Компьютерная математика

Инструментальные средства информационных технологий

Розглянуто процес моделе-орієнтованої побудови інструментальних засобів комп'ютерної алгебри. Запропоновані нові, модифіковані інструментальні засоби, що мають покращений інтерфейс користувача та можливості автоматизованої генерації коду мовою Maude. Такі інструментальні засоби реалізовані і протестовані в середовищі авторської експериментальної програмної системи. Вони можуть бути використані для автоматизації розробки застосувань комп'ютерної алгебри та переписування термів на основі моделе-орієнтованого підходу.

УДК 681.3

О.П. ДАВИДОВ

МЕТОДИ РОЗРОБКИ ІНСТРУМЕНТАЛЬНИХ ЗАСОБІВ КОМП'ЮТЕРНОЇ АЛГЕБРИ НА ОСНОВІ МОДЕЛЕ-ОРІЄНТОВАНОГО ПІДХОДУ

Вступ. На даний момент популярність моделе-орієнтованого підходу до розробки програмних застосунків зростає. Але для модель-орієнтованої розробки не існує ні мовних стандартів, ні середовища розробки. Ця робота покликана вирішити цю проблему шляхом інтеграції мови Maude з середовищем розробки Visual Studio.

Нині в галузі інструментальних засобів комп'ютерної алгебри існують такі проблеми: відсутність можливості генерації та перевірки працездатності застосування в процесі розробки, відсутність поняття «проектів» як незалежних одиниць коду, що виконується та відсутність модулів візуалізації синтаксичних одиниць, які роблять розробку більш швидкою та автоматизованою.

Розробка модулів для середовища Visual Studio, які б вирішили ці проблеми модель-орієнтованої розробки є нетривіальною задачею, тому що в середі Visual Studio відсутні спеціальні модулі для моделе-орієнтованої розробки.

Створення таких засобів дозволить автоматизувати процес розробки застосунків комп'ютерної алгебри та переписування термів.

Мета цієї статті полягає у розробці інструментальних засобів, що мають можливості генерації коду мовою Maude та пришвидшують та автоматизують процес розробки застосувань комп'ютерної алгебри та переписування термів. Для тестування і експериментів розробити набір програмних компонентів – додатків до середовища розробки Visual Studio.

Наведемо модель системи на якій ґрунтується подальша моделе-орієнтована розробка. Доцільно буде також визначити такі поняття як: модель, функції моделі, абстракція, проекція.

Будемо використовувати наступне визначення з [1] модель – це абстракція системи, яка дозволяє робити прогнози або висновки. Якщо позначити систему як *S*, а її модель як *M*, то зв'язок між моделлю і системою будемо позначати так:

$$S \lhd M.$$

В цілому, $\lhd$ встановлює зв'язок багатьох-до-багатьох, так як одна модель може описувати декілька систем і одна система може бути описана декількома моделями. З точки зору математики, $\lhd$ – бінарне відношення. Підмножина цього відношення – це відношення представлення ρ, таке що

$$\rho(S, M) \to S \lhd M.$$

Водночас коли відношення «модель-це» $(\lhd)$ включає у себе будь-які випадково відповідні пари «система/модель», під відношенням ρ мається на увазі відношення, яке сприймає тільки такі пари, де модель спеціально призначена представляти відповідну систему. Ми, таким чином, можемо заявити що модель M_2 є моделлю для іншої моделі M_1, і обидві представляють одну унікальну систему.

$$S \lhd M_1 \wedge \rho(S, M_1),$$

$$M_1 \lhd M_2 \wedge \rho(S, M_2),$$

$$\rho(M_1, M_2).$$

Для того, щоб мати можливість обговорювати різні властивості моделі пізніше, опишемо функцію абстракції α, яка виробляє модель системи за заданою системою.

