УДК 004.413.2

Є.Ю. Катаєва, Г.О. Заспа, Р.В. Форостянов

Черкаський державний технологічний університет, Черкаси

МОДЕЛЬНО-ОРІЄНТОВАНИЙ ПІДХІД ДО РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розглянуто загальні поняття та принципи модельно-орієнтованої інженерії як однієї з найвикористовуваних модельно-орієнтованого підходу від Object Management Group (OMG) – Model-Driven Architecture (MDA) та переваги, які надає MDA у порівнянні з традиційними підходами до розробки програмних систем. Особлива увага приділена інструментам для трансформацій моделей.

Ключові слова: модельно-орієнтований підхід, модель, мета-модель, модель незалежна від платформи, модельно-орієнтована інженерія

В работе рассмотрены общие понятия и принципы модельно-ориентированной инженерии (Model-Driven Engineering(MDE)) как одной из самих используемых модельно-ориентированного подхода от Object Management Group (OMG) – Model-Driven Architecture (MDA) и преимуществ, которые предоставляет MDA в сравнении с традиционными подходами к разработке программных систем. Особое внимание уделено инструментам для трансформации моделей.

Ключевые слова: модельно-ориентированный подход, модель, мета-модель, модель независимая от платформы, модельно-ориентированная инженерия

General concepts and principles of the model-oriented engineering (Model-driven Engineering(MDE)) are in-process considered, as to one of the known representatives of the model-oriented hike from Object Management Group (OMG) – Model-driven Architecture (MDA) and advantages which are given by MDA by comparison to the traditional going near development of the programmatic systems. The special attention is spared instruments for transformation of models.

Keywords: model-oriented approach, model, meta-model, a model is platform-independent, model-oriented engineerin

Вступ

Модельно-орієнтований підхід використовує моделі як основний артефакт протягом всього життєвого циклу розробки програмних систем. Цей підхід використовується для побудови програмних продуктів, проектування баз даних та систем загалом.

Модельно-орієнтований підхід є перспективним для подолання проблем третього покоління мов програмування пов'язаних із послабленням складності платформ та ефективним відображенням концепцій предметної області [3].

Але ефективність застосування цього підходу багато в чому залежить від ефективності наявних інструментів для роботи з моделями. Особливо важливим інструментом є інструмент для трансформації моделей.

Трансформація здійснюється для модифікації моделей з метою аналізу, оптимізації, розвитку, а також генерації коду.

Особливий інтерес в моделях вбачається в можливості їхнього застосування для заповнення прогалини між предметною областю (реальним світом) та машинними кодами: людина використовує предметну область (реальний світ) для постановки вимог (формулювання задач), в той час як машина лише виконує команди. Програмісти перетворюють вимоги в машинні коди.

Для того, щоб автоматизувати цей процес перетворення використовують напрямок розвитку модельно-орієнтованої інженерії (***MDE***).

Найбільш відомою організацією, яка розробляє і супроводжує стандарти в цій сфері (і не тільки), є OMG. Запропонований OMG підхід для розробки програмного забезпечення має назву ***Model-Driven Architecture (MDA).*** Іншою відомою гілкою MDE є ***Model Integrated Computing(MIC)***.

Отже, розглянемо загальні визначення основним поняттям MDE (модельно-орієнтованого підходу).

Слід спочатку зазначити, що серед спільноти, яка вивчає модельно-орієнтований підхід, в силу різних причин, не існує однозначного тлумачення основних термінів: моделі, мета-моделі [1; 2]. Зупинимось на більш компромісних, на наш погляд, варіантах.

1. Основні визначення MDE

„Все є моделями" – основний принцип MDE [8].

Модель – це спрощення наявної системи, побудоване у відповідності з певною метою. Іншими словами, модель є замінником реальної системи і має давати відповіді на поставлені питання на місці реальної системи.

