УДК 681.3.06

ТРАНСФОРМАЦІЙНИЙ ПІДХІД ТИПУ «МОДЕЛЬ-МОДЕЛЬ» ДЛЯ РЕАЛІЗАЦІЇ БАЙЄСІВСЬКИХ МЕХАНІЗМІВ ІНТЕЛЕКТУАЛЬНОГО АНАЛІЗУ ДАНИХ

І.М. Парасюк, С.В. Єршов, О.А. Алєксєєнко

Інститут кібернетики ім. В.М. Глушкова НАН України
03680, Київ-187, проспект Академіка Глушкова, 40,
тел.: (044) 266 6422, e-mail: ivpar1@i.com.ua

Розглядається трансформаційний підхід типу «модель-модель» до створення Байєсівських механізмів інтелектуального аналізу даних заснований на методі шаблонів. Запропоновано формалізацію даного підходу за допомогою логіки предикатів першого порядку з метою його автоматизації. Розроблені узагальнена модель Байєсівського механізму інтелектуального аналізу даних, концептуальні схеми механізму автоматизованої трансформації типу «модель-модель» з використанням логіки предикатів та мови OCL (мова об'єктних обмежень). Наведено приклад трансформації узагальненої моделі в уточнену.

Transformational approach of «model-to-model» type for the Bayesian means of intellectual data analysis creation based on the patterns method is considered. The approach formalization by means of the first-order predicate logic with the purpose of its automation is offered. The generalized model of the Bayesian means of intellectual data analysis, conceptual mechanism of the automated transformation of «model-to-model» type charts using predicate logic and the OCL (Object Constraints Language) are developed. The example of transformation of the generalized model in specified one is given.

Вступ

У даний момент Байєсівські мережі довіри (БМД) [1] є одним з провідних формалізмів для побудови механізмів інтелектуального аналізу даних. У зв'язку з тим, що активне дослідження і використання БМД почалося порівняно недавно, в цьому напрямку існує широке поле для розробки нових ідей, зокрема, нових типів вершин для представлення знань та алгоритмів для розповсюдження ймовірностей. З цим пов'язана така проблема, як реалізація нових алгоритмів розповсюдження, структур даних для збереження інформації щодо вершин. Програмна реалізація вищеописаних абстракцій (даних та алгоритмів) у загальному випадку є досить складною і вимагає значних затрат для налагодження і тестування. З огляду на те, що БМД як засіб подання знань про складну систему приваблює фахівців в області інформаційних технологій, є необхідність у розробці механізму швидкого та якісного відображення математичного опису БМД та алгоритмів у цілісну систему на конкретній мові програмування. Основою такого механізму може бути Model-Driven Architecture (MDA) [2, 3] запропонована Object Management Group (OMG). У цій технології у загальному випадку можна виділити три етапи, показані на рис. 1.

Рис. 1. Етапи застосування технології MDA

На всіх етапах технології MDA моделі визначаються у термінах мови Unified Modeling Language (UML) [4], яка являє собою стандартну мову об'єктно-орієнтованого моделювання.

На першому етапі створюється узагальнена, платформо-незалежна модель яка містить компоненти що реалізують вимоги, спільні для певного класу систем, у даному випадку це механізми інтелектуального аналізу даних засновані на БМД, та компоненти які є точками застосування невідомих або мінливих вимог.

На другому етапі визначаються вимоги до конкретної системи, створюються та вбудовуються шляхом трансформації в узагальнену, платформно-незалежну модель компоненти що їх реалізують. Слід зазначити, що нові вимоги можуть виникати як через зміни у принципах роботи системи, в даному випадку це поява нових методів обчислення розповсюдження ймовірності, а відповідно і нових вершин, так і через особливості платформи обраної для реалізації кінцевої системи. Отже отримані після трансформації моделі можуть бути як платформно-незалежними, так і адаптованими під вимоги конкретної платформи. Важливим є дослідження засобів формалізації цього етапу, з метою його автоматизації.

Останній етап полягає у трансформації уточненої моделі в програмний код. У даний момент є досить широкий набір інструментів, які частково реалізують трансформацію такого типу. Серед них треба відзначити такі як Rational Rose [5], що дозволяє генерувати порожні оболонки класів та методів, мову об'єктних обмежень

OCL [6], яка дозволяє описати певні семантичні властивості об'єктів стосовно обробки даних, Borland ECO [7], який дозволяє автоматично генерувати порожні оболонки класів для об'єктів моделі та обробляти OCL-вирази.

У даній роботі основна увага приділяється першим двом етапам застосування технології MDA. Зроблена спроба формалізації процесу трансформації типу «модель-модель», та запропоновані концептуальні схеми створення інструментарію для автоматизації цього процесу.

