Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Лабораторна робота №3

з дисципліни «Технології розроблення програмного забезпечення»

Тема: «Діаграма розгортання. Діаграма компонентів. Діаграма взаємодій та послідовностей.»

Варіант №15

Виконав:
студент групи ІА-23
Лядський Д.С.

Перевірив:
Мягкий М. Ю.

Київ 2024

Зміст

Тема

Діаграма розгортання. Діаграма компонентів. Діаграма взаємодій та послідовностей.

Мета

Метою цієї лабораторної роботи є розробка та візуалізація архітектури програмної системи для поштового клієнта за допомогою методів моделювання в UML. Студенти ознайомляться з принципами створення різних типів діаграм, таких як діаграма розгортання, діаграма компонентів, діаграма взаємодій та діаграма послідовностей, та навчаться використовувати їх для представлення фізичних, логічних та поведінкових аспектів програмних систем.

Короткі теоретичні відомості

Діаграма розгортання (Deployment Diagram)

Діаграми розгортання представляють фізичне розташування системи, показуючи, на якому фізичному обладнанні запускається та чи інша складова програмного забезпечення.

Основними елементами діаграми є вузли, з'єднані інформаційними шляхами. Вузол (node) — це те, що може містити програмне забезпечення. Вузли бувають двох типів. Пристрій (device) — це фізичне обладнання: комп'ютер або пристрій, пов'язаний з системою. Середовище виконання (execution environment) — це програмне забезпечення, яке саме може включати інше програмне забезпечення, наприклад операційну систему або процес-контейнер (наприклад, веб-сервер).

Між вузлами можуть бути з'єднання, зазвичай зображувані у вигляді прямої лінії. Як і на інших діаграмах, у з'єднань можуть бути атрибути множинності (для вказівки, наприклад, підключення двох і більше клієнтів до одного сервера) та назва. У назві зазвичай міститься спосіб зв'язку між двома вузлами — це може бути назва протоколу (http, IPC) або використана технологія для забезпечення взаємодії вузлів (.net Remoting, WCF).

Вузли можуть містити артефакти (artifacts), які є фізичним втіленням програмного забезпечення; зазвичай це файли. Такі файли можуть бути виконуваними файлами (як-от файли .exe, бінарні файли, файли DLL, файли JAR, зборки або сценарії) або файлами даних, конфігураційними файлами, HTML-документами тощо. Перелік артефактів всередині вузла вказує на те, що на даному вузлі артефакт розгортається в запускаючу систему.

Артефакти можна зображати у вигляді прямокутників класів або перераховувати їхні імена всередині вузла. Якщо ви показуєте ці елементи у вигляді прямокутників класів, то можете додати значок документа або ключове слово «artifact». Можна супроводжувати вузли або артефакти значеннями у вигляді міток, щоб вказати різну цікаву інформацію про вузол, наприклад постачальника, операційну систему, місцезнаходження — загалом усе, що прийде вам до голови.

Часто у вас буде кілька фізичних вузлів для вирішення однієї й тієї ж логічної задачі. Можна відобразити цей факт, намалювавши кілька прямокутників вузлів або поставивши число у вигляді значення-мітки.

Артефакти часто є реалізацією компонентів. Це можна показати, задавши значення-мітки всередині прямокутників артефактів.

Основні види артефактів:

- вихідні файли;
- виконувані файли;
- сценарії;
- таблиці баз даних;
- документи;
- результати процесу розробки, UML-моделі.

Можна також деталізувати артефакти, що входять до вузла; наприклад, додатково всередині розгорнутого файлу вказати, які компоненти або класи туди входять. Така деталізація, як правило, не має сенсу на діаграмах

розгортання, оскільки може зміщувати фокус уваги з моделі розгортання програмного забезпечення на його внутрішню структуру, однак іноді може бути корисною. При цьому, можливо, встановлювати зв'язки між компонентами/класами в межах різних вузлів.

Діаграми розгортання поділяються на два види: описові та екземплярні. На діаграмах описового типу вказуються вузли, артефакти та зв'язки між вузлами без вказівки конкретного обладнання або програмного забезпечення, необхідного для розгортання. Такий вид діаграм корисний на ранніх етапах розробки для розуміння, які фізичні пристрої необхідні для функціонування системи або для опису процесу розгортання в загальних рисах.

Діаграми екземплярного типу містять екземпляри обладнання, артефактів та зв'язків між ними. Під екземплярами розуміються конкретні елементи — ПК з відповідним набором характеристик і встановленим ПЗ; цілком може бути, що в межах однієї організації це буде якийсь конкретний вузол (наприклад, ПК тестувальника Василя). Діаграми екземплярної форми розробляються на завершальних етапах розробки ПЗ — коли вже відомі й сформульовані вимоги до програмного комплексу, обладнання закуплене і все готово до розгортання. Діаграми такої форми більше являють собою план розгортання в графічному вигляді, ніж модель розгортання.

Діаграма компонентів

Діаграма компонентів UML є представленням проектованої системи, розбитої на окремі модулі. Залежно від способу розділення на модулі розрізняють три види діаграм компонентів:

- Логічні
- Фізичні
- Виконувані

Найбільш часто використовується логічне розбиття на компоненти — у цьому випадку проектована система віртуально представлена як набір самостійних, автономних модулів (компонентів), що взаємодіють між собою.

Компоненти можуть розподілятися за фізичними одиницями — окремі вузли розподіленої системи — набір комп'ютерів і серверів; на кожному з вузлів можуть бути встановлені різні виконувані компоненти. Такий вид діаграм компонентів застарів і, як правило, замість нього використовується діаграма розгортання.

На діаграмах компонентів з виконуваним розподілом кожен компонент являє собою певний файл — виконувані файли (.exe), файли вихідного коду, сторінки html, бази даних і таблиці та інше.

- Діаграма компонентів розробляється для наступних цілей:
- Візуалізація загальної структури вихідного коду програмної системи.
- Специфікація виконуваного варіанту програмної системи.
- Забезпечення багаторазового використання окремих фрагментів програмного коду.
- Представлення концептуальної та фізичної схем баз даних.

