Інститут фізико-технічних та комп’ютерних наук

Відділ комп’ютерних технологій

Кафедра програмного забезпечення комп’ютерних систем

КУРСОВИЙ РЕФЕРАТ

на тему:

**ОБ′ЄКТНО-ЗОРІЄНТОВАНЕ ПРОГРАМУВАННЯ**

Студентки 2 курсу 243 групи

напряму підготовки

«Програмна інженерія»

Вітталь-Комолової К.Є.

Керівник

Доцент кафедри МПУіК,

Лазорик В.В.

Національна шкала \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кількість балів: \_\_\_\_\_\_\_\_\_\_Оцінка: ECTS \_\_\_\_\_

**Члени комісії** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (прізвище та ініціали)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (прізвище та ініціали)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (прізвище та ініціали)

Чернівці – 2015

**ЗМІСТ**

[Принципи ООП 3](#_Toc417334888)

[Об'єктно-орієнтований підхід 5](#_Toc417334889)

[Основні поняття об'єктно-орієнтованого підходу - об'єкт і клас 6](#_Toc417334890)

[Управління: контроль за рухом поїздів 9](#_Toc417334891)

[Визначення меж проблемної області 10](#_Toc417334892)

[Вимоги до системи управління рухом 11](#_Toc417334893)

[Системні і програмні вимоги: крихкий компроміс 16](#_Toc417334894)

[Ключові абстракції і механізми 20](#_Toc417334895)

[Модульна архітектура 22](#_Toc417334896)

Принципи ООП

Ми живемо у незвичайному часі. Адже зовсім недавно, наші батьки і в мріях не могли подумати про те, що коли-небудь настане той час, коли комп'ютер стане невід'ємною частиною нашого життя, і реально почне приносити величезну користь. Стане генератором ідей та їх втілювачем, відкриє нові горизонти в знаннях людства. Але комп'ютер не дивлячись ні на що, без людини ніщо. Ось чому так важливо донести до машини людську думку, а допомагають нам у цьому різні способи з проектування ПЗ.

Проектування економічних інформаційних систем (ЕІС) – логічно складна, трудомістка і тривала робота, що вимагає високої кваліфікації беруть участь у ній фахівців. На початку 70-х рр. в США була відзначена криза програмування (software crisis). Це виражалося в тому, що великі проекти стали виконуватися з відставанням від графіка або з перевищенням кошторису витрат, розроблений продукт не володів необхідними функціональними можливостями, продуктивність його була низька, якість одержаного програмного забезпечення не влаштовувала споживачів.

Аналітичні дослідження та огляди, що розглядаються протягом останніх років провідними зарубіжними аналітиками, показували не надто обнадійливі результати. Так, наприклад, в 1995 р. компанія StandishGroup проаналізувала роботу 364 американських корпорацій і підсумки виконання більше 23 тис. проектів, пов'язаних з розробкою ПЗ, і зробили наступні висновки:

* Тільки 16% проектів завершилися в строк, 52,7% завершилися з запізненням, витрати перевищили запланований бюджет.
* У числі причин невдач фігурують: нечітке і не повне формулювання вимог до ПЗ, недостатнє залучення користувачів в роботу над проектом, незадовільне планування і т.п.

На цьому тлі, вигідно відрізняється об'єктно - орієнтований підхід до проектування ПО, він усуває ці та інші недоліки, володіє багатим набором образотворчих засобів. Ось чому, метою реферату є розкриття сучасних методів і засобів проектування, зокрема в об'єктно-орієнтованому підході до проектування ПЗ.

Принципова відмінність між структурним і об'єктно-орієнтованим підходом полягає в способі декомпозиції системи. Об'єктно орієнтований підхід використовує об'єктну декомпозицію, при цьому статична структура системи описується в термінах об'єктів і зв'язків між ними, а поведінка системи описується в термінах обміну повідомленнями між об'єктами. Кожен об'єкт системи має свою власну поведінку, моделюючи поведінку об'єкта реального світу. Поняття "об'єкт" вперше було використано близько 30 років тому в технічних засобах при спробах відійти від традиційної архітектури фон Неймана і подолати бар'єр між високим рівнем програмних абстракцій і низьким рівнем абстрагування на рівні комп'ютерів. З об'єктно-орієнтованої архітектурою також тісно пов'язані об'єктно-орієнтовані операційні системи. Однак найбільш значний внесок у об'єктний підхід був внесений об'єктними та об'єктно-орієнтованими мовами програмування: Simula, Smalltalk, C ++, Object Pascal. На об'єктний підхід вплинули також розвиваючі досить незалежно методи моделювання баз даних, в особливості підхід "сутність-зв'язок".

