В приложении ЭнергоWatcher, созданном с использованием технологии METAS Control, основанной на интерпретации взаимосвязанных метаданных, пользователь разрабатывает собственную панель индикаторов, описываемую метаданными презентационного уровня. Пользователь сам выбирает, какие показатели и в какой конфигурации он хочет наблюдать, используя средства создания визуального интерфейса настраиваемой панели, а также предметно-ориентированные языки для задания ограничений, правил обработки данных.

Заключение

Представленная технология позволяет создавать гибкие масштабируемые распределенные приложения для мониторинга параметров энергопотребления, которые могут работать параллельно с программами, поставляемыми производителями оборудования, взаимодействуя с ними и дополняя их. Пользователи имеют возможность настраивать систему в соответствии со своими потребностями в ходе её эксплуатации.

Проект выполняется при поддержке РФФИ (проект № 12-07-00763-а).

Библиографический список

1. *Воронов В.А., Калашников Е.А., Лядова Л.Н.* Технология создания системы мониторинга энергопотребления // Труды Конгресса по интеллектуальным системам и информационным технологиям «AIS-IT'10» Научное издание в 4-х томах. Т. 1. – М.: Физматлит, 2010. С. 493-500.
2. *Лядова Л.Н.* Технология создания динамически адаптируемых информационных систем // Труды междунар. науч.-техн. конф. «Интеллектуальные системы» (AIS'07). Т. 2. – М.: Физматлит, 2007.
3. *Рыжков С.А., Лядова Л.Н.* CASE-технология METAS // Математика программных систем: Межвузовский сборник научных трудов / Перм. ун-т – Пермь, 2003, с. 4-18.
4. *Kalashnikov E.* EnergoWatcher – The Platform for Creating Adaptable Energy Monitoring Systems // Proceedings of the 6th Spring/Summer Young Researchers' Colloquium on Software Engineering (SYRCoSE 2012), May 30-31, 2012 – Perm, Russia. P.220-223.

МЕТОДЫ И СРЕДСТВА РАЗРАБОТКИ ЯЗЫКОВ ОПИСАНИЯ ТРАНСФОРМАЦИЙ ПРЕДМЕТНО-ОРИЕНТИРОВАННЫХ ЯЗЫКОВ

Л.Н. Лядова, А.П. Серый, А.О. Сухов
Пермский филиал НИУ ВШЭ,
г. Пермь

Введение. На различных этапах жизненного цикла информационных систем (ИС) используются модели, описывающие архитектуру и поведение системы, различные её характеристики. Современные CASE-средства позволяют строить различные типы моделей, представляющих систему, её предметную область с различных точек зрения и разной степенью детализации. Для их создания применяются языки моделирования: языки описания моделей предметных областей, схем баз данных, бизнес-процессов и др. При этом используются как текстовые, так и визуальные языки. Графические нотации обладают высокой выразительностью, позволяют строить наглядные, легко воспринимаемые различными категориями разработчиков и пользователей модели.

Существует множество визуальных языков, применяемых при создании программного обеспечения (UML, IDEF, DFD и др.). Чаще всего при разработке программного обеспечения используются универсальные языки моделирования, такие как язык UML. Однако у всех универсальных языков моделирования есть ряд недостатков. Во-первых, они оперируют небольшим набором элементов («класс», «ассоциация», «агрегация» и т.п.), но в реальных системах существует множество понятий, которые не

всегда можно адекватно отобразить с помощью этих понятий. Во-вторых, при использовании универсальных языков возникают проблемы, связанные со сложностью для восприятия моделей, созданных с помощью этих языков. Это снижает выигрыш от применения моделирования. Третьим недостатком универсальных языков является то, что для их использования необходимо знать синтаксис и семантику этих языков, таким образом, снижаются возможности привлечения к работе экспертов в предметной области, аналитиков.