Рекомендації по побудові діаграми компонентів

Розробка діаграми компонентів передбачає використання інформації як про логічне представлення моделі системи, так і про особливості її фізичної реалізації. До початку розробки необхідно прийняти рішення щодо вибору обчислювальних платформ і операційних систем, на яких передбачається реалізовувати систему, а також щодо вибору конкретних баз даних і мов програмування.

Після цього можна приступати до загальної структуризації діаграми компонентів. Спочатку необхідно вирішити, з яких фізичних частин (файлів) буде складатися програмна система. На цьому етапі слід звернути увагу на таку

реалізацію системи, яка забезпечувала б не лише можливість повторного використання коду завдяки раціональній декомпозиції компонентів, а й створення об'єктів лише за необхідності.

Загальна продуктивність програмної системи суттєво залежить від раціонального використання обчислювальних ресурсів. Для цієї мети необхідно більшу частину описів класів, їх операцій і методів винести в динамічні бібліотеки, залишивши в виконуваних компонентах лише найнеобхідніші для ініціалізації програми фрагменти програмного коду.

Після загальної структуризації фізичного представлення системи необхідно доповнити модель інтерфейсами та схемами баз даних. При розробці інтерфейсів слід звертати увагу на узгодження (стиковку) різних частин програмної системи. Включення в модель схеми бази даних передбачає специфікацію окремих таблиць і встановлення інформаційних зв'язків між таблицями.

Завершальний етап побудови діаграми компонентів пов'язаний з встановленням і нанесенням на діаграму взаємних зв'язків між компонентами, а також відносин реалізації. Ці відносини мають ілюструвати всі важливі аспекти фізичної реалізації системи, починаючи з особливостей компіляції вихідних текстів програм і закінчуючи виконанням окремих частин програми на етапі її виконання. Для цієї мети можна використовувати різні види графічного зображення компонентів.

Діаграми послідовностей

При розробці діаграми компонентів слід дотримуватися загальних принципів створення моделей мовою UML. Зокрема, необхідно використовувати вже наявні в мові UML компоненти та стереотипи. Для більшості типових проектів цього набору елементів може бути достатньо для представлення компонентів та залежностей між ними.

Якщо проект містить деякі фізичні елементи, опис яких відсутній в мові UML, слід скористатися механізмом розширення та використовувати додаткові стереотипи для окремих нетипових компонентів або позначені значення для уточнення їх окремих характеристик.

Слід звернути увагу, що діаграма компонентів зазвичай розробляється разом з діаграмою розгортання, на якій представлена інформація про фізичне розміщення компонентів програмної системи на її окремих вузлах.

При моделюванні поведінки проектованої або аналізованої системи виникає необхідність не лише представити процес зміни її станів, а й деталізувати особливості алгоритмічної та логічної реалізації виконуваних системою операцій.

Для моделювання процесу виконання операцій у мові UML використовуються діаграми діяльності. Використовувана в них графічна нотація в багатьох аспектах схожа на нотацію діаграми станів, оскільки на цих діаграмах також присутні позначення станів та переходів. Кожен стан на діаграмі діяльності відповідає виконанню деякої елементарної операції, а перехід до наступного стану виконується лише після завершення цієї операції.

Таким чином, діаграми діяльності можна вважати приватним випадком діаграм станів. Вони дозволяють реалізувати в мові UML особливості процедурного та синхронного управління, що обумовлені завершенням внутрішніх діяльностей та дій. Основним напрямком використання діаграм діяльності є візуалізація особливостей реалізації операцій класів, коли необхідно представити алгоритми їх виконання.

У контексті мови UML діяльність (activity) є сукупністю окремих обчислень, які виконує автомат, що призводить до певного результату або дії (action). На діаграмі діяльності відображається логіка та послідовність переходів від однієї діяльності до іншої, а увага аналітика фокусується на

результатах. Результат діяльності може призвести до зміни стану системи або повернення певного значення.

Стан дії

Стан дії (action state) є спеціальним випадком стану з певним вхідним дією та, принаймні, одним вихідним з нього переходом. Цей перехід неявно припускає, що вхідне дію вже завершено. Стан дії не може мати внутрішніх переходів, оскільки він є елементарним. Звичайне використання стану дії полягає в моделюванні одного кроку виконання алгоритму (процедури) або потоку управління.

Графічно стан дії зображується прямокутником з закругленими кутами. Всередині цього зображення записується вираз дії (action-expression), який повинен бути унікальним в межах однієї діаграми діяльності.

Дія може бути записана на природній мові, деякому псевдокоді або мові програмування. Жодних додаткових або неявних обмежень при записі дій не накладається. Рекомендується в якості імені простого дії використовувати дієслово з пояснювальними словами. Якщо ж дія може бути представлене в деякому формальному вигляді, то доцільно записати його на тій мові програмування, на якій передбачається реалізувати конкретний проект.

Іноді виникає необхідність представити на діаграмі діяльності деяку складну дію, яке, у свою чергу, складається з кількох більш простих дій. У цьому випадку можна використовувати спеціальне позначення стану під-діяльності (subactivity state). Таке стан є графом діяльності та позначається спеціальною піктограмою в правому нижньому куті символа стану дії. Ця конструкція може застосовуватися до будь-якого елемента мови UML, який підтримує вкладеність своєї структури. При цьому піктограма може бути додатково помічена типом вкладеної структури.

Кожна діаграма діяльності повинна мати єдине початкове та єдине кінцеве стан. Вони мають такі ж позначення, як і на діаграмі станів. При цьому

кожна діяльність починається в початковому стані і завершується в кінцевому стані. Саму діаграму діяльності прийнято розташовувати так, щоб дії слідували зверху вниз. У цьому випадку початкове стан буде зображуватися у верхній частині діаграми, а кінцеве — в нижній.

1. Переходи

Перехід як елемент мови UML був розглянутий на діаграмах станів. При побудові діаграми діяльності використовуються лише нетриггерні переходи, тобто ті, які виконуються відразу після завершення діяльності або виконання відповідної дії. Цей перехід переводить діяльність у наступний стан, як тільки завершиться дія в попередньому стані. На діаграмі такий перехід зображується суцільною лінією зі стрілкою.

