Нечіткий трансформаційний підхід до розробки програмних систем

І.В.Сергієнко, І.М.Парасюк, С.В.Єршов

Інститут кібернетики ім.В.М.Глушкова НАН України, 03680, Київ-187, просп. Акад. Глушкова, 40,
тел.2666422, e-mail: ivpar1@i.com.ua

У роботі дається огляд основних категорій трансформаційних підходів та формалізується процес розробки програмного забезпечення на основі нечітких графових моделей. Модель програмної системи представляється у вигляді нечіткого графу, вершинами якого є сутності та об'єкти. Процес розробки програмного забезпечення складається у виконанні системи трансформацій над нечіткими графами. Продемонстровано, що на основі нечіткої моделі більш адекватно, ніж на основі двозначної логіки, можна формалізувати розуміння правил розробниками програмного забезпечення завдяки можливості описати лінгвістичні вирази, що зустрічаються в реальних проектах.

In paper the overview of main categories of the transformational approaches is given and the process of software development on the base of fuzzy graph models is formalized. A model of the software system is represented as the fuzzy graph, which nodes are the entities and objects. A process of software development consists in implementation of a system of transformations on the fuzzy graphs. It is shown, that on the base of fuzzy model more adequate, than on the base of two-valued logic, it is possible to formalize perception of the rules by the software developers due to a possibility to describe linguistic expressions, which meet in the real projects.

Вступ

Незважаючи на інтенсивний розвиток нечітких інформаційних технологій [1], процес розробки з використанням нечіткої логіки не є строго формалізованим у застосуванні до процесу проектування програмного забезпечення (ПЗ). Методології, що використовують двозначну логіку мають ряд невирішених проблем. По-перше, двозначна логіка, яка звичайно використовується для вираження методологічних правил, не здатна описати цілком як інтуїтивне представлення розроблювачів методів, так і розуміння правил розроблювачами ПЗ. По-друге, сучасні методи класифікують об'єкти розробки лише на основі окремих суджень. По-третє, двозначна логіка явно не моделює контекстуальні фактори, що можуть впливати на придатність методологічних правил. По-четверте, сучасні методи не забезпечують засобів для представлення всієї множини альтернатив проектування і для виміру якості кожної альтернативи в процесі розробки.

Формалізація моделей програмних систем [2,3] – важливий крок на шляху до перевірки правильності таких систем, побудови процесу розробки програмних систем як системи трансформацій між моделями, в кінці якого знаходиться програмний код та документація.

Однією із головних тенденцій у програмній інженерії є дослідження у галузі архітектури програмного забезпечення, яка створена на основі моделей, так званої Model-Driven Architecture (MDA)[4], запропонованої Object Management Group (OMG). У відповідності до підходу MDA розробка програмного забезпечення включає етап моделювання, на якому будуються незалежні від платформи моделі, які потім автоматично перетворюються у платформно-залежні моделі. Такі моделі містять деталі програмної архітектури, яка отримується шляхом трансформації моделей із подальшим автоматичним породженням коду програмної системи мовою програмування, яка обрана для реалізації.

Обидві моделі (незалежна та платформно-залежна) визначаються у термінах мови Unified Modeling Language (UML) [5], яка являє собою стандартну мову об'єктно-орієнтованого моделювання [6].

На сьогодні розроблені різні види трансформацій моделей програмних систем, що відіграють важливу роль у архітектурі MDA. Однак, зазначені підходи не враховують вищезазначені проблеми специфікацій засобами чіткої логіки. Як відомо, створення сучасних моделей програмного забезпечення спирається на графові моделі [3]. Саме тому у даній роботі основна увага приділяється формалізації нечітких графових моделей, які можуть бути основою процесу розробки програмного забезпечення, як системи трансформацій, що виконуються над такими моделями.

Основні категорії трансформаційних підходів

На верхньому рівні розрізняють трансформаційні підходи модель-код та модель-модель. Взагалі, трансформації моделей у код можна розглядати як спеціальний випадок перетворень модель-модель. Необхідно тільки задати метамодель для цільової мови програмування. Однак, для практичних задач повторного використання існуючих технологій компіляції, часто код генерується у вигляді тексту, який потім подається на вхід компілятора. Тому розрізняють трансформації модель-код (які краще описуються як модель-текст оскільки можуть генеруватися артефакти, що не є програмним кодом, такі як код XML) та модель-модель. Деякі інструментальні засоби забезпечують обидва види трансформацій (наприклад, Jamda[7], XDE[8] та OptimalJ[9]).

У категорії модель-код виділяють підходи, які основані на механізмах відвідувача та шаблонах відповідно. У категорії модель-модель можна виділити підходи: на основі прямої маніпуляції, реляційні, на основі трансформації графів, на основі визначення структури та гібридні.

Підходи модель-код. Підходи, які основані на механізмі "відвідувача". Основний підхід для генерації коду складається у забезпеченні певного механізму "відвідувача" для обробки внутрішнього представлення моделі та для запису коду у текстовий потік. Прикладом такого підходу є Jamda[7], який являє собою об'єктно-

орієнтовану структуру, що включає множину класів для представлення моделей, що записані мовою UML, прикладного інтерфейсу програміста для перетворень над моделями та механізму "відвідувача" (так званого CodeWriters) для генерації коду. Jamda не підтримує стандарт MOF [10] для визначення нових метамоделей; однак нові типи елементів моделі можна ввести шляхом утворення підкласів для існуючих Java-класів, які представляють собою заздалегідь визначені типи елементів моделі.

Підходи, які основані на шаблонах. Більшість наявних інструментальних засобів MDA підтримують генерацію модель-код, яка основана на шаблонах, наприклад, b+m Generator Framework[11], JET[12], FUUT-je[13], Codagen Architect[14], AndroMDA[15], ArcStyler[16], OptimalJ та XDE (два останніх також підтримують трансформації модель-модель). AndroMDA використовує існуючі технології генерації, які основані на шаблонах Velocity [17] та XDoclet [18].

Шаблон найчастіше включає цільовий текст, що містить фрагменти метакоду для доступу до інформації із її джерела та для вибору конкретного коду і ітеративного розширення [19]. ЛЧП (ліва частина правила трансформації) використовує логіку виконання для доступу до вихідної моделі; ПЧП (права частина правила) поєднує не типізовані строкові шаблони із логікою виконання для вибору коду та ітеративного розширення, при цьому немає синтаксичного розділення між ЛЧП та ПЧП. Шаблонні підходи найчастіше покладають управління порядком використання правил на користувача у внутрішній формі, шляхом виклику одного шаблону із іншого.

Логіка доступу до вихідної моделі у ЛЧП може мати різні форми. Вона може бути або просто Java-кодом, який викликає прикладний інтерфейс, що наданий внутрішнім представленням вихідної моделі, наприклад, JMI[20], або представлена декларативними запитами, наприклад, мовою OCL[5,21] або XPath [22]. b+m Generator Framework підтримує ідею відокремлення більш складної логіки доступу до вихідної моделі від шаблонів шляхом переміщення її в операції елементів вихідної моделі, які визначає користувач.

У порівнянні до трансформацій, які основані на механізмах "відвідувача", структура шаблону більш подібна на код, який має бути генерований. Використання шаблонів приводить до ітеративної розробки оскільки вони можуть бути отримані із прикладів. Так як підхід на основі шаблонів оперує над текстами, зразки для порівняння, що використовуються, є не типізованими, вони можуть представляти синтаксично та семантично некоректні фрагменти коду. З іншого боку, текстові шаблони не залежать від цільової мови та спрощують генерацію будь-яких текстових артефактів, в тому числі документації.

