УДК 004.89

Є. Буров
Національний університет "Львівська політехніка",
кафедра інформаційних систем та мереж

ЗАСТОСУВАННЯ ВИКОНУВАЛЬНИХ МОДЕЛЕЙ ДЛЯ ПРОЕКТУВАННЯ СЕРВІСНО-ОРІЄНТОВАНИХ ІНТЕЛЕКТУАЛЬНИХ ІНФОРМАЦІЙНИХ СИСТЕМ

Запропоновано підхід до побудови виконувальних моделей сервісів інформаційних систем. Цей підхід спирається на існуючу методологію виконувального UML (xUML), але на відміну від нього використовує інтерпретовані, а не компільовані моделі, подані мовою XML.

An approach for building executable models of intellectual services in information systems is proposed. This approach is based on well known xUML methodology, but uses interpretable, not compiled models written in XML.

Постановка проблеми та аналіз останніх досліджень

Гострою проблемою у розвитку інформаційних систем (ІС) є їх дедалі більша складність, що ускладнює розв'язання задач проектування, налаштування таких систем та керування ними [1].

У роботі [2] запропоновано методологію проектування ІС, яка спирається на ієрархічні рівні моделювання, визначені як рівень бізнес-систем, рівень сервісів та рівень процесорів та пристроїв. При цьому використовується модельно-орієнтований підхід до проектування (MDD – Model Driven Design), головною ідеєю якого є застосування моделей для проектування, керування та побудови ІС [3].

У цій роботі розглянуто підхід для побудови виконувальних моделей, використовуваних сервісами ІС.

Для побудови моделей, які використовуються інтелектуальними сервісами, зокрема моделей виконання запитів, необхідно обрати мову, використовувану для подання структури моделі, порядку її роботи, взаємозв"язку частин. Така мова повинна відповідати таким вимогам:

- мати еквівалентні графічні та текстове подання – для опрацювання людиною та машиною;
- мати розвинені засоби відображення різних аспектів функціювання сервісу;
- бути стандартизованою міжнародними організаціями;
- бути гнучкою, з можливістю розширення та доповнень;
- мати визначену (стандартизовану) семантику;
- давати змогу будувати виконувальні моделі

Враховуючи те, що сервіси ІС переважно реалізуються програмно, при виборі мови подання моделей доцільно зупинитися на мові UML [4] (або SysML), яка є доволі поширеною під час проектування інформаційних систем.

Зокрема, для подання моделі виконання запиту доцільно використовувати діаграми діяльності (activity diagrams), різнотипні діаграми взаємодії (interaction diagram), а також діаграми станів.

З іншого боку, структурні діаграми UML використовують для подання моделей концептів, використовуваних під час моделювання. Наприклад, у моделі сервісу завантаження файлів такими структурними моделями (класами) можуть бути моделі сайта, провайдера зв'язку. Вони інкапсулюють знання про важливі сутності, що впливають на виконання сервісом своїх функцій. Засобами структурних моделей також відображають результати виконання діяльностей, ресурси, інші матеріальні та інформаційні об'єкти, які використовуються в процесі виконання запитів.

Сьогодні UML найчастіше використовують для початкового, концептуального проектування системи. Такі діаграми є засобом документування проектних рішень та засобом порозуміння між

розробниками. Часто буває так, що реальна система у процесі кодування перестає відповідати початковим UML – моделям, а самі моделі стають застарілими та непотрібними. У кращому випадку UML – моделі використовують для створення початкових програмних шаблонів, а надалі реальна система та набір моделей, створених на початку процесу розроблення, перестають відповідати один одному. У загальному випадку UML- моделі не є виконувальними.

Зазначені вище проблеми значною мірою вирішені у межах Виконувального UML (Executable UML, xUML). Основи xUML були закладені в роботах Шлаера (Shlaer) у 80-ті роки XX століття та сьогодні продовжені в роботах Меллора (Mellor) [5].

Особливістю розвитку концепції OOD (Object-Oriented Design) у xUML є детальне опрацювання UML моделей, яке ґрунтується на стантартизованій семантиці UML[6]. Підхід xUML переводить процес створення програмних систем на новий рівень абстракції – у ньому розглядають платформно-незалежну бізнес-логіку системи, абстрагуючись від суто програмних рішень (таких, як багатопотоковість, використання пам'яті, взаємодія компонент) подібно до того, як мова програмування високого рівня абстрагується від рішень щодо використання регістрів, стекових архітектур тощо. Комплекс взаємопов'язаних моделей компілюється спеціальним компілятором, який враховує вимоги щодо продуктивності системи та платформи реалізації, у виконувальний код. Цей код не потребує жодного ручного опрацювання та модифікації.

