А.О. Сухов[1]

Пермский государственный национальный исследовательский университет

Sukhov_PSU@mail.ru

СРАВНЕНИЕ СИСТЕМ РАЗРАБОТКИ ВИЗУАЛЬНЫХ ПРЕДМЕТНО-ОРИЕНТИРОВАННЫХ ЯЗЫКОВ

Языковой инструментарий, или *DSM-платформа*, – это инструментальное программное обеспечение, предназначенное для поддержки разработки и сопровождения языков предметной области. Использование при создании DSL языкового инструментария значительно упрощает процесс создания языков.

Достоинством DSM-платформы является то, что она предоставляет возможность интеграции в одной системе сразу нескольких DSL. Это часто бывает необходимо, поскольку при задании ограничений на объекты предметной области и описании бизнес-процессов используются совершенно разные языки.

Удобство использования языковых инструментариев проявляется и при внесении изменений в уже созданные DSL. Разработчику не придется разбираться в большом объеме кода парсера языка, а будет достаточно изменить описание DSL в языковом инструментарии. Ещё одним важным свойством является возможность «отчуждения» созданных языков, разработанных с их использованием моделей.

Существует большое число средств разработки DSL с возможностью задания собственной графической нотации. Такими средствами являются, например, MetaEdit+, MS DSL Tools, Eclipse GMF, QReal и др.

Языковой инструментарий MetaEdit+

MetaEdit+ является инструментальным средством CASE-системы, разрабатываемой финской компанией MetaCase [19, 23], предназначенным для создания DSM-приложений.

[1] Работа выполнена при поддержке Программы «Научный фонд НИУ ВШЭ» (проект № 12-09-0102)

MetaEdit+ состоит из следующих подсистем:

- *MetaEdit+ Workbench* – средство создания языков моделирования и генераторов.
- *MetaEdit+* – полнофункциональная среда разработки систем с поддержкой возможности использования собственных языков моделирования, генераторов кода и документации, созданных с помощью MetaEdit+ Workbench.

Таким образом, помимо возможности создания предметно-ориентированного языка, разработчик получает в распоряжение и CASE-средство, в котором может использовать этот язык.

MetaEdit+ позволяет применять в процессе создания информационных систем сразу несколько DSL. Мультиязыковые средства также позволяют переиспользовать конструкции существующих в инструментарии языков.

MetaEdit+ является многоплатформенной средой, данная DSM-платформа способна работать практически с любой операционной системой (Windows, Linux, Solaris, Mac OS X).

MetaEdit+ предоставляет пользователям возможность одновременной работы сразу с несколькими проектами, каждый из которых может иметь несколько моделей. Проекты могут содержать как описание языков моделирования, так и модели, созданные с их помощью.

Рассмотрим инструментарий, который предоставляет данная DSM-платформа, его возможности.

Инструментальные средства MetaEdit+

MetaEdit+ предоставляет разработчику большой набор инструментальных средств, предназначенных для работы с языками моделирования (редакторы, браузеры, репозиторий, генераторы).

Редакторы. Назначением редакторов является создание, изменение, удаление моделей, а также установление взаимосвязи между различными моделями. В MetaEdit+ присутствует три вида редакторов: редактор диаграмм, редактор матриц и редактор таблиц. Каждый из этих редакторов отображает модель в определенном виде.

Редакторы матриц и таблиц используют для представления моделей матрицы и таблицы соответственно. При работе с ними пользователь изменяет необходимое значение свойства объекта в соответствующей ячейке. Отличие редактора матриц от редактора таблиц состоит в том, что первый отображает все объекты графа, в то время как редактор таблиц в конкретный момент времени способен работать лишь с одним объектом.

Редактор диаграмм обеспечивает графическое представление мо-

дели. Каждый элемент модели представлен графическим символом, а связи между элементами изображаются различными видами линий.

Все редакторы оперируют одними и теми же понятиями проекта, но используют для этого различные представления. Применение графического редактора при создании и модификации моделей значительно удобней матричного и табличного.

Браузеры. Браузер графов обеспечивает представление моделей проекта: все элементы рассматриваются как части моделей, которым они принадлежат.

MetaEdit+ предоставляет в распоряжение пользователя следующие виды браузеров:

- браузер графа позволяет пользователю просмотреть содержимое выбранного графа и выполнять операции над ним и его содержимым;
- браузер объектов обеспечивает иерархическое представление объектов графа;
- браузер типов позволяет просматривать и открывать редакторы элементов метамодели, в том числе графа, объектов, отношений, ролей, типов.

Репозиторий. Репозиторий содержит информацию о доступных языках моделирования и моделях, включая все их элементы и свойства. Для генерации кода и документации используется информация, хранящаяся в репозитории. Все произведенные пользователем изменения в языке моделирования будут автоматически отражены во всех моделях. В однопользовательской версии репозиторий расположен на локальном компьютере пользователя, а в многопользовательской – на сервере.

Одновременные изменения различными пользователями одних и тех же данных блокируются. Операция чтения данных доступна всегда, независимо от операций, совершаемых другими пользователями.

Генераторы. Генераторы MetaEdit+ позволяют проверить правильность созданных моделей и сгенерировать на их основе код или документацию. Разработчик может модифицировать имеющиеся генераторы или создавать новые.

Помимо возможности использования встроенных генераторов кода, поддерживающих языки C++, Smalltalk, CORBA IDL, Java, Delphi, Mobile Information Device Profile (MIDP) – набор Java API функций для мобильных телефонов, MetaEdit+ предоставляет разработчику возможность создавать генераторы для собственных языков программирования и моделирования. Таким образом, можно говорить о том, что под-

держивается возможность определения правил трансформации моделей в код.

MetaEdit+ позволяет *импортировать* и *экспортировать* метамодели в формат MXT (файл MetaEdit+ XML Types), который базируется на общепринятом стандарте XML.

MetaEdit+ предоставляет возможность на основе существующих моделей *сгенерировать документацию к проекту*. Документация модели включает в себя: название модели, информацию о разработчиках. Если модель имеет графическое представление, то в документацию будет включена и диаграмма в формате PNG со ссылками на описание отдельных ее частей.

Метамоделирование в MetaEdit+

Благодаря реализации принципов метамоделирования, MetaEdit+ позволяет разработчику самому определять языки моделирования и генераторы для них.

Используя язык метамоделирования, разработчик создает метамодель предметной области, определяя ее основные элементы, их атрибуты, связи между элементами и различные правила интеграции нескольких языков в одной системе.

При построении метамодели в MetaEdit+ используется язык метамоделирования GOPRR, получивший свое название от основных понятий, которыми он оперирует: граф (Graph), объект (Object), свойство (Property), связь (Relation) и роль(Role) [18]. Рассмотрим эти понятия более подробно.