Подібна технологія – це обробка фреймів. Ця технологія розширює шаблони більш складними механізмами адаптації та структуризації (XVCL [23], FPL [24], XFramer [25], ANGIE [26]). Зазначимо, що FPL, Xframer та ANGIE застосовуються для генерації коду із моделей.

Підходи модель-модель. Трансформації модель-модель переводять початкові моделі у кінцеві, які можуть бути екземплярами тієї самої або інших метамоделей і забезпечують синтаксичну типізацію змінних та шаблонів.

Більшість існуючих інструментальних засобів MDA забезпечує тільки трансформації модель-код. Вони використовуються для генерації моделей, що залежать від платформи (в цьому разі це просто реалізація коду мовою програмування) на основі платформно-незалежних моделей. Оскільки існує велика різниця в рівні абстракції між платформно-залежними та платформно-незалежними моделями, набагато легше генерувати проміжні моделі замість того щоб безпосередньо перейти до цільової моделі. Наприклад, при переході від діаграми класів до реалізації в середовищі EJB, інструментальні засоби такі як OptimalJ генерують проміжну модель компонентів EJB, які містить всю необхідну інформацію для вироблення із неї реального Java-коду. Це робить трансформації більш модульними та такими, що піддаються супроводженню. Також, проміжні моделі можуть знадобитися для оптимізації та настроювання або по меншій мірі для налагодження програмної системи. Крім трансформацій із платформно-незалежних у платформно-залежні моделі, трансформації модель-модель є корисними для отримання різних поглядів на моделі системи та синхронізації між ними.

Підходи на основі прямої маніпуляції. Такі підходи направлені на внутрішнє представлення моделі та певний інтерфейс прикладного програмування для його перетворення. Найчастіше вони реалізуються як об'єктно-орієнтовані середовища, які також мають деяку мінімальну інфраструктуру для організації трансформацій (наприклад, абстрактний клас для трансформацій). Однак, користувачі при цьому мають реалізувати правила перетворення та порядок їх застосування із самого початку з використанням мови програмування такої як Java. Прикладами такого підходу є Jamda та реалізація трансформацій безпосередньо через певний інтерфейс програмування, що сумісний з MOF (наприклад, JMI).

Реляційні підходи. Ця категорія поєднує декларативні підходи у яких головною концепцією є математичне відношення (наприклад, [27], декларативні підходи в [28] та правила відображення). Основною ідеєю тут є установлення початкових та цільових типів елементів відношення та його специфікація з використанням обмежень. У чистому вигляді, така специфікація не виконується [27]. Однак, для декларативних обмежень може бути задана семантика виконання, така як у логічному програмуванні. До речі, логічне програмування із співпоставленням, пошуком із поверненням на основі уніфікації є природним вибором для реалізації реляційного підходу, у якому предикати використовуються для опису відношень. У [28] досліджується застосування логічного програмування (у конкретному випадку Mercury, типізованого діалекту Prolog та F-логіки, парадигми об'єктно-орієнтованої логіки) для реалізації трансформацій.

Усі реляційні підходи є вільними від побічних ефектів. Часто вони підтримують пошук із поверненням [28] та, на відміну від імперативних підходів на основі прямої маніпуляції, створюють цільові елементи неявно

[28]. Реляційні специфікації [27] можуть бути інтерпретовані в обидві сторони. Підходи на основі логічного програмування також природно підтримують двонаправленість відношень, проте деякі підходи при цьому фіксують напрямок трансформацій, що виконуються. Підходи на основі логічного програмування [28] вимагають суворого розмежування початкової та цільової моделей, тобто вони не дозволяють заміну моделі "на місці".

Підходи на основі трансформації графів. Дана категорія підходів до трансформації моделей ґрунтується на теоретичних роботах з трансформаціях графів. В конкретному випадку, ці підходи працюють на типізованих атрибутних помічених графах [29], які є різновидом графів, спеціально розроблених для представлення моделей, що задаються мовою UML. Прикладами підходів до трансформації моделей на основі трансформації графів є VIATRA[30], ATOM[31], GreAT[32], UMLX[33] та BOTL[34,35].

Правила трансформації графів складаються із зразків графів у ЛЧП та у ПЧП. Зразки графів можуть бути перекладені у конкретний синтаксис відповідної початкової або цільової мови (наприклад, у VIATRA) або в абстрактний синтаксис MOF (наприклад, в BOTL). Перевагою конкретного синтаксису є те що він більше ніж абстрактний синтаксис знайомий розробникам, які працюють із заданою мовою моделювання. Також, для складної мови такої як UML, зразки у конкретному синтаксисі є більш стислими ніж зразки у відповідному абстрактному синтаксисі (див приклади у [35]). З іншого боку, легше забезпечити переклад по умовчанню для абстрактного синтаксису, який працює із будь-якою мета моделлю. Це корисно коли не задано спеціалізованого абстрактного синтаксису.

Зразок ЛЧП порівнюється із моделлю, яка підлягає трансформації та замінюється зразком ПЧП. Часто ЛЧП містить умови на додачу до зразка у ЛЧП. Необхідне використання певної додаткової логіки (наприклад, над рядками та числовими доменами) для обчислення цільових значень атрибутів (таких як імена елементів). GreAT пропонує розширену форму зразків із визначенням множинності дуг та вершин. У більшості підходів, порядок виконання правил має зовнішню форму. Механізми керування порядком включають не детермінований вибір, явну умову та ітерації (включаючи ітерації нерухомої точки). Ітерації нерухомої точки використовуються для визначення транзитивних замикань.

Як і реляційний підхід, підхід на основі трансформації графів дозволяє виражати перетворення моделі у декларативний спосіб. Однак, введення спеціальних засобів таких як графові зразки та порівняння із графовим зразком відрізняє даний підхід від реляційного.

Підходи на основі визначення структури Підходи даної категорії мають дві чітко виражених фази: перша фаза стосується створення ієрархічної структури цільової моделі, в той час як друга фаза установлює атрибути та посилання у цільовій моделі. Загальне середовище визначає порядок виконання правил та стратегію їх застосування; користувачі мають справу тільки із завданням трансформаційних правил.

Прикладом підходу на основі визначення структури є трансформаційне середовище модель-модель, що надається OptimalJ. Середовище, реалізоване мовою Java, забезпечує так звані інкрементальні копіювальники (copiers), для яких користувачі мають визначити свої підкласи щоб задати власні правила трансформації. Основною метафорою такого середовища є ідея копіювання елементів моделі із початкової у цільову модель, яка потім може бути адаптована для отримання бажаного ефекту перетворення. Трансформаційне правило реалізується як метод із вхідними параметрами, тип яких визначає тип початкових елементів правила, та методу, який повертає Java-об'єкт, що належить класу, який представляє цільовий елемент моделі. Правила не повинні мати побічні ефекти, порядок їх виконання повністю задається середовищем.

Гібридні підходи. Гібридні підходи поєднують різні механізми із вищезазначених категорій.

Мова Atlas Transformation Language (ATL) [36] є гібридним підходом. Трансформаційні правила ATL можуть бути повністю декларативними, гібридними або повністю імперативними. ЛЧП повністю декларативного правила (так званий початковий зразок) складається із множини типізованих змінних разом із необов'язковим обмеженням мовою OCL у якості фільтра або порядку навігації. ПЧП повністю декларативного правила (так званий цільовий зразок) містить множину змінних та деякі декларативні вирази для зв'язування значень атрибутів у цільових елементах. У гібридних правилах початковий та/або цільовий зразки доповнюються блоком імперативних виразів, який виконується після застосування цільового зразка. Повністю імперативне правило (процедура) має назву, множину формальних параметрів та блок імперативних дій, але не містить зразків. Правила є двонаправленими, підтримується їх наслідування. ATL суворо розділяє початкову та цільову моделі; однак, трансформація із заміною моделі може бути промодельована шляхом механізму автоматичного копіювання. ATL надає можливість як явного так і неявного керування порядком виконання правил.