Наприклад глобус – модель реальної системи, нашої планети. Глобус не містить всі деталі реальної системи (є спрощеним), а містить лише ті, які нам зручні (розміри) та потрібні (материки, океани і т.д.) для пошуку відповіді на поставлене питання з географії (Де знаходиться Україна?). Інший приклад моделі: побудована за допомогою комп'ютера віртуальна модель літака. Ці приклади дають нам змогу зрозуміти декілька речей.

По-перше, що поняття моделі є абстрактним і воно може мати різне вираження: реального об'єкта, як глобус, або віртуальне, як програмна модель літака.

По-друге, розуміємо, що модель літака не завжди є спрощенням вже наявної системи. Літак, зазвичай, спочатку винаходять, моделюють. А поява літака як наявної системи – вторинна.

Так ми приходимо до визначення двох різних видів моделей: ***специфікуючої*** – опис системи, яка ще з'явиться (літака, як у випадку з моделлю літака) та ***описової*** – опис вже наявної системи (планети Земля, як у випадку з глобусом) [4]. Відповідність (невідповідність) моделі дійсності щодо означених деталей у кожному виді систем будемо називати по-різному. Для специфікуючої моделі це коректність (некоректність), для описової – істинність (неістинність або хибність). Цей поділ відносний. Так, модель літака є специфікуючою для ще неіснуючого літака, але, разом з тим, описовою для вже наявної, хоч і абстрактно, системи – в уяві інженера.

Модель є ***відображенням*** системи. Позначимо відношення відображення через μ.

Слід зазначити, що побудована модель також може виступати в якості системи, для якої в свою чергу також можна побудувати модель. Так, наприклад, змалювавши з глобуса країни Європи, ми можемо сказати, що побудували описову модель глобуса.

Такий ланцюжок відношень типу система-модель теоретично може бути побудований як завгодно довгим, як завгодно віддаленим від початкової системи (рис. 1).

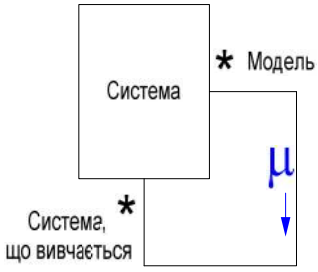

Рис. 1. Схема відношень відображення

Для нас важливо, щоб модель описувала частину або цілу систему, на формалізованій мові. Під ***формалізованою*** мовою будемо розуміти мову з чітко визначеним синтаксисом та значенням, яка придатна для інтерпретації комп'ютером. Яскравим прикладом формалізованих мов можуть бути мови програмування (Паскаль), мови запросів (SQL), мови опису документів (XML).

Моделюванням – побудовою моделей, людство займається протягом багатьох років. Історії відомі випадки побудови моделі світу, зокрема в стародавньому Єгипті.

Мета-модель – модель мови моделювання.

Так, модель написана мовою своєї мета-моделі. Прикладом мета-моделі можуть слугувати форми Бекуса-Наура (БНФ), які визначають мову програмування Паскаль. Так, програму на Паскалі, можна уявити як модель, а БНФ як мета-модель.

Модель до мета-моделі знаходиться у відношенні ***відповідності,*** оскільки модель написана на мові що мета-модель визначає. Позначимо це відношення через χ. Мета-модель за визначенням є моделлю, отже вона написана на мові своєї мета-моделі – так званої мета-мета-моделі. Ланцюжок відношень модель-мета-модель теоретично може бути як завгодно довгим.

Ми коротко оглянули загальні поняття MDE необхідні для кращого розуміння матеріалу висвітленого в цій статті.

2. Огляд MDA

Щоб краще зрозуміти призначення різних стандартів в MDA, розглянемо поняття ***рівнів моделювання***.

OMG використовує чотирирівневу архітектуру. Її часто називають, характеризуючи її вміст, архітектурою 3+1. Рівні в ній позначаються таким чином: M0, M1, M2, і M3 (рис. 2).

Самим нижнім рівнем є M0 – рівень досліджуваної нами системи. Модель відображає (μ) цю систему на рівні M1. Модель рівня M1 відповідає (χ) своїй мета-моделі, яка знаходиться на рівні M2. А мета-модель, в свою чергу, відповідає мета-мета-моделі на рівні M3.

