Архітектура. Логічна архітектура і діаграми пакетів

Архітектура. Логічна архітектура і діаграми пакетів

Зміст

Опис архітектури

Процес проектування архітектури ПЗ включає в себе збір вимог клієнтів, їх аналіз та створення проекту для компонента ПЗ у відповідність до вимог.

Успішна розробка ПЗ повинна:

забезпечувати баланс неминучих компромісів внаслідок суперечок вимог;

відповідати принципам проектування та рекомендованим методам, вироблених з часом;

доповнювати сучасне обладнання, мережі і системи управління.

Надійна архітектура ПЗ вимагає значного досвіду в теоретичних і практичних питаннях, а також уяви, необхідної для перетворення бізнес-сценаріїв і вимог в надійні і практичні робочі проекти.

Опис архітектури

Архітектура ПЗ включає в себе визначення структурованого рішення, у відповідності до всіх технічних і робочих вимог, одночасно оптимізуючи загальні атрибути якості, такі як продуктивність, безпеку і керованість.
Сюди входить серія рішень, заснованих на широкому діапазоні факторів, і кожне з цих рішень може значно впливати на якість, продуктивність, підтримуваний і загальний успіх ПЗ.
Сучасне ПЗ рідко буває автономним. Як мінімум, в більшості випадків воно буде взаємодіяти з джерелом даних.
Зазвичай сучасне ПЗ також має взаємодіяти з іншими службами і мережевими функція для виконання перевірки автентичності, отримання та публікації інформації та надання інтегрованих середовищ для роботи користувачам.
Без відповідної архітектури може бути складно, якщо взагалі можливо, здійснити розгортання, експлуатацію, обслуговування і успішну інтеграцію з іншими системами; крім того, вимоги користувачів не будуть дотримані.

Опис архітектури

Архітектуру ПЗ можна розглядати як зіставлення між метою компонента ПЗ і відомостями про реалізацію в коді.
Правильне розуміння архітектури забезпечить оптимальний баланс вимог і результатів.
ПЗ з добре продуманою архітектурою буде виконувати зазначені завдання з параметрами вихідних вимог, одночасно забезпечуючи максимально високу продуктивність, безпеку, надійність і багато інших чинників.

На найвищому рівні проект архітектури повинен:
надавати структуру системи, але приховувати деталі реалізації;
охоплювати всі випадки застосування і сценарії;
намагатися враховувати вимоги всіх зацікавлених осіб;
задовольняти настільки, наскільки можливо, всім функціональним вимогам і вимогам до якості.

Ознайомлення з представленнями архітектури

Архітектура ПЗ починається з набору вимог.
Вони можуть бути виражені у формі діаграм, блок-схем процесу, моделей або документованих списків завдань експлуатації, які має виконувати ПЗ.
Зазвичай клієнт або партнер також висловлює менш точні вимоги, такі як зовнішній вигляд, або спосіб роботи певних інтерфейсів користувача для завдань, що часто зустрічаються

Вимоги також повинні включати в себе:
інформацію про існуюче ПЗ, системи, обладнання та мережі, з якими буде взаємодіяти нове ПЗ;
та інші фактори, такі як план розгортання і обслуговування, і, звичайно ж, доступний бюджет проекту.

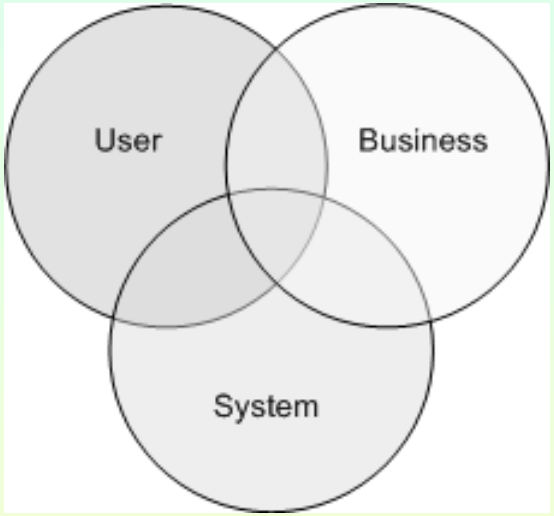
Ознайомлення з представленнями архітектури

Розробник архітектури ПЗ повинен враховувати потреби клієнта.
Однак загальний термін «клієнт» зазвичай складається з трьох суперечних областей відповідальності:
бізнес-вимоги,
вимоги користувача і
системні вимоги.

Бізнес-вимоги зазвичай визначають діапазон таких факторів, як
бізнес-процеси,
фактори продуктивності (такі як безпека, надійність і пропускна здатність), а також
бюджетні обмеження і обмеження на витрати.

Ознайомлення з представленнями архітектури

Вимоги користувача включають в себе
дизайн інтерфейсу,
виробничі можливості і
простоту використання ПЗ.



Системні вимоги:
можливості та обмеження обладнання,
мережі та
середовища виконання.

Розробнику необхідно створити архітектуру, яка підходить для областей, що перекриваються.

Ознайомлення з представленнями архітектури

Архітекторам часто доводиться відповідати на такі питання:
«Як користувачі будуть працювати з додатком?»;
«Як додаток буде розгорнуто у виробничому середовищі і управлятися?»;
«Які вимоги атрибутів якості для програми, такі як безпека, продуктивність, паралелізм, інтернаціоналізація і конфігурація?»;
«Яка повинна бути архітектура програми для забезпечення гнучкості і зручності в обслуговуванні з плином часу?» І
«Які архітектурні тенденції можуть впливати на додаток зараз і після його розгортання?».

Ознайомлення з представленнями архітектури

Останнє питання одночасно важливе і цікаве.
Хороша архітектура ПЗ не тільки відповідає поточним вимогам клієнтів, але й буде відповідати таким вимогам в осяжному майбутньому.
Це впливає на рішення, що приймаються розробником архітектури щодо обладнання, компонентів, платформ, середовищ виконання, програмних систем управління і безлічі інших функцій, вбудованих в ПЗ або з якими воно має інтегруватися.