1. Побудова узагальненої моделі

Узагальнена модель, включає в себе не лише абстракції які реалізують спільні вимоги до класу систем, що проектується, а й такі, що призначені для майбутнього уточнення вимог до конкретної системи. Тому, для проведення подальшої трансформації в якісні уточнені моделі (моделі другого та нижчих рівней) необхідно дотримуватись низки правил, щодо закладання майбутніх невідомих, або частково відомих абстракцій, виділення вузлів які умовно будуть носіями невідомих вимог. Ці правила не є строго формалізованими і не є обов'язковими до виконання, в найбільш загальному випадку виглядають так:

1. У випадку, якщо відомо, що майбутні уточнені моделі передбачають наявність певної сутності (абстракції), але нічого не відомо про неї, в узагальненій моделі необхідно створити порожній клас. При уточнені (відображенні) його можна легко замінити на однойменний, з необхідними властивостями, у загальному випадку не порушуючи сталої частини моделі.

2. У випадку, якщо відомо, що майбутні уточнені моделі передбачають наявність певної сутності (абстракції) і відома частина інформації про неї яка буде сталою, в узагальненій моделі необхідно створити клас з відповідним набором атрибутів та методів. Таким чином при уточнені (відображенні) буде можливість додавати/замінювати методи та атрибути в існуючого класу не порушуючи сталої частини моделі.

3. Якщо виникає ситуація розглянута у п. 1 або п. 2 і при цьому не є відомим кількість споріднених сутностей та ступінь їх унікальності (різниці між ними) у кінцевих моделях тоді необхідно створити базовий клас який в узагальненій моделі буде носієм загальних відомих вимог. При трансформації буде можливість включити загальні уточнюючи вимоги до базового класу як у п.2 і конкретні для кожної нової сутності шляхом додавання класів-нащадків.

Спираючись на ці правила побудована узагальнена модель механізму для розрахунку розповсюдження ймовірностей у Байєсівських мережах показана на рис. 2 засобами мови UML.

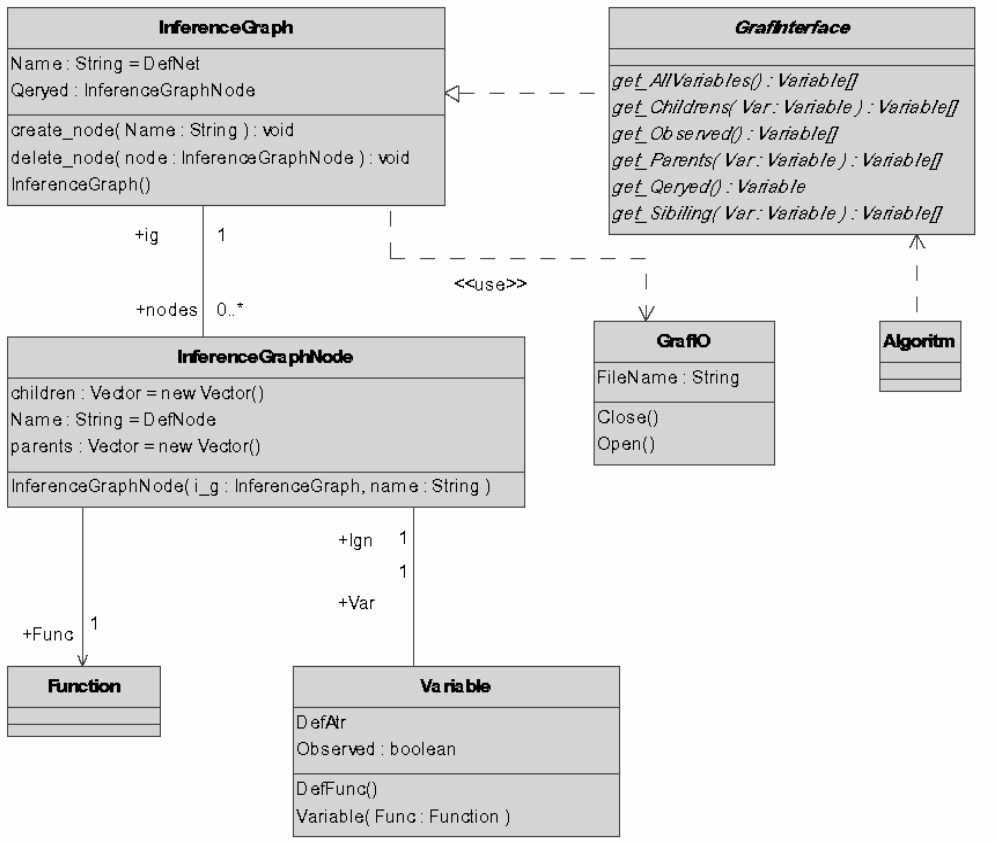

Рис. 2. Узагальнена модель зображена засобами мови UML

У цій моделі як стала (не змінна) частина запропоновані класи, що відповідають за збереження графа в пам'яті, створення/видалення вузлів графа, оперативний доступ до вершин графа. Такими класами є InferenceGraph, InferenceGraphNode, GrafInterface. Решта класів є носіями невідомих вимог до механізму