Под графом понимают создаваемый язык моделирования, т.е. *граф* – это совокупность объектов, связей, ролей объектов. Примером графа может служить метамодель диаграмм потоков данных, диаграмм прецедентов UML.

Объекты – основные понятия языка моделирования. Примером объекта является: деятельность в диаграммах активностей UML, класс в диаграммах классов UML, сущность в диаграммах ERD.

Связи используется для соединения двух и более объектов. Примеры связей: ассоциация, наследование в диаграммах классов UML.

В MetaEdit+ поддерживаются следующие виды связей:

- ассоциация соединяет два или более равноправных объектов графа;
- наследование позволяет создавать новый объект на основе уже существующего объекта;
- свойство определяет атрибуты, которые характеризуют некоторый объект графа, причем каждый атрибут является отдель-

ным объектом графа или внешним источником, таким как файл, программа или веб-сервис;
- декомпозиция позволяет для любого объекта графа задать уточняющий граф;
- детализация позволяет объектам, связям или ролям ссылаться на элементы других графов.

Роль определяет, каким образом объект участвует в связи. Так объекты, участвующие в связи наследования могут иметь одну из двух ролей «родитель» или «потомок».

Свойства – это атрибуты какого-либо объекта графа. Тип свойства может быть любым: строковым, целочисленным, логическим, ссылкой на другие концепты языка моделирования или внешние источники и др.

Создание метамодели начинается с построения GOPRR-модели, после чего она может быть открыта в MetaEdit+ Workbench для построения модели предметной области.

При этом метамодель может быть загружена в систему и использоваться в качестве метаязыка.

Благодаря подходу, основанному на интерпретации данных, MetaEdit+ позволяет производить *динамическое* изменение метамоделей.

Таким образом, к *достоинствам системы* MetaEdit+ следует отнести:
- возможность динамического изменения метамоделей;
- возможность изменения метаязыка;
- различные способы представления моделей и метамоделей: диаграммы, таблицы, матрицы;
- возможность генерации исходного кода системы и документации по заранее определенным шаблонам;
- генерация программного кода для нескольких языков программирования: C++, Java, Delphi и др.;
- возможность работы в разных операционных системах;
- многопользовательский репозиторий среды MetaEdit+, поддерживающий проекты в согласованном состоянии;
- возможность одновременной работы с несколькими проектами.

К *недостаткам системы* следует отнести:
- отсутствие полнофункциональной среды программирования, которая позволила бы «вручную» вносить изменения в сгенерированный системой код;
- отсутствие возможности трансформации визуальных моделей.

Microsoft Tools for Domain-Specific Languages

DSM-платформа *Microsoft Tools for Domain-Specific Languages (DSL Tools)* является частью нового подхода к созданию программного обеспечения – Software Factories. Этот подход компании Microsoft предназначен для ускорения процесса разработки программного обеспечения и минимизации затрат на разработку.

DSL Tools позволяет создавать собственные визуальные языки моделирования и строить для них графические редакторы.

Инструментальные средства DSL Tools

В состав DSL Tools входят:

- мастер проектов, предназначенный для создания новых проектов DSL;
- дизайнер, предназначенный для определения и визуализации метамодели предметной области;
- генераторы кода, которые на основе определенной пользователем модели и дизайнера строят по указанному шаблону код их реализации.

Создание нового DSL можно производить в соответствии с одним из семи шаблонов [13]:

- *Minimal Language* – шаблон создает простой язык, состоящий из двух понятий предметной области и одной связи между ними;
- *Task Flow* – шаблон для создания языка моделирования, схожего с диаграммами деятельности UML, основной элемент языка – деятельность, а основное отношение – переход между деятельностями; шаблон включает в себя несколько других элементов, таких как начальное состояние, конечное состояние и состояние синхронизации;
- *Class Diagrams* – шаблон создает предметно-ориентированный язык, который напоминает диаграммы классов UML, основные элементы – классы и интерфейсы вместе со связями ассоциации, обобщения и реализации;
- *Component Models* – шаблон создает предметно-ориентированный язык, который напоминает диаграммы компонентов UML, основные элементы – компоненты и порты;
- *Minimal WPF* – данный шаблон позволяет создавать приложения, в которых DSL интегрирован в WPF интерфейс, а не в графический редактор;

- *Minimal WinForm* – данный шаблон позволяет создавать приложения, в которых DSL интегрирован в Windows Form, а не в графический редактор;
- *DSL Library* – данный шаблон позволяет создавать частные DSL, которые могут быть интегрированы в другие предметно-ориентированные языки.

При создании нового DSL автоматически создаются два проекта: Dsl и DslPackage. Первый предназначен для хранения различных артефактов метамодели создаваемого DSL, второй содержит настройки графического редактора.

Спецификация метамодели и редактора подаются на вход генератору DSL Tools. После их компиляции запускается новый экземпляр MS Visual Studio, где в качестве языка разработки используется созданный предметно-ориентированный язык. Таким образом, создаваемый DSL используется исключительно как составная часть MS Visual Studio, вне этой среды программирования ни DSL, ни редактор использовать нельзя.

Графические конструкции DSL-дизайнера

Для описания метамодели DSL предоставляет разработчику следующий набор графических конструкций [14]:

- *Доменный класс* (*Domain Class*) используется для задания отдельного объекта создаваемого языка.
- *Именованный доменный класс* (*Named Domain Class*) – поименованный доменный класс.
- *Ассоциация* (*Reference Relationship*) соответствует обычной ассоциации между двумя классами в UML. Ассоциация, как и другие отношения между доменными классами, изображается на диаграмме специальным прямоугольником и также является классом, благодаря этому появляется возможность назначать различные свойства связям.
- *Агрегация* (*Embedding Relationship*) используется для задания агрегирования одного *доменного класса* другим. Агрегация отличается от ассоциации тем, что при удалении главного объекта по умолчанию удаляются зависимые объекты.
- *Наследование* (*Inheritance*) соответствует традиционному наследованию классов.

Рассмотрим элементы, с помощью которых в DSL-дизайнере задается конкретный синтаксис языка:

- *Геометрическая фигура* (*Geometry Shape*) используется в качестве пиктограммы для отображения доменных классов. Геометрическая фигура может быть одного из следующих видов:

прямоугольник, прямоугольник со скругленными углами, эллипс, окружность.