$$M = \alpha(S).$$

Передбачається, що для системи S вже є формальна інтерпретація. Наприклад, структурований набір «(S, r_s)», де S – елементи, а r_s – відношення між цими елементами є адекватним вибором [2]. Якщо система – реальна (наприклад, фізична) система яка досліджується, тоді S можна розглядати ніби то вона згенерована процесом «моделювання» самої системи. Водночас, як можна було б стверджувати, що процес «моделювання» втілює саму операцію, ми намагаємося дослідити, чи наш підхід працює без втрати загальності. Для наших цілей немає значення, чи є система, від якої ми абстрагуємося реальною, або вона вже є відображенням реальної системи. Будь-які філософські та епістемологічні питання процесу «моделювання», наприклад, як витягти структуру і властивості

з реальної системи з практично нескінченним числом властивостей і, можливо, неспостережуваної поведінки у формальне подання, не представляють інтересу для нас у цьому контексті, оскільки ми зосереджені на «моделі» в якості технічного терміну в модель-орієнтованій розробці, а не на дослідженні процесів створення відображень реальності.

В роботі [3] зазначено, що модель має мати три функції:

а) функція відображення – модель має базуватися на деякій системі;

б) функція редукції – модель відображає лише деякі властивості системи;

в) прагматична функція – модель має бути корисною, коли використовується замість системи щодо якоїсь мети.

Перші дві функції реалізуються відразу, якщо неформально говорити про модель як про «проекцію» системи, тому що це означає і те, що щось (система) є базою для даної моделі, і те, що деяка інформація втрачається під час проектування. Формально, це виражається рівнянням

$$\alpha = \tau \circ \alpha' \circ \pi. \quad '$$

Тут зазначено, що абстракція (α) складається з проекції (π), деякої ще більшої абстракції (α') і трансляції в інше уявлення (τ). З проекцією ми пов'язуємо будь-яку фільтрацію елементів – і таку, яка зменшує їх кількість, і таку, яка зменшує кількість інформації у елементів.

Для подальшого використання ми визначимо проміжний результат між проекцією та подальшою абстракцією як:

$$S_r = \pi(S).$$

Те, яка інформація залишиться після відображення залежить від кінцевої мети. Третя функція – можливість використання моделі, означає, що мають бути відповіді на питання «для кого ця модель була створена?» і «для якої мети ця модель була створена?». Модель – це інформація:

а) про щось (зміст, сенс);

б) створена кимось (відправник);

в) створена для когось (приймач);

г) створена з якоюсь метою (контекст використання).

Загальна мета створення моделі – використання моделі замість системи, будь-які відповіді, отримані з моделі, мають збігатися з відповідями системи, за умови що модель адекватна [4] / правильна [5]. Як правило, мотивуючими факторами для використання моделі є економія витрат і/або пришвидшення отримання відповідей від моделі, аніж від системи. Часто моделі бувають навіть неточними, або вони можуть помилятися у деяких випадках, але це не обов'язково значить, що модель неадекватна. Такі неточності, у кращому випадку, можуть взагалі не вплинути на властивості, які нас інтересують. Гірше коли ці неточності роблять моделі менш рентабельними, чи взагалі спотворюють модель.

Для подальшої розробки була побудована модель системи, яка зображена на рис. 1. Як можна бачити, все почалося з ідеї зробити роботу з Maude зручнішою, Опис цієї ідеї сформував своєрідне «Технічне завдання» того, що треба зробити, а саме:

а) покращити середовище редагування;

б) покращити модульність проектів;

в) зробити написання систем за допомогою Maude схожим на написання програм мовами програмування.

Ця задача могла бути реалізована двома шляхами: окремим середовищем розробки, чи набором розширень для вже існуючого середовища розробки. Після аналізу цих рішень, було прийнято рішення реалізовувати ідею за допомогою набору розширень, тому що цей підхід кращий за деякими характеристиками:

а) набір розширень займає значно менше дискового простору;

б) інсталятор для набору розширень генерується автоматично;

в) налаштування середовища розробки зберігаються.

