УДК 004.89

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

АРХІТЕКТУРА ІНСТРУМЕНТАЛЬНОГО КОМПЛЕКСУ ДЛЯ МОДЕЛЮВАННЯ ІНТЕЛЕКТУАЛЬНИХ СИСТЕМ

Наведено опис архітектури інструментального комплексу для моделювання керованих моделями інтелектуальних систем. Визначено структури даних моделі, головні операції інструментальної системи, зокрема ініціалізація, перевірка релевантності та пошук моделі. Детально розглянуто архітектуру взаємодії моделей.

Ключові слова: керована моделями система, концептуальна модель, інтелектуальна система

Paper describes architecture, data structures and basic operations of instrumental modeling tool for building model-driven intellectual applications. Based on modeling system use case analysis, detailed description of model metadata structure, relevance checking, model initialisation along with model interaction are provided.

Keywords: model driven design, conceptual model, intellectual system

Постановка проблеми у загальному вигляді

Однією з важливих проблем галузі програмної інженерії є високий рівень складності програмних систем і пов'язані з нею проблеми складності та високої вартості адміністрування, розроблення та модифікації, значний рівень дефектів у таких системах. За оцінками національного інституту стандартів та технологій (NIST), щорічний збиток від дефектів програмного забезпечення для економіки США оцінюється у 59,6 млрд доларів. Тільки третину цих дефектів можна виправити із застосуванням традиційних методів контролю якості програмного забезпечення, таких як тестування [1].

Високий рівень складності зумовлює низку недоліків та вузьких місць, якими характеризуються наявні архітектури програмних систем та підходи до їх проектування. Зокрема, існують такі проблеми:

• проектування програмної системи основане на фіксованому наборі вимог до системи, які часто визначені нечітко. В результаті такого проектування отримують системи, що погано реагують на зміну вимог та зовнішніх чинників, які дорогі у підтримці та експлуатації і в яких нерідко трапляються збої та аварії [2]. Зміна вимог вимагає перепроектування системи;

• проектування найчастіше виглядає як циклічне повторення таких робіт, як "Визначення вимог", "Проектування архітектури", "Придбання обладнання", "Кодування програмного забезпечення", "Тестування", "Системна інтеграція", "Впровадження та експлуатація". Реалізація такого підходу зазвичай займає багато часу і не дає змоги гнучко реагувати на зміни середовища та вимог до системи [3];

• у наявних програмних системах правила та алгоритми функціювання жорстко зашиті у коді. Будь-яка зміна алгоритму та правил вимагає перекодування та тестування, що займає багато часу та дорого коштує.

Аналіз останніх досліджень та публікацій

Перспективним напрямом розвитку архітектур програмних систем з метою вирішення зазначених вище проблем є створення програм, які функціонують на основі моделей. Головна мета таких підходів – не тільки спростити сам процес створення програм, але й відділити програмну

логіку від платформних залежностей, так, щоб з одного набору моделей-специфікацій можна було згенерувати еквівалентні програмні продукти для багатьох платформ.

Найвідомішим підходом, що реалізує зазначене вище завдання, є підхід MDA (Model Driven Architecture, Model Driven Design). Цей орієнтований на моделі підхід розробляє OMG (Object Management Group) протягом декількох років [4, 5]. Основою цього підходу є методологія побудови програмного забезпечення, в якій на всіх етапах цього процесу використовуються моделі – для покращення розуміння проектних рішень, проектування, побудови, розгортання, експлуатації, підтримки та модифікації. OMG розглядає MDA як подальший розвиток об'єктно-орієнтованого програмування (ООП), в якому за аналогією з ООП використовується базовий принцип "Усе є моделлю". OMG вважає, що застосування MDA під час створення програмних систем виводить на новий рівень компіляції моделей, аналогічно до того, як звичайний компілятор перетворює програму мовою програмування високого рівня на виконувальний код. Використання MDA надає програмним системам необхідної гнучкості, скорочує видатки на їх створення.

MDA визначає специфікацію програмної системи на доменно-залежній мові, так що створюється платформно-незалежна модель (PIM – Platform Independent Model). У наступному кроці з використанням моделі визначення платформи (PDM – platform definition model) PIM перетворюють на платформно-залежну модель (PSM – platform specific model), яку комп'ютер може виконувати.

Однією з найрозвиненіших реалізацій MDA є виконувальний UML (Executable UML, xUML). Основи xUML були закладені в роботах Шлаера (Shlaer) у 80-роках XX століття та у наш час продовжені в роботах Меллора (Mellor) [6,7].