Эти недостатки привели к появлению нового подхода к моделированию. Этот подход предполагает разработку специфических (предметно-ориентированных) языков (DSL, Domain Specific Language) моделирования для каждой конкретной предметной области. Такие языки могут разрабатываться как модели соответствующих предметных областей (метамодели) на некоем универсальном языке моделирования (метаязыке). При разработке метамодели нужно описать используемые в соответствующей области понятия и связи между ними. Логику работы с этими объектами нужно описывать уже в рамках разработки модели. Причём использование в качестве языка моделирования предметно-ориентированного языка, основанного на использовании понятий предметной области, позволяет без проблем привлечь к этому процессу экспертов.

Применение метамоделирования разбивает процесс разработки программного обеспечения на следующие этапы [2, 3]:

1) разработка языка описания предметной области (метамодели);
2) построение модели на созданном языке моделирования;
3) генерация кода по построенной модели или её интерпретация (интерпретация обеспечивает максимальную гибкость, адаптируемость системы к меняющимся условиям эксплуатации и потребностям пользователей и бизнес-процессов на основе оперативного изменения моделей).

Специальные программные средства, позволяющие автоматизировать эти этапы, снижают трудоёмкость работы по созданию и сопровождению системы. В таких средствах помимо (мета)модели строятся специальные правила, управляющие ходом преобразования (трансформации) модели при переходе от одного этапа к другому (от одного уровня моделей предметной области, к другому уровню). Такие трансформации носят название вертикальных [7, 8]. При использовании визуальных средств моделирования эти трансформации сводятся к трансформациям графов.

Применение предметно-ориентированных языков позволяет, как было сказано, упростить процесс разработки. Однако сложные системы, автоматизирующие деятельность в нескольких предметных областях, часто состоят из компонентов, чью структуру и поведение невозможно описать с помощью одного языка моделирования, так как у этих компонентов принципиально разная природа, определяемая спецификой этих предметных областей. В этом случае для построения моделей таких систем нужно использовать несколько формализмов и описывать каждую часть системы на том языке, который разработан специально для этой предметной области. Эти формализмы могут разрабатываться различными специалистами с применением разных метаязыков. Для представления системы в целом при использовании многопарадигматического моделирования все модели должны конвертироваться в общее описание, основанное на общем формализме. При переходе от одного формализма к другому (от модели одной предметной области к другой, связанной с ней) выполняются горизонтальные трансформации [7, 10] и для них, как и для DSL, необходимо задать правила. Причём для описания вертикальных и горизонтальных трансформаций можно использовать как разные методы и средства, так и общие. Построение трансформаций, удовлетворяющих заданным требованиям, само по себе является нетривиальным процессом, который может занимать довольно длительное время. Кроме того, для проверки правильности описанных трансформаций нужно обеспечить возможность их отладки.

Важной частью любой модели являются ограничения, которые налагаются на эту модель. При трансформации моделей необходимо учитывать эти ограничения и транслировать их вместе с моделью.

Таким образом, мы приходим к задаче создания средств описания трансформаций, которые должны удовлетворять следующим требованиям:

1) быть достаточно наглядным и простым в использовании, чтобы допускать возможность привлечения к работе над трансформациями не только программистов, но и экспертов – специалистов в предметных областях;

2) позволять задавать трансформации моделей как между формализмами (горизонтально), так и в рамках одного формализма (вертикально);

3) позволять применять созданные трансформации прямо в системе (в частности, для их отладки), т.е. производить интерпретацию моделей в том же интерфейсе, в котором они были разработаны;

4) позволять задавать преобразования ограничений, налагаемых на модель.

Существующие подходы [4] обладают значительными недостатками:

– ни в одном из них не предусмотрена возможность выбора языка спецификации метамоделей, изменения его описания, поэтому разработчикам приходится работать с инструментами, которые не обеспечивают достаточную выразительность;

– большинство существующих средств позволяют описывать лишь однонаправленные трансформации моделей, а для получения обратного преобразования необходимо вручную определить ещё одно отображение;

– серьёзным недостатком является и то, что большинство систем позволяют выполнять лишь полные эквивалентные отображения, не учитывая сложные преобразования атрибутов, которые часто встречаются при описании бизнес-процессов, более того, в большинстве реализаций существующих подходов не затрагивается проблема трансформации ограничений, налагаемых на концепты и отношения модели, а производится лишь преобразование объектов моделей и связей между ними.

