УДК 004.41
DOI: https://doi.org/10.17721/AIT.2024.1.07

Олексій БИЧКОВ, д-р техн. наук, проф.
ORCID ID: 0000-0002-9378-9535
e-mail: oleksiibychkov@knu.ua
Київський національний університет імені Тараса Шевченка, Київ, Україна

Микола МОРОЗ, асп.
ORCID ID: 0000-0001-6953-683X
e-mail: mukolamoroz@knu.ua
Київський національний університет імені Тараса Шевченка, Київ, Україна

"MANAGER-DISPATCHER": ПАТЕРН ЗАБЕЗПЕЧЕННЯ АДАПТИВНОЇ ПОВЕДІНКИ ПРОГРАМНИХ СИСТЕМ НА ОСНОВІ ПОДІЙ

Вступ. ***Адаптивна поведінка сучасних програмних систем стає ключовим чинником їх успішного функціонування в умовах зовнішніх і внутрішніх дестабілізуючих факторів. Програми, які працюють із критичними даними або виконують важливі операції, повинні забезпечувати безперервність роботи, незважаючи на збої, атаки чи помилки. Для досягнення цієї мети пропонують різні підходи. Одним із них є застосування патернів проєктування, які дозволяють забезпечити надійність та адаптивність системи. У цій статті представлено патерн "Manager-Dispatcher", який поєднує властивості патернів "Publish-Subscribe" та "Strategy" для забезпечення адаптивної поведінки програмних систем за рахунок оброблення подій.***

Методи. ***Для розроблення патерну "Manager-Dispatcher" використано методи модульного проєктування та динамічного оброблення подій. Патерн передбачає автоматичний вибір стратегії функціонування модуля на основі подій, що відбуваються у системі. Проведено теоретичний аналіз існуючих підходів до адаптації поведінки системи і на основі цього розроблено новий патерн, який дозволяє динамічно змінювати стратегії роботи модулів у відповідь на змінні умови середовища, які визначаються подіями у системі. Розглянуто кілька гіпотетичних сценаріїв застосування для ілюстрації роботи патерну. Розроблено й описано приклад програмної системи із застосуванням патерну.***

Результати. ***Розроблений патерн "Manager-Dispatcher" дозволяє програмним модулям автоматично адаптувати стратегії функціонування на основі подій у системі. Основні переваги патерну включають модульність, розширюваність та адаптивність поведінки. Патерн може бути корисним у вбудованих системах, системах реального часу й інтерактивних інтерфейсах, де важливо забезпечити швидку та гнучку реакцію на події.***

Висновки. ***Патерн "Manager-Dispatcher" пропонує перспективний підхід до проєктування адаптивних програмних систем, орієнтованих на події. Завдяки можливості динамічної зміни стратегій функціонування, патерн забезпечує високий рівень гнучкості в умовах динамічних змін. У подальших дослідженнях планується вдосконалення патерну та розроблення інструментів для його полегшеного впровадження і тестування. Запропонований підхід сприяє побудові модульних та адаптивних систем, здатних забезпечувати стабільне функціонування навіть у складних умовах.***

Ключові слова: ***патерни проєктування, адаптивна поведінка, оброблення подій, автономні системи.***

Вступ

Актуальність дослідження обумовлена зростанням складності сучасних програмних систем і вимогою до них працювати в умовах постійно змінного середовища. Програмні системи все частіше стикаються з вимогами забезпечення гнучкості й адаптивності для підтримки безперебійної роботи під впливом внутрішніх і зовнішніх подій. У контексті швидкого розвитку технологій і збільшення обсягів оброблюваної інформації, адаптивні системи дозволяють підвищити продуктивність і знизити ризики збоїв. Тому створення патернів, які надають системам можливість адаптувати поведінку в реальному часі, є важливим напрямом досліджень, що сприяє розвитку масштабованих рішень для сучасного програмного забезпечення.

Мета цього дослідження – розробити патерн проєктування, який забезпечує адаптивну поведінку програмних систем через динамічну зміну стратегій функціонування модулів у відповідь на події та зміни умов середовища.

Завданнями цього дослідження є: провести аналіз існуючих підходів і патернів для забезпечення адаптивної поведінки програмних систем, продумати функціональність, реалізувати необхідну функціональність у новому патерні проєктування, виконати аналіз отриманих теоретичних результатів щодо розробленого патерну.

Ідея патернів проєктування отримала популярність завдяки роботі "Банди чотирьох" (Gamma et al., 1995). У книзі "Design Patterns: Elements of Reusable Object-Oriented Software" автори виклали основи використання патернів для створення повторно використовуваного об'єктно-орієнтованого програмного забезпечення.