Водночас, практика впровадження вирішень на основі MDA у руслі запропонованих OMG технологій реалізації виявила такі проблеми [8]:

• непостійність, високий темп змін предметної області і, як наслідок, потреба у частій переробці пакета моделей;

• значна складність процесу створення повного комплекту модельних специфікацій для програми, порівнянна з вартістю ручного кодування;

• деякі важливі аспекти реальних систем, як і вимоги до них, важко відобразити у вигляді формальної моделі. Крім того, мови моделювання (UML та її розширення) мають істотні обмеження щодо можливостей подання таких аспектів та обмежень;

• на практиці не завжди можливо відділити програмну логіку від обмежень апаратної платформи.

Невиправдано жорсткі обмеження процесу побудови програмного продукту з використанням MDA привели до появи інших, гнучкіших підходів – таких як "прагматичний MDA" [9].

Зазначені проблеми MDA, які великою мірою пояснюються складністю як предметної галузі, так і відповідних моделей, можна вирішити за допомогою реалізації програмної системи як набору простих інтерпретованих моделей, що взаємодіють.

У роботі [10] запропоновано підхід та розроблено принципи організації програмної системи, яка керується моделями. Для практичної апробації та доведення перспективності та працездатності запропонованого підходу необхідно розв'язати задачу побудови інструментального комплексу для концептуального моделювання програмних систем, який і реалізує цей підхід.

Цілі статті

Метою цієї статті є розроблення архітектури, принципів побудови та функціювання, а також базових структур даних і процедур функціювання для інструментального комплексу моделювання керованих моделями інтелектуальних програмних систем.

Сценарії використання системи моделювання

Розв'язання поставлених задач передбачає аналіз акторів, способів використання, з подальшим визначенням та деталізацією процесів функціювання системи моделювання.

Головними акторами у системі моделювання є:

Експерт предметної галузі. Він створює або модифікує концептуальну модель, валідує її, здійснює нагляд за її використанням.

Програміст – створює, супроводжує, модифікує та тестує інтерпретатори моделей.

Агент моделювання – виконує моделі у середовищі моделювання.

Сценарій використання „Створення моделі”, своєю чергою, поділяється на сценарії „Завдання моделі”, „Оцінка результатів виконання та модифікація існуючої моделі”, „Опрацювання онтології”, „Тестування моделі”.

Сценарій „Виконання та підтримка моделей” поділяється на сценарії „Виконання моделі”, „Підтримка взаємодії моделей”, „Пошук інформації”. Кожен з цих сценаріїв виконується окремим сервісом середовища моделювання.

Зведену інформацію про акторів, сценарії та відповідні інструментальні програмні засоби або сервіси системи моделювання відображено у табл. 1.

Сценарії використання комплексу моделювання

Актор	Сценарій використання	Програмний засіб (сервіс)
Експерт предметної галузі	Створення моделі. Під час створення моделі відбувається її валідація та тестування, задаються правила для автоматизованої верифікації	Редактор моделей
	Оцінка результатів експлуатації моделей. Модифікація моделей.	
	Тестування моделі	
	Опрацювання онтології	
Програміст	Створення, модифікація та тестування інтерпретатора моделей	Інтегроване середовище для програмування
Агент моделювання	Виконання моделей	Інтерпретатор моделей
Середовище моделювання	Організація взаємодії моделей	Брокер взаємодії моделей
	Пошук інформації	Провайдер інформаційних послуг

Подання концептуальних моделей у системі моделювання

Концептуальна модель Md складається зі схеми ScMd та реалізації RlMd:

$$Md = (ScMd, RlMd). \quad (1)$$

Схема є частиною моделі, яку створює людина-фахівець у предметній галузі. Схема специфікує задіяні об’єкти та логіку їх опрацювання, визначаючи спосіб розв’язання певної задачі. Схема описується в термінах предметної області, незалежно від можливих технологій її реалізації. Наявність формалізованої схеми дає змогу не тільки забезпечити її машинну інтерпретацію та реалізацію, але й створює можливість для інших експертів проаналізувати та оцінити цю схему і за необхідності відкоригувати її. Наявність схеми, яку можна змінювати, не змінюючи реалізації, надає системі, побудованій з використанням моделей, необхідної гнучкості – адже у разі зміни середовища досить змінити схему для того, щоб змінити поведінку системи. При цьому стають непотрібними такі працемісткі та тривалі стадії традиційного процесу розроблення програмного забезпечення, як кодування та тестування. Крім того, наявність формалізованої схеми слугує засобом документації знань фахівця – автора моделі щодо способу розв’язання задачі.

З іншого боку, реалізацію моделі доцільно спрощувати, створюючи компоненти багаторазового використання або одну реалізацію для широкого класу моделей. Універсальність та простота реалізації моделі надають системі гнучкості, забезпечуючи незмінність реалізації при зміні схеми моделі.