Була сформована узагальнена модель типів (мета модель). У мета моделі розширення розглядається як «інтегратор» інформації про деяку мову програмування, у деяке середовище розробки. Одне розширення включає у себе декілька наборів інформації і цих розширень може бути декілька.

Модель системи має вигляд двох розширень: шаблон проекту і модуль візуалізації синтаксичних одиниць. Шаблон проекту складається з інформації, про те як треба будувати файли типу .maude, приклад інтерпретованого файлу, який буде інтегруватися в пустий проект за замовчуванням та інтерпретатор мови .maude. Модуль візуалізації синтаксичних одиниць інтегрує у середовище розробки інформацію, про те які слова як треба візуалізувати.

Модель системи є одночасно моделлю маркерів для класів і файлів, визначених у програмному продукті і моделлю типів для програмного продукту у момент його виконання. Модель маркерів і типів є підмножиною мета моделі, тому мета модель є також моделлю типів для програми у момент її виконання.

Класи і файли – це програмна реалізація моделі, вона включає у себе інтерпретатор maude.exe, класи інструменту для побудови (наприклад, BuildTask), файл граматики, який потрібен для коректної візуалізації синтаксичних одиниць (MaudeLexer.g4), пустий проект, який використовується як шаблон проектів і т. і. Докладніше про кожний модуль можна прочитати у відповідному розділі.

Виконання – це класи і файли системи, у момент виконання. Вони мають у собі об'єкти класів, які описані у класах і файлах, наприклад, помилка побудови, чи тип токена, який відповідає за ключове слово.

Створення спеціального інструменту для виконання застосувань, написаних на мові Maude у середовищі Visual Studio.

Очевидно, що інструментом для виконання інтерпретованої мови Maude має бути сам інтерпретатор, тому основна задача полягає у тому, щоб замість MSBuild.exe при побудові проекту в Visual Studio викликався Maude.exe.

Ця задача ускладнюється тим, що було вирішено не використовувати ізольовану оболонку для реалізації середовища розробки, тому було вирішено постачати інтерпретатор Maude.exe разом с інструментом для виконання у проекті, тобто у шаблоні проекту. Цей підхід має одну суттєву перевагу: такий проект – повністю самостійний, він постачається з інструментом для виконання і з інтерпретатором, таким чином він може бути виконаний без встановлення додатків на Visual Studio. Але також має і недолік: проект починає займати більше місця на диску, хоча інтерпретатор Maude не дуже великий (3,4 Мб).