Якщо з стану дії виходить єдиний перехід, то він може не бути позначений. Якщо таких переходів кілька, то може виконуватись тільки один із них. У цьому випадку для кожного з таких переходів має бути явно записано умову охорони в прямокутних дужках. Умова істинності повинна виконуватись тільки для одного з переходів. Така ситуація виникає, коли послідовно виконувана діяльність повинна розділитися на альтернативні гілки залежно від значення певного проміжного результату. Ця ситуація отримала назву гілкування, а для її позначення застосовується спеціальний символ.

Графічно гілкування на діаграмі діяльності позначається невеликим ромбом, всередині якого немає тексту. У цей ромб може входити тільки одна стрілка від того стану дії, після виконання якого потік управління має бути продовжений однією з взаємовиключних гілок. Прийнято входящу стрілку приєднувати до верхньої або лівої вершини символу гілкування. Вихідних стрілок може бути дві або більше, але для кожної з них явно вказується відповідна умова охорони у вигляді булевого виразу.

Один із недоліків звичайних блок-схем алгоритмів пов'язаний з проблемою зображення паралельних гілок окремих обчислень. Оскільки

паралелізація обчислень суттєво підвищує загальну швидкодію програмних систем, необхідні графічні примітиви для представлення паралельних процесів. В мові UML для цієї мети використовується спеціальний символ для розділення і злиття паралельних обчислень або потоків управління. Цей символ — пряма черточка. Як правило, така черточка зображується відрізком горизонтальної лінії, товщина якої кілька ширша за основні суцільні лінії діаграми діяльності. При цьому розділення (concurrent fork) має один вхідний перехід і кілька вихідних, а злиття (concurrent join) має кілька вхідних переходів і один вихідний.

2. Дорожки

Діаграми діяльності можуть бути використані не тільки для специфікації алгоритмів обчислень або потоків управління в програмних системах. Не менш важлива область їх застосування пов'язана з моделюванням бізнес-процесів. Дійсно, діяльність будь-якої організації також є сукупністю окремих дій, спрямованих на досягнення необхідного результату. Однак для бізнес-процесів бажано виконання кожної дії асоціювати з конкретним підрозділом компанії. У цьому випадку підрозділ несе відповідальність за реалізацію окремих дій, а сам бізнес-процес представляється у вигляді переходів дій з одного підрозділу до іншого.

Для моделювання цих особливостей в мові UML використовується спеціальна конструкція, що отримала назву дорожки (swimlanes). Мається на увазі візуальна аналогія з плавальними доріжками в басейні, якщо подивитись на відповідну діаграму. Усі стани дії на діаграмі діяльності діляться на окремі групи, які відокремлюються одна від одної вертикальними лініями. Дві сусідні лінії утворюють дорожку, а група станів між цими лініями виконується окремим підрозділом (відділом, групою, відділенням, філією) організації. Назви підрозділів явно вказуються в верхній частині дорожки.

Перетинати лінію дорожки можуть тільки переходи, які в цьому випадку позначають вихід або вхід потоку управління у відповідне підрозділ. Порядок

слідування дорожок не має жодної семантичної інформації і визначається зручністю.

3. Об'єкти

У загальному випадку дії на діаграмі діяльності виконуються над тими чи іншими об'єктами. Ці об'єкти або ініціюють виконання дій, або визначають певний їх результат. Дії специфікують виклики, які передаються від одного об'єкта графа діяльності до іншого. Оскільки в такому ракурсі об'єкти відіграють певну роль у розумінні процесу діяльності, іноді виникає необхідність явно вказати їх на діаграмі.

Для графічного представлення об'єктів використовується прямокутник класу, з тим відмінністю, що ім'я об'єкта підкреслюється. Далі після імені може вказуватись характеристика стану об'єкта в прямокутних дужках. Такі прямокутники об'єктів приєднуються до станів дії відносинами залежності пунктирною лінією зі стрілкою. Відповідна залежність визначає стан конкретного об'єкта після виконання попередньої дії.

На діаграмі діяльності з дорожками розташування об'єкта може мати певний додатковий сенс. А саме, якщо об'єкт розташований на межі двох дорожок, це може означати, що перехід до наступного стану дії в сусідній дорожці асоційований з готовністю деякого документа (об'єкт у певному стані). Якщо ж об'єкт повністю розташований всередині дорожки, то стан цього об'єкта повністю визначається діями цієї дорожки.

Для синхронізації окремих дій на діаграмі діяльності жодних додаткових позначень не використовується, оскільки синхронізація паралельних процесів може бути реалізована за допомогою переходів «розділення-злиття».

При розгляді діаграм стану та діяльності було зазначено, що хоча ці діаграми і використовуються для специфікації динаміки поведінки систем, час в явному вигляді в них не присутній. Однак часовий аспект поведінки може мати суттєве значення при моделюванні синхронних процесів, що описують

взаємодії об'єктів. Для моделювання взаємодії об'єктів у часі в мові UML використовуються діаграми послідовностей.

Об'єкти

На діаграмі послідовності зображаються лише ті об'єкти, які безпосередньо беруть участь у взаємодії. Ключовим моментом для діаграм послідовності є динаміка взаємодії об'єктів у часі.

У UML діаграма послідовності має як би два виміри. Перше — зліва направо у вигляді вертикальних ліній, кожна з яких зображає лінію життя окремого об'єкта, що бере участь у взаємодії. Крайнім зліва на діаграмі зображений об'єкт, що є ініціатором взаємодії. Правіше зображується інший об'єкт, що безпосередньо взаємодіє з першим. Таким чином, усі об'єкти на діаграмі послідовності утворюють певний порядок, визначений черговістю або рівнем активності об'єктів під час взаємодії один з одним.

Графічно кожен об'єкт зображується прямокутником і розташовується в верхній частині своєї лінії життя. Усередині прямокутника записуються ім'я об'єкта та ім'я класу, розділені двокрапкою. При цьому вся запис підкреслюється, що є ознакою об'єкта.

Другим виміром діаграми послідовності є вертикальна тимчасова вісь, спрямована згори вниз. Початковому моменту часу відповідає найвища частина діаграми. Взаємодії об'єктів здійснюються за допомогою повідомлень, які надсилаються одними об'єктами іншим. Повідомлення зображуються у вигляді горизонтальних стрілок із назвою повідомлення, а їхній порядок визначається часом виникнення. Тобто, повідомлення, що розташовані вище на діаграмі, ініціюються раніше тих, що розташовані нижче. Масштаб на осі часу не вказується, оскільки діаграма послідовності моделює лише тимчасову впорядкованість взаємодій типу «раніше-пізніше».