У роботі (Bruns, & Dunkel, 2013) автори пропонують каталог патернів, які забезпечують розроблення архітектур на основі подій. Вони вводять різні патерни для структурування і створення комплексних систем оброблення подій.

У роботі (Mannava, & Ramesh, 2012) запропоновано загальний патерн для адаптивної реконфігурації автономних обчислювальних систем. Цей патерн спрямовано на автоматизацію самокерування систем, що дозволяє їм адаптуватися до змін у середовищі без залучення адміністраторів. Патерн базується на потоках вхідних даних, які аналізуються і за наявності тригерів патерн на базі правил приймає рішення про реконфігурацію. Подібні ідеї авторів також присутні у роботі (Mannava, & Ramesh, 2011), де вони описують патерн, який дозволяє автономно компонувати й адаптувати вебсервіси на основі контексту. Цей патерн є розширенням патернів Case-based Reasoning, Strategy, Observer і спрямований на забезпечення адаптивності сервісів до змінних умов.

У роботах (Ramirez, 2008) та (Ramirez, & Cheng, 2010) автори аналізували понад тридцять досліджень і проєктів, виявляючи патерни, що сприяють розробленню адаптивних систем. У результаті аналізу узагальнено й описано каталог патернів для розроблення динамічних адаптивних систем; автори також запропонували класифікацію для розроблених патернів згідно з адаптивним функціями: патерни моніторингу, патерни прийняття рішення та патерни реконфігурації.

В цій роботі ми представляємо патерн адаптивного оброблення подій "Manager-Dispatcher", який поєднує властивості патернів "Publish-Subscribe" та "Strategy" для забезпечення адаптивної поведінки системи. На відміну від традиційних підходів, наш патерн дозволяє автоматично змінювати стратегії функціонування програмного модуля залежно від

© Бичков Олексій, Мороз Микола, 2024

поточного контексту та стану системи. Порівняно з рішеннями у джерелах (Mannava, & Ramesh, 2011; Mannava, & Ramesh, 2012; Ramirez, & Cheng, 2010), які розглядають реконфігурацію системи як необхідну умову забезпечення адаптивності системи, ми пропонуємо змінювати стратегію функціонування роботи програмного модуля. Також патерн забезпечує подальший розвиток підходів проєктування патернів, описаний у зазначених джерелах.

Методи

Оброблення подій є фундаментальною частиною багатьох програмних систем, від вбудованих систем контролю до комплексних користувацьких інтерфейсів. Історично, патерн "Observer" (Gamma et al., 1995) надав базовий механізм для взаємодії видавців і підписників на події, дозволяючи об'єктам реагувати на зміни в інших об'єктах без прямої залежності між ними, але натомість кожному підписнику на подію варто знати про видавця подій і самостійно підписуватись на події. Гнучкішим підходом до керування подіями став патерн "Publish-Subscribe" або його менш відома версія "Event Channel" (Buschmann, 1996, pp. 339–343). Цей патерн дозволяє видавцям генерувати події, не знаючи, хто є підписниками цих подій. Підписники, у свою чергу, можуть отримувати нотифікації про ті події, які їх цікавлять, без необхідності знати про джерело подій. Саме тому "Publish-Subscribe" забезпечує високий рівень децентралізації та гнучкості.

Завдяки подіям і реакціям на них програмні модулі можуть формувати неявні слабко зв'язані ієрархічні структури залежності, тобто один модуль залежатиме від подій, що виникають в іншому модулі. Таким способом формуватиметься певна ієрархічна модульна архітектура, в якій модулі не знатимуть про модулі, які залежні від них, й орієнтуватимуться лише на події, що виникають. Зв'язки модулів на основі подій визначатимуться динамічно, хоча можуть бути запропоновані статичні підходи для формування архітектури залежності.

Під програмними модулями ми розуміємо окремі частини програмної системи, які виконують певну функцію системи, причому за рахунок підписок на події ці модулі можуть змінювати алгоритми виконання завдань і забезпечення функцій, що ми називаємо загальним терміном "зміна стратегії". Патерн "Strategy" (Gamma et al., 1995) пропонує механізм вибору поведінки об'єкта під час виконання програми, надаючи можливість змінювати алгоритми оброблення без зміни контексту їх виконання. Зміна стратегії програмного модуля може полягати у запуску процесу відновлення, зміні алгоритму роботи для збереження виконання функції програмного модуля, переході на резервні ресурси, адаптації до нових умов середовища, зміні пріоритетів оброблення задач тощо.