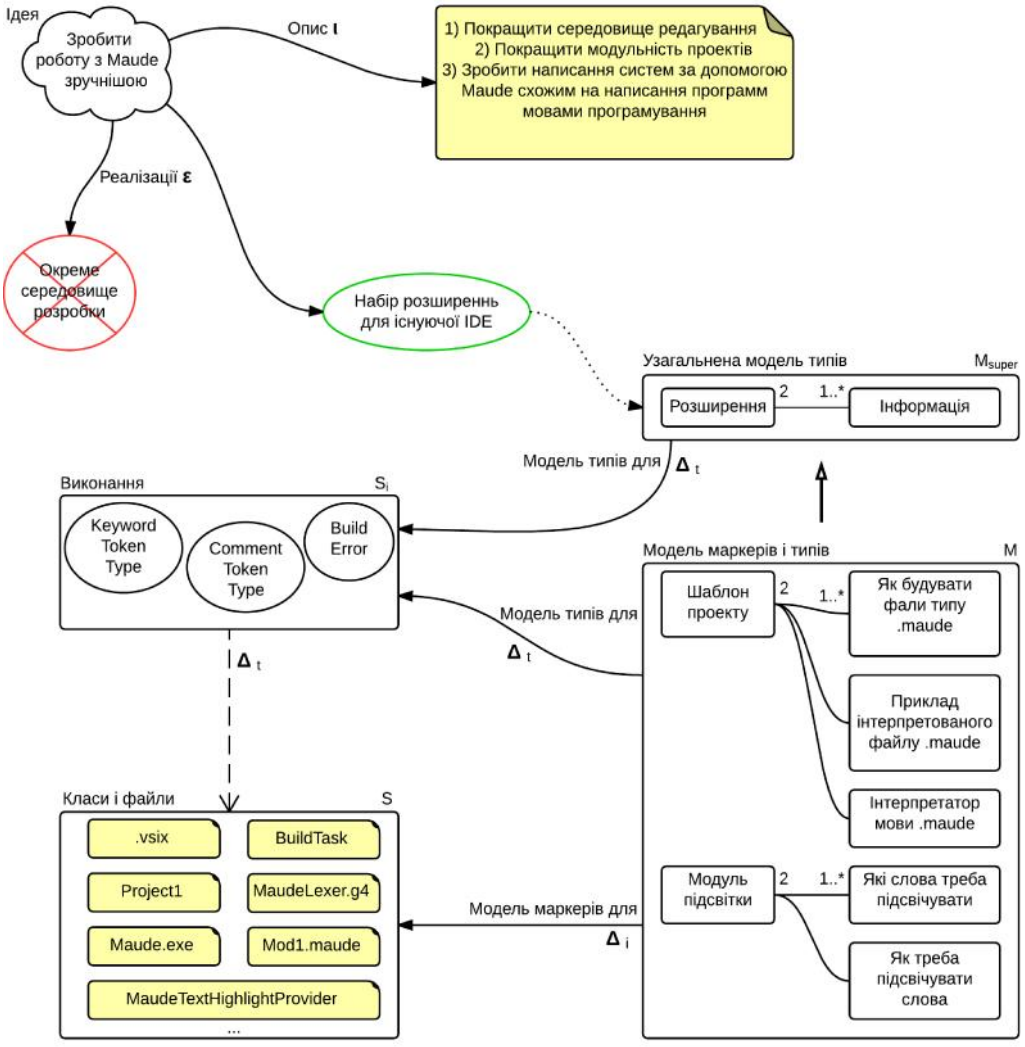

РИС. 1. Модель розробленої системи

Для реалізації інструменту для виконання потрібно було зробити клас, який би успадковувався від класу Task. Після цього потрібно перезавантажити метод базового класу Execute(). Цей метод вертає логічне значення true чи false. Це значення і є результатом виконання: була вона успішна чи ні. Цікаво те, що Visual Studio більше нічого не знає про те, що відбувається всередині цього виконання. Усередині цього методу викликається створення нового процесу Maude з необхідними параметрами, а саме:

а) ім'я файлу який буде будуватися. Це – основний параметр, який передається Maude.exe, щоб інтерпретатор знав, з якого файлу починати виконання проекту;

б) -no-prelude – це параметр, який ініціює запуск Maude.exe без стандартної прелюдії;

в) -no-banner – це параметр який ініціює запуск Maude.exe без стандартного банеру;

г) -no-advise – це параметр, який вимикає допоміжні повідомлення після старту Maude.exe;

д) -xml-log=log.xml – параметр, який перенаправляє логи до даного xml файлу.

Перераховані параметри запускають інтерпретацію .maude файлу інтерпретатором Maude.exe у «стерильному» середовищі, тобто Maude.exe не видає ніяких повідомлень які часто тільки заважають нормальній роботі з ним.

Зі сторони файлу проекту виклик цього спеціального завдання виглядає так:

$$< UsingTask\ TaskName="MyTask"/>. \quad (1)$$

$$<Target\ name="RunMyTask">. \quad (2)$$

$$< MyTaskFileName = "Mod1.maude" / >,$$

$$</Target>. \quad (3)$$

Тут у формулі (1) задається ім'я завдання, яке буде виконане замість MSBuild.exe. У формулі (2) задається ім'я цілі. У формулі (3) задається ім'я файлу, який буде інтерпретований.