Концептуальною основою об'єктно-орієнтованого підходу є об'єктна модель. Основними її елементами є:

* абстрагування (abstraction);
* інкапсуляція (encapsulation);
* модульність (modularity);
* ієрархія (hierarchy).

Крім основних є ще три додаткових елементи, які не є на відміну від основних суворо обов'язковими:

* типізація (typing);
* паралелізм (concurrency);
* стійкість (persistence).

Абстрагування - це виділення істотних характеристик деякого об'єкта, які відрізняють його від всіх інших видів об'єктів і, таким чином, чітко визначають його концептуальні межі щодо подальшого розгляду та аналізу. Абстрагування концентрує увагу на зовнішніх особливостях об'єкта і дозволяє відокремити найістотніші особливості його поведінки від деталей їх реалізації. Вибір правильного набору абстракцій для заданої предметної області являє собою головне завдання об'єктно-орієнтованого проектування.

Інкапсуляція - це процес відділення один від одного окремих елементів об'єкта, що визначають його пристрій і поведінку. Інкапсуляція служить для того, щоб ізолювати інтерфейс об'єкта, що відображає його зовнішнє поведінку, від внутрішньої реалізації об'єкта. Об'єктний підхід припускає, що власні ресурси, якими можуть маніпулювати тільки методи самого класу, приховані від зовнішнього середовища. Абстрагування і інкапсуляція є взаємодоповнюючими операціями: абстрагування фокусує увагу на зовнішніх особливостях об'єкта, а інкапсуляція (або, інакше, обмеження доступу) не дозволяє об'єктам-користувачам розрізняти внутрішнє пристрій об'єкта.

Об'єктно-орієнтований підхід

Модульність - це властивість системи, пов'язана з можливістю її декомпозиції на ряд внутрішньо зв'язкових, але слабко пов'язаних між собою модулів. Інкапсуляція і модульність створюють бар'єри між абстракціями.

Ієрархія - це впорядкована система абстракцій, розташувана за їх рівнями. Основними видами ієрархічних структур стосовно складних систем є структура класів (ієрархія по номенклатурі) і структура об'єктів (ієрархія по змісту). Прикладами ієрархії класів є просте і множинне успадкування (один клас використовує структурну або функціональну частину відповідно одного або декількох інших класів), а ієрархії об'єктів - агрегація.

Типізація - це обмеження, що накладається на клас об'єктів і перешкоджає взаємозамінності різних класів (або сильно звужує її можливість). Типізація дозволяє захиститися від використання об'єктів одного класу замість іншого або принаймні керувати таким використанням.

Паралелізм - властивість об'єктів знаходитися в активному або пасивному стані і розрізняти активні і пасивні об'єкти між собою.

Стійкість - властивість об'єкта існувати в часі (поза процесом, який породив даний об'єкт) та / або в просторі (при переміщенні об'єкта з адресного простору, в якому він був створений).

Основні поняття об'єктно-орієнтованого підходу - об'єкт і клас

Об'єкт визначається як відчутна реальність (tangible entity) - предмет або явище, що мають чітко визначену поведінку. Об'єкт характеризуєтьсяє станом, поведінкою і індивідуальністю; структура та поведінка схожих об'єктів визначають загальний для них клас. Терміни "екземпляр класу" і "Об'єкт '' є еквівалентними. Стан об'єкта характеризується переліком усіх можливих (статичних) властивостей даного об'єкта і поточними значеннями (динамічними) кожного з цих властивостей. Поведінка характеризує вплив об'єкта на інші об'єкти і навпаки щодо зміни стану цих об'єктів та передачі повідомлень. Інакше кажучи, поведінка об'єкта повністю визначається його діями. Індивідуальність – це властивості об'єкта, що відрізняють його від усіх інших об'єктів.

Певний вплив одного об'єкта на інший з метою викликати відповідну реакцію називається операцією. Як правило, в об'єктних і об'єктно-орієнтованих мовах операції, що виконуються над даним об'єктом, називаються методами і є складовою частиною визначення класу.

Клас - це безліч об'єктів, пов'язаних спільністю структури і поведінки. Будь-який об'єкт є екземпляром класу. Визначення класів і об'єктів - одне з найскладніших завдань об'єктно-орієнтованого проектування.

Наступну групу важливих понять об'єктного підходу складають успадкування та поліморфізм. Поняття поліморфізму може бути інтерпретовано як здатність класу належати більш ніж одному типу. Успадкування означає побудову нових класів на основі існуючих з можливістю додавання або перевизначення даних і методів.