Обробник події не обов'язково має змінювати стратегію. Для модуля може існувати окрема множина подій, які він не здатен самостійно опрацювати і запустити відповідні дії, серед яких можуть бути усунення причин, протидії, корекції стану. Саме тому, якщо модуль зазнає невідомої або критичної помилки, що перешкоджає його подальшому функціонуванню, він також може сповістити залежні модулі про те, що цей програмний модуль є несправним, а залежні модулі мають відповідним способом змінити свої стратегії. Таким чином програмний модуль передає відповідальність за опрацювання події на ієрархічно вищі модулі.

Саме тому, використовуючи запропонований патерн, програмний модуль може бути одночасно і видавцем, і підписником. Це означає, що він має можливість як реагувати на отримані події, так і генерувати нові події у відповідь на оброблення поточних подій. Модуль може генерувати більш високорівневу помилку та повідомити про неї своїх підписників, покладаючи відповідальність за її опрацювання на них. Цей підхід нагадує концепцію каскадного оброблення подій, де оброблення однієї події може призводити до генерації інших подій, які у свою чергу обробляються іншими модулями. Це забезпечує багаторівневе й ієрархічне оброблення помилок і подій, що підвищує надійність і гнучкість системи.

Опис патерну. "Manager-Dispatcher" – це поведінковий патерн проєктування, який дозволяє компонентам системи ефективно реагувати на різноманітні події, адаптуючи свою поведінку на основі поточного контексту. В основі патерну лежить ідея про те, що обробник подій повинен мати можливість адаптувати поведінку компонента у разі виникнення певних подій, вибираючи найбільш підходящу стратегію функціонування компонента з набору доступних стратегій. Такий підхід дозволяє системі залишатися гнучкою і стійкою у складних умовах. Також цей підхід змушує розробника визначати, на які події варто реагувати компонентові, а також проєктувати відповідні стратегії поведінки.

Назва патерну відображає функціональну роль двох фахів – "менеджера", який ухвалює рішення, і "диспетчера", який виконує координацію. Обираючи цю назву, ми вкладали такий сенс: менеджер керує і робить вибір (в нашому випадку – керує / приймає події і вибирає стратегії для компонента), а диспетчер виконує конкретні дії та розподіл роботи / обов'язків (в нашому випадку – розподіл подій між компонентами та виконання стратегій).

Діаграму класів патерну "Manager-Dispatcher" наведено на рис. 1. Патерн складається з таких основних елементів:

- подія (Event): об'єкт, що представляє зміну стану, зовнішні зміни тощо; кожна подія має заздалегідь визначений тип (EventType), а також містить корисну інформацію, яка може бути необхідна компоненту-підписнику (час, видавець, причина, опис тощо);
- видавець (AnEventProducer): компонент, який генерує події; видавець не залежить і не володіє інформацією про підписників;
- підписник (Subscriber): компонент, який реагує на певні типи подій; він динамічно підписується на ті типи подій, які його цікавлять, і має змогу вибирати відповідну стратегію функціонування як реакцію на подію;
- менеджер подій (EventManager): компонент, який відповідає за координацію між видавцями і підписниками, забезпечуючи доставку події тим підписникам, які підписались на цей тип події;
- селектор стратегій (StrategySelector): визначає логіку вибору відповідної стратегії функціонування компонента на основі поточного контексту; контекстом може служити як інформація, що надана у події, так і стан системи;
- стратегія (Strategies): алгоритм функціонування компонента, який може бути вибраний підписником у результаті оброблення конкретної події, адаптація поведінки компонента до змін.

Взаємодія компонентів. На стадії ініціалізації системи кожен компонент-підписник має здійснити підписку на тип події, що його цікавить, через інтерфейс IEventSubscriptionService. Коли подія генерується в системі, EventManager нотифікує всіх підписників, які підписались на цей тип події. Кожен підписник у власному методі оброблення події може використовувати StrategySelector для вибору найкращої стратегії подальшого функціонування залежно від поточного

ISSN 2788-6603

контексту. Контекстом, наприклад, може бути стан системи та/або стан зовнішнього середовища. Після вибору стратегії підписник продовжує функціонувати з адаптованою поведінкою.

