висновок про доцільність використання цього розширення у веб-переглядачі та методу логістичної регресії для валідації посилань на сервері. Варто зауважити, що цей додаток можна використовувати спільно з додатком AdBlock, вони не конфліктують і захищають від більшої кількості загроз. Також можна використовувати веб-сервер для перевірки шкідливості того чи іншого ресурсу.

1. VMware: the new Redis home [Електронний ресурс]. – Режим доступу: http://antirez.com/post/vmware-the-new-redis-home.html. 2. Документація Redis "Virtual Memory" [Електронний ресурс]. – Режим доступу: http://redis.io/documentation. 3. Nodejs [Електронний ресурс]. – Режим доступу: https://nodejs.org/. 4. NaturalNode документація [Електронний ресурс]. – Режим доступу: https://github.com/NaturalNode/natural/blob/master/README.md. 5. Express.js [Електронний ресурс]. – Режим доступу: http://expressjs.com/. 6. Node redis документація [Електронний ресурс]. – Режим доступу: https://github.com/mranney/node_redis. 7. The Little Redis Book [Електронний ресурс]. – Режим доступу: https://github.com/karlseguin/the-little-redis-book/blob/master/en/redis.md. 8. Mike Cantelon, Marc Harter, T. J. Holowaychuk and Nathan Rajlich. Node.js in Action // Manning Publications October. – 2013 – 416 с.

УДК 004.89;004.048;004.416.3

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

ПРОГРАМНІ СИСТЕМИ НА БАЗІ ОНТОЛОГІЧНИХ МОДЕЛЕЙ ЗАДАЧ

Розглянуто застосування онтологічних моделей задач для побудови програмних систем, здатних адаптуватися до зміни стану предметної області. Розроблено математичну формалізацію подання та опрацювання знань у системі, архітектуру та принципи функціонування програмної системи на базі онтологічних моделей. Розглянуто методи використання онтологічних моделей для розв'язання практичних задач. Проаналізовано переваги застосування онтологічних моделей порівняно з традиційним підходом до побудови програмних систем та розроблено формули для оцінювання їх переваг. Розроблено програмний інструментальний засіб для побудови та моделювання програмних систем на основі онтологічних моделей задач.

Ключові слова: онтологія, онтологічна модель, програмна система, адаптація.

The increased mobility of business processes today relies on extensive use of software and in turn puts high demands on the quality of software and its ability to be quickly and accurately adapted to the changes in the business environment. A promising approach to solving the problem of adapting the software to changes in its operational environment is the use of ontological modeling. In the article the theoretical principles of knowledge representation and processing in software based on ontological task models were developed. A formal model of knowledge representation was built. The method of using ontological task models for automated testing of nightly software builds was developed. This method also demonstrates the organization of models interaction in the multistage business process of automated testing.

Key words: ontology, ontological model, software system, adaptation.

Вступ та постановка проблеми

Аналіз тенденцій розвитку програмних систем показує, що темпи їх змін та складність продовжують зростати. Це відповідає загальним трендам глобалізації у світовій економіці та необхідності швидко реагувати на зміни у бізнесовому середовищі. Підвищення мобільності бізнес-

процесів передбачає широке використання програмних систем та технологій і ставить високі вимоги до їх здатності швидко та безпомилково адаптуватися до змін.

Архітектури програмних систем та підходи до їх проектування поки що не повною мірою вирішують цю проблему. Як наслідок, частка IT-бюджетів, які витрачаються на супровід та підтримку програмних систем, постійно збільшується (від 67 % у 1979 р. до 90 % у 2000 р.) [1]. Зазначену проблему вперше опрацював Леман у межах так званої теорії еволюції програмних систем. Дослідник виділив три класи систем (P, S та E) та сформулював закони еволюції програмних систем. Зокрема, для систем класу E (які відображають процеси зовнішнього світу та повинні постійно відповідати стану середовища) зазначено (закони 2 та 7), що з часом зростає складність та погіршується якість таких систем [2].

Причинами високого рівня складності розв'язання задачі модифікації програмної системи є недоліки та вузькі місця, притаманні архітектурам таких систем та методам їх побудови. Зокрема, у таких системах правила та алгоритми функціонування жорстко зашиті у коді; існує відрив системи бізнес-процесів від програмної системи, яка забезпечує її функціонування; проектування програмної системи ґрунтується на фіксованому наборі вимог до системи, які зазвичай визначені нечітко.

Отже, актуальним є розроблення нових архітектурних принципів, методів та засобів, спрямованих на спрощення побудови та модифікації програмних систем

Аналіз останніх досліджень і публікацій

На вирішення сформульованої вище науково-прикладної проблеми спрямовані дослідження за декількома теоретичними та технологічними напрямами.

Зокрема, в галузі технологій розроблення програмного забезпечення вплив нечіткості вимог до програмної системи нівелюється використанням методів гнучкої розробки (Agile programming). У них розроблення проводиться як послідовність коротких проектів з визначеними вимогами. Перед початком кожного нового проекту вимоги знову аналізують та уточнюють. Водночас такі методології розроблення є ефективними для створення порівняно простих програмних систем.

Загальноприйнятим принципом врахування вимог бізнес-середовища функціонування є моделювання бізнес-процесів та використання його результатів для побудови програмної системи.

З метою урахування специфіки бізнес-процесів у архітектуру програмної системи вводиться рівень бізнес-логіки (Microsoft .NET WPF application framework, Domain driven design). Водночас таке запровадження зазвичай є статичним та враховує лише стан бізнес-правил на певний момент часу, реалізація ж бізнес-правил як частини загального коду системи суттєво ускладнює її модифікацію [3].

У модельно-орієнтованих підходах реалізовано відокремлення алгоритму роботи програмної системи від системи опрацювання цього алгоритму. Такий підхід запропонували Меллор та Шлаер у методології xUML [4], в якій спочатку будують модель програмної системи, яка потім компілюється у код. Але практичне застосування цього підходу показало, що його складність є співмірною зі складністю створення програмної системи традиційним способом [5].

У роботах Росса, Хея (Ross, Hay) та деяких інших авторів [6] запропоновано декларативний опис та збереження бізнес-правил в окремій спеціалізованій системі. Частиною програмної системи є так звана машина правил для контролю виконання бізнес-правил. Такий підхід дає змогу гнучко налаштовувати систему в разі зміни бізнес-правил. Його обмеженість зумовлена неможливістю подати усі особливості бізнес-середовища у вигляді правил, відсутність декларативного опису середовища [7].

Перспективним напрямом у вирішенні проблеми адаптації програмної системи до змін середовища є використання підходів інженерії знань, зокрема онтологічного моделювання. На відміну від модельних підходів, орієнтованих на компільовані моделі чи опрацювання правил, у підході онтологічного моделювання будують формальну модель предметної області (онтологію), з урахуванням її особливостей та обмежень. Така модель може бути повторно використана для побудови інших програмних систем для цієї ж предметної області.

Узагальнені результати аналізу відомих програмних архітектур щодо моделювання предметної області наведено у табл. 1.

Таблиця 1

Моделювання предметної області в архітектурах програмних систем

Архітектура	Як відбувається моделювання предметної області	Недоліки
Клієнт-серверні, DCOM, CORBA, Java	У явному вигляді моделювання не проводиться	Бізнес-логіка та дані неявно включені у код, що ускладнює його модифікацію
NET, WPF, Common application architecture, Domain-driven architecture	Побудова окремої бізнес-моделі та на її основі – програмних бізнес-компонент та сервісів	Бізнес-компоненти відображають стан середовища в один момент часу
SOA	Побудова бізнес-моделі, аналіз стратегії та тенденцій розвитку бізнес-середовища	Складність реалізації непередбачених змін
MDA	На основі модельної специфікації предметної області будують специфікацію програмної системи, яку компілюють у код	Значна складність розроблення модельної специфікації
Архітектури, що базуються на правилах	Модель середовища, подана у вигляді правил, є частиною програмної системи	Відсутність цілісної моделі середовища, складність узгодження правил
Архітектури, що базуються на онтології	Модель предметної області, подана онтологією, є цілісною	Складність побудови та супроводу онтології

Формулювання мети

Для побудови програмних систем з використанням онтологічного підходу об’єктом розгляду вибрано задачу як найменшу ідентифіковану та необхідну частину будь-якого процесу. Поняття “задача” визначається в літературі як усвідомлена проблемна ситуація з визначеними умовами (даними) та метою [8].

Онтології задач запропоновані в результаті розвитку наукового напряму аналізу задач (task analysis). Методи аналізу задач використовуються для визначення та формалізації усіх факторів, що впливають на процес розв’язання задачі експертом. Такі методи широко застосовуються для проектування інтерфейсів комп’ютерних програм, в експертних системах, системах підтримки прийняття рішень [9]. Головним завданням цього напряму є визначення складових частин задачі, її структури та обмежень. Це допомагає експерту краще її зрозуміти, змоделювати процес розв’язання задачі та оцінити результати, передати свої знання іншим експертам.

Значних змін напрям аналізу задач зазнав з появою онтологічного моделювання. Запропоновано використовувати онтології задач (task ontologies) для формалізації концептів та відношень задачі [10]. На відміну від інших типів онтологій, таких як загальні, або онтології предметних областей, онтології задач будуються окремо для класів подібних задач, у них вводиться поняття дії [11] та забезпечується її виконання або моделювання виконання, реалізовано виконання моделі, побудованої на основі онтології задачі.

Дослідження онтологій задач тісно пов’язано з концептуальним моделюванням, адже в процесі побудови онтології задачі фактично створюється її формалізована концептуальна модель. [12]. Важливим аспектом як концептуального, так і онтологічного моделювання задачі є взаємодія з експертом предметної галузі, який створює та валідує онтологію.

У процесі досліджень онтологій задач реалізовані середовища моделювання, які дають змогу створювати та виконувати онтологічні моделі для окремих класів задач. Найрозвиненішим у класі таких середовищ є CLEPE (Conceptual level programming environment) [13].

Метою роботи є розроблення нових архітектурних принципів, методів та інструментальних засобів побудови програмних систем на основі онтологічних моделей задач.

Формальне подання знань у системі моделювання

Використання онтологічних моделей для побудови програмних систем потребує розроблення математичного обгрунтування у вигляді відповідної моделі, яка надалі буде використовуватися для формальної специфікації методів та системи моделювання. Для побудови формальної моделі використано апарат алгебри систем [14], що визначає алгебраїчну систему шляхом комбінації декількох алгебраїчних доменів.

Нехай у предметній області існує n множин об'єктів:

$$A_1, A_2, ... A_n \tag{1}$$

Об'єкти, що належать кожній множині, класифікують як екземпляри конкретного поняття. Ці множини є множинами-носіями для n багатосортних алгебр. Окремі екземпляри, що належать цим множинам, позначатимемо $a_1, ... a_n$.

Домени концептів (сутностей) E. На основі кожної множини A_i визначимо абстрактний тип даних E_i

$$E_i = (Name, \Sigma, Ex), \tag{2}$$

де *Name* – назва типу, Σ – сигнатура багатосортної алгебри, *Ex* – визначальні співвідношення типу. Окремі типи позначатимемо E_i (довільний тип, його назва не визначена) або E_{name}, де name – назва типу, якщо в контексті розгляду важливо вказати на конкретний тип. Змінні, що набувають значення екземплярів конкретного типу, позначимо X_{name}, або просто *Name*. Типи E_i утворюють множину алгебраїчних доменів *E*, які відповідають концептам (сутностям) предметної області:

$$E = \{E_1, E_2, ..., E_n, \} \tag{3}$$

Домен атрибутів At. Визначимо алгебраїчний домен атрибутів At на списку значень атрибутів у вигляді пар (key, value). Елемент пари key визначає ідентифікатор атрибута, а value – його значення. Згідно з [5] для цього домена визначено операції об'єднання, підстановки, видалення, інтерпретації (merge, substitute, delete, interp).