Як і більшість завдань в світі проектування та розробки ПЗ, розробка архітектури - це одночасно попереджувальний і ітеративний процес.

Багато початкових завдань, такі як аналіз вимог, технічне дослідження і визначення цілей, зазвичай виконуються на початку процесу.

Ознайомлення з представленнями архітектури

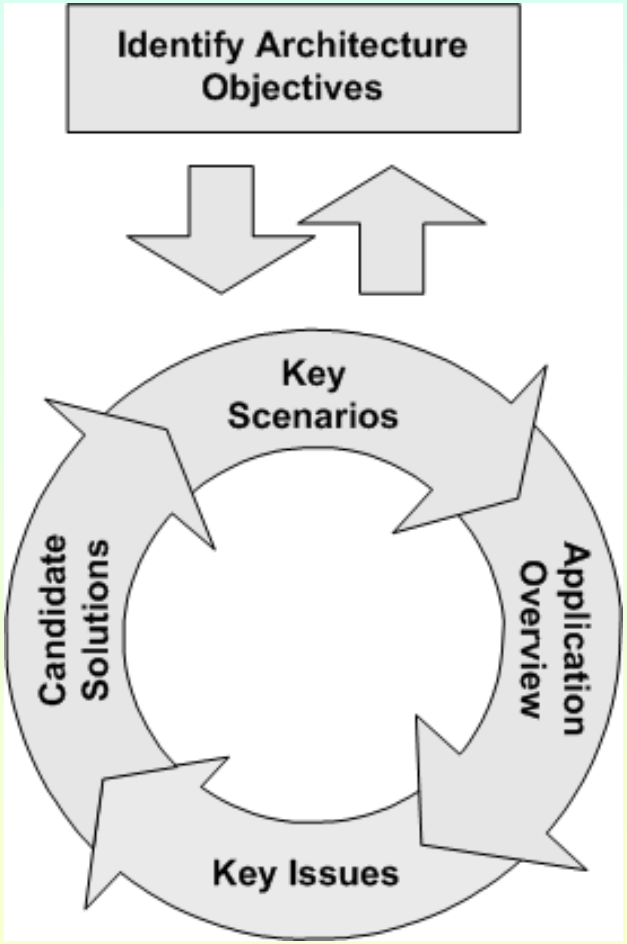
Наступний етап - визначення ключових сценаріїв для архітектури. Це основні вимоги, яким має відповідати ПЗ, та обмеження, в рамках яких воно має працювати. На підставі цієї інформації розробник архітектури може створити огляд програми. Цей огляд охоплює такі високорівневі відомості, як тип програми (для Інтернету, телефонів, настільне або хмарне), архітектура розгортання (зазвичай багаторівнева архітектура, компоненти якої зв'язуються через кордони обладнання та мережі), відповідні застосовні стилі архітектури (наприклад, n-рівнева, клієнт сервер або сервіс-орієнтована) і технології реалізації, найкраще підходять для сценарію.

Ознайомлення з представленнями архітектури

Після цього розробник може приступити до створення кандидатів архітектури, що задовольняють високорівневим і найбільш важливим вимогам, визначеним раніше.

Після цього проект архітектури перевіряється і тестується в ключових сценаріях, часто в поєднанні з відгуками клієнтів і пробними або тестовими версіями, для забезпечення отримання оптимального результату.

Це малоймовірно при першій ітерації, але при повторенні циклу буде досягнуто відповідність проекту вимогам і ключовим сценаріями.



Ознайомлення з представленнями архітектури

У міру деталізації проекту і визначення окремих завдань і компонентів архітектор може уточнити і додати подробиці на кожному етапі. Наприклад, після визначення архітектурного стилю і підходу до розгортання архітектор може прийняти рішення про зв'язок рівнів і компонентів.

Сюди може входити вибір протоколу на основі існуючих і майбутніх вимог, а також прийняття до уваги оглядів нових технологій і можливостей, визначених у майбутніх стандартах.

Ознайомлення з представленнями архітектури

Остаточний продукт роботи архітектора - це зазвичай набір схем, моделей і документів, що визначають застосування (додаток) з декількох точок зору, щоб при об'єднанні вони могли надати розробникам, групам тестування, адміністраторам і управлінню всю необхідну інформацію для реалізації проекту.

Ця інформація буде описувати
структуру і розміщення компонентів і рівнів застосування (додатку);
спосіб обробки горизонтального перетину ієрархії, наприклад, протоколювання та перевірки;
плани тестування і розгортання; та
документацію для розробників, адміністраторів і персоналу служби підтримки.

Ознайомлення з представленнями архітектури

В остаточному проекті також можуть бути вказані атрибути якості, яким має відповідати додаток.
Це результат прийнятих рішень і компромісів розробника архітектури за консультаціями з клієнтом.

До них відносяться
визначення вимог безпеки та план реалізації безпеки,
необхідна масштабованість і продуктивність при розгортанні на цільовій платформі,
способи реалізації можливості обслуговування і розширюваності і функції забезпечення взаємодії з іншими системами.

Логічна архітектура

Логічна архітектура описує систему в термінах її принципової організації у вигляді рівнів, пакетів (чи просторів імен), програмних класів і підсистем. Вона називається логічною, оскільки не визначає способи розгортання цих елементів в різних операційних системах чи на фізичних комп'ютерах в мережі (це відноситься до архітектури розгортання).

Типічні рівні ОО системи:

інтерфейс користувача;

рівень застосування чи об'єктів предметної галузі;

рівень технічних служб. Об'єкти і підсистеми, які забезпечують підтримку взаємодії з базою даних чи журналами реєстрації помилок. Ці служби зазвичай незалежні від застосування, і їх можна повторно використовувати в декількох системах.

