чіпі. Пристрій досягає пропускної здатності 16 MIPS на 16 МГц і працює в діапазоні 2,7-5,5 вольт [4]

Таким чином, враховуючи вище зазначене, можна зробити висновок, що мікроконтролери є центром вбудованої системи і забезпечує виконання програм, які керують пристроєм, що у цілому сприяє продуктивності роботи плати.

Список використаних джерел та літератури

1. Мікроконтролер Arduino URL: https://bitkit.com.ua/shho-take-arduino.

2. О.В. Глухов, О.О. Кравчук, Є.В. Левченко Вивчення властивостей мікроконтролерів і електронних систем на базі платформи Ардуіно: навч. посібник. – Харків: ХНУРЕ, 2019. 192 с.

3. Atmel Corparation. URL: https://web.archive.org/web/20120125022835/http://atmel.com/dyn/products/product_card.asp?part_id=3632.

4. Atmel Corparation Microcontrollers. URL: https://web.archive.org/web/20120125020530/http://atmel.com/dyn/products/product_card.asp?part_id=3633.

Наконечна Оксана,
кандидат технічних наук,
доцент кафедри комп'ютерних наук та інформаційних технологій,
Житомирський державний університет імені Івана Франка,
м. Житомир, Україна
Гук Віталій,
кандидат технічних наук,
старший викладач кафедри програмного забезпечення
автоматизованих систем,
Черкаський національний університет імені Б. Хмельницького,
м. Черкаси, України

СУЧАСНІ ТЕХНОЛОГІЇ ТА ЗАСОБИ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Постановка проблеми. Сучасні засоби розробки програмного забезпечення та технології дозволяють створювати та розвивати надзвичайно складні, розподілені програмні системи, що взаємодіють з різними зовнішніми агентами. Складній програмній системі, як і будь-якій іншій системі, притаманні певні загальні закономірності, що вивчаються у загальній теорії систем [1]. Збільшення складності, численність підсистем з суперечливими цілями і велика кількість взаємозв'язків призводить до появи в галузі проектування, розробки та експлуатації програмних систем мета-системного переходу [2].

Це виявляється у появі на певному етапі розвитку системи, з появою нового рівня ієрархії. Він починає контролювати нижні рівні, визначаючи нові обмеження на множині їх допустимих станів, а процеси, агенти та дані, які до цього моменту були зовнішніми по відношенню до системи, стають об'єктами у складі системи вищого рівня складності (мета-системи). Це призводить до того,

що узгоджене управління параметрами раніше незв'язаних компонентів стає одним із важливих завдань (визначення обмежень та контроль за їх дотриманням), необхідних для досягнення мети мета-системи, що утворилася. При цьому мета мета-системи виражається через нові (emergent), більш абстрактні властивості, а досягнення цієї мети вимагає вирішення нових інтелектуальних завдань.

Для більшості прикладних задач найефективніша стратегія – використання найбільш абстрактного рівня в системі технологій та найбільш абстрактного рівня у системі проектування.

Мета роботи – розглянути найбільш відомі технології, інструментальні засоби та системи об'єктно-орієнтованого моделювання з погляду рівня абстрактності та основні засади модельно-орієнтованого підходу, що максимально сприяють якісній та ефективній інженерній діяльності.

Виклад основного матеріалу. В даний час лідируючим методом проектування та розробки програмного забезпечення є об'єктно-орієнтований підхід [3]. Свого часу розробка об'єктно-орієнтованого підходу до проектування та реалізації програмного забезпечення стала прикладом мета-системного переходу в програмній інженерії. Дані та прикладні алгоритми елементів програми, що моделюють поведінку різних сутностей предметної області розглядаються спільно в рамках одного поняття «об'єкт». Це дозволяє уніфікувати велику кількість допоміжних алгоритмів та абстракцій, вивівши їх з поля зору розробників прикладних алгоритмів на рівень системного ПЗ.

З використанням засобів та методів об’єктно-орієнтованого дизайну з’явилась можливість створювати більш складне програмне забезпечення, відповідно і виникли завдання вищого рівня абстракції, які потребують вирішення. До найважливіших завдань належать: забезпечення ефективної розподіленої роботи множини об'єктів; підтримка транзакцій; безпека об'єктно-орієнтованої системи; довго тривале збереження даних; сумісність нелокальних ефектів поведінки об'єктів у контексті складної розподіленої системи.