Об'єктно-орієнтована система спочатку будується з урахуванням її еволюції. Успадкування та поліморфізм забезпечують можливість визначення нової функціональності класів за допомогою створення похідних класів - нащадків базових класів. Нащадки успадковують характеристики батьківських класів без зміни їх первісного опису і додають при необхідності власні структури даних та методи. Визначення похідних класів, при якому задаються тільки відмінності або уточнення, в величезній мірі економить час і зусилля при виробництві та використанні специфікацій і програмного коду.

Важливою властивістю об'єктного підходу є узгодженість моделей діяльності організації і моделей проектованої системи від стадії формування вимог до стадії реалізації. Вимога узгодженості моделей виконується завдяки можливості застосування абстрагування, модульності, поліморфізму на всіх стадіях розробки. Моделі ранніх стадій можуть бути безпосередньо піддані порівнянні з моделями реалізації. За об'єктним моделям може бути простежено відображення реальних сутностей модельованої предметної області (організації) в об'єкти і класи інформаційної системи.

Таким чином, концепція об'єктно-орієнтованого програмування має на увазі, що основою управління процесом реалізації програми є передача повідомлень об'єктам. Тому об'єкти повинні визначатися спільно з повідомленнями, на які вони повинні реагувати при виконанні програми. У цьому полягає головна відмінність ООП від процедурного програмування, де окремо певні структури даних передаються в процедури (функції) в якості параметрів. Таким чином, об'єктно-орієнтована програма складається з об'єктів - окремих фрагментів коду, що обробляє дані, які взаємодіють один з одним через певні інтерфейси.

Основна мета об'єктно-орієнтованого програмування, як і більшості інших підходів до програмування - підвищення ефективності розробки програм. Ідеї об'єктно-орієнтованого програмування виявилися плідними і знайшли застосування не тільки в мовах програмування, але і в інших областях комп′ютерних наук, наприклад, в області розробки операційних систем.

Основні ідеї об'єктно-орієнтованого підходу спираються на наступні положення:

- Програма являє собою модель деякого реального процесу, частини реального світу.

- Модель реального світу або його частини може бути описана як сукупність взаємодіючих між собою об'єктів.

- Об'єкт описується набором параметрів, значення яких визначають стан об'єкта, і набором операцій (дій), які може виконувати об'єкт.

Базовим в об'єктно-орієнтованому програмуванні є поняття об'єкта. Об'єкт має певні властивості. Стан об'єкта задається значеннями його ознак. Об'єкт «знає», як вирішувати певні завдання, тобто має у своєму розпорядженні методами рішення. Програма, написана з використанням ООП, складається з об'єктів, які можуть взаємодіяти між собою.

Об'єкт - це відчутна реальність, що характеризується чітко визначеною поведінкою.

Об'єкт - особливий упізнаваний предмет, блок або сутність (реальна чи абстрактна), що має важливе функціональне призначення в даній предметній області.

Об'єкт може бути охарактеризований структурою, станом об'єкта, його поведінкою і індивідуальністю.

Практично всі об'єктно-орієнтовані мови програмування є країнами, що розвиваються мовами, їхні стандарти регулярно уточнюються і розширюються. Наслідком цього розвитку є неминучі відмінності у вхідних мовах компіляторів різних систем програмування.

Управління: контроль за рухом поїздів

Програмна індустрія розвилася настільки, що охоплює багато нових областей додатків: від вбудованих мікрокомп'ютерів для керування двигуном автомобіля до виконання рутинної роботи при виготовленні фільмів і забезпечення інтерактивного доступу мільйонів телеглядачів до баз відеоінформації. Відмінною особливістю таких великих систем є їх надзвичайна складність. Звичайно, побудувати компактну реалізацію системи - завдання почесне, але деякі великі завдання безсумнівно вимагають великого обсягу коду. У великих проектах нерідко беруть участь програмістські організації в сотні осіб, які мають написати мільйони рядків коду. Програми повинні задовольняти вимоги, які неминуче змінюються в процесі роботи. Як правило, в рамках таких проектів створюється не одна програма, що працює на одному комп'ютері, а комплекс програм, що функціонують в паралельно розподіленому середовищі на декількох комп'ютерах, зв'язаних між собою різноманітними каналами передачі інформації. Для того, щоб зменшити ймовірність невдачі, в таких проектах передбачається звичайно центральна організація, що відповідає за архітектуру і цілісність системи. Деякі частини системи нерідко виконуються по субконтрактам іншими компаніями. Таким чином, команда розробників ніколи не збирається разом, вона розподілена в просторі і, - оскільки у великих проектах відбувається постійне оновлення кадрів, - в часі.