На практиці корисними є функції, які визначені на змінних різних доменів, наприклад, функція вибору значення атрибута:

$$F_{selval}(At, key) \to value \tag{4}$$

Булевий домен Cs. До булевого домена належать вирази, результат підрахунку яких належить булевій множині {true, false}. Операндами виразів є змінні, що належать іншим алгебраїчним доменам. Будемо інтерпретувати елементи булевого домена як обмеження Cs. Операціями для булевого домена є булеві операції {and, or, negation}, а також операція інтерпретації interp, яа відображає спрощення та підрахунок булевих виразів. Екземпляром булевого домена є конкретне обмеження.

Домен сутностей з атрибутами T. Визначено на множині кортежів $\{(E_i, At_j, Cs_j^*)\}$. Для кожного *i* існує тільки один *j*, що входить в елемент цього домена:

$$\forall i \exists^1 j : (E_i, At_j) \tag{5}$$

Кожне обмеження $Cs_j \in Cs_j^*$ є виразом з операндами, що належать домену At_j. Операціями над елементами цього домена є об'єднання та поділ сутностей {merge, split}. Операція об'єднання сутностей інтерпретується як формування нової сутності як спільного набору властивостей та обмежень сутностей-операндів. Операція поділу сутності – зворотна до об'єднання. Екземпляром домена сутностей з атрибутами є кортеж відповідних фактів.

Домен відношень Rl. Множиною – носієм цього домена є структури виду:

$$\{(T_{1,i} \times T_{2,i} \times ... \times T_{k,i}, At_i^r, Cs_i^*)\} \tag{6}$$

Кожна структура є кортежем, що містить декартовий добуток алгебраїчних типів даних з домена T, тип з домена атрибутів At (визначає атрибути відношення) та множина обмежень Cs.

Кожне обмеження $Cs_i \in Cs_i^*$ є булевим виразом з операндами, що належать доменам

$$At_{1,i}, At_{2,i}, ..., At_{k,i}, At_i^r . \quad (7)$$

Над відношеннями визначено операції об'єднання відношень, відокремлення, підстановки {merge, split, substitute}. Операції об'єднання та відокремлення відношень трактуються подібно до аналогічних операцій з домена E. В операції підстановки замість однієї сутності домена T підставляємо відношення, яке при цьому трактується як сутність з атрибутами.

Онтологія є кортежем, поданим доменами понять (концептів) з атрибутами, відношеннями та обмеженнями з булевого домена, що стосуються доменів понять та відношень.

Визначимо онтологію задачі On_{TS} яка є частиною загальної онтології On і містить об'єкти, необхідні для розв'язання конкретної задачі

$$On_{TS} \subseteq On \quad (8)$$

Онтологічна модель Md визначається кортежем з трьох елементів:

$$Md =< On_{TS}, Ac_{TS}^*, Cs_{TS}^* >, \quad (9)$$

де On_{TS} – онтологія задачі, Ac_{TS}^* – множина дій, Cs_{TS}^* – множина додаткових обмежень.

Дія Ac є сутністю з атрибутами (алгебраїчний домен T), що інтерпретується як команда до певного зовнішнього сервісу або команда на виконання іншої онтологічної моделі. Для команди визначені параметри, які отримують зі значень атрибутів елементів, що входять у On_{TS} (або ж вони задані як константи). Функцію ініціалізації параметрів InstPar подамо як набір відображень між атрибутом дії та значеннями атрибутів сутностей та відношень, що входять в онтогічну модель:

$$InstPar : (Ac_{l,i}, pkey_{l,i}) \to SelValue(At_{l,j}, key_{l,k}).$$
$$Ac_{l,i} \in Md_l, At_{l,j} \in On_{TS}^l \in Md_l. \quad (10)$$

У табл. 2 подані основні алгебраїчні домени, використані у математичній формалізації.

Таблиця 2

Алгебраїчні домени моделі

Домен	Множина-носій	Операції
Сутностей (n доменів) – E	$\{a_i \mid Type(a_i) = E_i\}$	
Атрибутів – At	$\{(key, value)^*\}$	{merge, substitute, delete, interp}
Булевий домен Cs	{true, false}	{and, or, negate, interp}
Сутностей з атрибутами –T	$\{(E_i, At_j, Cs_j^*)\}$	{merge, split}
Відношень	$\{(T_{1,i} \times T_{2,i} \times ... \times T_{k,i}, At_i^r, Cs_i^*)\}$	{merge, split}
Онтологія	(T, Rl, Cs^*)	{merge, split}
Онтологічних моделей	$\{(On_{TS}, Ac_{TS}^*, Cs_{TS}^*)\}$	

Аналіз методів подання та опрацювання знань показує, що значна їх частина ґрунтується на декларативному підході. Зокрема, до цієї категорії зараховують фреймові системи, онтологічні системи на базі OWL тощо. Перевагою декларативного підходу до подання знань є можливість створення цілісної формальної моделі предметної області та використання потужних механізмів логічного виведення. Серед його недоліків – значна складність онтології та процесів менеджменту, непристосованість до подання та використання процедурних абстракцій.

Альтернативою декларативного підходу до подання знань є процедурний підхід, зокрема SBD (Schema based design) [15]. Перевагами процедурного підходу є простота побудови та використання схем, подібність схем до ментальних моделей, орієнтація на розв'язання задач. Недоліками процедурного підходу є слабка пов'язаність та узгодженість схем, відсутність цілісного подання предметної області та інструментальних засобів збереження та опрацювання схем.

Враховуючи взаємодоповнюваність декларативного та процедурного підходів, їх доцільно поєднати. При цьому онтологія визначає набір сутностей, відношень, аксіом та обмежень для всієї предметної області. Роль схем відіграють онтологічні моделі задач. Вони використовують компоненти онтології та визначають сутності, відношення, обмеження та дії у контексті окремої задачі.

На рис. 1 наведена загальна структура програмної системи на основі онтологічних моделей задач.

Центральним елементом системи є база знань, яка містить онтологію, базу фактів та набір моделей. База фактів зберігає факти про об'єкти та події зовнішнього світу, які є екземплярами класів, визначених онтологією. Онтологія містить модель предметної області, подану як таксономію класів. Це створює можливість однозначного трактування усіх об'єктів з бази фактів та елементів онтологічних моделей. Онтологічні моделі відображають знання про способи розв'язання задач. На основі моделей формують факт-моделі – моделі, ініціалізовані конкретними фактами з бази фактів.

Моделі виконують дії, формуючи команди та запити до зовнішніх відносно системи моделювання сервісів. Такими сервісами є сервіси операційної системи, сервіси інформаційної системи підприємства, побудованої з дотриманням вимог SOA, або довільні веб-сервіси.

Онтологічна модель використовується для розв'язання порівняно простих задач. Для розв'язання складних задач, а також моделювання та підтримки виконання процесів, необхідно забезпечити взаємодію онтологічних моделей задач.

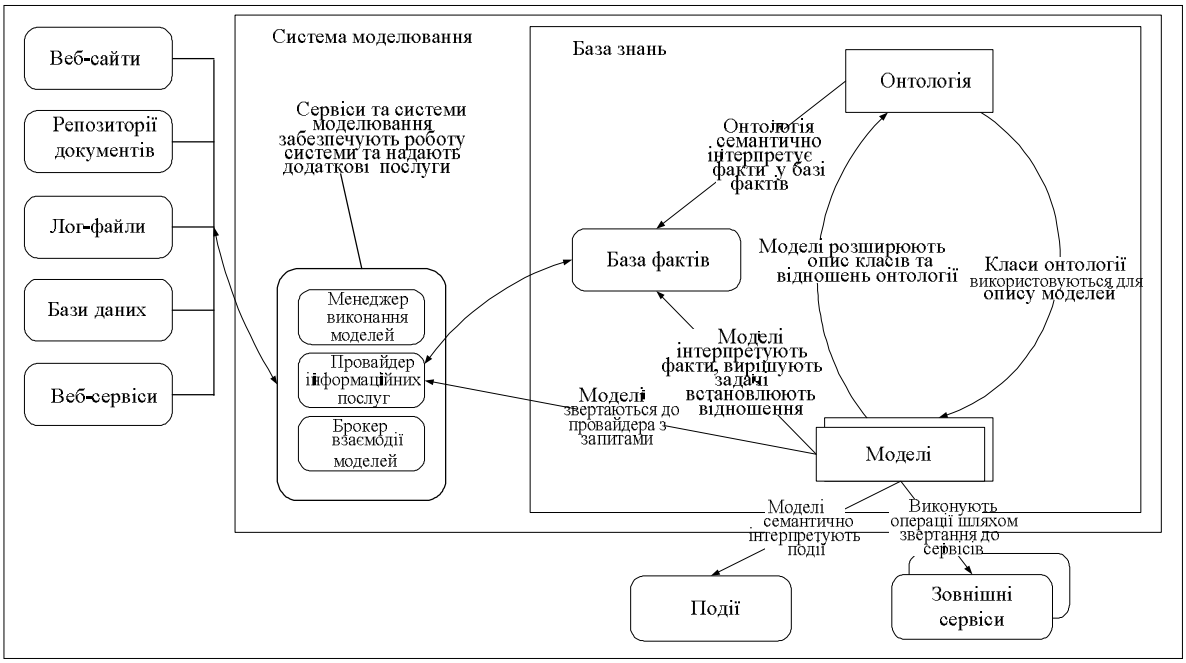

Рис. 1. Структура програмної системи на основі онтологічних моделей задач

Формальне визначення поняття контексту необхідне для забезпечення такої взаємодії. Моделі обмінюються порівняно невеликими обсягами інформації, передаючи тільки посилання на певні факти. Усю потрібну для виконання операції інформацію модель повинна отримати самостійно, з контексту цих фактів. Формальне визначення контексту дано на рівні онтології (контекст знань) для певного класу та на рівні фактів для конкретного факту. Центральним елементом контексту на рівні знань є певний клас T^{co}. Контекстом знань нульового рівня є сам клас та порожня множина його зв'язків:

$$Con^0_{kn}(T^{co}) = (S^0_{co}(T^{co}), S^0_{co}(T^{co}_{RCL})) = (T^{co}, \varnothing). \tag{11}$$

Контекст знань першого рівня містить усі зв'язки та класи, з якими клас контексту поєднаний безпосередньо:

$$Con_{kn}^{1}(T^{co}) = Con_{kn}^{0} \bigcup (S_{co}^{1}(T^{co}), S_{co}^{1}(T_{RCL}^{co})), \quad (12)$$

де $\forall T^i \in S_{co}^1(T^{co}) \exists T_{RCL}^j : T^{co} \in T_{CL-RCL}^j, T^i \in T_{CL-RCL}^j$ та

$\forall (T^i, T^j) \in T_{CL-RCL}^i \in T_{RCL}^i \in S_{co}^1(T_{RCL}^{co}) : T^i \in S_{co}^0(T^{co}), T^j \in S_{co}^1(T^{co})$.

Аналогічно контекстом рівня k є

$$Con_{kn}^{k}(T^{co}) = Con_{kn}^{k-1} \bigcup (S_{co}^{k}(T^{co}), S_{co}^{k}(T_{RCL}^{co})), \quad (13)$$

де $\forall T^i \in S_{co}^k(T^{co}) \exists (T_{RCL}^j, T^l \in S_{co}^{k-1}(T^{co})) : T^i \in T_{CL-RCL}^j, T^l \in T_{CL-RCL}^j$ та

$\forall (T^i, T^j) \in T_{CL-RCL}^i \in T_{RCL}^i \in S_{co}^k(T_{RCL}^{co}) : T^i \in S_{co}^{k-1}(T^{co}), T^j \in S_{co}^k(T^{co})$.