Ефективно використовувати розподілені об'єктно-орієнтовані технології, здійснювати підтримку та розвивати у кожному конкретному випадку можна на основі використання моніторів компонентної взаємодії (монітор компонентних транзакцій). Найбільш поширена практична реалізація серверних компонентів та моніторів компонентної взаємодії – EJB-технологія Enterprise Java Beans [4]. Серед реалізацій EJB-технологій, що активно використовуються, відзначимо вільно-поширюваний продукт WildFly, і комерційний продукт BEA WebLogic.

Технологія EJB визначає модель для проектування, реалізації, розгортання та управління життєвим циклом незалежних серверних компонентів на мові Java, які інкапсулюють дані та правила бізнес-логіки для локальних задач.

Технологія EJB підтримує принцип організації розподіленої системи з набору незалежно створених компонентів. Функціональність, з точки зору прикладного розробника представляється у формі параметрів мета-системи. Всі (або більшість) питань життєвого циклу, іменування компонентів, безпеки, довговічності, прозорого мережного доступу об'єктивно визначені правилами

взаємодії серверного EJB-компоненту та монітора компонентних взаємодій. Розробник може розглядати функціонування системи в термінах уявлень набагато вищого рівня абстракції та проводити налаштування EJB-компонентів шляхом зміни їх атрибутів у декларативному вигляді, здійснюючи програмування з урахуванням атрибутів (attribute-based programming). Це дозволяє ефективно зв'язувати систему з іншими системами (зовнішнім світом), розмірковувати про функції та можливості створеного ПЗ.

Технологія EJB створює новий рівень абстракції доступу до даних програми, що повинні тривалий час зберігатись. Цей рівень називається «керування довготривалим зберіганням» (Container-Managed Persistence, CMP) [4] і дозволяє розробнику використовувати поняття «постійних» об'єктів (persistent object), абстрагуючись від питань з'єднання з базами даних, формування SQL-запитів, визначення моментів доступу до баз тощо. Виконується проекція властивостей EJB-компоненти на модель бази даних, що використовується, проводиться декларативним способом у процесі розгортання на моніторі компонентної взаємодії. Тому той самий EJB-компонент у різних розподілених системах може представляти різні бази даних.

В результаті використання можливостей компонентного програмування та моніторів компонентної взаємодії програміст звільняється від необхідності детально вивчати специфіку реалізації таких важливих функцій, як життєвий цикл, сталість даних, транзакції та ін. Однак, користувач повинен розуміти усі деталі синтаксису та семантики параметрів об'єктивізованого процесу управління компонентами. Для здійснення процесу параметризації та полегшення створення артефактів (класи, інтерфейси, конфігураційні файли тощо), а також збірки компонентів розподіленої програми застосовуються спеціалізовані інструментальні засоби, наприклад: засіб автоматизації побудови ANT; додаток AndroMDA; Jenkins; Maven; Gradle.

Розглянемо основні принципи проектування ПЗ, а саме використання мета-технологій проектування, заснованих на ієрархії формальних моделей та мета-моделей предметної галузі та технологій програмування. Оскільки із зростанням популярності Інтернету та підвищенням важливості завдань з інтеграції різнорідних даних мета-моделювання стало важливішим напрямком програмної інженерії. Мета-моделі – основа автоматизованої інтеграції різнорідної інформації.

Моделі визначають структури даних, які використовуються в додатку. Так в об'єктно-орієнтованому програмуванні об'єкти в додатку - це екземплярами класів моделі. Тому моделі можна назвати мета-даними (інформація про властивості та структуру даних, що використовуються).

Проілюструвати взаємовідносини між різними рівнями мета-моделювання можна з урахуванням стандартів організації OMG (Object Management Group). Для моделювання та інтеграції складних об'єктно-орієнтованих програм OMG пропонує використовувати взаємозалежну ієрархію мета-моделей та моделей із чотирьох рівнів (Meta Object Facility, MOF). Приклад однієї з можливих реалізацій кожного із чотирьох рівнів ієрархії MOF представлено на рис. 1. Де

напівжирним шрифтом на кожному рівні ієрархії MOF виділено поняття, що вводяться на цьому рівні. У прикладі лише на рівні мета-моделі використовується мова для представлення простих, не пов'язаних між собою записів, тобто, визначається складовий тип Record (наприклад, записи про працівників). Вміст рівнів визначається так:

– рівень інформації користувача містить конкретні екземпляри записів з інформацією про певних співробітників;

– рівень моделі включає мета-дані, що описують внутрішню структуру типу даних Record з ім'ям Employee для зберігання інформації про співробітника. Цей тип даних має два поля із заданими іменами та типами;

– рівень мета-моделі визначає, що буде відноситись до типу даних Record і визначається через вміст мета-класу «Record». Він містить два мета-атрибути: перший визначає ім'я типу Record, а другий визначає атрибути. Так само особливий мета-клас визначає вміст елемента Field.

