мережі з порівняно невеликою кількістю нейронів у внутрішніх шарах та реалізувати відповідні процедури навчання нейронної мережі на основі заданих навчальних виборок даних.

1. Нейроматематика (Нейрокомпьютеры и их применение. Кн. 6: Учеб. пособие для вузов) / Агеев А.Д., Балухто А.Н., Бычков А.В. и др. Под общ. ред. А.И. Галушкина. – М.: ИПРЖР, 2002. – 448 с. 2. Колмогоров А.Н. О представлении непрерывных функций нескольких переменных в виде суперпозиций непрерывных функций одного переменного и сложения // ДАН СССР. – 1957. – Т. 114. – № 5. – С. 953–956. 3. Бойков И.В. К проблеме Колмогорова о представлении аналитических функций нескольких переменных суперпозициями непрерывно дифференцируемых функций меньшего числа переменных // Вестник ОГГГГН РАН. – 2000.- № 3 (13). – 13 с. – http://www.scgis.ru/russian/cp25/ h_dgggms/3-2000/boikov.htm#begin. 4. Brattka V. A Computable Kolmogorov Superposition Theorem / Theoretische Informatik I, Fern Universität Hagen, Germany. – 2000. – 16 p. - http:// www.informatik.fernuni-hagen.de/thi1/vasco.brattka/publications/kolmogorov.pdf. 5. Sprecher D.A., Draghici S. Space-filling curves and Kolmogorov superposition-based neural networks // Neural Networks. – 2002. – No. 15. – P.57–67. - http://vortex.cs.wayne.edu/ papers/space_ filling_curves_preprint.pdf. 6. Nakamura M., Mines R., Kreinovich V. Guaranteed Intervals for Kolmogorov's Theorem (and Their Possible Relation to Neural Networks). – 1993. – 13 p. – http://citeseer.ist.psu.edu/pdf/697218.

УДК 681.3.049

Д.В. Корпильов, С.П. Ткаченко, Т.В. Свірідова
Національний університет "Львівська політехніка",
кафедра САПР

АНАЛІЗ І СИНТЕЗ АРХІТЕКТУРИ СКЛАДНИХ АПАРАТНИХ І ПРОГРАМНИХ СИСТЕМ

Проаналізовано актуальну тему синтезу архітектури програмних і апаратних систем, моделювання й верифікацію розроблювальних програмних систем.

In this paper overview of architecture synthesis, modeling and verification of developed software and hardware systems are presented.

Вступ. Проектування програмного забезпечення CAD/CAM-систем має специфічний характер, оскільки способи зображення вхідної і вихідної інформації дуже залежать від використання зовнішніх пристроїв і від стандартів з взаємодії цих систем зі зовнішніми. Від CAD/CAM-систем такаож вимагається короткий час реакції на дію користувача. Розробка і створення CAD/CAM-систем є достатньо складним і тривалим процесом, який вимагає значних затрат матеріальних і людських ресурсів. Під час проектування CAD/CAM-систем розробники стараються отримати кращі експлуатаційні характеристики за рахунок раціонального вибору структур програмних систем та структур даних.

Існує багато інтерпретацій терміна «архітектура програмного забезпечення» і «архітектура апаратних засобів» (тут ми свідомо не обмежуємося поняттям «архітектури обчислювальних засобів»). Найвдаліше визначення терміна «архітектура» було наведено вперше в 1994 році дискусійною групою з архітектури програмного забезпечення (ПО) із Software Engineering Institute. Це визначення можна поширити як на програмні, так і апаратні засоби. Архітектура – структура компонентів програми чи системи, їхнього взаємозв'язку, принципи і керівництва для їхнього проектування й розвитку в часі [1].

Більшість описів архітектури – це малюнки, у яких прямокутниками показані компоненти, що виконуються, а різними стрілками – взаємодія серед цих компонентів. Ці малюнки акумулюють досвід створення конкретної системи, що пропонується розповсюджувати на системи, подібні до проектованої. Такий підхід для практики проектування був основним приблизно до 80-х років.