Контекстом певного класу T^{co} вважаємо контекст максимального рівня m, який можна побудувати у цій базі знань:

$$Con_{kn}(T^{co}) = Con_{kn}^{m}(T^{co}) \mid m = \max(k). \quad (14)$$

Визначимо контекст моделі T_{MD}^{co} на рівні знань як контекст усіх класів, які використовуються у моделі:

$$Con_{kn}(T_{MD}^{co}) = \bigcup Con_{kn}(T^{co}) \mid T^{co} \in T_{RG}^{co}. \quad (15)$$

Факт-моделі створюються на вимогу інших моделей або у разі настання певних подій. Їх використовують для розв'язання поточних задач системи. Активна факт-модель – це екземпляр визначеного типу моделі, ініціалізований інформацією з певного контексту

Взаємодія моделей T_{RMD} – це тип даних, що відображає відношення активації, яке використовується для прийняття рішення щодо активації моделі (створення та запуску на виконання нової факт-моделі) (рис. 2). Ініціатором є модель-активатор. Потреба в активації та створенні нової факт-моделі виникає тоді, коли для розв'язання основної задачі потрібно розв'язати допоміжні задачі, подані в інших моделях.

Кожному підтипу T_{RMD} – T_{RMD}^i відповідає клас задач T_{PR}^j, які необхідно розв'язати в результаті взаємодії. Своєю чергою, класу задач T_{PR}^j відповідає множина моделей $S(T_{MD}^j)$, які можна застосувати для розв'язання задач цього класу. Під час взаємодії моделей послідовно розв'язуються задачі визначення релевантності, оптимального вибору серед релевантних моделей та ініціалізації вибраної моделі.

Функція релевантності є відображенням поточного контексту моделі-активатора $Con(t_{MD}^{act})$ та множини альтернатив $S(T_{MD})$ у множину {true, false}:

$$F_{rel} : Con(t_{MD}^{act}), S(T_{MD}) \to (true, false). \quad (16)$$

Визначення релевантності моделей дає змогу відібрати для процедури вибору тільки релевантні моделі $S(T_{MD}^{re}) \subseteq S(T_{MD})$, тобто такі типи моделей T_{MD}^{re}, для яких

$$F_{rel}(Con(t_{MD}^{act}), T_{MD}^{re}) = true. \quad (17)$$

За відсутності релевантних моделей система моделювання повертає активатору повідомлення про неможливість розв'язання задачі.

Задача вибору визначає у множині релевантних моделей одну T_{MD}^{op}, застосування якої максимізує функцію вибору F_{ch} з урахуванням критеріїв вибору $S(t_{CR})$ та контексту $Con(t_{MD}^{act})$:

$$F_{ch}(T_{MD}^{op}, S(t_{CR}), Con(t_{MD}^{act})) \to \max. \quad (18)$$

Функція ініціалізації F_{in} здійснює відображення поточного контексту $Con(t_{MD}^{act})$ у множину значень атрибутів вибраної моделі – $\{At^j \in T_{MD}^{sel}\}$:

$$F_{in} : Con(t_{MD}^{act}) \to \{At^j \in T_{MD}^{sel}\}. \quad (19)$$

Використання онтологічних моделей задач дає змогу спростити проведення *менеджменту складних онтологій*, тобто виконання завдань створення, модифікації, відслідковування походження та валідації онтології.

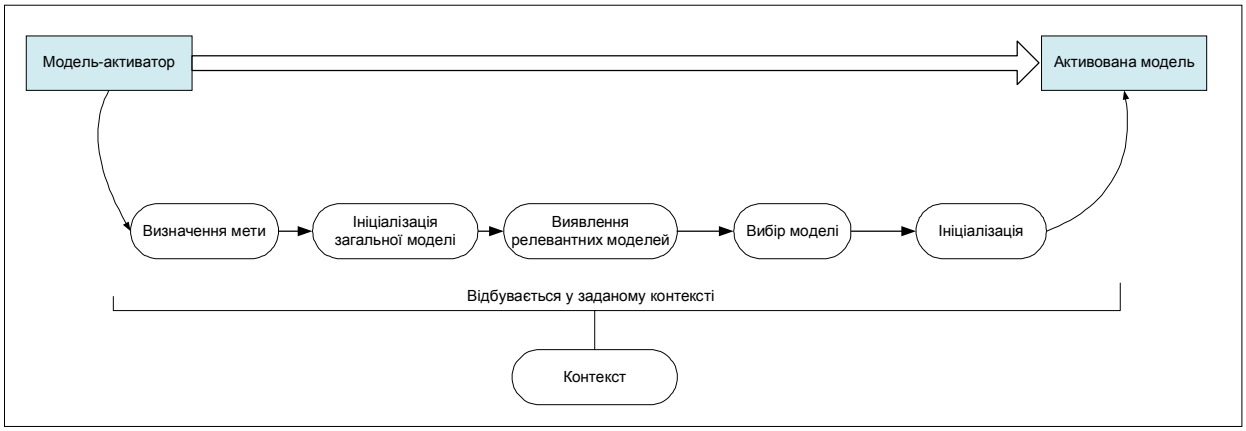

Рис. 2. Процес активації моделі задачі

Процес створення онтології виконується для кожної задачі системи та містить такі кроки: а) кожну задачу аналізують та визначають усі необхідні для її розв'язання сутності та відношення, обмеження та операції; б) будують онтологічну модель задачі з використанням виявлених компонент, причому ця модель повинна використовувати передусім компоненти з наявної онтології; в) в онтологію додають компоненти задачі, яких у ній не було; г) оновлену онтологію валідують та вирішують можливі протиріччя.

Використання онтологічних моделей задач для побудови онтології має переваги над традиційним підходом створення онтології експертом предметної області. Зокрема, на кожному кроці визначено чіткі критерії включення компонента в онтологію; побудова онтології є ітеративним процесом, в якому додавання нової задачі до компетенції онтології призводить до її розширення; на кожному етапі зміст онтології відповідає комплексу задач, що розв'язуються за її допомогою; порівняно невелика кількість компонент задачі дає змогу онтологічному інженеру зосередитися на виявленні та формалізації відношень та обмежень, забезпечити потрібну глибину онтології; при цьому повторно використовують знання, отримані під час онтологічного моделювання попередніх задач.

Як приклад застосування *методології побудови онтології* на основі аналізу бізнес-процесів і задач розроблено онтологію (рис. 3) галузі тестування програмного забезпечення.

За основу аналізу бізнес-процесів галузі тестування прийнято процеси, сформульовані у стандарті ISO/IEC/ieee 29119 [16]. Цей стандарт визначає три групи процесів:

- організаційні – в результаті виконання цих процесів формулюють загальну політику та стратегію тестування на рівні проектної організації; ці процеси виконують керівники організації;
- керування у межах проекту – планування тестування, відстеження стану тестування продукту, критеріїв його якості, завершення;
- тестування – створення тестових сценаріїв, формування тестових середовищ, виконання тестів, звітування про дефекти.

Розглянуто процеси тестування, що виконуються у контексті конкретного проекту, тобто процеси другої та третьої груп. Визначено типові процеси та задачі й побудовано їх онтологічні моделі. Приклади графічного подання онтологічних моделей задачі та процесу наведено на рис. 4, 5.

Процес тестування виконання вимоги розпочинається з підготовки середовища тестування, інсталяції усіх необхідних програм. Вимога, що тестується, визначена у документі "Специфікація вимог". Якщо в результаті тестування виявляють, що вимога не виконується, то відкривають відповідний дефект. Про результат тестування звітують у програмі керування тестуванням.

Валідація онтології, побудованої на основі моделей задач, полягає у валідації комплексу онтологічних моделей, покладених в основу онтології. Відомими вимогами до валідної онтології є

вимоги повноти, коректності та відсутності надлишковості [17]. Так, онтологія вважається повною, якщо усі релевантні для її компетенції аспекти предметної області відображені. Онтологія вважається коректною, якщо знання, визначені у ній, коректні для визначеної предметної області та релевантні для компетенції онтології.

За аналогією з цими визначеннями вважаємо повною онтологічну модель, якщо вона містить усі необхідні для розв'язання поставленої задачі сутності, відношення, обмеження та операції. Коректною вважаємо модель, яка розв'язує поставлену задачу згідно із заданими критеріями ефективності.

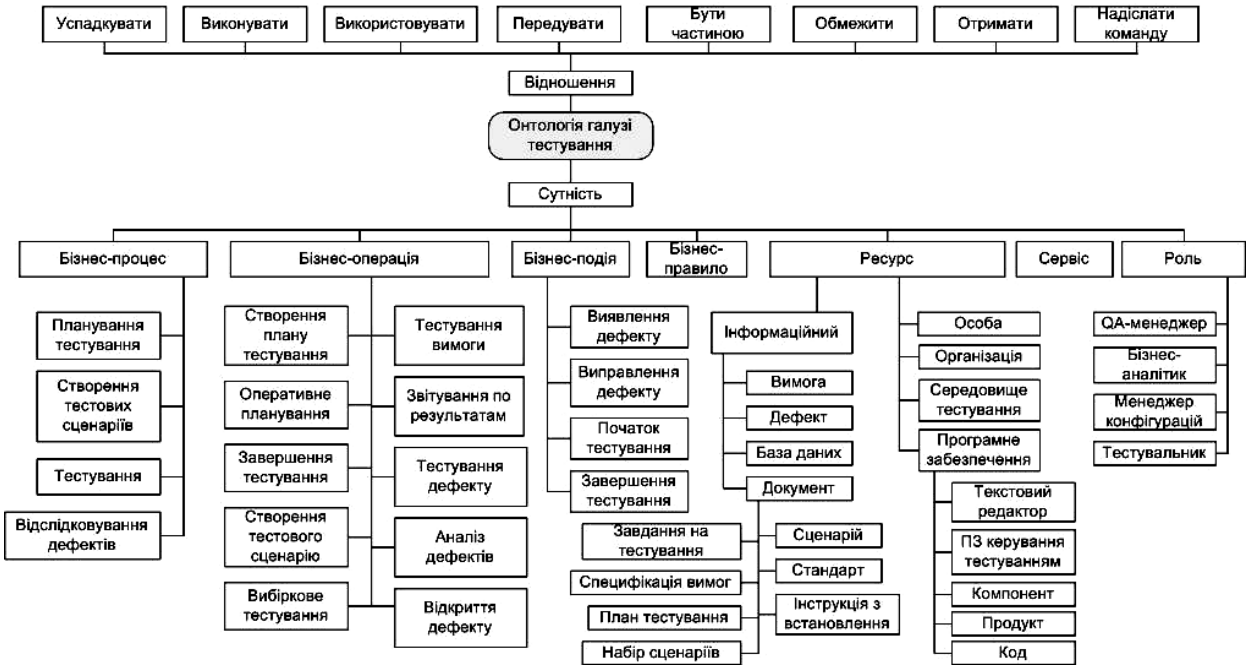

Рис. 3. Сутності та відношення онтології тестування програмного забезпечення

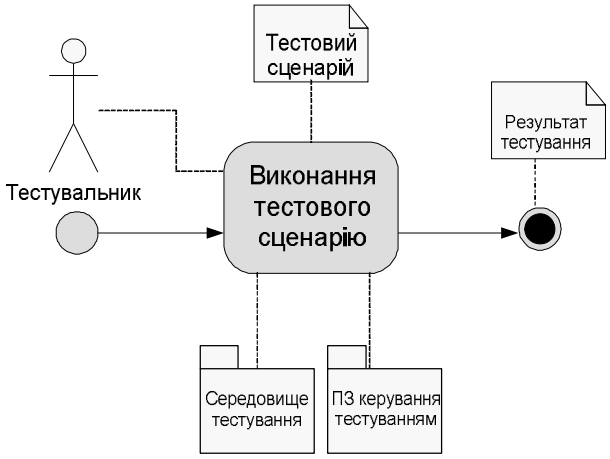

Рис. 4. Онтологічна модель задачі "Виконання тестового сценарію"

Нехай для заданої предметної області визначено множину повних та коректних моделей $S(T_{MD})$. Тоді онтологія, побудована на основі цієї множини, є повною, якщо:

$$\forall Cm_i \in T^i_{MD} \in S(T_{MD}) : Cm_i \in On, \tag{20}$$

де Cm_i – довільний компонент онтології. Онтологія коректна, якщо побудована на основі коректних моделей.