– рівень мета-мета-моделі жорстко заданий заздалегідь і визначає всі конструкції, що використовуються для визначення мета-моделей. У розглянутому прикладі, на цьому рівні визначаються поняття мета-клас, мета-атрибут тощо.

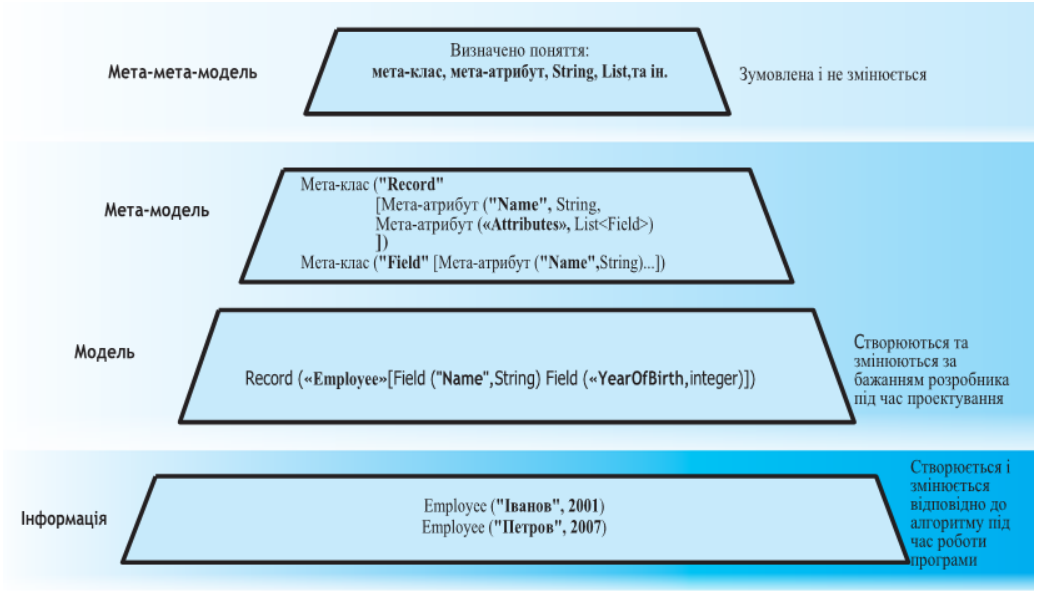

Рис. 1. Приклад конкретної реалізації ієрархії MOF

Підхід на основі мета-моделювання надає розробнику розширений механізм для опису всіх деталей архітектури програмного забезпечення, який дозволяє повторно використовувати існуючі можливості (інтерфейс користувача, управління життєвим циклом об'єктів тощо) і забезпечує модульну інтеграцію з зовнішніми системами і структурами даних.

З використанням механізму мета-моделей інтеграція структур даних стає можливою за рахунок того, що опис цих структур доступний у програмному засобі у вигляді даних, з якими можна здійснювати типові операції (читання, модифікація, створення, видалення тощо).

Для створення мета-моделей і моделей використовується мова об'єктно-орієнтованого моделювання – мову UML. У цьому випадку мета-модель

визначає модель мови UML на найвищому рівні абстракції і є найбільш компактним її описом – усі основні поняття мови UML (клас, атрибут, операція, компонент, асоціація тощо), їх атрибути та взаємозв'язки формально визначаються на рівні мета-моделі. Повна мета-модель мови UML має складну структуру і включає близько 90 мета-класів, більше 100 мета-асоціацій, організованих у три логічні пакети: основні елементи, елементи поведінки та загальні механізми. Будь-яка UML-модель ПЗ є екземпляром мета-моделі. Це означає: для побудови моделі можуть використовуватися лише поняття, визначені мета-моделями, які розробник конкретизує певним чином.

Таким чином, можливості мови UML практично повністю задовольняють вимогам мета-моделювання, дозволяє створювати узгоджені ієрархії моделей та мета-моделей, а також проводити їх трансформації в рамках єдиної мета-технології. Приклад такої мета-технології – підхід Model-Driven Architecture (MDA), який також розвивається під егідою OMG. Цей підходу дозволяє відокремити специфікацію функціональності ПЗ та опис концепцій предметної області від специфікації альтернативних методів реалізації з використанням певної програмної технології або платформи.