Якщо за створення великої системи візьметься розробник, який займався написанням у віконному середовищі невеликих програм, розрахованих на одного користувача, його безсумнівно злякають виникаючі проблеми; можливо, навіть настільки, що він вважатиме дурницею спробу створити таку програму. Але дійсність така, що великі системи повинні будуватися. І в деяких випадках нерозумно не спробувати. Уявімо собі ручне управління авіаційними польотами навколо столичного аеропорту, систему життєзабезпечення космічної станції, залежну від "людського фактора" або ведення обліку в міжнародному банку, що виконується на рахунках. Успішна автоматизація таких систем призводить не тільки до вирішення очевидних проблем, але і приносить безліч несподіваних вигод: зниження експлуатаційних витрат, підвищення надійності, збільшення функціональних можливостей. Звичайно ж, ключове слово тут - успішна. З усього сказаного зрозуміло, що створення великих систем - надзвичайно важке завдання. Тому при його вирішенні необхідно застосовувати все найкраще з інженерної практики і використовувати інтуїцію провідних проектувальників.

Визначення меж проблемної області

Для більшості людней, що живуть в США, потяги є символом давно минулої епохи. У Європі та країнах Сходу ситуація абсолютно протилежна. На відміну від США, в Європі мало національних і міжнародних автомобільних магістралей, а ціни на бензин і газ порівняно високі. Тому поїзда складають основу транспортної мережі континенту; по десяткам тисяч кілометрів шляхів щодня перевозиться безліч людей і вантажів - і в окремих містах, і між різними країнами. Заради справедливості відзначимо, що в США поїзди грають і раніше важливу роль у перевезенні вантажів. Із зростанням міст їх центри стають все більш і більш перевантаженими, і на легкий рейковий транспорт покладаються надії вирішити проблему перевантаження і забруднення навколишнього середовища двигунами внутрішнього згоряння.

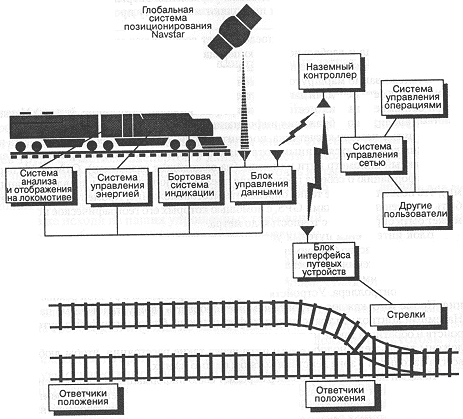
Залізниці і зараз є комерційними, отже, вони повинні бути прибутковими. Залізничні компанії зобов'язані постійно підтримувати баланс між вимогами економії і безпеки та зростаючою інтенсивністю перевезень з одного боку і ефективним і передбачуваним розкладом - з іншого. Ці протиріччя наводять на думку, що рішення про управління рухом поїздів необхідно приймати автоматично, і, в тому числі, проводити контроль за всіма елементами залізниці за допомогою комп'ютера.

Такі автоматичні і напівавтоматичні системи сьогодні існують в Швеції, Великобританії, Німеччини, Франції та Японії. Подібна система, називається просунутою системою управління залізничним транспортом, була розроблена в Канаді і США за участю наступних компаній: Amtrak, Burlington, Canadian National Railway Company, CP Rail, CSX Transportation, Norfolk and Western Railway Company, Southern Railway Company, Union Pacific. Ефект від кожної з цих систем був і економічний, і соціальний; результатом їх впровадження стало зниження експлуатаційних витрат, підвищення ефективності використання ресурсів, безпека.

Вимоги до системи управління рухом

Система управління рухом виконує дві основні функції: вибір маршрутів залізничних перевезень та контроль систем, що забезпечують перевезення. Ці функції включають: планування перевезень, контроль місцезнаходження поїздів, контроль за перевезеннями, запобігання конфліктів, прогнозування порушенні, реєстрацію всіх операцій. На рис. 1 показана схема основних елементів системи управління рухом.

Система аналізу і відображення інформації на локомотиві складається з безлічі дискретних і аналогових датчиків для контролю за такими параметрами, як температура, тиск масла, кількість палива, напруга і сила струму на генераторі, число обертів вала двигуна в хвилину, температура води, тягова потужність. Значення параметрів з датчиків надходять до машиніста через дисплейну систему, а до диспетчера і обслуговуючому персоналу поза поїзда - через мережу. Попередження або сигнал тривоги видається і реєструється всякий раз, коли показання датчика виходять за межі нормального режиму. Журнал показань датчиків використовується при проведенні експлуатаційних робіт і для управління витратою палива.



