УДК 004.89

Є.В. Буров
Національний університет “Львівська політехніка”,
кафедра інформаційних систем та мереж

ЗАСТОСУВАННЯ ОНТОЛОГІЧНИХ МОДЕЛЕЙ ДЛЯ ПОБУДОВИ ПРОГРАМНИХ СИСТЕМ

© *Буров Є.В., 2012*

Розглянуто математичну формалізацію програмної системи, побудованої на основі онтологічних моделей за допомогою апарату алгебраїчної теорії типів. Розроблено формальне подання моделей та системи їх опрацювання.

Ключові слова: база знань, математична модель, онтологія, модель знань.

In this paper mathematical formalization for software system based on ontological models is proposed. Formalization is built using algebraic types system approach. Developed formal representation of models and modeling system.

Keywords: knowledge base, ontology, algebraic type system, knowledge model, algebraic type.

Вступ та постановка проблеми

Збільшення ступеня інтеграції бізнес-процесів підприємств, розвиток електронної комерції та глобалізація світової економіки підвищують темпи змін бізнес-середовища і вимагають постійної адаптації програмних систем до змінного середовища їх функціювання.

Традиційні методи проектування та архітектурні принципи побудови програмних систем орієнтовані на створення програмного продукту на основі фіксованого набору вимог. У результаті отримують системи, які слабо реагують на зміну вимог та зовнішніх чинників, які дорогі у підтримці та експлуатації і в яких нерідко трапляються збої та аварії [1].

Врахування змінених вимог, як правило, спричиняє випуск нової версії пограмного продукту, що передбачає тривалі та ресурсомісткі етапи аналізу, дизайну, кодування, тестування та впровадження нової версії продукту.

Адаптація програмної системи вимагає врахування знань експертів відповідних предметних областей. Як правило, такі експерти не є спеціалістами у програмуванні. Отже, необхідно чітко формулювати вимоги експертів до розробників системи. Помилки та непорозуміння у керуванні вимогами до системи під час концептуалізації становлять значну частину причин невдач у розробленні програмних продуктів [2].

Аналіз останніх досліджень і публікацій

Одним з шляхів вирішення зазначеної проблеми є відокремлення логіки функціювання програмної системи від механізму її опрацювання та реалізації. При цьому логіка функціювання подається у вигляді певної формальної моделі.

Одним із способів реалізації такого підходу є MDD/MDA (Model – driven design/model-driven architecture), започаткований у роботах Шлаера (Shlaer) та розвинений Меллором (Mellor) [3]. Цей підхід полягає у формалізації програми у вигляді моделей та подальшої їх компіляції у код. Багаторічне використання цього підходу, крім переваг, виявило і його значні недоліки, зокрема складність створення та модифікації комплексу моделей, яка є порівнянною зі складністю створення системи традиційним шляхом.

Іншим способом реалізації MDD є використання онтологій для побудови програмних систем. За визначенням Грубера [4], онтологія є формальною моделлю концептуалізації предметної області. Така модель містить визначення сутностей передметної області та залежностей між ними. Будуючи програмні системи на основі онтологій уникають повторної концептуалізації предметної області, що дає змогу скоротити використання ресурсів на етапі аналізу та дизайну системи. Перепоною до використання онтології під час побудови програмних систем є суто декларативний характер онтологій, відсутність подання в них процедурних знань.

Lviv Polytechnic National University Institutional Repository http://ena.lp.edu.ua

Формулювання цілі статті

У [5] було запропоновано підхід до побудови програмних систем з використанням інтерпретованих онтологічних моделей, архітектуру та порядок роботи системи моделювання. При цьому недостатньо висвітленими залишається аспект математичної формалізації запропонованого підходу, зокрема подання знань з використанням онтологій та моделей, залежностей між ними та питання практичної реалізації системи моделювання.

Метою цієї роботи є побудова математичної моделі подання знань в інтелектуальній системі, що грунтується на онтологічних моделях.

Виклад основного матеріалу

Структура та порядок роботи системи моделювання

Система моделювання, що реалізує запропонований підхід, складається з таких компонент (рис.) База фактів містить факти про об’єкти та події зовнішнього світу, необхідні для розв’язання задач системою. Всі факти семантично інтерпретовані, тобто подані як об’єкти певних класів, визначених онтологією. Інформація в базі фактів впорядкована за часом та змінами, що дає змогу відслідкувати стан бази на довільний момент часу або на довільну зміну. База фактів, онтологія та моделі у сукупності утворюють базу знань системи.