Передбачається, що цей інструмент для виконання використовуватися таким чином: у файлі задаються функції, операції, правила переписування тощо. Потім проект будується і вже у консолі перевіряється задана модель чи система, наприклад, пишуться редукції та перевіряється що вони редукуються правильно. Це – передбачене, але не єдине використання інструменту для побудови. Є можливість задати деякі тривіальні редукції у файлі, чи визначити деякі функції у консольному режимі.

Створення шаблону проекту. Шаблон проекту може бути використаний для створення розширень або для упакування вже існуючого розширення для подальшого розгортання.

Для створення шаблону проекту потрібно [6]:

а) створити проект, з якого ви хочете створити шаблон;

б) експортувати проект у форматі .zip;

в) створити пустий шаблон проекту;

г) налаштувати шаблон проекту так, щоб він брав проект з .zip файлу;

д) зібрати проект;

е) запустити проект і встановити новий шаблон проекту на вашу поточну Visual Studio;

ж) перезапустити Visual Studio.

Після цих маніпуляцій у діалозі File->New->Project Visual Studio вам стане доступний новий шаблон проекту. Було вирішено створити два шаблони проектів: пустий проект та проект з початковим файлом.

Архітектура пустого шаблону проекту показана на рис. 2.

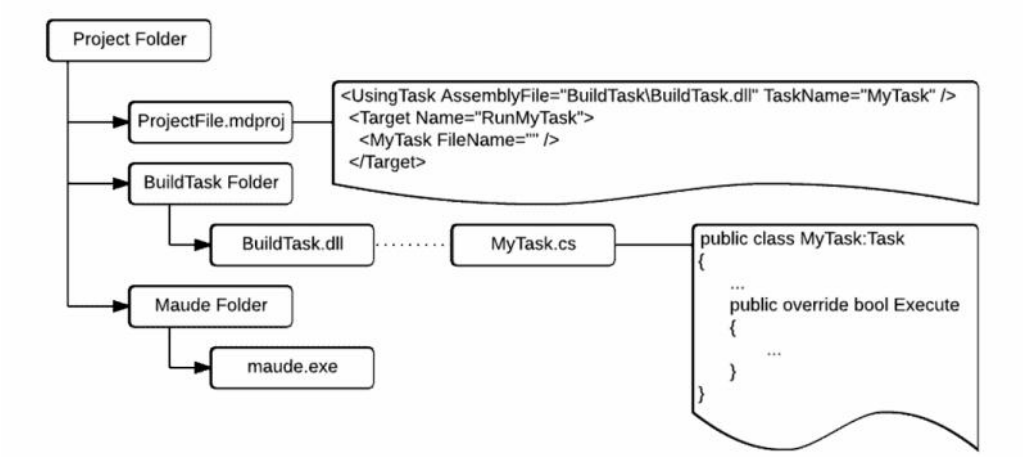

РИС. 2. Архітектура пустого шаблону проекту

Project Folder – папка проекту;

Project File – файл проекту;

Build Task Folder – папка, де знаходиться спеціальний інструмент для побудови;

BuildTask.dll – бібліотека де зберігаються інструкції, необхідні для побудови файлу проекту;

MyTask.cs – файл, який «прихований» у скомпільованій BuildTask.dll, в якому міститься функція, яка виконується при побудові файлу проекту;

Maude Folder – папка, де знаходиться інтерпретатор Maude;

Maude.exe – інтерпретатор Maude.

Архітектура шаблону проекту з початковим файлом показана на рис. 3.