Головні вимоги до схеми:

1. Схема повинна бути простою. Схему створює та попередньо валідує людина – фахівець, і вона відображає певну ментальну модель цієї людини. Рекомендована кількість концептів у схемі не перевищує 5–8, адже стільки концептів, згідно з дослідженнями психологів, людина може

тримати у фокусі уваги одночасно [11]. Ця вимога безпосередньо випливає з вимоги співпраці інтелектуальної системи з людиною. Крім того, прості, універсальні моделі, що що не мають функціональної надлишковості, просто комбінувати у складніші.

2. Схема створюється з використанням концептів онтології, спільної для всіх моделей. Під час створення схеми перевіряються обмеження, визначені в онтології. Отже, під час створення схеми враховується досвід інших експертів, формалізований в онтології і забезпечується дотримання різними моделями однакових обмежень. Натомість при створенні схеми додаються нові, додаткові обмеження.

Розглянемо детальніше головні складові частини схеми моделі. Деякі з цих частин створює автор, інші – автоматично заповняються інструментальним середовищем проектування моделей.

Схема моделі має такі інформаційні секції:

- **Метадані**. Відомості про модель загалом: автор, час створення, історія модифікації, посилання на онтологію, тип (типи) моделі. Визначення типу моделі дає змогу зарахувати модель до певної класифікаційної категорії залежно від типів задач та способів їх розв'язання і знайти потрібні моделі на основі відомого класу задач.
- **Базова модель**. Визначає посилання на базову схему (дані та обмеження на них), які потрібно заповнити активатору цієї моделі. Також специфікує відображення між слотами моделі-активатора та базової моделі. Містить визначення функції релевантності методу розв'язання задачі.
- **Інформація для ініціалізації**. Визначає обов'язкові та необов'язкові для заповнення слоти моделі, вказує на можливі додаткові джерела для отримання даних.
- **Передумови**. Список умов, які потрібно виконати для активації моделі. Умови перевіряються на основі даних базової моделі та додаткових даних, одержаних з контексту. Якщо передумови не виконуються, то модель не активується, а повертається повідомлення про порушені умови.
- **Тіло моделі**. Формат визначається залежно від типу моделі. Специфікує задіяні об'єкти та логіку розв'язання задачі.
- **Інформація для верифікації.** Секція верифікації використовується при модифікації моделі або при створенні іншої моделі на основі заданої. У цій секції визначено обмеження цілісності моделі, які потрібно дотримати при всіх модифікаціях.
- **Додаткові**. Додаткові секції визначаються залежно від типу моделі. Наприклад, для моделей, які виконують завдання моніторингу, є додаткова секція, що визначає періодичність активації моделі.

Реалізація моделі є програмним компонентом, який опрацьовує схему моделі, ініціалізує її фактами з бази фактів, виконує задані моделлю дії. На відміну від схеми, реалізацію моделі створює фахівець – програміст. Основні вимоги до реалізації пов'язані з потребою забезпечити можливість зміни схеми у широких межах без зміни реалізації. При цьому дозволяється змінювати не тільки параметри схеми або обмеження, але й до певної міри змінювати і структуру схеми.

Вимоги до реалізації:

- Реалізація повинна бути, по можливості, універсальною і працювати для широкого класу типів моделей. При цьому використовується семантична інтерпретація даних моделі.
- Для зменшення складності переналаштування реалізація повинна використовувати компоненти багаторазового використання, основуватися на простих функціональних програмних компонентах, однакових для усіх реалізацій.

Реалізації моделі відповідає програмний компонент – інтерпретатор моделі. Цей компонент отримує специфікацію моделі у вигляді XML-файла, а також ініціалізовану базову модель від брокера взаємодії моделей, активує та опрацьовує активовану модель. Головні функції інтерпретатора моделей такі:

- Ініціалізація моделі. Отримання всіх необхідних фактів з бази фактів.
- Перевірка виконання передумов.
- Опрацювання тіла моделі.
- Підтримка протоколу взаємодії моделей та співпраця з брокером взаємодії моделей.

Запропонуємо деякі рекомендації щодо створення інтерпретатора моделей, дотримання яких істотно спростить процес створення та модифікації інтерпретатора.

