Использование логического программирования для решения задач автоматизации порождения программного кода

Е.А.Черкашин, В.В.Парамонов, Р.К.Федоров,
И.Н.Терёхин, Д.В.Анненков, Е.И.Поздняк
ИДСТУ СО РАН, ИМЭИ ИГУ, НИ ИрГТУ
e-mails: {eugeneai, slv, fedoprov}icc.ru

Рассматривается подход к автоматизации синтеза каркаса информационной системы ранга предприятия, реализующий MDA–архитектуру. Реализация базируется на комбинировании логических и императивных языков программирования. Процедуры преобразования представлены в виде правил и шаблонов исходного кода. Рассмотрены дальнейшее направления совершенствования реализаций трансформации.

Введение

Развитием идеи визуального проектирования информационных систем (ИС) выступает подход, называемый MDA (Model Driven Architecture) [1, 2] и предложенной в 2001 г. консорциумом OMG (Object Management Group). MDA преследует целью отделить спецификацию функций системы от способа реализации этих функций в различных платформах, при этом обеспечивается спецификация системы независимо от платформы, на которой она будет функционировать, спецификация собственно платформы, преобразование спецификации системы в конкретную систему, функционирующую на конкретной платформе. Три основные цели, преследуемые MDA, — это переносимость, открытость к взаимодействию с другими системами и повторное использование.

Ключевыми концепциями MDA являются CIM, PIM, PSM, PDM. *Вычислительнонезависимая модель* (Computation Independent Model, CIM) отражает внешние требования к ИС — его интерфейсы. CIM скрывает внутренние структурные элементы и таким образом может быть использована для задания спецификаций и проверки требований. *Платформонезависимая модель* (Platform Independent Model, PIM) — это модель ИС, отражающая большинство структурных и некоторые семантические аспекты ИС, но данная модель не содержит каких–либо точных сведений о реализации структур на выбранной программной архитектуре. Диаграмма классов UML дополненная с помощью некоторых тэговых значений и дополнительных стереотипов является примером PIM. Данные дополнения позволяют учитывать нюансы при реализации структур программной системы. *Платформозависимая модель* (Platform Specific Model, PSM) — это модель, которая может быть реализована непосредственно в виде исходного кода подсистем разрабатываемой системы, например, физическая структура базы данных, которая напрямую (алгоритмическим или с помощью шаблонов исходного кода) транслируется в DDL запросы SQL.

MDA — это методология для разработки ИС на основе частичного порождения исходного кода из визуальных моделей. MDA можно рассматривать как подход к порож-

дающему программированию [3, 4], при этом конфигурация в процессе трансформации PIM. Основным отличием от CASE–систем является то, что процедуры генерации кода не фиксированы, а могут быть приспособлены к требованиям проекта, используемым разработчиками методикам и инструментальным средствам реализации ИС. В области ИС методология MDA применима и для автоматизации разработки, основанной на компонентной архитектуре.

Трансформация PIM в PSM выполняется под управлением *модели, описывающей платформу* (Platform Description Model, PDM). PDM содержит информацию и алгоритмы анализа структур PIM, а также порождение соответствующих структур PSM. Иногда PSM представляется как специализированный вариант PIM. Специальные варианты трансформации PIM задаются при помощи тэговых значений и стереотипов.

Целью данного исследования является создание технологии трансформации MDA, основанной на использовании логического программирования и способной к интегрированию с существующими методиками и инструментальными средствами разработки ИС.

1. Реализация трансформаций

Существует множество подходов к реализации трансформаций: алгоритмический подход, где все процедуры трансформации реализуются на императивном языке программирования; XSLT–преобразование позволяет представить трансформацию в виде продукционных правил; теория графов и преобразования графов; использование предметно–ориентированных языков программирования [5]. В данной работе использован логический подход для задания трансформаций как набора правил в стиле типовых конфигураций [6] (ТК); процесс трансформации управляется при помощи сценариев трансформации.