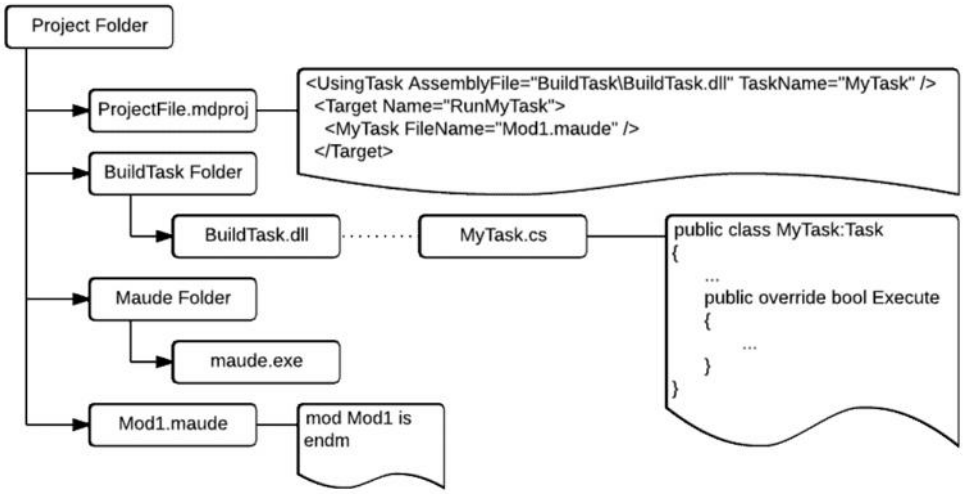

РИС. 3. Архітектура шаблону проекту з початковим файлом

Mod1.maude – початковий файл з мінімальним інтерпретованим текстом.

Шаблон проекту з початковим файлом є більш зручним, тому що у випадку пустого шаблону початковий файл, необхідний для побудови не буде заданий, це вимусить користувача самостійно у текстовому режимі редагувати файл проекту. У цьому файлі є підказка як редагувати і вносити початковий файл проекту, але це все одно не зручно.

Розробка модулю візуалізації синтаксичних одиниць. Модуль візуалізації синтаксичних одиниць можна розбити на два незалежних модуля:

а) модуль, який призначений для візуалізації коментарів, ключових слів, ідентифікаторів, класів і т. і., для однозначності цей модуль будемо називати класифікатором.

б) модуль, який призначений для візуалізації слова (наприклад, назви змінної) на якому стоїть курсор миші у всьому тексті.

Уся розробка починається з оголошення типу контенту. Тут оголошується те, файли якого типу (з яким розширенням) будуть оброблятися новим, модифікованим редактором. Ця інформація зберігається у реєстрі.

Після цього для класифікатора задається набір форматів і типів, наприклад, формат ключового слова, формат коментаря і тип ключового слова, тип коментаря. У типі задається по суті тільки назва типу, наприклад, «maude.comment», який потім використовується у форматі коментаря для надання йому будь-яких візуальних ефектів, наприклад зміни фону тексту. Після цього реалізується класифікатор, основна робота якого – розбити вхідний потік символів на

незалежні частини. Після цього такі незалежні частини, подаються на вхід лексеру, в якому у спеціальному форматі задані всі конструкції та ключові слова мови Maude, наприклад:

$$LINE\_COMMENT\_STARS : '***' \sim [\backslash r \backslash n] *$$

так задається коментар однієї строки коду у вигляді регулярного виразу: спочатку йдуть три зірочки ***, а потім будь-який символ, окрім символів кінця строки будь-яке число раз.

Лексер – це обгортка над граматикою мови. Для імплементації лексеру було використано Antlr.

Після цього дані, розбиті за такими категоріями, потрапляють у своєрідний асемблер, який з'єднує деякі категорії, наприклад, у граматиці зазначено два види коментарів: коментарі, які починаються з трьох зірочок, і коментарі, які починаються з трьох дефісів, у асемблері вони всі приводяться до типу і формату коментаря, який був зазначений вище.

