МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: **«ДОСЛІДЖЕННЯ ШАБЛОНІВ ПРОЕКТУВАННЯ ДЛЯ ІНТЕГРАЦІЇ У ДОДАТКАХ КОРПОРАТИВНОГО РІВНЯ»**

Виконав: студент 2 курсу, групи 8.1218

спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)

освітньої програми інженерія програмного забезпечення
(назва освітньої програми)

Д.А. Саржан
(ініціали та прізвище)

Керівник завідувач кафедри програмної інженерії, доцент, к.ф.-м.н. Лісняк А.О.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент доцент кафедри фундаментальної математики, доцент, к.ф.-м.н. Панасенко Є.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2020

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти магістр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма інженерія програмного забезпечення

ЗАТВЕРДЖУЮ

Завідувач кафедри програмної інженерії, доцент, к.ф.-м.н.

_____ Лісняк А.О.
(підпис)

« 29 » травня 2019 р

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Саржану Дмитру Андрійовичу
(прізвище, ім'я та по-батькові)

1. Тема роботи Дослідження шаблонів проектування для інтеграції у додатках корпоративного рівня

керівник роботи Лісняк Андрій Олекандрович, к.ф.-м.н., доцент
(прізвище, ім'я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 29 » травня 2019 року № 811-с

2. Строк подання студентом роботи 26 грудня 2019 року

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі
2. Основні теоретичні відомості

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Презентація

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання 29.05.2019

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	05.09.2019	
2.	Збір вихідних даних.	13.09.2019	
3.	Обробка методичних та теоретичних	29.09.2019	
	джерел.		
4.	Розробка першого та другого розділу.	10.10.2019	
5.	Розробка третього розділу.	23.11.2019	
6.	Оформлення та нормоконтроль	27.12.2019	
	кваліфікаційної роботи.		
7.	Захист кваліфікаційної роботи.	15.01.2020	

Студент _____ Д.А. Саржан
(підпис) (ініціали та прізвище)

Керівник роботи _____ А.О. Лісняк
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ О.В. Кудін
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота магістра «Дослідження шаблонів проектування для інтеграції у додатках корпоративного рівня»: 56 сторінки, 34 рисунки, 12 джерел.

ENTERPRISE INTEGRATION PATTENS, HTTP, API, SPRING, JAVA, SPRING INTEGRATION, APACHE CAMEL, MESSAGE BROKER, JMS, WEB SERVICE, XML, GATEWAY, CSV, FTP.

Об’єкт дослідження – патерни та засоби інтеграції додатків корпоративного рівня.

Мета роботи: розробка дизайну системи інтеграції інтернет-магазину з зовнішніми сервісами.

Метод дослідження – аналітичний.

У роботі досліджуються патерни проектування для інтеграції у додатках корпоративного рівня. Розглядаються різні засоби та технології для вирішення проблем інтеграції, запропоноване рішення на основі якого розробляються дизайн системи.

В роботі представлен детальний огляд технологій і засобів для інтеграції додатків корпоративного рівня на основі Enterprise integration patters.

SUMMARY

Master’s Qualifying Thesis «Researh of Design Patterns for Corporative Software Integration»: 56 pages, 34 images, 12 sources.

APACHE CAMEL, API, CSV, ENTERPRISE INTEGRATION, FTP, GATEWAY, HTTP, JAVA, JMS, MESSAGE BROKER, PATTERNS, SPRING, SPRING INTEGRATION, WEB SERVICE, XML.

The objects of the study are patterns and tools for integration enterprise applications.

The aim of the study is design of system integration for online store with external services.

The method of research is analytical.

The project explores design patterns for integration of enterprise-applications. There are considering various tools and technologies which solve the integration problems, based on it here present system design for e-commerce system.

The project provides a detailed overview of technologies and tools for enterprise-level application integration based on Enterprise integration patters.

ЗМІСТ

Завдання на кваліфікаційну роботу 2
Реферат 4
Summary 5
Вступ 7
1 Патерни проектування 8
1.1. Види шаблонів проектування 8
1.2 Інтеграційні рішення для технологій обміну повідомленнями 9
1.3 Обмін повідомленнями 12
1.4 Використання повідомлень 18
2 Шаблони обміну повідомленнями 23
2.1 Вступ до стилів інтеграції 23
2.1.1 Критерії інтеграції додатків 23
2.1.2 Параметри інтеграції додатків 25
2.2 Вступ в систему обміну повідомленнями 26
2.3 Канали обміну повідомленнями 28
2.4 Структура повідомлень 34
2.5 Управління системою 38
2.6 Вступ до кінцевих точок повідомлень 44
3 Практичні застосування шаблонів для інтеграції на прикладі інтернет-магазину 49
Перелік посилань 56

ВСТУП

Сьогодні програми рідко живуть ізольовано. Користувачі очікують миттєвого доступу до всіх функцій, які можуть бути надані різними програмами та послугами всередині або поза підприємством. Інтеграція програм та служб залишається складнішою, ніж повинна бути.

Інтеграція підприємства – це завдання змусити окремі програми працювати разом, щоб створити єдиний набір функціональних можливостей. Програми, ймовірно, працюють на декількох комп'ютерах, які можуть представляти кілька платформ, і можуть бути географічно розповсюджені. Деякі програми можуть запускатися за межами підприємства бізнес-партнерами або замовниками. Деякі програми можуть знадобитися інтегрувати, хоча вони не були розроблені для інтеграції і не можуть бути змінені. Ці проблеми та інші подібні до них ускладнюють інтеграцію додатків.

Метою роботи є розробка дизайну системи інтеграції інтернет-магазину з зовнішніми сервісами.

Задачі:

- огляд шаблонів програмування;
- аналіз шаблонів що використовуються для відправки повідомлень;
- аналіз параметрів повідомлення;
- практичне застосування шаблонів у системах електронної комерції.

Об’єктом дослідження є патерни та засоби інтеграції додатків корпоративного рівня.

Розробка дизайну буде проведена на основі шаблонів інтеграції. За вимогами система повинна бути гнучкою та легко розширюваною у випадку зміну вимог до неї

1 ПАТЕРНИ ПРОЕКТУВАННЯ

Патерни проектування – це напрацьовані ефективні підходи, техніки та правила вирішення задач при створенні програмного забезпечення. Вони не прив’язуються до певної мови програмування і можуть бути застосованими в основному незалежно від конкретної мови.

1.1. Види шаблонів проектування

Основа шаблонів містить 65 моделей, які утворюють мову шаблонів. Мова шаблону – це мережа споріднених шаблонів, де кожен шаблон призводить до інших, направляючи читача в процесі прийняття рішень. Цей підхід є потужною технікою документування знань експерта, щоб його можна було легко зрозуміти та застосувати не експерту.

Мова шаблону вчить читача, як вирішувати необмежену різноманітність проблем у обмеженому просторі. Оскільки загальна проблема, яка вирішується, щоразу різна, шлях через шаблони та спосіб їх застосування також унікальний. Використання будь-яких засоби обміну повідомленнями чи інтеграцію для будь-яких цілей, але може застосовуватися спеціально для вас та конкретного застосування повідомлень, з якими ви стикаєтеся [1].

Шаблони описують загальноприйняті рішення постійних проблем, тому якщо ви досвідчений розробник рішень для інтеграції, орієнтованих на повідомлення, багато з цих моделей вам здадуться знайомими. Але навіть якщо ви вже розпізнаєте більшість цих моделей. Це дає вам консолідовану довідку, яка допоможе вам ефективно передати свої знання менш досвідченим колегам. Він також документує деталі рішень та відносини між ними їх, про які ви, можливо, не знали.

1.2 Інтеграційні рішення для технологій обміну повідомленнями

Товар розроблений для розробників та інтеграторів, що використовують різні продукти та технології обміну повідомленнями, такі як:

- програмне забезпечення, орієнтоване на повідомлення (MOM) та комплекти інтеграції, пропоновані такими постачальниками, як IBM (WebSphere MQ Family), Microsoft (BizTalk), TIBCO, WebMethods, SeeBeyond, Vitria та інші;
- реалізація Java Message Service (JMS), включена в комерційні та відкриті сервери додатків J2EE, а також окремі продукти;
- черга повідомлень Microsoft (MSMQ), доступна через декілька API, включаючи бібліотеки System.Messaging у Microsoft.NET;
- нові стандарти веб-служб, які підтримують асинхронні веб-сервіси (наприклад, WS-надійний Messaging) та пов'язані з ними API, такі як Java-API Sun для обміну повідомленнями XML (JAXM) Sun або розширення веб-служб Microsoft (WSE).

Інтеграція підприємства виходить за межі створення єдиного додатка з розподіленою n-ярусною архітектурою, що дозволяє розподілити одну програму на декількох комп'ютерах. У той час як один рівень в розподіленій програмі не може запускатися сам по собі, інтегровані програми - це незалежні програми, які можуть виконуватись самостійно, але вони функціонують, координуючись між собою слабко пов'язаними способами. Повідомлення дозволяє передавати дані або команди по всій мережі, використовуючи підхід «надіслати і забути», коли абонент надсилає інформацію, а потім переходить до іншої роботи, поки інформація передається системою обміну повідомленнями. За бажанням, абонент може пізніше бути повідомлений про результат за допомогою зворотного дзвінка.

Асинхронні виклики та зворотні дзвінки можуть зробити дизайн складнішим, ніж синхронний підхід, але асинхронний дзвінок можна повторити, поки він не вдається, що робить спілкування набагато надійнішим. Асинхронний обмін повідомленнями також дає ряд інших переваг, таких як дроселювання запитів та врівноваження навантаження [2].

Інтегрування корпоративних програм здійснюється, розуміючи:

- переваги та обмеження обміну повідомленнями порівняно з іншими методами інтеграції;
- як визначити канали повідомлень, які знадобляться вашим програмам, як контролювати, чи можуть кілька споживачів отримувати одне і те ж повідомлення та як обробляти недійсні повідомлення;
- коли надсилати повідомлення, що воно повинно містити та як використовувати спеціальні властивості повідомлення;
- як перенаправити повідомлення до його кінцевого пункту призначення, навіть коли відправник не знає, де це;
- як конвертувати повідомлення, коли відправник та одержувач не домовляються про загальний формат;
- як створити код, який з'єднує додаток до системи обміну повідомленнями;
- як керувати та контролювати систему обміну повідомленнями, коли вона використовується у складі підприємства.

Цікаві програми рідко живуть ізольовано. Незалежно від того, чи має ваша заявка на продаж взаємодіяти з вашою інвентарною заявкою, ваша заявка на закупівлю повинна підключатися до аукціонного сайту, або PIM вашого PDA повинен синхронізуватися з сервером корпоративного календаря [2].

Усі інтеграційні рішення повинні вирішувати кілька основних завдань:

- мережі ненадійні. Інтеграційні рішення повинні транспортувати дані з одного комп'ютера на інший через мережі. Порівняно з процесом, що працює на одному комп’ютері, розподілені обчислення повинні бути готові

до вирішення набагато більшого набору можливих проблем. Часто дві системи, які потрібно інтегрувати, розділені континентами, і дані між ними мають проходити через телефонні лінії, сегменти локальної мережі, маршрутизатори, комутатори, мережі загального користування та супутникові зв’язки. Кожен з цих кроків може спричинити затримки або перебої;

- мережі повільні. Надсилання даних через мережу на кілька порядків повільніше ніж виклик локального методу. Розробляючи широко розповсюджене рішення так само, як ви підходили б до однієї програми, це може мати катастрофічні наслідки для продуктивності;
- будь-які дві програми різні. Інтеграційні рішення повинні передавати інформацію між системами, що використовують різні мови програмування, операційні платформи та формати даних. Інтеграційне рішення повинне мати можливість взаємодіяти з усіма цими різними технологіями;
- зміни неминучі. Програми змінюються з часом. Рішення для інтеграції повинно йти в ногу зі змінами в додатках, до яких він підключається. Інтеграційні рішення легко можуть потрапити в лавиноподібний ефект змін - якщо одна система зміниться, всі інші системи можуть постраждати. Рішення інтеграції повинно мінімізувати залежності від однієї системи до іншої, використовуючи нещільне з'єднання між додатками.

З часом розробники подолали ці виклики за допомогою чотирьох основних підходів [3]:

- передача файлів – одна програма записує файл, який інший читає пізніше. Програми повинні узгодити ім'я та місце розташування, формат файлу, терміни, коли він буде записаний та прочитаний, та хто видалить файл;
- спільна база даних – кілька додатків поділяють одну і ту ж схему бази даних, розташовану в одній фізичній базі даних. Оскільки не існує

дублікату зберігання даних, жодні дані не повинні передаватися з однієї програми в іншу;

- виклик віддалених процедур – один додаток демонструє деяку його функціональність, щоб доступ до нього можна було віддалено дістати іншими програмами як віддалену процедуру. Спілкування відбувається в режимі реального часу та синхронно [3];
- повідомлення – одні програми публікують повідомлення на загальному каналі повідомлень. Інші програми можуть пізніше прочитати повідомлення з каналу. Заявки повинні домовитися про канал, а також про формат повідомлення, зв'язок асинхронний.

Хоча всі чотири підходи вирішують по суті одну і ту ж проблему, кожен стиль має свої виразні переваги та недоліки. Насправді програми можуть інтегруватися за допомогою декількох стилів, щоб кожна точка інтеграції використовувала перевагу стилю, який їй найбільше підходить.

1.3 Обмін повідомленнями

На прикладі обміну повідомленнями розглянемо інтеграцію. Простий спосіб зрозуміти, що робить обмін повідомленнями – це розглянути телефонну систему. Телефонний дзвінок – це синхронна форма зв'язку. Є можливість спілкуватися з іншою стороною лише в тому випадку, якщо інша сторона доступна в момент, коли я здійснюю дзвінок. З іншого боку, голосова пошта дозволяє асинхронне спілкування. За допомогою голосової пошти, коли приймач не відповідає, абонент може залишити йому повідомлення; пізніше одержувач (за його зручністю) може прослуховувати повідомлення, що стоять у черзі, у його поштовій скриньці. Голосова пошта дозволяє абоненту залишити повідомлення зараз, щоб приймач міг його слухати пізніше, що набагато простіше, ніж намагатися одночасно зателефонувати або телефоністу або телефону, що телефонує. Пакети

голосової пошти (принаймні, частина) телефонного дзвінка в повідомлення та чергують його для подальшого споживання; це по суті, як працює обмін повідомленнями [1].

Повідомлення – це технологія, яка забезпечує високошвидкісний, асинхронний зв’язок між програмою та надійною доставкою. Програми спілкуються, посилаючи один одному пакети даних, що називаються повідомленнями. Канали, також відомі як черги – це логічні шляхи, що з'єднують програми та передають повідомлення. Канал веде себе як колекція або масив повідомлень, але такий, який магічним чином ділиться на декількох комп'ютерах і може одночасно використовуватись у кількох програмах. Відправник або виробник – це програма, яка надсилає повідомлення, записуючи повідомлення на канал. Одержувач або споживач - це програма, яка отримує повідомлення, читаючи (і видаляючи) його з каналу.

Саме повідомлення – це просто якась структура даних – наприклад, рядок, байтовий масив, запис чи об’єкт. Це можна інтерпретувати просто як дані, як опис команди, яку потрібно викликати на приймачі, або як опис події, що сталася у відправника. Повідомлення насправді містить дві частини – заголовок та тіло. У заголовку міститься метаінформація про повідомлення – хто його надіслав, куди йде тощо; ця інформація використовується системою обміну повідомленнями і в основному (але не завжди) ігнорується програмами, що використовують повідомлення. Тіло містить дані, що передаються, а система обміну повідомленнями ігнорується. У розмові, коли розробник програми, який використовує обмін повідомленнями, розповідає про повідомлення, він зазвичай посилається на дані в тілі повідомлення.