XDE є хорошим прикладом гібридного підходу. XDE підтримує трансформацію модель-модель через свій механізм шаблонів. Оригінальною метою для введення шаблонів в XDE було забезпечення автоматичного застосування шаблонів проектування [37]. Через певний час, основною концепцією механізму шаблонів XDE стала параметризована кооперація, яка є засобом мови UML для моделювання шаблонів проектування. Наступною метою стали узагальнені трансформації модель-модель, основний механізм шаблонів перетворився у гібридний і дуже складний підхід. Шаблон представляється як пакет, що містить параметри зовану кооперацію та певну кількість інших моделей, які можуть бути автоматично пристосовані і скопійовані та/або вставлені у цільову модель з використанням імперативних викликів мовою Java. При застосування шаблону, параметри можуть бути інтерактивно прив'язані через програму-майстер або вони можуть бути прив'язані автоматично. Автоматичний вибір початкових елементів можна забезпечити декларативним шляхом – засобами

запитів OCL або імперативними викликами мовою Java. Повторне застосування шаблонів підтримується через параметри, які мають тип “колекція”. Кожне застосування шаблону записується разом із всіма його пов’язаними параметрами, і цей запис можна використовувати пізніше для повторного застосування шаблону із цими оригінальними параметрами. XDE не накладає жодних обмежень на відношення між початковою та цільовою моделлю, тобто допускається створення нової моделі, її заміна та оновлення іншої існуючої цільової моделі. Керування послідовністю застосування правил забезпечується шляхом вкладення шаблонів. Більш складне керування може бути запрограмоване мовою Java. Для виконання перетворення модель-код шаблони можуть бути асоційовані із фрагментами коду, які схожі на мову JSP (так звані скріптлети).

Інші підходи модель-модель. Для повноти огляду необхідно зазначити ще два підходи: трансформаційне середовище, що визначено у специфікаціях OMG Common Warehouse Metamodel (CWM) [38] та трансформації, які реалізуються з використанням XSLT [39].

Трансформаційне середовище CWM надає механізм поєднання початкових та цільових елементів, але отримання цільових елементів має бути реалізовано деякою конкретною мовою програмування, яка не описана у межах CWM. Точніше, CWM представляє загальну модель трансформації, але ніяк не реальний механізм реалізації трансформації моделей.

Оскільки моделі можуть бути записані у вигляді тексту XML з використанням XML Metadata Interchange (XMI) [40], дуже актуальною є реалізація трансформації моделей з використанням XSLT, що є стандартною технологією перетворення текстів мовою XML. На жаль, даний підхід має деякі обмеження масштабу використання. Зроблена користувачем реалізація трансформації моделей в XSLT швидко стає незрозумілою через багатослів’я та погану читаємість специфікацій XMI та XSLT. Можливим рішенням направленим на подолання цієї проблеми є генерація правил XSLT із більш декларативних описів правил як показано у [41, 42]. Однак, навіть такий підхід має низьку ефективність через необхідність копіювання, яка вимагається семантикою XSLT “передачі по значенню” та низькою компактністю специфікації XMI.

Таким чином, підходи на основі трансформації графів та реляційні підходи мають чисельні переваги – декларативна форма опису, близькість понять до мови моделювання, що використовується розробниками, відсутність побічних ефектів тощо. Однак такі підходи не використовують нечітких понять при описі моделі та системи перетворень. Щоб використати переваги нечіткої трансформації моделей програмних систем, в даній роботі представлено формалізацію нечітких графів та відношень, які вони породжують.

Формалізація нечітких графів

Нечіткий граф $\langle A,\alpha\rangle$ – це пара, що складається із чіткої множини A та нечіткого відношення $\alpha : A \to A$. Частковий морфізм нечіткого графа $\langle A,\alpha\rangle$ у нечіткий граф $\langle B,\beta\rangle$ - це часткова функція $f : A \to B$, яка задовольняє умові $d(f)\alpha f \subseteq f\beta$, де $d(f)$ – домен відношення f.

Означення 1. Π-об’єкт для пари морфізмів $f : C \to A$ та $g : C \to B$ - це категорія Π разом із парою морфізмів $f' : A \to \Pi$ та $g' : B \to \Pi$ таких, що:

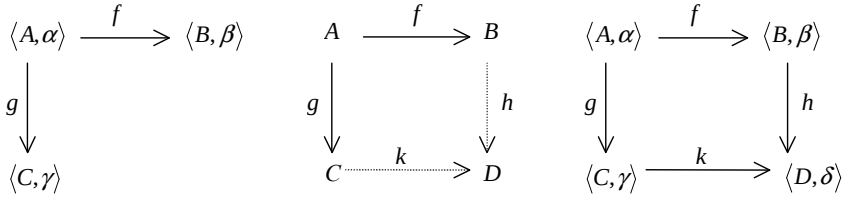
1. $fg' = gf'$;

2. для будь-якого іншого об’єкта X та пари морфізмів $i : A \to X$ та $j : B \to X$, де $fi = gj$ існує тільки один морфізм $k : \Pi \to X$ такий що $i = g'k$ та $j = f'k$.

Позначимо категорію нечітких графів та їх часткових морфізмів як *Pfn(FG)*, $\alpha^{\#}$ – інверсне відношення, id_A – відношення ідентичності на множині A, $Supr(\alpha,\beta)$ – супремум двох відношень. Використовуючи той факт, що категорія *Pfn* множин та часткових функцій є Π-об’єктом, доведемо наступну теорему.

Теорема 1. Категорія *Pfn(FG)* нечітких графів має Π-об’єкт.

Доведення: Розглянемо наступну діаграми, що являють собою *Pfn(FG)* та *Pfn* відповідно. Покажемо зліва часткові морфізми нечітких графів f та g; побудуємо посередині Π-об’єкт для категорії *Pfn*.

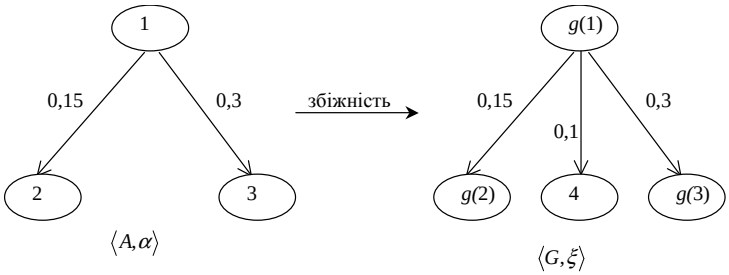


Нарешті, визначимо $\delta = Supr(k^{\#}\gamma k, h^{\#}\beta h)$ як структуру нечіткого графа на D. Отримаємо квадрат морфізмів Π-об’єкта у правій частині малюнку. ■

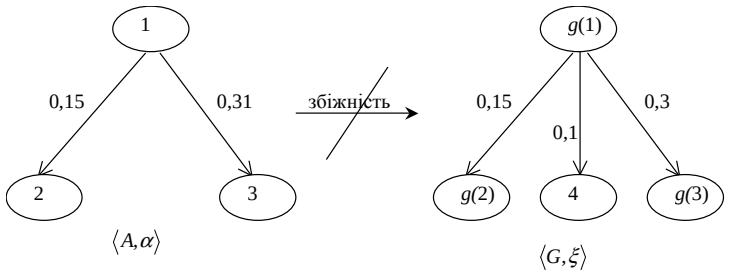
Трансформація нечітких графів включає два окремих поняття. Одне із них – це правило заміни, яке встановлює відповідність між вершинами та може бути формалізоване як часткова функція на множині вершин. Друге – це “збіжність”, при наявності якої можуть бути відповідні правила заміни. Збіжність має виділяти певні підграфи у графах, що підлягають трансформації.