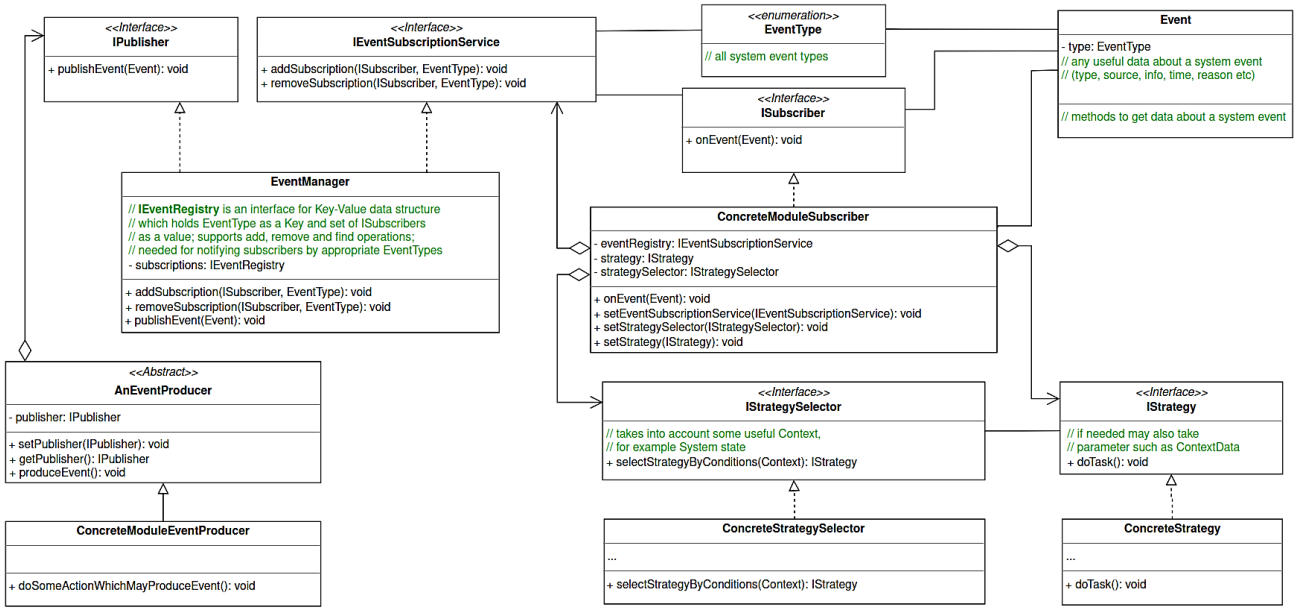

Рис. 1. UML-діаграма класів патерну "Manager-Dispatcher"

Цей патерн забезпечує високий рівень модульності та розширюваності, дозволяючи легко додавати нові стратегії, селектори стратегій, підписників, видавців і події без необхідності зміни вже існуючого коду. Він також сприяє відокремленню відповідальності – кожен компонент фокусується на своєму конкретному завданні, знижуючи складність системи та полегшуючи тестування й підтримку.

Для ілюстрації процесу оброблення подій у патерні на рис. 2 наведено діаграму послідовності. Ця діаграма демонструє взаємодію між основними компонентами патерну. Вона відображає послідовність кроків, що відбуваються від моменту генерації події до її оброблення підписником, при цьому оброблення події може полягати і у виборі найкращої стратегії подальшого функціонування, і у нотифікації підписників, що підписані на події, створювані поточним підписником.