Асинхронні архітектури обміну повідомленнями є потужними, але вимагають від нас переосмислення підходу до розробки. Порівняно з іншими трьома підходами до інтеграції порівняно небагато розробників зазнали впливу систем обміну повідомленнями та повідомленнями. Як результат,

розробники додатків взагалі не так знайомі з ідіомами та особливостями цієї комунікаційної платформи [4].

Можливості обміну повідомленнями зазвичай надаються окремою програмною системою, що називається системою обміну повідомленнями (СОП) або програмним забезпеченням, орієнтованим на повідомлення. Система обміну повідомленнями управляє повідомленнями способом управління системою баз даних. Як адміністратор повинен заповнити базу даних схемою даних програми, так і адміністратор повинен налаштувати систему обміну повідомленнями з каналами, що визначають шляхи зв'язку між програмами. Потім система обміну повідомленнями координує та управляє надсиланням та отриманням повідомлень. Основна мета бази даних – забезпечити безпечне зберігання кожного запису даних, а також головним завданням системи обміну повідомленнями є надійне переміщення повідомлень з комп'ютера відправника на комп'ютер одержувача.

Причина, яка потрібна системі обміну повідомленнями для переміщення повідомлень з одного комп’ютера на інший, полягає в тому, що комп’ютери та мережі, які їх з'єднують, по суті не є надійними. Тільки тому, що одна програма готова надіслати повідомлення, не означає, що інша програма готова її отримати. Навіть якщо обидва додатки готові, мережа може не працювати або може передати дані належним чином. Система обміну повідомленнями долає ці обмеження, намагаючись передати повідомлення, поки це не вдасться. За ідеальних обставин повідомлення передається успішно з першої спроби, але обставини часто не є ідеальними.

По суті, повідомлення передається в п'ять кроків:

1) створити – відправник створює повідомлення та заповнює його даними;

2) надіслати – відправник додає повідомлення до каналу;

3) доставити – система обміну повідомленнями переміщує повідомлення з комп’ютера відправника на комп'ютер одержувача, роблячи його доступним для одержувача;

4) отримати – одержувач зчитує повідомлення з каналу;

5) процес – одержувач витягує дані з повідомлення.

На рисунку 1.1 зображена схема передачі даних ілюструє, що складається з п’яти етапів, які комп’ютер виконує кожен, а які етапи передбачає система обміну повідомленнями:

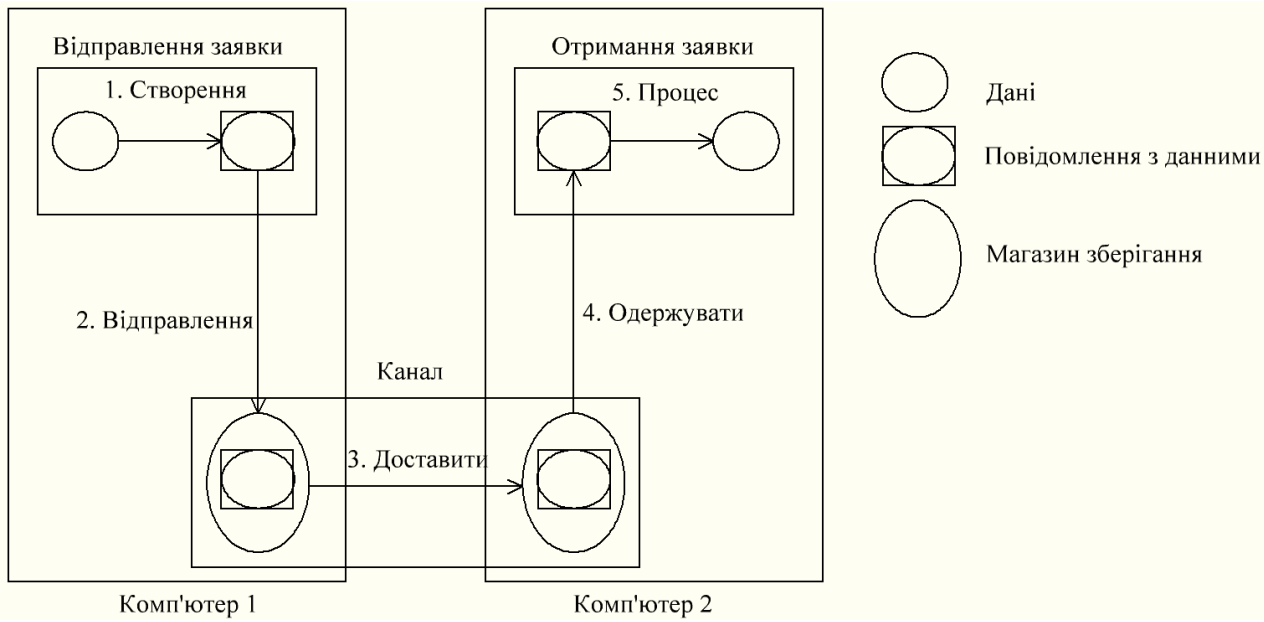

Рисунок 1.1 – Поетапна передача повідомлень

Ця діаграма також ілюструє дві важливі концепції обміну повідомленнями:

1) надіслати та забути – на кроці 2 програма, що надсилає, надсилає повідомлення на канал повідомлення. Після того, як відправлення завершено, відправник може перейти до іншої роботи, поки система обміну повідомленнями передає повідомлення у фоновому режимі. Відправник може бути впевнений, що одержувач зрештою отримає повідомлення, і не потрібно чекати, поки це станеться.

2) зберігання та пересилання – на кроці 2, коли програма, що надсилає, надсилає повідомлення на канал повідомлення, система обміну повідомленнями зберігає повідомлення на комп’ютері відправника, або в пам'яті, або на диску. На кроці 3 система обміну повідомленнями доставляє

повідомлення, пересилаючи його з комп'ютера відправника на комп'ютер одержувача, а потім знову зберігає повідомлення на комп'ютері одержувача. Цей процес зберігання і пересилання може повторюватися багато разів, коли повідомлення переміщується з одного комп'ютера на інший, поки не потрапить на комп'ютер приймача [5].

Етапи створення, надсилання, отримання та обробки можуть здатися непотрібними накладними. Чому б просто не доставити дані до приймача? Обгортаючи дані як повідомлення та зберігаючи їх у системі обміну повідомленнями, програми делегують системі обміну повідомленнями відповідальність за доставку даних. Оскільки дані обгортаються як атомне повідомлення, доставку можна повторити, поки це не вдасться, і одержувач може бути впевнений у надійному отриманні точно однієї копії даних.

Існує велика кількість шаблонів обміну повідомленнями. Розглянемо основні шаблони обміну повідомленнями, які використовуються в системах обміну повідомленнями є типовими рішеннями, що дозволяють інтегрувати їх в додатки корпоративного рівня.

Вони діляться на такі типи [1]:

1) інтеграційні стилі;

2) системи відправки повідомлень;

3) канали обміну повідомленнями;

4) структура повідомлень;

5) інтерлюдія: прості повідомлення

6) маршрутизація повідомлень;

7) перетворення повідомлень;

8) інтермедія: складене повідомлення;

9) пункти призначення повідомлень;

10) управління системою;

11) інтермедія: приклад управління системою;

12) інтеграція шаблонів на практиці.

Інтеграційні стилі поділяють на:

1) вступ до стилів інтеграції;

2) передача файлів;

3) спільна передача даних;

4) віддалений виклик процедур;

5) повідомлення.

Системи відправки повідомлень:

1) вступ до систем обміну повідомленнями;

2) канал повідомлення;

3) повідомлення;

4) труби та фільтри;

5) маршрутизатор повідомлень;

6) перекладач повідомлень;

7) кінцева точка повідомлення.

Канали обміну повідомленнями:

1) знайомство з каналами обміну повідомленнями;

2) канал "точка-точка;

3) публікація-опис каналу;

4) тип даних каналу;

5) недійсний канал повідомлення;

6) канал мертвих повідомлень;

7) гарантована доставка повідомлення;

8) адаптер каналу;

9) міст обміну повідомленнями;

10) шина повідомлень.

Структура повідомлень:

1) вступ до побудови повідомлень;

2) командне повідомлення;

3) повідомлення документа;

4) повідомлення про подію;

5) запит-відповідь;

6) зворотня адреса;

7) ідентифікатор кореляції;

8) послідовність повідомлень;

9) термін дії повідомлення;

10) індикатор форматування.

Управління системою:

1) вступ до управління системою;

2) шина управління;

3) обхід;

4) точка з'єднання;

5) історія повідомлень;

6) магазин зберігання повідомлень;

7) розумний проксі;

8) тестове повідомлення;

9) чистка каналів.

Розглянемо основні з них, та зупинимось на них більш детально.

1.4 Використання повідомлень

Тепер, коли ми знаємо, що таке обмін повідомленнями, слід запитати: навіщо використовувати обмін повідомленнями? Як і будь-яке витончене рішення, немає однієї простої відповіді. Швидка відповідь полягає в тому, що повідомлень більше безпосередньо, ніж передача файлів, краще інкапсульований, ніж спільна база даних, і надійніший за віддалене запрошення до процедури. Однак це лише початок переваг, які можуть бути отриманий за допомогою обміну повідомленнями.

Конкретні переваги обміну повідомленнями включають.

Віддалене спілкування. Повідомлення дозволяє окремим програмам спілкуватися та здійснювати передачу даних. Два об'єкти, які перебувають в одному процесі, можуть просто обмінюватися одними і тими ж даними в пам'ять. Надсилання даних на інший комп'ютер набагато складніше і вимагає дані для копіювання з одного комп'ютера на інший. Це означає, що об’єкти повинні бути перетворені в простий потік байтів, який може бути надісланий поперек мережа. Якщо віддалений зв’язок не потрібен, повідомлення не потрібні.

Інтеграція платформи або мови. При підключенні декількох комп'ютерних систем через віддалений спілкування, ці системи, ймовірно, використовують різні мови, технології та платформи, можливо тому, що вони були розвинені з часом незалежними командами. Інтеграція таких розбіжні програми можуть вимагати демілітаризованої зони проміжного програмного забезпечення для проведення переговорів між додатками, часто використовують найменший загальний знаменник - наприклад, плоскі дані файли з незрозумілими форматами. У цих умовах система обміну повідомленнями може бути універсальною перекладач між програмами, які працюють з мовою та платформою кожного з них власні умови, але дозволяє їм спілкуватися через загальний обмін повідомленнями парадигма. Цей універсальний зв’язок є серцем схеми шини повідомлень [6].

Асинхронна комунікація. Повідомлення дозволяє надіслати та забути підхід спілкування. Відправник не повинен чекати, коли одержувач прийме та обробить повідомлення; навіть не потрібно чекати, коли система обміну повідомленнями доставить повідомлення. Відправника потрібно лише чекати надсилання повідомлення, наприклад, для повідомлення до успішно зберігаються в каналі системою обміну повідомленнями. Як тільки повідомлення є після цього відправник може виконувати інші роботи під час передачі повідомлення фон. Одержувач може захотіти надіслати підтвердження або результат назад до відправника, якому потрібне інше

повідомлення, доставку якого потрібно буде виявити механізм зворотного виклику відправника.

Змінна хронологія. При синхронному зв’язку абонент повинен дочекатися отримання приймача закінчити обробку дзвінка, перш ніж абонент може отримати результат і продовжити. Таким чином, абонент може телефонувати лише настільки швидко, наскільки приймач може їх виконувати. З іншого боку, асинхронний зв'язок дозволяє відправнику партії запитів до одержувача на його власний темп, а одержувач споживає запити у своєму різному темпі. Це дозволяє обом програмам працювати з максимальною пропускною здатністю і не витрачати час на очікування один одного (принаймні, поки у приймача не закінчиться повідомлення для обробки) [6].

Дросельний. Проблема з викликами віддалених процедур полягає в тому, що їх занадто багато на одному приймачі одночасно може перевантажувати приймач. Це може спричинити продуктивність деградація і навіть призведе до виходу з ладу приймача. Асинхронний зв'язок дозволяє приймач контролює швидкість, з якою він споживає запити, щоб не стати перевантажений занадто великою кількістю одночасних запитів. Несприятливий вплив на абонентів, викликаних це дроселювання зведено до мінімуму, оскільки зв'язок асинхронний, тому дзвонять не заблоковані в очікуванні на приймачі.

Надійне спілкування. Повідомлення забезпечує надійну доставку, що вимагає виклику віддаленої процедури (ВВП) не може. Причина повідомлень надійніша за ВВП - це те, що для обміну повідомленнями використовується зберігати та направляти підхід до передачі повідомлень. Дані пакуються у вигляді повідомлень, які є атомними, незалежними одиницями. Коли відправник надсилає повідомлення, обмін повідомленнями система зберігає повідомлення. Потім воно доставляє повідомлення, пересилаючи його на комп’ютер приймача, де воно зберігається знову. Зберігання повідомлення на комп’ютері відправника та комп’ютер приймача вважається надійним. (Щоб

зробити його ще більш надійним, повідомлення можуть зберігатися на диску замість пам'яті. Ненадійно пересилає (переміщує) повідомлення з комп’ютера відправника на комп’ютер приймача, оскільки приймач або мережа можуть не працювати належним чином. Система обміну повідомленнями долає це шляхом повторного надсилання повідомлення до його успіху. Це автоматична спроба дозволяє системі обміну повідомленнями подолати проблеми з мережею таким чином, що відправник і одержувач не повинні турбуватися про ці деталі.

Відключена робота. Деякі програми спеціально розроблені для запуску відключеного від мережі, але для синхронізації з серверами, коли доступне мережеве з'єднання. Такі програми розміщені на платформах, таких як портативні комп'ютери, КПК та автомобільні приладові панелі. Повідомлення ідеально підходить для можливості синхронізації цих додатків - дані для синхронізації можна ставити в чергу, як вони створюються, чекаючи, поки програма знову підключиться до мережі.

Посередництво. Система обміну повідомленнями діє як посередник - як у шаблоні Посередника – між усіма програмами, які можуть надсилати та приймати повідомлення. Додаток може використовувати його як каталог інших програм або служб, доступних для інтеграції. Якщо програма відключається від інших, їй потрібно лише підключитися до системи обміну повідомленнями, а не до всіх інших програм обміну повідомленнями. Система обміну повідомленнями може використовуватися для забезпечення великої кількості розподілених з'єднань до спільного ресурсу, наприклад, до бази даних. Система обміну повідомленнями може використовувати надлишкові ресурси для забезпечення високої доступності, балансування навантаження, перенаправлення навколо невдалих мережевих з'єднань, налаштування продуктивності та якості обслуговування.

Управління лініями зв'язку. Асинхронний зв'язок означає, що одна програма не робить доведеться заблокувати, чекаючи, коли інша програма виконує завдання, якщо цього не хоче. Замість того, щоб блокувати

очікування відповіді, абонент може використовувати зворотний дзвінок, який сповістить абонента, коли відповідь надійде. Велика кількість заблокованих потоків або потоків, заблокованих протягом тривалого часу, може бути проблематичною. Занадто багато заблокованих потоків може залишити програму занадто мало доступних потоків для виконання справжньої роботи. Якщо програма з деякою динамічною кількістю заблокованих потоків виходить з ладу, коли програма перезапуститься та відновить колишній стан, відновити ці потоки буде складно. Що стосується зворотних дзвінків, єдиними потоками, які блокують, є невелика, відома кількість слухачів, які чекають відповідей. Це залишає більшість потоків, доступних для інших робіт, і визначає відому кількість потоків слухачів, які можна легко відновити після збоїв [7].