Означення 2. Нехай $\langle A,\alpha\rangle$ та $\langle B,\beta\rangle$ - нечіткі графи. Правило трансформації – це трійка $p=(\langle A,\alpha\rangle,\langle B,\beta\rangle,\ f:A\to B)$, де f є частковою функцією (не обов'язково частковим морфізмом).

Означення 3. Нехай $\langle A,\alpha\rangle$ та $\langle G,\xi\rangle$ - нечіткі графи. Збіжність між $\langle A,\alpha\rangle$ та $\langle G,\xi\rangle$ є ін'єктивний морфізм $g:\langle A,\alpha\rangle\to\langle G,\xi\rangle$, тобто $g^{\#}g\subseteq id_G$, $gg^{\#}\subseteq id_A$ та $\alpha g\subseteq g\beta$.



У прикладі, що наведений вище, $\langle A,\alpha\rangle$ збігається з $\langle G,\xi\rangle$ по g, але не збігається у прикладі, що наведений нижче.



Щоб характеризувати результат виконання правила трансформації над нечітким графом створимо *П*-об'єкт для *Pfn* для даного правила трансформації $p=(\langle A,\alpha\rangle,\langle B,\beta\rangle,\ f:A\to B)$ та збіжності $g:\langle A,\alpha\rangle\to\langle G,\xi\rangle$.

Означення 4. Граф $\langle G,\xi\rangle$ може бути перетворений у $\langle H,\eta\rangle$ застосуванням правила трансформації p при збіжності g, якщо відношення η визначено як $\eta=Supr\left(k^{\#}\left(\xi-g^{\#}\alpha g\right)k,\ h^{\#}\beta h\right)$. Застосування правила трансформації можна розглядати як квадрат:

$$\begin{array}{ccc}\langle A,\alpha\rangle & \xrightarrow{\ f\ } & \langle B,\beta\rangle \\ {\scriptstyle g}\downarrow & & \downarrow{\scriptstyle h} \\ \langle G,\xi\rangle & \xrightarrow{\ k\ } & \langle H,\eta\rangle\end{array}$$

Відзначимо, що квадрат трансформації не обов'язково є *П*-об'єктом. Покажемо що квадрат трансформації є *П*-об'єктом для категорії *Pfn*(*FG*), якщо правила трансформації є частковими морфізмами на графах. Для цього дамо наступне означення.

Означення 5. Нехай $\langle A,\alpha\rangle$ та $\langle G,\xi\rangle$ - нечіткі графи. Ін'єктивна функція $g:A\to G$ є строгою збіжністю між $\langle A,\alpha\rangle$ та $\langle G,\xi\rangle$, якщо $g^{\#}\alpha g$ є чітким відносно ξ.

Для строгої збіжності $g:\langle A,\alpha\rangle\to\langle B,\beta\rangle$ виконуються наступні властивості:

$$\left(\xi-g^{\#}\alpha g\right)=Supr(\xi,\ g^{\#}\alpha g),$$
$$Inf\left(\xi-g^{\#}\alpha g,\ g^{\#}\alpha g\right)=O.$$

Теорема 2. Нехай $p=(\langle A,\alpha\rangle,\langle B,\beta\rangle,\ f:A\to B)$ - правило трансформації та $g:\langle A,\alpha\rangle\to\langle B,\beta\rangle$ – строга збіжність. Якщо $f:\langle A,\alpha\rangle\to\langle B,\beta\rangle$ – частковий морфізм, правило трансформації є *П*-об'єктом для *Pfn(FG).*

Доведення: Якщо f являє собою частковий морфізм, отримуємо $f^{\#}\alpha f=f^{\#}d(f)\alpha f\subseteq f^{\#}f\beta\subseteq\beta$ при $f=d(f)f$ та $k^{\#}g^{\#}\alpha gk=h^{\#}f^{\#}\alpha fh\subseteq h^{\#}\beta h\subseteq\eta$. Це означає що

$$\eta=Supr\left(k^{\#}\left(\xi-g^{\#}\alpha g\right)k,\ h^{\#}\beta h\right)=Supr\left(k^{\#}\left\lfloor Supr\left(\xi-g^{\#}\alpha g,\ g^{\#}\alpha g\right)\right\rfloor k,\quad h^{\#}\beta h\right).$$

Необхідно показати, що $\eta=Supr(k^{\#}\xi k,\ h^{\#}\beta h)$. Але $Supr(\xi-g^{\#}\varepsilon g,\ g^{\#}\alpha g)=\xi$, тому що $g^{\#}\alpha g$ - чітке відносно ξ. Доведення завершено. ■

Визначимо неоднозначну збіжність – так звану ε-збіжність. Результуючий граф, що отримується по неоднозначній збіжності є ε-подібним відносно *П*-об'єктів нечітких графів.

Означення 6. Збіжність $g:\langle A,\alpha\rangle \to \langle G,\xi\rangle$ є ε-збіжністю між $\langle A,\alpha\rangle$ та $\langle G,\xi\rangle$ тоді і тільки тоді, якщо відношення $g^{\#}\alpha g$ є ε-чітким відносно ξ.

Теорема 3. Нехай $p=(\langle A,\alpha\rangle,\langle B,\beta\rangle,\ f:A\to B)$ - правило трансформації. Якщо f у правилі p частковим морфізмом нечітких графів, а $g:\langle A,\alpha\rangle \to \langle G,\xi\rangle$ - ε-збіжність, то $\eta = Supr(k^{\#}\xi k,\ h^{\#}\beta h)$ та $\hat{\eta} = Supr\left(k^{\#}\left(\xi - g^{\#}\alpha g\right)k,\ h^{\#}\beta h\right)$ є ε-подібними. Іншими словами, нечіткий граф $\langle H,\hat{\eta}\rangle$ являє собою наближення до нечіткого графа $\langle H,\eta\rangle$.

Доведення: Легко показати, що

$$\hat{\eta} = Supr\left(k^{\#}\left[Supr(\xi - g^{\#}\alpha g,\ g^{\#}\alpha g)\right]k,\quad h^{\#}\beta h\right).$$

Оскільки $g^{\#}\alpha g$ є ε-чітким відносно ξ, то два нечітких відношення - $Supr(\xi - g^{\#}\alpha g,\ g^{\#}\alpha g)$ та ξ є ε-подібними. Очевидно, що $k^{\#}\left[Supr(\xi - g^{\#}\alpha g,\ g^{\#}\alpha g)\right]k$ та $k^{\#}\xi k$ є ε-подібними. Таким чином, η та $\hat{\eta}$ також є ε-подібними. Доведення завершено. ■

Приведемо приклади квадратів трансформації нечітких графів, які демонструють ε-подібність результуючих графів до *П*-об'єктів.

Приклад 1. Нехай $\xi = 0{,}2$. Оскільки правило трансформації f є частковим морфізмом нечітких графів, то перший приклад на рис.1,а показує квадрат *П*-об'єкта.

Приклад 2. Другий приклад на рис.1,б показує квадрат трансформації із неоднозначною збіжністю. Відмітимо, що результуючий граф є ε-подібним до результуючого графа у першому прикладі.