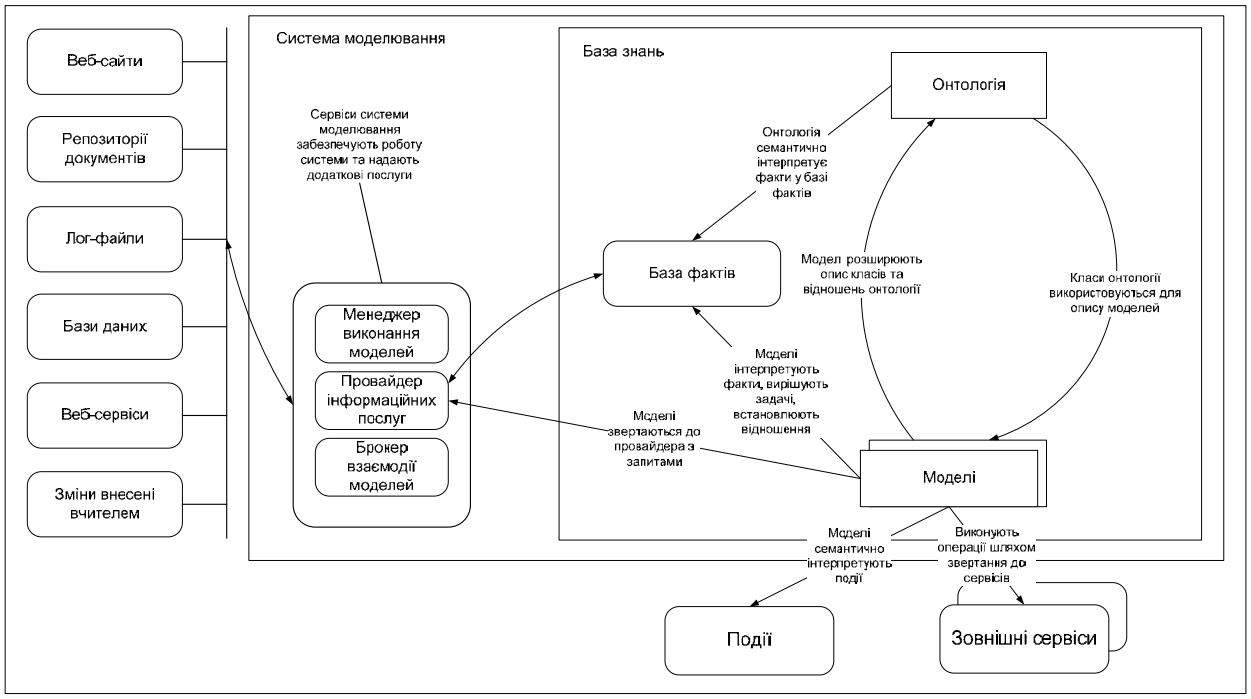

Рис. 1. Структура системи моделювання

Онтологія містить модель предметної області, подану як таксономія класів. Це створює можливість однозначного трактування усіх об’єктів з бази фактів, визначення єдиної структури слотів та типів властивостей для них. Крім того, онтологія містить відношення, правила та обмеження, які мають загальний, системний характер.

Посилання на моделі використовуються в онтологіях для зберігання складних правил та обмежень, зокрема динамічних обмежень, значення яких залежить від стану бази знань або зовнішнього світу.

Система реагує на визначене коло подій зовнішнього світу та опрацьовує іх, створюючи нові або модифікуючи існучі факти. Для опрацювання цих подій використовуються відповідні моделі.

Важливими компонентами системи є сервіси системи моделювання, які забезпечують виконання моделей, їх взаємодію, пошук потрібної інформації у зовнішніх джерелах.

Так, Менеджер виконання моделей реалізує запуск або зупинку моделей, відслідковує використання ресурсів моделей. Брокер взаємодії моделей реалізує пошук релевантних моделей для

Lviv Polytechnic National University Institutional Repository http://ena.lp.edu.ua

вирішення задач, ініціалізує та запускає обрані моделі. Провайдер інформаційних послуг за запитом моделі системи звертається до зовнішніх джерел, реалізує пошук необіхних даних та їх семантичну інтерпретацію. Такими зовнішніми джерелами є, наприклад, бази даних, документи, веб-сервіси та ін. Важливим зовнішнім джерелом для надходження інформації у систему є коментарі та зміни, внесені вчителем.

