С. Г. Московко

ПРОЕКТУВАННЯ РОЗПОДІЛЕНИХ СИСТЕМ УПРАВЛІННЯ НА ОСНОВІ ПРЕДМЕТНО-ОРІЄНТОВАНОГО ПРОЕКТУВАННЯ

Вінницький національний технічний університет

У роботі досліджується застосування принципів предметно-орієнтованого проектування (DDD) при проектуванні розподілених систем управління. Показано, як DDD сприяє чіткому розподілу системи на контексти, підвищуючи модульність і гнучкість. Обговорюються методи інтеграції доменних моделей у розподілену архітектуру та переваги цього підходу для масштабованості й надійності систем управління.

Ключові слова: предметно-орієнтоване проектування, обмежені контексти, системи управління, розподілена архітектура.

Вступ

У сучасному світі розподілені системи управління стали невід'ємною частиною багатьох галузей, від промисловості до інформаційних технологій. Зі збільшенням складності та масштабів таких систем виникає необхідність у нових підходах до їхнього проектування та розробки. Традиційні методи часто не справляються з викликами, пов'язаними з модульністю, масштабованістю та підтримкою складних систем.
Предметно-орієнтоване проектування (DDD) пропонує підхід, який фокусується на глибокому розумінні домену та побудові програмних систем навколо бізнес-логіки. Використання DDD дозволяє розділити складну систему на окремі контексти з чітко визначеними межами (обмежені контексти), що сприяє кращій модульності та спрощує інтеграцію компонентів [1,2].
Метою роботи є дослідження застосування принципів DDD при проектуванні розподілених систем управління. Ми розглянемо методи визначення меж контекстів, побудови доменних моделей та їх інтеграції в розподілену архітектуру. Також проаналізуємо переваги та виклики використання DDD у цьому контексті, включаючи підвищення масштабованості, надійності та спрощення підтримки систем управління.

Результати дослідження

Визначення доменів та субдоменів. Першим кроком нашого дослідження було глибоке розуміння предметної області системи управління. Це включало детальний аналіз бізнес-процесів, вимог користувачів та існуючих системних обмежень. За допомогою експертів предметної області ми ідентифікували основні домени та субдомени, що відображають структуру та функціональність системи.
Визначено такі ключові субдомени:

1. **Управління виробництвом**: відповідає за планування, контроль і оптимізацію виробничих процесів.
2. **Логістика та постачання**: охоплює управління ланцюгами постачання, складування та розподіл ресурсів.
3. **Контроль якості**: забезпечує моніторинг якості продукції та процесів, включаючи тестування та сертифікацію.
4. **Обслуговування та технічна підтримка**: відповідає за підтримку обладнання та інфраструктури, а також за обслуговування клієнтів.
5. **Фінансовий менеджмент**: включає облік витрат, бюджетування та фінансовий аналіз.

Це розбиття дозволило чітко окреслити межі відповідальності кожного субдомену та забезпечило основу для подальшого моделювання.

Встановлення меж контекстів. Після визначення субдоменів ми встановили межі контекстів для кожного з них. Межі контекстів визначають області, в яких певні терміни, моделі та процеси мають однозначне значення. Це критично важливо для уникнення неоднозначностей та конфліктів між різними частинами системи.

Для кожного субдомену було створено окремий контекст:

1. **Контекст "Управління виробництвом"**: зосереджується на процесах виробництва, плануванні ресурсів і розкладів.
2. **Контекст "Логістика та постачання"**: охоплює управління запасами, постачальниками та транспортом.
3. **Контекст "Контроль якості"**: включає процеси перевірки якості, стандарти та протоколи тестування.
4. **Контекст "Обслуговування та технічна підтримка"**: зосереджується на підтримці обладнання та взаємодії з клієнтами.
5. **Контекст "Фінансовий менеджмент**": охоплює фінансові операції, звітність та аналітику.

Встановлення чітких меж контекстів дозволило ізолювати доменні моделі та забезпечило незалежний розвиток кожного компонента системи.