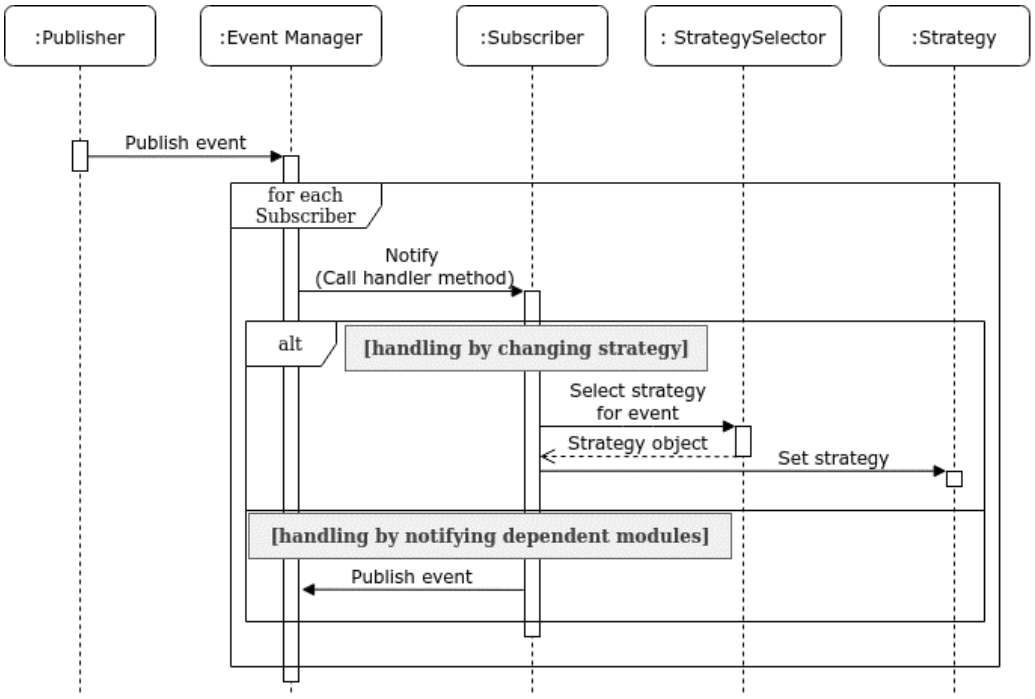

Рис. 2. UML-діаграма послідовності оброблення подій у патерні "Manager-Dispatcher"

Приклади використання. Щоб продемонструвати можливості патерну "Manager-Dispatcher", розглянемо кілька гіпотетичних сценаріїв його застосування:

1. Вбудовані системи реального часу. У системах реального часу, таких як контролери промислового обладнання, важливо швидко реагувати на зміни стану сенсорів та інших входів. Використовуючи наш патерн, система може автоматично змінювати стратегії залежно від поточного стану обладнання, забезпечуючи безперервність роботи та мінімізуючи ризик збоїв.

2. Інтерактивні інтерфейси. У інтерактивних інтерфейсах користувача, де необхідна гнучка реакція на дії користувачів, наш патерн дозволяє швидко адаптувати стратегії функціонування для забезпечення плавної та ефективної роботи системи. Наприклад, інтерфейс може автоматично змінювати спосіб відображення даних у відповідь на зміну умов навантаження або вимог користувача.

3. Системи моніторингу. У системах моніторингу, таких як системи безпеки, патерн використовують для автоматичної адаптації до нових загроз. Наприклад, у разі виявлення підозрілої активності система може змінити стратегії реагування – активувати додаткові заходи безпеки або перенаправити трафік.

Ми вважаємо, що патерн буде особливо корисним для вбудованих систем, оскільки вони зазвичай функціонують у межах однієї програми (тобто не мають ОС і різних процесів, які потрібно синхронізувати між собою), а також потребують гнучкості та необхідності реагування на події середовища.

Результати

Теоретичні результати (оцінювання патерну), що представлені у статті, реалізовано для побудови простої системи для трекінгу цілі. Приклад створено з використанням мови C++. У прикладі відбувається аналіз зображення з відеопотоку та пошук об'єкта. Критерій детектування об'єкта – це наявність червоного кольору і розмір не менше ніж 50 x 50 пікселів. Детектування об'єкта відбувається за окремим алгоритмом, а коли об'єкт буде знайдено на зображенні, то буде застосовано алгоритм трекінгу об'єкта. У прикладі патерн керує подіями в описаній програмній системі; у разі виникнення події відбувається адаптування програмної системи до події, що виникла.

Описана система має такі модулі: модуль аналізу зображення, модуль-стратегія детектування об'єкта, модуль-стратегія трекінгу об'єкта, модуль відео провайдера, модуль менеджера подій (клас EventManager), модуль нотифікацій. Модуль аналізу зображення та модуль нотифікацій є підписниками на події. В системі є два типи подій: "ціль знайдено" та "ціль втрачено". За наявності цих подій модуль аналізу зображення змінює свою стратегію функціонування, а саме: за замовчуванням модуль аналізу зображення має стратегію аналізу – детектування; якщо стратегія детектування знаходить об'єкт, то генерується відповідна подія, а модуль аналізу зображення змінює стратегію аналізу на трекінг; якщо стратегія трекінгу детектує втрату цілі, то генерується відповідна подія і модуль аналізу зображення змінює стратегію аналізу на детектування. Модуль нотифікацій підписується на обидві події і пише інформацію про отриману подію у консоль.

Наступний код демонструє роботу запропонованого патерну.