інтелектуального аналізу даних заснованого на БМД, побудовані у відповідності до вищеописаних рекомендацій, а саме:

- Function – відповідає за представлення, та збереження у пам'яті функції ймовірності певної вершини. Оскільки в різних БМД для означення цієї функції використовуються різні методи (дискретний розподіл, Гаусівський розподіл, дзвоноподібні функції належності [8] і т.д.) при створенні загальної моделі ми не маємо інформації щодо цієї сутності, а отже залишаємо цей клас порожнім, для його подальшої трансформації;
- Variable – відповідає за представлення, та збереження у пам'яті вершини БМД, як математичної сутності. На етапі проектування нам відомі певні вимоги до цієї сутності, наприклад, що стан вершини може бути заданий апріорно, або той факт, що вершина має бути пов'язана з відповідною функцією ймовірності, для більш зручної обробки їх відповідним алгоритмом. Але відкритим є питання як визначити апріорний стан, адже множина станів на етапі створення узагальненої моделі ще не визначена, отже під час трансформації буде необхідним додати до цього класу необхідні атрибути та методи, які відповідатимуть загальній концепції відображення;
- Algoritm – відповідає за реалізацію алгоритмів обчислення розповсюдження ймовірності. По-перше, вже існує досить широке коло таких алгоритмів [1, 8, 9], призначених як для загальних випадків так і для часткових, по-друге, враховуючи той факт що у цій галузі постійно ведуться дослідження, неминуча поява нових алгоритмів обчислення розповсюдження ймовірності. Під час створення узагальненої моделі неможливо врахувати майбутні вимоги до алгоритмів такої системи. Оптимальним рішенням є створення порожнього базового класу, для подальшого наповнення його спільними для обраних алгоритмів атрибутами та методами і реалізації індивідуальних властивостей алгоритмів у класах-нащадках під час трансформації;
- GrafIO – відповідає за збереження інформації про роботу системи на зовнішніх носіях. При створенні узагальненої моделі ми можемо визначити лише певні базові функції, такі як Open() або Close(), адже вони є відомими так як система орієнтована на збереження інформації про свою роботу у файл на жорсткому диску. Проте не відомо які саме дані (структура мережі, результати обчислень) і у яких форматах (текст, XML) необхідно зберігати у файли, або завантажувати з них, отже відповідні класи можуть з'явитися лише під час трансформації коли будуть відомі вимоги до кінцевої системи.

На цьому етапі отримано модель з чітко визначеною сталою частиною, яка не потребує змін при використанні її для проектування певного класу систем (механізмів інтелектуального аналізу даних заснованих на БМД), та закладеними у неї (модель) порожніми або неповними класами, які являють собою точки застосування нових, уточнених вимог.

2. Трансформація моделі

Для трансформації моделі пропонується використовувати метод шаблонів [2], який може бути описаний правилами типу:

якщо "шаблон n" , то замінити на "шаблон m"

Звісно, що при використанні такого підходу важливим є порядок застосування правил, та спосіб навігації за моделлю. Для формалізації шаблонів скористаємось логікою предикатів першого порядку. В якості предикатів визначимо більш прості елементи шаблонів (методи, функції, класи та зв'язки), далі за допомогою логічних зв'язок будемо утворювати з них складні (структури класів) або більш точні. Так для опису заміни порожнього класу однойменним, але наповненим необхідними для реалізації уточнених вимог атрибутами та методами, потрібно визначити такі предикати:

C(c) – ***c є класом***

Ca(c) – ***c – має атрибути***

Cm(c) – ***c – має методи***

На основі цих предикатів складемо необхідні шаблони:

(C(Function) *and* (*not* Ca(Function)) *and* (*not* Cm(Function))) //Function є класом в якому не має атрибутів і немає методів.

(C(Function) *and* Ca(Function) *and* Cm(Function)) //Function є класом в якому є і атрибути, і методи.