При розробці моделі використовується строго специфікований набір елементів та типів їх взаємозв'язків, що задає певну мета-модель. Таким чином, в моделі чітко визначено її форму (синтаксис), значення (семантика), а також правила аналізу та логічного виводу істинності та логічної цілісності її складових. Це дозволяє спільно використовувати різні засоби проектування, моделювання та проведення імітаційних експериментів за побудованими моделями, використовувати єдині поняття різними групами розробників.

Отримані платформо-незалежні моделі потім використовуються як вихідні дані для автоматичної генерації різних конкретних сутностей реалізації (артефактів). Звичайно, повністю уникнути ручного програмування не вдається, але завдяки потужній інструментальній підтримці (захист змін користувача при перегенерації артифактів, стандартні текстові шаблони, макроси, тощо) обсяг ручного програмування значно знижується.

Сукупність створених автоматично або вручну артефактів становить єдину модель (або єдину групу моделей), яка описує систему вже у зв'язку з конкретними обраними для реалізації технологічними рішеннями. Таку модель називають платформо-залежною моделлю (PSM – Platform-Specific Model).

Важлива особливість MDA-підходу є великим коефіцієнтом повторного використання компонентів моделі. Одні й ті ж самі моделі можуть бути використані для генерації різних артефактів, залежно від мови реалізації, конфігурації, програмних технологій. Така можливість з'являється завдяки тому, що в MDA-підході явно виділяється інтерфейс та вимоги до компоненту мапінгу, відповідального за генерацію певної PSM моделі абстрактної PIM-моделі. Такий компонент зазвичай називається MDA-картридж. Окремий MDA-картридж є модулем розширення інструментального засобу та вміє проводити генерацію артефактів для конкретного набору параметрів, мов та технологій (наприклад, бізнес-логіка: Java + EJB + WebLogic, GUI: Java + JSP + WebLogic, засоби

підтримки розробки, командні файли для ANT з використання відповідних бібліотек.

Для поширення принципів модельно-орієнтованого проектування та розробки крім однотипних принципів використання ієрархій моделей та мета-моделей необхідний єдиний текстовий формат представлення створених UML-моделей, щоб можна було поводитися з моделями, створеними у різних засобах проектування та моделювання. Хоча при використанні різних засобами розробки (Rational Rose, Together, MagicDraw) та системами, що підтримують генерацію артефактів (AndroMDA) виявлено велику відмінність у існуючих версіях XMI. Це призводить до того, що на практиці перенесення моделей з одного засобу до іншого на рівні XMI-документів вкрай обмежене.

Висновок. Таким чином, реалізація внутрішніх рівнів архітектури програмних засобів та системних алгоритмів здійснюється на основі використання моніторів компонентної взаємодії, розподілених багатоагентних системах та модельно-орієнтованих принципах проектування та розробки програмного забезпечення.

Сучасні тенденції розвитку техніки та апаратної архітектури спонукають до постійних досліджень в галузі підвищення рівня абстракції методів та технологій проектування та реалізації складних програмних засобів. Широке застосування модельно-орієнтованих методів розробки програмних систем змушує постійно шукати нові принципи для уточнення вимог, побудови архітектури, реалізації, тестування та інтеграції програм.

Список використаних джерел та літератури

1. Прокопенко Т.О. Теорія систем і системний аналіз: навч. посіб. [Електронний ресурс] / Т. О. Прокопенко; М-во освіти і науки України, Черкас. держ. технол. Ун-т. Черкаси: ЧДТУ, 2019. 139 с.

2. Турчин В.Ф. Феномен науки: Кібернетичний підхід до еволюції. М.: ЕТС, 2000. URL: https://www.rulit.me/books/fenomen-nauki-kiberneticheskij-podhod-k-evolyucii-read-206999-1.html#.

3. Табунщик Г. В., Каплієнко Т. І., Петрова О. А. Проектування та моделювання програмного забезпечення сучасних інформаційних систем. Запоріжжя: Дике Поле, 2016. 250 с. URL: http://eir.zntu.edu.ua/bitstream/123456789/1824/1/Tabunshchik_Software_Design.pdf.

4. Специфікація JavaBean. URL: https://docs.oracle.com/javase/6/docs/api/java/beans/package-summary.html

5. AndroMDA – розробка програмного забезпечення. URL: https://tecnologiandroid.com/que-es-andromda-desarrollo-de-software.