Отже, існує декілька різних причин, яким програма чи підприємство можуть отримати вигоду від обміну повідомленнями. Деякі з них - це технічні деталі, до яких розробники додатків найбільше ставляться, тоді як інші - це стратегічні рішення, які найкраще співпадають з архітекторами підприємств. Яка з цих причин є найбільш важливою, залежить від сучасних вимог ваших конкретних заявок.

Всі вони є вагомими причинами для обміну повідомленнями, тому скористайтеся тими, які причини принесуть вам найбільшу користь.

Сьогодні у вік інформаційних технологій додатки не можуть існувати самостійно, і вони мусять взаємодіяти з іншими додатками, створюючи екосистему. Тому актуальність роботи полягає в тому, що для інтеграції що є доволі типовою. Інтеграція без проблем здійснюється за допомогою типових рішень (шаблонів) і дозволяє інтегрувати будь-яку систему в екосистеми.

2 ШАБЛОНИ ОБМІНУ ПОВІДОМЛЕННЯМИ

2.1 Інтеграція додатків

Інтеграція підприємства – це завдання змусити окремі програми працювати разом, щоб створити єдиний набір функціональних можливостей. Деякі додатки можуть бути розроблені на замовлення, а інші підтримуватись сторонніми розробниками. Програми, ймовірно, працюють на декількох комп'ютерах, які можуть представляти кілька платформ, і можуть бути географічно розповсюджені. Деякі програми можуть запускатися за межами підприємства бізнес-партнерами або замовниками. Деякі програми можуть знадобитися інтегрувати, хоча вони не були розроблені для інтеграції і не можуть бути змінені. Ці проблеми та інші подібні до них ускладнюють інтеграцію додатків [8].

2.2.1 Критерії інтеграції додатків

Якби потреби в інтеграції були завжди однаковими, був би лише один стиль інтеграції. Однак, як і будь-яке складне технологічне зусилля, інтеграція додатків передбачає цілий ряд міркувань та наслідків, які слід враховувати при будь-якій можливості інтеграції.

Перший критерій – сама інтеграція додатків. Якщо ви можете розробити єдину автономну програму, яка не потребує співпраці з будь-якими іншими програмами, ви можете повністю уникнути проблеми інтеграції. Однак реально, навіть у простого підприємства є кілька додатків, додатків, які повинні працювати разом, щоб забезпечити єдиний досвід для працівників, партнерів та клієнтів підприємства.

Іншими основними критеріями рішення є:

1) з'єднання програм – навіть інтегровані програми повинні мінімізувати залежність один від одного, щоб кожен міг розвиватися, не створюючи проблем для інших. Щільно поєднані програми роблять численні припущення щодо роботи інших додатків; коли програми змінюють і порушують ці припущення, інтеграція порушується. Інтерфейс для інтеграції програм повинен бути достатньо конкретним, щоб реалізувати корисну функціональність, але достатньо загальним, щоб дозволити зміні цієї реалізації за потреби;

2) простота інтеграції – інтегруючи додаток у підприємство, розробники повинні прагнути мінімізувати зміну програми та мінімізувати кількість необхідного коду інтеграції. Однак зміни та новий код, як правило, знадобляться для забезпечення хорошої функціональної інтеграції, а підходи, що мають найменший вплив на додаток, можуть не забезпечити найкращої інтеграції в підприємство;

3) інтеграційна технологія – різні методи інтеграції вимагають різної кількості спеціалізованого програмного та апаратного забезпечення. Ці спеціальні інструменти можуть бути дорогими, можуть призвести до блокування постачальника та збільшити навантаження на розробників, щоб зрозуміти, як використовувати інструменти для інтеграції програм;

4) формат даних – інтегровані програми повинні узгоджувати формат даних, якими вони обмінюються, або повинні мати проміжний перекладач для уніфікації програм, які наполягають на різних форматах даних. Пов'язана проблема – це еволюція та розширення формату даних – як формат може змінюватися з часом і як це вплине на програми;

5) своєчасність даних – інтеграція повинна мінімізувати тривалість часу, коли одна програма вирішує поділитися деякими даними, а інші програми мають ці дані. Дані слід часто передавати невеликими об 'ємами, а не чекати обміну великим набором непов'язаних пакетів повідомлень. Про додатки слід повідомити, як тільки спільні дані будуть готові до споживання. Затримка обміну даними повинна враховуватися в інтеграційному дизайні;

чим довший обмін може зайняти, тим більше можливостей для спільних даних старі, і тим складнішою стає інтеграція.

6) дані або функціональність – інтегровані програми можуть не просто обмінюватися даними, вони можуть поділитися функціональністю таким чином, що кожна програма може викликати функціональність у інших.

Існує кілька різних критеріїв, які необхідно враховувати при виборі та розробці інтеграційного підходу [9].

2.2.2 Параметри інтеграції додатків

Існує не один підхід до інтеграції програм. Кожен підхід відповідає деяким критеріям інтеграції краще, ніж інші. Різні підходи можна підсумувати в чотирьох основних стилях інтеграції:

- передача файлів – дозволяйте кожній програмі створювати файли спільних даних, щоб інші могли їх споживати, і споживати файли, які інші створили;
- спільна база даних – дозволяє програмам зберігати дані, якими вони хочуть поділитися, у загальній базі даних;
- виклик віддалених процедур. Попросіть кожну програму розкрити деякі свої процедури, щоб їх можна було віддалено викликати, а програми застосували їх для запуску поведінки та обміну даними;
- повідомлення - кожен додаток підключається до загальної системи обміну повідомленнями та обмінюється даними та викликає поведінку за допомогою повідомлень.

У вступі до стилів інтеграції ми обговорили різні варіанти з'єднання програм між собою, включаючи повідомлення. Повідомлення дозволяє додаткам вільно поєднуватися шляхом асинхронного спілкування, що також робить зв’язок більш надійним, оскільки два додатки не повинні працювати одночасно. Повідомлення робить систему обміну повідомленнями відповідальною за передачу даних з однієї програми в іншу, тому програми

можуть зосередитись на тому, якими даними вони повинні ділитися, але не так сильно хвилюватися, як ними поділитися.

2.3 Система обміну повідомленнями

Більшість технологій, обмін повідомленнями включає певні основні поняття. Існують такі основні концепції обміну повідомленнями:

- канали – програми обміну повідомленнями передають дані через канал повідомлень, віртуальний міст, який з'єднує відправника з приймачем. Щойно встановлена система обміну повідомленнями не містить жодних каналів; ви повинні визначити, яким чином ваші програми потребують зв’язку, а потім створити канали для його полегшення;
- повідомлення – це пакет даних, який можна передавати по каналу. Таким чином, для передачі даних програма повинна розбити дані на один або кілька пакетів, обернути кожен пакет як повідомлення, а потім надіслати повідомлення по каналу. Так само програма-одержувач отримує повідомлення і повинна витягувати дані з повідомлення для його обробки. Система повідомлень намагатиметься неодноразово доставляти повідомлення (наприклад, передавати його від відправника до одержувача), поки воно не стане успішним;
- багатоетапна доставка – у найпростішому випадку система повідомлень доставляє повідомлення безпосередньо з комп'ютера відправника на комп’ютер приймача. Однак часто потрібно виконувати дії над повідомленням після того, як воно надсилається його первинним відправником, але до того, як воно буде отримане його кінцевим одержувачем. Наприклад, повідомлення, можливо, доведеться перевірити або трансформувати, оскільки одержувач очікує іншого формату повідомлення, ніж відправник. Архітектура труб та фільтрів описує, як кілька кроків обробки можна пов’язати разом за допомогою каналів;

- маршрутизація – у великому підприємстві з численними програмами та каналами для їх підключення може з’явитися повідомлення, яке пройде через кілька каналів, щоб досягти остаточного пункту призначення. Маршрут, який повинен дотримуватися повідомлення, може бути настільки складним, що початковий відправник не знає, який канал отримає повідомлення кінцевому одержувачу. Натомість оригінальний відправник відправляє повідомлення на Маршрутизатор повідомлень, компонент програми та фільтр в архітектурі труб і фільтрів, який визначатиме, як орієнтуватися в топології каналу та спрямовувати повідомлення до кінцевого приймача або принаймні до наступний маршрутизатор [10];
- трансформація – різні програми можуть не погоджуватися щодо формату одних і тих же концептуальних даних; відправник форматує повідомлення в один бік, але одержувач очікує, що воно буде відформатоване іншим способом. Щоб узгодити це, повідомлення повинно пройти через проміжний фільтр, Перекладач повідомлень, який перетворює повідомлення з одного формату в інший;
- кінцеві точки – додаток не має вбудованої можливості взаємодіяти з системою обміну повідомленнями. Швидше, він повинен містити частину коду, який знає, як працює програма, і як працює система обміну повідомленнями, з'єднуючи їх між собою, щоб вони працювали разом. Цей міст – це набір узгоджених кінцевих точок повідомлення, які дозволяють програмі надсилати та отримувати повідомлення.

Нижче, на рисунку 2.1 зображено схему взаємозв'язку кореневих шаблонів та глав:

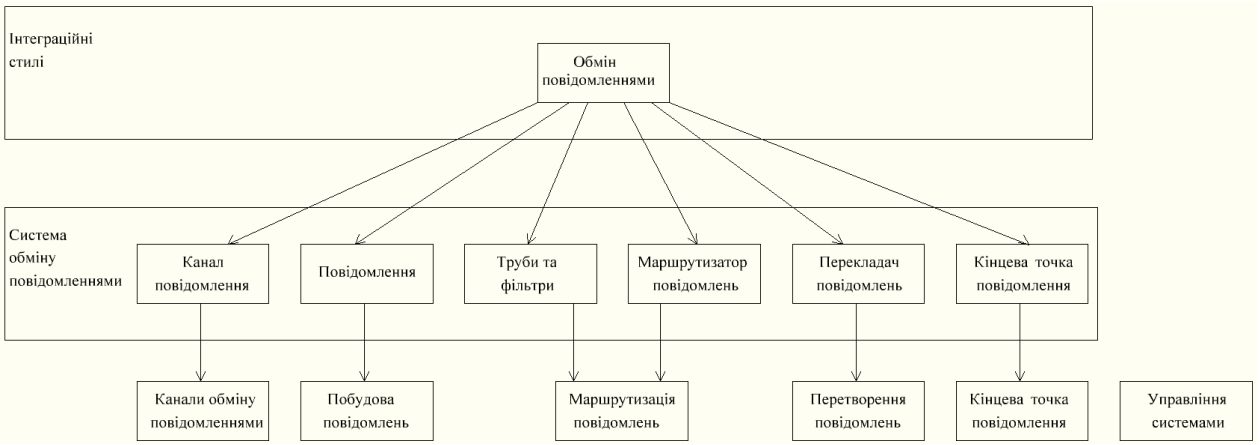

Рисунок 2.1 – Схема взаємодії кореневих шаблонів

2.4 Канали обміну повідомленнями

Коли дві програми хочуть обмінюватися даними, вони роблять це шляхом надсилання даних через канал, який з'єднує їх. Програма, що надсилає дані, може не знати, яка програма отримає ці дані, але, вибравши певний канал для надсилання даних, відправник знає, що одержувач буде тим, хто шукає такий тип даних, шукаючи їх на цьому канал. Таким чином, програми, які виробляють спільні дані, мають спосіб спілкування з тими, хто бажає їх приймати.

Перший тип це – канал "точка-точка" забезпечує те, що лише один приймач споживає будь-яке повідомлення. Якщо канал має кілька приймачів, лише один з них може успішно споживати певне повідомлення. Якщо декілька приймачів намагаються споживати одне повідомлення, канал забезпечує успіх лише одного з них, тому приймачі не повинні узгоджувати один з одним. На каналі все ще може бути декілька приймачів, щоб одночасно споживати декілька повідомлень, але лише один приймач споживає одне повідомлення. Схема каналу "точка-точка" зображена на рисунку 2.2:

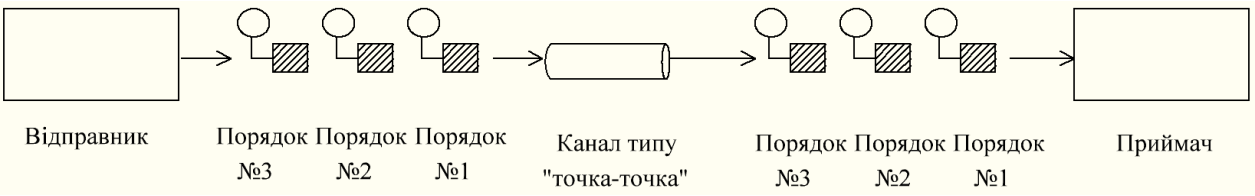

Рисунок 2.2 – Схема каналу типу "точка-точка"

Маршрутизатор працює так: у нього є один вхідний канал, який розбивається на кілька вихідних каналів, по одному для кожного споживача, зображена на рисунку 2.3. Коли інформація доходить до маршрутизатора, то копія повідомлення доставляється кожному з вихідних каналів. Кожен вихідний канал має лише одного абонента, якому дозволено споживати повідомлення лише один раз. Таким чином, кожен абонент отримує повідомлення лише один раз, і спожиті копії зникають зі своїх каналів.

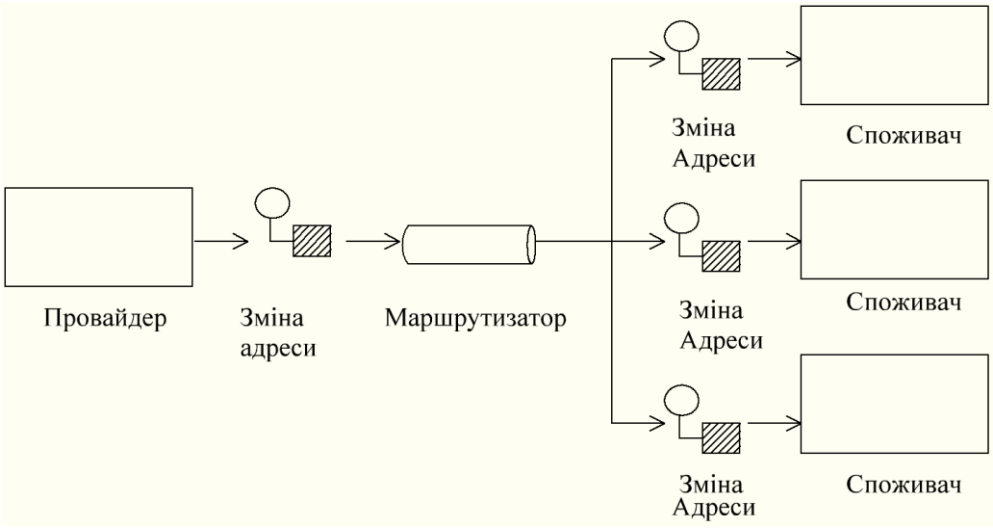

Рисунок 2.3 – Схема маршрутизатора

Канал типу даних, використовуючи окремий канал даних типу для кожного типу даних, усі повідомлення на даному каналі будуть містити один і той же тип даних, зображений на рисунку 2.4. Відправник, знаючи, який тип даних, повинен буде вибрати відповідний канал для надсилання. Одержувач, знаючи, на який канал були отримані дані, буде знати, що це його тип.