Завершується цей ланцюжок відношень рівнем мета-мета-моделі встановленням відношення відповідності на себе.

Ми вже говорили, що мета-модель визначає мову своєї моделі, а модель, в свою чергу, написана на мові своєї мета-моделі. Іншими словами: модель, за визначенням, написана згідно (χ) зі своєю мета-моделлю. В архітектурі 3+1 ми бачимо три відношення відповідності (χ): M1-M2, M2-M3,M3-M3 та одне відношення відображення (μ): M0-M1.

Існує дві причини чому побудова мета-моделей важлива для модельно-орієнтованого підходу.

По-перше, нам потрібен механізм для визначення недвозначних (формалізованих) мов моделювання. В модельно-орієнтованому підході такий механізм здійснюється через побудову мета-моделей (мета моделювання).

По-друге, ключові для модельно-орієнтованого підходу механізми трансформації моделей використовують мета-моделі. Інструменти, що реалізують механізми перетворень, вміщують опис, як модель на вхідній мові може бути перетворено в моделі на вихідній мові. Ці правила використовують мета-моделі початкових і цільових мов для визначення перетворення.

Наявність трьох рівнів моделювання, кожен з яких знаходиться у відповідності з вищим рівнем (крім M3), дає змогу проводити трансформації моделей на всіх трьох рівнях.

Одним з головних намірів MDA є відділення логіки предметної області від конкретної технології реалізації, дозволяючи цим двом частинам змінюватись незалежно одна від одної.

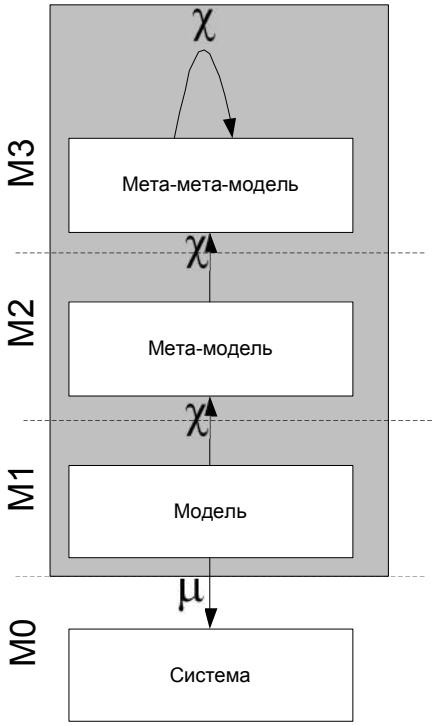

Рис. 2. Архітектура 3+1

Для позначення моделей, що вміщують логіку предметної області (незалежних від платформи моделей) в MDA використовується термін PIM (Platform Independent Model).

Модель незалежна від платформи (PIM) – це модель побудована на високому рівні абстракції, незалежна від будь-якої технології реалізації.

PIM моделюється з точки зору предметної області.

Не дивлячись на те, що PIM є достатньо детальною, її ще не використовують як програмний продукт. Другим кроком є перетворення PIM в платформно-залежну модель – PSM (platform-specific model). Це перетворення відбувається за допомогою стандартних інструментів трансформацій. PSM додатково ще містить елементи характерні для конкретної технології реалізації. Одна PIM трансформується, зазвичай, не в одну PSM, а в декілька. Так, з одної PIM можна одночасно взяти інформацію для побудови PSM БД та PSM програми, яка б працювала з БД (рис. 4).

Дійсно, тяжко провести межу між моделлю незалежною від платформи та моделлю залежною від платформи. Єдине, що можна стверджувати напевно, це те, що одна модель має бути менш (більш) платформно-залежна ніж інша. Отже, в модельно-орієнтованому підході ми отримуємо з моделі менш платформно-залежної на більш платформно-залежну.

Наступним кроком є генерація коду програми з платформно-залежної моделі. Після цього, з кодом проводиться оптимізація, доопрацювання, де це потрібно, звична компіляція, збирання та налагодження системи.