Моделі виконують операції, відповідно до відображеної в них логіки, звертаючись з запитами до зовнішніх відносно системи моделювання сервісів. Такими сервісами є сервіси операційної системи, сервіси інформаційної системи підприємства, побудованої з дотриманням вимог SOA [10], або довільні веб-сервіси.

Математична модель подання та опрацювання знань у системі моделювання

Визначимо онтологію On як множину символів сутностей $\overline{E}$ та відношень між ними $\overline{R}$:

$$On = \{\overline{E}, \overline{R}\}.$$

Кожне відношення $R \in \overline{R}$ задане на множині ролей $\{P_1, P_2, ..., P_n\}$:

$$R(P_1, P_2, ..., P_n).$$

У загальному випадку для кожної ролі P_k визначено функцію ініціалізації $F_{in}{}^k$, яка визначає підмножину сутностей, елементам якої дозволено ініціалізувати роль:

$$F_{in}{}^k : P_k \to E_{in}{}^k \subseteq \overline{E}.$$

У найпростішому випадку, коли $\forall k : | E_{in}{}^k | = 1$, для кожної ролі існує тільки один тип сутностей, яким вона може бути ініціалізована. У цьому випадку можна у позначенні відношення замінити ролі відповідними сутностями онтології: $R(E_1, E_2, ..., E_n)$.

Сутності онтології утворюють ієрархічну структуру (таксономію) з використанням відношення успадкування (isa-відношення) R_{isa}.

Бінарне відношення R_{isa} задане на впорядкованій парі ролей *Нащадок-Предок*:

$$R_{isa}(P_{ch}, P_{pr}).$$

Відношення успадкування є транзитивним, так що, якщо $R_{isa}(E_1, E_2)$ та $R_{isa}(E_2, E_3)$ то справедливе $R_{isa}(E_1, E_3)$.

Визначимо функцію F_{pr}, яка для кожної сутності E_j визначає впорядкований список її предків $(E_1, E_2, ..., E_k)$, так що виконуються $R_{isa}(E_i, E_{i+1})$ та $R_{isa}(E_j, E_1)$. Визначимо також функцію $F_{pr}{}^1(E_j)$, яка повертає безпосереднього предка сутності E_j або порожню множину $\varnothing$

Важливим різновидом відношення є відношення *Ціле–Частина* (has-parts відношення): $R_{has}(P_{wh}, P_{pt})$, де P_{wh} – роль цілого, а P_{pt} – роль частин цього цілого. Підтип такого відношення R'_{has} визначає конкретні сутності для цілого, та набори дозволених сутностей – для частин:

$$R'_{has}(E'_{wh}, \{E^1_{pt}, E^2_{pt}, ..., E^n_{pt}\}').$$

Для простоти при поданні такого відношення ми будемо використовувати нотацію:

$$E_{wh} = \{E^1_{pt}, E^2_{pt}, ..., E^n_{pt}\}.$$

Для подання системи моделювання визначимо алгебраїчну систему абстрактних типів даних $\overline{T}$, в якій для кожного елемента *T* існує взаємнооднозначна відповідність з певною сутністю онтології.

$$\forall i \exists^1 j : T_i \to E_j$$

$$\forall j \exists^1 i : E_j \to T_i$$

Функція *TypeEn*() повертає для кожного типу даних T_i з $\overline{T}$ відповідну сутність онтології E_j:

$$TypeEn(T_i) = E_j$$

Lviv Polytechnic National University Institutional Repository http://ena.lp.edu.ua

і так визначає семантичну інтерпретацію цього типу даних.

Відповідно до підходу з [11], абстрактний тип даних алгебраїчно подається трійкою

$$T = (Name, \Sigma, Ex)\ , \ (1),$$

де *Name* – назва типу, Σ – сигнатура багатосортної алгебри, *Ex* – множина рівнянь у сигнатурі Σ, що специфікує визначальні співвідношення абстрактного типу даних.