- *Сложная фигура* (*Compartment Shape*) используется для визуализации доменных классов, позволяет задавать прямоугольники, у которых можно скрывать/отображать отдельные блоки (например, блоки свойств и методов).
- *Изображение* (*Image Shape*) позволяет задать в качестве пиктограммы для отображения конструкции DSL произвольное изображение.
- *Соединитель* (*connector*) используется для задания объектов визуализации отношений DSL.
- *Разделитель* (*Swimlane*) – конструкция, которая позволяет разделить диаграмму на несколько секций, например, секции *Classes and Relationships* и *Diagram Elements*, а также задать, какие графические конструкции в какой секции будут располагаться.

Метамодель обязательно должна включать корневой класс, который будет содержать в себе все остальные элементы. Все классы располагаются в виде дерева, соединяясь с другими элементами агрегацией, ассоциацией или наследованием.

Для соединения конструкции абстрактного и конкретного синтаксиса в DSL-дизайнере предусмотрена специальная конструкция – *диаграммный соединитель* (*Diagram Element Map*). При соединении доменных классов и отношений с фигурами и линиями диаграммные соединители автоматически определяют соответствие между элементом абстрактного и конкретного синтаксиса. Далее следует задать соответствие между доменными свойствами и декораторами.

Свойства элементов дизайнера

Рассмотрим свойства элементов DSL-дизайнера. Для удобства использования они разделены на две группы:

- *доменные свойства* (Domain Properties) есть у всех классов;
- *декораторы* (Decorators) есть только у графических классов (фигур и соединителей).

При создании классов в DSL Tools, разработчик может задать у них произвольное количество *доменных свойств*. Для доменных классов такие свойства используются для задания атрибутов. Для графических классов доменные свойства позволяют определить характеристики этих классов, связанные с метамоделью, в то время как *декораторы* определяют свойства графических классов, связанные с графическими характеристиками фигур.

Каждое доменное свойство имеет следующие атрибуты:

- *code* – характеристики данного свойства как свойства C#-класса, например, его видимость (public, protected);
- *definition* – определение таких характеристик доменного свойства, как его имя (Name), значение по умолчанию (Default Value), быть именем класса или нет (Is Element Name) – если да, тогда генератор обеспечит уникальность имен по умолчанию на диаграммах создаваемого графического редактора;
- еще один атрибут из этой группы – *kind*, который определяет тип свойства, он может принимать значения *normal* (обычное) или *calculated* (вычислимое).

В DSL Tools выделяют следующие виды декораторов:

- *текст* (Text);
- *изображение* (Icon);
- *сворачиваемая область* (Expand Collapse).

Создание предметно-ориентированных языков с помощью DSL Tools

Создание метамодели происходит в два этапа [20]:

1) описание абстрактного синтаксиса с помощью диаграмм классов UML;
2) описание конкретного синтаксиса языка.

После выполнения этих этапов метамодель и редактор дополняются различными свойствами, определяющими настройки редактора. Также задается пользовательский интерфейс для работы с редактором. Так, например, программно можно реализовать различные графические символы «ромб», «трапеция», отсутствующие в стандартной палитре DSL Tools и используемые для отображения объектов метамодели.

Компонент валидации позволяет пользователю задать правила, которым должны удовлетворять модели и/или редактор. Определить собственные правила валидации новых визуальных моделей, а также дополнительные свойства нового редактора в тех случаях, когда не хватает стандартных возможностей DSL Tools, можно с помощью фрагментов кода на языке C#.

В рамках DSL Tools можно, по специальным правилам, написать модуль генерации исходного кода, который будет выполнять генерацию кода на основе моделей, созданных в новом редакторе. Для этого требуется описать шаблон генерируемого файла: там, где нужно, вставить статический текст или, если выводимая информация зависит от свойств модели, сделать вставку на языке C#, которая извлекает нужную информацию из моделей.

После определения метамодели DSL и свойств редактора необходимо провести валидацию. В случае успешной валидации на основе спроектированной модели производится генерация кода редактора в тексты на языке C#.

Дополнительная функциональность, такая как процедуры валидации моделей, создается разработчиком в «ручном» режиме в виде частичных классов, расширяющих функциональность сгенерированных классов. Использование частичных классов гарантирует сохранение дополнительного кода, добавленного разработчиком, при последующих изменениях модели предметной области или перегенерации кода.

После компиляции кода получается редактор с палитрой графических элементов, браузером модели, редактором диаграмм, валидатором, генератором кода. Созданный редактор может быть открыт в новом экземпляре MS Visual Studio или для него может быть создан Installer. С помощью этого редактора пользователь может строить модели.

Преимуществами системы DSL Tools являются:

- наличие шаблонов предметно-ориентированных языков, модификация которых позволяет пользователю достаточно просто создать собственный DSL;
- широкие возможности настройки редактора моделей;
- возможность задания ограничений на языке C#;
- возможность генерации кода и XML-файлов на основе созданных моделей;
- возможность задания шаблонов для генерации кода;
- наличие полнофункциональной среды доработки исходного кода – MS Visual Studio.

Недостатками системы являются:

- отсутствие возможности динамического изменения метамоделей;
- описание метаязыка изменить невозможно;
- созданный язык моделирования может использоваться лишь в MS Visual Studio;
- неудобный редактор DSL: при построении метамоделей получаются громоздкие и сложные для восприятия диаграммы;
- отсутствие возможности трансформации моделей;
- медленная работа;
- возможность генерации исходного кода лишь на языках C# и VB;
- несогласованность визуальных метамоделей и сгенерированного на их основе программного кода.

Технология Eclipse Graphical Modeling Framework

Среда *Eclipse* – это многоплатформенная интегрированная среда разработки программного обеспечения с открытыми исходными кодами [16]. Основной язык разработки, поддерживаемый этой средой, – Java, хотя имеется также поддержка C++, Perl, Fortran и др. На базе этой среды создаются различные дополнительные технологии, одна из которых – Eclipse Graphical Modeling Framework (GMF), первая версия которой появилась в середине 2006 года.

Технология GMF предназначена для быстрой разработки графических средств, главным образом, интегрируемых в Eclipse. Она является Open Source разработкой и развивается, в основном, специалистами компаний IBM и Borland. GMF интегрирует две широко используемые и известные Eclipse-библиотеки – Eclipse Modeling Framework (EMF) и Graphical Editing Framework (GEF).

Архитектура DSL, созданного с применением GMF, строится на основе MVC-шаблона (Model/View/Controller). Для создания уровней представления и контроллеров используется технология GEF, для создания моделей – технология EMF.

Особенности технологий GEF и EMF

Библиотека GEF состоит из двух частей: модуля OED, встроенного в Eclipse, и самой базовой библиотеки GEF [0]. Модуль OED предоставляет средства программного создания и обработки графических объектов: менеджеры размещения графических объектов, палитра стандартных объектов, средства создания собственных графических объектов, средства создания связей между графическими объектами, средства работы со слоями изображений и пр.