На початку проектування предметна область поділяється на низку підобластей (доменів). Для кожного домену визначають свій словник концептів та будують модель. Домени не перетинаються, чим спрощується процес проектування.

Залежності між моделями з різних доменів описують з використанням мостів, які формулюють додаткові обмеження. Моделі взамодіють одна з однією шляхом обміну повідомленнями та генеруванням сигналів.

При побудові моделей значну увагу приділяють питанням синхронізації моделей, визначенню життєвого циклу об'єктів моделювання та пов'язаним з цим змінам у параметрах, структурі, обмеженнях.

Для моделювання концептів (об'єктів) предметної області використовують діаграми класів. Динаміка поведінки об'єкта, а також зміна об'єктів та класів у часі подаються діаграмами станів. Виконувальний характер специфікації xUML визначають власне машини станів – з кожним переходом і зміною стану машини асоційовано виконувальний код. Інші діаграми (активностей, взаємодії) мають додатковий, роз'яснювальний характер при проектуванні. Діаграми прецедентів (use-case) традиційно використовують для початкового формулювання вимог до системи та визначення доменів.

Виконувальний код записують формально з використанням спеціалізованої мови дій (Action language), яка оперує поняттями, визначеними у цьому домені.

Готовий комплекс моделей верифікують, перевіряючи структурну та функціональну коректність моделей, відповідність початковим вимогам.

Для перетворення розробленої платформно-незалежної моделі у виконувальний код застосовують спеціальний компілятор. Компілятор враховує типові механізми реалізації функцій на визначеній платформі. На виході компілятора отримують виконувальний код. Для об'єднання моделей з різних доменів та створення коду компілятор використовує архетипи та мову архетипів.

Після компіляції отриманий продукт тестують, і якщо тестуванням виявлено дефекти, то, залежно від типу виявленого дефекту, повертаються до одного з етапів побудови моделей. Характерною особливістю методології xUML є те, що жодного кодування вручну не допускається.

Методологія xUML була випробувана у десятках проектів з розроблення програмних систем [5].

Особливості реалізації проектування моделей сервісів

Методологію xUML пропонується покласти в основу побудови моделей інтелектуальних сервісів. Однак запропонований підхід істотно відрізняється від xUML, зокрема:

• xUML використовують для створення програмних систем. Результатом проектування є завершений програмний продукт – двійковий код. У цій роботі пропонується архітектура побудови інтелектуальних сервісів, яка безпосередньо визначає порядок роботи сервісу, а не підхід до створення програм;

• xUML використовує компілятор для перетворення моделей на код. Запронований підхід передбачає інтерпретатор моделей. Будь-яка зміна у моделі відразу ж впливає на процес роботи системи. У xUML аналогічна зміна в моделях вимагає перекомпіляції та повторного розгортання системи, що вимагає часу.

• у запропонованому підході, на відміну від xUML, не ставиться задача побудови програмної системи. Натомість пропонується в процесі виконання моделі звертатися до визначених програмних модулів. Ці модулі виконують базові функції у системі, призначені для повторного використання та працюють як сервіси. Зовнішні програмні продукти також подаються як сервіси (рис. 1).