PIM, PSM та програмний код є моделями M1-го рівня.

Трансформаційні інструменти спрощують процес розробки, автоматизуючи його, де це можливо.

Як бачимо, інструменти трансформацій є одними з ключових в модельно-орієнтованому підході. Всередині інструмента трансформації існують визначення, які описують: яким чином модель має бути трансформована.

Загалом, ми можемо сказати, що трансформаційні визначення складаються із трансформаційних правил, які однозначно визначають кроки, за допомогою яких одна модель (або її частина) трансформується в іншу.

Отже, ***трансформація*** – це автоматична генерація цільової моделі з початкової моделі, згідно з трансформаційними визначеннями.

Трансформаційні визначення – це набір трансформаційних правил, які разом описують як модель, описана на початковій мові, може бути трансформована в модель на цільовій мові.

Трансформаційні правила – це опис, як одна, або більше конструкцій на початковій мові можуть бути трансформовані в одну, або більше конструкцій на цільовій мові.

Початкова та цільова моделі можуть бути написані як однією мовою, так і різними. Наприклад, техніка рефакторингу моделі, або ж програмного коду (код є також моделлю) може бути описана трансформаційними визначеннями між моделями написаних на одній мові.

2.1. Стандарти OMG

MOF (Meta Object Facility). MOF – стандарт OMG, який визначає мову, яка слугує для визначення мов моделювання.

MOF знаходиться на найвищому рівні (M3) моделі чотирирівневої архітектури OMG. MOF є мета-мета-моделлю для всіх мета-моделей (UML і CWM зокрема). MOF використовується не лише для побудови мов моделювання, але й для побудови інструментів для побудови мов моделювання. Тому MOF забезпечений додатковою функціональністю.

MOF також використовується для визначення потокового або файлового формату обміну для моделей рівня M1. Кожна мова моделювання визначена згідно з мета-моделлю описаної в MOF. MOF визначає стандартний шлях проводити обмін для мов моделювання. Цей формат обміну базується на XML і має назву XMI (XML Metadata Interchange). Оскільки MOF є мета-моделлю для себе, XMI може використовуватися також для генерації стандартного формату обміну для мета-моделей.

UML (Unified Modeling Language). OMG підтримує декілька мов для моделювання, які можуть бути застосовані для написання PIM або PSM. Найвідомішою з них є UML.

UML стандартизована мова для побудови моделей. Знаходиться на рівні M2 архітектури.

UML містить механізми профілювання для розширення власного словника, дозволяючи на основі наявних конструкцій створювати нові, специфічні для вирішення конкретної проблеми. Багато профілів стандартизовані OMG.

Є два різні напрямки застосування UML в MDA.

Перший – найпоширеніший, в якому розробник повинен уміти використовувати UML, щоб створити модель системи. Він має знати, як і де застосувати мову UML, щоб розробляти моделі, які точні і достатньо формалізовані для використання в MDA.

Другий – спеціальний, яким займається набагато менше розробників має за завдання написання перетворення між моделями. Ця група розробляє трансформаційні інструменти, які використовуються для багатьох моделей, для моделей різних систем. Будемо називати таких спеціалістів мета-розробниками. Подібно до розробника, мета-розробник повинен мати досконале розуміння мови UML і її використання. Додатково він має бути добре ознайомленим з мета-моделлю UML.

OCL (Object Constraint Language). Мова запитів та виразів для UML, яка є складовою частиною UML стандарту. Слово Constraint (обмеження) в абревіатурі OCL залишилося з часів, коли ця мова використовувалася лише для визначення обмежуючих умов для моделей написаних на UML. Нині OCL є повнофункціональною мовою запитів. Вона використовується також для написання різного роду виразів до моделей (наприклад, правил розрахунку атрибутів, інваріантів, передумов, післямов і т.д.). OCL може використовуватись також для MOF моделі.

За допомогою OCL розширюється виразність UML/MOF. Це дозволяє розробнику створювати точніші і вичерпніші моделі.