У галузі мов опису апаратної частини (hardware) відомими прикладами є VHDL [2] і Verilog [3]. Мови опису hardware забезпечують найзрозуміліше подання архітектури комунікацій. Наприклад, у VHDL компоненти інтерфейсів (декларативні графічні примітиви) можуть бути об'єднані в архітектуру до того, як вони будуть зв'язані (у конфігурації) для виконання. Архітектура в VHDL може мати багато різних варіантів зв'язування компонентів (конфігурацій), але усі вони мають ту ж саму схему взаємодії. Можливі варіанти архітектур під час опису на VHDL обмежені режимом статики: кількість компонентів і їхніх з'єднань визначаються під час компіляції.

Синтез і аналіз архітектури програмних засобів. Програмна архітектура – важливий напрямок для вивчення практиків і вчених. Важливість цього напрямку підтверджується появою великої кількості робіт у таких галузях, як мови модульного інтерфейсу й опису архітектури, зразки і довідники, що формально описують проектування архітектури й архітектурне проектування інтерфейсу ПЗ.

Однак підвищення важливості розуміння архітектури ПЗ призводить до більш явного використання архітектурного проектування. Це спостерігається в таких підходах: стандартизація компонентів, спільність продуктів, платформи, доменно-орієнтована архітектура.

Використання стандартизованих компонентів існує там, де виробники ПЗ усвідомлюють, що є загальна безліч компонентів, що використовується паралельно з деякою безліччю продуктів. Приклад цього – BaseWorx [5], безліч стандартних компонентів якого формує базис для безлічі зв'язаних операцій, підтримуваних системами. Специфічні системи створюються додаванням специфічних компонентів до стандартних компонентів.

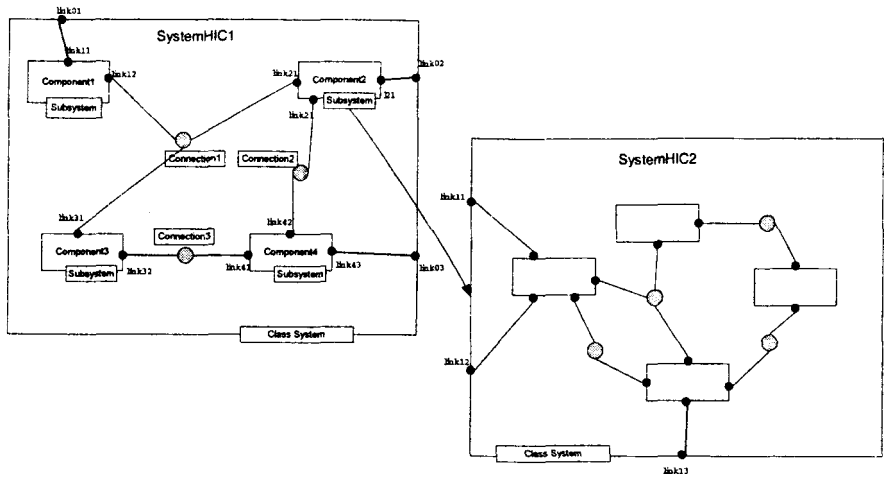
Спільність продуктів є одним із засобів нагромадження коштовних властивостей продуктів – активу. Базу активу використовують для створення низки тісно зв'язаних архітектур прикладних систем.

Платформа загалом – це безліч компонентів, що формують базис деякої розмаїтості зв'язаних продуктів. Ці компоненти є звичайно загальними характеристиками, такі як бази даних, генератори графічного інтерфейсу користувача тощо. Компоненти забезпечені засобами спеціалізації або за допомогою спеціальних декларативних мов, або мовами сценаріїв спеціального призначення. Платформа подібна до безлічі стандартизованих компонентів, але вона заповнена з великою деталізацією компонентів, що необхідна, щоб бути спеціалізованою для прикладних програмних систем.