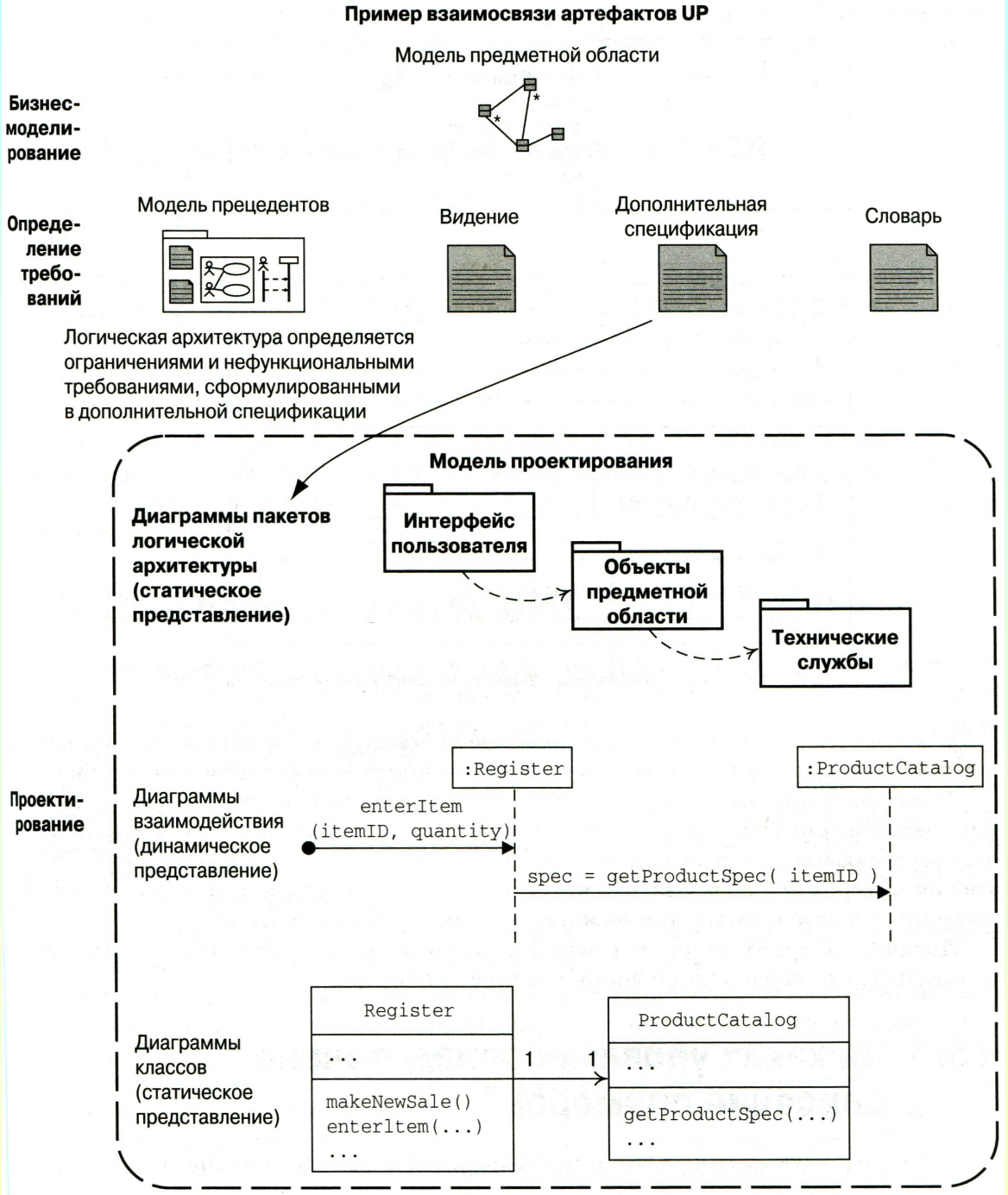


Рис. 13.1. Пример взаимосвязи артефактов UP

Рис. 13.1. стр 229.