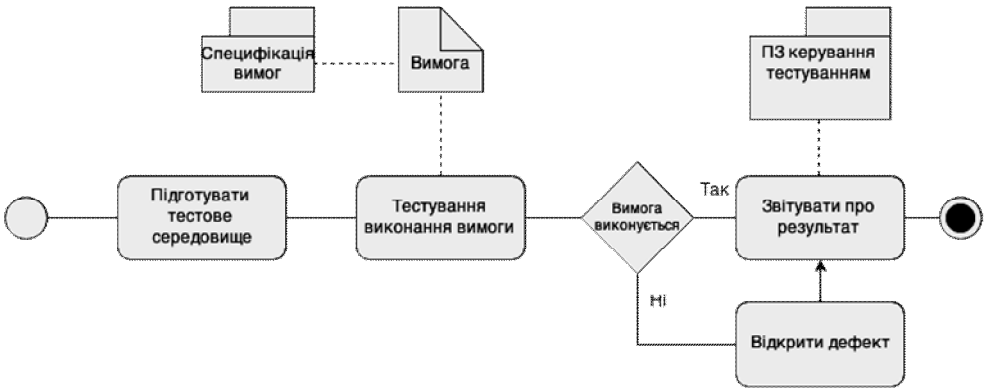

Рис. 5. Онтологічна модель процесу "Тестування на відповідність вимозі"

Принципи побудови та функціонування програмної системи на базі онтологічних моделей задач

Недоліки використовуваних модельно-орієнтованих підходів до побудови програмних систем значною мірою пояснюються складністю як предметної галузі, так і відповідних моделей. Ці недоліки доцільно усунути, реалізуючи програмну систему як набір простих інтерпретованих моделей задач, що взаємодіють. Виконувальний та інтерпретований характер опрацювання моделей створює додаткові переваги порівняно з традиційними методами розроблення програмного забезпечення і з xUML, зокрема:

- зникає необхідність у разі зміни бізнес-логіки проводити перекомпіляцію та поновне розгортання системи – необхідно модифікувати тільки ті моделі, що реалізують цю логіку;
- використання онтологічних моделей дає змогу накопичувати та повторно використовувати знання про способи розв'язання задач.

Накопичення та повторне використання знань у програмній системі здійснюється у результаті створення та використання онтологічних моделей задач на трьох рівнях концептуальної моделі програмної системи: рівні бізнес-процесів, рівні сервісів тв застосувань, рівні пристроїв. Зокрема, на рівні бізнес-процесів визначають моделі бізнес-процесів та оперують поняттями, що відображають характеристики бізнес-процесів.

На рівні сервісів і застосувань розглядають застосування та загальносистемні сервіси і ресурси, які використовують бізнес-процеси. Тут враховують сумісне використання ресурсів, формулюють та впроваджують загальносистемні корпоративні політики, керують пріоритетністю виконання завдань.

Рівень пристроїв містить моделі виконавців завдань. Виконавцем може бути як машина певного типу, так і людина. На цьому рівні визначають загальні характеристики, обмеження та моделі функціонування процесорів.

Отже, програмна система розглядається як кортеж множин моделей:

$$PS = (S_{MD-BP}, S_{MD-SE}, S_{MD-PC}), \tag{21}$$

де S_{MD-BP} – множина моделей бізнес-процесів, S_{MD-SE} – множина моделей сервісів, S_{MD-PC} – множина моделей пристроїв.

На кожному рівні розглядається комплекс взаємопов'язаних моделей, які використовують для розв'язання задач адаптації у межах цього рівня. Так, на рівні бізнес-процесів адаптувати поведінку системи можна, змінюючи пріоритетність виконання завдань, з переходом на інший варіант структури процесу. Одним з варіантів адаптаційних рішень є зміна вимог до рівнів процесів або зміна налаштувань пристроїв. Рішення про вибір того чи іншого варіанта адаптаційної поведінки вибирається у межах рівня на основі наявної бази знань залежно від характеристики ситуації, яка потребує адаптації системи.

На рівні моделювання бізнес-процесів головну увагу приділено тій частині загальної бізнес-моделі, яка виконується з використанням комп'ютерної техніки та інформаційних технологій –

рівні керування інфраструктурою та реалізації процесів за Остервальдером [18]. Визначено типи моделей, які використовуються на рівні бізнес-процесів (рис. 6).

Так, за ознакою належності до конкретної предметної області визначено загальні та специфічні моделі. Загальні моделі використовуються для подання знань у багатьох предметних областях. Прикладом таких загальних моделей є часові моделі, моделі особи, деякі моделі документів та моделі структури організації. Специфічні моделі використовують для подання знань у конкретній предметній області. Прикладом такої моделі є модель процесу розроблення програмного продукту. Ще одна класифікація моделей визначена за класами задач, які вони розв'язують, та особливостями архітектури моделі: алгоритмічні, ситуаційні моделі, моделі класифікації, моделі операцій тощо. Як правило, усі моделі одного класу посилаються на одну спільну, загальну модель, що подає задачу моделі у найзагальнішому вигляді.

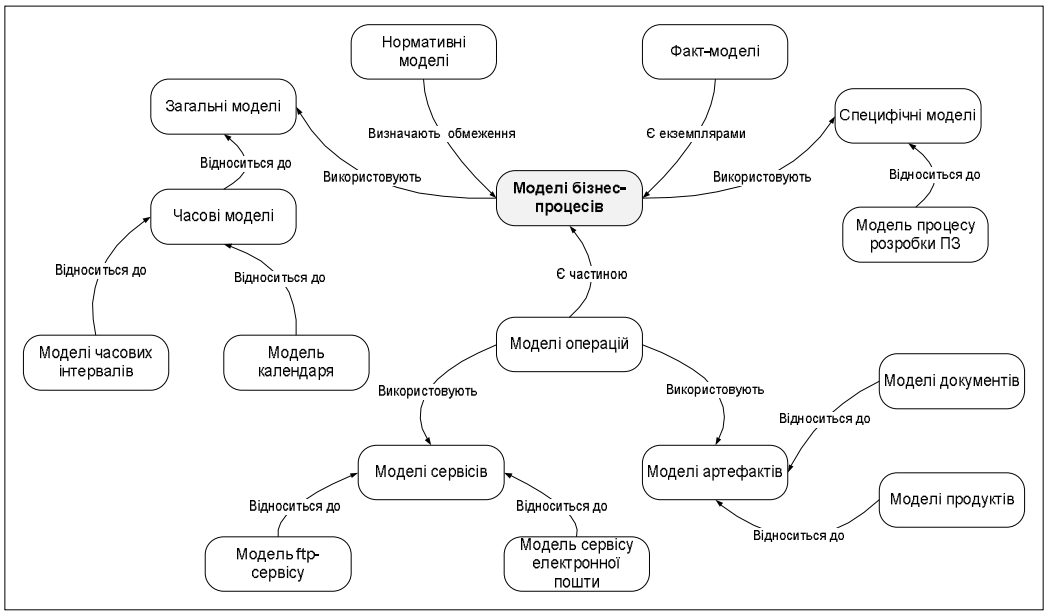

Рис. 6. Схема залежностей між різними типами моделей у програмній системі на базі онтологічних моделей задач

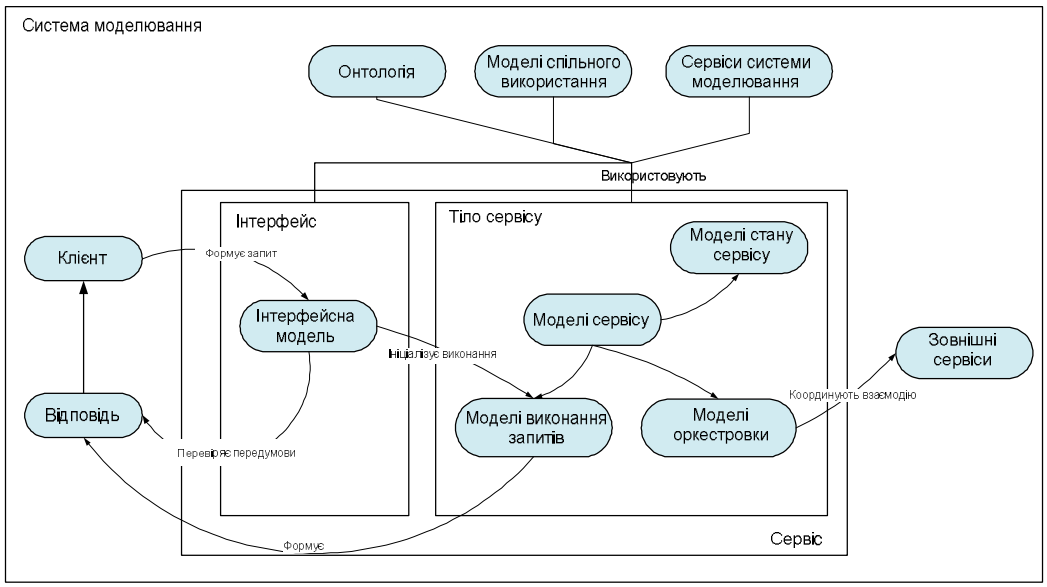

Рис. 7. Архітектура інтелектуального сервісу програмної системи на базі онтологічних моделей задач

Головними компонентами другого рівня моделювання є *моделі сервісів*. На рис 7 наведено архітектуру інтелектуального сервісу, побудованого на основі онтологічних моделей задач, яка специфікує як семантику доступу до сервісу, так і семантику виконання запитів сервісом. Розробникам та користувачам таких інтелектуальних сервісів важливо розуміти не тільки мету та параметри сервісу, але й порядок його роботи, функціональні обмеження та залежності. Цю інформацію клієнти використовують для прийняття рішення у процесі взаємодії з сервісом. Онтологічні моделі, також використані для подання механізмів роботи самого сервісу. Сервіс, що використовує моделі враховує свій стан та стан зовнішнього середовища.

Для вирішення завдання пошуку та вибору сервісів відповідно до задач, які вони розв'язують, використовується окрема онтологія сервісів.

Запит до сервісу формує зовнішня відносно сервісу активована модель. Інтелектуальний сервіс, з метою організації взаємодії з клієнтами, має одну або декілька схем моделей, що відіграють роль інтерфейсу сервісу. В процесі формування запиту ролі інтерфейсної моделі заповнюються значеннями з контексту моделі-клієнта. Після цього генеруються команди виконання запиту. Запити класифікуються та типізуються відповідно до існуючої онтології типів запитів. Різні типи запитів мають різні передумови та алгоритми виконання. При цьому інтерфейсна модель перевіряє необхідні передумови у сенсі наявності потрібних для виконання запиту даних та відомостей про клієнта залежно від типу запиту. Якщо усі передумови для виконання запиту дотримані, запит виконується сервісом. При цьому виконуються його внутрішні моделі.