Тоді правило відображення виглядатиме так:

якщо (C(Function) *and* (*not* Ca(Function)) *and* (*not* Cm(Function))), ***то замінити на*** (C(Function) *and* Ca(Function) *and* Cm(Function)). (1)

Використовуючи вищеописані предикати та шаблони, виникає проблема необхідності збереження узагальненої моделі, і сутностей (класів, методів) для підстановки в одному просторі (діаграма класів, файл і т.д.), а відповідно і необхідність в унікальності імен елементів моделі та шаблонів. Це пов'язано з тим що немає механізму означення простору в якому буде проводитися пошук шаблонів. Простим рішенням є введення строгого постулату: шаблон із лівої частини правила слід шукати у просторі шаблонів, а шаблон із правої

частини у просторі моделі. Такий метод має певні недоліки, наприклад не можливість формування шаблонів динамічно, під час трансформації. Пропонується більш гнучке вирішення цієї проблеми, а саме виділення належності сутностей до певного джерела (простір моделі або простір шаблонів) у окремий предикат:

InM(c) – ***с належить простору моделі.***

Враховуючи можливість використання предикату InM, можемо переписати попередні шаблони з більшим уточненням. Відповідне правило матиме наступний вигляд:

якщо (C(Function) *and* (*not* Ca(Function)) *and* (*not* Cm(Function)) *and* InM(c)), ***то замінити на*** (C(Function) *and* Ca(Function) *and* Cm(Function) *and* (*not* InM(c))). (2)

У реальних моделях є необхідність у більш складних трансформаціях ніж заміна одного класу на інший, відповідно необхідні більш складні шаблони для визначення відповідних правил перетворення. Розглянемо декілька таких прикладів.

У вищенаведеній моделі закладений клас Algoritm, який являє собою точку застосування нових вимог до алгоритмів у майбутніх системах. Припустимо, що кінцева система використовує два різних алгоритми, реалізованих у класах Algoritm1 та Algoritm2. Для побудови правила трансформації визначимо такі предикати та шаблони:

G(c1, c2) – ***с1 нащадок с2***

(C(Algoritm) *and* InM(Algoritm)) //клас Algoritm, який знаходиться в узагальненій моделі.

(C(Algoritm) *and* C(Algoritm1) *and* G(Algoritm1, Algoritm) *and* C(Algoritm2) *and* G(Algoritm2, Algoritm)) //клас Algoritm з двома класами нащадками: Algoritm1 та Algoritm2.

Правило трансформації виглядатиме так:

якщо (C(Algoritm) *and* InM(Algoritm)), ***то замінити на*** (C(Algoritm) *and* C(Algoritm1) *and* G(Algoritm1, Algoritm) *and* C(Algoritm2) *and* G(Algoritm2, Algoritm)). (3)

Часто трапляються випадки коли у структурах, подібних до вищеописаної, необхідно залишити базовий клас без змін. У такому разі трансформацію необхідно здійснювати в два етапи: з початку динамічно сформувати шаблон, а потім зробити заміну за правилом 3.

Описану методику можна використовувати для трансформації на рівні атрибутів та методів, за умови визначення відповідних предикатів та умовних порожнього методу та порожнього атрибуту.

Спробуємо трансформувати вище подану узагальнену модель в уточнену модель механізму, що буде відповідати таким вимогам:

- у БМД використовуються дискретні вершини, з кінцевим числом станів;
- алгоритм заснований на формулі спільного розподілу ймовірностей мережі для загальних топологій;
- алгоритм заснований на векторному способі обчислення ймовірностей, для відповідних часткових випадків;
- завантаження і збереження БМД у файл формату XML або текстовий файл.

Для реалізації цих вимог у полі шаблонів створимо відповідні класи:

- уточнений Function;
- уточнений Algoritm та його нащадки ProbabilityDistribution і Vectorial;
- порожній GrafIO з уточнюючими нащадками XMLIO і TextIO;
- контейнер для збереження атрибутів та методів уточнюючих клас Variable – VariableAddAtrFun.

Для побудови системи правил трансформації необхідні наступні предикати:

C(c) – **с є класом;**
Ca(c) – **с – має атрибути;**
Cm(c) – **с – має методи;**
AiC(a, c) – **а атрибут класу с;**
MiC(m, c) – **m метод класу с;**
InM(c) – **с належить простору моделі;**
G(c1, c2) – **с1 нащадок с2.**

Система правил трансформації:

(C(Function) *and* InM(Function)) = (C(Function) *and* (*not* InM(Function))). (4)

Відповідно до правила 4 необхідно знайти клас Function який належить простору моделі і замінити його на клас Function який не належить простору шаблонів. На рис. 3 правило представлено мовою UML.