4. Лінія життя об'єкта

Лінія життя об'єкта (object lifeline) зображується пунктирною вертикальною лінією, асоційованою з єдиним об'єктом на діаграмі послідовності. Лінія життя служить для позначення періоду часу, протягом якого об'єкт існує в системі і, відповідно, може потенційно брати участь у всіх її взаємодіях. Якщо об'єкт існує в системі постійно, то й його лінія життя повинна продовжуватися по всій площині діаграми послідовності від найвищої її частини до найнижчої.

Окремі об'єкти, виконавши свою роль у системі, можуть бути знищені, щоб звільнити займані ними ресурси. Для таких об'єктів лінія життя обривається в момент їхнього знищення. Для позначення моменту знищення об'єкта в мові UML використовується спеціальний символ у формі латинської літери «X». Нижче цього символа пунктирна лінія не зображується, оскільки відповідного об'єкта в системі вже немає, і цей об'єкт повинен бути виключений з усіх наступних взаємодій.

Не обов'язково створювати всі об'єкти діаграми на початковий момент часу. Окремі об'єкти можуть створюватися за потребою, економлячи ресурси системи та підвищуючи її продуктивність. У цьому випадку прямокутник об'єкта зображується не в верхній частині діаграми послідовності, а в тій частині, яка відповідає моменту створення об'єкта. При цьому прямокутник об'єкта вертикально розташовується в тому місці діаграми, яке по осі часу співпадає з моментом його виникнення в системі. Об'єкт обов'язково створюється зі своєю лінією життя і, можливо, з фокусом управління.

5. Фокус управління

У процесі функціонування об'єктно-орієнтованих систем одні об'єкти можуть перебувати в активному стані, безпосередньо виконуючи певні дії, або в стані пасивного очікування повідомлень від інших об'єктів. Щоб явно виділити таку активність об'єктів, в мові UML застосовується спеціальне поняття, яке отримало назву фокусу управління (focus of control). Фокус управління зображується у формі витягнутого вузького прямокутника, верхня

сторона якого позначає початок отримання фокусу управління об'єкта (початок активності), а його нижня сторона — закінчення фокусу управління (закінчення активності).

Прямокутник розташовується нижче позначення відповідного об'єкта і може замінювати його лінію життя, якщо на всьому її протязі він є активним.

Періоди активності об'єкта можуть чергуватися з періодами його пасивності або очікування. У цьому випадку в такого об'єкта є кілька фокусів управління. Важливо усвідомлювати, що фокус управління може отримати лише існуючий об'єкт, у якого в цей момент є лінія життя. Якщо ж якийсь об'єкт був знищений, то він уже не може виникнути в системі. Замість нього може бути створений інший екземпляр того ж класу, який, строго кажучи, буде іншим об'єктом.

У окремих випадках ініціатором взаємодії в системі може бути актор або зовнішній користувач. У цьому випадку актор зображується на діаграмі послідовності найпершим об'єктом зліва зі своїм фокусом управління. Найчастіше актор і його фокус управління будуть існувати в системі постійно, позначаючи характерну для користувача активність у ініціюванні взаємодій з системою. При цьому актор може мати власне ім’я або залишатися анонімним. Іноді деякий об'єкт може ініціювати рекурсивне взаємодія з самим собою. Наявність у багатьох мовах програмування спеціальних засобів побудови рекурсивних процедур вимагає візуалізації відповідних понять у вигляді графічних примітивів. На діаграмі послідовності рекурсія позначається невеликим прямокутником, що приєднується до правої сторони фокуса управління того об'єкта, для якого зображується це рекурсивне взаємодія.

6. Повідомлення В UML кожне взаємодія описується сукупністю повідомлень, якими учасники взаємодії обмінюються між собою. Повідомлення (message) є завершеним фрагментом інформації, що відправляється одним об'єктом іншому. Приймання повідомлення ініціює виконання певних дій,

спрямованих на вирішення окремої задачі тим об'єктом, якому це повідомлення надіслано.

Таким чином, повідомлення не лише передають певну інформацію, а й вимагають або передбачають виконання очікуваних дій від приймаючого об'єкта. Повідомлення можуть ініціювати виконання операцій об'єктом відповідного класу, а параметри цих операцій передаються разом з повідомленням. На діаграмі послідовності всі повідомлення впорядковані за часом їх виникнення в моделюваній системі. У такому контексті кожне повідомлення має напрямок від об'єкта, який ініціює та надсилає повідомлення, до об'єкта, який його отримує. Іноді відправника повідомлення називають клієнтом, а отримувача – сервером. Тоді повідомлення від клієнта має форму запиту певного сервісу, а реакція сервера на запит після отримання повідомлення може бути пов’язана з виконанням певних дій або передачею клієнту необхідної інформації також у вигляді повідомлення.

Повідомлення зображуються горизонтальними стрілками, що з’єднують лінії життя або фокуси управління двох об'єктів на діаграмі послідовності. У мові UML розрізняють кілька різновидів повідомлень, кожне з яких має своє графічне зображення:

- перший різновид повідомлення є найбільш поширеним і використовується для виклику процедур, виконання операцій або позначення окремих вкладених потоків управління. Початок цієї стрілки завжди торкається фокуса управління або лінії життя того об'єкта-клієнта, який ініціює це повідомлення. Кінець стрілки торкається лінії життя того об'єкта, який приймає це повідомлення і виконує в відповідь певні дії. Приймаючий об'єкт зазвичай отримує фокус управління, стаючи активним;
- другий різновид повідомлення використовується для позначення простого потоку управління. Кожна така стрілка вказує на виконання одного кроку потоку. Такі повідомлення зазвичай є асинхронними, тобто можуть

виникати в будь-який момент часу. Передача такого повідомлення, як правило, супроводжується отриманням фокусу управління прийнятим об'єктом;

- третій різновид явно позначає асинхронне повідомлення між двома об'єктами в певній процедурній послідовності. Прикладом такого повідомлення може бути переривання операції при виникненні виключної ситуації. У цьому випадку інформація про таку ситуацію передається викликаючому об'єкту для продовження процесу подальшої взаємодії;
- четвертий різновид повідомлення використовується для повернення з виклику процедури. Прикладом може бути просте повідомлення про завершення певних обчислень без надання результату розрахунків об'єкту-клієнту. У процедурних потоках управління ця стрілка може бути опущена, оскільки її наявність неявно передбачається в кінці активізації об'єкта. Вважається, що кожен виклик процедури має свою пару – повернення виклику. Для непроцедурних потоків управління, включаючи паралельні та асинхронні повідомлення, стрілка повернення повинна вказуватися явним чином.