Таким образом, актуальной становится задача разработки языкового инструментария для создания предметно-ориентированных языков описания трансформаций (Domain Specific Transformations Language, DSTL), который должен быть интегрирован с языковым инструментарием, с помощью которого описываются языки DSL [1-3].

Архитектура инструментария. На рис. 1 показана архитектура разработанного прототипа инструментария DSTL. Тонкие стрелки на этом рисунке показывают связи по управлению, а широкие – передачу данных.

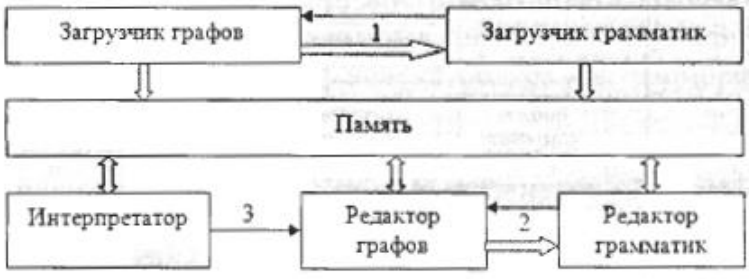

Рисунок 1 - Архитектура прототипа языкового инструментария

Разработанная система включает следующие компоненты:

– *Загрузчик графов*, *загрузчик грамматик* служат для сохранения и загрузки данных (хост-графа и графовой грамматики) из внешней памяти. Информация хранится в типизированных файлах в сериализованном виде.

– *Редактор графов*, *редактор грамматик* служат для создания и редактирования моделей и графовых грамматик пользователем. В качестве редактора графов использован компонент MindFusion.

– *Интерпретатор* применяет загруженную в память грамматику к хост-графу.

Так как составными частями правила грамматики являются графы, загрузчик грамматик в своей работе обращается к загрузчику графов (1, рис. 1). Тот считывает граф и передаёт его загрузчику грамматик. Аналогично, редактор грамматик использует возможности редактора графов (2, рис. 1). Редактор графов используется также при интерпретации грамматики (3, рис. 1). После каждого применения правила пользователю сообщается название правила, которое было применено и в редакторе показывается, как изменился хост-граф. Для реализации редактора графов использован компонент Mind Fusion. Он предоставляет обширные средства для рисования и обработки графов.

Алгоритм исполнения грамматики. Для внутреннего представления правил грамматики используется один граф, элементы которого содержат информацию о том, к какой части правила они относятся. Для пользователя же строится стандартное представление графовых грамматик, где правила имеют левую и правую части.

Система последовательно проверяет каждое правило в грамматике на возможность его применения. Для этого в хост-графе ищется подграф, изоморфный левой части правила. На рис. 2 показана блок-схема алгоритма исполнения графовой грамматики.

Для перебора используется специальный класс NodeSet, хранящий набор вершин и их индексы в хост-графе. В ходе исполнения грамматики система строит набор вершин, содержащий вершины с индексами 0..*M*, где *M* – количество вершин в левой части применяемого правила. Затем последовательно строятся все перестановки данного набора, и каждая проверяется на соответствие искомому графу. Если ни одна из перестановок не соответствует искомому графу, то набор вершин NodeSeta заменяется на следующий по порядку. Для него выполняется перебор перестановок. Это происходит до тех пор, пока не найдётся подграф, изоморфный левой части правила, либо будет установлено, что подходящей перестановки нет. Если какое-то правило можно применить к данному хост-графу, оно применяется. Каждая вершина и дуги из левой части правила заменяются соответствующими элементами из правой части согласно правилу. Значения полей новых элементов вычисляются на основе данных начальных вершин и дуг по правилам, указанным в грамматике. Если ни одно из правил применить нельзя, то исполнение грамматики завершено.