(C(Algoritm) *and* InM(Algoritm)) = (C(Algoritm) *and* (*not* InM(Algoritm)) *and* C(ProbabilityDistribution) *and* G(ProbabilityDistribution, Algoritm) *and* C(Vectorial) *and* G(Vectorial, Algoritm)). (5)

Згідно до правила 5 необхідно знайти клас Algoritm який належить моделі, та замінити його на класи Algoritm, ProbabilityDistribution, Vectorial за умови, що ProbabilityDistribution і Vectorial є нащадками класу Algoritm. На рис. 4 показано дане правило засобами UML.

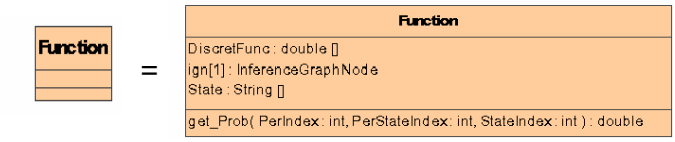

Рис. 3. Правило трансформації 4 зображене мовою UML

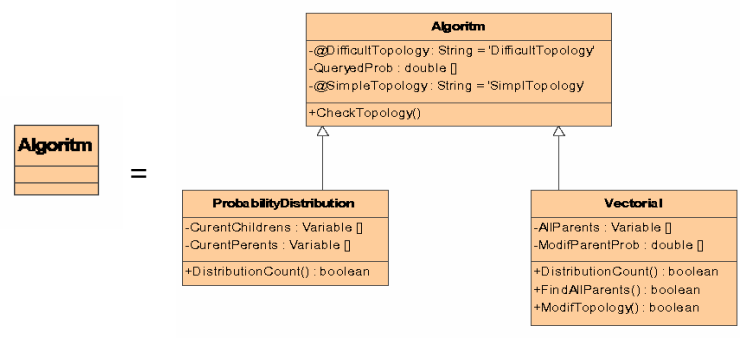

Рис. 4. Правило трансформації 5 зображене мовою UML

(C(GrafIO) *and* (*not* InM(GrafIO) = (C(GrafIO) *and* InM(GrafIO)). (6)

Правило 6 є проміжним і застосовується для динамічного формування шаблону. Відповідно до цього правила необхідно замінити клас GrafIO який належить простору шаблонів, на клас GrafIO який належить простору моделі. На рис. 5 правило представлено мовою UML. Після його застосування у просторі шаблонів з'являється структура яка складається з класу GrafIO, еквівалентного однойменному класу в просторі моделі, та його нащадків TextIO і XMLIO, які були присутні у просторі шаблонів до застосування правила 6.

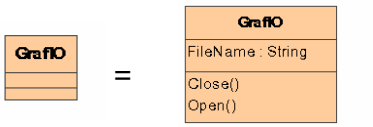

Рис. 5. Правило трансформації 6 зображене мовою UML

(C(GrafIO) *and* InM(GrafIO)) = (C(GrafIO) *and* (*not* InM(GrafIO)) *and* C(TextIO) *and* G(TextIO, GrafIO) *and* C(XMLIO) *and* G(XMLIO, GrafIO)). (7)

Правило 7 описує заміну класу GrafIO з простору моделі, де він не має нащадків і реалізує лише загальні вимоги, на структуру отриману в просторі шаблонів після застосування правила 6. Тут слід зазначити, що правила трансформації 6–7 фактично реалізують механізм додавання до узагальненої моделі класів TextIO та XMLIO, за допомогою динамічного створення проміжного шаблону. Запропоновану схему зручно використовувати у випадках, коли під час проектування узагальненої моделі не було можливості закласти необхідні порожні класи, і під час трансформації є можливість лише додати класи для реалізації уточнених вимог. Особливу увагу при використанні подібних схем треба приділяти порядку застосування правил. На рис. 6 показано дане правило засобами UML.

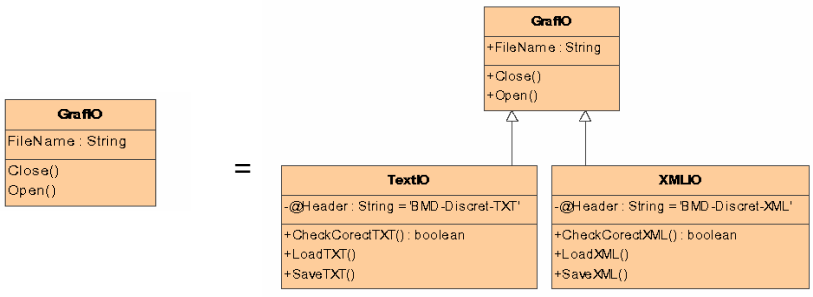

Рис. 6. Правило трансформації 7 зображене мовою UML

(AiC(DefAtr, Variable) *and* InM(Variable)) = *Для Всіх* a(AiC(a, VariableAddAtrFun)). (8)

Правило 8 описує заміну атрибуту DefAtr, введеного в модель, як порожній атрибут з метою подальшої трансформації, на всі атрибути класу VariableAddAtrFun (рис. 7).