Передбачається, що час передачі повідомлення достатньо малий у порівнянні з процесами виконання дій об'єктами, тобто, за час передачі повідомлення з відповідними об'єктами не може відбутися ніяких змін. Якщо ж це припущення не можна визнати справедливим, то стрілка повідомлення зображується під певним нахилом, так щоб кінець стрілки розташовувався нижче її початку.

У окремих випадках об'єкт може посилати повідомлення самому собі, ініціюючи так звані рефлексивні повідомлення. Такі повідомлення зображуються прямокутником зі стрілкою, початок і кінець якої збігаються. Подібні ситуації виникають, наприклад, при обробці натискань на клавіші клавіатури під час введення тексту в редагований документ, при наборі цифр номера телефону абонента.

Таким чином, в мові UML кожне повідомлення асоціюється з певною дією, яку має виконати приймаючий його об'єкт. Дія може мати деякі аргументи або параметри, залежно від конкретних значень яких може бути отриманий різний результат. Відповідні параметри буде мати і викликаюче це дію повідомлення. Більше того, значення параметрів окремих повідомлень можуть містити умовні вирази, утворюючи гілкування або альтернативні шляхи основного потоку управління.

7. Гілкування потоку управління Для зображення гілкування потоку управління малюються дві або більше стрілок, що виходять з однієї точки фокуса управління об'єкта. При цьому відповідні умови повинні бути явно вказані поруч з кожною з стрілок у вигляді сторожового умови. Сторожові умови повинні взаємно виключати одночасну передачу альтернативних повідомлень.

8. Стереотипи повідомлень У мові UML передбачено деякі стандартні дії, що виконуються у відповідь на отримання відповідного повідомлення. Ці дії можуть бути явно вказані на діаграмі послідовності у вигляді стереотипу поруч з повідомленням, до якого вони відносяться. У цьому випадку вони записуються в лапках. Використовуються наступні стереотипи повідомлень:

- «call» (виклик) – повідомлення, що вимагає виклику операції або процедури приймаючого об'єкта. Якщо повідомлення з цим стереотипом рефлексивне, то воно ініціює локальний виклик операції у самого надіславшого це повідомлення об'єкта;
- «return» (повернути) – повідомлення, що повертає значення виконаної операції або процедури викликавшому її об'єкту. Значення результату може ініціювати гілкування потоку управління;
- «create» (створити) – повідомлення, що вимагає створення іншого об'єкта для виконання певних дій. Створений об'єкт може отримати фокус управління, а може і не отримати його;

- «destroy» (уничтожити) – повідомлення з явною вимогою знищити відповідний об'єкт. Надсилається в тому випадку, коли необхідно припинити небажані дії зі сторони існуючого в системі об'єкта, або коли об'єкт більше не потрібен і повинен звільнити задіяні ним системні ресурси;
- «send» (послати) – позначає посилку іншому об'єкту певного сигналу, який асинхронно ініціюється одним об'єктом і приймається іншим. Відмінність сигналу від повідомлення полягає в тому, що сигнал повинен бути явно описаний в тому класі, об'єкт якого ініціює його передачу.

Окрім стереотипів, повідомлення можуть мати власне позначення операції, виклик якої вони ініціюють у приймаючого об'єкта. У такому разі поруч зі стрілкою записується ім'я операції в круглих дужках, в яких можуть вказуватися параметри або аргументи відповідної операції. Якщо параметри відсутні, то дужки все одно повинні бути після імені операції.

9. Часові обмеження на діаграмах послідовності У окремих випадках виконання тих чи інших дій на діаграмі послідовності може вимагати явної специфікації часових обмежень, накладаються на сам інтервал виконання операцій або передачу повідомлень. У мові UML для запису часових обмежень використовуються фігурні дужки. Часові обмеження можуть стосуватися як виконання певних дій об'єктами, так і самих повідомлень, явно специфікуючи умови їх передачі або прийому. На відміну від умов гілкування, які повинні виконуватись альтернативно, часові обмеження мають обов'язковий або директивний характер для асоційованих з ними об'єктів.

Часові обмеження можуть записуватись поруч з початком стрілки відповідного повідомлення. Але найчастіше вони записуються ліворуч від стрілки на одному рівні з нею. Якщо тимчасова характеристика стосується конкретного об'єкта, то ім’я цього об'єкта записується перед ім'ям характеристики і відділяється від неї крапкою.

10. Коментарі або примітки Коментарі або примітки можуть включатись у діаграми послідовності, асоціюючись з окремими об'єктами або повідомленнями. При цьому використовується стандартне позначення для коментаря у вигляді прямокутника з загнутим правим верхнім кутом. Всередині прямокутника записується текст коментаря на природній мові.

Завдання

1. Ознайомитися з короткими теоретичними відомостями.

2. Розробити діаграму розгортання для проектованої системи.

3. Розробити діаграму компонентів для проектованої системи.

4. Розробити діаграму послідовностей для проектованої системи.

5. Скласти звіт про виконану роботу

Обрана тема

15 E-mail клієнт (singleton, builder, decorator, template method, interpreter, SOA)

Поштовий клієнт повинен нагадувати функціонал поштових програм Mozilla Thunderbird, The Bat і т.д. Він повинен сприймати і коректно обробляти pop3/smtp/imap протоколи, мати функції автонастройки основних поштових провайдерів для України (gmail, ukr.net, i.ua), розділяти повідомлення на папки/категорії/важливість, зберігати чернетки незавершених повідомлень, прикріплювати і обробляти прикріплені файли.