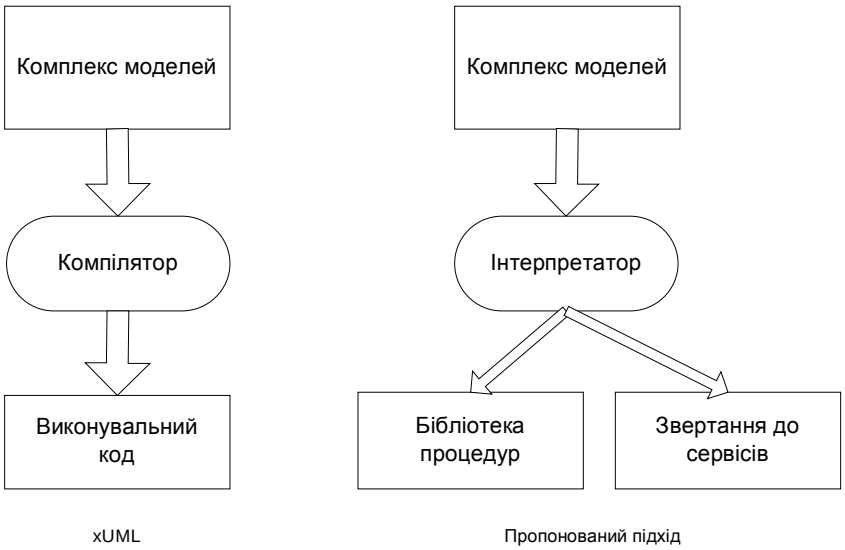

Рис.1. Компіляція/інтерпретація в xUML та у запропонованому підході

Виконувальний та інтерпретований характер опрацювання моделей у запропонованому підході створює ряд додаткових переваг порівняно з традиційними методами розроблення програмного забезпечення, і з xUML:

• Алгоритм виконання, який у традиційних методах розроблення ПЗ був жорстко прописаний у коді, у запропонованій системі подається як набір правил, даних та фрагментів коду. При цьому фрагменти коду, як правило, не несуть у собі бізнес логіки і мають просту структуру. Тобто стає неможливим відрив моделі від реальної системи, оскільки модель завжди визначає, як функціонує система.

• Інтерпретований характер опрацювання моделі дає змогу просто вносити в неї зміни, змінюючи поведінку системи без грунтовної її перебудови

• Виконувальність моделі дає змогу використати знання, інкапсульовані у моделях, правилах, онтологіях для збільшення інтелекту системи та її адаптаційних можливостей;

• Наявність виконувальної та релевантної у будь-який час моделі дає змогу опублікувати інформацію про модель та використати її в операціях аналізу та оптимізації самої моделі.

Відмінності між запропонованим піходом та методологією xUML узагальнені у таблиці.

Для побудови моделей виконання запитів планується використовувати такі діаграми та моделі UML:

• Діаграми прецедентів (use-case) використання сервісу. Використовуються на стадії аналізу та виявлення комплексу запитів, які опрацьовує сервіс. Як правило, модель виконання запиту відповідає одному з прецедентів використання сервісу.

Порівняння запропонованого підходу з xUML

Властивість	xUML	Запропонований підхід
Об’єкт проектування	Закінчена програмна система за замовленням певного бізнес-клієнта	Модель виконання запиту сервісом
Етап аналізу	Для аналізу вимог використовують діаграми прецедентів (use-case)	Модель виконання запиту відповідає одному з прецедентів використання
Етап розроблення моделей для окремих доменів	Проводиться декомпозиція предметної області на домени. Для кожного домена розробляється свій словник понять. Будується модель для кожного домена	Визначаються вимоги до виконання запиту, ресурси, результати виконання, процесори, задіяні у виконанні запиту. Визначають, чи потрібні звертання до зовнішніх сервісів та яких. Визначають, які процедури потрібні для виконання запиту. Будують ресурсну модель, яка відображає залежність між параметрами запиту та вимогами до ресурсів. За потреби визначають альтернативні моделі виконання запиту та умови їх застосування.
Етап узгодження моделей	Моделі різних доменів узгоджуються – формулюються додаткові обмеження та вимоги, визначаються операції обміну повідомленнями, синхронізації. Розглядається життєвий цикл об’єктів.	Розглядаються всі моделі сервісу. Узагальнюються вимоги до ресурсів та зовнішніх сервісів. Визначаються допоміжні моделі та механізми підтримки роботи сервісу загалом. Проектування інтелектуальних компонент.
Розробка специфікації на мові дій	Розробляють детальну специфікацію виконання кожної моделі мовою дій	Уточнення та деталізація моделей до рівня, при якому можливе їх автоматичне виконання інтерпретатором.
Виконання моделі	Використовується компілятор	Використовується інтерпретатор

• Діаграми класів (class). Інкапсулюють інформацію про всі статичні об’єкти, що використовуються під час моделювання – сервіси, моделі, ресурси, запити та їхні залежності. Екземпляри відповідних класів використовуються і для подання елементів динамічних моделей – активностей, станів, переходів, вузлів прийняття рішення.

• Діаграми активностей (activity). Використовуються для подання динаміки проведення операцій. Під активністю розуміють етап виконання запиту, під час якого отримують важливий кінц планування та аналізу процесу виконання роботи як набору пов’язаних операцій. З діаграмою активностей асоціюють необхідні та задіяні у процесі виконання ресурси, процесори.

• Діаграми станів (state). Вживаються для подання процесу виконання активності та використовуються інтепретатором моделей для виконання моделей (аналогічно, як в xUML).