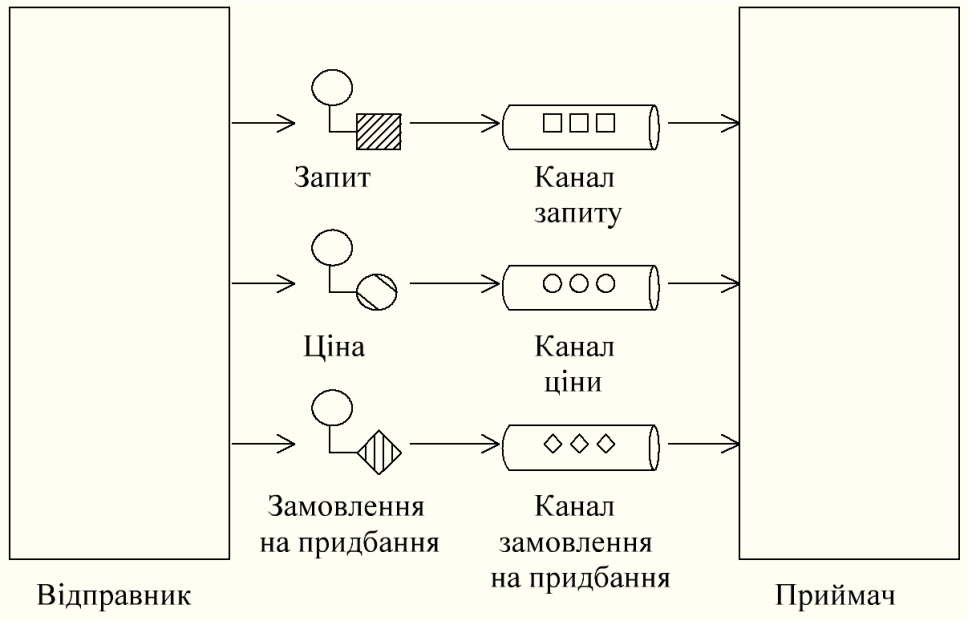

Рисунок 2.4 – Схема каналу типу даних

Невалідний канал повідомлень. Одержувач повинен перенести неправильне повідомлення до невалідного каналу повідомлень, спеціального каналу для повідомлень, які не могли обробити їх приймачі. Розробляючи систему обміну повідомленнями для програм, які використовуватимуться, адміністратору потрібно буде визначити один або кілька недійсних каналів повідомлень для програм, які використовуються. Схему невалідного каналу повідомлення зображено на рисунку 2.5:

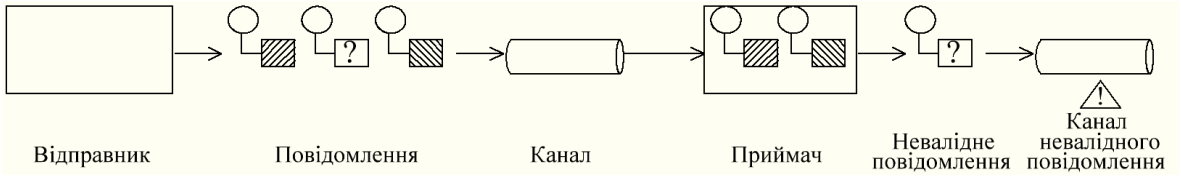

Рисунок 2.5 – Схема невалідного каналу повідомлень

Коли система обміну повідомленнями визначає, що не може або не повинна доставляти повідомлення, вона може вибрати, щоб перенести повідомлення на канал мертвих повідомлень, зображений ні рисунку 2.6 [3]:

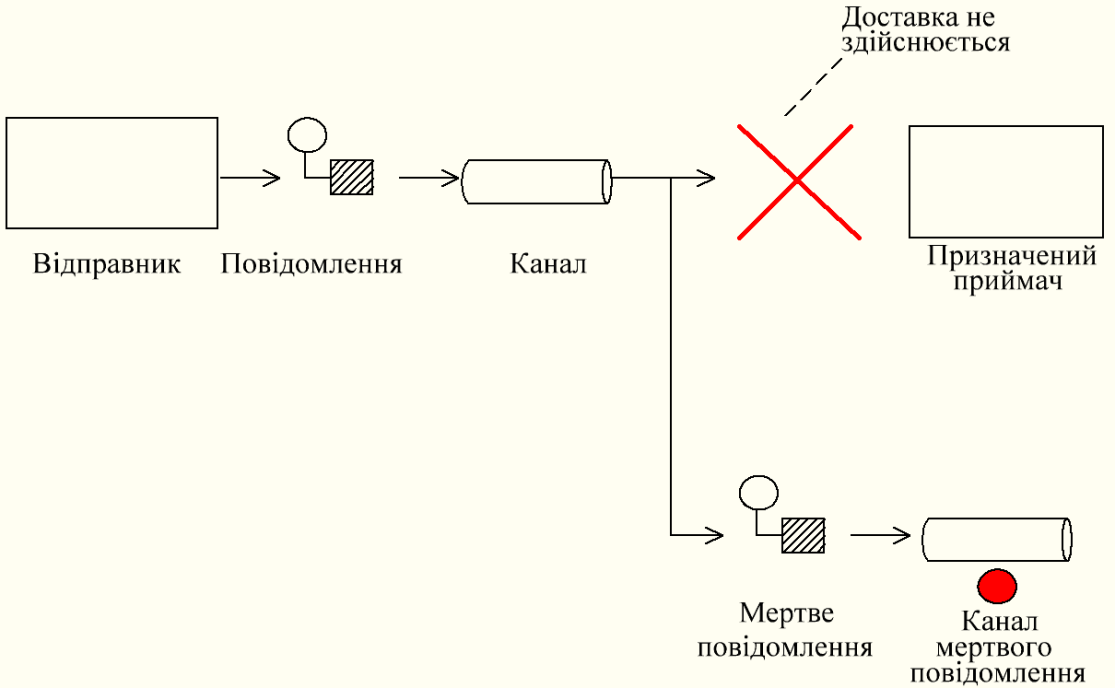

Рисунок 2.6 – Схема мертвого каналу повідомлень

Гарантована доставка повідомлення використовується, щоб повідомлення стали стійкими, щоб вони не були втрачені, навіть якщо система обміну повідомленнями зламалась. При гарантованій доставці система обміну повідомленнями використовує вбудований сховище даних для збереження повідомлень. Кожен комп'ютер, на якому встановлена система обміну повідомленнями, має власний сховище даних, щоб повідомлення могли зберігатися локально. Коли відправник надсилає повідомлення, операція надсилання не завершується успішно, поки повідомлення надійно не зберігається у сховищі даних відправника. Згодом повідомлення не видаляється з одного сховища даних, поки воно не буде успішно переслане та збережено у наступному сховищі даних. Таким чином, щойно відправник успішно надсилає повідомлення, воно завжди зберігається на диску щонайменше на одному комп’ютері, поки він не буде успішно доставлений і не підтверджений одержувачем. Схема гарантованої доставки зображена на рисунку 2.7

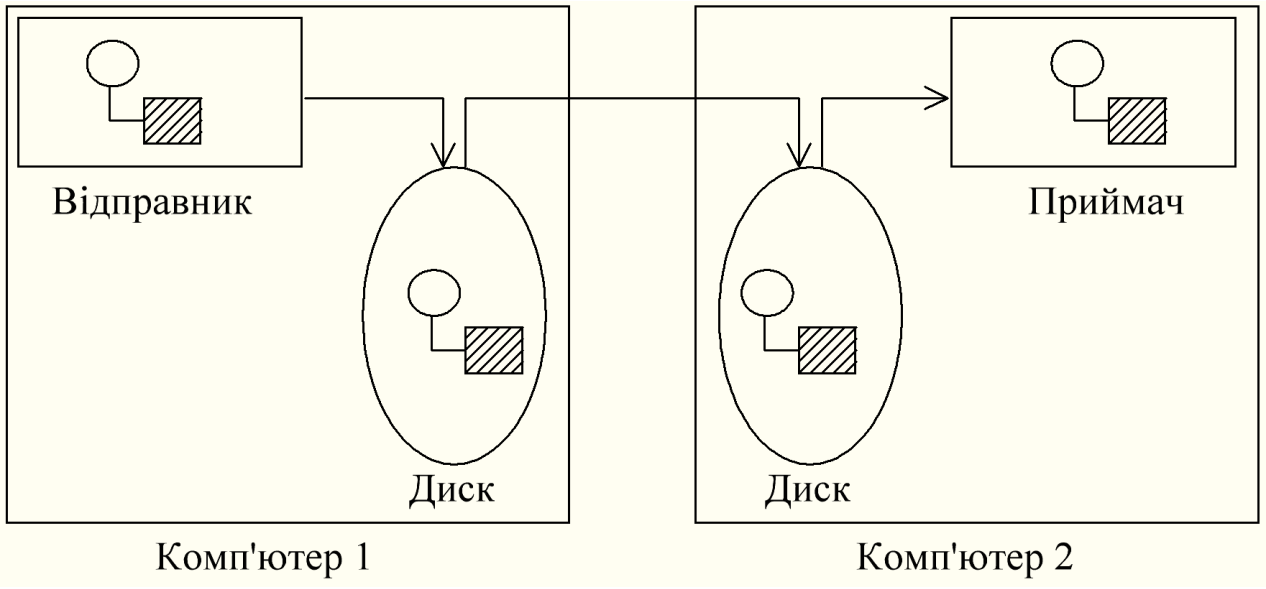

Рисунок 2.7 – Схема гарантованої доставки

Адаптер каналу використовується, якщо може отримати доступ до API програми або даних програми та публікувати повідомлення на каналі на основі цих даних, а також може отримувати повідомлення та викликати функціональні можливості всередині програми. Адаптер діє як клієнт обміну повідомленнями в системі обміну повідомленнями і викликає функції програм через інтерфейс, що постачається додатком. Таким чином, будь-яка програма може підключитися до системи обміну повідомленнями та інтегруватися з іншими програмами, якщо у неї є належний адаптер каналу, схема якого зображена на рисунку 2.8.

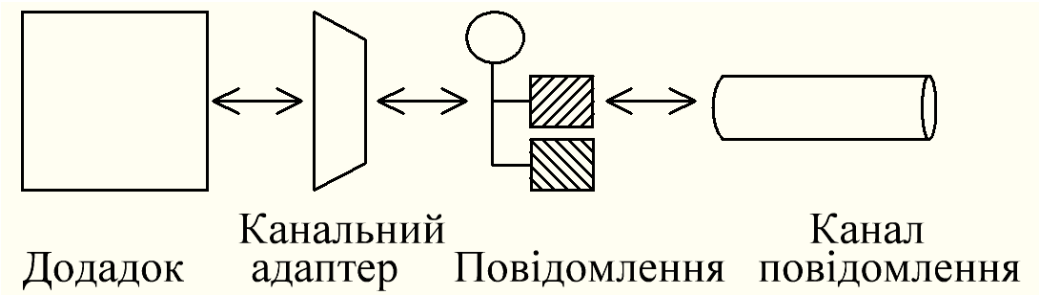

Рисунок 2.8 – Схема адаптеру каналу

Міст обміну повідомленнями використовуються для з'єднання між системами обміну повідомленнями для копіювання повідомлень між

системами. Зазвичай не існує практичного способу з'єднання двох повноцінних систем обміну повідомленнями, тому натомість ми з'єднуємо окремі відповідні канали між системами обміну повідомленнями. Міст обміну повідомленнями – це набір адаптерів каналів, де клієнт, що не повідомляє повідомлення, є фактично іншою системою обміну повідомленнями, і де кожна пара адаптерів з'єднує пару відповідних каналів. Міст, зображений на рисунку 2.9 виступає як карта від одного набору каналів до іншого, а також перетворює формат повідомлення однієї системи в іншу. Підключені канали можуть використовуватися для передачі повідомлень між традиційними клієнтами системи обміну повідомленнями або строго для повідомлень, призначених для інших систем обміну повідомленнями.

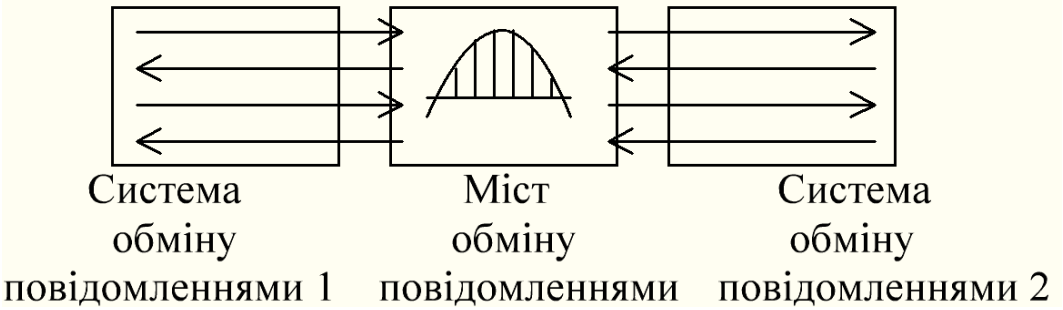

Рисунок 2.9 – Схема мосту обміну повідомленнями

Структура з’єднання проміжного слою між цими програмами як шина повідомлень, яка дозволяє їм працювати разом за допомогою повідомлень. Шина повідомлень – це комбінація загальної моделі даних, загального набору команд та інфраструктури обміну повідомленнями, що дозволяє різним системам спілкуватися через спільний набір інтерфейсів. Це аналогічно шині зв'язку в комп'ютерній системі, яка служить центром зв’язку між процесором, основною пам'яттю та периферійними пристроями. Так само, як і в апаратній аналогії, існує ряд фрагментів, які складаються разом для формування шини повідомлень, що зображена на рисунку 2.10.

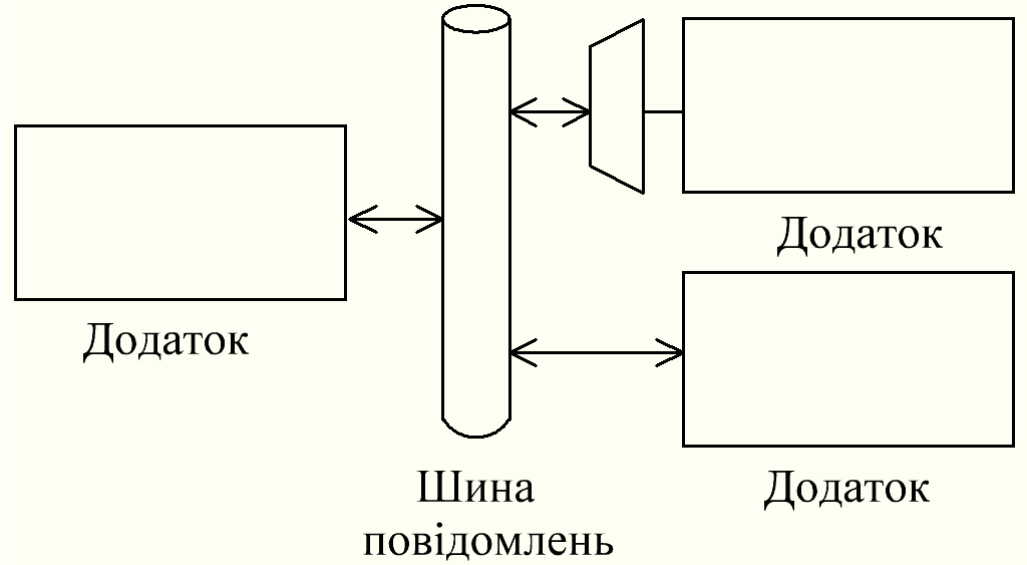

Рисунок 2.10 – Схема шини повідомлень

2.5 Структура повідомлень

Коли дві програми хочуть обмінятися частиною даних, вони роблять це, загортаючи його в повідомлення. У той час як канал повідомлень не може передавати необроблені дані самі по собі, він може передавати дані, загорнуті в повідомлення.

Використовуйте командне повідомлення, щоб надійно викликати процедуру в іншій програмі. Немає конкретного типу повідомлення для команд; командне повідомлення – це звичайне повідомлення, яке містить команду. У JMS командним повідомленням може бути будь-який тип повідомлення; приклади включають повідомлення об'єкта, що містить об'єкт команди, текст повідомлення, що містить команду у формі XML тощо. У .NET командне повідомлення - це повідомлення з збереженою в ньому командою. Запит простого протоколу доступу до об’єктів – це командне повідомлення.