Інтелектуальні сервіси у програмній системі утворюють мережу сервісів (реєстр сервісів, service inventory), яка є множиною сервісів:

$$S(t_{SE}) = \{t_i \mid Type(t_i) = T_{SE}\}. \tag{22}$$

Моделювання на рівні пристроїв належить до третього рівня моделювання. Моделювання на цьому рівні вирішує завдання забезпечення ефективної роботи конкретних пристроїв, їх компонент або комплексу пристроїв, а також завдання адаптації налаштувань пристрою до вимог продуктивності, які ставлять моделі рівня сервісів та бізнес-процесів.

У першому випадку об'єктами керування є комп'ютери, жорсткі диски або набір комп'ютерів, об'єднаних у мережу. Моделі відстежують стан пристроїв, виявляють ранні ознаки майбутніх збоїв та відмов, автоматизують виконання деяких функцій людини-адміністратора. Моделі також забезпечують інтелектуальне реагування комп'ютерної системи на непередбачувані події. У другому випадку онтологічні моделі виконують вимоги вищих рівнів моделей щодо зміни налаштування пристроїв з метою підвищення пріоритетів виконання певних застосувань. Наприклад, на період проведення відеоконференції пріоритезують трафік конференції у мережі та блокують усі не пов'язані з виконанням виробничих завдань передавання.

Методи розв'язання задач у прикладних програмних системах на базі онтологічних моделей

У процесі створення та практичного використання програмних систем на базі онтологічних моделей розроблено методи розв'язання задач для предметних областей розроблення програмного забезпечення, підтримки прийняття рішень та туризму.

Зокрема, онтологічні моделі задач застосовані для *відображення структури бізнес-процесів* та *розв'язання задач бізнес-аналітики* інформаційних систем підприємств з метою підвищення відповідності вимогам та збільшення адаптаційних можливостей таких систем до зміни бізнес-процесів. Модель бізнес-процесу Md_{bp} визначена кортежем:

$$Md_{bp} = (S(Md_{bo}), S(Md_{dc}), S_{rl}, S_{ac}, S_{cs}), \tag{23}$$

де $S(Md_{bo})$ – множина моделей бізнес-операцій; $S(Md_{dc})$ – множина моделей прийняття рішення; S_{rl} – множина відношень; S_{ac} – множина дій; S_{cs} – множина обмежень.

З метою підтримки виконання бізнес-процесів була розроблена високорівнева онтологія бізнес-процесів [19]. В основу цієї онтології покладено концепти та відношення мови BPMN. Модель бізнес-процесу визначає логічну послідовність операцій, які здійснюються у бізнес-процесі.

Ці операції існують як окремі типи сутностей в онтології. Визначено такі типи сутностей, як *БізнесОперація*, *БізнесПодія* і *ПрийняттяРішення*. Всі ці сутності походять від сутності онтології *БізнесОб'єкт*. Експерт предметної області працює із системою, використовуючи як текстове, так і графічне подання моделі бізнес-процесу. Графічне подання дає змогу охопити бізнес-процес загалом. Приклад графічного подання процесу оновлення програмного продукту наведено на рис. 8.

Побудова моделі бізнес-операції дає експерту можливість виявити і детально проаналізувати всі релевантні для виконання операції сутності, відношення та операції, сформувати звертання до інших моделей. Загальна модель бізнес-операції містить такі елементи:

- релевантні бізнес-операції сутності з онтології;
- релевантні відношення;
- вихідний стан, поданий набором умов на значеннях атрибутів відповідних сутностей та відношень;
- кінцевий стан виконання операції. Це список, який містить як успішні, так і неуспішні варіанти закінчення операції;
- механізм переходу між початковим і кінцевим станами, визначення звертань до зовнішніх сервісів;
- визначення числових показників успіху операції з посиланням на зовнішні моделі, що реалізують розрахунок цих показників.

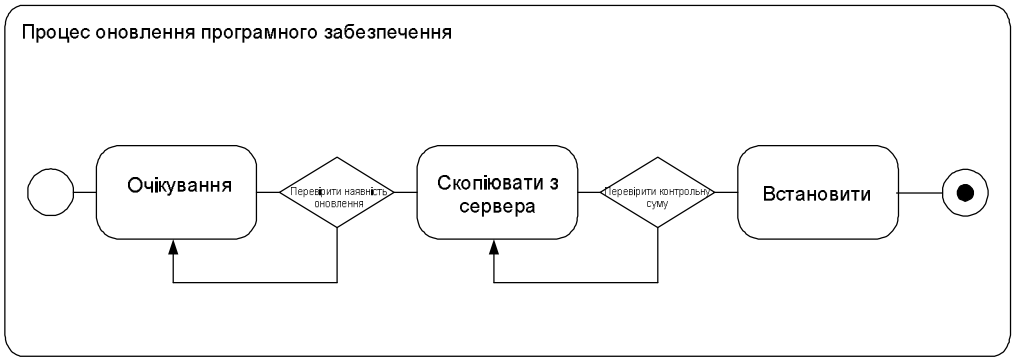

Рис. 8. Графічне подання бізнес-процесу оновлення програмного забезпечення

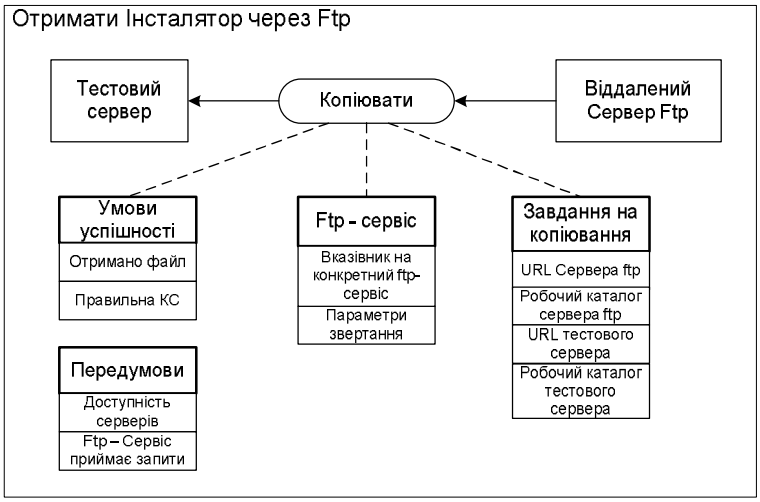

Рис. 9. Графічне подання моделі Отримати ІнсталяторЧерезFtp

Розроблено метод використання онтологічних моделей для *автоматизованого тестування складання програмних продуктів.* Метод реалізує автоматичне виконання багатоетапного процесу тестування, який використовує онтологічні моделі простих задач перевірки наявності файла-інсталятора, його закачування на тестову машину, встановлення, тестування, розсилки повідомлень

про результати тестування, деінсталяції та очищення тестового середовища. На рис. 9 наведено графічне подання моделі задачі отримання інсталяційного файла з використанням ftp сервісу.

З використанням цього підходу створено та апробовано автоматизовану систему тестування програмних продуктів, яка успішно функціонує. Для реалізації системи вибрано скриптову мову VBScript. В ролі програмного засобу автоматизованого тестування використовується HP QuickTest Professional. Розроблена система дала змогу систематично перетестовувати декілька значних за розміром (розмір файла інсталятора 500–700 Мбайт) програмних продуктів за одну ніч. Експлуатація системи тестування довела її високу надійність, гнучкість, простоту модифікації та розвитку.

Використання онтологічних моделей для автоматизації тестування програмних продуктів дає змогу створювати інтелектуальні системи тестування, які здатні швидко адаптуватися до змін у функціональності тестованого продукту, а також врахувати та адекватно відреагувати на зміни в апаратно-програмному тестовому середовищі.

Онтологічні моделі задач використано для *керування доступом до ресурсів інформаційної системи*. Загальну схему моделі керування доступом до ресурсів наведено на рис. 10. Така модель містить ролі, які під час ініціалізації моделі заповнюються об'єктами типу *Працівник* та *Ресурси*. Між ролями та ресурсами встановлюється відношення надання доступу. Ці відношення мають атрибути, які деталізують права доступу, що надаються.

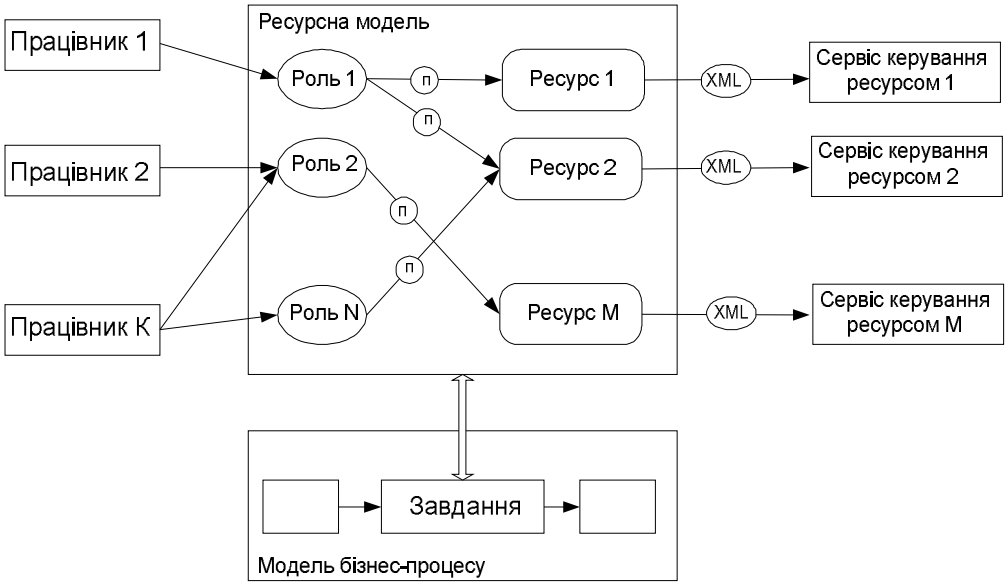

Рис. 10. Структура моделі керування доступом до ресурсів у програмній системі

У роботі [20] показано, що порівняно з відомими методами DAC, MAC, RBAC, ABAC метод керування доступом, який використовує онтологічні моделі, забезпечує динамічне надання та вилучення прав доступу в контексті бізнес-процесів, що виконуються у системі, відсутність розширення прав з часом, спрощення процесу призначення прав та документальне обгрунтування усіх операцій, підтримку використання нормативних процедур.

Важливим напрямом застосування програмних систем на базі онтологічних моделей задач є *системи підтримки прийняття рішень*. Онтологічні моделі дають змогу забезпечити автоматичну ідентифікацію проблемної ситуації та виконання відповідних дій на основі формалізованого у моделях досвіду прийняття рішень у таких ситуаціях.

Загалом, процес прийняття рішення починається з аналізу ситуації та закінчується вибором рішення з множини альтернатив. Такий процес містить фази: аналізу, проектування, вибору рішення та реалізації. На етапі аналізу розглядають бізнесове середовище з врахуванням досвіду експерта. Експерт ідентифікує та класифікує проблему, виробляє вимоги щодо її вирішення.

На цьому етапі доцільно застосувати онтологічні моделі, які відображають досвід розв'язання попередніх задач та порівнюють параметри фактичного стану предметної області зі станами предметної області у випадку попередньо розв'язаних задач.

На етапі проектування експерт визначає цілі, розробляє альтернативні способи досягнення цілей, критерії якості способу розв'язання задачі та вибирає метод її розв'язання . При цьому експерт постійно зіставляє та оцінює вплив запропонованого ним альтернативного рішення на предметну область, виконуючи валідацію свого підходу до вирішення проблеми. Моделі цього етапу дозволяють визначити методи реалізації поставлених цілей, обмеження та побудувати альтернативні рішення.