Моделювання активностей та модель станів залежні. Як правило, модель станів будують для моделювання процесу виконання активності. Отже, модель станів існує у контексті певної активності. Якщо активність доволі складна, проводять її декомпозицію та розробляють відповідну діаграму активностей. Модель станів будують тільки для активності, яка не підлягає декомпозиції.

Приклад фрагмента діаграми станів наведено на рис.2.

Діаграма станів містить стани та переходи між ними. Для кожного стану визначено передумови переходу у цей стан, набір операцій, що виконуються при переході у стан, набір асоційованих об’єктів. При виконанні умови виходу зі стану переходять у вузол (junction point) прийняття рішення. У загальному випадку цей вузол синхронізує закінчення декількох станів (у прикладі – стани 1,2,3). У вузлі прийняття рішення залежно від параметрів, які характеризують роботу системи, визначають переходи у наступні стані з відповідними параметрами.

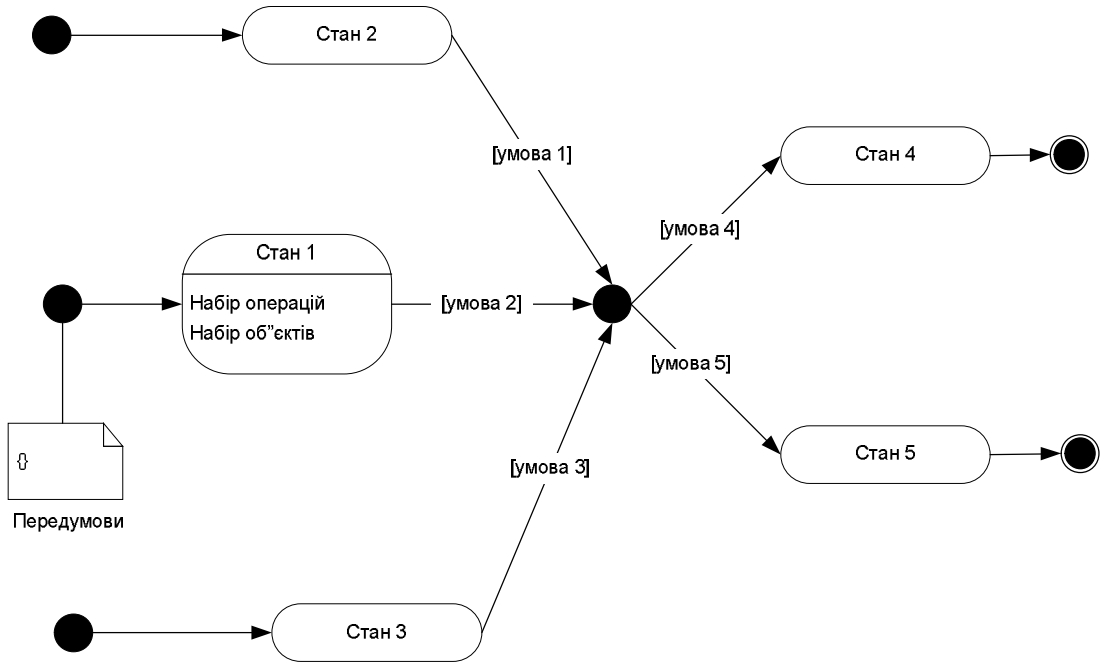

Рис. 2. Приклад діаграми станів

Наприклад, клас, що визначає стан, складається з таких елементів:

$$St = \{IdSt, NmSt, LiPmSt, Li\,\mathrm{Pr}\,cSt, LiCdSt, OuSt\},$$

де $IdSt$ – ідентифікатор, $NmSt$ – назва, $LiPmSt$ – параметри, $Li\,\mathrm{Pr}\,cSt$ – передумови, $LiCdSt$ – посилання процедури (код), $OuSt$ - умови переходу у інші стани.

Вузол прийняття рішення визначається так:

$$Dc = \{IdDc, NmDc, InpStDc, OutStDc, RuInStDc, RuOuStDc\}$$

де $IdDc$ – ідентифікатор вузла, $NmDc$ -назва, $InpStDc, OutStDc$ -множини вхідних та вихідних станів, $RuInStDc, RuOuStDc$ - вхідні та вихідні правила переходу.

При використанні UML – моделей для проектування інтелектуальних систем необхідно додати нові дані до кожного елемента моделі. Така інформація потрібна для логічного висновку щодо елементів та моделі загалом, прийняття рішень, а також для виконання моделей.