Сигнатура

$$\Sigma = (S, OP)\ ,$$

де *S* – множина імен базових множин, а *OP* – множина імен операцій:

$$S = \{S_1, S_2, ..., S_n\}\ ;$$

$$OP = \{F_1, F_2, ..., F_k\}\ .$$

Кожна операція F_i визначає відображення

$$F_i : S_{a(1,i)} \times ... \times S_{a(n,i)} \to S_{m(i)}\ .$$

Відношення *R* є різновидом алгебраїчної операції, що діє на визначеному наборі типів-аргументів та визначає відображення у булеву множину

$$R_i : S_{a(1,i)} \times ... \times S_{a(n,i)} \to S_{BOOL} = \{true, false\}\ .$$

Модель *Md* також можна розглядати як різновид операції, що діє на визначеному наборі типів аргументів та визначає відображення у множину результатів, які в загальному випадку мають різні типи:

$$Md_i : S_{a(1,i)} \times ... \times S_{a(n,i)} \to \{S_{r(1,i)}, ..., S_{r(l,i)}\}\ .$$

З іншого боку, відношення та моделі на рівні онтології є окремими сутностями. Цим сутностям в алгебраїчній системі типів $\overline{T}$ відповідають типи даних, екземпляри яких зберігають дані про ці відношення та моделі (метадані).

Важливою складовою визначення відношень в онтології є обмеження цілісності, що накладаються на сутності, поєднані відношенням. У визначеній системі типів таким обмеженням відповідає набір булевих виразів $Cs_{T_{REL}}$ для кожного типу відношення T_{REL}, які повинні бути істинними:

$$\forall {cs^i}_{T_{REL}} \in Cs_{T_{REL}} : ev({cs^i}_{T_{REL}}) = true\ ,$$

де *ev*() – функція оцінюванння значення виразу.

Вирази, які визначають обмеження цілісності, є складовою множини визначальних співвідношень абстрактного типу даних:

$$Cs_{T_{REL}} \subseteq Ex_{T_{REL}}\ .$$

У деяких випадках обмеження накладають і на значення атрибутів сутностей. Розглянемо такі обмеження як обмеження цілісності для унарних відношень.

Аналогічно до сутностей онтології, типи даних з $\overline{T}$ утворюють ієрархію типів, в якій підтип T_{pid} успадковує властивості супертипу $T_{\sup}$ тоді і тільки тоді, якщо між сутностями онтології, що відповідають підтипу та супертипу, визначено відношення R_{isa}, тобто:

$$\exists R_{isa}(TypeEn(T_{pid}), TypeEn(T_{\sup}))\ .$$

Подібно до сутностей онтології, типи даних будують на основі інших, простіших типів. При цьому ці складніші типи фактично є структурами, що містять елементи простіших типів. Позначимо тип-структуру як T_{wh}, а типи – складові частини як T_{pt}^i. Тоді

$$T_{wh} = \{T_{pt}^1, T_{pt}^2, ..., T_{pt}^n\}\ ,$$

якщо відповідні типам даних сутності онтології сполучені відношенням R'_{has}:

$$\exists R'_{has}(TypeEn(T_{wh}), \{TypeEn(T_{pt}^1), TypeEn(T_{pt}^2), ..., TypeEn(T_{pt}^n)\}')\ .$$

Визначимо функцію *TypeParents*(), яка для кожного типу T повертатиме впорядковану множину його супертипів і є транзитивним замиканням відношення успадкування

Lviv Polytechnic National University Institutional Repository http://ena.lp.edu.ua

$$TypeParents(T) = \{T^1, T^2 ..., T^n\},$$

де T^{i+1} є супертипом відносно T^i

Визначимо функцію *TypeName*(), яка для кожного типу даних T повертає унікальний ідентифікатор (опис, назву) цього типу. Наприклад, для типу даних T_{MD}, що відповідає моделі:

$$TypeName(T_{MD}) = "Model".$$

Для екземплярів t довільного типу T визначимо функції, що повертають тип, та відповідну сутність онтології:

$$Type(t) = T;$$

$$Entity(t) = TypeEn(Type(t)) = E.$$

Мультимножину усіх екземплярів типу T позначимо $Population(T)$:

$$Population(T) = \{t \mid Type(t) = T\}.$$

Позначимо мультимножину екземплярів певного типу T_i як $\hat{t}_i$:

$$\hat{t}_i \mid \forall t_i \in \hat{t}_i : Type(t_i) = T_i;$$

$$\hat{t}_i \subseteq Population(T_i).$$

Абстрактний тип даних, що відповідає мультимножині еземплярів $\hat{t}_i$, позначимо $\hat{T}_i$.

У загальному випадку довільний програмний об'єкт o можна ідентифікувати як об'єкт багатьох типів.