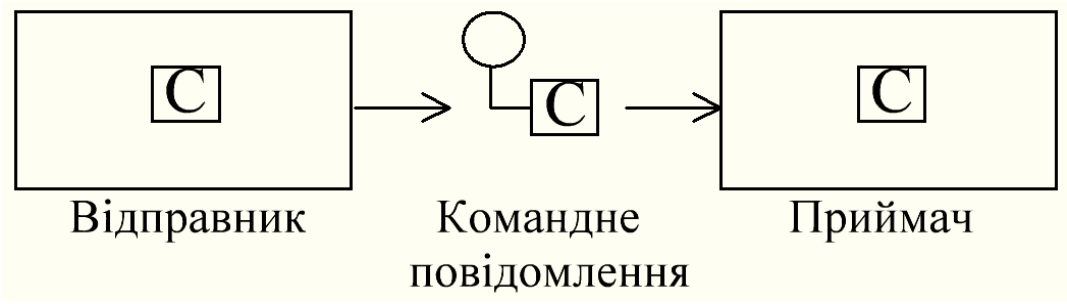

Рисунок 2.11 – Схема командного повідомлення

Повідомлення документа, схема якого зображено на рисунку 2.12, що використовується для надійної передачі структури даних між додатками. Поки командне повідомлення повідомляє одержувачу викликати певну поведінку, документ-повідомлення просто передає дані і дає можливість одержувачу вирішити, що, якщо що, робити з даними. Дані – це одна одиниця даних, окремий об'єкт або структура даних, яка може розкладатися на менші одиниці [3].

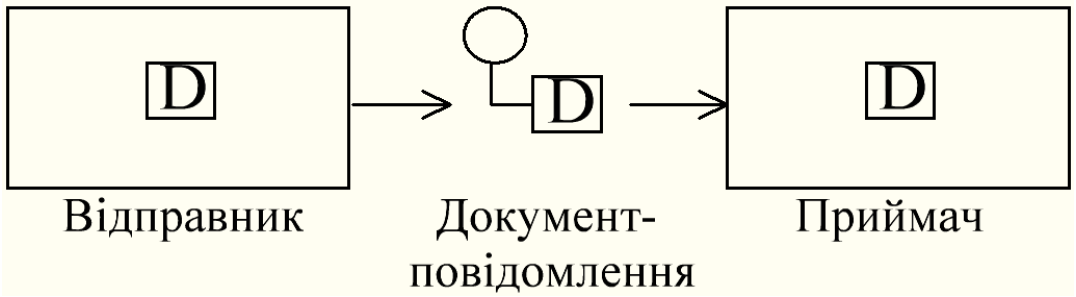

Рисунок 2.12 – Схема повідомлення документа

Повідомлення про подію, схема якого зображено на рисунку 2.13, що використовується для надійного, асинхронного сповіщення про події між додатками. Коли суб’єкт має оголосити подію, він створить об'єкт події, загорнуть його у повідомлення та надішле на канал. Спостерігач отримає повідомлення про подію, отримає подію та обробить її. Передача повідомлень не змінює сповіщення про подію, а лише гарантує, що сповіщення потрапить до спостерігача.

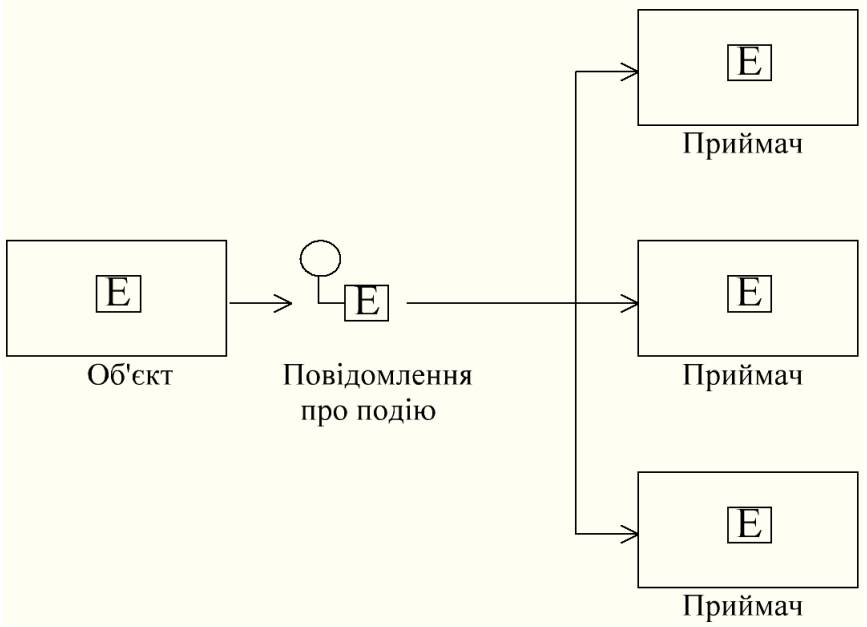

Рисунок 2.13 – Схема повідомлення про подію

Повідомлення запиту повинно містити зворотню адресу, яка вказує, куди надсилати відповідь. Таким чином, відповідачу не потрібно знати, куди надсилати відповідь, він може просто задати запит. Якщо різні повідомлення одному і тому ж відповідач вимагають відповіді в різні місця, відповідач знає, куди надсилати відповідь на кожен запит. Це інкапсулює знання про те, які канали використовувати для запитів та відповідей у запитувачі, щоб ці рішення не повинні бути чітко кодованими в відповідачі. У заголовку повідомлення міститься зворотна адреса, оскільки вона не є частиною даних, що передаються. Схема повідомлення документа відповідача зображена на рисунку 2.14:

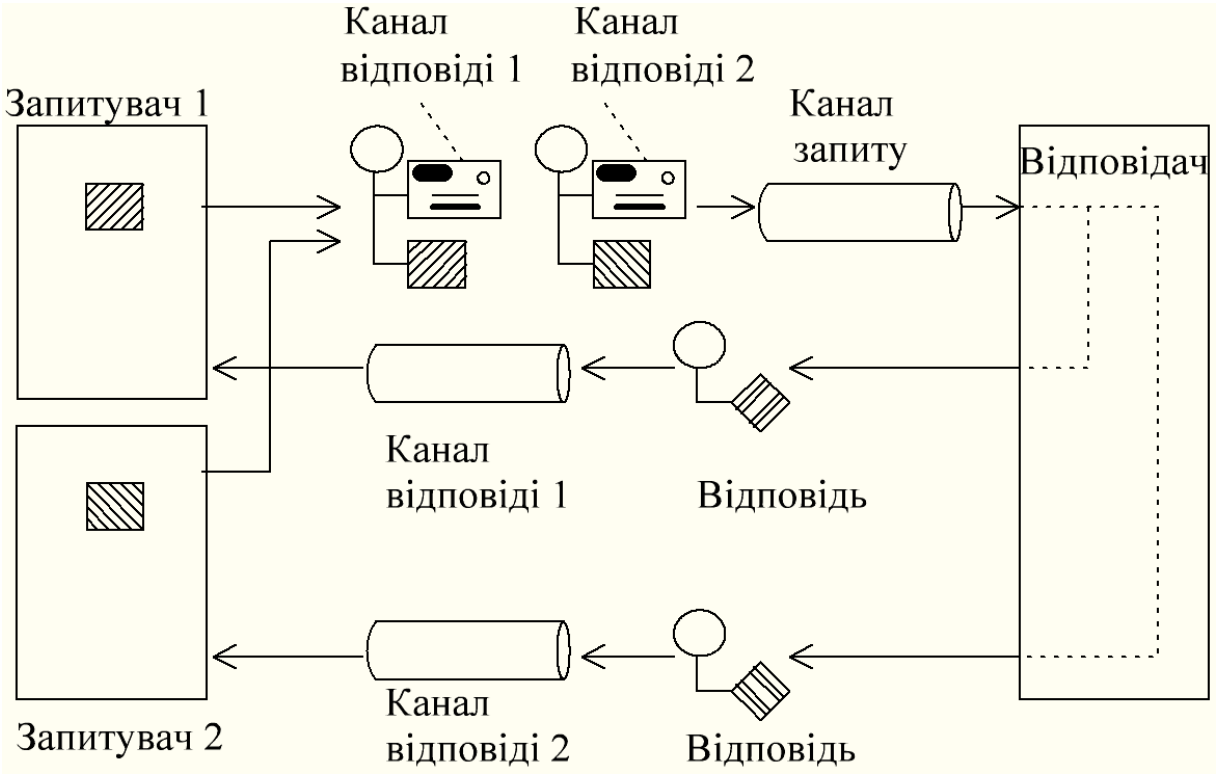

Рисунок 2.14 – Схема повідомлення документа відповідача

Термін дії повідомлення встановлюється, щоб вказати обмеження часу, на який термін діє повідомлення. Після того, як пройде час, протягом якого повідомлення є життєздатним, і повідомлення все ще не було спожито, повідомлення закінчується. Споживачі системи обміну повідомленнями проігнорують повідомлення про термін дії; вони ставляться до повідомлення так, ніби в першу чергу ніде не надсилаються. Більшість реалізацій системи обміну повідомленнями перенаправляють прострочені повідомлення на канал мертве повідомлення, в той час як інші просто відкидають прострочені повідомлення; це може бути налаштовано. Схема терміну дії повідомлення зображена на рисунку 2.15:

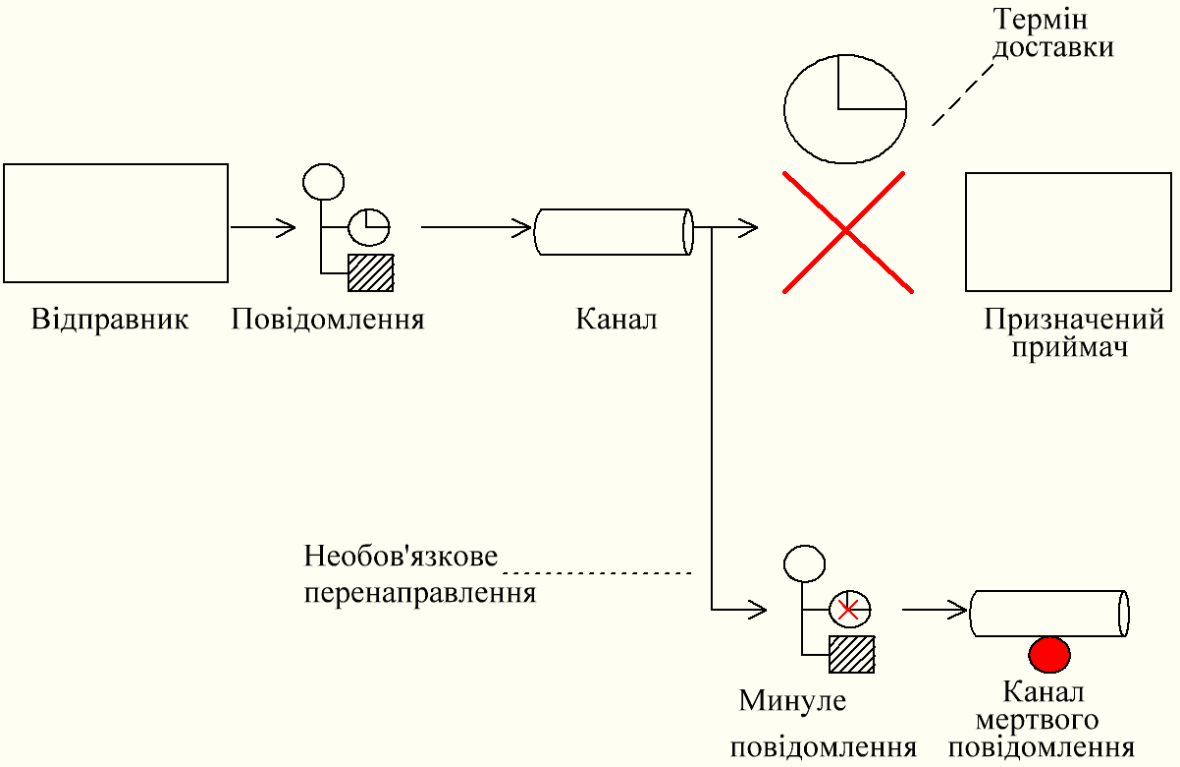

Рисунок 2.15 – Схема терміну дії повідомлення

2.6 Управління системою

Хоча розробка рішення для обміну повідомленнями не є легким завданням, оперувати таким рішенням у виробництві однаково складно: рішення інтеграції на основі повідомлень можуть виробляти, маршрутизувати та перетворювати тисячі чи навіть мільйони повідомлень за день. Ми маємо мати справу з винятками, вузькими місцями та змінами в системах-учасницях. Щоб зробити все більш складним, компоненти розподіляються на багатьох платформах і машинах, які можуть розміщуватися в декількох місцях.

Окрім властивих їм складностей та масштабів інтеграції розподілених пакунків та спеціальних програм, архітектурні переваги нещільного з’єднання фактично ускладнюють тестування та налагодження системи. Мартін Фаулер називає це симптомом "мрія архітектора, жах розробника":

архітектурні принципи нещільного зчеплення та непрямості зменшують припущення, що системи роблять одна щодо одної, і тому забезпечують гнучкість. Однак тестування системи, коли виробник повідомлень не знає, хто такі споживачі, може бути складним. Додайте до цього, що асинхронні та часові аспекти обміну повідомленнями та речі ще більше ускладнюються. Наприклад, рішення для обміну повідомленнями може навіть не бути розроблене для того, щоб виробник повідомлень отримував відповідь від одержувачів. Так само інфраструктура обміну повідомленнями зазвичай гарантує доставку повідомлення, але не час доставки. Це ускладнює розробку тестових випадків, які покладаються на результати доставки повідомлень.

Шина управління використовується для управління системою інтеграції підприємства. Контрольна шина використовує той самий механізм обміну повідомленнями, який використовується даними програми, але використовує окремі канали для передачі даних, що мають відношення до управління компонентами, задіяними в потоці повідомлень. Кожен компонент у системі тепер підключений до двох підсистем обміну повідомленнями. Схема шини управління зображено на рисунку 2.16.

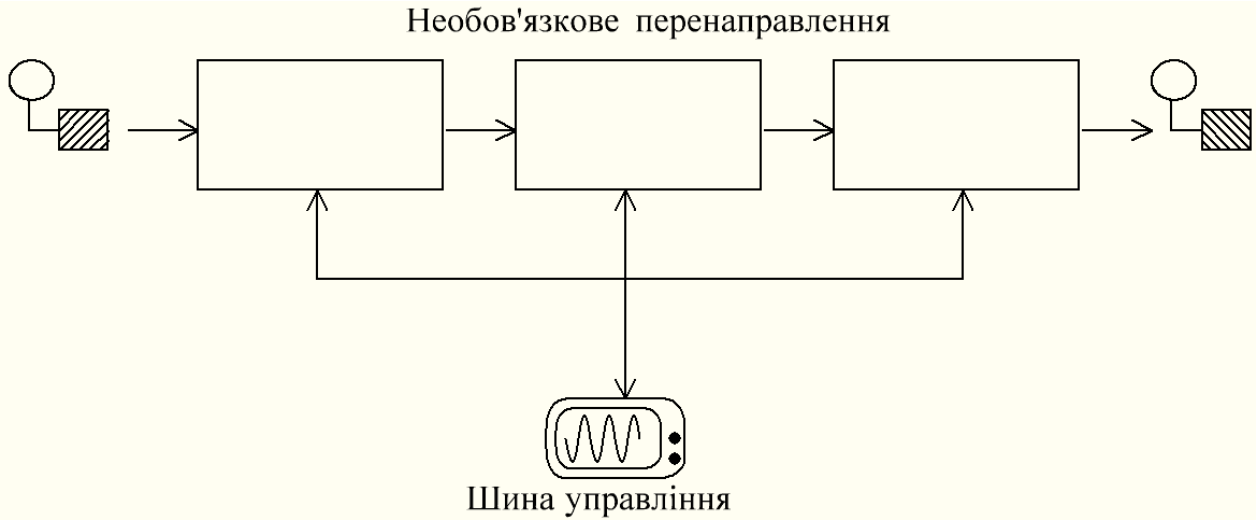

Рисунок 2.16 – Схема шини управління

Об'їзд за допомогою маршрутизатора на основі контексту, керованого через шину управління. В одному стані маршрутизатор спрямовує вхідні

повідомлення через додаткові кроки, а в іншому він спрямовує повідомлення безпосередньо до пункту призначення. Об'їзд використовує простий контекстний маршрутизатор з двома вихідними каналами. Один вихідний канал передає немодифіковане повідомлення до вихідного пункту призначення. Відповідно до розпорядження шини управління, об'їзд відправляє повідомлення на інший канал. Цей канал передає повідомлення додатковим компонентам, які можуть перевірити та, або змінити повідомлення. Зрештою, ці компоненти спрямовують повідомлення до того самого пункту призначення. Схема об'їзду за допомогою маршрутизатора зображена на рисунку 2.17 [1].