Для модуля візуалізації однакових слів, треба також зазначити формат, тобто ті візуальні ефекти, які будуть з'являтися для усіх слів, однакових зі словом, на якому в даний момент часу стоїть курсор. Окрім цього треба просто налагодити коректний імпорт слова, на якому в даний момент знаходиться курсор і пошук такого слова у файлі.

Висновок. У даній статті розглянуто процес моделе-орієнтованої побудови інструментальних засобів комп'ютерної алгебри. Проблема відсутності середовища редагування для модель-орієнтованої розробки, була вирішена створенням набору розширень до середовища Visual Studio. Розробка проводилася на основі моделі системи. Модель представляє собою два розширення: шаблон проекту та модуль візуалізації синтаксичних одиниць. Шаблон проекту постачає у середовище розробки інформацію про те, як потрібно будувати файли з розширенням .maude, інтерпретатор мови maude, та приклад пустого файлу, який буде за замовчуванням включено в проект при його створенні. Модуль візуалізації синтаксичних одиниць, у свою чергу, постачає в середовище розробки інформацію про те які символи як треба візуалізувати. Подальші дослідження можливі в наступних областях:

а) реалізація інтелектуального аналізу тексту для викривання помилок до фактичної інтерпретації файлу;

б) реалізація компоненту, який би в кожен момент часу мав список ключових слів та термів, які доступні для вставки у текст (аналог IntelliSense);

в) реалізація механізму, який би компонував проект типу maude у деякий виконуваний файл (.exe файл), у якому були б всі необхідні файли та інтерпретатор.

А.П. Давидов

МЕТОДЫ РАЗРАБОТКИ ИНСТРУМЕНТАЛЬНЫХ СРЕДСТВ
КОМПЬЮТЕРНОЙ АЛГЕБРЫ НА ОСНОВЕ МОДЕЛЬ-ОРИЕНТИРОВАННОГО ПОДХОДА

Рассмотрен процесс модель-ориентированной разработки инструментальных средств компьютерной алгебры. Предложены новые, модифицированные инструментальные средства, имеющие улучшенный интерфейс пользователя и возможности автоматизированной генерации кода на языке Maude. Такие инструментальные средства реализованы и протестированы в среде авторской экспериментальной программной системы. Они могут быть использованы для автоматизации разработки приложений компьютерной алгебры и переписывания термов на основе модель-ориентированного подхода.

O.P. Davydov

THE METHODS OF COMPUTER ALGEBRA TOOLS DEVELOPMENT BASED ON A MODEL-ORIENTED APPROACH

The model of tools to automate and accelerate the development of the computer algebra and term rewriting applications is described. A description of software components that implements this model as a set of Visual Studio add-ins, i.e., the building tool, the project pattern, and the code highlight for the Maude language, is given. Such tools are implemented and tested in the author's experimental software.

1. *Favre J.* Towards a basic theory to model driven engineering. In Third Workshop in Software Model Engineering. – 2004. – P. 262 – 271.
2. *Kaschek R.* A little theory of abstraction // In Bernhard Rumpe and Wolfgang Hesse. – 2004. – Vol. 1. – 153 p.
3. *Stachowiak H.* Allgemeine Modelltheorie. – Wien: Springer-Verlag, 1973. – 325 p.
4. *B´ezivin J., Gerb´e O.* Towards a precise definition of the OMG/MDA framework // In Proceedings of the 16th International Conference on Automated Software Engineering Coronado Island. – 2001. – P. 273 – 280.
5. *Seidewitz E.* What models mean // IEEE Software. – 2003. – Vol. 20, N 5. – P. 26 – 32.
6. *Microsoft Developer Network.* How to: Create Project Templates. – https://msdn.microsoft.com/en-us/library/xkh1wxd8.aspx

Одержано 29.10.2015

Про автора:

Давидов Олександр Павлович,
аспірант
Інституту кібернетики імені В.М. Глушкова НАН України.
E-mail: duff1994@gmail.com