Інтелектуальна складова, наприклад стану, визначається так:

$$IntSt = \{LiRuSt, LiMoSt, LiRsSt, LiOnSt\},$$

де $LiRuSt$ – обмеження, правила, що визначають застосування коду та вибір параметрів цього застосування, $LiMoSt$ - посилання на релевантні моделі, $LiRsSt$ – посилання на задіяні ресурси, $LiOnSt$ – посилання на релевантні онтології.

Аналогічна інформація асоціюється з іншими компонентами діаграм UML.

Початковими даними для формування моделі сервісу є множина варіантів (прецедентів) використання сервісу (use-cases) або набір можливих запитів до сервісу.

Для кожного запиту (прецеденту) будують модель реалізації запиту. З кожним елементом пов'язані дані, правила, знання, ресурси, процесори та ін. До моделі виконання запиту входять зв'язки між елементами, операції прийняття рішення, посилання на артефакти.

Для початкового формування моделі виконання запиту ми рекомендуємо використовувати готові шаблони, які відображають знання щодо готових проектних рішень.

Готову модель верифікують та валідують. При цьому перевіряють відсутність тупикових станів, врахованість усього діапазону можливих значень параметрів. Також перевіряють виконання усіх початкових вимог за моделлю прецедента використання.

Модель відображається мовою XML відповідно до підходів, використаних у [7, 8] та є готовою для опрацювання інтерпретатором. Перед використанням розроблену модель тестують на відповідність початковим вимогам та варіантам використання.

Висновки

Запропонований у роботі підхід до побудови виконувальних моделей сервісів інформаційних систем дасть змогу створювати системи, які порівняно з існуючими швидше адаптуються до змін у навколішньому середовищі за зменшення видатків на проектування та супровід.

1. Balmelli L, Brown D, Cantor M, Mott M. Model-driven systems development [Text] / Balmelli L, Brown D, Cantor M, Mott M // IBM Systems Journal .- 2006.-, vol 45, # 3. 2. Буров Є.В. Система формальних специфікацій для проектування розподілених інформаційних систем [Текст]/Буров Є.В.// Вісник держуніверситету ‘ Львівська політехніка‘ ‘Інформаційні системи та мережі’.- 2000.- № 406. 3. MDA Distilled. Principles of Model Driven Architecture, Stephen Mellor, Kendall Scott, Axel Uhl, Dirk Weise.[Text].- Addison-Wesley Professional.- 2004. 5. Stephen J. Mellor, Marc J. Balcer. Executable UML: A foundation for model-driven architecture [Text]./Mellor S, Balcer M.- Addison Wesley,2002.- 416p.-ISBN 0-201-74804-5 6. OMG. Semantic profile for UML[Electronic resource]. - Mode of access: URL: http://www.omg.org/cgi-bin/doc?formal/03-03-01. 7. Avesha Malik. Design XML schemas using UML. [Electronic resource].- Mode of access: URL: http://www.ibm.com/developerworks/library/x-umlschem/.-Title from the screen. 8. Benoit Marshal. Working XML: UML,XMI and code generation. [Electronic resource]. – Mode of access: URL: http://www.ibm.com/developerworks/xml/library/x-wxxm23/.- Title from the screen.

УДК 622.02.658.284

Б. Демида
Національний університет “Львівська політехніка”,
кафедра автоматизованих систем управління

МЕТОД РОЗПІЗНАВАННЯ ШТРИХ-КОДОВОЇ ІНФОРМАЦІЇ (Ч. ІІ)

Розглянуто метод автоматичного розпізнавання штрих-кодової інформації зі звітних листків ідентифікації виробничих операцій виконаної роботи для системи автоматизації обліку на швейному виробництві.

This paper considers the method of automatic recognition of bar code information from reports of identification of production operations of the performed work for the account automation system in sewing industry.

1. Загальний опис алгоритму

У першій частині статті [1] описано алгоритм пошуку та локалізації штрих-кодових об’єктів із застосуванням таких кроків:

- сегментація графічного образу листка формату А4 з наклеєними мітками на окремі графічні елементи з перевіркою наявності на цих елементах бар-кодів за допомогою функції фільтрації випадкових величин.

У цій статті опишемо:

1. Алгоритм визначення граничних областей, графічного центру та кута повороту елементу бар-коду. Для реалізації цього алгоритму застосуємо:

- періодичне n-разове сканування по вертикалі локалізованої області з певним кроком із застосуванням методу пошуку;