Визначимо функцію *TypeId*(), яка повертатиме множину типів, якими можна ідентифікувати об'єкт.

$$TypeId(o) = \{T_o^1, T_o^2, ..., T_o^m\}.$$

Оскільки між сутностями онтології та типами даних існує ізоморфне відображення, доцільно іменувати типи даних так само, як відповідні сутності онтології. Тобто:

$$TypeName(T) = Name.$$

У тих випадках, коли необхідно наголосити на семантичній інтерпретації певного типу даних позначимо скорочену назву типу у вигляді індекса. Наприклад, позначимо тип даних моделі як T_{MD}, конкретний елемент цього типу – t_{MD} або мультимножину екземплярів моделей – $\hat{t}_{MD}$.

Враховуючи наведені позначення, визначимо тип бази знань як абстрактний тип даних, T_{BKN}, поданий множиною (структурою), що містить тип бази фактів T_{BFC}, тип онтології T_{ON} та тип мультимножини моделей $\hat{T}_{MD}$:

$$T_{BKN} = \{T_{BFC}, T_{ON}, \hat{T}_{MD}\}.$$

Екземпляр бази знань t_{BKN} у кожен момент часу подано структурою $t_{BKN} = \{t_{BFC}, t_{ON}, \hat{t}_{MD}\}$ в якій $t_{BFC}, t_{ON}, \hat{t}_{MD}$ відповідають екземплярам бази фактів, онтології та мультимножини моделей.

База фактів t_{BFC} – це множина фактів про об'єкти та події зовнішнього світу та відношення між ними.

$$t_{BFC} = \{\hat{t}_{FC}, \hat{t}_{RF}\}.$$

Елементами мультимножини $\hat{t}_{FC}$ є екземпляри даних, що мають тип факту – T_{FC}, $TypeName(T_{FC}) = "Fact"$.

З іншого боку, ці елементи також мають один з типів T_{pid}, який є похідним від T_{FC}, тобто:

$$\exists R_{isa}(TypeEn(T_{pid}), TypeEn(T_{FC})).$$

Кожен факт та відношення є семантично – інтерпретованим, тобто його тип визначено в онтології On:

$$\forall t_{FC}^i : Entity(t_{FC}^i) \neq \varnothing;$$

$$\forall t_{RC}^i : Entity(t_{RC}^i) \neq \varnothing.$$

Lviv Polytechnic National University Institutional Repository http://ena.lp.edu.ua

Тип даних містить типи мультимножини визначень класів $\hat{T}_{CL}$ та відношень між ними – $\hat{T}_{RCL}$.

$$T_{ON} = \{\hat{T}_{CL}, \hat{T}_{RCL}\}.$$

Кожен клас T_{CL} визначається набором атрибутів $\hat{T}_{SL}$, правил $\hat{T}_{RU}$, обмежень $\hat{T}_{CS}$, що визначені на цих атрибутах:

$$T_{CL} = \{\hat{T}_{SL}, \hat{T}_{RU}, \hat{T}_{CS}\}.$$

На рівні онтології j-й атрибут A_i^j сутності E_i подається атрибутивним відношенням, що поєднує цей атрибут з іншою сутністю E_k:

$$R_{atr}(A_i^j, E_k),$$

так що для кожного екземпляра атрибуту a_i^j його значення визначається екземпляром e_k.

Кожен атрибут задано типом його значень T_{VSL}, мультимножинами правил та обмежень, що діють на рівні атрибута – $\hat{T}_{RUS}, \hat{T}_{CSS}$

$$T_{SL} = \{T_{VSL}, \hat{T}_{RUS}, \hat{T}_{CSS}\}.$$

Значення атрибута t_{SL} належить множині допустимих значень RgVSL:

$$\forall i : t_{SL}^i \in RgSL.$$

Обмеження T_{cs} визначає відображення множини значень атрибутів у множину булевих значень {true, false}:

$$T_{cs} : \hat{T}_{SL} \to \{true, false\}.$$

Правило T_{RU} визначає відображення однієї підмножини значень атрибутів в іншу. З кожним правилом T_{RU}^j пов'язані дві підмножини A_{bas}^j, A_{der}^j атрибутів класу, що не перетинаються:

$$T_{RU}^j : A_{bas}^j \to A_{der}^j;$$
$$A_{bas}^j \cap A_{der}^j = \varnothing.$$

Тип відношення між класами T_{RCL} задано на впорядкованій послідовності типів класів, які воно сполучає: $(T_{CL}^1, T_{CL}^2, ..., T_{CL}^n)$.

Відношення між класами само є класом у тому розумінні, що визначається набором атрибутів, правил та обмежень:

$$T_{RCL} = \{(T_{CL}^1, T_{CL}^2, ..., T_{CL}^n), \hat{T}_{SL}, \hat{T}_{RU}, \hat{T}_{CS}\}.$$

Такий підхід дає змогу розглядати відношення як окремі типи даних та передбачити можливість формування структур відношень.