- Інтерпретатор моделей доцільно створювати на скриптовій мові програмування (такі як Python, Javascript, VBasic, Perl). Це зменшує час на модифікацію програми та не вимагає високої кваліфікації програміста.
- Оскільки модель-активатор передає активованій моделі тільки мінімальний обсяг даних (у базовій моделі), а решту даних сама модель отримує з контексту, неможливо наперед забезпечити наявність потрібних моделі даних у базі фактів. Тому важливим стає урахування можливості відсутності потрібних даних. Інтерпретатор повинен опрацьовувати ситуацію відсутності необхідних даних в базі знань і мати напоготові рішення "за замовчуванням", або повідомляти про неможливість розв'язання задачі та причину цього.
- Для зменшення складності створення моделі в будові інтерпретатора доцільно використати базові функціональні та логічні компоненти, наприклад, такі, що реалізують операції „Пошук у базі фактів", „Перевірка передумов", „Звернення до сервіса", „Перевірка умови", „Створення або модифікація факту" тощо. Такі функціональні примітиви, які використовують усі інтерпретатори, дають змогу не тільки спростити процедури створення та підтримки інтерпретатора, але й реалізувати єдиний підхід до реалізації типових операцій інтерпретації моделей.
- Один інтерпретатор доцільно сторювати для цілого класу моделей. Він має бути незалежним від платформи виконання та від логіки, поданої у моделі. Водночас в інтерпретаторі відображена „логіка інтерпретації" – логіка опрацювання моделей, що унеможливлює використання єдиного інтерпретатора для всіх класів моделей.
- Дії, які виконує інтерпретатор, пропонується оформити як звертання до сервісів інформаційної системи, з визначеними форматами цих звертань. Отже, власне наявні сервіси визначають увесь спектр можливих дій у системі.

Опрацювання концептуальних моделей системою моделювання

Розглянемо детальніше окремі операції та аспекти опрацювання моделей. Ці операції виконують різні програмні компоненти і на різних стадіях опрацювання. Наприклад, інтерпретатор моделей виконує операції ініціалізації моделі, перевірки передумов та виконання моделей. Редактор моделей реалізує створення моделі, верифікацію на основі онтології, тестування та моделювання. Брокер взаємодії підтримує процес взаємодії моделей. Зосередимо увагу на питаннях взаємодії моделей та розв'язанні складних задач з використанням моделей. Важливими завданнями, що реалізуються на всіх рівнях опрацювання моделей, є верифікація, валідація, забезпечення релевантності та вибір моделі для виконання з декількох альтернатив.

Ініціалізація. Операцію ініціалізації виконує інтерпретатор моделей одразу після її активації та заповнення слотів базової моделі. Метою ініціалізації є отримання усіх необхідних для виконання моделі даних з локальної бази фактів та, можливо, із зовнішніх джерел. Початковими даними для ініціалізації є семантично-інтерпретовні дані базової моделі, зокрема специфікація мети, з якою активізована модель. Заповнення слотів базової моделі, яке відбувається у процесі взаємодії з моделлю-активатором і здійснюється брокером взаємодії моделей, вважатимемо першою стадією ініціалізації. Під час другої стадії ініціалізації відбувається видобування потрібних даних з контексту базової моделі у базі фактів.

У схемі моделі, у секції яка специфікує дані для процесу ініціалізації, визначено прийнятні конфігурації заповнень слотів залежно від мети задачі, що поставлена перед моделлю. У найпростішому випадку визначено обов'язові та необов'язкові для заповнення слоти моделі. В складніших випадках у разі відсутності даних у базі фактів вказують на модель, яку можна використати для пошуку потрібних даних.

Перевірка передумов. Перевірку передумов виконує інтерпретатор моделей після заповнення слотів моделі даними. Метою цієї перевірки є визначити релевантність моделі на основі наявних даних у базі фактів. У процесі перевірки передумов виконується перевірка умов релевантності заданих автором моделі.

Ці умови подаються як набори вимог:

$$Re_{ft} = \bigcup_{i=1}^{n} \bigcap_{j=1}^{m} Asr(M(ValSlMd)) \quad (2)$$

де Asr(M(ValSlMd)) – це твердження, задане на множині значень ініціалізованої моделі, яке набуває значення true або false:

$$Range(Asr(M(ValSlMd))) = \{true, false\}. \quad (3)$$

Модель задовольняє передумови, якщо Re_{ft} = true.

Наприклад, модель, яка керує доступом до будинку фірми, після ідентифікації особи, яка претендує на доступ, визначає позитивні передумови надання доступу для таких випадків: а) претендент є постійним працівником фірми; б) є тимчасовим працівником за контрактом; в) є запрошеною особою, про що є запис у відповідному реєстрі. Якщо жодна з цих передумов не виконується, запит на доступ відхиляється.

Виконання моделей. Реалізується інтерпретатором моделей і розглянуто у розділі, що стосується принципів побудови інтерпретаторів.

Взаємодія моделей. Загальні принципи взаємодії моделей розглянуто у [10]. Зокрема, введено поняття моделі – активатора та активованої моделі, базової моделі, функцій релевантності та вибору моделі, протоколу взаємодії моделей. Розглянемо механізм взаємодії моделей детальніше, стосовно його технічної реалізації (рис. 1).