Графический редактор, созданный с помощью OED, визуализирует экземпляры Java-классов, которые описывают объекты предметной области. Для синхронизации классов и визуальных диаграмм между ними должна существовать «прослойка» (Controller), которая создается с помощью библиотеки GEF. Изменение модели производится не напрямую, а с помощью определенного набора команд, которые вносят изменения одновременно и в классы, и в визуальные диаграммы. Это позволяет поддерживать систему в синхронизированном состоянии и отменять все произведенные изменения.

Визуальные DSL с помощью GEF можно создавать на базе любых моделей, однако наибольшей эффективности можно достичь, интегрируя технологию GEF с EMF.

Технология EMF предназначена для проектирования приложений, использующих модели бизнес-процессов, имеющих сложную структу-

ру. Создание таких приложений производится с помощью средств генерации EMF по созданной или импортированной модели. При этом разработчику предоставлена возможность расширения функциональности «вручную». Внесенные изменения будут сохранены и при последующих генерациях кода.

Технологии GEF и EMF создавались независимо друг от друга, поэтому существует несогласованность их интерфейсов. Кроме того, каждая из них предназначена для решения более широкого класса задач, чем поддержка создания DSL. Для преодоления этих проблем на основе двух технологий был реализован проект GMF.

Создание DSL с помощью Eclipse GMF

Процесс создания DSL с помощью технологии Eclipse GMF состоит из следующих этапов [17]:

1. Описание доменной модели (абстрактного синтаксиса). В случае описания языка UML элементами доменной модели будут «класс», «наследование», «ассоциация» и др. Метамодель разрабатывается с помощью графического редактора GMF Ecore.
2. Разработка графической модели (конкретного синтаксиса). Например, для UML-редактора и элемента «класс» элементом графической модели будет прямоугольник с секциями для имени, свойств и методов класса.
3. Разработка модели инструментов – описание элементов панели инструментов будущего редактора: палитры графических объектов, списка действий над ними, меню графических объектов и т.д.
4. Задание модели соответствия. Все предыдущие модели являются независимыми, никак не связанными друг с другом, каждая из них располагается в одном или нескольких файлах. В них хранятся объекты, которые могут использоваться при построении графического редактора, а как именно они будут использоваться, определяется в модели соответствия. Для элементов доменной модели задаются связи с его графическим представлением, а также соответствующими инструментами. Например, для UML-класса будет указано, что для его отображения используется прямоугольник, который создается с помощью кнопки с именем «Класс». Не для всех элементов доменной модели должны быть указаны соответствия.
5. Создание модели генератора, по которой производится генерация графического редактора для DSL. Данная модель явля-

ется промежуточным представлением будущего редактора. Она автоматически генерируется по модели соответствия и дополняется разработчиками «вручную».

6. Генерация кода создаваемого языка.

К *достоинствам* рассмотренной системы можно отнести:

- возможность создания отчуждаемых от Eclipse предметно-ориентированных языков;
- наличие инструментальной панели Dashboard, упрощающей процесс создания DSL;
- существование надстроек над Eclipse, позволяющих производить трансформацию одной модели в другую, например, с помощью языка трансформации моделей ATL [12];
- наличие полнофункциональной среды доработки исходного кода – Eclipse;
- открытые исходные коды DSM-платформы, что позволяет более точно настроить ее на потребности пользователя.

Недостатками системы являются:

- отсутствие возможности динамического изменения метамоделей;
- описание метаязыка изменить невозможно;
- неудобные редакторы конкретного синтаксиса, модели инструментов и модели соответствия;
- медленная работа;
- сложность работы с системой для непрофессиональных программистов.

State Machine Designer

Помимо коммерческих проектов по разработке инструментариев для создания DSL существуют и исследовательские. Разработанный на кафедре компьютерных технологий Санкт-Петербургского государственного национального исследовательского университета информационных технологий механики и оптики визуальный язык автоматного программирования State Machine Designer [5] создан на основе описанной ранее DSM-платформы DSL Tools.

Разработчики State Machine Designer создают свой язык с целью устранить ряд недостатков, существующих в DSL Tools. Так, например, абстрактный синтаксис не всегда остается синхронизированной с исходным кодом. Если создать на диаграмме класс, а потом удалить диаграмму, созданный класс все равно останется в проекте, и наоборот, если создать класс отдельно от диаграммы, то по умолчанию, новый класс на диаграмме не появится.

Рассмотрим описание языка State Machine Designer более подробно. Родительским элементом метамодели является класс *StateMachine* (конечный автомат). Класс *StateMachine* содержит класс *BaseState* (базовое состояние). Абстрактный класс *BaseState* является родительским для классов *FinalState, StartState, State*. Класс *State* имеет три свойства: *EntryActivity* (действие, выполняемое на входе в состояние), *IncludeActivity* (действие, выполняемое в состоянии), *ExitActivity* (действие, выполняемое при выходе из состояния). Класс *Transition* соответствует переходам между состояниями. *Transition* содержит три свойства: *Trigger* – событие, по которому следует производить переход, *Guard* – условие которое должно быть выполненным для перехода, *Activity* – действие, выполняемое при переходе.

В State Machine Designer был разработан собственный генератор кода, который построен на основе генератора кода DSL Tools. Разработанный генератор кода загружает текстовый шаблон из ресурсов модуля и подставляет в текст шаблона нужные имена файлов для корректной генерации кода по модели.

Во время исполнения модели, описанной с помощью конечных автоматов, происходит журнализация всех изменений данной модели. Протоколируются следующие события: запуск выполнения модели автомата, входы в состояния, действия на входах в состояния, выходы из состояний, действия на выходах из состояний, остановка выполнения модели автомата, действия на переходах.

Благодаря созданию собственного генератора и журнализации событий, разработчикам State Machine Designer удалось исправить некоторые недостатки DSL Tools.

Система QReal

Развитием системы Real-IT [1] стала система QReal, разрабатываемая на кафедре системного программирования Санкт-Петербургского государственного университета под руководством профессора А.Н. Терехова. Изначально планировалось, что QReal будет являться развитием технологии Real-IT, основывающимся на использовании новой версии языка UML 2.0 и удовлетворяющим требованиям многоплатформенности. Однако впоследствии разработчиками было принято решение также включить в систему элементы метамоделирования, которые позволили упростить процесс создания новых редакторов [4].

QReal имеет клиент-серверную архитектуру. На сервере расположен репозиторий, хранящий информацию обо всех метамоделях. Каждый клиент имеет доступ к репозиторию. В основу системы положен шаблон проектирования Model/View, в качестве модели (Model) здесь

выступает бизнес-логика, расположенная на клиентах, а в качестве представления (View) – различные элементы пользовательского интерфейса.