ТК представляются как программные объекты, включающие исходный код языков Prolog и Python. Секция “если” ТК — это правило (программа) Prolog, помещенное в так называемую `__doc__`–строку метода экземпляра Python. Тела методов являются исполнительными (“делай”) частями ТК. Формальные параметры, передаваемые методам, — это результаты запросов к правилу Prolog “если”. Экземпляры объектов Python являются модулями, программными объектами, включающими множество шаблонов и алгоритмов, которые трансформируют заданную часть PIM в PSM. Рассмотрим пример ТК и модуля, которые генерируют SQL–запрос на создание реляционной базы данных для хранения экземпляров объектов ИС.

```
class RulesMixing: # Набор ТК модуля
    def rule_primitive_class(self, cls, oid, oidType): # Распознавание способа
       # идентификации экземпляров объектов в реляционной базы данных (БД)
       """ # здесь начинается __doc__--строка
       primitive_class(Cls, OIDAttr, OidType):-
           element(Cls, 'Class'),
           \+stereotype(Cls, 'abstract'),
           stereotype(Cls, 'OODB'),
           \+internal_only(Cls),
       . . . . . .
       """ # здесь __doc__--строка заканчивается
       self.BASE_CLASS = cls             # Обнаружение корня иерархии классов
       self.BASE_CLASS_NAME = self.getName(cls)
```

```
        self.OID_NAME= self.getName(oid)  # Аттрибут-идентификатор объекта
        self.OID_TYPE = self.coerceAttrType(oid, oidType)
                                          # Подобрать тип идентификатора
        return self.BASE_CLASS_NAME, cls, oid, self.OID_NAME
. . . . . .
class SQLTranslator(Translator, RulesMixing): # Модуль трансформации,
   def genClass(self, cls):                   # порождает SQL-скрипт
        . . . . . .# структуры таблицы класса cls
        attribs = self.genSchema(cls)         # порождение атрибутов таблицы
        if not self.isEmpty(attribs):
            answer.append("CREATE TABLE %s (" % name)
            answer.append(attribs) # атрибуты таблицы
            answer.append(")%s;" % self.getTableType(cls))
        else:
            print "The class has no attributes."
        self.addFact("oodb_table('%s', '%s')" %
            (cls.getId(), name)) # сохранение факта о связи класса и таблицы
        self.generated.append(cls)
        return answer # возвратить сгенерированный код
```

Механизмом вывода на ТК для каждого ответа `Cls` запроса `persistent_class(Cls)`, т.е. распознанного класса, запускается метод `genClass`. Структура базового класса, распознанного правилом `rule_primitive_class`, отражает способ представления идентификаторов экземпляров объектов для всех таблиц в реляционной базе данных.

Созданная система модулей представляет процесс трансформации PIM в PSM как многоэтапный процесс последовательной конкретизации PIM (рис. 1). Исходная модель PIM загружается из XMI–файла (XML Metadata Interchange), который является разновидностью XML, в структуру DOM2. Полученное DOM2–дерево транслируется в факты Prolog. Выражения OCL (Object Constraint Language) извлекаются из PIM и представляются в виде деревьев.

В начале трансформации производятся общие рассуждения относительно структуры объектов, распознаются их основные свойства. Результаты этих рассуждений используются другими модулями для определения вариантов реализации порождаемых программных объектов. Каждый модуль доопределяет существующие структуры PIM дополнительными фактами о свойствах уже имеющихся структур, а также создает новые объекты и отношения. Например, модуль трансформации SQL объединяет унаследованные от абстрактных предков списки атрибутов в класс и порождает соответствующую таблицу. Процесс в общем виде задается сценарием, представляющим собой список листовых вершин сети модулей трансформации, которые должны быть выполнены. После исполнения сценария множество всех фактов, находящихся в рабочей памяти определяет PSM. Исходный код генерируется на основе полученного PSM. Модуль генератора выполняет запрос и заполняет шаблоны исходного кода результатами этого запроса.

Результаты трансформации и порождения кода собираются в библиотеки объектов. Объекты из библиотек используются в программировании бизнес–логики ИС через непосредственные вызовы и наследование. Это позволяет уменьшить вероятность правки непосредственно сгенерированного кода.