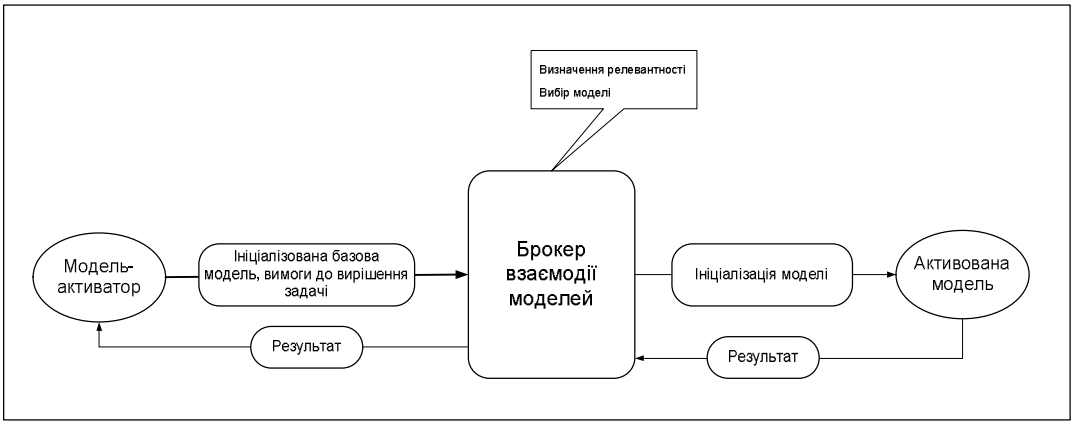

Рис. 1. Процес взаємодії моделей

Ініціатором організації взаємодії моделей виступає активна у поточний момент часу модель, яка, щоб розв'язати свою основну задачу, потребує розв'язання другорядних, допоміжних задач. Наприклад, модель, яка формує туристичну поїздку, на певному етапі свого виконання потребує вирішення завдання придбання білетів на автобус. При цьому вона звернеться до моделі, яка здійснить таку покупку.

З метою зменшення складності реалізації моделі-активатора введемо окремий сервіс системи моделювання – „Брокер взаємодії моделей", який забезпечуватиме реалізацію взаємодії моделей. До функцій брокера взаємодії зарахуємо:

1. Аналіз мети та пошук відповідних моделей.
2. Визначення релевантних моделей.
3. Вибір з множини релевантних моделей однієї моделі та активація її.
4. Ініціалізація активованої моделі.
5. Відслідковування її процесу виконання та повернення результату активатору.
6. Підтримка за необхідності сполучення між активатором та активованою моделлю.

Як видно з опису функцій сервісу, він є доволі складним програмним компонентом, водночас він створюється один раз та обслуговує взаємодію усіх компонент системи моделювання, забазпечуючи тим самим єдиний підхід та правила підтримки взаємодії моделей.

Структурою даних, яка використовується для передавання параметрів між моделлю-активатором та активованою моделлю, є базова модель. Таку модель розглядатимемо як загальну постановку задачі, яка специфікує тип задачі, об'єкти, для яких цю задачу розв'язують, обмеження на процес розв'язання задачі.

Активатор передає брокеру взаємодії структуру даних, у яку внесено відображення слотів активатора у слоти базової моделі та вимоги щодо процесу розв'язання поставленої задачі та очікуваних результатів.

Базова модель фактично є схемою, а не повною моделлю, тобто вона не має реалізації. Натомість базова модель специфікує постановку задачі для всього класу подібних задач. Ієрархія (таксономія) базових моделей відповідає таксономії цілей (задач), які розв'язуються з використанням моделей. Моделі, що розв'язують таку задачу, хоч, можливо, різними методами і для різних ситуацій, мають спільну базову модель. Слоти базової моделі відповідають ролям, які заповнюються значеннями слотів моделі-активатора.

Наприклад, загальна задача здійснення фінансової трансакції має такі слоти у базовій моделі, як *Покупець*, *Продавець*, *Товар*, *Гроші*. Вона є базовою для таких конкретніших моделей здійснення фінансових трансакцій, як „Покупка нерухомості", „Оплата послуги", „Придбання продукту харчування" та інших, які відрізняються способом виконання трансакції.

Вибір тієї чи іншої базової моделі в процесі взаємодії моделей здійснює активатор. Можливі такі варіанти визначення базової моделі:

а) релевантну модель (або набір моделей) вказав явно автор моделі. У цьому випадку базова модель визначається через вже задану модель, оскільки кожна модель має посилання на базову модель, яку вона використовує;

б) задано тільки мету активації, а релевантну базову модель потрібно визначити. У такому разі базова модель і відповідна множина підлеглих моделей визначається за онтологією цілей (задач), яка є частиною загальної онтології системи.