CWM (Common Warehouse Metamodel). Інша мова моделювання, яку підтримує OMG. Ця мова спеціально створена для моделювання реляційних баз даних. Мета-модель CWM має багато спільного з мета-моделлю UML, але ще має ряд спеціальних мета-класів для моделювання реляційних баз даних. Динамічні частини мета-моделі UML (подібно statemachines або collaborations) в CWM відсутні.

***QVT* (Query, Views, and Transformations).** Цей стандарт визначає яким чином відбувається перетворення між моделями написаних на мовах, які визначені за допомогою MOF. Цей стандарт, в майбутньому, буде частиною MOF, і буде вміщувати:

1. Мову для створення відображень моделей (views);
2. Мову запитів до моделей;
3. Мову для написання трансформаційних визначень.

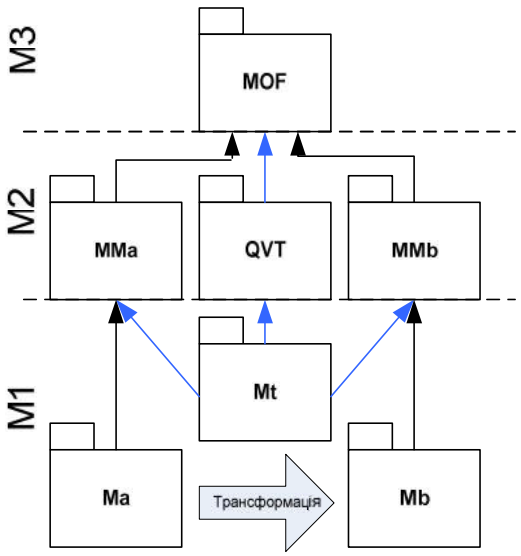

Рис. 3. Трансформація моделей

3. Переваги в процесі розробки, які надає модельно-орієнтований підхід у порівнянні з традиційним

Цикл розробки програмного забезпечення, який ми знаємо нині (традиційний), включає в себе такі стадії:

1. Концептуалізація та збирання вимог;
2. Аналіз і функціональний опис;
3. Проектування;
4. Кодування;
5. Тестування;
6. Розгортання.

Цикл розробки за модельно-орієнтованим підходом:

1. Концептуалізація та збирання вимог;
2. Аналіз і функціональний опис (PIM);
3. Проектування (PSM);
4. Кодування (Код програми);
5. Тестування;
6. Розгортання.

Цикл розробки за модельно-орієнтованим підходом, на перший погляд, не відрізняється від традиційного циклу, але існує одна суттєва різниця. Вона полягає в результатах роботи кожної стадії процесу розробки. Результати роботи кожної стадії за модельно-орієнтованим підходом – формалізовані моделі (описані за допомогою формалізованої мови).

Розглянемо, які переваги дає нам возначена різниця.

Продуктивність

Недоліки традиційного підходу

Документація створюється на 1-3 стадіях традиційного процесу розробки і займає досить багато часу від загального часу розробки. Вона містить опис програмного забезпечення, зазвичай у вигляді тексту, зображень, діаграм класів, діаграм прецедентів та інших. Кількість рядків та зображень, які містить документація, часом вражає.

Крім того, документація втрачає свою цінність з початком стадії кодування. Справа в тому, що змінюючи код, розробники часто не роблять відповідних поправок в документації. Отже, розвиваючись, з часом система набуває все більшої і більшої невідповідності зі своєю документацією.

Переваги модельно-орієнтованого підходу

В модельно-орієнтованому підході характерним є те, що особлива увага приділяється розробці PIM – моделі, яка сама по собі є документацією.

Крім того, приділяючи більше уваги PIM, розробники краще вирішують проблеми саме предметної області.

Також, розробляючи PIM, програмісти витрачають, в порівнянні зі стандартним підходом, значно менше часу, оскільки велика частина роботи виконується автоматично програмними інструментами.

Отже, користувач отримує кращий за функціональністю продукт в коротший термін.