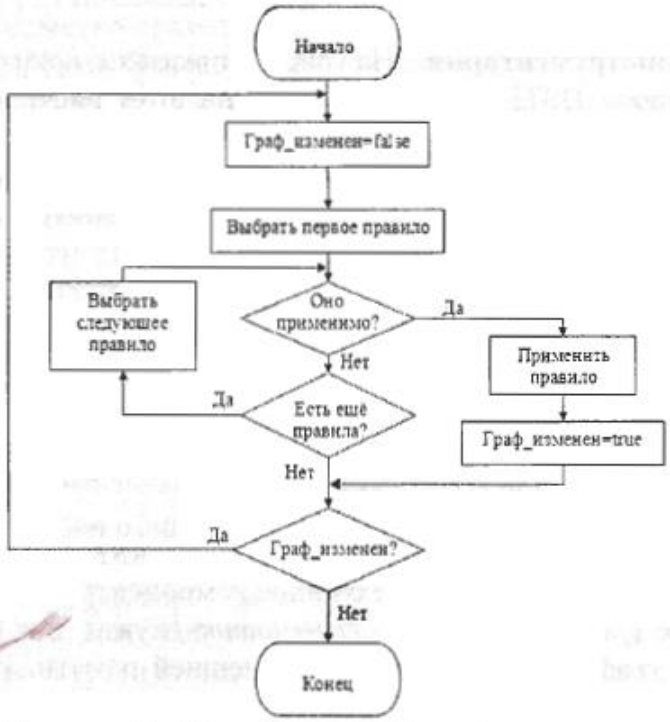

Рисунок 2 - Исполнение графовой грамматики

Данная задача является *NP*-полной и в разработанном прототипе решается полным перебором всех подграфов с таким же числом вершин, что и в левой части правила. Однако для данной задачи разработано значительное количество алгоритмов, работающих с полиномиальной сложностью на некоторых типах графов. Такие алгоритмы часто называют «алгоритмами сопоставления графов» (graph matching algorithms). В дальнейшем планируется заменить перебор одним из таких методов.

Разрабатываемый инструментарий должен принимать от системы MetaLanguage два формализма, для которых будет строиться трансформация, и граф-модель в терминах начального формализма. На основе этих данных пользователь должен построить трансформацию и применить её к заданному графу. Полученный граф-результат будет возвращён в систему MetaLanguage. Формализмы могут совпадать, при этом разработанная грамматика будет изменять модель, например, для автоматизации выполнения одной операции несколько раз – над разными частями модели.

Для передачи формализмов из системы MetaLanguage используется класс Formalism. Он содержит в себе списки вершин и дуг, присущие данному формализму, и имеет имя.

При разработке грамматики пользователь может расширить формализмы новыми вершинами и дугами. Такие расширения могут использоваться для снижения трудоёмкости работы по рисованию графов. Так, если в табличном формализме нет связи M:M, пользователь будет вынужден рисовать дополнительную таблицу для реализации каждой такой связи. При использовании созданного инструментария он может создать новую дугу на основе имеющихся элементов. Аналогично, если в предметной области часто используется определённая структура, имеет смысл создать новую вершину, которая будет представлять соответствующий подграф. Для этого необходимо описать «расшифровку» новой вершины/дуги с помощью имеющихся элементов формализма, её название и, если создаётся новая вершина, визуальный образ (дуги их не имеют). Этот новый тип элементов будет содержать поля всех элементов соответствующего подграфа (расшифровки). Созданный элемент может быть использован для описания новых типов вершин и дуг. При создании нового элемента пользователь может создать его в специальном окне, а может сначала нарисовать граф в редакторе правил, выделить в нем соответствующий подграф и создать представляющий его элемент.

Заключение. Анализ различных используемых в настоящее время подходов к описанию графовых трансформаций показал, что графовые грамматики являются наиболее наглядным и простым из подходов, поэтому было решено использовать именно этот аппарат в разрабатываемом инструментальном средстве, предназначенном для описания трансформаций визуальных моделей.

Реализован исследовательский прототип инструментария, позволяющий производить описание как горизонтальных, так и вертикальных трансформаций (мета)моделей. Планируется реализовать алгоритм быстрого поиска подграфов, изоморфных левой части правила.

Проект выполняется при поддержке РФФИ (проект № 12-07-00763-а).

Библиографический список

1. *Лядова Л.Н., Сухов А.О.* Языковой инструментарий системы MetaLanguage // Математика программных систем: Межвуз. сб. научн. тр. / Перм. ун-т. Пермь, 2009.