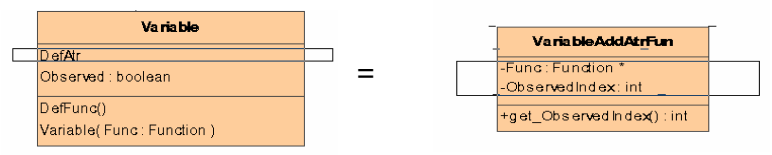

Рис. 7. Правило трансформації 8 зображене мовою UML

(MiC(DefFunc(), Variable) *and* InM(Variable)) = *Для Всіх* m(AiC(m, VariableAddAtrFun)). (9)

Правило 9 описує заміну функції DefFunc(), введеної в модель, як порожня функція з метою подальшої трансформації, на всі функції класу VariableAddAtrFun, який використовується як сховище для функцій і атрибутів реалізуючих уточнені вимоги до класу Variable у просторі шаблонів (рис. 8).

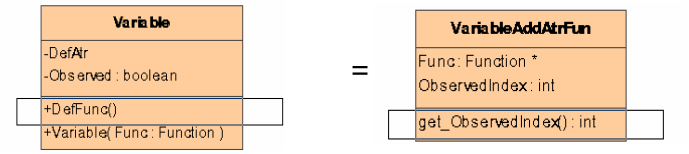

Рис. 8. Правило трансформації 9 зображене мовою UML

Після застосування усіх правил уточнена модель механізму інтелектуального аналізу даних заснованого на БМД має вигляд зображений засобами мови UML на рис. 9, та відповідатиме уточненим вимогам висунутим перед етапом трансформації.