Тип факту бази знань T_{FC} є супертипом для типів класів T_{CL} та відношень T_{RCL}:

$$\exists R_{isa}(TypeEn(T_{CL}), TypeEn(T_{FC})),$$
$$\exists R_{isa}(TypeEn(T_{RCL}), TypeEn(T_{FC}))$$

Моделі утворюють мережу, що має тип T_{NMD}, який визначається як множина, до якої входять мультимножини типу моделей $\hat{T}_{MD}$ та по відношень $\hat{T}_{RMD}$:

$$T_{NMD} = \{\hat{T}_{MD}, \hat{T}_{RMD}\}.$$

На відміну від класів онтології, моделі не утворюють чіткої ієрархії і формують динамічну мережу, в якій відношення та самі моделі можуть змінюватися, відображаючи процеси навчання, зміни у зовнішньому світі або процес розв'язання певної задачі.

Кожна модель може бути в одному з двох станів – активному або пасивному. Відповідно, множину екземплярів моделей поділимо на підмножини активних та пасивних моделей, які не перетинаються:

Lviv Polytechnic National University Institutional Repository http://ena.lp.edu.ua

$$Population(T_{MD}) = \hat{t}_{MD}^{ac} \cup \hat{t}_{MD}^{ps};$$
$$\hat{t}_{MD}^{ac} \cap \hat{t}_{MD}^{ps} = \varnothing,$$

де $\hat{t}_{MD}^{ac}, \hat{t}_{MD}^{ps}$ позначено мультимножини екземплярів активних та пасивних моделей.

Активна модель – це модель, ініціалізована інформацією з певного контексту. Моделі переходять в активний стан на вимогу інших моделей, або у разі настання певних подій. Активні моделі використовують для розв'язання поточних задач системи, інтерпретації знань у системі. Якщо потреба в моделі відпала (отримано результат, досягнуто мети), то модель переходить з активного у пасивний стан.

Взаємодія моделей T_{RMD} – це тип даних, що відображає відношення активації, яке використовується для прийняття рішення щодо активації моделі (моделей).

Розглянемо процес та структуру взаємодії та активації моделей детальніше. Ініціатором встановлення зв'язку між моделями є модель – активатор. Потреба у встановленні зв'язку виникає не завжди, а тільки тоді, коли для розв'язання основної задачі потрібно вирішити допоміжні задачі, подані в інших моделях. Наприклад, якщо вхідні дані, отримані активатором з контексту, є неповними і необхідно активізувати інші моделі для довизначення даних. Таке довизначення може полягати в пошуку потрібної інформації в базі даних, інтернеті, звертанні до консультанта тощо.

Кожному типу T_{RMD}^{i} відповідає клас задач, які необхідно розв'язати T_{PR}^{j} у результаті взаємодії. Своєю чергою, класу задач T_{PR}^{j} відповідає множина моделей $\hat{T}_{MD}^{j}$, які можна використати для розв'язання задач цього класу.

Під час взаємодії моделей послідовно розв'язують задачі визначення релевантності, оптимального вибору серед релевантних моделей та ініціалізації обраної моделі.

Функція релевантності є відображенням поточного контексту t_{CON} та множини альтернатив $\hat{t}_{MD}$ у множину {true, false}

$$F_{rel} : t_{CON}, \hat{t}_{MD} \to (true, false).$$

Визначенням релевантності моделей можна відібрати для процедури вибору тільки релевантні моделі:

$$\hat{t}_{MD}^{re} \subseteq \hat{t}_{MD},$$

тобто такі моделі t_{MD}^{re}, для яких

$$F_{rel}(t_{CON}, t_{MD}^{re}) = true.$$

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

Задача оптимального вибору визначає у множині релевантних моделей одну t_{MD}^{op}, застосування якої максимізує функцію вибору F_{ch} з врахуванням критеріїв вибору $\hat{t}_{CR}$ та контексту t_{CON}.

$$F_{ch}(t_{MD}^{op}, \hat{t}_{CR}, t_{CON}) \to \max.$$

Функція ініціалізації F_{in} відображає поточний контекст t_{CON} у множину значень атрибутів обраної моделі – t_{VSL}

$$F_{in} : t_{CON} \to t_{VSL}.$$

Отже, визначаємо T_{RMD} як множину

$$T_{RMD} = \{T_{PR}, \hat{T}_{MD}, F_{rel}, F_{ch}, F_{in}\}.$$

Тип моделі T_{MD} складається з типів схеми T_{SCM} та реалізації T_{IMD}:

$$T_{MD} = \{T_{SCM}, T_{IMD}\}.$$

Lviv Polytechnic National University Institutional Repository http://ena.lp.edu.ua

Схема моделі описує її структуру, складові, визначає правила та обмеження на використання моделі, а також перелік можливих операцій та запитів. Схема є компонентом моделі, який видимий для зовнішнього світу. Її використовують для взаємодії з моделлю.