На основі заповненої базової моделі брокер взаємодії визначає множину моделей, які реалізують релевантні методи розв'язання поставленої задачі. Для визначення множини релевантних моделей використовується функція релевантності ReLnMd. Зазначимо, що поняття релевантності у випадку моделей має два аспекти. У першому визначається релевантність методу розв'язання поставленої задачі. В іншому – релевантність на основі поточного стану бази фактів. Релевантність методу розв'язання задачі визначається за базовою моделлю, а релевантність на основі стану бази фактів – у передумовах виконання моделі на основі повністю заповнених слотів цієї моделі. Тому доцільно розбити функцію релевантності на дві – Re_{mt} та Re_{ft} і вважати, що

$$ReLnMd = Re_{mt} \,\&\, Re_{ft} \qquad (4)$$

Такий підхід, що розглядає окремо релевантність методу та релевантність даних, дає змогу зменшити час опрацювання моделей, адже ініціалізація моделі та пошук усіх потрібних даних у контексті цієї моделі потребує довшого часу. Тому попереднє визначення релевантності моделей здійснюють тільки у контексті базової моделі.

Базова модель, як правило, відповідає групі задач, які мають подібну постановку. В контексті ініціалізованої базової моделі переглядають список моделей, що підлягають цій базовій моделі. З кожною підлеглою моделлю асоціюється функція релевантності Re_{mt} яка на основі даних базової моделі визначає релевантність застосування методу розв'язання поставленої задачі, реалізованого у моделі.

Функцію релевантності Re_{mt} визначимо як набір тверджень у контексті базової моделі

$$Re_{mt} = \bigcup_{i=1}^{k} \bigcap_{j=1}^{l} Asr(Con(Md_b)). \qquad (5)$$

Наприклад, якщо потрібно купити квиток на автобус, активізується загальна базова модель торгової трансакції. Покупець визначається як *Турист*, слот *Гроші* ініціалізуються посиланням на банківський рахунок туриста, *Товаром* є послуга перевезення, а *Продавцем* – підприємство-перевізник. На основі заповнених слотів базової моделі, і звертаючись до контексту визначених у

ній фактів, функція релевантності визначає, що у списку моделей торгових трансакцій є одна релевантна модель – „Оплата послуги”, адже товаром є послуга перевезення.

У множині релевантних моделей треба вибрати одну, яку буде використано для розв’язання поставленої задачі. Для вибору моделі використовують функцію вибору Fc, яка максимізує (або мінімізує) певний визначений критерій (такий як час виконання моделі або оцінку точності результату). Метою операції вибору моделі є визначення у непорожній множині релевантних моделей однієї моделі, яка і буде активована для розв’язання поставленої задачі.

Діапазон складності підходів у її реалізації змінюється від просто випадкового вибору будь-якої моделі зі списку релевантних до застосування складних методів оцінки моделей-кандидатів за різноманітними критеріями. Загалом, для розв’язання задачі вибору моделі необхідно створити цілу інфраструктуру вибору, яка використовує свої моделі, таксономії, секції у схемі моделі-активатора та активованої моделі.

Визначимо складові частини цієї інфраструктури.

• Онтологія методів вибору та асоційованих з ними моделей вибору. До неї входить, наприклад, випадковий вибір, методи одно- та багатокритеріального вибору тощо.

• Онтологія критеріїв вибору. Визначає семантику можливих критеріїв вибору моделі. Наприклад, такими критеріями є час (складність) опрацювання моделі, очікувана точність результатів.

• Частина метаданих активатора, яка специфікує переваги та вимоги щодо процесу виконання моделі. Ці дані визначають обмеження та очікування активатора щодо часу виконання або точності результатів. Використання цих даних дасть змогу прийняти рішення щодо вимог до вибору та виконання активованої моделі й буде використано для формування аналогічної секції у метаданих активованої моделі. Наприклад, якщо активатор сам обмежений за часом виконання, то він у виборі активованих моделей надаватиме перевагу швидким моделям.

• Частина метаданих моделі – кандидата містить інформацію щодо характеристик цієї моделі, наприклад, складності, часу виконання та точності. Можуть бути посилання на інші моделі, які дадуть змогу наперед оцінити час виконання або точність результатів.

• Моделі вибору, які формують вимоги та переваги щодо активованої моделі і реалізують сам вибір. Крім того, ці моделі враховують обмеження і на сам процес вибору. Наприклад, в деяких випадках вибір треба зробити якнайшвидше і витрачати час на точну оцінку параметрів, що впливають на вибір недоцільно. Моделі вибору є універсальними та використовуються і в інших ситуаціях, коли необхідно здійснити вибір з декількох альтернатив.