Хід роботи

Завдання №2

Діаграма розгортання:

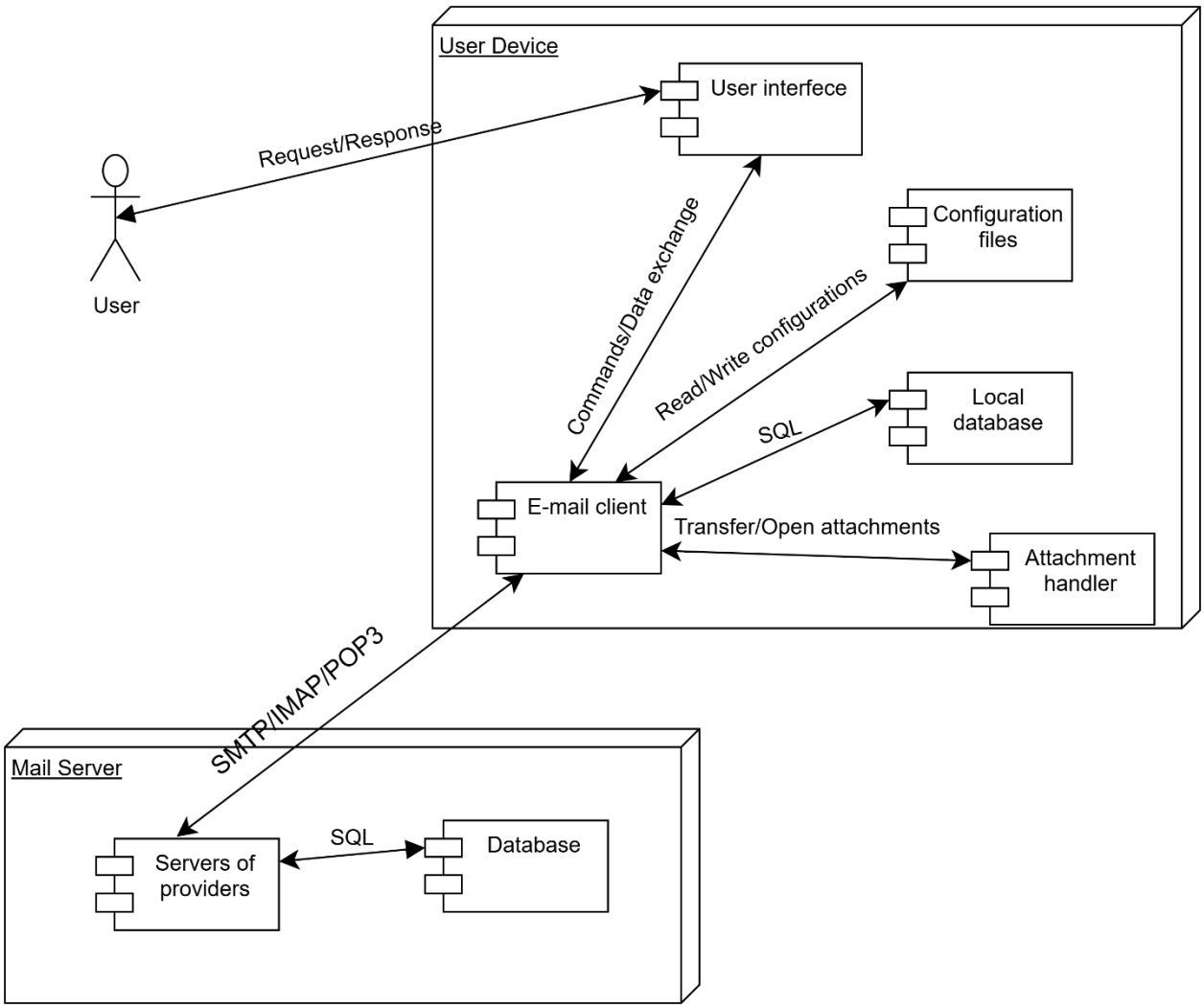

Рисунок 1. Діаграма розгортання

User Device (Користувацький пристрій):

- User Interface (Інтерфейс користувача): Взаємодіє з користувачем, забезпечуючи управління електронною поштою (написання, перегляд, сортування повідомлень тощо).
- E-mail client (Клієнт електронної пошти): Основний компонент, який здійснює зв’язок між усіма іншими компонентами системи. Відповідає за комунікацію з серверами провайдерів, роботу з базами даних, обробку конфігурацій і вкладень.

Mail Server (Поштовий сервер):

- Servers of Providers (Сервери провайдерів): Здійснюють зберігання та доставку електронних листів. Використовують протоколи (наприклад, SMTP, IMAP, POP3) для взаємодії з клієнтом електронної пошти.
- Database (База даних): Зберігає інформацію, пов’язану з листами, акаунтами користувачів тощо.

Локальні компоненти:

- Configuration Files (Файли конфігурацій): Зберігають налаштування клієнта електронної пошти (акаунти, сервери, параметри синхронізації тощо). Клієнт читає та записує дані конфігурації через цей компонент.
- Local Database (Локальна база даних): Використовується для збереження локальних копій електронних листів, чернеток, історії повідомлень і категоризації. Взаємодіє з клієнтом через SQL-запити.
- Attachment Handler (Обробник вкладень): Обробляє додаткові файли, прикріплені до листів, дозволяючи завантажувати та відкривати їх.

Основні взаємодії між компонентами

User Interface ↔ E-mail client:

Користувач надсилає команди (написання листів, завантаження повідомлень) через інтерфейс, а клієнт обробляє їх і передає відповідні запити до серверів чи інших компонентів.

E-mail client ↔ Servers of Providers:

Через протоколи SMTP, IMAP або POP3 клієнт надсилає або отримує електронні листи від серверів провайдерів.

E-mail client ↔ Local Database:

Зберігає або отримує дані (наприклад, електронні листи, чернетки) для локального доступу користувача.

E-mail client ↔ Configuration Files:

Читає/записує налаштування користувача.

E-mail client ↔ Attachment Handler:

Забезпечує відкриття, збереження та обробку вкладень.