Кроме информации о структуре модели PIM содержит семантические данные. Семан-

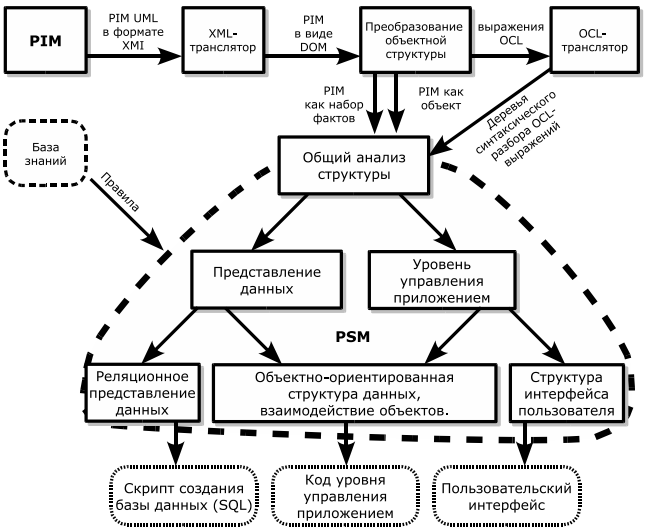

Рис. 1. Архитектура системы трансформации

тика используется для уточнения варианта процедуры трансформации, используемой для конкретного элемента. Семантика в UML задается при помощи *стереотипов*, *теговых значений* и OCL–ограничений, используемых для формального определения логических ограничений, инвариантов, пред– и постусловий корректности методов.

Рассмотрим пример. Предположим, что в диаграмме классов UML существует класс, который содержит в себе поле *name* . Если класс помечен стереотипом «Reference Book», тогда все отношения типа “многие–к–одному” к данному классу могут быть проинтерпретированы в реляционной базе данных как ссылки на соответствующую таблицу (как на справочник) при помощи поля *ID*. Встретив такой стереотип и это отношение, модуль генератора интерфейса пользователя может построить виджет и контроллер (в смысле парадигмы Model-View-Controller), связывающий логически таблицы и справочник. Таким образом, по сравнению с CASE–системами процедуры порождения программного кода различных подсистем в MDA логически зависимы.

Для того, чтобы адаптировать механизм трансформации и его базу знаний к своим технологиям и инструментальным средствам разработчику требуется импортировать Python–модули, произвести модификацию набора ТК и процедур порождения исходного кода через наследование существующих классов модулей. Например, в одном из приложений системы трансформации при помощи такой адаптации создана реализация БД в системе MySQL, где таблицам были заданы некоторые параметры, специфичные MySQL–серверу.

2. Современные направления разработки

Дальнейшее развитие предлагаемого метода нацелено на повышение производительности и выразительности средств системы трансформаций, а также повышение её надежности. Основной проблемой является то, что очень трудно обеспечить одновременно эффективную и полную интеграцию программ на Python и Prolog: обе системы обладают собственными средствами управления памятью, но при этом различными методи-

ками управления. Было принято решение переключится на использование логического языка LogTalk [7]. LogTalk является объектно–ориентированным языком логического программирования, который функционирует при поддержке различных компиляторов Prolog. Являясь мультипарадигмальным объектно-ориентированным языком программирования, LogTalk поддерживает наследование, причем, как основанное на прототипах, так и на классах, интерфейсы, компонентное программирование, многопоточное программирование и т.п.

В новой реализации [8] трансформации будет реализована та же базовая архитектура, что и ранее, но ставиться новая цель — поддержка трансформаций в обе стороны: из абстрактной PIM в PSM и, далее, в программный код, и в обратном направлении, от исходного кода в абстрактные модели. Это должно позволить разработчикам:

- Модифицировать сгенерированный код и обеспечить учет изменений в PIM;
- Разрабатывать ИС на различных уровнях абстракции представления;
- Накапливать библиотеки комплексов моделей и их реализаций.