На рівні онтології модель визначається як кортеж з множин операцій $\hat{E}_{op}$, сутностей $\hat{E}$, відношень $\hat{R}$, додаткових обмежень $\hat{E}_{cs}$ та правил $\hat{E}_{ru}$:

$$E_{md} = (\hat{E}_{op}, \hat{E}, \hat{R}, \hat{E}_{cs}, \hat{E}_{ru}).$$

Схема моделі складається зі слотів $\hat{T}_{SLM}$, відношень між ними $\hat{T}_{RSM}$, правил $\hat{T}_{RUM}$, обмежень $\hat{T}_{CSM}$ та операцій $\hat{T}_{OPM}$:

$$T_{SCM} = \{\hat{T}_{RO}, \hat{T}_{RRO}, \hat{T}_{RUM}, \hat{T}_{CSM}, \hat{T}_{OPM}\}.$$

Слот моделі є атрибутом-роллю. Для кожного слоту визначено функцію F_{RG}, яка специфікує множину класів $\hat{T}_{CL}^{RG}$, об'єктами яких дозволено ініціалізувати слот:

$$F_{RG} : T_{SLM} \to \hat{T}_{CL}^{RG}.$$

Крім того, для слоту моделі визначено правила та обмеження, що діють на рівні слоту – $\hat{T}_{RUSM}$, $\hat{T}_{CSSM}$, операції над значеннями слоту $\hat{T}_{OPSM}$:

$$T_{SLM} = \{\hat{T}_{CL}^{RG}, \hat{T}_{RUSM}, \hat{T}_{CSSM}, \hat{T}_{OPSM}\}.$$

Відношення слотів T_{RESM} специфікується множиною слотів, яку він сполучає $\hat{T}_{SLM}^{resm}$, множиною класів онтології, які використовують для семантичної інтерпретації відношення – $\hat{T}_{CL}^{resm}$, множиною моделей, яку використовують для розуміння та проведення операцій з відношенням – $\hat{T}_{MD}^{resm}$.

$$T_{RESM} = \{\hat{T}_{SLM}^{resm}, \hat{T}_{CL}^{resm}, \hat{T}_{MD}^{resm}\},$$

Для кожного екземпляра відношення слоти, що ним сполучені, належать слотам моделі:

$$\hat{t}_{SLM}^{resm} \subseteq \hat{t}_{SLM}.$$

Відношення моделей відповідає одному з типів відношень, визначених в онтології On:

$$TypeEn(T_{RESM}) = E_{RESM} \in \bar{E}.$$

Модель, що описує відношення, є елементом загальної множини моделей:

$$t_{MD}^{resm} \in Population(T_{MD}).$$

Нехай t'_{BFC} – це певна ситуація, стан, знімок бази фактів. Визначимо тип даних мети T_{GL}, екземпляри якого є специфікаціями певної множини станів бази фактів, кожен з яких відповідає досягнутій меті:

$$t_{GL} = \hat{t}_{BFC}^{GL}.$$

Для визначення мети корисно задати функцію мети, за якою можна визначити, чи у певному стані t'_{BFC} мети Gl досягнуто:

$$F_{gl}(t'_{BFC}) = \begin{cases} true \mid t'_{BFC} \in t_{GL}; \\ false \mid t'_{BFC} \notin t_{GL}. \end{cases}$$

Одним з можливих способів завдання функції мети є її завдання у вигляді впорядкованого списку тверджень-вимог $\hat{t}_{ASR}$ щодо об'єктів інформаційної бази, які можна перевірити.

$$F_{gl}(t'_{BFC}) = \hat{t}_{ASR}(t'_{BFC}).$$

де $t_{ASR}(t'_{BFC})$ – це вимога-твердження (assertion) щодо значень властивостей об'єктів та їхніх зв'язків у ситуації t'_{BFC}.