В QReal представление и хранение данных основано на разделении графической и логической модели. Некоторые элементы модели могут не иметь визуального представления, например, значения перечислимых типов, некоторые, наоборот, имеют только визуальное представление и на логику работы системы никак не влияют, например, просто линия или прямоугольник на диаграмме, некоторые элементы могут иметь несколько представлений. В CASE-пакете реализованы средства, позволяющие разработчикам создавать различные представления одних и тех же логических моделей.

Для обеспечения контроля версий и многопользовательской работы в QReal используется SVN-сервер, доступ к которому осуществляется посредством репозиториев клиентов, которые хранят свои модели в виде файлов, организованных в рабочую копию системы контроля версий.

Основой системы QReal является написанное вручную абстрактное ядро, реализующее общую для всех редакторов и элементов диаграмм функциональность, и подключаемые модули – наследуемые от ядра классы, реализующие специфику конкретных редакторов и генерируемые автоматически по XML-описаниям метамоделей создаваемых языков. Изначально QReal позволял описывать метамодели лишь в текстовом виде: на языке, являющимся расширением XML. Позже разработчики предоставили пользователю возможность графического задания как метамоделей создаваемых языков (с помощью метаредактора), так и графического представления их элементов на диаграммах (средствами встроенного в QReal графического редактора векторных изображений), но на физическом уровне метамодель хранится в виде XML-описания. Для визуального построения метамоделей редактор предоставляет пользователям визуальный метаязык, являющийся подмножеством языка стандарта MOF, который содержит следующие элементы:

- *классы* (*сущности*) – множество элементов конкретного типа;
- *ассоциации* (*отношения*) между классами;
- *атрибуты классов*, используемые для задания свойств элементов заданного типа.

Построение визуального DSL состоит из следующих этапов:

1) описание абстрактного синтаксиса с помощью метаредактора;
2) описание конкретного синтаксиса с помощью метаредактора;

3) генерация кода на C++, реализующего функционал создаваемого редактора DSL и использующего интерфейсы, объявленные в основной части системы;
4) компиляция сгенерированного кода в динамическую библиотеку – подключаемый модуль.

Каждый подключаемый модуль содержит информацию о наборе объектов, допустимых на диаграммах данного типа, позволяет правильно интерпретировать хранящиеся в репозитории значения атрибутов элементов и предоставляет информацию о логических правилах размещения элементов на диаграммах.

Графический интерфейс рассматриваемой системы содержит следующие компоненты [1]:

- Инспектор объектов, представляющий собой дерево объектов текущего проекта, содержащее все его элементы, сгруппированные по их типам. Инспектор объектов удобен для обзора проекта в целом и при добавлении на текущую диаграмму уже существующего в проекте объекта.
- Инспектор диаграмм, который отображает все пользовательские диаграммы и их элементы в виде дерева. Один элемент может принадлежать нескольким диаграммам, тогда в инспекторе он будет отображаться как потомок всех диаграмм, которым он принадлежит, тогда как в инспекторе объектов и в репозитории этому элементу будет соответствовать только один объект.
- Редактор свойств, позволяющий просматривать и изменять атрибуты выделенного элемента.
- Палитра компонентов, содержащая все доступные для создания элементы метамодели.
- Меню и панели инструментов предоставляют доступ к основным операциям над репозиторием, диаграммами, объектами и др.
- Рабочая область графического редактора диаграмм.

Для быстрого создания трансляторов визуальных диаграмм в исходный код на целевом языке в QReal используется специальный язык описания генераторов, который позволяет для визуального языка задать правила обхода созданных с его помощью моделей и генерации кода по ним. В дальнейшем эти правила интерпретируются для конкретных диаграмм, порождая соответствующий им исходный код.

Генерация кода в рассматриваемой системе происходит следующим образом [11]: построенные пользователем описания метамоделей подаются на вход специальному генератору, который осуществляет

анализ предоставленных ему XML-описаний. В процессе разбора соответствующих XML-описаний генератор заменяет теги параметризации заданным кодом на C++ для получения нужных значений атрибутов элемента, при этом теги форматирования текста сохраняются. После разбора XML-описаний метамоделей производится построение набора свойств сущностей, списка допустимых ассоциаций, проверяется ссылочная целостность, выполняются другие проверки семантической корректности описаний редакторов.

В ходе заключительной фазы происходит генерация необходимых артефактов:

- классов на языке C++ для всех элементов и ассоциаций диаграмм;
- файлов с описаниями графического представления элементов;
- внутренних средств доступа к значениям атрибутов элементов репозитория из любых модулей проекта;
- дополнительного кода для присоединения генерируемых классов к проекту, в него входят фабрика объектов для создания экземпляров описанных типов элементов и ассоциаций, а также служебные файлы – файл ресурсов проекта и файл, осуществляющий включение генерируемых файлов с классами описанных сущностей в сборку проекта.

В системе QReal реализован визуальный интерпретатор, который позволяет разработчику пошагово выполнять созданные поведенческие диаграммы, основанные на сетях Петри. При этом среда на каждом шаге автоматически проверяет синтаксическую и семантическую корректность описанных диаграмм. При этом разработчик может менять диаграмму непосредственно в процессе интерпретации. В будущем разработчики системы планируют создать полноценный отладчик сгенерированного по диаграммам исполняемого кода.

К *преимуществам* данной системы можно отнести:

- возможность задания метамодели как в визуальном, так и в текстовом виде;
- многоплатформенность, система поддерживает возможность генерации исходного кода под Windows и Linux, сборка продукта возможна на всех платформах, поддерживаемых библиотекой Qt (Windows, Mac OS, Linux/X11, Embedded Linux, Windows CE и S60);
- наличие возможности автоматической генерации визуальных редакторов диаграмм без написания исходного кода «вручную»;

- наличие интерпретатора, позволяющего пошагово выполнять поведенческие диаграммы;
- возможность одновременной удаленной работы с репозиторием сразу нескольких клиентов.

К *недостаткам* системы QReal следует отнести:

- сложность модификации созданного языка моделирования, поскольку после внесения изменений в метамодель необходимо повторно произвести генерацию кода и его сборку в динамическую библиотеку;
- отсутствие возможности итеративного определения метаязыка;
- отсутствие возможности изменения метаязыка, т.е. конечный пользователь не может внести изменения в метаязык и должен довольствоваться возможностями, предоставляемыми MOF.