Вибір моделі для активації здійснюється моделлю вибору за ініціативою брокера взаємодії моделей. Порядок взаємодії моделей та брокера визначається протоколом взаємодії. У найпростішому випадку цей протокол містить команди запиту до брокера, запиту до активованої моделі, відповіді брокера та активованої моделі. У складнішому випадку, якщо активована модель визначена як багатофункціональний сервіс, протокол додатково визначає команди та відповіді відповідно до опублікованого моделлю інтерфейсу.

Створення та модифікація моделей

Моделі створює та валідує людина-експерт у певній предметній галузі, вони відображають знання цієї людини щодо способу розв’язання певної задачі. Для створення або модифікації моделі використовують інструментальний засіб – програму “Редактор моделей”.

Головні функції (варіанти використання) цієї програми такі:

• Створення схеми моделі з використанням класів та фактів онтології. Визначення додаткових обмежень. Схему моделі створюють у графічному поданні, а на основі готової схеми генерують XML- файл опису моделі.

• Робота з онтологією. Пошук класів онтології та фактів. Модифікація класів та відношень. Визначення залежностей між певними класами онтології та наявними моделями. Визначення впливу змін в онтології на наявні моделі.

• Робота з базою фактів. Пошук потрібних фактів, додавання або вилучення фактів з бази.

• Робота з репозиторієм моделей. Пошук моделей-шаблонів з використанням різних критеріїв.

• Визначення XML-шаблону моделі та завдання способу генерації XML-опису схеми моделі для визначеного типу моделей.

• Завдання блока верифікації моделі як набору правил та залежностей, який повинна задовольняти модель. Для цього можна використати, наприклад, OCL. Різні набори правил валідації застосовують для створення на основі цієї моделі нової або під час ініціалізації моделі.

• Визначення або налаштування інтерпретатора моделей – компонента, що виконує модель.

• Тестування моделей та мереж (агрегатів) моделей. Запуск та виконання моделей на тестовій базі фактів, перевірка відповідності поведінки моделей очікуваним результатам.

Отже, у функції редактора моделей входять як операції створення схеми, так і створення реалізації моделі. Процес створення моделі відрізняється у разі створення моделі “з нуля”, та її створення на основі існуючої моделі-шаблону.

Розглянемо послідовність етапів створення моделі.

Завдання типу моделі. Створюючи схеми моделі, визначають її тип, користучись онтологією типів моделей. Тип визначають за метою або класом задач, які розв’язує модель. Як правило, моделі, що мають один тип, є взаємозамінними і можуть використовуватися для розв’язання подібних задач. Отже, усі моделі у репозиторії моделей впорядковані відповідно до онтології задач, які вони розв’язують. Тип моделі також визначає і реалізацію – оскільки інтерпретатор моделі доцільно створювати один для цілої групи однотипних моделей. Тип моделі накладає обмеження на допустимі слоти та відношення у моделі.

Якщо модель створюється на основі шаблону, то вона є деталізацією наявної моделі-шаблону і її тип є підтипом цієї моделі – шаблону. Відповідно, така модель успадковує визначення слотів та обмежень з моделі шаблону. При цьому, можливо, до шаблону додаються нові слоти та обмеження.

Для кожного типу моделей визначена базова модель, яка відображає мінімальний набір концептів та обмежень, наявних в усіх моделях цього типу. Базові моделі доступні іншим моделям, які активують моделі цього типу. Наявність базової моделі дає змогу використовувати різні моделі та методи для розв’язання однієї задачі.

Наприклад, для класифікаційних моделей базова модель складається з двох елементів – класифікованого факту (довільний тип) та класифікації (тип – класифікація), поданої як перелік категорій. Механізм побудови відповідності факту певній категорії визначається у конкретних моделях. Моделі нижчого рівня ієрархії визначають спосіб розв’язання задачі, та додаткову інформацію, яку потрібно отримати з контексту.

Визначення релевантності методу розв’язання задачі моделлю. У контексті попередньо визначеної базової моделі специфікують умови релевантності моделі, відповідно до визначення (). Умови релевантності записують у секцію “Базова модель”.

Виявлення варіантів використання моделі. Визначення варіантів використання моделі передбачає ідентифікацію операцій, які виконує модель, якщо модель є багатофункціональною. Наприклад, модель торгової трансакції передбачатиме операції моделювання та оцінки результатів, фактичного виконання трансакції, пошуку товару для трансакції. Як правило, багатофункціональна модель має можливість виконувати декілька операцій з однією ініціалізованою схемою.