Завдання №3

Діаграма компонентів:

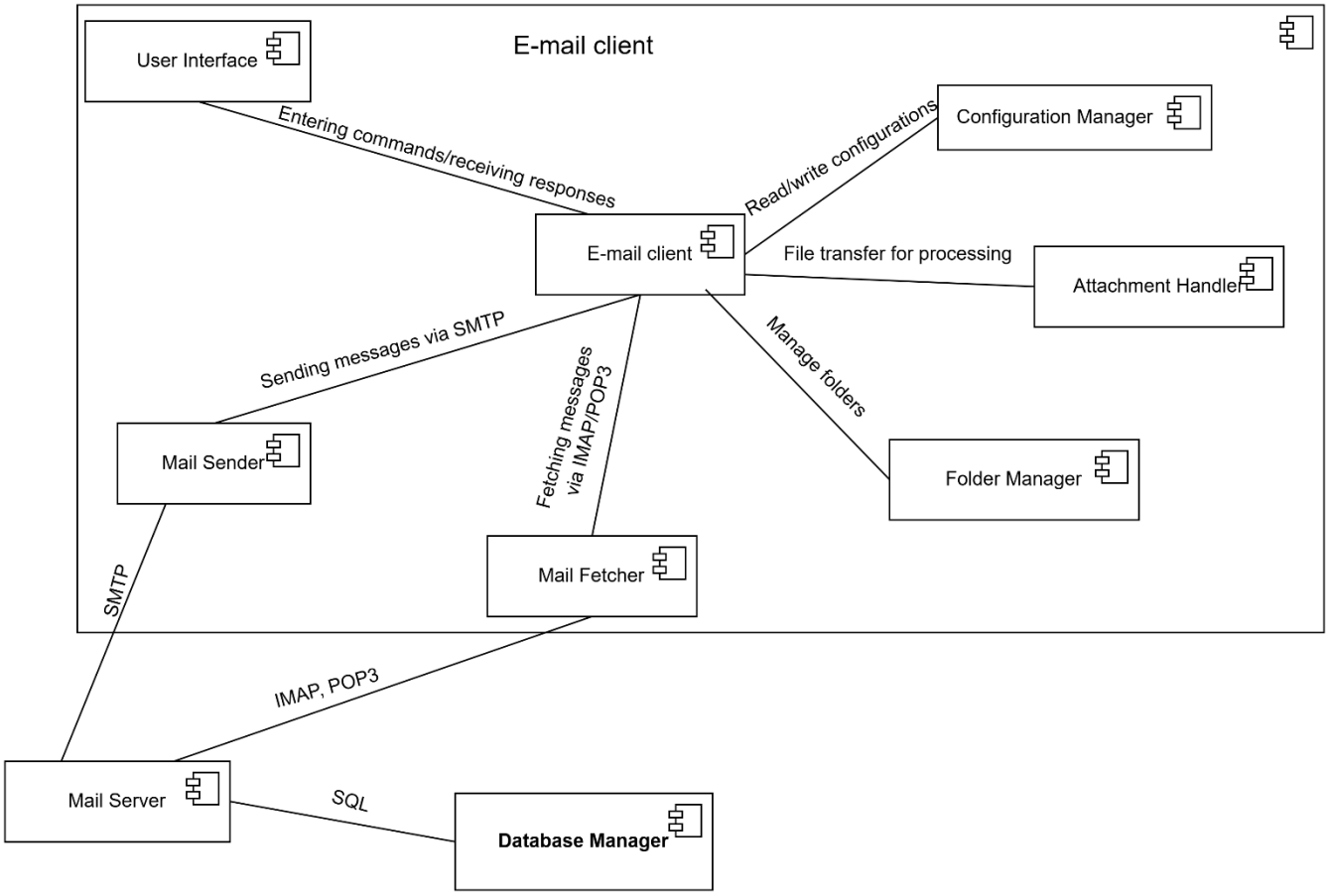

Рисунок 2. Діаграма компонентів

User Interface (Інтерфейс користувача): Компонент, через який користувач взаємодіє з клієнтом електронної пошти. Приймає команди від користувача (написання листів, перегляд, управління папками) та передає відповіді.

E-mail Client (Клієнт електронної пошти): Центральний компонент, який координує роботу всієї системи. Взаємодіє з іншими компонентами для

виконання основних функцій, таких як надсилання, отримання, обробка листів, а також управління папками та вкладеннями.

Mail Sender (Відправник пошти): Відповідає за надсилання електронних листів. Використовує протокол SMTP для передачі повідомлень до сервера електронної пошти.

Mail Fetcher (Отримувач пошти): Займається отриманням електронних листів із сервера. Підтримує протоколи IMAP та POP3 для завантаження листів.

Configuration Manager (Менеджер конфігурацій): Зберігає та управляє параметрами системи (налаштування облікових записів, параметри синхронізації тощо). Забезпечує зчитування та запис конфігурацій на запит клієнта.

Attachment Handler (Обробник вкладень): Відповідає за обробку файлів, прикріплених до листів. Дозволяє завантажувати та відкривати вкладення.

Folder Manager (Менеджер папок): Керує організацією листів у папках (вхідні, вихідні, чернетки, користувацькі папки). Відповідає за створення, видалення та сортування папок.

Mail Server (Поштовий сервер): Віддалений сервер, який зберігає електронні листи та забезпечує їх доставку. Підключається до клієнта через SMTP, IMAP або POP3.

Database Manager (Менеджер бази даних): Здійснює зберігання локальних даних, таких як електронні листи, чернетки, історія повідомлень тощо. Використовує SQL-запити для збереження та отримання інформації.

Основні взаємодії

User Interface ↔ E-mail Client: Користувач передає команди клієнту, а клієнт відображає відповіді через інтерфейс.

E-mail Client ↔ Mail Sender: Для надсилання електронних листів клієнт передає дані Mail Sender'у, який відправляє їх через SMTP.

E-mail Client ↔ Mail Fetcher: Клієнт запитує отримання нових листів через Fetcher, який завантажує їх з сервера через IMAP/POP3.

E-mail Client ↔ Configuration Manager: Здійснюється зчитування або оновлення параметрів конфігурації клієнта.

E-mail Client ↔ Attachment Handler: Використовується для обробки вкладень під час перегляду або збереження електронних листів.