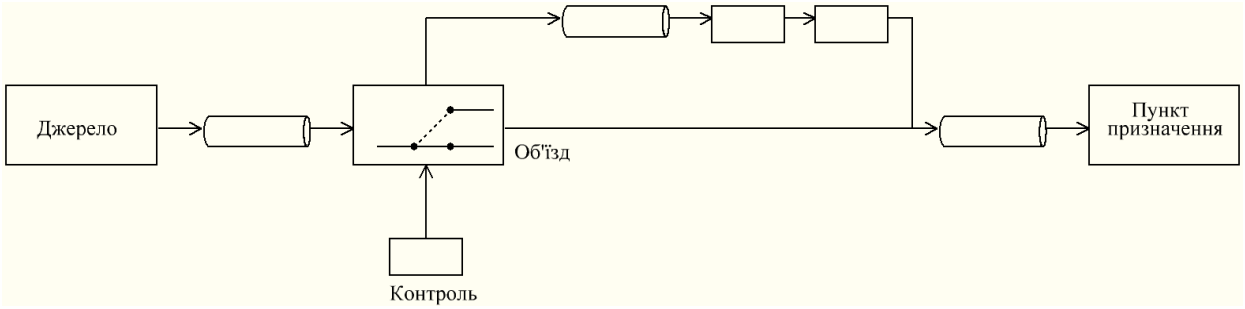

Рисунок 2.17 – Схема об'їзду за допомогою маршрутизатора

Точка з'єднання – це список одержувачів у канал, який публікує кожне вхідне повідомлення на головний і вторинний канал, фіксований список одержувачів з двома вихідними каналами. Він споживає повідомлення з вхідного каналу та публікує немодифіковане повідомлення обом вихідним каналам. Щоб вставити точку з'єднання в канал, вам потрібно створити додатковий канал і змінити цільовий приймач, щоб споживати другий канал. Оскільки логіка аналізу знаходиться всередині другого компонента, ми можемо вставити загальну точку з'єднання в будь-який канал без жодної небезпеки змінити поведінку основного каналу. Це покращує повторне використання та зменшує ризик підбору існуючого рішення. Схема об'їзду за допомогою маршрутизатора зображеного на рисунку 2.18.

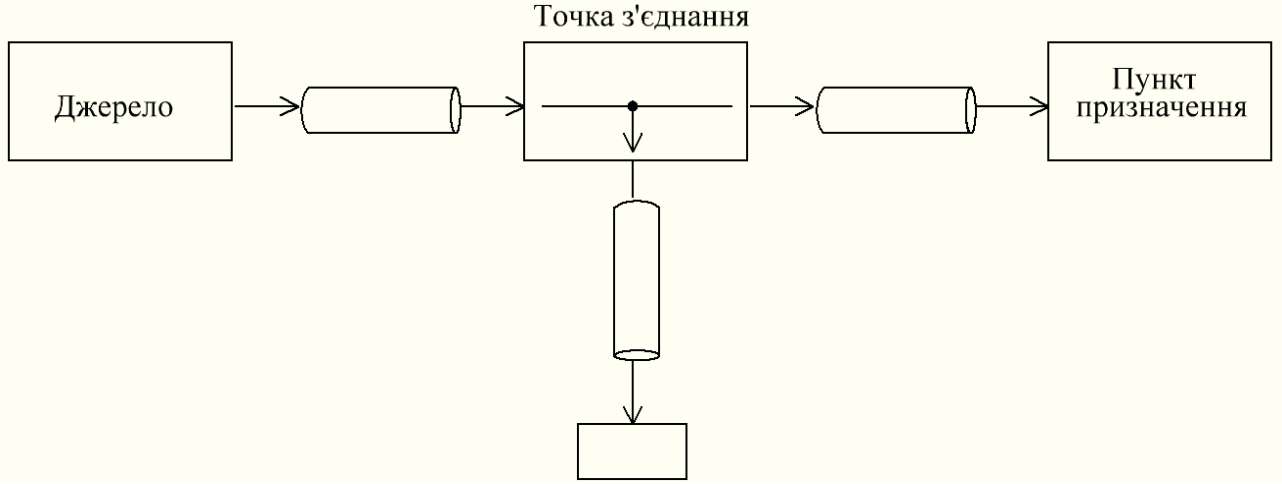

Рисунок 2.18 – Схема точки з’єднання

Історія повідомлень – це список усіх програм, через які повідомлення пройшло з моменту появи. Історія повідомлень містить список усіх компонентів, через які повідомлення пройшло. Кожен компонент, який обробляє повідомлення (включаючи ініціатора), додає до списку один запис. Історія повідомлень має бути частиною заголовка повідомлень, оскільки містить специфічну для управління інформацію. Зберігання цієї інформації у заголовку відокремлює її від тіла повідомлення, що містить конкретні дані програми. Схема історія повідомлень зображена на рисунку 2.19.

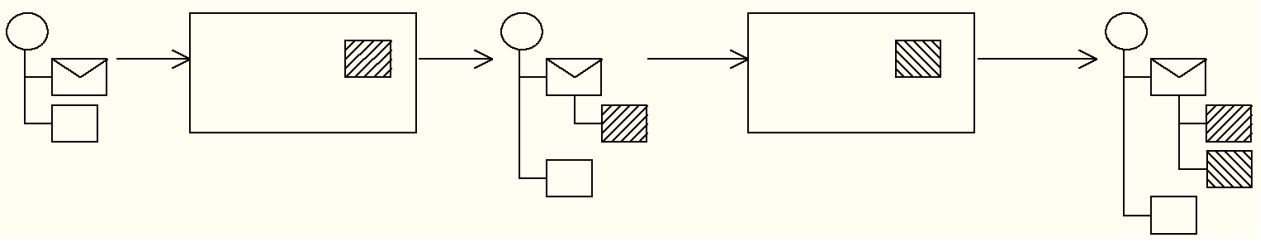

Рисунок 2.19 – Схема історії повідомлень

Магазин зберігання повідомлень, схема якого зображена на рисунку 2.20 використовується для збору інформації про кожне повідомлення в центральному місці. Використовуючи магазин зберігання повідомлень, ми можемо скористатися асинхронним характером інфраструктури обміну повідомленнями. Коли ми надсилаємо повідомлення каналу, ми надсилаємо

дублікат повідомлення на спеціальний канал, який збирає магазин повідомлень. Це може бути виконано самим компонентом або ми можемо вставити точка з'єднання в канал. Ми можемо розглядати вторинний канал, який несе копію повідомлення, як частину шини управління. Надсилання другого повідомлення в режимі "пожежа і забудь" не уповільнить потік основних повідомлень програми. Однак це збільшує мережевий трафік. Тому ми можемо не зберігати повне повідомлення, а лише декілька ключових полів, необхідних для подальшого аналізу, наприклад, ідентифікатор повідомлення або канал, по якому повідомлення було надіслане, та часова мітка [11].

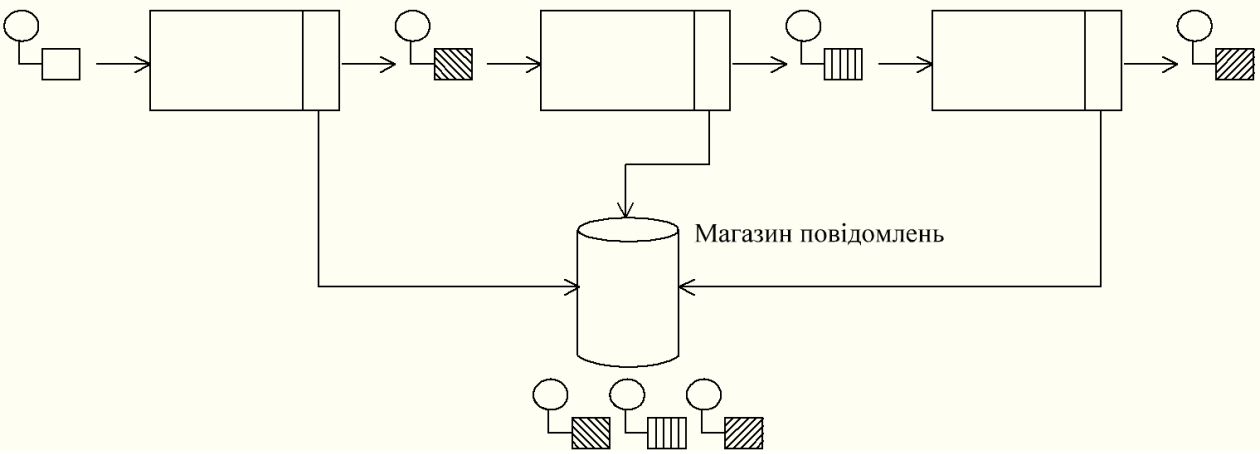

Рисунок 2.20 – Схема магазину повідомлень

Смарт-проксі, схема якого зображена на рисунку 2.21, що перехоплює повідомлення, надіслані на каналі запиту, на сервіс запит-відповідь. Для кожного вхідного повідомлення Смарт-проксі зберігає адресу повернення, вказану оригінальним відправником. Потім він замінює повернення адреси в повідомленні каналом відповіді, який слухає смарт-проксі. Коли на цьому каналі надходить відповідне повідомлення, смарт-проксі отримує збережену зворотну адресу та використовує маршрутизатор повідомлень для пересилання немодифікованої адреси відповіді на цей канал.

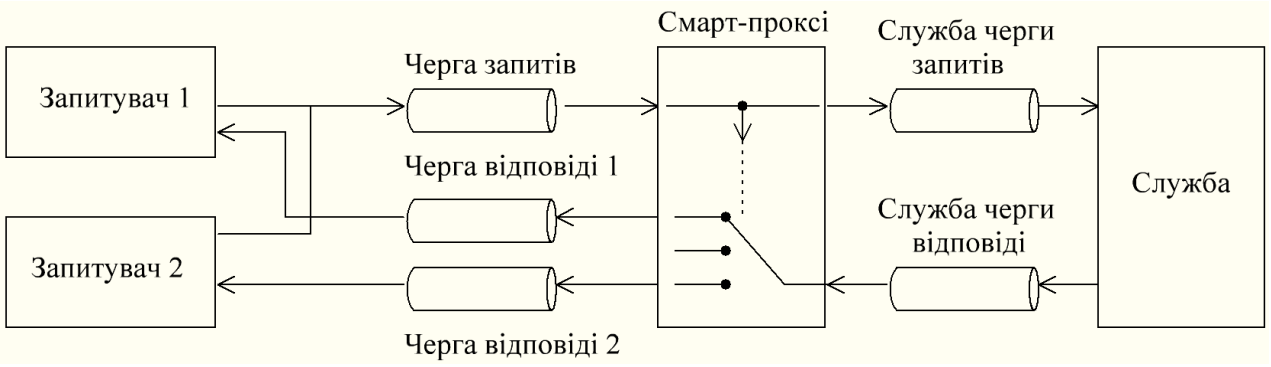

Рисунок 2.21 – Схема старт-проксі

Тестове повідомлення, схема якого зображена на рисунку 2.22 використовується щоб забезпечити здоров'я компонентів для обробки повідомлень. Шаблон тестового повідомлення спирається на такі компоненти: шина управління, зворотна адреса.

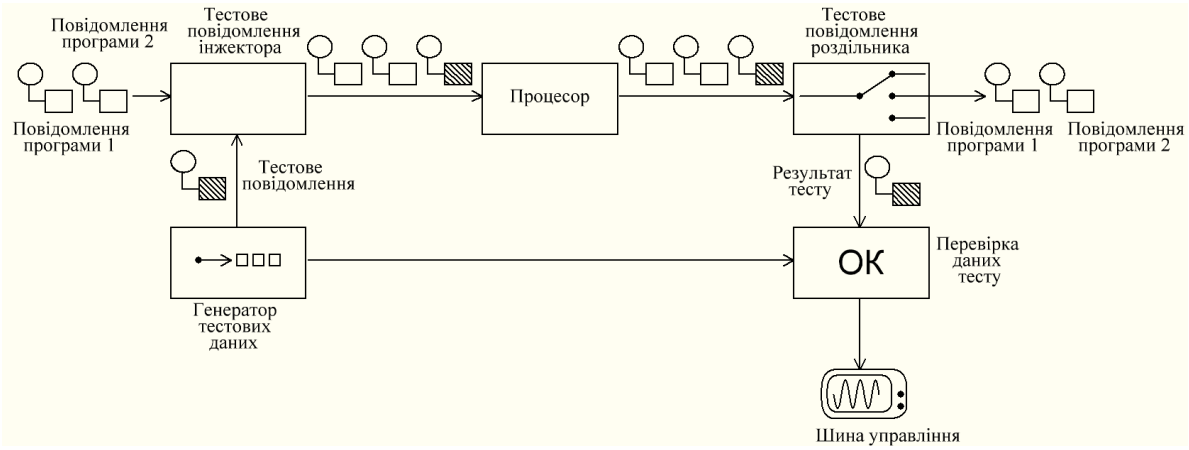

Рисунок 2.22 – Схема тестового повідомлення

Чистка каналів відповідає за видалення всіх повідомлень з каналу. Цього може бути достатньо для тестових сценаріїв, коли ми хочемо повернути систему в стійкий стан. Якщо ми налагоджуємо виробничу систему, нам може знадобитися видалити окреме повідомлення або набір повідомлень на основі конкретних критеріїв, таких як ідентифікатор повідомлення або значення певних полів повідомлень. Чистка каналів зображена на рисунку 2.23.

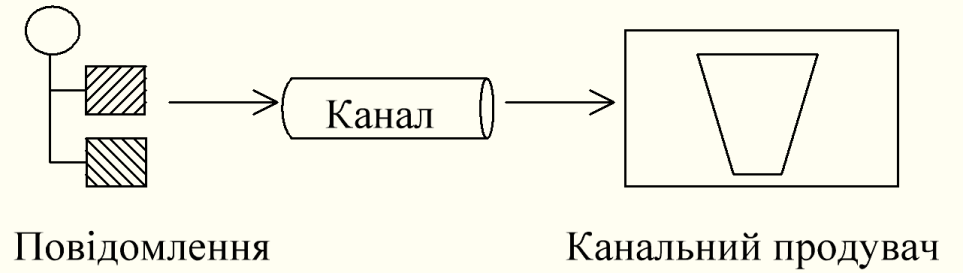

Рисунок 2.23 – Схема чистки каналів

2.7 Кінцеві точки повідомлень

У вступі до систем обміну повідомленнями ми обговорювали кінцеву точку повідомлення. Ось так додаток підключається до системи обміну повідомленнями, щоб вона могла надсилати та отримувати повідомлення. Як програміст програми, коли ви програмуєте на API обміну повідомленнями, наприклад, JMS Система. Відправлення ви розробляєте код кінцевої точки. Якщо ви використовуєте комерційний пакет проміжного програмного забезпечення, більшість цього коду вже зроблено для вас, використовуючи бібліотеки та інструменти, надані постачальником.