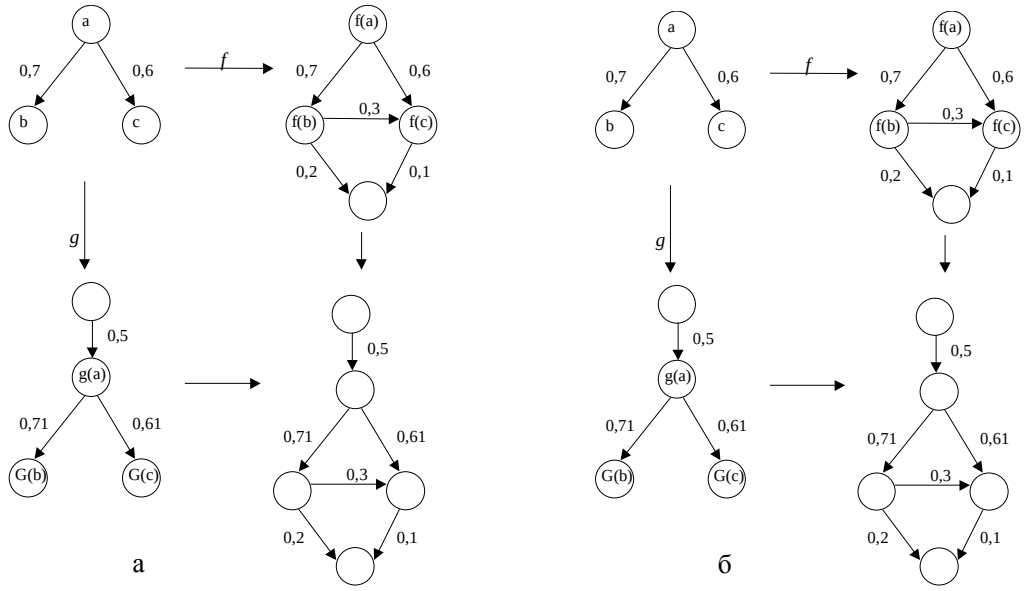


Рис.1. Приклади квадратів трансформації нечітких графів

Трансформація нечітких моделей програмних систем

Абстрактний синтаксис візуальної мови моделювання (такої як UML) визначається як метамодель, основою якої є спрощена діаграма класів (у позначеннях UML). Мовою UML моделі визначаються як конструкції, що є екземплярами метамоделей. Як моделі, так і метамоделі представляють собою направлені графи, вершини та дуги яких мають певні атрибути.

На рівні моделей, сутності та відношення між ними відображаються у вершини та дуги відповідно модельного графу М, який називають також діаграмою.

Означення 7. Застосуванням трансформації нечіткого графа моделі *M* програмної системи (тобто моделі UML користувача) є перетворення нечіткого графа моделі заміною підграфа, який виділяє збіжність (ε-збіжність) у лівій частині правила на підграф, який визначає правило трансформації. Застосування включає наступні кроки:

- пошук у графі моделі *M* ε-збіжності, яке встановлене у нечіткому правилі;
- перевірку можливості застосування правила трансформації, яка виконується якщо певна збіжність знайдена;
- вилучення підграфа моделі *M*, який відповідає знайденій ε-збіжності (отримання контекстного нечіткого графу);
- вставка у контекстний граф нечіткого графа $\langle B,\beta\rangle$ і отримання результуючого графа моделі *M'*.

Процес трансформації складається із застосування трансформацій (мікрокроків) у відповідності до семантики (макрокроків) графа потоку керування трансформаціями.

Означення 8. Система трансформацій нечіткої моделі програмної системи – це трійка $FTS = \langle FG_{INIT}, P, CONF \rangle$, де FG_{INIT} - початковий нечіткий граф моделі, P – множина нечітких правил моделі, а $CONF$ – множина графів потоку керування. Граф потоку керування включає наступні типи вершин: ЗАСТОСУВАТИ, ДЛЯ ВСІХ, ЦИКЛ та дві додаткові вершини – УСПІХ та НЕВДАЧА.

Потік керування, починаючи з початкової вершини, проходить множину визначених вершин, причому з кожним типом вершини пов'язані наступні кроки:

- при досяганні вершини ЗАСТОСУВАТИ, виконується правило трансформації, яке із ним пов'язане. Якщо у графі моделі є певна ε-збіжність, що указана у правилі, перехід здійснюється у вершину УСПІХ, інакше – у вершину НЕВДАЧА;
- при досяганні вершини ЦИКЛ, правило трансформації виконується повторно доки, доки це можливо (існує певна ε-збіжність);
- при вході у вершину ДЛЯ ВСІХ зазначене правило виконується паралельно для всіх знайдених ε-збіжностей у нечіткому графі.

Модель M програмної системи можна представити нечітким графом $\langle A, \alpha \rangle$, де A – множина сутностей, а α – наведені на множині сутностей відношення. Такі відношення можуть задаватися нечітким графом на основі застосування "емпіричних" нечітких правил.

Розглянемо формалізацію відношення на прикладі створення нечіткої асоціації між класами мови UML. Нечіткі відношення між об'єктами x та y, які є екземплярами класів X та Y відповідно, можна записати у вигляді

IF $x \in X$ THEN $y \in Y$ (F_1),

IF $y \in Y$ THEN $x \in X$ (F_2),

де F_1, F_2 – коефіцієнти впевненості. Наприклад, задані правила

IF "Петренко є Учасником" THEN "Київ є Місцем проведення" (0,8),

IF "Київ є Місцем проведення" THEN "Петренко є Учасником" (0,75).

Для визначення функції належності результуючого відношення можна використати принцип максмінного згортання [43]:

$$\mu_B(y) = \max_{x \in X} \{\min\{\mu_A(x), F_{AB}\}\}.$$

Отримуємо $\mu_B(y) = \min\{\mu_A(x), 0{,}8\}$, $\mu_A(x) = \min\{\mu_B(y), 0{,}75\}$, тобто $\mu_A(x) = \mu_B(y) = 0{,}75$.

Нечітка асоціація, що реалізує зазначене відношення, на рис.2 представлена на діаграмі класів.

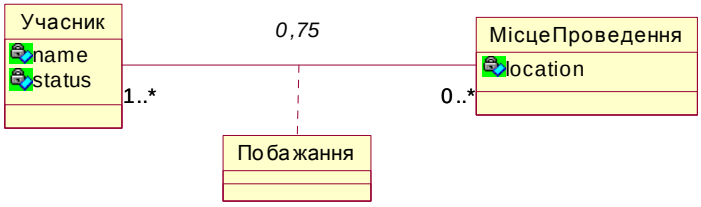


Рис.2. Нечітка асоціація на діаграмі UML

Нечіткість асоціації означає, що об'єкти беруть участь у ній лише певною мірою, яка визначається дугою нечіткого графа.

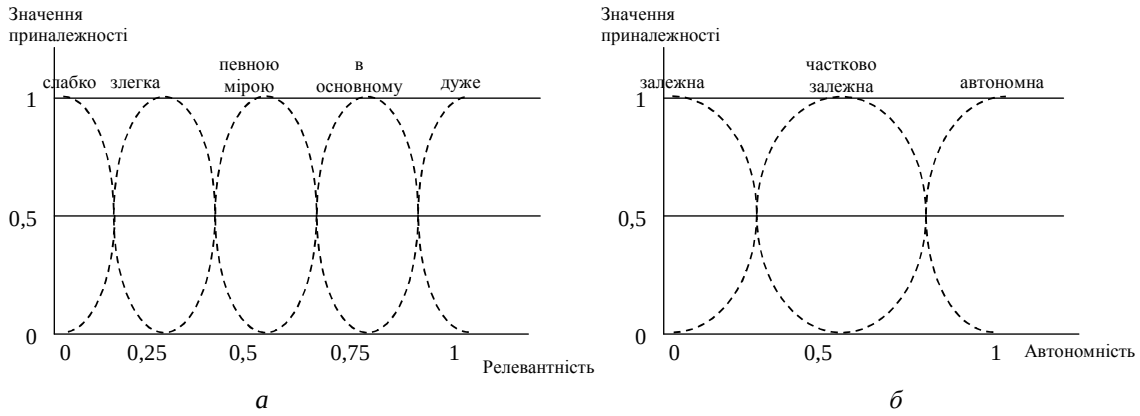
Аналогічно в об'єктно-орієнтованій моделі можна розглядати нечітке узагальнення, нечітке відношення між класом і об'єктом та нечіткий поліморфізм.