Разработка DSM-платформы на основе системы MS Visio

В работе [3] рассмотрен подход к созданию комплекса средств разработки предметно-ориентированных визуальных языков на основе пакета Microsoft Visio. Данный редактор был выбран авторами подхода в качестве основы для DSM-платформы в силу следующих причин [10]:

- данная система имеет широкое распространение, поэтому с ее помощью можно достаточно просто создавать новые проекты и производить интерпретацию существующих;
- Microsoft Visio является достаточно простым инструментарием, не загроможденным лишним функционалом по сравнению с другими системами разработки предметно-ориентированных языков, например, с DSL Tools или Eclipse GMF;
- возможность быстрой разработки демонстрационных прототипов системы.

Данный подход базируется на архитектурной модели Model/View/Controller, которая позволяет производить связывание визуальных объектов, созданных в среде Microsoft Visio, и объектов модели, хранящихся в репозитории.

Модель (Model) состоит из объектов, хранящихся в репозитории и являющихся теми объектами, которые визуализирует графический редактор создаваемого DSL. Все модельные объекты разбиты на классы, которые наследуются от абстрактного класса *ModelEntity*.

Графические объекты, созданные с помощью Microsoft Visio, составляют *представление* (View). Графические объекты не содержат информации о других частях DSM-пакета, за исключением уникальных

идентификаторов модельных классов, сущности которых они визуализируют, а также уникальных идентификаторов соответствующих модельных объектов, хранящихся в репозитории. Такая информация необходима для динамического создания репозиторных сущностей в момент размещения нового графического элемента на диаграмме, а также для создания связей между графическими и модельными объектами после загрузки ранее сохраненных диаграмм из репозитория проекта.

Роль *контроллеров* (Controller) играют потомки абстрактного базового класса *Controller*. Экземпляр контроллера создается для каждого графического объекта и связывает его с объектом, хранящимся в репозитории. Контроллер отвечает за обработку событий, происходящих с модельным или графическим объектом. Фактически, контроллер осуществляет контроль над соблюдением правил, определяемых синтаксисом DSL, при создании и модификации диаграмм.

Репозиторий отвечает за создание, хранение и доступ к модельным объектам. Также он позволяет выполнять сериализацию созданной модели. При этом информация о визуальном представлении модели хранится отдельно. Данные хранящиеся в репозитории используются генераторами кода и документации, компонентами валидации модели.

Процесс создания предметно-ориентированного языка с использованием данного подхода состоит из следующих этапов:

1) описание метамодели создаваемого DSL в виде диаграммы классов в специализированном редакторе классов на базе пакета MS Visio, созданном целиком «вручную»;
2) генерация исходного кода реализации объектно-ориентированного репозитория, генерация осуществляется на основе модели классов, созданной ранее;
3) описание графических объектов языка с помощью средств пакета MS Visio;
4) создание контроллеров для описания связи сгенерированного кода и графических элементов;
5) компиляция и отладка.

Для автоматической генерации исходного кода репозитория на основе метамодели DSL авторами подхода был создан специальный генератор репозиториев. При его разработке использовался стандарт ODMG, а также идеи, реализованные в рамках проекта Real-IT. Для генерации исходного кода была использована технология CodeDOM, что позволило генерировать код на различных языках платформы .Net,

для которых существует реализация абстрактного класса CodeDomProvider из пространства имен System.CodeDom.Compiler.

Генератор кода репозитория способен обрабатывать следующие элементы диаграммы классов: определения классов, определения свойств, отношения наследования, бинарные ассоциации, различные типы композитного агрегирования. В результате генерации порождается исходный код классов, типизированных коллекций и делегатов.

Сгенерированный репозиторий содержит необходимые конструкторы и методы для реализации расширенного механизма сериализации графических объектов, которая позволяет управлять порядком сохранения и восстановления графа объектов, дает возможность осуществлять выборочную сериализацию, а также избегать ошибок времени исполнения при различиях в версиях сохраненных объектов и измененных классах метамодели репозитория.

Кроме того, репозиторий имеет унифицированный набор событий, охватывающих различный функционал работы с репозиторием, например, корректное отображение репозиторных объектов в браузере проекта.

Сгенерированный репозиторий поддерживает ссылочную целостность отношений, описанных в метамодели за счет отношений ассоциации и композиции. Данный механизм был предложен в стандарте объектно-ориентированных баз данных ODMG. Помимо этого репозиторий позволяет производить каскадное удаление объектов. Так, например, в случае наличия композиции перед удалением репозиторного объекта будут сначала удалены отношения, связывающие удаляемый объект и связанные с ним объекты, затем все агрегируемые репозиторные объекты, а лишь потом сам объект.

Предлагаемый авторами подход был использован при создании DSM-пакета, предназначенного для автоматизации проектирования аппаратуры семейства систем телевещания. На основе пакета MS Visio был разработан графический редактор схем, реализован генератор Excel-отчетов по диаграммам, а также генератор загрузочной конфигурации программного обеспечения целевой системы [9].

Основным *преимуществом* данного подхода является то, что он позволяет существенно упростить процесс создания DSM-системы за счет использования пакета MS Visio, благодаря чему появляется возможность быстрой разработки демонстрационных прототипов систем. Кроме того, полученный инструментарий не перегружен функционалом, который используется профессиональными программистами и является довольно часто невостребованным при создании DSL.

Однако данный подход имеет и некоторые *недостатки*, так при его использовании отсутствует возможность изменения описания языка моделирования во время работы системы, для модификации языка необходимо изменить метамодель, перегенерировать исходный код, изменить контроллеры, откомпилировать и отладить проект. Для построения метамодели языка пользователь может использовать лишь диаграммы классов UML, что ограничивает его возможности. Кроме того, использование данной DSM-платформы без пакета Microsoft Visio невозможно, хотя данная система имеет меньшую стоимость, чем среда MS Visual Studio, но конечному пользователю все равно придется ее приобрести.

Методы алгоритмизации предметных областей

В работах [6, 7] автором предложен метод алгоритмизации предметных областей, который позволяет прикладным программистам решать на компьютере типовые задачи, не прибегая каждый раз к трудоемкому программированию на языках общего назначения. Алгоритмизация предметной области включает в себя следующие этапы [8]:

1) определение предметно-ориентированных структур данных, представляющих объекты предметной области;
2) реализация программных процедур обработки этих структур;
3) разработка методов и алгоритмов решения типовых задач предметной области – последовательностей действий, приводящих к необходимому результату.

Степень алгоритмизации может быть различной: чем больше типовых задач можно решить в данной предметной области, и чем меньше затраты на их решение, тем выше степень алгоритмизации. Степень алгоритмизации M определяется как среднее геометрическое двух показателей: доли решаемых типовых задач T (отношение числа типовых задач, которые можно решить средствами оцениваемой алгоритмизации, к числу всех идентифицированных типовых задач), и относительного снижения трудозатрат S (отношение средних трудозатрат на решение типовых задач после оцениваемой алгоритмизации к средним трудозатратам на решение той же совокупности задач до алгоритмизации): $M = \sqrt{T(1-S)}$.