Визначення складових частин моделі. Для кожного варіанта використання визначають істотні ролі та типи з онтології, які можуть заміщувати ці ролі. Визначають істотні для моделі зв’язки та обмеження. Загалом, ролі, зв’язки та обмеження є основою для формування схеми моделі. На цьому етапі доцільно встановити відповідність між визначеними слотами моделі та слотами базової моделі згідно з попередньо визначеним типом моделі.

Специфікація механізмів реалізації операцій. Для кожного варіанта використання визначають способи досягнення мети, вхідні дані та результати. Водночас також специфікують обмеження на вхідні дані та результати.

Визначення передумов. Для кожного варіанта використання визначають передумови – як вимоги щодо типів вхідних даних та допустимих діапазонів їхніх значень.

Завдання умов та процедур ініціалізації. Використовуючі отримані на попередніх етапах результати, для кожного варіанта використання аналізують множину вхідних даних, визначаючи конфігурації обов'язкових та необов'язкових даних. Наприклад, у моделі торгової трансакції для варіанта використання *Пошук продавця товару* обов'язковими слотами є *Покупець*, *Гроші*, *Товар*, а *Продавець* – визначається у процесі роботи моделі. Для варіанта використання *Проведення трансакції* обов'язковим є заповнення усіх слотів.

Для зменшення залежності успішності виконання ініціалізації від стану бази фактів, коли це можливо, для відсутніх значень специфікують значення за замовчуванням. Альтернативою є визначення релевантних моделей пошуку даних, із завданням відображення слотів моделі у слоти базової моделі пошуку.

Специфікація умов для верифікації моделі. Як правило, з моделями, що визначають спосіб розв'язання певної задачі під час створення моделі, асоціюються правила верифікаціїі. Умови верифікації використовуються для перевірки цілісності моделі, якщо вона змінюється. Наприклад, якщо визначено модель класифікації з використанням заданої шкали, то правила верифікації перевіряють, чи усі інтервали шкали сумарно перекривають область визначення класифікаційного параметра, і чи немає перекриття інтервалів.

Розроблення або модифікація інтерпретатора моделі. Завершена схема моделі передається програмісту, який створює для виконання моделі новий або визначає один з наявних інтерпретаторів моделей.

Тестування моделі. Завершена модель тестується та валідується для різних станів бази фактів та для різних варіантів використання на предмет досягення мети. Виявлені недоліки є підставою для зміни та перетестування моделі.

Висновок

Запропоновану архітектуру та функції інструментального комплексу для моделювання керованих моделями інтелектуальних систем можна використати для створення керованих моделями інтелектуальних програмних систем, які швидко адаптуються до зміни зовнішнього середовища.

1.Software Bugs Cost U.S. Economy $59.6 Billion Annually, RTI Study Finds [Electronic resource]. – Режим доступу: http://www.nist.gov/director/prog-ofc/report02-3.pdf. 2. Balmelli L. Model-driven systems development/ Balmelli L, Brown D, Cantor M, Mott M. // IBM Systems Journal, vol 45, Number 3,2006. – P.569 – 585. 3. Charvat J. Project management methodologies / Charvat J .-John Whiley&Sons, 2003. – 264 p. 4. MDA Specifications. [Electronic resource]. – Режим доступу: http://www.omg.org/ mda/specs.htm. 5. Pastor O. Model-Driven Architecture in Practice / Oscar Pastor, Juan Carlos Molina. - Springer-Verlag Berlin Heidelberg, 2007. – 302 p. 6. Mellor S. MDA Distilled, Principles of Model Driven Architecture/ Stephen Mellor, Kendall Scott, Axel Uhl, Dirk Weise.- Addison-Wesley Professional, 2004. – 176 p. 7. Mellor J. Executable UML: A foundation for model-driven architecture / Mellor S, Balcer M. – Addison Wesley, 2002. – 416 p. 8. MDA – Nice idea, shame about the... [Electronic resource]. – Режим доступу: http://www.theserverside.com/news/1365166/MDA-Nice-idea-shame-about-the. 9. MDA Is DOA, Partly Thanks To SOA. [Electronic resource]. – Режим доступу: http:// www.forrester.com/rb/ Research/mda_is_doa _partly_thanks_to_soa /q/id/39156/t/2. 10. Буров Є.В. Опрацювання знань у когнітивній інформаційній системі, керованій моделями / Буров Є.В / Східно-Європейський журнал передових технологій, № 6/7. – Харків: Технологічний центр, 2009. – С. 40–49. 11. Психология внимания / Под ред. Ю.Б. Гиппенрейтер и В.Я. Романова. – М.: ЧеРо, 2001. – 858 с.