Нечіткі правила розробки

Об'єктно-орієнтовані методи описуються в термінах об'єктів розробки, правил ідентифікації, визначення і перетворення об'єктів та системи позначень для представлення об'єктів. Класи об'єктів розробки називають класифікаторами. Для ідентифікації або виключення об'єкта з розробки, а також для зв'язування одних об'єктів з іншими методи задають продукційні правила. Звичайно такі правила визначені неформально текстовими формами природною мовою. Правила неявно виражають, як один класифікатор причинно зв'язаний з іншими класифікаторами.

Розглянемо властивості *Релевантність* і *Автономність* класифікатора *Сутність*. Припустимо, що значення цих властивостей можуть бути виражені дійсними числами в інтервалі [0,1]. У сучасних методах властивості *Релевантність* і *Автономність* мають бути проквантовані відповідно значеннями *релевантна* (1) та *нерелевантна* (0), *автономна* (1) і *неавтономна* (0). Це відбувається тому, що результатом застосування правила *Ідентифікація Попереднього Класу* є груба класифікація сутності як члена категорії *Попередній Клас*. У теорії нечітких множин сутність може бути екземпляром класифікатора *Попередній Клас* лише до деякої міри. Так, властивості *Релевантність* і *Автономність* класифікатора *Сутність* можуть приймати будь-які значення в інтервалі [0,1] і ці значення визначають ступінь членства сутності в категорії *Попередній Клас*.

Опис числових значень для таких властивостей, як *Релевантність* і *Автономність*, може бути дуже непростим, а надійність значень – дуже сумнівною. Для вирішення даної проблеми нечітка логіка пропонує поняття лінгвістичної змінної. Властивості *Релевантність* і *Автономність* можуть бути представлені як лінгвістичні змінні. Значущими лінгвістичними значеннями для цих двох властивостей можуть бути: *слабко*, *злегка*, *певною мірою*, *в основному* та *дуже релевантна* (для властивості *Релевантність*) і *залежна*, *частково залежна* та *цілком автономна* (для властивості *Автономність*). Зміст, пов'язаний з лінгвістичними значеннями *Релевантность* і *Автономність*, показаний на рис.3. При цьому осі *X* та *Y* позначають відповідно простір тверджень і значення приналежності.



а *б*

Рис.3. Зміст лінгвістичних значень *Релевантність* (*а*) і *Автономність* (*б*)

Однак одного лінгвістичного представлення властивостей не досить для рішення проблем з виразністю. Необхідні також засоби для логічного виведення на основі лінгвістичних змінних. Щоб їх забезпечити, нечітка логіка вводить поняття нечітких правил. Нечітке правило виражається як **IF** *X* = *A* **THEN** *Y* = *B*, де *X* і *Y* – лінгвістичні змінні, а *A* та *B* – лінгвістичні значення. Антецедентна частина методологічного нечіткого правила виражає умову, якій повинні задовольняти лінгвістичні значення властивості. Результуюча частина визначає, у якому ступені елемент процесу розробки є членом категорії, що ідентифікується класифікатором. Наприклад, правило *Ідентифікація Попереднього Класу* може бути виражене в такий спосіб:

IF СУТНІСТЬ У СПЕЦИФІКАЦІЇ ВИМОГ Є РЕЛЕВАНТНОЮ ЗНАЧЕННЮ РЕЛЕВАНТНОСТИ **ТА** МОЖЕ ІСНУВАТИ ЗНАЧЕННЯ АВТОНОМНОСТІ У ПРИКЛАДНІЙ ГАЛУЗІ **THEN** ВОНА Є ЗНАЧЕННЯМ ЕКЗЕМПЛЯРА ПОПЕРЕДНЬОГО КЛАСУ.

При цьому *Сутність* і *Попередній Клас* є класифікаторами, над який буде здійснюватися логічне виведення, *Релевантність* та *Автономність* – властивостями, значення релевантності та значення автономності – області значень цих властивостей, значення екземпляра визначає область значень ступеня членства сутності в класифікаторі *Попередній Клас*. Такі ступені членства виражаються лінгвістичними змінними. Достовірними є значення *слабко*, *злегка*, *певною мірою*, *в основному* та *дуже*. Їхній зміст аналогічний показаному на рис.3, *а*. Кожна комбінація значень релевантності й автономності сутності має бути відображена на одне з п'яти значень членства. Дане відображення ґрунтується на інтуїтивному представленні розроблювача методу щодо того, що є попереднім класом. У даному випадку інтуїтивне представлення полягає в тому, що чим більш релевантною й автономною є сутність, тим у більшому ступені сутність є екземпляром класифікатора *Попередній Клас*. Це означає, що значення екземпляра збільшується зі збільшенням релевантності й автономності. Отримані в результаті цього відображення 15 підправил наведені у таблиці. Кожен елемент таблиці, показаний курсивом, представляє вихідне значення підправила, тобто значення членства сутності в класифікаторі *Попередній Клас*. Дані вихідні значення були обрані на основі інтуїції і знання об'єктно-орієнтованих методів. Аналогічно, методологічні правила, що використовуються в сучасних об'єктно-орієнтованих методологіях, можуть бути перетворені в більш інтуїтивні правила, які основані на нечіткій логіці.

Таблиця. Підправила правила *Ідентифікація Попереднього Класу*
(ступінь членства *Сутності* в *Попередньому Класі*)

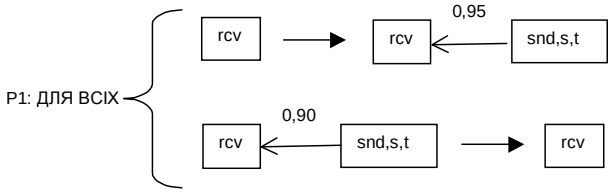
Сутність: Автономність	Сутність: Релевантність				
	Слабко	Злегка	Певною мірою	В основному	Дуже
Залежна	*Слабко*	*Слабко*	*Слабко*	*Слабко*	*Слабко*
Частково залежна	*Слабко*	*Злегка*	*Злегка*	*Певною мірою*	*Певною мірою*
Цілком автономна	*Слабко*	*Злегка*	*Певною мірою*	*В основному*	*У великому ступені*

Оскільки отримані нечіткі правила спираються на представлення артефактів у вигляді графових моделей, то дефазифікація [44] нечітких правил розробки дозволяє отримати систему трансформацій над нечіткими графами.

Приклад 3. Агенти та мультиагентні системи – це нова парадигма програмування, яка призначена для створення розподілених гетерогенних інформаційних систем, що працюють в Інтернеті[45]. Покажемо, яким чином агенти можуть моделюватися системою трансформацій моделей на основі нечітких графів.