Приклад рівнів

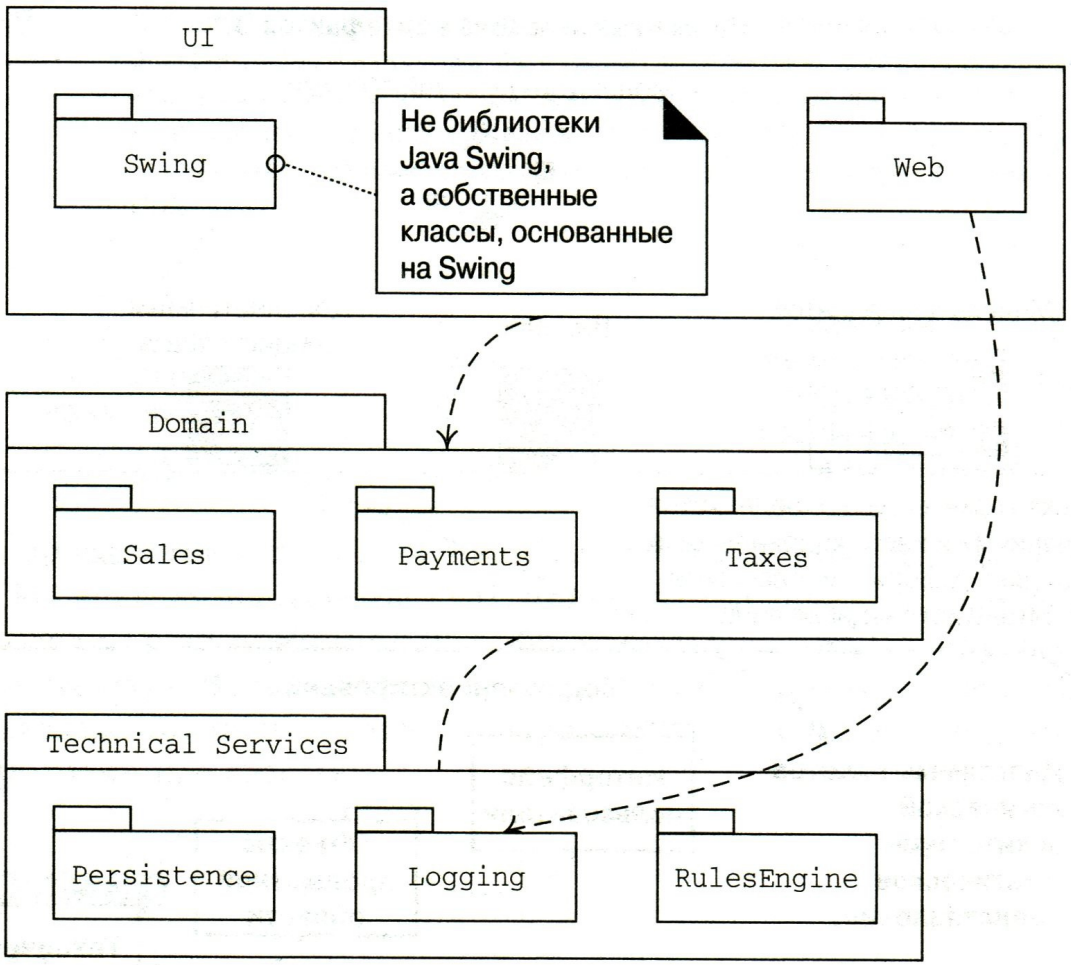


Рис. 13.2. Представление уровней на диаграмме пакетов UML

Рис. 13.2 стр. 230

Логічна архітектура і діаграми пакетів

Зміст

Рівні

В жорстко структурованої багаторівневої архітектурі об’єкти кожного рівня можуть викликати служби тільки рівня, який знаходиться безпосередньо під ними.
В інформаційних системах зазвичай використовується гнучка багаторівнева архітектура, при якій об’єкти можуть звертатися до декількох рівнів, які розташовані нижче.

Наприклад: об’єкти рівня інтерфейсу користувача можуть звертатися до елементів нижнього рівня технічних служб.

Логічна архітектура і діаграми пакетів

Зміст

Програмна архітектура системи

Архітектура – це набір важливих рішень, які стосуються організації програмної системи, вибору структурних елементів і їх інтерфейсів, які зачіпають поведінку і взаємодію цих елементів, їх групування до більш крупних підсистем і архітектурний стиль застосування.

Незалежно від визначення, архітектура системи включає крупномасштабні елементи – основні ідеї організації, шаблони, спосіб розподілення обов’язків, взаємодію і мотивацію поведінки системи і її основних підсистем.

Логічна архітектура і діаграми пакетів

Зміст

Діаграми пакетів

Діаграми пакетів UML використовуються для ілюстрації логічної архітектури застосування – рівней, підсистем, пакетів.
Кожний рівень можна представити у вигляді пакету.
Діаграма пакетів забезпечує один із способів групування елементів.
В одному пакеті UML можуть об'єднуватися різні елементи: класи, інші пакети, прецеденти і т.і.

Ім'я пакета розташовується на корінці, якщо на діаграмі відображається внутрішній вміст пакету, чи на самому позначені пакету.

Часто зображають залежності між пакетами, щоб розробникам був зрозумілим взаємозв'язок елементів системи. Для цього використовується лінія залежності UML, яка представляє собою пунктирну лінію зі стрілкою, яку направлено в бік незалежного пакету.
Пакет UML представляє собою простір імен а, отже, іноді використовують повну кваліфікацію імен для опису вкладеності пакетів.
Іноді вкладені пакети графічно розміщуються всередині загального пакету.