*Рис. 1. Система управління рухом*

Система управління енергією в режимі реального часу підказує інженеру поїзда, як найбільш ефективно використовувати установку. Вхідними даними для цієї системи є: профіль і якість шляху, обмеження по швидкості, розклад, завантаження поїзда, максимальна потужність, що розвивається. Виходячи з цих даних, система може визначити оптимальний по витраті палива режим роботи рухових установок, узгоджуючи із заданим розкладом і вимогами безпеки. Рекомендації системи, профіль і якість шляху, місце розташування і швидкість поїзда можуть відображатися за допомогою бортової системи індикації.

Бортова система індикації забезпечує людино-машинний інтерфейс для машиніста. На неї може виводитися інформація з системи аналізу і відображення інформації на локомотиві, системи управління енергією і блоку управління даними. Спеціальні клавіші дозволяють машиністу переглядати різні дані.

Блок управління даними являє собою шлюз між усіма бортовими системами поїзда і глобальною мережею передачі даних, до якої підключені всі поїзди, диспетчери та інші користувачі.

Відстеження маршрутів руху поїздів здійснюється за допомогою підключених до мережі передачі даних відповідачів місця розташування і глобальної супутникової системи вказівки позиціонування (GPS, Global Positioning System) Navstar. Система аналізу і відображення інформації на локомотиві може обчислювати пройдений шлях за допомогою лічильника, підраховувати число обертів колеса. Ця інформація доповнюється даними датчиків місця розташування, які розміщені через кожен кілометр шляху або частіше (на найважливіших розвилках). Датчики передають інформацію про себе на потяги (використовуючи блок управління даними), що дозволяє більш точно визначити місце розташування. Крім того, поїзд може бути оснащений приймачами GPS, за допомогою яких його географічне положення може бути визначено з точністю до метра.

Блок інтерфейсу колійних пристроїв розміщується там, де є якийсь керований пристрій (наприклад, стрілка), або датчик (наприклад, інфрачервоний датчик для виявлення перегріву підвісок коліс). Кожен блок інтерфейсу отримує команди (наприклад, команди на включення і виключення сигналу) від локального наземного контролера. Пристрої можуть бути переведені в ручний режим управління. Крім того, кожен пристрій може повідомляти свої установчі параметри. Наземний контролер транслює інформацію на блоки інтерфейсу колійних пристроїв і назад, а також на що проходять повз поїзда і назад. Контролери розташовані вздовж залізничної колії через такі відстані, щоб будь-який потяг завжди знаходився в зоні дії хоча б одного з них.

Кожен наземний контролер передає свою інформацію на об'єднану систему управління мережею. Зв'язок між системою управління мережею і наземним контролером може здійснюватися по радіо в мікрохвильовому діапазоні, по наземних лініях або по оптоволокну в залежності від віддаленості даного контролера. Система управління мережею забезпечує функціонування всієї мережі. Вона може автоматично направляти інформацію за іншим маршрутом в мережі, якщо на одному з шляхів відбудеться відмова обладнання,

Система управління мережею, в свою чергу, під'єднується до одного або декількох диспетчерських центрів, які об'єднані в систему управління операціями. Система управління мережею з'єднана і з іншими користувачами. У системі управління операціями диспетчери можуть задавати маршрути поїздів і відстежувати їх пересування. Для управління різними ділянками виділяються окремі диспетчери; кожна диспетчерська керуюча консоль відповідає за одну або декілька територій. Маршрутизація поїздів на увазі видачу інструкцій для автоматичного перекладу поїзда з колії на колію, установку обмеження швидкості, керування пропуском автомобілів на переїздах, дозвіл і заборона руху поїзда в залежності від зайнятості певних ділянок шляху. Диспетчери можуть спостерігати за станом шляхів попереду по маршруту поїзда і передавати цю інформацію машиністу. Потяги можуть бути зупинені системою управління операціями (вручну диспетчерами або автоматично), коли виявляється небезпека (вихід поїзда з графіка, пошкодження колії, можливість зіткнення). Диспетчери можуть також викликати на екран будь-яку інформацію, доступну машиністам окремих поїздів, розіслати розпорядження з руху, встановити параметри колійних пристроїв і переглянути план руху.

Розташування шляхів і шляхове обладнання можуть з часом змінюватися. Число поїздів і маршрути їх руху можуть змінюватися щодня. Система повинна забезпечувати можливість підключення нових датчиків, мереж та обладнання, виконаних за більш досконалим технологіям.