Поведінка кожного агента визначається множиною графів потоку керування CONF. Агенти можуть модифікувати своє середовище, тобто нечіткий граф, який його задає. Мультиагентна система – це множина агентів, які обмінюються повідомленнями у спільному середовищі. У нашому прикладі для взаємодії використовуються механізми обміну повідомленнями та "дошки оголошень". Інші, більш складні механізми взаємодії агентів [45] можуть моделюватися з використанням двох вищезазначених. Кожний із агентів має неповне знання, але вирішення задачі досягається шляхом їх кооперації.

Припустимо, що користувачу, якого представляє певний агент A_u, необхідна інформація щодо надійності взаємодії між пунктами a та b. Це виражається як функція належності $\mu_A : E \to [0,1]$, де $E = \{e_1, e_2, ..., e_n\}$- множина дуг графа. Чим більше вірогідність відмови, тим більше значення функції належності дуг нечіткого графа.



Правило, що наведене вище, реалізує асинхронну передачу повідомлення між агентом, що посилає та агентом, що приймає повідомлення. При цьому агент *rcv* приймає повідомлення з вірогідністю не більше 0,9 через ненадійність комунікаційного середовища. Результат застосування цієї трансформації наведений на рис.4.

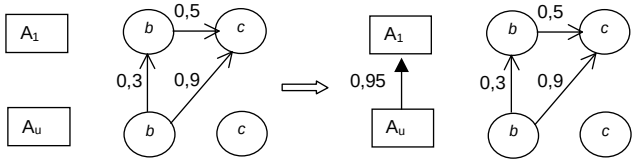


Рис.4. Надсилання повідомлення

Коли агент приймає повідомлення, він може реагувати або ні. Ця властивість демонструє надання сервісу та автономність агентів. Для кожного типу реакції всі агенти мають ті ж самі нечіткі трансформації:

ДЛЯ ВСІХ {ДЛЯ ВСІХ P1; ЗАСТОСУВАТИ P2}.

Наведене нижче правило складається у виділенні початкової та кінцевої вершини із повідомлення, що надійшло, та модифікації локального графа агента. Воно має бути застосовано синхронно із правилом p_1 для того, щоб запобігти повторній обробці повідомлення:

Застосування правила p_2 показано на рис.5.

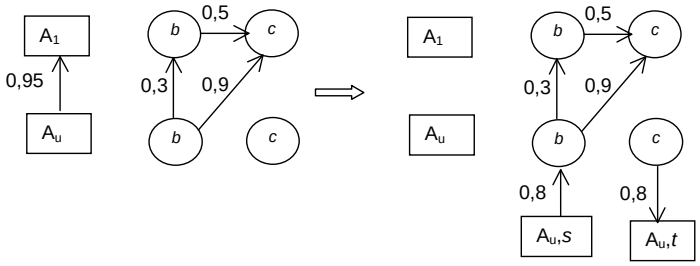
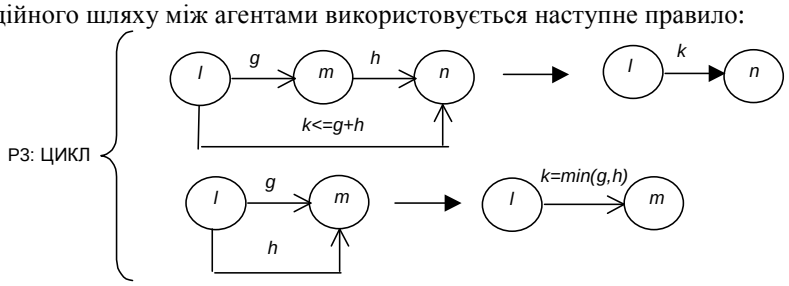
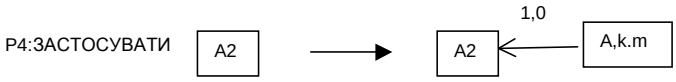


Рис.5. Модифікація локального графа агента

Для пошуку надійного шляху між агентами використовується наступне правило:



Після закінчення циклу маємо найнадійніший шлях між початковим та кінцевим пунктами, функція належності якого має мінімальне значення. Якщо агент не знаходить шляху між початковою та заключною вершиною, він надсилає повідомлення іншому агенту, нечіткий граф якого може відрізнятися від графу агента A_1 (A_2 має інші знання про середовище).



Граф потоку керування CONF системи трансформацій нечіткого агента представлено на рис.6.

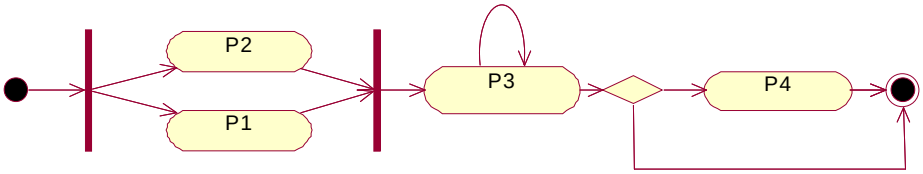


Рис.6. Граф потоку керування системи трансформацій нечіткого агента

Структури даних програмного агента можуть бути представлені як нечіткі графи. Політика, яку використовує окремий агент для визначення дій верхнього рівня, відповідь на запити або події, описується у вигляді графа системи трансформацій моделі агента.

Таким чином, в мультиагентних системах може застосовуватися розподілена система трансформацій нечіткої моделі, у якій перетворення на нечітких графах виконуються паралельно для кожного окремого агента.

Нечіткий трансформаційний підхід до розробки програмних систем означає ітеративне виконання наступних етапів:

- введення лінгвістичних змінних, що характеризують артефакти розробки;
- установлення нечітких правил розробки програмних систем (ПС) як нечіткої трансформаційної системи, що виконується над моделями;
- дефазифікацію системи нечітких правил і побудову нечіткого графу трансформацій, дуги якого мають певні значення імовірностей;
- побудову нечіткої моделі програмної системи: статичної, динамічної тощо з використанням нечітких графів і відношень;
- послідовне виконання трансформацій моделі ПС з метою розробки більш детальних моделей, або моделей, які залежать від конкретного середовища. Такі перетворення можуть бути виконані для включення елементів програмного забезпечення проміжного прошарку (CORBA, EJB, DCOM) у склад моделі.

Дана методологія, яка ґрунтується на нечітких графах, була реалізована в експериментальному середовищі CASE і протестована на серії задач-зразків. Середовище надає інструментальні засоби для визначення лінгвістичних змінних, значень і нечітких правил з використанням засобів моделювання мовою UML[5], розширених засобами нечіткої логіки. Розроблювач ПЗ взаємодіє з інструментальними засобами, що містять нечіткі правила проектування та з моделлю системи, представленій у вигляді нечіткого графу. Основні компоненти трансформаційного середовища наведені на рис.7.

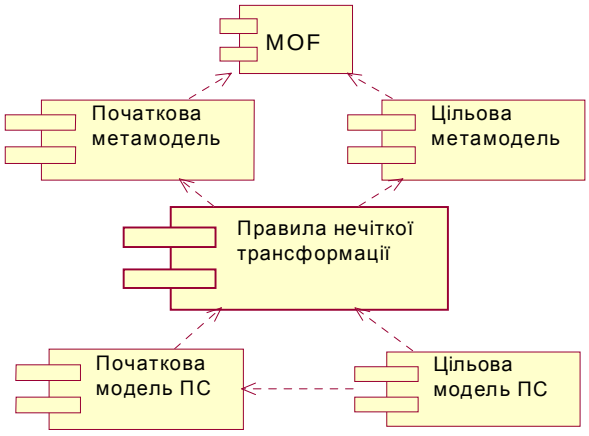


Рис.7. Структура системи трансформації моделей на основі нечітких графів

Висновки

Основний акцент у даній роботі зроблено на формалізації процесу розробки програмного забезпечення на основі нечітких графових моделей. Модель програмної системи представляється у вигляді нечіткого графу,

вершинами якого є сутності та об'єкти. Процес розробки програмного забезпечення полягає у виконанні системи трансформацій над нечіткими графами.