Приклади зображення вкладених пакетів

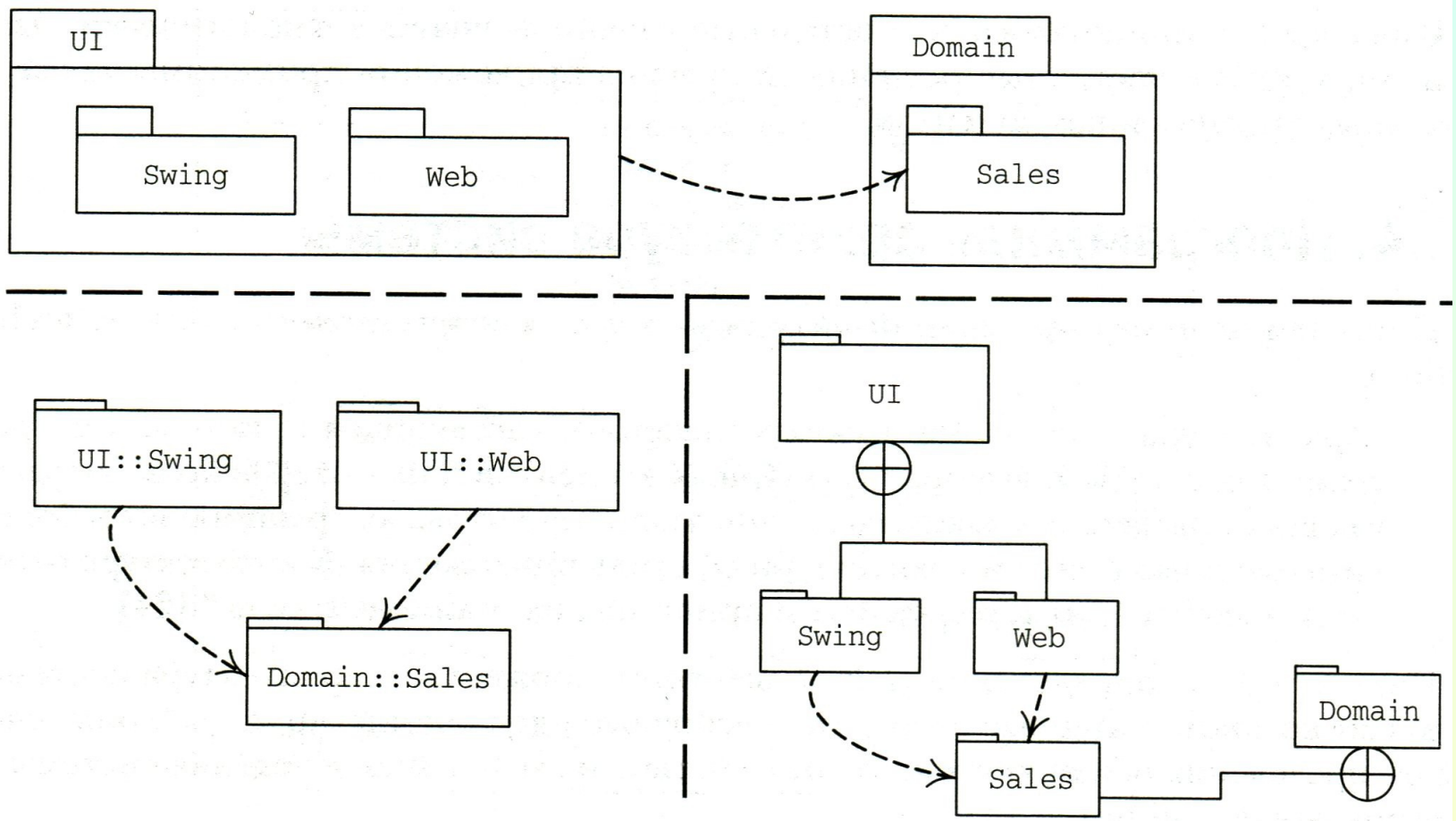


Рис. 13.3. Альтернативные подходы к изображению вложенных пакетов в UML

Рис. 13.3 стр. 232

Логічна архітектура і діаграми пакетів

Зміст

Проектування на основі шаблона Layers

Основні принципи шаблону Layers:

Організувати крупномасштабні структурні елементи системи в окремі рівні із взаємозв'язаними обов'язками таким чином, щоб на нижньому рівні розташовувались низькорівневі служби і служби загального призначення, а на більш високих рівнях – об'єкти рівня логіки застосування.

Взаємодія і зв'язування рівнів відбувається зверху до низу. Необхідно уникати зв'язування об'єктів знизу вгору.

Проектування на основі шаблона Layers

Використання шаблону Layers дозволяє вирішити проблеми:

Зміна вихідного коду тягне за собою коректування всіх елементів системи, оскільки всі частини системи тісно пов’язані одна з іншою.

Логіка застосування переплітається з інтерфейсом користувача, тому в застосуванні неможливо змінити інтерфейс чи принципи реалізації логіки застосування.

Загальні технічні служби тісно зв’язані з бізнес логікою застосування, тому їх не можна використовувати повторно, розповсюдити на інші системи чи змінити їх реалізацію.

Із-за високого ступеню зв’язування складно модифікувати функції застосування, масштабувати систему чи переходити на нові технології.

Проектування на основі шаблона Layers

Переваги використання шаблону Layers:

У цілому, цей шаблон забезпечує поділ різних аспектів, високорівневих служб від низькорівневих, спеціалізованих функцій до загальних. Це знижує рівень зв'язування і залежності в застосуванні, підвищує ступінь зачеплення, збільшує потенціал повторного використання і вносить додаткову ясність.

Складні елементи піддаються декомпозиції і підлягають інкапсуляції.

Деякі рівні замінюються новими реалізаціями. В загальному випадку це не можливо для низькорівневих технічних служб і базового рівня, проте цілком реально для рівней інтерфейсу, застосування і предметної галузі.

Нижні рівні містять функції, які використовуються повторно.

Деякі рівні (особливо рівні предметної галузі і технічних служб) можуть бути розподіленими.

Логічна сегментація забезпечує можливість роботи над застосуванням групи розробників.