Сформульовані основні вимоги до системи управління рухом поїздів. Очевидно, вони сильно спрощені. На практиці детальні вимоги до великої системи виробляються після демонстрації життєздатності програмного рішення проблеми. При цьому аналіз скасовує сотні людино-місяців праці за участю експертів в даній області і користувачів системи. У кінцевому рахунку вимоги до системи можуть складатися з тисяч сторінок документації, специфікувати не тільки базову поведінку, але й такі деталі, як прототипи форм інтерфейсу.

Але навіть виходячи з наших спрощених вимог, ми можемо зробити два зауваження про розробку системи управління рухом:

• Архітектура повинна бути відкрита для розвитку.

• Реалізація повинна спиратися на існуючі стандарти.

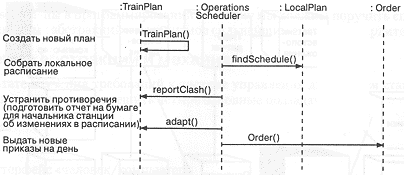
Наш досвід розробки великих систем показує, що первісне формулювання вимог ніколи не буває повним, воно завжди в деякій мірі невизначене і суперечливе. Відповідно, ми повинні бути готові управляти виникаючими в процесі розробки невизначеностями. Ми наполегливо рекомендуємо здійснювати еволюцію подібних систем у вигляді покрокового, ітеративного процесу. Сам цикл розробки дає користувачам і розробникам можливість зрозуміти, які вимоги насправді істотні; саме процес розробки, а не вправи в краснописі специфікацій у відсутності готової часткової реалізації або прототипу. Крім того, необхідно враховувати, що на створення великої системи може бути витрачено кілька років. За цей час сильно зміниться апаратна частина (В дійсності для багатьох систем такого рівня складності характерно, що в них входять комп'ютери найрізноманітніших типів. Добре продумана і стабільна архітектура пом'якшує ризик зміни техніки в процесі розробки, яка часто-густо відбувається в мінливому комп'ютерному світі. Нові моделі приходять і йдуть, тому важливо чітко уявляти кордон між технікою і програмами, щоб можна було ввести в систему нові комп'ютери або контролери, що знижують витрати або поліпшують характеристики роботи, і зберегти при цьому цілісність архітектури). Тому вимоги до програми повинні передбачати адаптацію до нової техніки. Безглуздо створювати елегантну архітектуру для апаратури, яка гарантовано застаріє за час розробки. Ми вважаємо, що в архітектуру програмної системи слід включати тільки ті апаратні особливості, які безпосередньо спираються на існуючі стандарти: зв'язок, мережі передачі даних, графіку і протокол роботи датчиків. Для абсолютно нових систем іноді доводиться ставати першопрохідцями апаратних і програмних засобів. Це призводить до підвищення ризику, який для більшості систем і без того високий. Розробка програмного забезпечення, особливо, коли мова йде про успішне завершення великого додатку, неминуче пов'язана з ризиком, і наша мета - знизити цей ризик до мінімуму.

Системні і програмні вимоги: крихкий компроміс

Великі проекти, подібні розглянутому, зазвичай організовуються навколо невеликої центральної групи, відповідальної за глобальну архітектуру системи, а сама розробка передається стороннім субпідрядникам або іншим групам всередині тієї ж організації. Вже на стадії аналізу системні архітектори мають деяку концептуальну модель, яка розділяє апаратну і програмну частини реалізації. Багато хто, правда, вважає, що це вже не аналіз, а проектування. Це - спірне питання. Справді, важко вирішити, що показано на схемі рис. 1- вихідні вимоги або проект системи. Але в будь-якому випадку схема припускає, що на даній стадії розробки архітектура системи принципово об'єктно-орієнтована. Наприклад, на схемі присутні такі складні об'єкти, як система управління енергією або система управління операціями. Кожен з них виконує одну з основних функцій всієї системи. Об'єкти найвищого рівня абстракції відповідають за основні функції системи. Тому процес аналізу в даному випадку мало відрізняється від процесу проектування.

Коли ми вже маємо скелет архітектури (як на рис. 1), можна за допомогою експертів в даній прикладній області приступати до розробки основних сценаріїв поведінки системи. Щоб докладніше описати очікувану поведінку системи, можна використовувати діаграми взаємодії, діаграми об'єктів, протоколи дій чи прототипи. На рис. 2 наведена діаграма взаємодії компонента системи, що відображає сценарій підготовки щоденних наказів з руху поїздів. На даному рівні аналізу нас цікавлять саме основні події та взаємодії, що визначають поведінку системи. Такі деталі, як сигнатури операцій і асоціацію - це тактичні подробиці, які знадобляться на наступних фазах проектування.