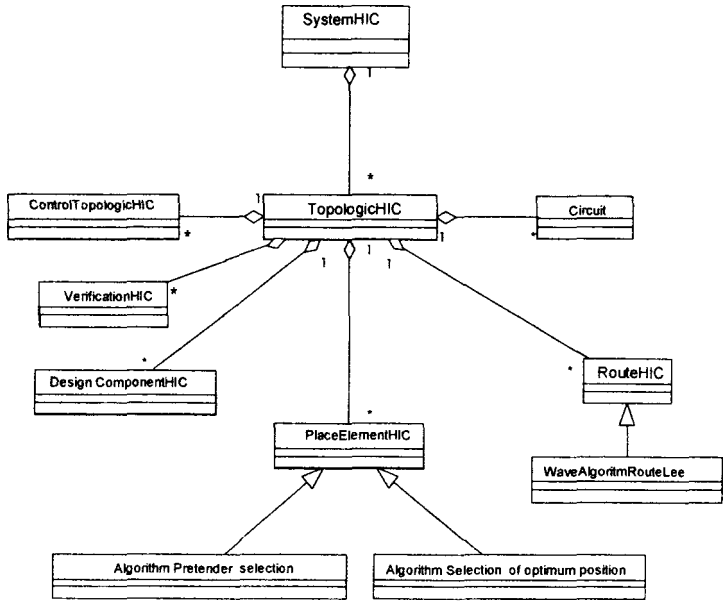
В останні роки виразно простежуються два напрямки розвитку архітектури ПЗ. Перше – проектувальники розвивають широкий набір методів, технічних прийомів, моделей і ідіом для структурування складних програмних систем. Наприклад, блок-схеми і принципові (лінійні) схеми і пояснення до них, що звичайно супроводжують високорівневі описи системи, часто посилаються до таких пристроїв і систем, як або конвеєр, або клієнт-серверна система. Другий напрямок пов'язаний з можливістю багаторазового використання частин інтегрованої системи для продуктів, що мають загальні риси. Напрямок заснований на тій ідеї, що загальні аспекти набору зв'язаних систем можуть бути виділені так, що кожну нову систему можна побудувати за порівняно низьку вартість. Прикладами цього напрямку є стандартна декомпозиція компілятора, стандартизовані протоколи обміну даними, мови програмування четвертого покоління, інструментальні засоби користувальницького інтерфейсу і структури, що забезпечують як багаторазово використовувані структури для розвитку інтерфейсів, так і безліч повторно використовуваних компонентів, таких як меню і діалогові вікна.

Узагальнюючи ці напрямки, можемо показати чотири виразних розбіжності між ними. Перша – між традиційним інтересом до проектування алгоритмів і структури даних, з одного боку, й інтересами архітектури з організації великих систем, з іншого. Зокрема у проектуванні архітектури менше цікавляться алгоритмами і структурами даних, використовуваними усередині модулів, ніж загальною організацією і глобальним керуванням структури, протоколами обміну, синхронізацією, доступом до даних,

призначенням функціональності до проектованих елементів, фізичним розміщенням, об'єднанням проектованих елементів, масштабуванням і зображенням, і вибором серед проектованих альтернатив.



а)



б)

Рис. 1. Зображення системи, яка моделюється:
а) – концептуальна модель,
б) – об'єктна модель в нотації UML

Друга розбіжність полягає в способі зображення – між описом системи, заснованому на визначенні-використанні компонентів структури, і архітектурним описом, заснованим на графах взаємодіючих компонентів. У минулому розбивання системи на модулі в термінах вихідного коду звичайно пояснює залежність між використовуваними місцями коду і відповідними місцями визначення. У сучасному підході модульність системи розглядається як граф чи конфігурація компонентів і з'єднань. Компоненти визначають прикладний рівень обчислень і дані системи, які зберігаються. Наприклад, клієнти, сервери, фільтри, бази даних і об'єкти. З'єднання визначають взаємодії між цими компонентами. Ці взаємодії можуть бути такими простими, як виклики процедур, передача чи повідомлення, і складнішими, а також клієнт-серверні протоколи, протоколи доступу до баз даних тощо.