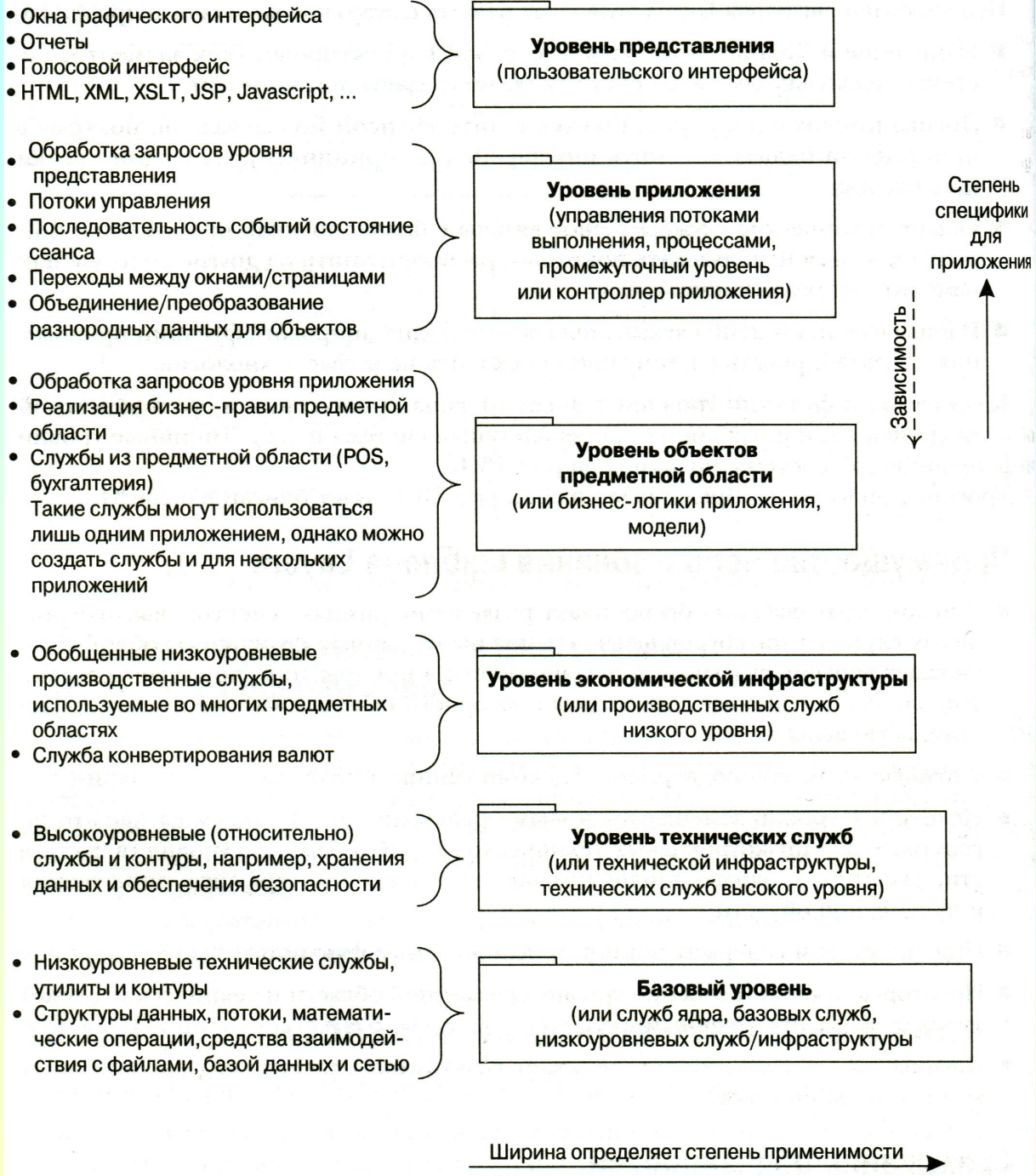


Рис. 13.4. Типичные уровни логической архитектуры информационной системы[1]

Рис.13.4 стр. 234

Взаємозв’язок рівня і моделі предметної галузі

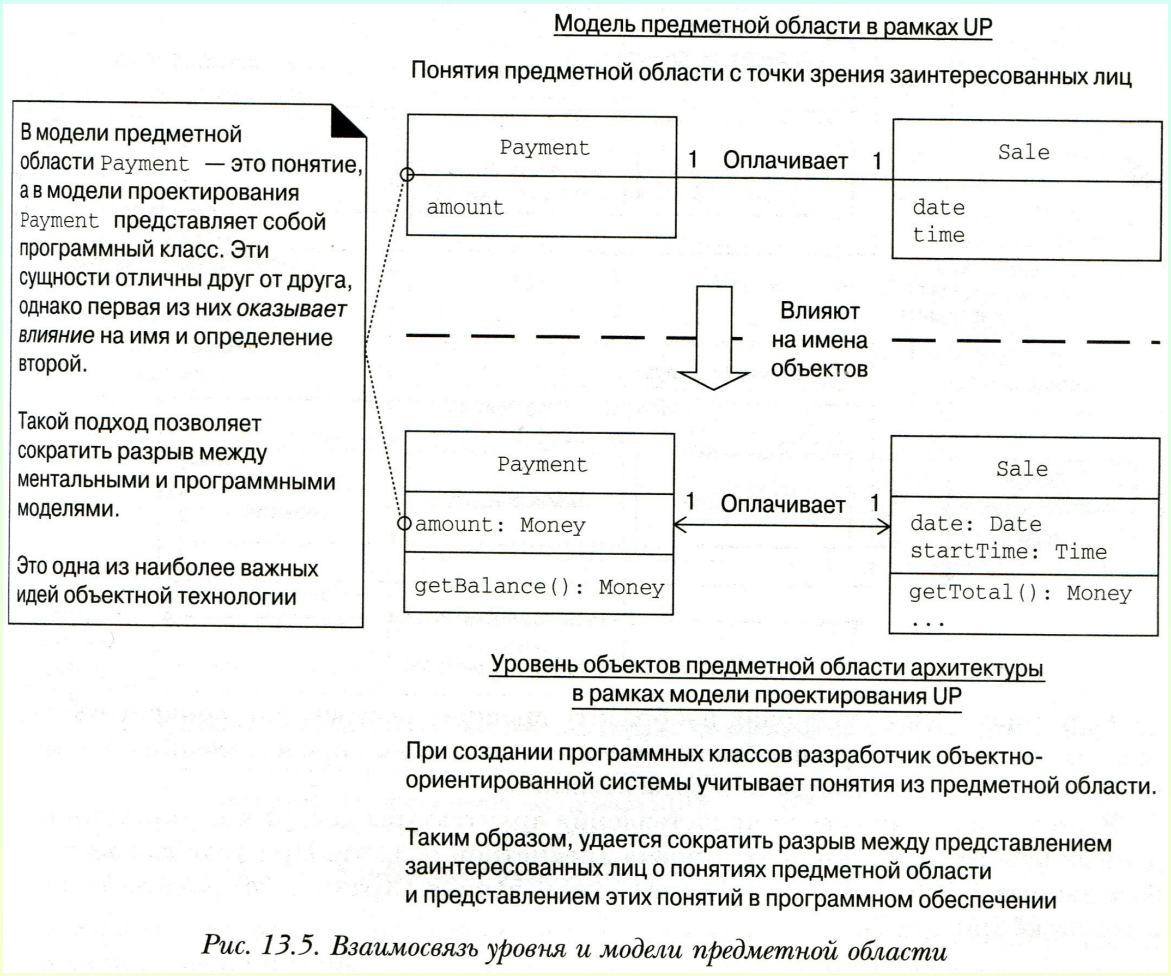


Рис. 13.5. Взаимосвязь уровня и модели предметной области

Рівень предметної галузі є частиною програмної реалізації, а модель предметної галузі – частиною концептуального аспекту аналізу.

Рис. 13.5 стр. 237

Логічна архітектура і діаграми пакетів

Зміст

Рівні і розділи

Архітектурні рівні представляють ділення системи по вертикалі, а розділи – по горизонталі на паралельні підсистеми в рамках одного рівня.

Наприклад, рівень служб можна розділити на розділи, які відповідають за виконання вимог безпеки і формування звітів.