E-mail Client ↔ Folder Manager: Забезпечується управління папками (створення, видалення, переміщення листів).

Mail Sender ↔ Mail Server: Передає повідомлення до сервера електронної пошти.

Mail Fetcher ↔ Mail Server: Завантажує повідомлення з сервера.

E-mail Client ↔ Database Manager: Локально зберігає електронні листи, чернетки та інші дані.

Завдання №4

Діаграма послідовностей:

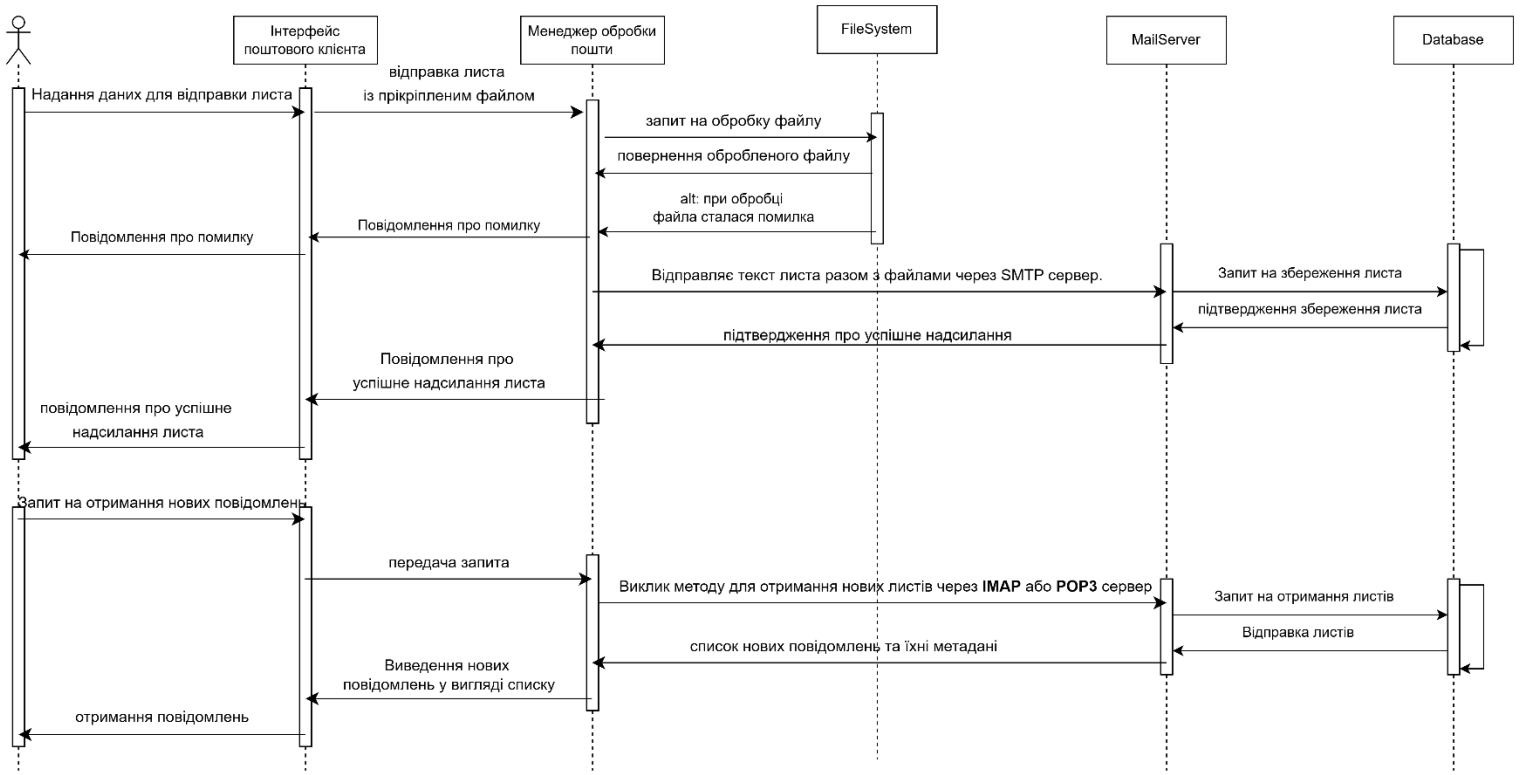

Рисунок 3. Діаграма послідовностей

Опис учасників

Інтерфейс поштового клієнта:

- Забезпечує взаємодію користувача з системою.
- Передає команди (наприклад, відправлення листа) до інших компонентів.

Менеджер обробки пошти:

- Відповідає за основну обробку листів.
- Організовує отримання, відправлення та зберігання повідомлень.

FileSystem (Файлова система): Використовується для роботи з файлами, наприклад, вкладеннями або локальними даними.

MailServer (Поштовий сервер): Зовнішній сервер, який зберігає листи, обробляє запити на відправлення або отримання.

Database (База даних): база даних, яка використовується для зберігання інформації.

Сценарій взаємодії

Ініціація дії користувачем:

- Користувач через інтерфейс поштового клієнта створює запит (наприклад, відправлення листа).
- Інтерфейс передає запит до Менеджера обробки пошти.

Обробка запиту менеджером:

- Менеджер аналізує запит і визначає, які дії необхідно виконати (наприклад, перевірити дані листа, прикріплені файли тощо).
- Робота з файловою системою: Якщо є вкладення, менеджер звертається до FileSystem, щоб отримати файли для передачі.

Передача даних на сервер:

- Менеджер ініціює відправку листа, надсилаючи запит до MailServer через відповідний протокол (SMTP).
- Сервер підтверджує успішне відправлення.

Робота з базою даних:

Менеджер обробки пошти записує інформацію про відправлений лист у Database.

Завершення сценарію:

Після виконання всіх дій менеджер повертає відповідь інтерфейсу, який повідомляє користувача про результат.

Висновок

У ході лабораторної роботи було розроблено діаграми UML для поштового клієнта: діаграма розгортання, діаграма компонентів, діаграма послідовностей. Це дозволило візуалізувати фізичну архітектуру системи,

логічне розбиття на компоненти, взаємодію об’єктів та сценарії виконання основних функцій.