Крім того, в роботі розглянуто методологічні правила об'єктно-орієнтованої розробки програмного забезпечення, які основані на нечіткій логіці. Продемонстровано, що на основі нечіткої моделі більш адекватно, ніж на основі двозначної логіки, можна формалізувати розуміння правил розробниками програмного забезпечення завдяки можливості описати лінгвістичні вирази, що зустрічаються в реальних проектах.

У даній роботі, був представлений огляд процедури трансформацій середовища програмних агентів на основі нечітких графів. Такий підхід може використовуватися як на етапі аналізу для формалізації вимог до програмних систем, так і для автоматизованого перетворення моделей вимог і аналізу до платформно-залежних моделей.

1. *Сергієнко І.В., Парасюк І.М.* Нечіткі інформаційно-діагностичні технології: проблеми становлення // Вісн. НАН України. – 2002. – №7. – С. 21–28.
2. *Ершов С.В.* Спецификация отношений между визуальными моделями программных систем // Пробл. программирования. – 2000. – №3-4. – С. 82–96.
3. *Broy M.* Toward a Mathematical Foundation of Software Engineering Methods // IEEE Transactions on Software Engineering, Vol.27, №1, 2001. – pp.42-57.
4. *Object Management Group*. Model Driven Architecture – A Technical Perspective, 2001. http://www.omg.org/mda
5. *Буч Г., Рамбо Д., Джекобсон А.* UML. Руководство пользователя. – М.: ДМК, 2000. – 432 с.
6. *Элиенс А.* Принципы объектно-ориентированной разработки программ. 2-е изд. : Пер. с англ. – М.: Изд. дом "Вильямс", 2002. – 496 с.
7. *Jamda*: The Java Model Driven Architecture 0.2, May 2003, http://sourceforge.net/projects/jamda/
8. *Rational XDE*, http://www.rational.com/products/xde
9. *OptimalJ* 3.0, User's Guide, http://www.compuware.com/products/optimalj
10. *OMG/MOF* Meta Object Facility (MOF) Specification. OMG Document AD/97-08-14, September 1997. http://www.omg.org
11. *b+m ArchitectureWare*, Generator Framework, http://www.architectureware.de
12. *Java Emitter Templates* (JET). Part of the Eclipse Modeling Framework, JET Tutorial by Remko Pompa, http://eclipse.org/articles/Article-JET2/jet_tutorial2.html
13. *FUUT-je*, Eclipse Generative Model Transformer (GMT) project website, http://dev.eclipse.org/viewcvs/indextech.cgi/~checkout~/gmt-home/download/index.html
14. *Codagen Architect 3.0*, http://www.codagen.com/products/architect/default.htm
15. *AndroMDA* 2.0.3, July 2003, http://www.andromda.org
16. *ArcStyler* 4.0, September 2004, http://www.arcstyler.com/
17. *Velocity* 1.3.1, The Apache Jakarta Project, March 2003, http://jakarta.apache.org/velocity/
18. *XDoclet* - Attribute Oriented Programming, http://xdoclet.sourceforge.net/
19. *Cleaveland C.*.Program Generators with XML and Java. Prentice-Hall, 2001, http://www.craigc.com/pg/
20. *Java Metadata Interface* 1.0, July 2002, http://java.sun.com/products/jmi
21. *OMG*, The Object Constraint Language Specification 2.0, OMG Document: ad/03-01-07, http://www.omg.org
22. *W3C*, XML Path Language Version 1.0, November 1999, http://www.w3.org/TR/xpath
23. *XML-based Variant Configuration Language*, http://fxvcl.sourceforge.net/
24. *Frame-Processing-Language*, http://sourceforge.net/projects/fpl
25. *Emrich M.* Generative Programming Using Frame Technology. Diploma Thesis, University of Applied Sciences Kaiserslauten, Department of Computer Science and Micro-System Engineering, Zweibrücken 2003, http://www.geocities.com/mslerm/xframerf.htm
26. *Frame Processor ANGIE*, Delta Software Technology, http://www.d-s-t-g.com/neu/pages/pageseng/et/common/techn_angie_frmset.htm
27. *Akehurst D. H., Kent S.* A Relational Approach to Defining Transformations in a Metamodel // Lecture Notes in Computer Science 2460, 2002. – P.243-258.
28. *Gerber A., Lawley M., Raymond K., Steel J., Wood A.* Transformation: The Missing Link of MDA // Lecture Notes in Computer Science, vol. 2505, Springer-Verlag, 2002. – PP.90 – 105.
29. *Andries M., Engels G., Habel A., Hoffmann B., Kreowski H.-J., Kuske S., Kuske D., Plump D., Schürr A., Taentzer G.* Graph Transformation for Specification and Programming. Technical Report 7/96, Universität Bremen, 1996. – 82 p.
30. *Varro D., Varro G., Pataricza A.* Designing the automatic transformation of visual languages // Science of Computer Programming, vol. 44(2), 2002. – PP. 205-227.
31. *ATOM 3*: A Tool for Multi-Paradigm modeling, http://atom3.cs.mcgill.ca/
32. *Agrawal A., Karsai G., Shi F.* Graph Transformations on Domain-Specific Models // *Journal on Software and Systems Modelin*g, 2003. – P.24-37.
33. *Willink E.D.* UMLX: A graphical transformation language for MDA // CTIT Technical Report TR–CTIT–03–27, University of Twente, 2003. – PP. 13-24.
34. *Braun P., Marschall F.* The Bi-directional Object-Oriented Transformation Language. Technical Report, Technische Universität München, TUM-I0307, May 2003. – 41 p.
35. *Marschall F., Braun P.* Model Transformations for the MDA with BOTL // CTIT Technical Report TR–CTIT–03–27, University of Twente, 2003. – PP.25-36.
36. *Bézivin J., Dupé G., Jouault F., Rougui J. E.* First experiments with the ATL model transformation language: Transforming XSLT into XQuery // OOPSLA'03 Workshop on Generative Techniques in the Context of the MDA, http://www.softmetaware.com/oopsla2003/mda-workshop.html
37. *Gamma E., Helm R., Johnson R., Vlissides J.* Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, Reading, MA, 1995.
38. *OMG*, The Common Warehouse Model 1.1., OMG Document: formal/2003-02-03, http://www.omg.org
39. *W3C*, XSL Transformations (XSLT) Version 1.0, November 1999, http://www.w3.org/TR/xslt
40. *OMG*, XML Metadata Interchange Specification 1.2, OMG Document: formal/02-01-01, http://www.omg.org
41. *Peltier M., Bézivin J., Guillaume G.* MTRANS: A general framework based on XSLT for model transformations // WTUML'01, Proceedings of the Workshop onTransformations in UML, Genova, Italy, April 2001. – PP.123-126.
42. *Peltier M., Ziserman F., Bézivin J.* On levels of model transformation // XML Europe 2000, Paris, France, June 2000, Graphic Communications Association. – PP.1–17.
43. *Прикладные* нечеткие системы : Пер. с японского / Под ред. Т.Тэрано, К.Асаи, М.Сугэно. – М.: Мир, 1993. – 366 с.
44. *Zadeh L.A.* Fuzzy Logic = Computing with Words // IEEE Transactions on Fuzzy Systems. – 1996. – **4**, № 2. – P. 103–111.
45. Subrahmanian V. S., Bonatti P., Dix J., Eiter T., Kraus S., Ozcan F., Ross R. Heterogeneous Agent Systems. The MIT Press, 2000. – 640 pp.