Зовнішні ресурси чи рівень зовнішньої бази даних

Більшість систем спирається на зовнішні ресурси чи служби, такі як бази даних.
В логічному представлені архітектури можна використовувати не тільки рівні, але і фізичні реалізації компонентів.

В термінах логічного представлення архітектури доступ до конкретної бази даних відображається як пакет рівня предметної галузі. При цьому для доступу до бази даних можна використовувати пакет Persistence (Сховище) розділу технічних служб.

Зовнішні ресурси чи рівень зовнішньої бази даних

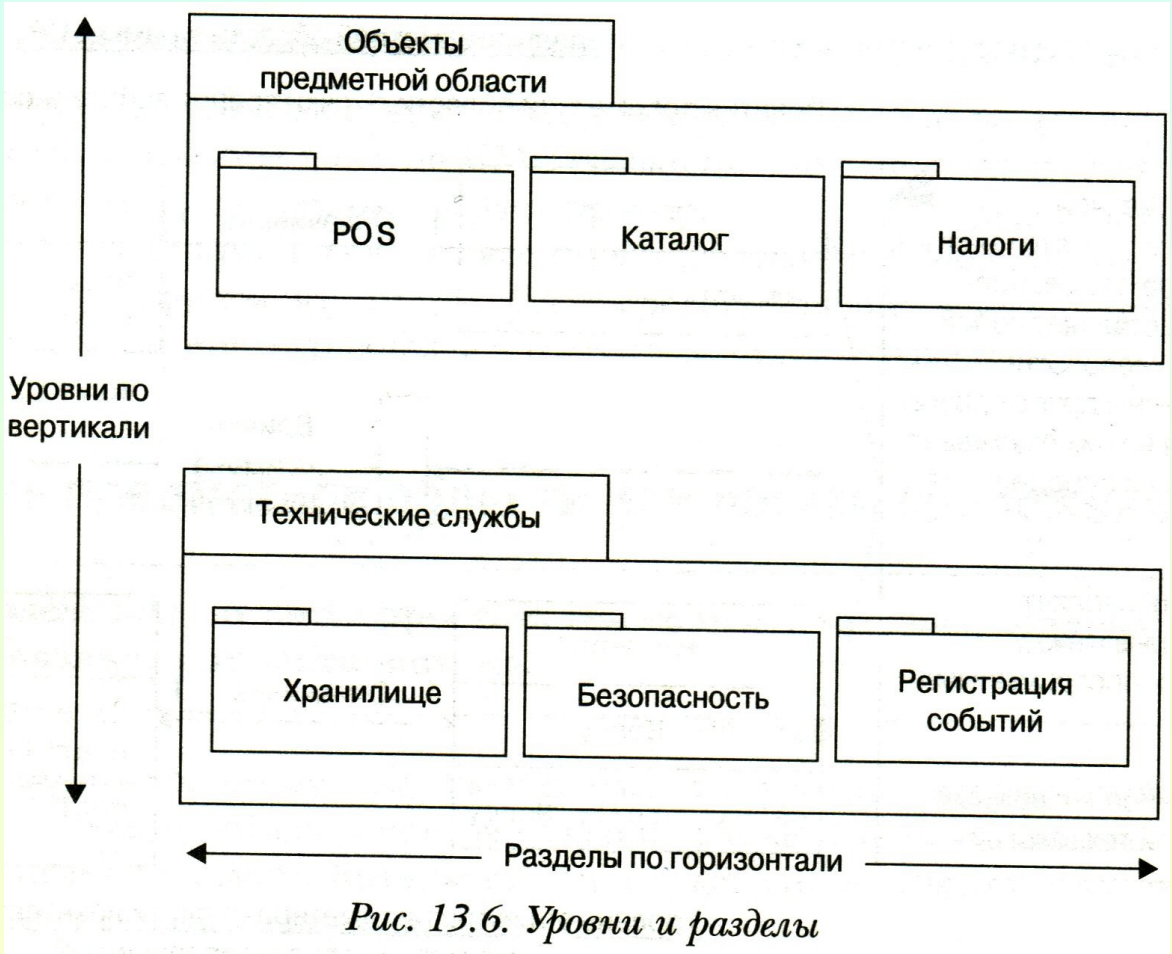


Рис. 13.6. Уровни и разделы

Рис. 13.6 стр. 238

Зовнішні ресурси чи рівень зовнішньої бази даних

Рис. 13.7. Комбинация разных представлений архитектуры

Рис. 13.7 стр. 239

Логічна архітектура і діаграми пакетів

Зміст

Принцип Model-View Separation

Цей принцип можна сформулювати так:

Не зв’язуйте об’єкти інтерфейсу користувача з об’єктами інших рівнів напряму.

Не використовуйте імена об’єктів логіки застосування в методах об’єктів інтерфейсу користувача. Об’єкти інтерфейсу користувача повинні лише ініціалізувати відповідні елементи інтерфейсу, отримати повідомлення про події (наприклад: про переміщення миші чи натисканні на кнопку) і делегувати запити об’єктам рівня логіки застосування.

Принцип Model-View Separation

Використання принципу MVS надає можливості:

Підтримує високий рівень зчеплення спеціальних об’єктів і дозволяє сконцентрувати увагу на процесах предметної галузі, а не на питаннях інтерфейсу.

Дозволяє розділити процес розробки рівнів моделі і інтерфейсу користувача.

Мінімізує вплив змін інтерфейсу на спеціальні об’єкти.

Дозволяє легко підключати новий інтерфейс, не торкаючись рівня реалізації.

Дозволяє одночасно використовувати декілька різних представлень для одного і того ж спеціального об’єкта (вивести інформацію у вигляді таблиці і діаграми).

Забезпечує виконання задач рівня реалізації незалежно від інтерфейсу користувача (обробка повідомлень).

Спрощує перехід до іншого каркасу інтерфейсу користувача.