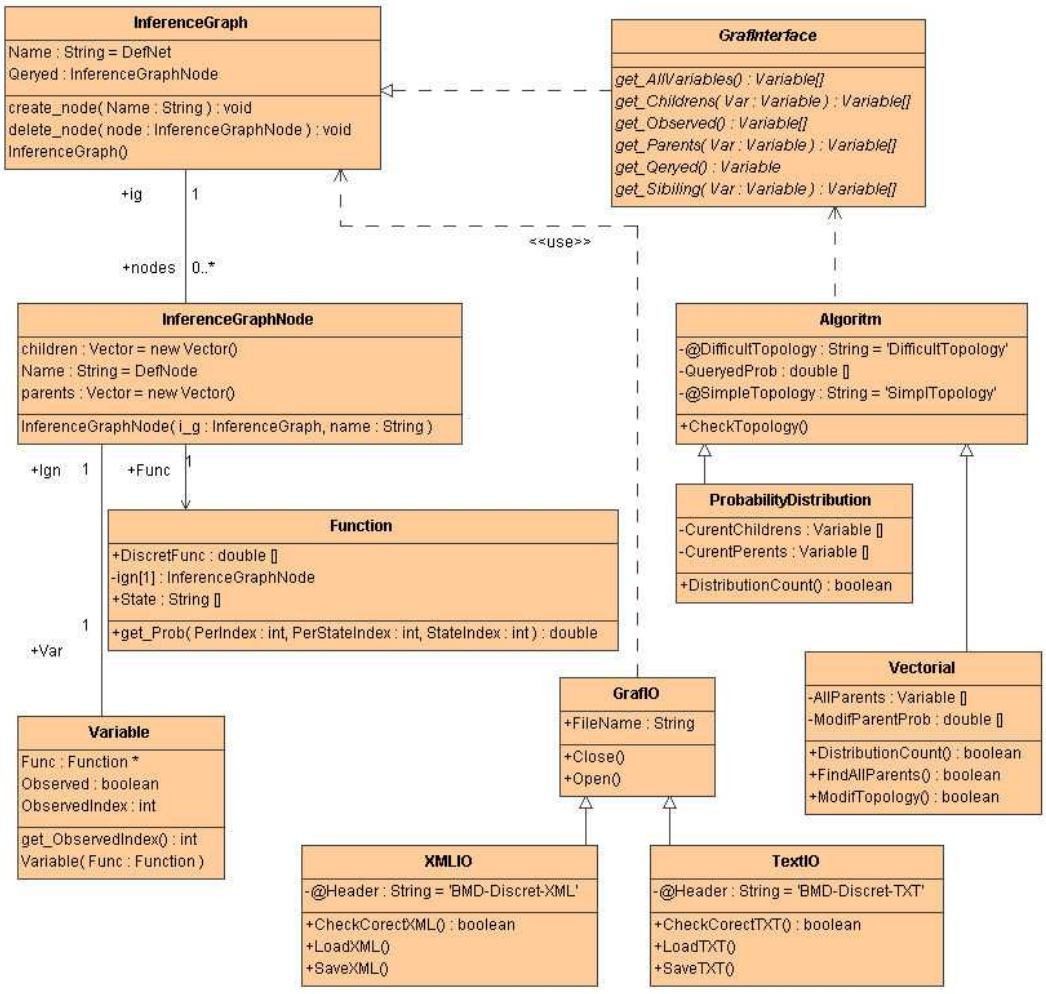

Рис. 9. Уточнена модель зображена мовою UML

Логіка предикатів першого порядку є досить сильною базою для опису правил трансформації. Недоліком використання цього способу формалізації є відсутність готових програмних компонентів для обробки таких виразів. Отже для автоматизації процесу трансформації необхідна розробка модуля на вхід якого буде подаватися модель, перелік шаблонів, предикатів та правила трансформації, на виході відповідно уточнена модель. У якості моделі та переліку шаблонів можуть виступати файли проектів будь якого редактора UML у якому є можливість збереження діаграм у форматі XML [10], прикладом такого редактора є Magic Draw UML 10.5 [11]. Перелік предикатів може бути визначеним у окремому файлі за допомогою тегів XML, які використовуються для збереження моделей редактором. При такому підході процес трансформації у загальному вигляді зводиться до аналізу файлу моделі, порівнянні певних його фрагментів з лівими частинами правил трансформації та при відповідності заміні цих фрагментів на праві частини правил трансформації. Концептуальна схема роботи такого інструменту показана на рис. 10.

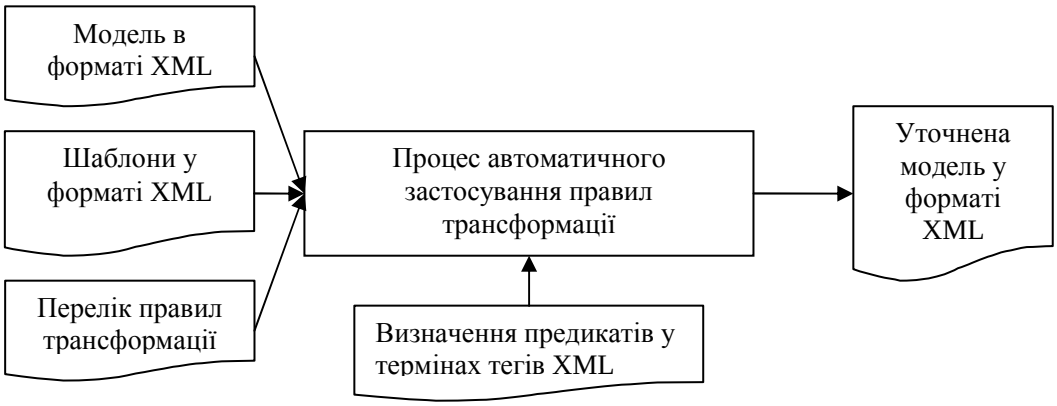

Рис. 10. Концептуальна схема механізму автоматизованої трансформації типу «модель-модель» з використанням логіки предикатів

Частково проблема відсутності програмного забезпечення для обробки виразів логіки предикатів вирішується шляхом використання мови OCL (мова об'єктних обмежень), яка фактично є надбудовою над логікою предикатів, для визначення правих і лівих частин правил трансформації. Перевага використання OCL полягає у тому що, по-перше, існують готові транслятори OCL виразів, по-друге OCL є стандартизованою та загальновідомою мовою для опису маніпуляцій з даними в об'єктному просторі. Недолік такого підходу полягає в тому, що є необхідність побудови мета моделі, яка включатиме всі абстракції концепції об'єктно орієнтовного програмування (такі поняття як клас, метод, атрибут та ін.). До того ж необхідно забезпечити кореляцію (зв'язок) абстракцій об'єктно орієнтовного програмування закладених у мета модель із засобами їх представлення у файлах які містять узагальнену модель та шаблони, за умови використання MagicDrow UML 10.5 це відповідні XML тегі. Концептуальна схема механізму автоматизованої трансформації типу «модель-модель» з використанням мови OCL показана на рис. 11.