На етапі вибору експерт оцінює побудовані альтернативи з використанням критеріїв та вибирає одну з альтернатив. При цьому для вибраного рішення додатково прогнозується ефект від його впровадження та проводиться валідація. Моделі, які використовують на цьому етапі, допомагають експерту застосувати математичні та інші методи порівняння альтернатив.

Як приклад, у роботі [21] запропоновано *онтологічні моделі для підтримки консенсусної валідації моделей* колективом експертів.

Переваги застосування онтологічних моделей для побудови програмних систем

Задача аналізу переваг використання онтологічних моделей для побудови програмних систем розв'язана як задача оцінки ступеня підвищення якості програмного забезпечення в результаті використання онтологічних моделей на всіх етапах його життєвого циклу. В основу покладено характеристики якості, визначені у стандарті ISO/IEC 9126–1.

Для проведення аналізу з метою отримання числових оцінок впливу застосування онтологічних моделей задач на характеристики якості програмного забезпечення вибрано п'ятнадцать програмних систем схожого рівня складності (кількість функцій 7–10, кількість сутностей 7–11). Ці програмні системи за сферою застосування належать до двох груп: програмні веб-системи та обліково-інформаційні системи. У кожній групі наявні як подібні за функціями і сферою застосування, так і відмінні системи. Для кожної програмної системи запропоновано по чотири варіанти змін та розроблено проекти з реалізації цих змін. Для кожного проекту розроблено план виконання та оцінено необхідну тривалість часу на проведення змін.

На етапі концептуалізації розробник програмного продукту розв'язує задачу побудови концептуалізації предметної області для завдань, які повинен вирішувати програмний продукт. Великий обсяг робіт з первинної концептуалізації та аналізу предметної області для різних продуктів фактично дублюється. За онтологічного підходу розробник використовує концептуалізацію предметної області, відображену в онтології. Уникнення повторної концептуалізації безпосередньо впливає на такі характеристики якості, як відповідність вимогам, точність, інтероперабельність, зрілість, використання ресурсів на етапі проектування, переносимість та можливість співіснування з іншими продуктами.

Для різних програмних продуктів, що використовують спільну концептуалізацію, простіше організувати взаємодію, адже вони ґрунтуються на спільному наборі концептів з однаковими атрибутами. Завдяки багатократному використанню єдиної онтології у різних програмних продуктах, що реально функціонують, досягається високий ступінь зрілості концептуалізації, а також стабільності та надійності, що досягається багатократним та різнобічним тестуванням, яке проходить онтологія під час створення та експлуатації продуктів на її базі.

Для оцінювання ступеня повторного використання концептуалізації запропоновано використовувати розроблену формулу:

$$C_{en-\det} = 1 - \frac{N_{el-new}}{N_{el-new} + N_{el-old}}, \tag{24}$$

де N_{el-new} – кількість доданих (або змінених) елементів онтології, N_{el-old} – кількість використаних (та не змінених) елементів онтології. Аналіз вибірки проектів показав, що показник повторного використання концептів для різних пар продуктів змінювався в інтервалі від 0,1 до 0,88 (рис. 11).

Продукти, основані на спільній концептуалізації, мають кращу інтероперабельність завдяки уникненню помилок, пов'язаних з різним поданням предметної області в них. Подамо оцінку $C_{\text{int}}(A, B)$ погіршення інтероперабельності, яке виникає через відмінності концептуалізації для двох продуктів A та B, у вигляді:

$$C_{\text{int}}(A,B) = \frac{N_{diff}^{A} + N_{diff}^{B}}{N_{comm}^{AB}}, \tag{25}$$

де N_{diff}^{A} – кількість відмінних елементів у онтології продукту A порівняно з продуктом B, N_{diff}^{B} – кількість відмінних елементів у онтології продукту B порівняно з продуктом A, N_{comm}^{AB} – кількість спільних елементів онтології, які використовують продукти A та B.

Використання єдиної онтології на *етапі дизайну та кодування* дає змогу підвищити ступінь повторного використання коду, поданого онтологічними моделями. Якщо порівняти підхід, що ґрунтується на онтологічних моделях, з об'єктно-орієнтованим програмуванням, кожен клас ООП має декларативну частину, подану оголошенням змінних, та процедурну частину, подану специфікацією методів. У разі використання онтологічних моделей всі моделі (які є аналогами методів) мають спільну декларативну частину, подану онтологією.

У разі використання об'єктно-орієнтованого програмування переносимість обмежена повторним використанням груп взаємозалежних класів. При цьому методи класу не є переносимими і можуть використовуватися тільки з екземплярами цього класу. Використання онтологій для побудови програмної системи дає змогу уникнути багатократного декларування об'єктів у різних класах, підвищити переносимість коду, зменшити використання ресурсів на стадії кодування повторним використанням коду. Характеристики якості, які покращуються у разі використання онтологічного підходу: *переносимість*, *зменшення використання ресурсів у процесі проектування.*

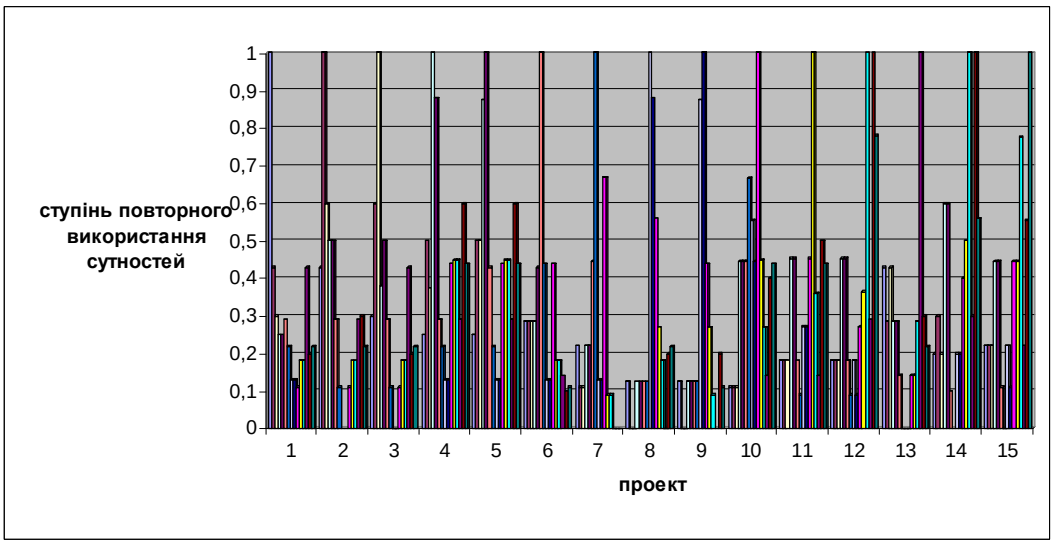

Рис. 11. Оцінка ступеня повторного використання сутностей

Для визначення ступеня переносимості коду розроблено та пропонується використовувати такі оцінки:

1. Відсоток програмних конструкцій (класів, моделей, сервісів) у наявних бібліотеках коду, які можна використовувати без змін:

$$C_{port-code1} = \frac{N_{portable}}{N_{all}}. \tag{26}$$

Зауважимо, що у випадку використання онтологічних моделей $C_{port-code1} = 1$.

2. Кількість додаткових програмних конструкцій (класів, моделей), які необхідно долучити до коду продукту з використанням конкретного класу (моделі), – $C_{port-add}$.

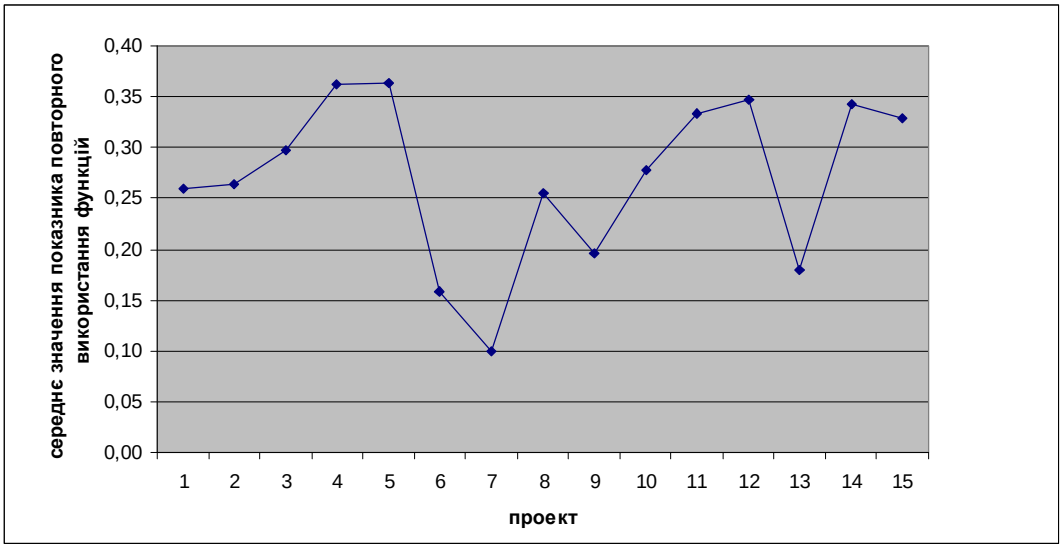

Рис. 12. Середнє значення показника повторного використання функцій у проаналізованих програмних системах

Аналіз вибірки проектів зі створення програмного забезпечення продемонстрував, що різні програмні продукти мають однакові функції, що можуть бути подані моделями. Середнє значення показника повторного використання функцій для проаналізованих проектів наведено на рис. 12.

На етапі тестування програмного продукту використання онтологічних моделей задач приводить до зменшення обсягів тестування. При цьому тестуванню підлягають розроблені або модифіковані моделі, а не продукт загалом.

Оцінити ступінь зменшення трудомісткості тестування запропоновано для програмного продукту, перенесеного на нову (онтологічну) платформу. При цьому доступні тестові сценарії як для традиційної архітектури, так для нової. Оцінкою, що вимірює ступінь зменшення трудомісткості проведення тестування, є відношення кількості сценаріїв у новій платформі до кількості сценаріїв у традиційній платформі з врахуванням середньої складності (наприклад, кількості кроків) кожного сценарію:

$$C_{test} = \frac{N_{mod} \times C_{av-test-mod}}{N_{old} \times C_{av-test-old}}. \qquad (27)$$

Важливим додатковим фактором покращення якості програмного продукту, побудованого з використанням онтологій, є те, що моделі, які повторно використовуються у програмному продукті, попередньо вже пройшли тестування та практично застосовуються в інших програмних продуктах, що дає змогу говорити про повторне використання тестування, проведеного для інших програмних продуктів.

Основні характеристики якості програмного продукту, покращені за умови використання онтологічних моделей на етапі тестування: *придатність до тестування, стабільність, зрілість*.

На етапі супроводу та експлуатації програмного продукту розв'язуються задачі адаптації у разі змін у бізнесовій ситуації та проблемній області.

Для оцінювання складності модифікації програмного продукту традиційним способом використано формулу:

$$C_{m-old} = T_{ov} \times \sum_{i=1}^{n} K_{ov}^{i} + T_{ch} \times \sum_{j=1}^{m} K_{ch}^{j}, \qquad (28)$$

де n – загальна кількість програмних елементів (класів, функцій); m – кількість програмних елементів, які підлягають модифікації; K_{ov} – метрика складності i-го компонента; T_{ov} – середній час, що витрачається на аналіз, тестування та компонування з розрахунку на один програмний компонент та одну одиницю складності; K_{ch} – метрика складності компонента, що змінюється; T_{ch} –

середній час, що витрачається додатково на аналіз, перекодування та перетестування з розрахунку на один програмний компонент, що змінюється.