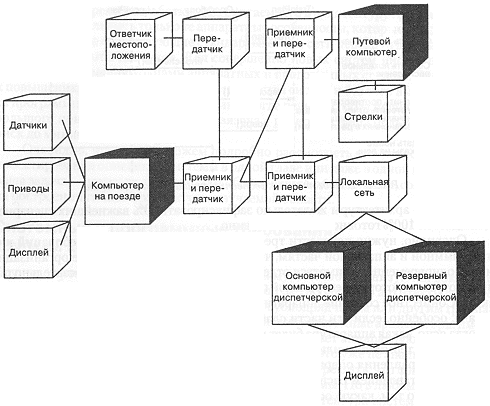
У системі таких розмірів запросто можна знайти сотні первинних. Ми вже встановили "правило 80%". Це означає, що до переходу до проектування архітектури бажано зафіксувати 80% найважливіших сценаріїв. Чекати 100% готовності безглуздо.



*Рис. 12-2. Подготовка ежедневных приказов по движению*

Очевидно, потрібно перевести вимоги до системи на мову вимог до її програмної та апаратної частинах, щоб різні компетентні організації могли одночасно займатися окремими частинами задачі (але обов'язково під наглядом деякої центральної групи, що забезпечує спільне бачення проекту). Спільне створення апаратного та програмного забезпечення - складне завдання, особливо, якщо ці частини слабко пов'язані і створюються різними фірмами. Іноді ясно, яка апаратура буде використовуватися. Наприклад, можна використовувати готові термінали або робочі станції для бортових дисплейних систем і в центрах управління операціями. Аналогічно, видається цілком очевидним, що складанням розкладів поїздів займаються програми. Остаточне рішення про те, яку основу, апаратну або програмну, використовувати в кожному конкретному випадку, залежить від уподобань розробників не менше, аніж від всього іншого. Спеціалізовану апаратуру можна використовувати, коли важливіше продуктивність, а використання програм доцільніше, коли необхідно забезпечити гнучкість.

Будемо вважати, що початковий варіант апаратної архітектури обраний архітекторами системи. Цей вибір не повинен вважатися остаточним, але принаймні він дає відправну точку для уточнення вимог до програмного забезпечення. У ході аналізу, а потім і проектування, нам необхідна свобода у виборі апаратної або програмної реалізації тієї чи іншої функції: пізніше може виявитися, що потрібна додаткова апаратура, або що дану функцію можна реалізувати програмно.



*Рис. 3. Диаграмма процессов системы управления движением*

На рис. 3 показано цільове апаратне забезпечення для системи управління рухом; тут використовуються наші позначення для діаграм процесів. Ця архітектура процесів відповідає схемі на рис. 1. Зокрема, передбачено один бортовий комп'ютер на кожному поїзді, що з'єднує систему збору та передачі інформації про локомотиві, систему управління енергією, бортовий дисплей і пристрій керування даними. Ми припускаємо, що деякі бортові пристрої, такі, як дисплей, володіють мінімальним інтелектом, але, можливо, не всі вони програмовані. Ми вважаємо, що кожен відповідач приєднаний до передавача, який посилає повідомлення на проходить повз нього потяг; комп'ютер до відповідача місця розташування не підключений. Всі групи колійних пристроїв (кожне з яких логічно складається з інтерфейсу і перемикача) управляються комп'ютером, який може взаємодіяти з проходять поїздом або з наземним контролером через їх передавачі та приймачі. Кожен наземний контролер приєднується через глобальну мережу до диспетчерського центру (який входить в систему управління операціями). Для забезпечення безперебійного обслуговування ми вирішили розмістити на кожному диспетчерському центрі два комп'ютери: основний і резервний (другий включиться в разі відмови основного комп'ютера). У вільний час резервний комп'ютер може використовуватися для обслуговування інших, фонових користувачів.

На експлуатаційному рівні система управління рухом може містити сотні комп'ютерів: по одному на кожен поїзд, по одному на кожен блок інтерфейсу колійних пристроїв і по два на кожен диспетчерський центр. На діаграмі процесів показані тільки деякі комп'ютери, так як зайво показувати повторювані компоненти конфігурації.