Мобільність

Недоліки традиційного підходу

Час від часу неодмінно з'являється певна нова технологія і стає популярною (наприклад Java, Linux, XML, HTML, SOAP, UML, J2EE, .NET, JSP, ASP, Flash, Web Services і т.д.). Багатьом компаніям необхідно слідувати за новими технологіям з таких причин:

1. Нова технологія є вимогою клієнта (Web interfaces);
2. Вона вирішує певну проблему (XML для обміну або Java для крос-платформеності);
3. Постачальники програмного забезпечення припиняють підтримку старих технологій (в UML відмовились від OMT).

Нові технології пропонують нові корисні рішення, тому розробники мають вивчати нові технології досить швидко. Як висновок, інвестиції у попередні технології втрачають свою цінність.

Переваги модельно-орієнтованого підходу

Всім системам, що змодельовані на рівні PIM, властива мобільність. Вони можуть бути перенесені на будь-яку платформу. Для нових технологій і, що цікаво, для технологій, що з'являться в майбутньому за наявності відповідних засобів для трансформацій (перетворень) моделей, система розгортається відносно легко.

Комунікабельність

Недоліки традиційного підходу

Програмні системи як правило не існують в ізоляції. Більшість потребує зв'язку з іншими, часто вже наявними програмними системами.

З часом ми перестали розробляти величезні монолітні системи. Замість цього ми розробляємо компоненти, які виконують ті ж функції взаємодіючи. Такий компонентний підхід полегшує процес внесення змін в програму. Різні компоненти побудовані на різних технологіях, що якнайкраще підходять для реалізації функцій того чи іншого компонента. Така ситуація вимагає створення механізмів для взаємодії між компонентами.

Переваги модельно-орієнтованого підходу

Модельно-орієнтований підхід вирішує цю проблему за допомогою генерації не тільки двох PSM моделей, але й необхідного для взаємодії містка. Якщо ми здійснюємо перетворення з PIM в

дві PSM, всю необхідну інформацію для побудови містка ми маємо. Для кожного елемента PSM моделі ми знаємо з якого елемента PIM моделі він перетворений. Далі, для цього елемента PIM моделі ми можемо знайти відповідний елемент в другій PSM. Отже, знаючи це, ми можемо вивести, як елемент з однієї PSM зв'язаний з елементом іншої PSM. І цього достатньо для того, щоб згенерувати місток між двома PSM (рис. 4).

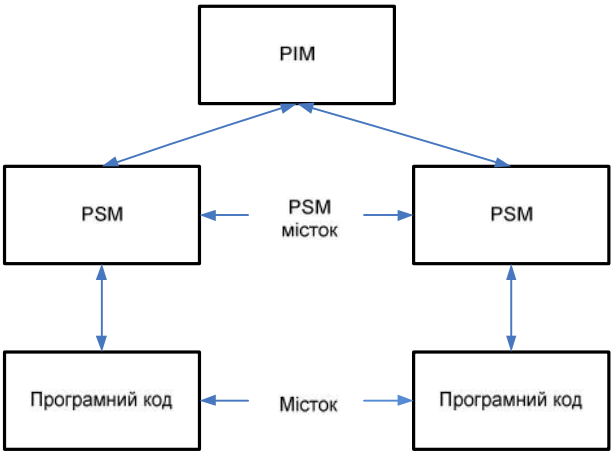

Рис. 4. Трансформація однієї PIM в дві PSM та комунікаційний місток

Документація і супровід

Недоліки традиційного підходу

Більшість програмістів не любить писати документацію до програм, оскільки написання документації протягом розробки потребує часу і затримує завершення розробки.

Задача програміста розробляти такі програми, які б надалі відносно легко підтримувалися та дороблялися не тільки ними, але й іншими розробниками. Отже, написання документації одна з основних задач для програміста.

Існує підхід до цієї проблеми, що забезпечує генерацію документації прямо з коду програми. Таким чином, документація може бути завжди в актуальному стані, але цей підхід не вирішує проблеми високорівневої документації. Високорівнева документація, на відміну від документації по програмному коду, дає уявлення про систему на вищому рівні абстракції і мусить бути обов'язковою складовою складної програмної системи.

