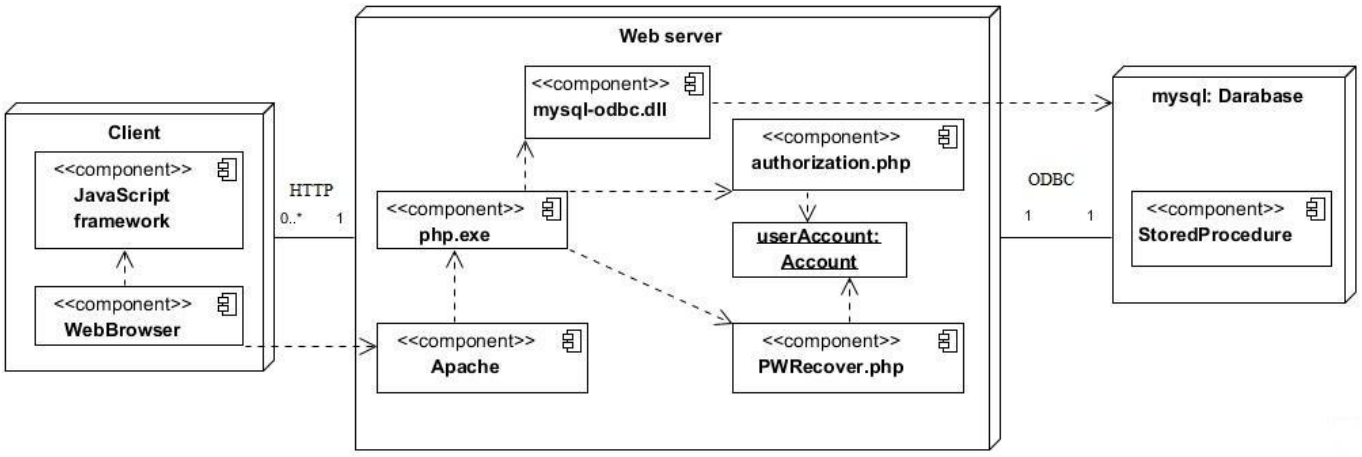


Рисунок 5.4 – Діаграма розгортання у разі реалізації на PHP

Вузол *Web server* – це веб-сервер системи, клієнт взаємодіє з ним за допомогою протоколу HTTP. Розмірність відповідного зв'язку зазначено як «0..* : 1», тобто передбачено можливість одночасної роботи декількох клієнтів із 1 Web-сервером.
На ньому розгорнуто такі компоненти:

1. Apache – це компонент веб-сервера, що забезпечує виконання PHP-скриптів (сторінок).
2. authorization.php – файл зі скриптом, що реалізує процес аутентифікації в системі.
3. PWRecover.php – файл зі скриптом, що реалізує процес відновлення пароля користувача.
4. php.exe – інтерпретатор мови PHP.
5. php-mysql.dll – це бібліотека драйверів, що дозволяє виконувати підключення php-сторінок до бази даних системи.
6. userAccount – це об'єкт, що містить обліковий запис користувача. Цей об'єкт створюється при аутентифікації в системі або при відновленні пароля.

Вузол *Database* – це вузол розміщення бази даних системи, яка реалізована в цьому проекті за допомогою конкретної СКБД MySQL, зв'язок з цим вузлом забез-

печується за допомогою протоколу ODBC, при цьому розмірність цього зв'язку визначена як «1:1», тобто у системі, що розробляється, передбачено мати 1 Web-сервер і 1 сервер БД. У контейнері сервера БД передбачено також наявність компонента StoredProcedure – це відповідні збережені процедури, що імплементують додаткову бізнес-логіку системи.

Вузол Database – це база даних системи, сервер взаємодії з БД за допомогою протоколу JDBC. Компонент StoredProcedure – це хранимі процедури, що містяться у базі даних.

5.2. Виконання лабораторної роботи

1. Розробити діаграму компонентів для обраної предметної області.
2. Розробити діаграму розгортання для обраної предметної області.

5.3. Висновки до виконаної роботи та оформлення звіту

Зміст звіту:

1) титульний аркуш;
2) короткі теоретичні відомості;
3) результати практичного виконання, висновки та рекомендації.

Контрольні запитання

1. Для якого етапу проектування ПС використовуються діаграми компонентів та діаграми розгортання?
2. Для чого розробляється діаграма компонентів?
3. Для чого розробляється діаграма розгортання?
4. Що являють собою компоненти на UML–діаграмах розгортання?
5. Для чого застосовуються діаграми розгортання?
6. Що таке вузол? Як він зображується в UML?
7. У чому полягає різниця між діаграмами розгортання компонентів, розроблених з використанням технологій JEE та PHP?
8. Що таке вузол Web-server? Які компоненти розгорнуті на ньому?

9. З яких компонентів складається вузол Client?

10. Яку функціональність реалізує в обох діаграмах вузол Database?

СПИСОК ДЖЕРЕЛ ІНФОРМАЦІЇ

1. Офіційний сайт спільноти користувачів CASE – засобу Visual Paradigm [Електонний ресурс:]. Режим доступу // https://www.visual-paradigm.com/

2. Стандарт UML 2.0 [Електронний ресурс:]. Режим доступу // http://www.omg.org/spec/UML/2.0/

3. Лавріщева К.М. Програмна інженерія / Лавріщева К.М. – Київ: КНУ ім. Тараса Шевченка, 2008. – 319 с.

4. Ларман К. Применение UML 2.0 и шаблонов проектирования : практическое руководство / Ларман К.: пер. с англ. – М. ООО «И.Д. Вильямс», 2009. – 736 с.

Навчальне видання

МЕТОДИЧНІ ВКАЗІВКИ

до виконання лабораторних робіт з дисципліни

«Архітектура та проектування програмного забезпечення»

за розділом

«Мова моделювання UML як засіб аналізу та проектування програмних систем»

для студентів з напряму підготовки 6.050103 «Програмна інженерія»

Укладачі :

ТКАЧУК Микола Вячеславович, СОКОЛ Володимир Євгенович, ГАМЗАЄВ Рустам Олександрович, МАРТІНКУС Ірина Олегівна, НАГОРНИЙ Костянтин Анатолійович, НАУМЕНКО Ольга Сергіївна, БРОНІН Сергій Вадимович

Відповідальний за випуск М.Д. Годлевський

Роботу до видання рекомендував О.В. Горілий

Редактор Н.В. Верестюк

План 2017 р., поз. 128

Підписано до друку 04.10.2017 Формат 60х84 1/16 Папір друк.

Гарнітура Times New Roman. Ум. друк. арк. 2,6.

Наклад 100 прим. Зам. ____ Ціна договірна

Електронний варіант