Інтеграція між контекстами. Одним з найважливіших аспектів проєктування розподілених систем є забезпечення ефективної інтеграції між різними контекстами. У нашому дослідженні ми застосували кілька підходів для досягнення цієї мети

1. **Доменні події**: використовувалися для асинхронної комунікації між контекстами. Наприклад, коли в контексті "Управління виробництвом" завершується виробничий цикл, генерується подія, яка сповіщає контекст "Контроль якості" про необхідність перевірки.
2. **Анти-корупційний шар (Anticorruption Layer)**: введений для адаптації та перетворення моделей даних між контекстами, запобігаючи проникненню деталей однієї доменної моделі в іншу.
3. **API Gateway**: централізований шлюз для управління запитами між сервісами, що забезпечує безпеку та контроль доступу.

Ці методи дозволили зменшити зв'язність між контекстами та підвищити гнучкість системи.

Розробка розподіленої архітектури. На основі встановлених меж контекстів була спроектована розподілена архітектура системи з використанням мікросервісного підходу. Кожен контекст був реалізований як окремий мікросервіс, що має власну базу даних та незалежно розгортається.

Ключові аспекти архітектури:

1. **Незалежність сервісів**: кожен мікросервіс може бути розроблений, розгорнутий та масштабований незалежно від інших.
2. **Використання контейнеризації**: для спрощення розгортання та управління сервісами використовувалися технології Docker та Kubernetes.
3. **Моніторинг та логування**: впроваджені централізовані системи для відстеження стану сервісів та аналізу журналів подій.

Ця архітектура підвищила стійкість системи до збоїв та спростила її масштабування.

Аналіз переваг застосування DDD. Застосування принципів DDD при розробці розподіленої системи управління призвело до наступних переваг:

1. **Покращена модульність**: чітке розділення системи на контексти зменшило складність та підвищило зрозумілість коду.
2. **Гнучкість розвитку**: можливість незалежного оновлення та розширення функціональності кожного сервісу.
3. **Покращена масштабованість**: сервіси можуть масштабуватися окремо, залежно від навантаження.
4. **Спрощена підтримка**: ізоляція проблем в межах окремих сервісів спростила виявлення та виправлення помилок.

Виклики та обмеження. Під час впровадження DDD у розподілену систему управління ми зіткнулися з рядом викликів:

1. **Складність початкового моделювання**: глибоке розуміння домену вимагало значного часу та залучення експертів.
2. **Інтеграційні складнощі:** забезпечення надійної та ефективної комунікації між сервісами потребувало додаткових зусиль та використання спеціалізованих інструментів.
3. **Управління транзакціями**: підтримка цілісності даних між сервісами потребувала впровадження складніших механізмів, таких як саги або двофазне підтвердження.

Незважаючи на ці виклики, загальний вплив впровадження DDD був позитивним, і переваги перевищили витрати.

Висновки

У цій статті досліджено застосування принципів Domain-Driven Design (DDD) при проєктуванні розподілених систем управління. Використання DDD дозволило чітко визначити домени та субдомени, встановити межі контекстів і розробити модульну розподілену архітектуру. Це сприяло підвищенню модульності, гнучкості та масштабованості системи, а також спростило її підтримку та розвиток. Незважаючи на виклики, такі як складність початкового моделювання та високі вимоги до кваліфікації команди, переваги застосування DDD перевищили витрати, роблячи цей підхід ефективним для створення стійких та адаптивних систем управління.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

[1] E. Evans, Domain-Driven Design: Tackling Complexity in the Heart of Software. 2003, 320 p., ISBN: 9780321125217.
[2] V. Vernon, Implementing Domain-Driven Design. Addison-Wesley Professional, 2013, 656 p., ISBN: 978-0321834577.
[3] S. Kapferer and O. Zimmermann, "Domain-specific language and tools for strategic domain-driven design, context mapping and bounded context modeling," in Proceedings of the 8th International Conference on Modeling and Software Development - Volume 1, SciTePress, 2020, pp. 299-306.

Московко Сергій Геннадійович — факультет інтелектуальних інформаційних технологій та автоматизації, e-mail: smoskovko@icloud.com.
Вінницький національний технічний університет.

S. G. Moskovko

Design of distributed control systems based on domain-driven design

Vinnytsia National Technical University

The paper explores the application of Domain-Driven Design (DDD) principles in the design of distributed control systems. It demonstrates how DDD facilitates a clear separation of the system into bounded contexts, enhancing modularity and flexibility. Methods of integrating domain models into distributed architecture are discussed, along with the benefits of this approach for scalability and reliability of control systems.

Keywords: Domain-Driven Design, bounded contexts, control systems, distributed architecture

Moskovko Serhii G. — Department of intelligent information technologies and automation, e-mail: smoskovko@icloud.com.