Файл *main.cpp*:

```
int main()
{
	cv::Mat image;
	cv::namedWindow("Video Player");
	app::VideoProvider video_provider{};

	core::EventManager event_manager{};
	core::AnEventProducer event_producer{&event_manager};
	app::ImageAnalysisStrategySelector strategy_selector{&event_producer};
	// модуль аналізу зображення
app::ImageAnalyzer image_analyzer{&event_manager, &strategy_selector};
	// модуль нотифікацій
	app::Notifier logger{&event_manager};

	while (true) {
	if (!video_provider.getFrame(image)) { break; }

	// зміна стратегій оброблення кадрів відбувається всередині модуля аналізу зображення
	if (!image_analyzer.processFrame(image))
	std::cout << "Image analysis module could not process frame!" << std::endl;

	cv ::imshow("Video Player", image);
	}
	return 0;
}
```

image_analyzer – це об'єкт, який відповідає за аналіз зображення, поведінку якого було описано вище. Його обробник подій та метод оброблення кадру виглядає так:

```
void ImageAnalyzer::onEvent(core::Event event)
{
    strategy_ = strategy_selector_->selectStrategyForEvent(event);
}
    bool ImageAnalyzer::processFrame(cv::Mat& image)
{
    if (!strategy_) { return false; }
    return strategy_->process(image);
}
```


ISSN 2788-6603

А метод для вибору стратегії:

```
core::IImageAnalysisStrategy* ImageAnalysisStrategySelector::selectStrategyForEvent(core::Event event)
{
    switch (event.getType()) {
    case core::EventType::OBJECT_FOUND:
    return new Tracker(event_producer_, *((cv::Rect*)event.getData()));
    case core::EventType::OBJECT_LOST:
    return new Detector(event_producer_);
    default:
    return getDefaultStrategy();
}
}
```

Клас нотифікатора є дуже простим і ось так виглядає його конструктор (де відбувається підписка на події) та обробник подій:

```
Notifier::Notifier(core::IEventSubscriptionService* event_subscription_service)
{
    event_subscription_service->addSubscription(this, core::EventType::OBJECT_FOUND);
    event_subscription_service->addSubscription(this, core::EventType::OBJECT_LOST);
}

void Notifier::onEvent(core::Event event)
{
    switch (event.getType()) {
    case core::EventType::OBJECT_FOUND:
    std::cout << "Got an event with a type: OBJECT_FOUND." << std::endl;
    break;
    case core::EventType::OBJECT_LOST:
    std::cout << "Got an event with a type: OBJECT_LOST." << std::endl;
    break;
    default:
    break;
}
}
```

Отже, з прикладу можна побачити, що модуль аналізу зображення демонструє адаптацію поведінки у відповідь на подію; цей компонент працює на основі стратегій і здатний змінювати їх за логікою окремого селектора стратегій, що є гнучким і розширюваним рішенням. Зауважимо, що модуль аналізу зображення нічого не знає про окремі стратегії. Модулі у цьому прикладі зв'язані подіями, що також сприяє гнучкості й подальшому масштабуванню.

З повною версією прикладу можна ознайомитись у роботі М. Мороза (https://github.com/mr-holodok/director-dispatcher-pattern-example).

Узагальнюючи отримані результати, архітектура патерну "Manager-Dispatcher" демонструє такі властивості:

- гнучкість реакції: система може автоматично адаптуватися до змін у середовищі, що значно підвищує її гнучкість;
- модульність, розширюваність і масштабованість патерну;
- зниження залежності між компонентами: завдяки централізації керування подіями через EventManager, зменшується зв'язність між видавцями та підписниками, що спрощує модифікацію та розширення системи;
- підвищення відмовостійкості: динамічний вибір стратегій дозволяє системі краще протистояти помилкам або збоям, адаптуючи поведінку підписників залежно від поточного стану системи та змін середовища.

Наукова цінність дослідження полягає у тому, що розроблено новий патерн проєктування, який забезпечує адаптивну поведінку програмних модулів у відповідь на події, що виникають у програмній системі. Ця функціональність може використовуватись для розв'язання таких практичних задач: автоматичного налаштування програмних компонентів під поточні умови роботи, підвищення надійності систем завдяки швидкому реагуванню на зовнішні та внутрішні зміни, а також для створення масштабованих і модульних рішень у вбудованих системах, системах реального часу й інтерактивних інтерфейсах.

Дискусія і висновки

Розроблений патерн "Manager-Dispatcher" є одним з інструментів для створення адаптивних і гнучких програмних систем, але він не є "срібною кулею", яка розв'яже всі можливі проблеми. Як і будь-який інший патерн, він має свої обмеження та вимоги, які слід враховувати за його впровадження.