Для построения предметно-ориентированных языков автором предлагается автоматный метод, основанный на метамоделях и системах взаимодействующих автоматных объектов. Данный метод позволяет определить все аспекты языка: структуру, внешнее представление,

контекстные условия и операционную семантику. Полное определение языка автоматным методом является реализацией языка.

Автоматный метод заключается в определении следующих четырех элементов языка:

1) абстрактного синтаксиса как иерархической композиции конструкций языка;
2) метамодели как абстрактного синтаксиса, дополненного системой неиерархических отношений между конструкциями языка;
3) конкретного синтаксиса как распознавателя, конструирующего абстрактную программу по её представлению;
4) операционной семантики как интерпретатора абстрактных программ.

В качестве средства определения всех составляющих языка используется UML: так для описания абстрактного синтаксиса и метамодели используются диаграммы классов, а для задания конкретного синтаксиса и операционной семантики используются системы взаимодействующих автоматов, заданные расширенными диаграммами автоматов UML.

Авторы подхода рекомендуют строить описание DSL последовательно, начиная с определения совокупности понятий языка и отношений классификации между ними, построенную модель они называют абстрактным синтаксисом. Далее следует описание информации, которой в традиционных грамматических структурах выразить затруднительно: различные контекстные условия и отношения между понятиями языка. За счет добавления к абстрактному синтаксису отношений ассоциации, обобщения и ограничений получается *метамодель*, которая исчерпывающим образом описывает структуру языка.

На третьем этапе следует задать две системы автоматов, первая система должна строить экземпляр полученной метамодели, а вторая – является семантическим интерпретатором экземпляра метамодели или генератором кода на целевом языке.

Таким образом, определение структуры проблемно-ориентированного языка необходимо, но не достаточно. Для использования языка, кроме абстрактной структуры, требуются конкретная реализация его абстрактной структуры – конкретный синтаксис и способ применения абстрактной структуры для конкретных целей – семантика языка. Соответствие, сопоставляющее программе ее смысл, называется семантикой программы. В автоматном методе семантика задается системой взаимодействующих автоматов, интерпретирующих экземпляр метамодели, при этом система автоматов взаимодействует, как с эк-

земпляром метамодели, получая из него информацию для интерпретации, так и с внешней средой, получая от нее запросы и команды и выдавая ответы.

Преимуществами данного подхода являются:

1) универсальность, которая состоит в том, что данный метод позволяет совместно и унифицированными средствами определить абстрактный синтаксис, конкретный синтаксис, контекстные условия и операционную семантику, при этом не требуется никаких дополнительных средств и формализмов;
2) возможность построения, как интерпретатора, так и генератора экземпляра метамодели;
3) наличие формального математического аппарата, позволяющего проверить корректность синтезированной программы.

К *недостаткам* данного подхода следует отнести:

1) отсутствие возможности выбора формализма построения абстрактного синтаксиса, поскольку в автоматном методе для этих целей используются диаграммы классов языка UML;
2) высокий уровень требований к конечному пользователю, поскольку он при разработке предметно-ориентированных языков должен не только владеть знаниями диаграмм классов языка UML, но и уметь строить конечные автоматы;
3) отсутствие возможности динамического изменения описания DSL;
4) отсутствие возможности изменения описания метаязыка.

Сравнение инструментариев создания предметно-ориентированных языков

На сегодняшний день существует несколько инструментариев, позволяющих создавать и использовать визуальные предметно-ориентированные языки в процессе построения информационных систем. Основная идея при этом заключается в описании абстрактного и конкретного синтаксиса языка и разработке визуального редактора. Каждый из рассмотренных выше языковых инструментариев обладает как достоинствами, так и недостатками.

Так, благодаря использованию подхода, основанного на интерпретации метамоделей, а не на генерации исходного кода системы, система MetaEdit+, в отличие от других инструментариев, позволяет вносить изменения в описание DSL во время работы системы и модифицировать метаязык. Кроме того, данный инструментарий позволяет интегрировать несколько языков в одной системе. К недостаткам MetaEdit+ можно отнести то, что эта DSM-платформа для экспорта моделей ис-

пользует свой собственный формат файлов (MXT), который отличается от общепризнанного стандарта – XML, что сказывается на открытости данной технологии.

Языковые инструментарии DSL Tools и Eclipse GMF помимо визуального редактора метамоделей предоставляют в распоряжение пользователя развитые среды программирования MS Visual Studio и Eclipse соответственно. Благодаря этому существует возможность «ручной» доработки кода на языках высокого уровня.

DSM-платформа Eclipse GMF является самой мощной из рассмотренных выше. Фактически все, что умеют выполнять другие платформы, умеет и она. Использование в качестве метаязыка стандарта MOF 2.0 и возможность доработки исходного кода инструментария делает данную систему открытой. Наличие инструментальной панели Dashboard упрощает процесс создания DSL, автоматизирует его и позволяет даже начинающему пользователю системы описать некоторый предметно-ориентированный язык. Помимо этого существуют дополнительные расширения системы Eclipse, позволяющие производить трансформацию моделей из одной нотации в другую, например, ATL [12] и QVT [21]. Данные расширения реализованы в виде плагинов, которые можно подключить к системе.

Использование технологии Eclipse GMF затруднено отсутствием документации, громоздкостью и сложностью, а также частым выходом новых версий. Фактически, GMF находится еще в стадии интенсивного развития. Первые версии технологии GMF не позволяли создавать DSL, которые могут работать отдельно от платформы Eclipse. Однако в последних версиях существует возможность создания независимых от Eclipse приложений. Но если появится необходимость внести какие-либо изменения в существующий DSL, то это возможно лишь при наличии платформы Eclipse.

Помимо упомянутой ранее проблемы, связанной с несогласованностью диаграмм классов и исходного кода, недостатком DSL Tools является и то, что данный инструментарий позволяет описывать DSL, которые могут быть использованы только в среде MS Visual Studio. Помимо этого, данный языковой инструментарий не позволяет производить изменения предметно-ориентированного языка без повторной генерации кода, а также определять DSL итеративно. Графический редактор, встроенный в эту платформу, не позволяет создавать компактные, удобные и наглядные диаграммы. А генерация исходного кода системы лишь на языках C#, VB значительно ограничивает круг разработчиков, которые могут использовать данную платформу для «ручной» доработки кода.

State Machine Designer по существу является лишь надстройкой над DSL Tools, поэтому данный подход наследует ряд недостатков данной DSM-платформы. State Machine Designer позволяет создавать DSL лишь с помощью диаграмм деятельности UML, что значительно сужает круг решаемых задач. Язык «ручной» доработки сгенерированного кода остается достаточно сложным для того, чтобы с ним могли работать пользователи – непрофессиональные программисты.