Зміни у предметній області, які визначають необхідність розроблення нової версії продукту, мають різний характер, що обгрунтовує доцільність проведення їх окремого аналізу. Проаналізовано такі варіанти змін:

1) заміна методу розв'язання задачі або формування нового методу комбінацією наявних;

2) розроблення нового методу розв'язання задачі або уточнення наявного;

3) уточнення подання предметної області введенням нових сутностей, відношень та обмежень (при цьому визначені попередньо елементи онтології не змінюються);

4) зміна розуміння предметної області, яка відображена у модифікації онтології.

У випадку заміни методу розв'язання задачі або формування нового методу як комбінації з наявних модифікується модель. Додатковий час при цьому витрачається на тестування та підготовку даних для нього. Оцінка складності проведення зміни подана формулою:

$$C_{md} = K_m \times (T_a + T_m + T_t), \tag{29}$$

де K_m – оцінка складності (кількість компонент) моделі; T_a – час, що витрачається у середньому на аналіз та документування, T_m – модифікацію та T_t – тестування у розрахунку на один компонент.

Аналіз прикладів зміни методу розв'язання задачі для вибірки реальних проектів показав суттєве (в 5–8 разів) зменшення часових витрат на проведення зміни завдяки використанню моделей задач. Аналіз тривалості модифікацій у цьому випадку показав значні переваги модельного підходу порівняно з традиційним (рис. 13).

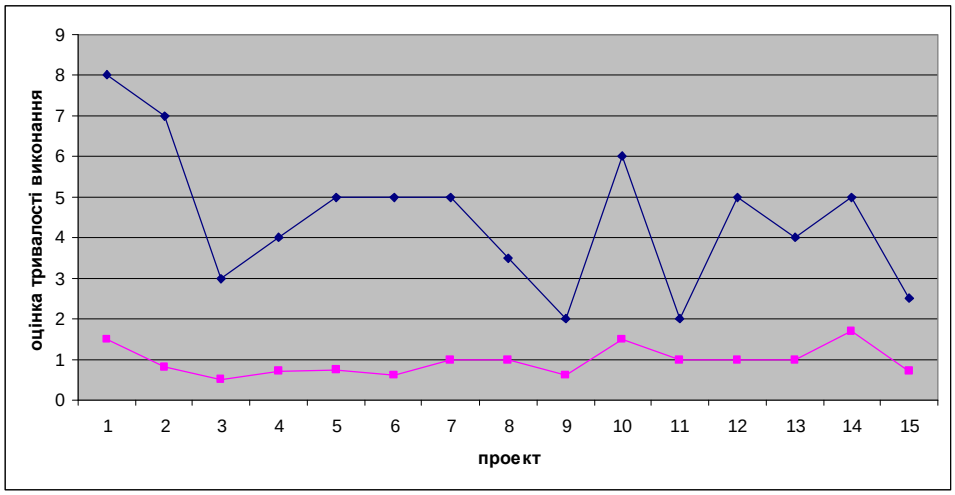

Рис. 13. Оцінки тривалості виконання змін першого типу в програмних системах

Якщо потрібно розробити нову модель або модифікувати наявну, то порівняно з першим випадком збільшуються витрати часу на створення і тестування моделі. Аналіз змін з вибірки реальних проектів показав, що зменшення часу у цьому випадку сягає 2–5 разів порівняно з традиційним підходом (рис. 14).

У випадку додавання нових сутностей, відношень або обмежень в онтологію зі збереженням наявних її частин без змін, проектувальник онтології проводить валідацію запропонованих змін. Він аналізує ці зміни на предмет відсутності конфліктів з наявними елементами онтології та обгрунтовує доцільність модифікації онтології. Складність проведення модифікації залежить від кількості нових елементів, доданих до онтології:

$$C_{md-ont-add} = K_{mod} \times (T_{an} + T_{mon}) + T_{md}, \tag{30}$$

де $C_{mod\text{-}ont\text{-}add}$ – оцінка складності; K_{mod} – кількість доданих до онтології компонент; T_{an} – час на аналіз та валідацію; T_{mon} – час на проведення модифікації у редакторі онтологій з розрахунку на один компонент; T_{md} – час на модифікацію залежних від введених елементів онтології моделей.

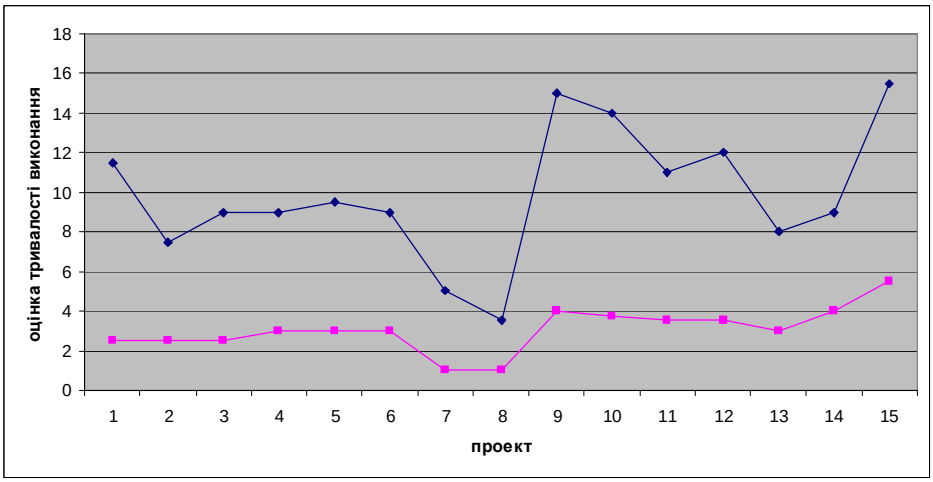

Рис. 14. Оцінки тривалості виконання другого типу в програмних системах

Аналіз змін з вибірки реальних проектів показав, що зменшення часу в цьому випадку сягає 1,5–3 рази порівняно з традиційним підходом (рис. 15).

Причиною зміни існуючих компонент онтології є виявлення змістових та логічних помилок. Щоб змінити елементи онтології, треба змінити та перетестувати усі моделі, які працюють зі зміненими елементами. Така задача є доволі трудомісткою, адже кількість моделей у репозиторії моделей для великої програмної системи є значною. Архітектура та функціональні можливості системи моделювання дають змогу визначити усі моделі, на які впливають зміни в онтології. Для кожної такої моделі необхідно проаналізувати вплив змін, визначити обсяг необхідних модифікацій, провести зміну моделі та перетестувати змінену модель:

$$C_{md-on-rev} = (K_{on} \times T_{on}) + K_{md-on} \times (T_{an} + T_{md-mod} + T_{test}), \quad (31)$$

де $C_{mod\text{-}on\text{-}rev}$ – оцінка складності; K_{on} – кількість змінених елементів онтології; T_{on} – час, що витрачається на модифікацію одного елемента онтології; $K_{md\text{-}on}$ – кількість моделей, які необхідно змінити; T_{an} – час на аналіз та проектування зміни в моделі; $T_{md\text{-}mod}$ – час на проведення зміни; T_{test} – час на тестування.

Аналіз змін з вибірки реальних проектів показав, що зменшення часу в цьому випадку сягає 1,2–2 рази порівняно з традиційним підходом, але істотно збільшуються загальні часові витрати на проведення модифікації (рис. 16).

Результати аналізу ступеня зменшення часу на проведення модифікації програмної системи для вказаних вище випадків показують значний розкид оцінок залежно від особливостей конкретних проектів. Водночас доцільно ранжувати їх на основі експертних оцінок. Порівнюючи складності, отримуємо шкалу оцінок, в якій складність зростає:

$$(C_{md}, C_{md-ont-add}, C_{md-ont-rev}, C_{m-old}). \quad (32)$$

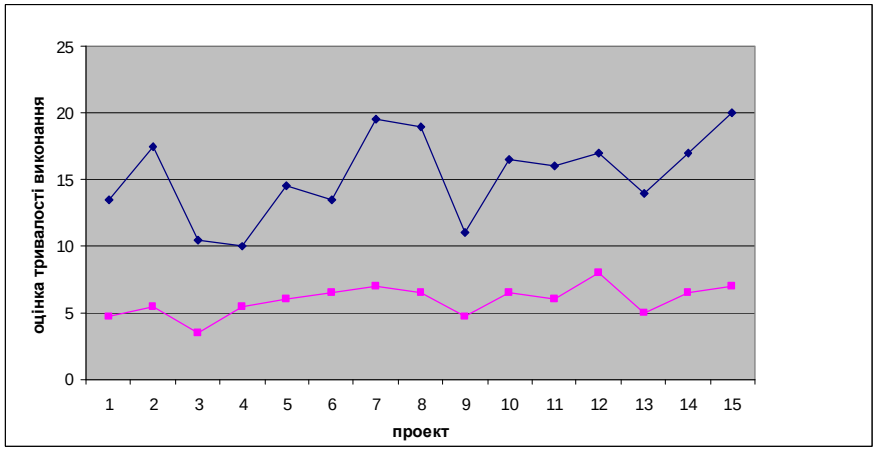

Рис. 15. Оцінки тривалості виконання змін третього типу в програмних системах

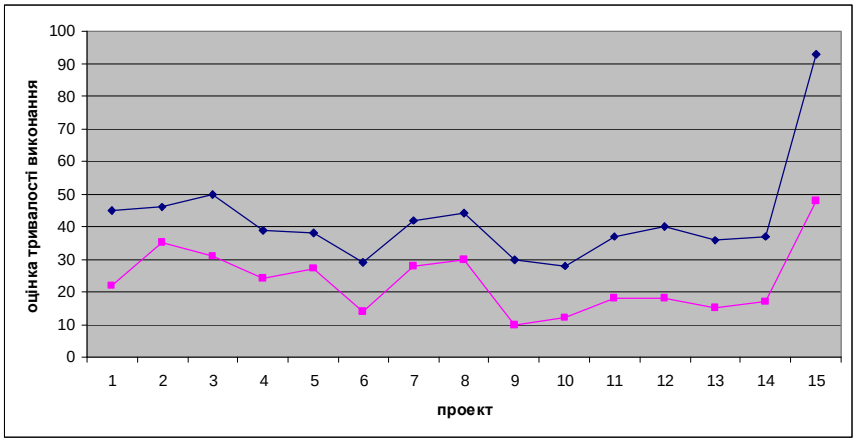

Рис. 16. Оцінки тривалості виконання змін четвертого типу в програмних системах

Отже, використання онтологічних моделей для побудови програмних систем призводить до покращення таких характеристик якості, як *коректність*, *кількість дефектів*, *відповідність функціональним вимогам* та *зручність впровадження змін*.

Інструментальний засіб для побудови програмних систем на базі онтологічних моделей

Для проведення досліджень програмних систем на базі онтологічних моделей задач створено прототип інструментального засобу середовища їх онтологічного моделювання.

Основні функції цього програмного комплексу такі:

- робота з онтологією: створення та модифікація класів та відношень; визначення обмежень для атрибутів;
- робота з базою фактів: створення та модифікація окремих фактів, які є екземплярами класів онтології, перевірка обмежень та валідація фактів;
- створення та модифікація онтологічних моделей з використанням класів та фактів онтології; визначення додаткових обмежень; визначення елементів у контексті моделі, реалізація їх пошуку; визначення операцій та їх прив'язка до сервісів виконання; визначення операцій, що виконуються залежно від результатів виконання моделі;
- виконання моделей та мереж моделей: запуск та виконання моделей на тестовій базі фактів, перевірка відповідності поведінки моделей очікуваним результатам.