Третя розбіжність – між архітектурною реалізацією й архітектурним стилем. Архітектурна реалізація програми належить до архітектури специфічної системи. Блок-схеми і принципові схеми, що супроводжують системну документацію, описують архітектуру програми, оскільки вони застосовні до індивідуальних програм. Архітектурний стиль визначає внутрішню несуперечність форми і структури сукупності архітектурних реалізацій, що мають загальні риси. Архітектурний стиль описує, наприклад, словник компонентів і зв'язку (наприклад, фільтри і канали), топологічні обмеження, що забезпечують внутрішню несуперечність (наприклад, граф повинний бути без циклів) і семантичні (наприклад, фільтри не можуть спільно використовувати стан).

Четверта розбіжність – це розбіжність між методами проектування ПО (такими як об'єктно-орієнтоване проектування і структурний аналіз) і програмною архітектурою. Хоча обидва методи проектування й архітектура пов'язані з проблемою подолання розриву між вимогами і реалізацією, існує важлива розбіжність між їх сферами інтересу. В ідеалі метод проектування визначає кожен ступінь, що здійснює проектувальник системи від вимог до рішення. Програмна архітектура вивчає простір архітектурних проектів. У середині цього простору об'єктно-орієнтовані структури і структури потоків даних є двома з безлічі можливих.

Проектування архітектури великих систем завжди мало важливе значення. Принципове використання програмою архітектури може позитивно вплинути принаймні в п'ятьох аспектах розвитку ПЗ.

1. Розуміння – програмна архітектура підсилює нашу здатність осягти, охопити великі системи зображенням їх на такому рівні абстракції, на якому проекти великої складності можуть бути зрозумілі.

2. Багаторазове використання – опис архітектури підтримує багаторазове використання на багатьох рівнях і як правило на компонентах бібліотек. Проектування архітектури підтримує як багаторазове використання великих компонентів, так і структур, в яких компоненти можуть бути інтегровані.

3. Розвиток – програмна архітектура дає змогу визначити межі, уздовж яких очікується розвиток системи. Крім того опис архітектури допомагає розділити те, що зв'язано з функціональністю компонентів, від способу яким ці функціональні елементи з'єднані (тобто взаємодіють) з іншими компонентами. Це дозволяє змінювати механізм з'єднання під керуванням еволюції зображення, багаторазового використання тощо.

4. Аналіз – опис архітектури дає нові можливості для аналізу, а також високорівневі форми системно несуперечливого контролю, відповідність архітектурному стилю і якості атрибутів, доменно-специфічний аналіз для архітектури, що погодиться зі специфікою стилів.

5. У керуванні – тепер зрозуміло, що досягнення життєздатної програмної архітектури є ключовою віхою в індустріальному процесі розвитку ПО. Архітектура системи повинна будуватися з урахуванням очікуваного зростання системи, її розмірності, задовольняючи початкові вимоги і передбачаючи напрямок зростання.

Дослідження в галузі програмної архітектури найактивніше здійснюються в таких галузях:

– мови опису архітектури;

– формальне обґрунтування програмної архітектури;

– техніка архітектурного аналізу;

– розвиток методів архітектури;

– регенерація архітектури і перепроектування;

– кодифікація і посібники в галузі архітектури;

– засоби й оточення архітектурних проектів.

Регенерація архітектури і перепроектування. Для великих систем із тривалим часом життя можливість керувати, модифікувати вихідний код ПЗ є критичним.

Дослідження в цій галузі вирішують такі питання, як:

– виділення архітектурних проектів з існуючих систем;

– уніфікація зв'язаних архітектурних проектів;

– абстракція, узагальнення, конкретизація доменно-специфічних компонентів і структур (термін домен належить до класу додатків);

– здатність до взаємодії – техніка для визначення несполучення компонентів і переносимість програм (і даних) з однієї ЕОМ на іншу.