Кожне твердження $t_{ASR}(t'_{BFC})$ є функцією, заданою на множині {true,false}:

$$Range(t_{ASR}(t'_{BFC})) = \{true, false\},$$

Lviv Polytechnic National University Institutional Repository http://ena.lp.edu.ua

де функція Range(f) задає область значення функції f.

Тобто,

$$F_{gl}(t'_{BFC}) = true \Leftrightarrow \forall t_{ASR}(t'_{BFC}) \in \hat{t}_{ASR}(t'_{BFC}) : t_{ASR}(t'_{BFC}) = true .$$

Виконувані онтологічні моделі призначені для розв'язання конкретних задач, специфікованих у вигляді мети. Для зручності пошуку потрібних моделей доцільно організувати інформацію про моделі у вигляді онтології $OnGl \subset On$, зі своїми сутностями та відношеннями. У цій онтології визначимо категорії моделей за класами задач, які вони розв'язують. Так, наприклад, окремо визначимо моделі класифікації, алгоритмічні моделі, моделі об'єктів та сервісів, моделі керування доступом, ситуаційні моделі. Інформацію про об'єкти, які описують окремі екземпляри моделей, використовує сервіс Менеджер Виконання моделей. Брокер взаємодії моделей використовує онтологію мети для пошуку моделі, потрібної для розв'язання поставленої задачі.

Висновок

Застосовуючи онтологічні моделі, можна підвищити ефективність розв'язання задач у різних предметних областях. Зокрема в роботі [12] було продемонстровано ефективність застосування такого підходу для розв'язання задач підтримки прийняття рішень, керування доступом до інформаційних ресурсів, автоматизованого тестування програмних продуктів, обслуговування клієнтів туристичної галузі.

Теоретичні та експериментальні дослідження з розроблення методів використання онтологічних моделей у системах бізнес-аналітики проводять у межах дербюджетної тематики на кафедрі "інформаційних систем та мереж" Національного університету "Львівська політехніка" протягом декількох років. Зокрема, розробляється дослідний макет інструментального програмного комплексу для моделювання систем з використанням онтологічного підходу та моделей. Описана в цій роботі математична модель подання знань є формальною основою для побудови інструментального комплексу та його валідації.

1. Balmelli, L. Model-driven systems development / Balmelli, L., Brown, D., Cantor, M., Mott, M. // IBM Systems Journal. – 2006. – vol.45. – P. 569–585. 2. Leffingwell D. Managing software requirements. A use-case approach. /Leffingwell Dean, Widrig Don.– Addison-Wesley Professional, 2003. – P. 544. 3. Mellor, S.J. Executable UML: A foundation for model-driven architectures / Mellor, S.J., Balcer, M.J. – Addison-Wesley, 2002. 4. Gruber, T.R. A translation approach to portable ontology specifications / Gruber, T.R. // Knowledge Acquisition. – 1993. – Vol 5. – P. 199–220. 5. Буров Є.В. Опрацювання знань у когнітивній інформаційній системі керованій моделями / Є.В. Буров // Східно-Європейський журнал передових технологій. – 2009. – №. 6/7(42). – С. 40–49. 6. Daconta, M.C. The Semantic Web: A Guide to the Future of XML, Web Services, and Knowledge Management. / M.C. Daconta, L.J. Obrst, K.T. Smith. – Wiley, 2003. 7. Pezzulo, G Schema-Based Design and the AKIRA Schema Language: An Overview / Pezzulo, G., Calvi, G // Anticipatory Behavior in Adaptive Learning Systems. – 2007. – P. 128–152. 8. MIT Encyclopedia of the cognitive sciences. Edited by Robert A. Wilson and Frank C. Keil. – MIT Press Cambridge, Massachusetts, London, England, 1999. – 1097 p. 9. Minsky, M. A Framework for Representing Knowledge / Minsky, M; P.H. Winston, ed // The Psychology of Computer Vision. – McGraw-Hill, 1975. – P. 211–277. 10. Erl, T. SOA Principles of Service Design / Erl, T. – Prentice Hall, 2007. 11. Бениаминов Е.М. Алгебраические методы в теории без данных и представлений знаний / Бениаминов Е.М..– М.: Научный мир, 2003. – С. 184. 12. Буров Є.В. Концептуальне моделювання інтелектуальних програмних систем / Є.В. Буров – Львів: Вид-во Львівської політехніки, 2012. – С. 432.

Lviv Polytechnic National University Institutional Repository http://ena.lp.edu.ua