Для організації збереження даних моделювання розроблено відповідні структури даних на основі мови XML. Створено також XML-орієнтовані мовні засоби для подання та опрацювання онтологічних моделей, визначено протокольні засоби взаємодії із сервісами середовища виконання моделей.

Прототип системи моделювання складається з чотирьох компонент, об'єднаних спільним інтерфейсом: *Редактора Онтологій*, *Редактора фактів*, *Редактора Моделей та Програми Моделювання*. Цей факт відображено і у робочому вікні середовища (рис. 17), в якому наявні вкладки редактора онтологій, редактора фактів, редактора моделей та програми моделювання.

Основними вимогами до системи моделювання є гнучкість, можливість розширення, швидкість розроблення та модифікації, підтримка можливості реалізації графічного інтерфейсу. Відповідно до цих вимог у ролі програмної платформи для створення прототипу вибрано мову Python (версії 2.7) та графічну бібліотеку PyQt, яка є перенесенням на платформу Python відомої відкритої та безплатної бібліотеки Qt. Мова Python є доволі простою у використанні та модифікації і широко застосовується як мова для побудови програмних прототипів. Програму середовища моделювання створено з використанням патерну проектування MVC (Model-View-Controller).

Функціональні можливості прототипу середовища моделювання проілюстровано на прикладах розв'язання деяких практичних задач, зокрема задач інтервальної класифікації та автоматизованого тестування (рис. 18, 19).

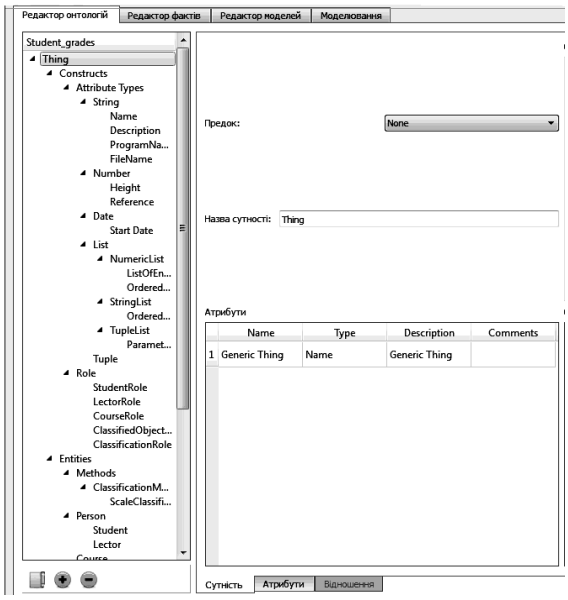

Рис. 17. Структурні компоненти середовища моделювання (а) та головне вікно середовища моделювання

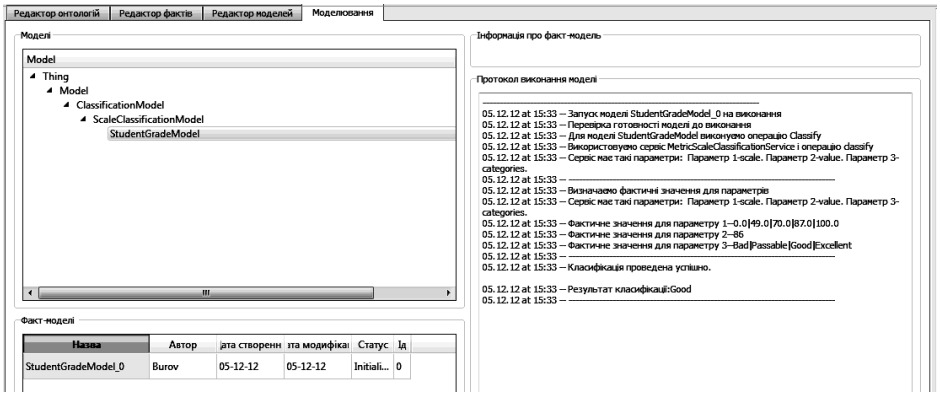

Рис. 18. Розв'язання задачі класифікації у середовищі моделювання

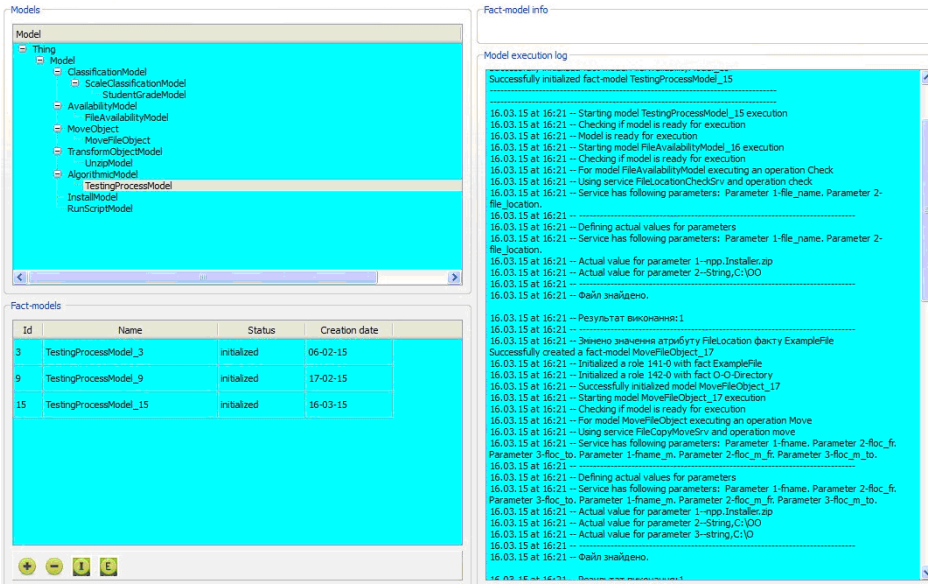

Рис. 19. Розв'язання задачі автоматизованого тестування програмного продукту

Подальший розвиток системи моделювання передбачає розширення функцій взаємодії моделей, реалізації механізмів логічного виведення на основі онтології, автоматичне та асинхронне виконання моделей.

Висновки та перспективи подальших розвідок

У результаті проведених досліджень вирішена актуальна науково-прикладна проблема застосування онтологічних моделей задач для побудови інтелектуальних програмних систем; обгрунтована доцільність використання онтологічних моделей задач, розроблена математична формалізація подання знань, визначено структуру та принципи функціонування програмної системи на базі онтологічних моделей, проаналізовано переваги використання онтологічних моделей порівняно з традиційним підходом, розроблено прототип інструментальної системи моделювання. Дослідження, результати яких подані у статті, проводились впродовж 2007–2015 років на кафедрі інформаційних систем та мереж Національного університету "Львівська політехніка". Практичне впровадження та верифікацію здійснено в компаніях-лідерах української ІТ-галузі, таких як "Софтсерв", "Елекс", отримані фахові відгуки є схвальними. В 2015 р. результати наукових розвідок за цим науковим напрямом захищено у формі докторської дисертаційної роботи за науковою спеціальністю "Математичне та програмне забезпечення обчислювальних машин та систем". Водночас у контексті отриманих результатів поки що не вирішеними залишилися питання використання дескриптивної логіки та механізмів логічного виведення у системі на базі моделей задач. Подальшого розроблення потребують методи взаємодії моделей, реалізація багатоваріантних обчислень, контекстозалежні обчислення на базі моделей.

1. Koskinen J. Software Maintenance Costs / J. Koskinen // Information Technology Research Institute, ELTIS-Project. – University of Jyväskylä. – 2003.2. Lehman Meir M Software Evolution – Background, Theory, Practice / Meir M. Lehman, and F. Ramil Juan // Information Processing Letters. – 2003. – P.33–44. 69. 3. Balmelli L. Model-driven systems development / L. Balmelli, D. Brown, M. Cantor, M. Mott // IBM Systems Journal, 2006. – Vol. 45. – P. 569–585. 4. Mellor S. J. Executable UML: A foundation for model-driven architectures / S. J. Mellor, M. J. Balcer // Addison-Wesley, 2002. 5.MDA: Nice idea, shame about the: [Electronic resource]. – Access mode: http://www.theserverside. com/ news/1365166/MDA-Nice-idea-shame-about-the. 6. Ross G. Principles of the Business Rule Approach / Ronald G. Ross. – Addison-Wesley Professional. – 2003. – P. 372. 7. Van Harmelen, Frank, Vladimir Lifschitz and Bruce Porter. Handbook of Knowledge Representation / Van Harmelen, Frank Vladimir Lifschitz and Bruce Porter. – Vol. 1. – Elsevier, 2008. – P.1034. 8. Фридман Л. М. Основы проблемологии / Л. М. Фридман // Серия: Проблемология. – М: Синтег, 2001. – С. 228. 9. Johnson P. Task-Related Knowledge Structures:Analysis, Modelling and Application / P. Johnson, H. Johnson, R. Waddington, and A. Shouls // Knowl. Creat. Diffus. Util. – 1988. – P.35–62. 10. Van Welie M. An Ontology for Task World Models. / M. Van Welie, G. C. Van Der Veer, and A. Eliëns // Methods. – 1998. – Vol. 98. – P. 57–70. 11. Taylor P. Ontology-Based Task Simulation / P. Taylor, M. Raubal and W. Kuhn // Spat. Cogn. Comput. – 2004. – Vol. 4, no. 917247301. – P. 15–37. 12. Буров Є. В. Концептуальне моделювання інтелектуальних програмних систем: монографія / Є. В. Буров. – Львів: Вид-во Львівської політехніки, 2012. – С. 432. 13. Kazuhisa S. Building ontologies for conceptual model management / Seta Kazuhisa, Koyama Kazuya, Hayashi Yusuke, Ikeda Mitsuru // WSEAS Transactions on Information Science and Applications. – 2006. – Vol. 3. – P. 546–553. 14. Koo B. Algebra of systems: a metalanguage for model synthesis and evaluation / B. Koo, W. Simmons // IEEE Transactions on systems, man and cybernetics. – 2009. – Vol. 39, N 3. – P.501–513. 15. Pezzulo G., Schema-Based Design and the AKIRA Schema Language: An Overview / G. Pezzulo, G. Calvi // Anticipatory Behavior in Adaptive Learning Systems/ – 2007. – P. 128–152. 16. ISO/IEC/IEEE 29119–2: Test Processes[Electronic resource]. – Access mode: http://www.softwaretestingstandard.org/part2.php. 17. Olive A. Conceptional modeling of information system / A. Olive. – Berlin Heidelberg: Springer, 2007. – P.471. 18.Osterwalder A., Business Model Ontology – a proposition in a design science approach / A. Osterwalder. – Citeseer, 2004. – P. 172. 19.Burov Y. Business process modelling using ontological task models / Y. Burov // Econtechmod. – Lublin : Polish Academy of Sciences, 2014. – № 1. – P. 11–23. 20. Буров Є. В. Використання моделей для керування доступом до ресурсів інтелектуальної інформаційної системи / Є. В. Буров, А. В. Гульова // Вісник Нац. ун-ту "Львівська політехніка" "Інформаційні системи та мережі". – Львів: Вид-во Львівської політехніки, 2010. – № 673. – С. 59–68. 21. Буров Є. В. Консенсус експертів як засіб для підвищення достовірності моделей знань в інтелектуальній системі / Є. В. Буров, М. Б. Крамаренко // Вісник Нац. ун-ту "Львівська політехніка" "Комп'ютерні науки та інформаційні технології". – Львів: Вид-во Львівської політехніки, 2011. – № 694. – С. 212–220.