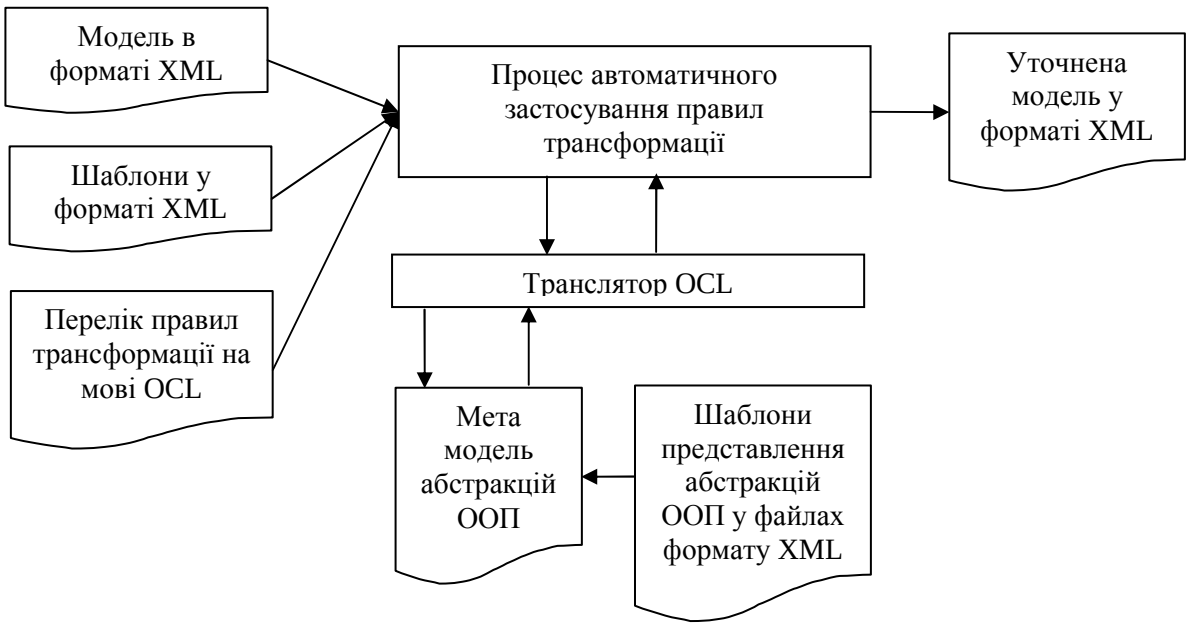

Рис. 11. Концептуальна схема механізму автоматизованої трансформації типу «модель-модель» з використанням мови OCL

Висновки

У роботі розглянуте використання моделе-орієнтовного підходу до створення Байесівських механізмів інтелектуального аналізу даних. Визначені вимоги до побудови узагальненої моделі систем такого класу. Запропонована узагальнена модель Байєсівського механізму інтелектуального аналізу даних, яка відповідає спільним вимогам до механізмів цього класу. Досліджений етап трансформації типу «модель-модель» , при розробці механізмів інтелектуального аналізу даних, та запропонований метод формалізації цього етапу для подальшої автоматизації. Здійснена трансформація висунутої узагальненої моделі в уточнену, з використанням описаного механізму формалізації. Запропоновані концептуальні схеми інструментів для автоматичної трансформації типу «модель-модель» за допомогою логіки предикатів та мови об'єктних обмежень OCL.

1. *Pearls J.* Bayesian inference methods // Encyclopedia of Artificial Intelligence, Sec. Ed. - New York: Wiley-Interscience Publication, 1992. – Vol. 1, A. - P. 89–98.
2. *Сергієнко І.В., Парасюк І.М., Ершов С.В.* Нечіткий трансформаційний підхід до розробки програмних систем // Проблемы программирования. – 2004. – № 2-3. – С. 122–132.
3. *OMG*/MDA Guide V1.0.1. OMG, Document 2003-06-01, 12th June 2003. – http://www.omg.org
4. *OMG*/UML Superstructure Specification, v2.0, Document -- formal/05-07-04, August 2005 – http://www.omg.org
5. *Rational* Software Architect V6.0 - http://www-128.ibm.com/developerworks/downloads/r/rswa/?S_TACT=105AGX14&S_CMP=DWNL
6. *OMG*/OCL 2.0 Specification, Document -- ptc/05-06-06, June 2005 - http://www.omg.org
7. *Borland*® Enterprise Core Objects II (ECO™ II) for .NET - http://www.borland.com
8. *Верьовка О.В., Парасюк І.М., Карпінка Є.С.* Концептуальні основи Байєсівської діагностики у розмитому інформаційному просторі при дзвоноподібних функціях належності // Проблемы программирования. – 2004. – № 2-3. – С. 328–333.
9. *F. G. Cozman.* Credal networks, Artificial Intelligence. 2000. - vol. 120. - P. - 199–233.
10. W3C/Extensible Markup Language (XML) - http://www.w3.org/XML/
11. *Magic* Draw UML 10.5 - http://www.magicdraw.com/