Можливість повторного використання програмного коду, один з основних принципів об'єктної ідеології, залишається нездійсненною мрією розроблювачів протягом уже двох десятків років – з тих пір, як на ринку з'явилися хороші комерційні версії об'єктно-орієнтованих мов програмування. Особливо великі надії покладалися на C++. Здавалося, що компанії випустять на його основі безліч типових об'єктів і програми можна буде збирати з готових блоків. Однак нездійснена ідея повторного використання об'єктів виявилася одним з найбільших розчарувань в об'єктно-орієнтованому програмуванні. Це сталося внаслідок того, що у світі поширена велика кількість несумісних платформ. Розроблювачам великих інформаційних систем вимагаються цеглинки, здатні функціонувати на різних операційних системах. Але придатної об'єктної кросс-платформенної технології не знайшлося. Мова Java поки не виправдовує надій – складна. Java-прикладання працюють занадто повільно. Програмісти побоюються, що через відсутність незалежного стандарту, як у разі з Microsoft Windows, розвиток Java буде залежати тільки від Sun. Крім того, існує величезна кількість старих, але активно експлуатованих додатків для Unix і мейнфреймів, а переписувати їх наново дуже дорого.

На початку 90-х років стала очевидною потреба в новій технології створення розподілених додатків будь-якого масштабу, незалежних від платформи, здатних працювати в зв'язуванні зі старим ПЗ і легко нарощуваних додаванням будівельних блоків за принципом «підключи і працюй». Такі блоки повинні бути невеликими, що легко стикуються один з одним і дозволяють інтеграцію з Internet програмними модулями, що могли б випускати різні компанії. При цьому бажано, щоб робота цих модулів не залежала від конфігурації мережі і конкретної клієнт-серверної архітектури.

Необхідно спеціально проектувати від початку ПЗ багаторазового використання для того, щоб виконати принцип багаторазового використання. Розглянемо на прикладі основні риси програмної архітектури, що володіє властивістю багаторазового застосування.

Ієрархічна орієнтована на специфічні галузі архітектура, заснована на сполученнях поділу за рівнями сукупності логічно зв'язаних чи засобів понять, потоків даних і робітників галузей. Пропонована програмна архітектура DSSA орієнтована на додатки для широкого класу адаптивних інтелектуальних систем: автономні роботи (промислового й офісного застосування), моніторингові системи (медичні, для енергетичних об'єктів тощо), компоненти ПЗ для зв'язку й обміну інформацією (e-mail manager, meeting manager тощо). Запропонована архітектура DSSA містить у собі:

– архітектуру посилань, що описує загальну інтегровану комп'ютерну систему, для важливих класів;

– бібліотеку компонентів, що містить ПЗ багаторазового використання;

– метод прикладної конфігурації для автоматичного виділення і конфігурування цих компонентів усередині архітектури.

Висновки. Можна вважати, що саме об'єктно-орієнтований підхід забезпечує необхідну функціональність і інструментальність системного середовища, яке розробляється. Розробка середовища САПР ГІС "TOPOS" з використанням об'єктно-орієнтованого проектування дає можливість ефективніше розробляти та використовувати програмні модулі системи, а також створити систему, яка буде відповідати міжнародним стандартам САПР.

1. D.Garlan, D.E.Perry. IEEE Transactions on Software Engineering. Vol. 21, No. 4, 1995, pp. 269–274. 2. IEEE, IEEE Standard VHDL Language Reference Manual, IEEE Standard 1076-1987, Mar. 1987. 3. D.E. Thomas and P.R. Moorby. The Verilog Hardware Description Language. – New York: Kluwer-Academic, 1991. 4. Крон Г. Исследование сложных систем по частям (диакоптика). –М.: Наука, 1972. 5. Михалевич В.С., Волкович В.Л. Вычислительные методы исследования и проектирования сложных систем. – М.: Наука, 1982. 6. Молчанов А.А. Моделирование и проектирование сложных систем. - К.: Выща шк., 1988. 7. В.А. Селютин, Машинное проектирование электронных устройств. – М.: Сов. Радио, 1977. 8. Г. Буч, Д. Рамбо, А.Джекобсон, Язык UML. Руководство