- Обмеження патерну:
- складність впровадження: впровадження цього патерну може вимагати значних зусиль і часу, особливо для великих і складних систем;
- підтримка та тестування: динамічна природа патерну може ускладнити процес тестування та відлагодження системи;
- непридатність для всіх сценаріїв: патерн може бути неефективним і громіздким для систем, де зміни умов і контексту відбуваються зрідка або де переважає стабільне середовище.

Патерн "Manager-Dispatcher" закладає основу для побудови систем, орієнтованих на події, що змушує архітекторів приділяти значну увагу можливим подіям та реакціям на них. Це сприяє більш гнучкому та масштабованому підходу до проєктування, де кожен модуль може адаптуватися до змін у реальному часі.

Однак важливо пам'ятати, що цей патерн не розв'язує всіх проблем. Він вимагає ретельного планування та розуміння специфіки системи, в якій він буде використовуватися. Архітектори та розробники повинні враховувати можливі події та відповідні реакції на них, що може збільшити складність початкового етапу розроблення.

У цій статті ми представили патерн "Manager-Dispatcher", який поєднує властивості патернів "Publish-Subscribe" і "Strategy" для забезпечення адаптивності поведінки програмних систем. Наш підхід дозволяє динамічно змінювати

стратегії функціонування програмного модуля у відповідь на змінювані умови середовища, забезпечуючи гнучкість та адаптивність поведінки системи.

Ми також розглянули кілька гіпотетичних сценаріїв застосування патерну разом із прикладом використання патерну; наведені результати демонструють потенціал патерну для підвищення гнучкості різних типів систем. Застосування цього патерну сприяє побудові модульних, орієнтованих на події систем, що змушує архітекторів приділяти значну увагу можливим подіям і реакціям на них.

Подальші дослідження можуть зосередитися на покращенні патерну, а також на розробленні інструментів і методологій для полегшення його впровадження та тестування.

Патерн "Manager-Dispatcher" пропонує перспективний підхід до проєктування адаптивних систем, що можуть забезпечити гнучкість у динамічних умовах сучасного програмного забезпечення.

Внесок авторів: Олексій Бичков – концептуалізація, написання – перегляд і редагування; Микола Мороз – аналіз джерел, підготовка огляду літератури, проєктування патерну, написання – оригінальна чернетка.

Список використаних джерел

Мороз, М. (2024). Приклад використання патерну "Manager-Dispatcher". https://github.com/mr-holodok/director-dispatcher-pattern-example

Bruns, R., & Dunkel, J. (2013). Towards pattern-based architectures for event processing systems. In R. Buyya, N. Horspool, A. Wellings (Eds.), Software: *Practice and Experience*, 44(11), 1395–1416. https://doi.org/10.1002/spe.2204

Buschmann, F., Sommerlad, P., Stal, M., Rohnert, H., & Meunier, R. (1996). *Pattern-Oriented software architecture volume 1. A system of patterns.* Wiley.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design patterns: Elements of reusable object oriented software.* Addison-Wesley.

Mannava, V., & Ramesh, T. (2011). A Novel Event Based Autonomic Design Pattern for Management of Webservices. In D. C. Wyld, M. Wozniak, N. Chaki, N. Meghanathan, D. Nagamalai (Eds.) *Advances in Computing and Information Technology*. *Communications in Computer and Information Science*. 198 (pp. 142–151). Springer. https://doi.org/10.1007/978-3-642-22555-0_16

Mannava, V., & Ramesh, T. (2012). A novel adaptive re-configuration compliance design pattern for autonomic computing systems. In E. P. Sumesh (Ed.). *Procedia Engineering*, 30, 1129–1137. https://doi.org/10.1016/j.proeng.2012.01.972

Ramirez, A. J. (2008). *Design patterns for developing dynamically adaptive systems* [Master's thesis]. Michigan State University. https://doi.org/doi:10.25335/ttfn-qx40

Ramirez, A. J., & Cheng, B. H. C. (2010). Design patterns for developing dynamically adaptive systems. *In the 2010 ICSE workshop on Software Engineering for Adaptive and Self-Managing Systems* (pp. 49–58). ACM Press. https://doi.org/10.1145/1808984.1808990

References

Bruns, R., & Dunkel, J. (2013). Towards pattern-based architectures for event processing systems. In R. Buyya, N. Horspool, A. Wellings (Eds.), Software: *Practice and Experience*, 44(11), 1395–1416. https://doi.org/10.1002/spe.2204

Buschmann, F., Sommerlad, P., Stal, M., Rohnert, H., & Meunier, R. (1996). *Pattern-Oriented software architecture volume 1. A system of patterns.* Wiley.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design patterns: Elements of reusable object oriented software.* Addison-Wesley.