Результаты исследования предполагается реализовать в виде среды разработки ИС, которая объединит в себя визуальное средство редактирования UML и среду для редактирования исходного кода. Методика трансформации должна базироваться на современном видении процесса как распространении изменений [9, 10]: произведенные изменения должны распознаваться средой и передаваться на соответствующие уровни (модели).

3. Распространение изменений

Распространение изменений от абстрактных моделей к конкретным, упомянутый в [9], с позиции методологии MDA может быть проинтерпретирован как методика трансформации. Механизм трансформации сравнивает две версии PIM, распознает различия и соответственно корректирует PSM и исходный код. Такой подход использует специальные ссылки между объектами в PIM и сгенерированным объектами в PSM. Когда объект в PIM меняется или удаляется, то его сгенерированный образ в PSM отслеживается по ссылке.

Мы предлагаем расширить эту методику, обеспечив в некоторой степени распространение изменений в обоих направлениях. Это даст следующие дополнительные преимущества:

- Фиксация истории разработки как комплексов абстрактных моделей и соотвествующих фрагментов исходного кода (по аналогии с системами контроля версий).
- Позволить проектировщикам передавать комплексы моделей между проектами, как это частично реализовано в системах распределенного контроля версий.

В качестве теоретического базиса предлагается применить теорию систем комплексов (конфигураций), предположив, что процесс разработки программного обеспечения является естественным. В [10] показано, что изменения моделей (конфигураций) представимы при помощи тех же языковых средств, что и сами комплексы моделей. Аналогично показано,что модули трансформации применимы для трансформации как модели, так и ее изменений. Для подтверждения теоретических результатов проведено исследование существующих технологий, ориентированных на сравнение программных компонент и структур данных, а также представления результатов сравнения. Обнаружены примеры существующих инструментальных технологий, подтверждающих теоретические результаты.

Заключение

В работе рассмотрена методика реализации процедур трансформации моделей информационных систем (ИС) в рамках подхода к проектированию ИС на основе архитектуры, управляемой моделированием (Model Driven Architecture, MDA). ИС синтезируется по формальному описанию, представленному в UML, под управлением формального представления свойств программной платформы, на которой эта ИС реализуется. Процедуры преобразования построены с использованием методов искусственного интеллекта, а именно, систем, основанных на формализованных знаниях. Формальное описание модели ИС обрабатывается набором модулей анализа и порождения программного кода, в результате чего, порождается программный код базовых подсистем ИС (SQL–скрипт создания базы данных, объекты приложения и вариант пользовательского интерфейса). Рассмотрены дальнейшие направления развития средств трансформации и методики MDA.

Список литературы

[1] OMG Model Driven Architecture. URL: http://www.omg.org/mda/ (дата обращения - 24.09.2012)

[2] Frankel D. Model driven architecture : applying MDA to enterprise computing. – New York: Wiley, 2003.

[3] Горбунов–Посадов М.М. Как растет программа. – Препринты ИПМ им.М.В.Келдыша. – 2000. – N 50 – 16 с.

[4] Чернецки К., Айзенекер У. Порождающее программирование: методы, инструменты, применение / Пер. с англ. – СПб: Питер, 2005.

[5] Кузнецов М.Б. Трансформация UML-моделей и ее использование в технологии MDA // Программирование, Вып. 33, 2007, с. 44–53.

[6] Братко И. Программирование на языке Пролог для искусственного интеллекта. Пер. с англ. – М.: Мир, 1990. - 560 с., ил.

[7] Язык программирования Logtalk. URL: http://logtalk.org/ (дата обращения – 29.09.2012)

[8] Черкашин Е.А. Ипатов С.А. Логический подход к обработке UML–моделей информационных систем // Современные технологии. Системный анализ. Моделирование. 2009. N 3. с. 91–97.

[9] Alanen M., Porres I., "Change Propagation in a Model-Driven Development Tool,"доклад представлен на WISME(Workshop in Software Model Engineering) в 2004, 2004

[10] Cherkashin E.A., Paramonov V.V., et al, “Model Driven Architecture is a Complex System”, E-Society Journal Research and Applications. Volume 2, Number 2, 2011, pp. 15-23.