Здоровий глузд підказує, що при розробці великого проекту величезну роль грають розумність і ясність інтерфейсів між ключовими частинами системи. Особливо це важливо для інтерфейсу міжпрограмної та апаратної частинами системи. На початку роботи над проектом інтерфейс може бути визначений не повністю, але він повинен бути досить швидко формалізований, щоб різні частини системи можна було розробляти, тестувати та інтегрувати одночасно. Добре спроектований інтерфейс дозволяє проводити збірку системи без істотних переробок її частин. Крім того, ми не розраховуємо, що всі розробники, які беруть участь у проекті, будуть однаково сильні в програмуванні. Тому ми повинні доручити специфікації ключових абстракцій і механізмів найсильнішим системним архітекторам.

Ключові абстракції і механізми

В результаті вивчення вимог до системи управління рухом стає очевидно, що ми повинні вирішити чотири основні підзадачі:

* мережу
* база даних
* інтерфейс "людина / комп'ютер"
* управління аналоговими пристроями в реальному часі.

Систему пов'язує воєдино розподілена мережа передачі даних. За допомогою радіо передаються повідомлення: між відповідачами і поїздами, між поїздами і наземними контролерами, між поїздами і блоками інтерфейсів колійних пристроїв, між наземними контролерами і колійними пристроями. Крім того, повідомлення повинні передаватися між диспетчерськими центрами та окремими наземними контролерами. Надійна робота всієї системи забезпечується своєчасним і надійним прийомом і передачею повідомлень.

Крім того, система повинна одночасно зберігати інформацію про місцезнаходження і планованих маршрутах безлічі поїздів. Ми повинні підтримувати постійно оновлювану інформацію та гарантувати її цілісність навіть у разі спроб одночасно записати і зчитати інформацію з різних місць мережі. Отже, нам потрібна розподілена база даних.

Проектування людино-машинного інтерфейсу ставить ще одну групу завдань. Справа в тому, що користувачами системи в основному є машиністи і диспетчери; але ніхто з них не зобов'язаний володіти професійними навичками роботи з комп'ютером. Користувальницький інтерфейс операційних систем, таких як UNIX або Windows, придатний (здебільшого) для фахівця-програміста, але вважається занадто ворожим для кінцевих користувачів таких середовищ, як система управління рухом. Отже, всі форми взаємодії повинні бути спроектовані в розрахунку на цю особливу групу користувачів.

Нарешті, система управління рухом повинна взаємодіяти з різноманітними датчиками і виконавчими механізмами. Не зупиняючись тут на природі цих пристроїв, відзначимо, що принципи управління ними не залежить від конкретного типу пристрою і повинні бути обрані однотипними у всій системі.

Кожна з цих чотирьох підзадач включає цілий ряд відокремлених питань. Системні архітектори повинні знайти ключові абстракції і механізми кожного завдання, і тоді ми зможемо запросити експертів для вирішення кожної окремої підзадачі незалежно від інших. Однак, ні аналіз, ні проектування не вдасться завершити за один прохід, - коло за колом аналіз буде виявляти нові архітектурні проблеми, вирішення яких потребують нового аналізу. Таким чином, розробка буде неминуче покрокової та ітеративної.

З короткого проблемного аналізу чотирьох головних підзадач ми бачимо, що існують три високорівневі ключові абстракції.

|  |  |
| --- | --- |
| Поїзди | Локомотиви і вагони. |
| Шляхи | Профиль шляху, його якість і шляхові пристрої |
| Плани | Розклад, накази, усунення неполадок, призначення повноважень і підбір бригад. |

Кожен поїзд характеризується поточним становищем на шляхах і може мати тільки один активний план руху. Аналогічно, в кожній точці шляху може бути найбільше один поїзд. Кожен план відноситься тільки до одного поїзду, але до багатьох точок шляху.

Ми можемо виділити ключовий механізм для кожної з чотирьох (майже незалежних) підзадач:

• передача повідомлень

• планування руху поїздів

• відображення інформації

• збір даних від датчиків.

Отже, ці чотири механізму складають душу нашої системи. Вони є найбільш складними і ризикованими частинами проекту.

Модульна архітектура

Проектування програмного забезпечення для дуже великих систем має починатися до повного завершення проектування апаратних засобів. Написання програми займає, як правило, навіть більше часу, ніж розробка апаратури. Крім того, по ходу процесу функціональність може перерозподілятися між апаратної й програмної частинами. Тому залежність від апаратури повинна бути максимально ізольована, так, щоб програмні засоби можна було почати проектувати без прив'язки до апаратури. Це означає також, що розробка повинна грунтуватися на ідеї взаємозамінних підсистем. У системах управління та контролю, таких, як система управління рухом, потрібно зберегти можливість задіяти нові апаратні рішення, які можуть з'явитися в процесі розробки програмного забезпечення.