Mannava, V., & Ramesh, T. (2011). A Novel Event Based Autonomic Design Pattern for Management of Webservices. In D. C. Wyld, M. Wozniak, N. Chaki, N. Meghanathan, D. Nagamalai (Eds.) *Advances in Computing and Information Technology*. *Communications in Computer and Information Science*. 198 (pp. 142–151). Springer. https://doi.org/10.1007/978-3-642-22555-0_16

Mannava, V., & Ramesh, T. (2012). A novel adaptive re-configuration compliance design pattern for autonomic computing systems. In E. P. Sumesh (Ed.). *Procedia Engineering*, 30, 1129–1137. https://doi.org/10.1016/j.proeng.2012.01.972

Moroz, M. (2024). An example of using the "Manager-Dispatcher" pattern [in Ukrainian]. https://github.com/mr-holodok/director-dispatcher-pattern-example

Ramirez, A. J. (2008). *Design patterns for developing dynamically adaptive systems* [Master's thesis]. Michigan State University. https://doi.org/doi:10.25335/ttfn-qx40

Ramirez, A. J., & Cheng, B. H. C. (2010). Design patterns for developing dynamically adaptive systems. In *the 2010 ICSE workshop on Software Engineering for Adaptive and Self-Managing Systems* (pp. 49–58). ACM Press. https://doi.org/10.1145/1808984.1808990

Отримано редакцією журналу / Received: 22.09.24
Прорецензовано / Revised: 01.10.24
Схвалено до друку / Accepted: 07.11.24

Oleksii BYCHKOV, DSc (Engin.), Prof.
ORCID ID: 0000-0002-9378-9535
e-mail: oleksiibychkov@knu.ua
Taras Shevchenko National University of Kyiv, Kyiv, Ukraine

Mykola MOROZ, PhD Student
ORCID ID: 0000-0001-6953-683X
e-mail: mukolamoroz@knu.ua
Taras Shevchenko National University of Kyiv, Kyiv, Ukraine

"THE MANAGER-DISPATCHER": A DESIGN PATTERN FOR ENSURING ADAPTIVE BEHAVIOR OF EVENT-DRIVEN SOFTWARE SYSTEMS

Background. ***Adaptive behavior in modern software systems is becoming a key factor in their successful operation under external and internal destabilizing influences. Programs that work with critical data or perform essential operations must ensure continuity of service, despite failures, attacks, or errors. Various approaches are proposed to achieve this goal, one of which is the application of design patterns that ensure system reliability and adaptability. This paper presents the "Manager-Dispatcher" pattern, which combines the features of the "Publish-Subscribe" and "Strategy" patterns to enable adaptive behavior in software systems through event processing.***

Methods. ***The development of the "Manager-Dispatcher" pattern was based on modular design and dynamic event processing methods. The pattern provides an automatic strategy selection for module operation based on events occurring within the system. A theoretical analysis of existing approaches to adaptive behavior in systems was conducted, leading to the creation of a new pattern that enables dynamic strategy changes in modules in response to environmental changes determined by system events. Several hypothetical application scenarios were considered to illustrate the pattern's functionality, and an example software system utilizing the pattern was developed and described.***

Results. ***The developed "Manager-Dispatcher" pattern allows software modules to automatically adapt their operational strategies based on system events. Key advantages of the pattern include modularity, extensibility, and adaptive behavior. The pattern may be particularly useful in embedded systems, real-time systems, and interactive interfaces, where fast and flexible responses to events are essential.***

Conclusions. ***The "Manager-Dispatcher" pattern offers a promising approach to the design of event-driven adaptive software systems. With its ability to dynamically change operational strategies, the pattern ensures a high degree of flexibility in dynamic environments. Future research will focus on improving the pattern and developing tools to facilitate its implementation and testing. This proposed approach supports the development of modular and adaptive systems capable of maintaining stable operation even under complex conditions.***

Keywords: ***design patterns, adaptive behavior, event processing, autonomous systems.***

Автори заявляють про відсутність конфлікту інтересів. Спонсори не брали участі в розробленні дослідження; у зборі, аналізі чи інтерпретації даних; у написанні рукопису; в рішенні про публікацію результатів.

The authors declare no conflicts of interest. The funders had no role in the design of the study; in the collection, analyses or interpretation of data; in the writing of the manuscript; in the decision to publish the results.

ISSN 2788-6603