Деякі зразки кінцевих точок стосуються як відправників, так і приймачів. Вони стосуються того, як програма взагалі стосується системи обміну повідомленнями.

Інкапсуляція коду обміну повідомленнями – загалом, програма не повинна знати про те, що вона використовує пересилання для інтеграції з іншими програмами. Більшість додатків кодів потрібно писати, не маючи на увазі повідомлення. У точках, де додаток інтегрується з іншими, повинен бути невелика частина коду, яка виконує частину програми інтеграції. Коли інтеграція реалізована за допомогою повідомлень, невеликою частиною коду, який приєднує додаток до системи обміну повідомленнями, є шлюз обміну повідомленнями.

Використовуйте шлюз обміну повідомленнями, клас, який обгортає виклики, пов’язані з повідомленнями, та піддає додатку специфічні доменні методи.

Шлюз обміну повідомленнями інкапсулює специфічний для повідомлення код (наприклад, код, необхідний для надсилання або отримання повідомлення) та відокремлює його від решти коду програми. Таким чином, лише система шлюзу повідомлень знає про систему обміну повідомленнями; решта коду програми не робить. Шлюз обміну повідомленнями відкриває ділову функцію для решти програми, так що замість того, щоб вимагати від програми встановлювати такі властивості, як повідомлення. Повідомлення ReadPropertyFilter.AppSpecific, шлюз обміну повідомленнями відкриває такі методи, як GetCreditScore, які приймають сильно набрані параметри, як і будь-який інший метод. Шлюз обміну повідомленнями - це специфічна для обміну повідомленнями версія загальнішої схеми шлюзу, що зображена на рисунку 2.24 [8]:

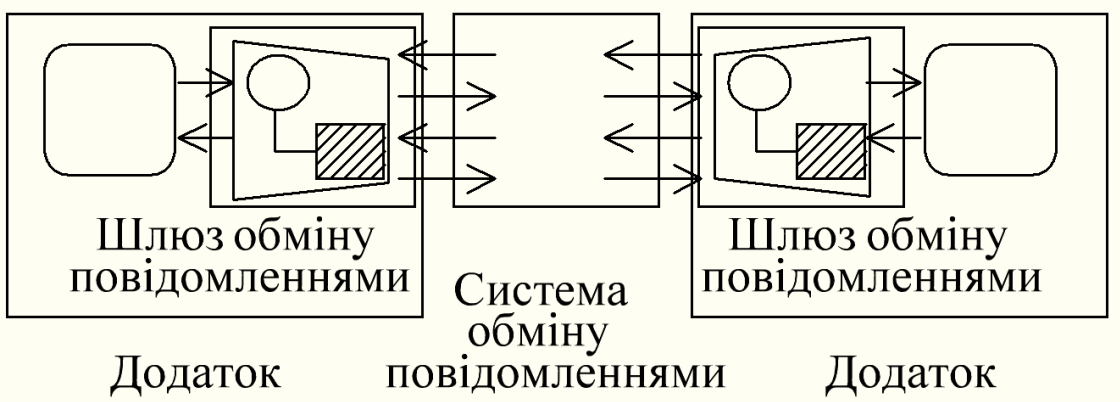

Рисунок 2.24 – Схема шлюзу обміну повідомленнями

Створіть окремий картограф повідомлень, який містить логіку відображення між інфраструктурою обміну повідомленнями та об'єктами домену. Ні об’єкти, ні інфраструктура не знають про існування картографа повідомлень. Схема картографу повідомлень зображено на рисунку 2.25.

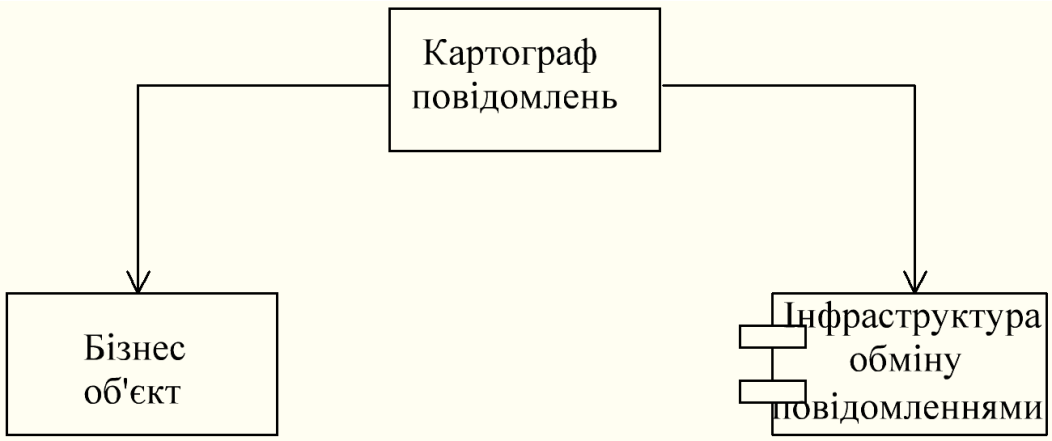

Рисунок 2.25 – Схема картографу повідомлень

Використовувати транзакційний клієнт - зробіть сеанс клієнта із системою обміну повідомленнями транзакційним, щоб клієнт міг задати межі транзакцій.

Як відправник, так і одержувач можуть бути транзакційними. Що стосується відправника, повідомлення не "дійсно" додається до каналу, поки відправник не здійснить транзакцію. З одержувачем повідомлення не видаляється з каналу "дійсно", поки одержувач не здійснить транзакцію. Відправник, який використовує явні транзакції, може використовуватися з одержувачем, який використовує неявні транзакції, і навпаки. Один канал може мати поєднання неявно та явно транзакційних відправників; він також може мати комбінацію приймачів. Створіть одержувач таким чином, щоб він був Ідентифікаційним приймачем, який може безпечно отримувати одне і те ж повідомлення кілька разів. Схему клієнту транзакцій показано на рисунку 2.26.

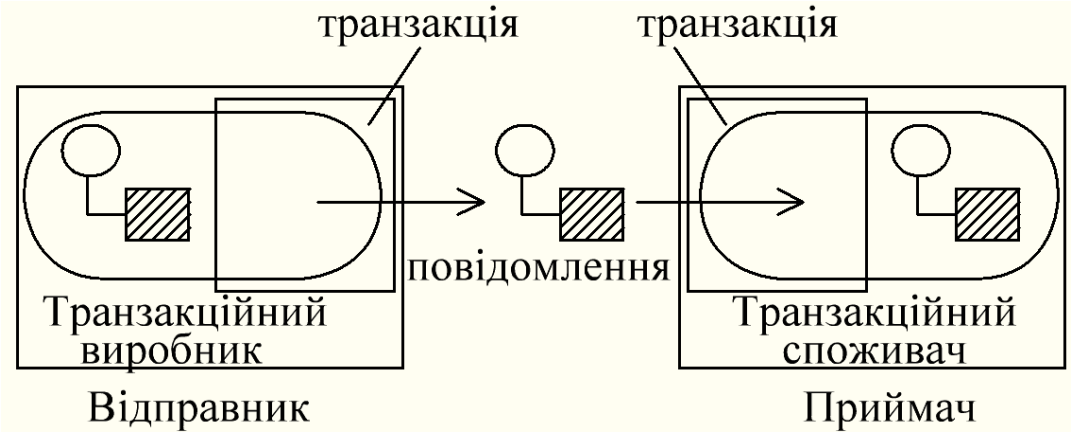

Рисунок 2.26 – Схема клієнту транзакцій

Термін ідентифікації використовується в математиці для опису функції, яка дає той самий результат, якщо вона застосовується до себе, тобто f (x) = f (f(x)). У пересиланні повідомлення це поняття перекладається на повідомлення, яке має той самий ефект, будь то отримане один чи кілька разів. Це означає, що повідомлення можна сміливо відновлювати, не викликаючи проблем, навіть якщо одержувач отримує дублікати одного і того ж повідомлення [1].

Створіть сервіс-активатор, який з'єднує повідомлення на каналі з послугою, до якої можна отримати доступ.

Активатор послуги може бути одностороннім (лише запит) або двостороннім. Послуга може бути такою ж простою, як і виклик методу - синхронний та не віддалений - можливо, частина службового шару. Активатор може бути жорстко кодованим, щоб завжди викликати одну і ту ж послугу, або використовувати відображення для виклику послуги, зазначеної в повідомленні. Активатор обробляє всі деталі обміну повідомленнями та викликає послугу, як і будь-який інший клієнт, так що служба навіть не знає, що її викликають через обмін повідомленнями. Схема сервісу активатора зображено на рисунку 2.27

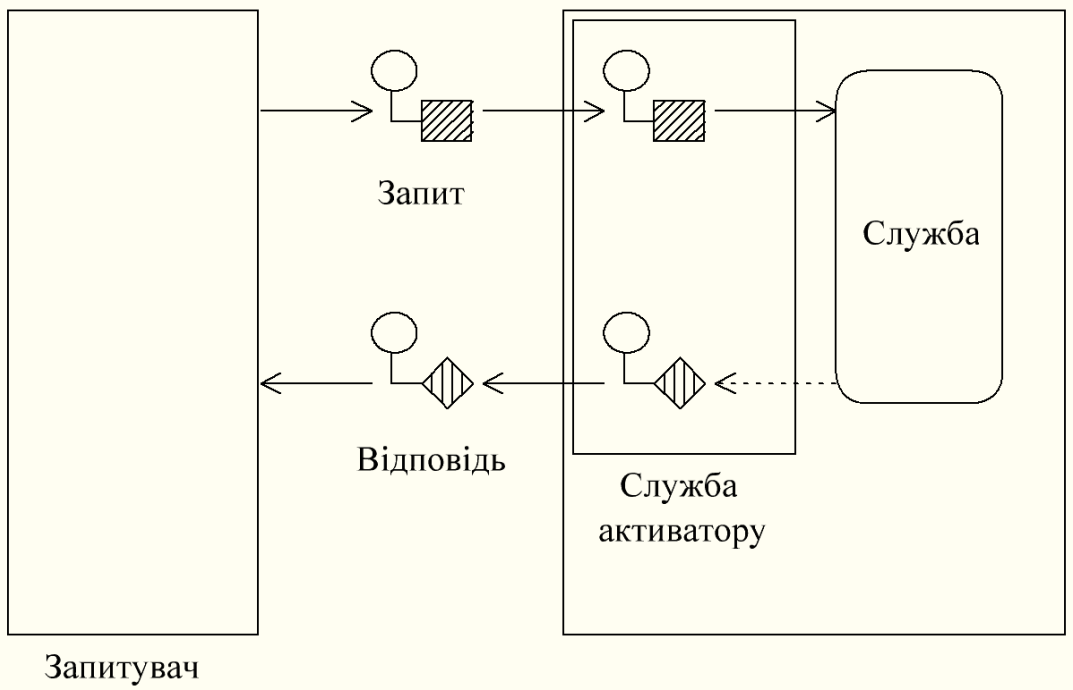

Рисунок 2.27 – Схема сервісу активатора

3 ПРАКТИЧНІ ЗАСТОСУВАННЯ ШАБЛОНІВ ДЛЯ ІНТЕГРАЦІЇ НА ПРИКЛАДІ ІНТЕРНЕТ-МАГАЗИНУ

Розглянемо інтернет-магазин одягу. Постачальник разом з товаром записує надходження у файл-накладну певного формату. Необхідно зберегти надходжння у внутрішню базу даних та сповістити зовнішню систему та цільову аудиторію про надходження та кількість товарів. При цьому система повинна бути гнучкою та легко розширюваною. У випадку, коли доведеться внести зміни у бізнес-модель.

Проаналізувавши наявні імплементації шаблонів інтеграції у додатки корпоративного рівня, ми дійшли висновку, що найбільш повними та стабільними є Apache Camel, та Spring Integration. Так як більша частина існуючої системи використовує за основу Spring Framework, то перевагу було віддано Spring Integration через більш щільну інтеграцію.

Spring Integration має чималу частину вже існуючих імплементацій адаптерів, Gateway, каналів тощо. Такий підхід значно прискорює розробку.

Точкою входу в систему є файл формату csv., приклад якого представлено на рисунку 3.1:

	A	B	C	D	E
1	name	type	size	count	price
2					
3	Nike V Cush Ankle-3P	socks	M	4	416
4	GoodSox Skeleton Fish	socks	L	20	49
5	HEAD Performance Short Crew	socks	S	7	299
6	Nike M Nsw Ce Top Ss Wash	T-shirt	M	9	690
7	YAPPI MB1073	T-shirt	L	15	279
8	Superdry	T-shirt	S	7	792
9	Cornette 007-18 Merry Christmas	underwear	M	9	690
10	Atlantic MP-1440	underwear	M	12	478
11	Cornette Classic	underwear	L	30	275

Рисунок 3.1 – Приклад файлу csv формату

Система інтеграціє є досить об’ємною, тому спочатку розглянемо принцип роботи загалом, а після будемо розглядати детально кожну частину. З файлового адаптеру повідомлення йде через перетворювач у дублікатор повідомлень. У цьому місці програма розділяється на 2 потоки керування. Кінцем роботи одного з потоків є пост на сторінці акаунту інтернет-магазину у Twitter. Другий потік робить низку перетворень та в результаті дані про товар можуть бути записані у базу даних чи будуть експортовані у стороні сервіси в залежності від типу товару. Повна схема системи інтеграції для інтернет-магазину представлена на рисунку 3.2:

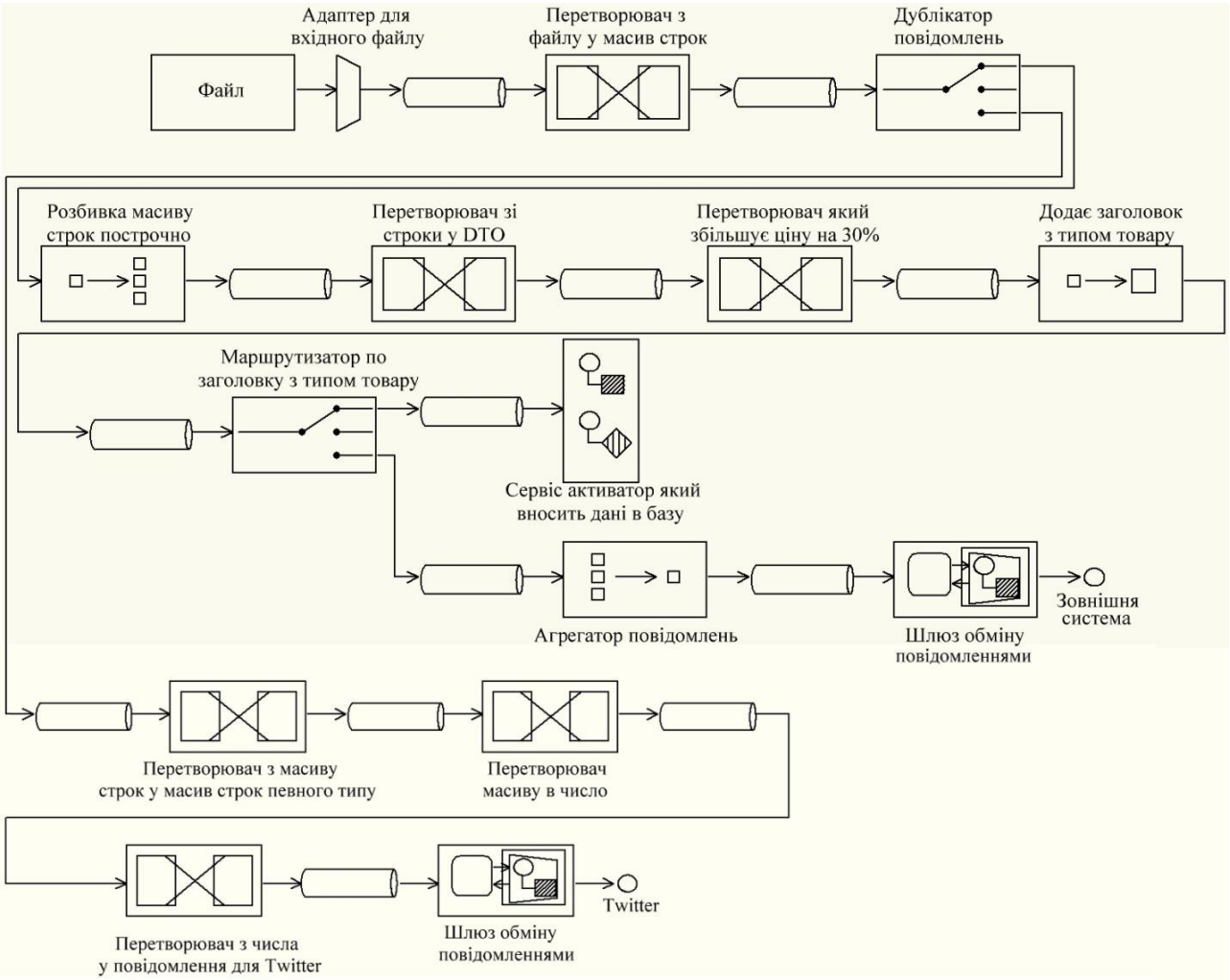

Рисунок 3.2 – Схема системи інтеграції

Умовно розділимо нашу систему на 4 частини. Розглянемо принцип роботи більш детально. У конфігурації адаптера вказується шлях до папки,

яку адаптер періодично опитує у пошуках нового файлу. Майже одразу після того як файл надходить до цієї папки він зчитується адаптером та потрапляє до каналу у вигляді повідомлення. На цей канал підписаний перетворювач, який трансформує файл у масив строк після цього відправляє повідомлення на наступний канал, з якого зчитує дублікатор повідомлень, який клонує повідомлення у 2 різні канали, які є початком 2 незалежних потоків виконання програми. Схема першої частини схеми зображено на рисунку 3.3:

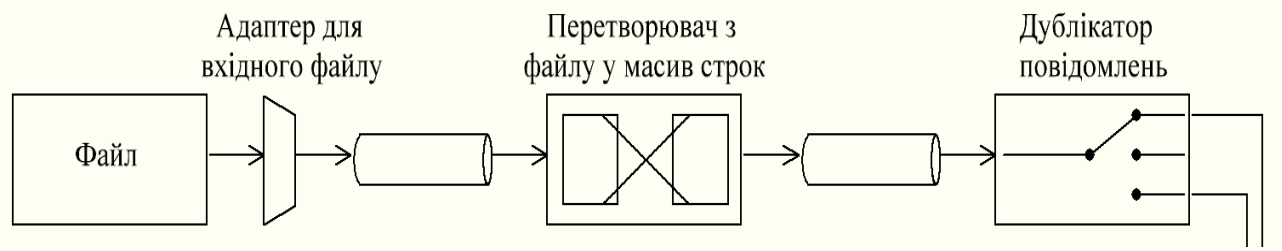

Рисунок 3.3 – Схема першої частини системи інтеграції

Одне з цих повідомлень потрапляє у канал, на який слухає перетворювач, що видаляє з масиву строк строки з товарами певного типу. Далі повідомлення надходить у канал на який підписаний перетворювач, що рахує кількість строк певного типу з наступного каналу повідомлення витягує перетворювач, що містить текст повідомлення про надходження групи товарів певного типу, та додає до цього тексту число про надходження. Одразу після цього повідомлення потрапляє через канал у попередньо сконфігурований шлюз обміну повідомленнями для Twitter. Результатом цієї операції є пост про надходження товару є пост на сторінці акаунту інтернет-магазину у Twitter. Схема другої частини схеми зображено на рисунку 3.4:

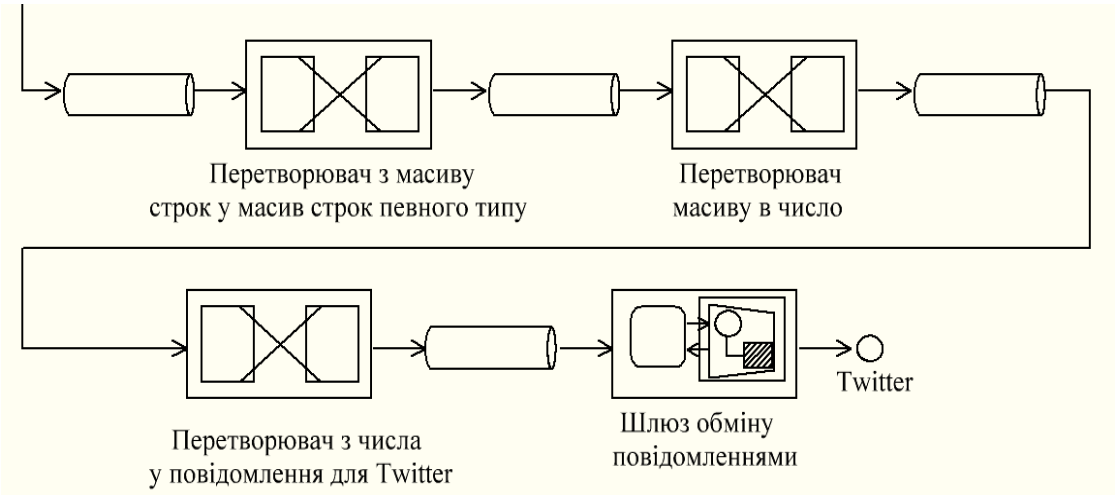

Рисунок 3.3 – Схема другої частини системи інтеграції

Друге з клонованих повідомлень потрапляє у канал, який слухає розділяч повідомлень, що розділяє масив строк у строки як окремі повідомлення. Наступним етапом є робота перетворювача, що трансформує строку у DTO (date transfer object). Далі потрапляє у канал і прямує до перетворювача, що підвищує початкову ціну на 30%. Після цього повідомлення потрапляє у канал на який підписаний header enricher, що додає у метадату повідомлення додає тип товару. Схема третьої частини системи інтеграції зображено на рисунку 3.4

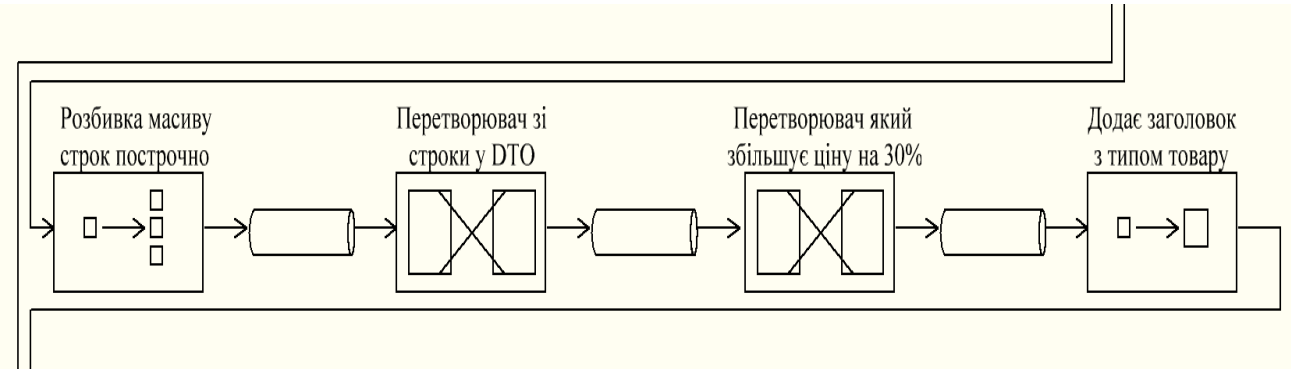

Рисунок 3.4 – Схема третьої частини системи інтеграції

Продовженням потоку є канал з якого читає повідомлення маршрутизатор, що у основі заголовку у метадаті повідомлення, відправляє повідомлення у один з 2 каналів. Перший канал надсилає повідомлення на

сервіс активатор, що зберігає дані про товар у базі. Другий канал функцією якого є передача даних до агрегатора повідомлень, який накопичує повідомлення у групи по n одиниць, є можливість конфігурувати значення n. Після накопичення повної групи, розміром n, вона відправляється до HTTP шлюзу обміну повідомленнями, що за допомогою HTTP-запиту надсилає цю групу товарів у стороній сервіс для подальшої обробки. Групування зроблено з метою зменшення навантаження на сторонній сервіс. Схема четвертої частини схеми зображено на рисунку 3.5:

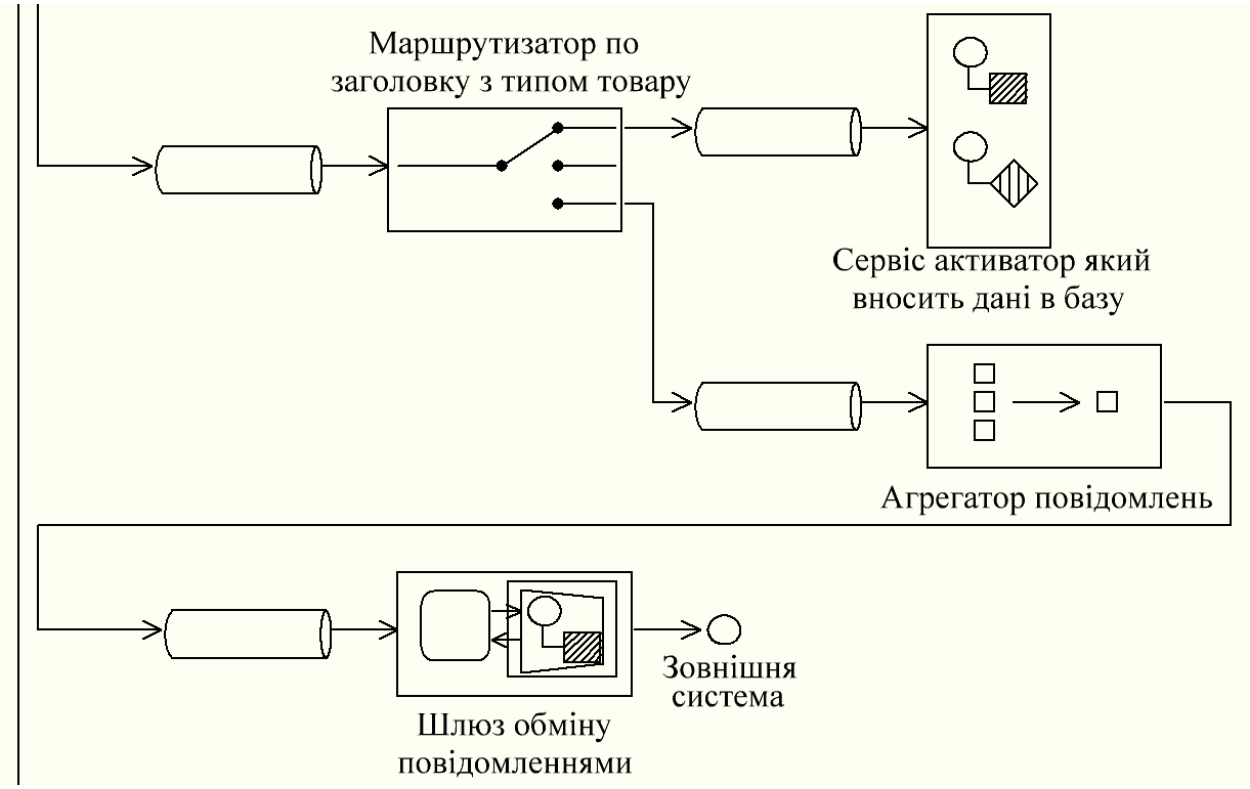

Рисунок 3.5 – Схема четвертої частини системи інтеграції

На основі вимог на технічного завдання була проведена науково-дослідницька робота та розглянуті різні способи та шляхи інтеграції різних систем. Результатом роботи є система-інтеграції для інтернет-магазину. Вона інтегрується з іншими зовнішніми системами за допомогою FTP (file transport protocol), HTTP(hyper text transport protocol), бази даних. Розроблена система розроблялась з вимогою бути гнучкою та легко розширюваною у випадку зміну вимог до неї.

ВИСНОВКИ

Робота присвячена дослідженню шаблонів проектування для інтеграції у додатках корпоративного рівня. Розглянуті різні засоби та технології для вирішення цієї проблеми, запропоноване рішення на основі якого розроблено дизайн системи. Було ураховано вимоги на технічне завдання, та проведена науково-дослідницька робота та розглянуті різні способи та шляхи інтеграції різних систем.

В роботі представлен детальний огляд технологій і засобів для інтеграції додатків корпоративного рівня на основі Enterprise integration patters.

В рамках цієї роботи, за результатами проведеного дослідження моделі, був розроблений дизайн інтеграції інтернет-магазину з зовнішніми сервісами.

Мета кваліфікаційної дипломної роботи досягнута в повному обсязі.

ПЕРЕЛІК ПОСИЛАНЬ

1. Addison W. Enterprise integration patterns - designing, building and deploying messaging solutions, London:Williams, 2017. 672p.

2. Messaging pattens overview URL: https://www.enterpriseintegrationpatterns.com/patterns/messaging/ (дата звернення: 01.10.2019).

3. Enterprise integration patterns URL: https://en.wikipedia.org/wiki/Enterprise_Integration_Patterns (дата звернення: 08.10.2019).

4. Enterprise integration patterns URL: https://camel.apache.org/manual/latest/enterprise-integration-patterns.html (дата звернення: 09.10.2019).

5. Spring integration URL: https://spring.io/projects/spring-integration (дата звернення: 15.10.2019).

6. Spring integration references guide URL: https://docs.spring.io/spring-integration/docs/5.2.2.RELEASE/reference/html/ (дата звернення: 16.10.2019).

7. Camel Apache summary URL: https://camel.apache.org/manual/latest/index.html (дата звернення: 24.10.2019).

8. Enterprise integration patterns URL: https://martinfowler.com/books/eip.html (дата звернення: 28.10.2019).

9. Enterprise integration patterns with WSO2 ESB URL: https://docs.wso2.com/display/IntegrationPatterns/Enterprise+Integration+Patterns+with+WSO2+ESB (дата звернення: 01.11.2019).

10. Pattern-oriented software architecture on patters and pattern languages URL: https://books.google.com.ua/books?id=wzplRf3uhEC&printsec=frontcover&hl=ru&vq=%22Enterprise+Integration+Patterns%22&source=gbs_citations_mod

ule_r&cad=4#v=onepage&q=%22Enterprise%20Integration%20Patterns%22&f=false (дата звернення: 02.11.2019).

11. The making of information systems software engineering and management in a globalized world URL: https://books.google.com.ua/books?id=ggVaezlfOCcC&printsec=frontcover&hl=ru&vq=%22Enterprise+Integration+Patterns%22&source=gbs_citations_module_r&cad=4#v=onepage&q=%22Enterprise%20Integration%20Patterns%22&f=false (дата звернення: 04.11.2019).