Принцип Model-View Separation

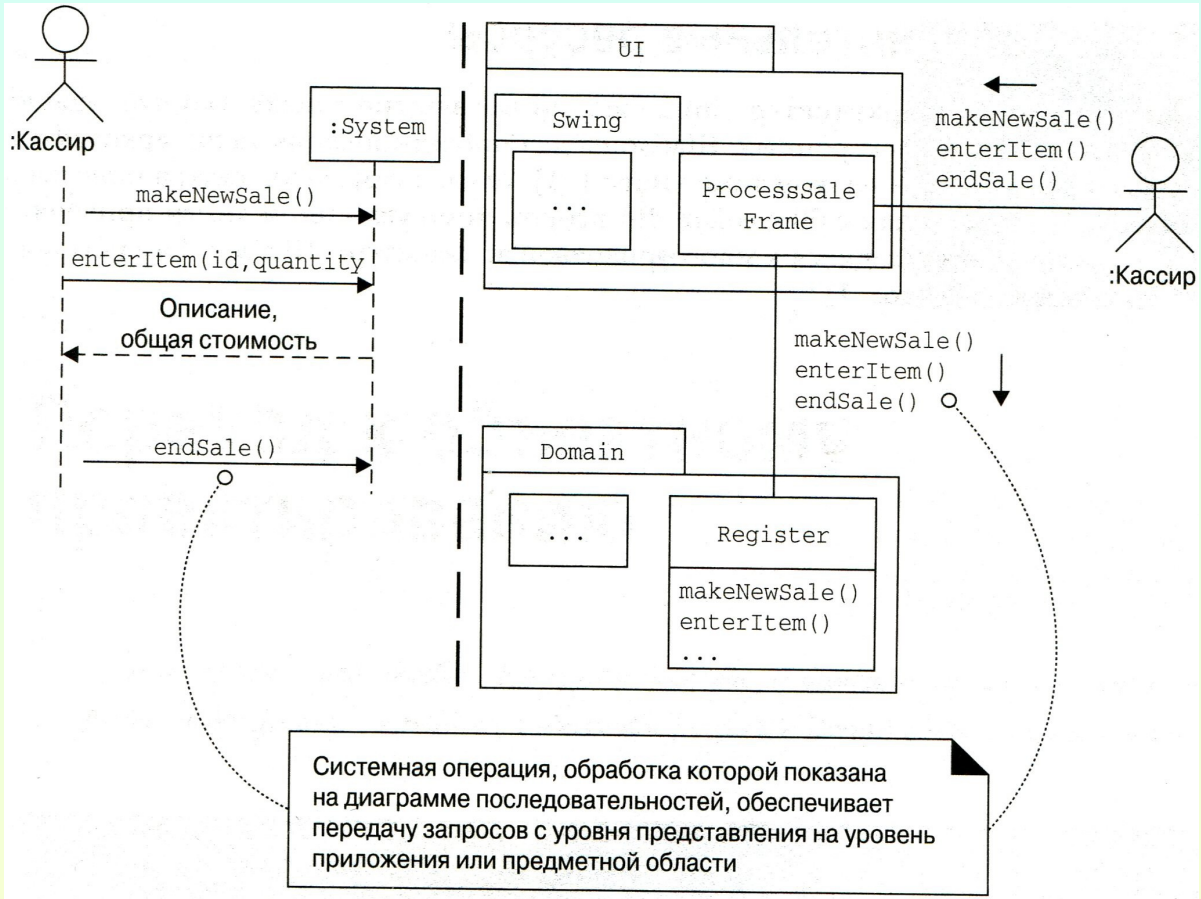


Рис. 13.8. Системные операции на системной диаграмме последовательностей в терминах уровней системы

Рис. 13.8 стр. 241

Заключна частина

Вимоги до сучасного програмного забезпечення стають все більш складними, оскільки користувачі очікують все більше від додатків. Можливостей простих автономних настільних застосувань більше не достатньо для більшості комерційних і ділових ситуацій.

У світі розвиненою зв'язку застосування повинні взаємодіяти з іншими застосуваннями і службами, а також працювати в різних середовищах, наприклад у хмарі, і на портативних пристроях.

На зміну поширеним у минулому монолітних архітектур прийшло компонентне сервіс-орієнтоване програмне забезпечення, що використовує платформи, операційні системи, хост-додатки та мережі для реалізації функцій, про яких було не відомо всього кілька років тому.

Заключна частина

Ці труднощі впливають не тільки на архітектуру, але також і на розгортання, обслуговування і адміністрування програмного забезпечення.

Сукупна вартість володіння програмним забезпеченням тепер в основному складається з витрат, що виникають після розгортання.

Додаток з добре продуманою архітектурою забезпечить мінімальну сукупну вартість володіння завдяки зниженню витрат і часу, необхідних на розгортання додатку, забезпечення його роботи, оновлення для задоволення мінливих вимог та усунення проблем. Адміністрування та підтримка користувачів також будуть спрощені.

Заключна частина

Програмне забезпечення має відповідати кільком важливим критеріям:
Воно повинно забезпечувати безпеку, щоб додаток і його дані були захищені від атак зловмисників і випадкових помилок.
Воно повинно бути стійким і надійним для мінімізації збоїв і відповідних витрат.
Воно повинно працювати з необхідними параметрами у відповідність до вимог користувачів, такими як максимальний час відгуку або певне робоче навантаження.

Воно має бути простим у підтримці для зниження витрат на адміністрування і підтримку, а також досить розширюваним для включення неминучих змін і оновлень, які з часом будуть потрібні.

З усіма цими факторами пов'язані деякі компроміси:
Наприклад, реалізація найбільш безпечних механізмів з використанням складного шифрування вплине на продуктивність.
Реалізація безлічі параметрів конфігурації та оновлення може ускладнити розгортання і адміністрування.
Крім того, чим складніше архітектура, тим дорожче її реалізація.

Правильна архітектура повинна забезпечувати баланс цих факторів з метою отримання оптимального результату для певного сценарію.

Логічна архітектура і діаграми пакетів

Дякую за увагу