2. *Сухов А.О.* Предметно-ориентированный язык в адаптируемых информационных системах // Технологии Microsoft в теории и практике программирования / Материалы конф. Новосибирск, 2008. С. 25 26.

3. *Сухов А.О.* Среда разработки визуальных предметно-ориентированных языков моделирования // Математика программных систем: Межвуз. сб. научн. тр. / Перм. ун-т. Пермь, 2008. С. 84-94.

4. *Сухов А.О.* Анализ формализмов описания визуальных языков моделирования / А.О. Сухов // Современные проблемы науки и образования. – 2012. – № 2; URL: www.science-education.ru/102-5655 (дата обращения: 02.04.2012).

5. *Balasubramanian D., Narayanan A., Buskirk C., Karsai G.* The Graph Rewriting and Transformation Language: GReAT // Lecture Notes in Computer Science. – 2008. - №5088/2008. – с. 410-425.

6. *Balogh A., Varro D.* Advanced Model Transformation Language Constructs in the VIATRA2 Framework: [Электронный документ] (http://static.inf.mit.bme.hu/pub/varro/2006/sac2006_vtcl.pdf). Проверено 01.06.2012.

7. *Jouault F., Kurtev I.* Transforming Models with ATL: [Электронный документ] (http://wwwhome.cs.utwente.nl/~kurtev/files/Models2005.pdf). Проверено 19.06.2012.

8. *Markovic S., Baar T.* Semantics of OCL specified with QVT // Lecture Notes in Computer Science. – 2006. - №4199/2006. – с. 661-675.

9. MDA Guide Version 1.0. OMG document, Miller, J. and Mukerji, J. Eds., 2003 [Электронный документ] (http://www.omg.org/docs/omg/03-06-01.pdf). Проверено 19.06.2012.

10. *Stevens P.* Bidirectional Model Transformations in QVT: Semantic Issues and Open Questions: [Электронный документ] (http://homepages.inf.ed.ac.uk/perdita/bidirectional.pdf). Проверено 19.06.2012.

ТРАНСЛЯЦИЯ ФУНКЦИОНАЛЬНЫХ ПРОГРАММ В ОПИСАНИЕ УСТРОЙСТВ С АРХИТЕКТУРОЙ ПОТОКА ДАННЫХ

И.Ю. Никляев

Волгоградский государственный технический университет, г. Волгоград

Введение. Постановка задачи. При программировании устройств, построенных на архитектуре потоков данных, в настоящее время используются языки уровня абстракции RTL (например, VHDL или LabView G) либо алгоритмического уровня (SystemC, Lucid, Bluespec). Однако даже в последних выразительная способность ограничена, что в главной степени связано с трудностями реализации в языках описания аппаратуры рекурсивных алгоритмов и работы со сложными типами данных. Так, примитивные типы указанных выше языков не обладают свойством полиморфизма и не могут быть абстрактными. Кроме того, в них отсутствуют механизмы построения алгебраических типов. Возможности использования рекурсии же ограничиваются в лучшем случае хвостовой рекурсией (SAFL) [2].

Задачей данной работы является разработка способа реализации алгебраических типов данных и рекурсивных алгоритмов в устройствах, построенных на архитектуре потоков данных, а также являющихся статически распределенными. Согласно [2], статически распределенные системы занимают объем памяти, известный на этапе компиляции программы (или синтеза устройства) и не зависящий от вызова функций или создания объектов во время выполнения. Также в [2] показано, что статически распределенные вычислительные системы могут быть реализованы аппаратно прямым отображением функций на вычислительные устройства.

Общая схема. Базовым элементом программы на языках потоков данных является поток, представляющий собой последовательность объектов первого класса, при этом сами потоки объектами первого класса не являются. Например, в Lucid поток определяется как $s = s_0 \; fby \; s_n(s_{n-1})$, где s_0 - начальное значение потока, а $s_n(s_{n-1})$ - некоторая рекуррентная формула. На первом шаге синхронизации поток содержит начальное значение, на каждом следующем оно рассчитывается по рекуррентной формуле из предыдущего