пользователя: – М.: ДМК, 2000. 9. Г. Буч, Объектно-ориентированный анализ и проектирование с примерами приложений на С++. – 2-е изд. (Пер. с англ.). – М.; СПб, 1998. 10. Д. Корпильов, С. Ткаченко, Об'єктно-орієнтована методологія розробки середовища САПР Гібридних інтегральних схем "TOPOS" // Вісник держ. ун-ту "Львівська політехніка". "Радіоелектроніка та телекомунікації". Львів 2000. № 387

УДК 681.3

В.І. Каркульовський, А.Б. Керницький, І.І. Мотика, І.І. Чура
Національний університет "Львівська політехніка",
кафедра САПР

ПОБУДОВА АРХІТЕКТУРИ НАВЧАЛЬНОЇ САПР

Розглянуто узагальнений підхід до побудови універсальної архітектури навчальної САПР.

The general approach to development of educational CAD multipurpose architecture.

Вступ. Аналіз існуючих підходів до побудови навчальних САПР показує, що більшість з них мають жорстку архітектуру і є закритими системами, використовуються тільки для вирішення проблем у вузькій предметній області.

Мета роботи полягає в розробці узагальненої архітектури навчальних САПР, під час реалізації якої користувач, котрий володіє експертними знаннями, зміг вирішувати широкий клас системних задач.

1. Розробка архітектури навчальної САПР. Концептуальна схема є ядром навчальної САПР. Вона є мовою, що використовується в навчальній САПР для опису виділених типів систем, вимог і задач. Коли йдеться про архітектуру, то фіксується тільки епістемологічна ієрархія системних типів. Решта понять, такі як типи вимог або методологічні відмінності систем, описуються в архітектурі навчальної САПР тільки в загальних термінах. Точніший їх опис є предметом конкретної реалізації навчальної САПР, що базується на конкретній множині системних типів, кожний з яких визначається сукупністю епістемологічних і методологічних особливостей, а також конкретною множиною типів вимог і, як наслідок, типами задач; вони називаються допустимими типами систем, вимог і задач (тобто вони допустимі для даної реалізації навчальної САПР).

Концептуальна схема – це лінгвістичне середовище, в якому користувач спілкується з навчальною САПР, що складається з відповідної бази знань, множини метаметодологічних засобів, а також блока управління. Зв'язок користувач – навчальна САПР двосторонній і з кожної сторони забезпечений відповідним інтерфейсом. Назвемо інтерфейс з боку користувача зовнішнім інтерфейсом, а з боку навчальної САПР – внутрішнім інтерфейсом.

Виділимо два типи зовнішнього інтерфейсу, кожний з яких може входити в конкретну реалізацію навчальної САПР. Перший спроектований з розрахунку на досвідченого користувача, який достатньою мірою знайомий з концептуальною схемою НСАПР і обмеженнями тієї реалізації НСАПР, яку він збирається використовувати. Цей тип інтерфейсу базується на припущенні, що користувач не потребує допомоги при формальному визначенні його систем і вимог, і отже, єдина функція інтерфейсу полягає в перевірці на можливі невідповідності у формулюваннях користувача. Інший тип зовнішнього інтерфейсу спроектований з розрахунку на звичайного користувача, чиє знання концептуальної схеми НСАПР є недостатнім. Функція цього інтерфейсу полягає не тільки в перевірці можливих невідповідностей у формулюваннях користувача, але і в тому, щоб надати користувачу широкий спектр послуг при формулюванні його задачі. Це означає, що зовнішній інтерфейс повинен містити відповідні процедури опиту для ідентифікації типів систем і вимог, а також конкретних систем і вимог заданих типів. Такі процедури можуть виявитися дуже складними (а може бути, і абсолютно нереальними) за умови, що користувач нічого не знає про концептуальну схему НСАПР. Тому від звичайного корис-