Переваги модельно-орієнтованого підходу

Розробляючи PIM, ми створюємо високорівневу модель нашої програми. Отже, PIM виконує функції високорівневої документації.

Перевагою є також те, що зроблені зміни в PIM, PSM, або в коді програми можуть автоматично відобразитись (трансформаційними інструментами) на всіх інших стадіях процесу розробки. Таким чином документація буде залишатись актуальною. Однак слід зазначити, що потреба заповнення документації певною інформацією вручну залишається.

4. Бажані характеристики інструментів перетворень

Ми мали змогу побачити наскільки важливу роль відіграє механізм трансформацій в модельно-орієнтованому підході. Слід зазначити, що трансформація моделей, на відміну від генерації коду по моделі, є досить новою сферою діяльності в програмній інженерії. Відтак, хоч і існують значні напрацювання в цій області, багато роботи щодо покращення характеристик інструментів трансформацій ще належить зробити.

Ми вже визначили трансформацію як процес генерації цільової моделі з початкової моделі. Процес описується трансформаційними визначеннями, які складаються з ряду трансформаційних правил і виконуються програмним інструментом для процесу перетворення. Існує ряд властивостей інструмента трансформації, над якими нині ведуться роботи задля покращення. Наведемо ці властивості за порядком їх значущості:

1. Можливість налаштування (Tunability) – означає, що хоча загальне правило було надане у трансформаційному визначенні, параметри цього правила можуть бути налаштовані; наприклад, перетворюючи рядок UML до рядку типу VARCHAR в SQL, можна вибрати, щоб довжина VARCHAR відрізнялася для кожного рядка в UML.

2. Можливість трасування (Traceability) – означає, що можна простежити зв'язок елемента в цільовій моделі з елементом (елементами) в початковій.

3. Можливість регенерації без втрат (Incremental consistency) – означає, що, коли інформація була додана до цільової моделі і відбувається її регенерація додаткова інформація залишається.

4. Можливість оберненого перетворення (Bidirectionality) – означає, що трансформація може бути застосована не тільки в напрямку від початкової моделі до цільової, але і в зворотному – від цільової до початкової.

Висновки

„Все є моделями” – основний принцип модельно-орієнтованої інженерії (MDE).

Моделі, що описані на мові з чітко визначеним синтаксисом та значенням придатної для обробки комп’ютером, набувають особливої ваги. Так, використання моделей як основного артефакту протягом всього життєвого циклу розробки програмних систем, дозволяє автоматизувати велику частину роботи, яка за традиційного підходу робиться вручну.

Найбільш відомим представником модельно-орієнтованого підходу є Model-Driven Architecture (MDA).

Одним з головних намірів MDA є відокремлення логіки предметної області від конкретної технології реалізації, дозволяючи цим двом частинам змінюватись незалежно одна від одної.

Використання MDA для розробки та інтеграції програмного забезпечення дозволяє зберегти інвестиції, зроблені в розробку бізнес-логіки навіть у разі зміни технологічних платформ.

Список літератури

1. *Jean-Marie Favre Megamodelling and Etymology. A story of Words: from MED to MDE via MODEL in five millenniums ADELE Team, LSR-IMAGUniversity of Grenoble, France http://www-adele.imag.fr/~jmfavre*

2. *Грибачев К.Г. Delphi и Model Driven Architecture. Разработка приложений баз даннях / К.Г. Грибачев.— СПб.: Питер, 2004. — 348 с.*

3. *Arend Rensink Subjects, Models, Languages, Transformations Department of Computer Science, University of Twente P.O.Box 217, 7500 AE, The Netherlands rensink@cs.utwente.nl*

4. *Douglas C. Schmidt Model-Driven Engineering Vanderbilt University*

5. *E. Seidewitz, "What Models Mean", IEEE Software, September, 2003.*

Стаття надійшла до редколегії 22.05.2013

Рецензент. д-р техн. наук, проф. В.М. Рудницький, Черкаський державний технологічний університет, Черкаси.