Многоплатформенная система QReal позволяет определять метамодели как в визуальном, так и в текстовом виде, поэтому разработчик имеет возможность выбрать наиболее подходящий для него формат представления описания языка. Наличие интерпретатора поведенческих диаграмм и отладчика сгенерированного кода ставит эту систему в один ряд с инструментариями DSL Tools Eclipse GMF, которые используют для этих целей развитые IDE. Но при этом данная система не позволяет изменять описание метаязыка, т.е. конечный пользователь должен довольствоваться возможностями, предоставляемыми ему MOF 2.0. Кроме того, в данной системе отсутствует возможность внесение изменений в описание DSL динамически.

Платформа, описанная в работах [3, 10], позволяет существенно упростить процесс определения DSL за счет использования привычной для большинства пользователей системы MS Visio. В данной системе отсутствует лишний функционал, который используется лишь профессиональными программистами и является довольно часто невостребованным при создании DSL. Однако данный подход имеет и некоторые недостатки, так при его использовании отсутствует возможность изменения описания языка моделирования во время работы системы: для модификации языка необходимо изменить метамодель, перегенерировать исходный код, изменить контроллеры, откомпилировать и отладить проект.

Алгоритмизация предметных областей позволяет унифицированными образом определять абстрактный и конкретный синтаксис, контекстные условия и операционную семантику описываемого языка. Наличие формального математического аппарата позволяет проверить корректность синтезированных программ, однако такой формализм предъявляет высокий уровень требований к конечному пользователю, поскольку он при разработке предметно-ориентированных языков должен не только владеть знаниями диаграмм классов языка UML, но и уметь строить конечные автоматы.

Нередки случаи когда DSL становятся частью других приложений, например, язык описания бизнес-процессов может быть интегрирован в систему документооборота, поэтому еще одной важной харак-

теристикой языка моделирования является его отчуждаемость (как физическая, так и правовая) от среды разработки. Следует отметить, что DSL, созданные с помощью инструментариев DSL Tools, MetaEdit+, QReal, сильно связаны с платформами разработки. Особенно технология DSL Tools, которая требует для работы не только многочисленные библиотеки времени исполнения, но и среду разработки, поскольку созданный язык может быть использован лишь в MS Visual Studio.

Ни одна из рассмотренных систем, за исключением MetaEdit+, не позволяет изменять описание метаязыка и, как следствие, итеративно определять языки моделирования. Кроме того, ни одна из технологий, за исключением MetaEdit+, не позволяет создавать динамически настраиваемые языки, поэтому для настройки языка на меняющиеся условия эксплуатации и потребности пользователя необходимо сначала внести изменения в исходный код проекта создания DSL, перекомпилировать этот проект, а лишь потом с помощью нового языка изменить саму модель предметной области.

Ни один из рассмотренных языковых инструментариев не позволяет производить трансформацию созданных моделей, за исключением расширений, созданных для системы Eclipse.

Устранение отмеченных недостатков DSM-платформ – цель разработки языкового инструментария MetaLanguage [22].

Библиографический список

1. Архитектура среды визуального моделирования QReal / А.Н. Терехов [и др.] // Системное программирование. – СПб., 2009. – Вып. 4. – С. 171-196.
2. *Иванов А.Н.* Технологическое решение REAL-IT: создание информационных систем на основе визуального моделирования // Системное программирование / под ред. проф. А.Н. Терехова и Д.Ю. Булычева. – СПб., 2004. – Вып. 1 – С.89-100.
3. Комплекс средств разработки проблемно-ориентированных визуальных языков / А.А. Павлинов [и др.] // Вестник Санкт-Петербургского университета. – СПб, 2007. – Сер. 10, Вып. 1 – С. 86-96.
4. *Кузенкова А.С.*, *Дерипаска А.О.*, *Литвинов Ю.В.* Поддержка метамоделирования в среде визуального программирования QReal // Материалы межвузовского конкурса-конференции студентов, аспирантов и молодых ученых Северо-Запада "Технологии Microsoft в теории и практике программирования". – СПб., 2011. – С. 100-101.

5. *Ларионов А.В.* Разработка визуального языка автоматного программирования на основе инструментов для создания специализированных языков предметной области среды разработки Microsoft Visual Studio 2005 [Электронный ресурс]. URL: http://is.ifmo.ru/papers/StateMachineDesigner.pdf (дата обращения: 11.12.2012).
6. *Новиков Ф.А., Тихонова У.Н.* Автоматный метод определения проблемно-ориентированных языков (Часть 1) // Информационно-управляющие системы. – СПб., 2009. – № 6. – С. 34-40.
7. *Новиков Ф.А., Тихонова У.Н.* Автоматный метод определения проблемно-ориентированных языков (Часть 2) // Информационно-управляющие системы. – СПб., 2010. – № 2. – С. 31-37.
8. *Новиков Ф.А., Тихонова У.Н.* Автоматный метод определения проблемно-ориентированных языков (Часть 3) // Информационно-управляющие системы. – СПб., 2010. – № 3. – С. 29-37.
9. Опыт создания визуальных средств проектирования для систем телевизионного вещания / Д.В. Кознов [и др.] // Системное программирование. – СПб, 2006. – Вып. 2 – С. 54-62.
10. О средствах разработки проблемно-ориентированных визуальных языков / А.А. Павлинов [и др.] // Системное программирование. – СПб, 2006. – Вып. 2 – С. 121-147.
11. *Подкопаев А.В., Брыксин Т.А.* Генерация кода на основе графической модели // Материалы межвузовского конкурса-конференции студентов, аспирантов и молодых ученых Северо-Запада "Технологии Microsoft в теории и практике программирования". – СПб., 2011. – С. 112-113.
12. ATL: Atlas Transformation Language // ATL Starter's Guide. – LINA & INRIA Nantes, 2005. – 23 p.
13. Creating Domain-Specific Languages [Электронный ресурс]. URL: http://msdn.microsoft.com/en-us/library/bb126259(v=vs.80).aspx (дата обращения: 11.12.2012).
14. Domain-Specific Development with Visual Studio DSL Tools / S. Cook [et al.]. – Reading : Addison-Wesley, 2007. – 560 p.
15. Eclipse Model-to-Model Transformation (M2M) [Электронный ресурс]. URL: http://www.eclipse.org/proposals/m2m (дата обращения: 11.12.2012).
16. Graphical Modeling Framework/FAQ [Электронный ресурс]. URL: http://wiki.eclipse.org/Graphical_Modeling_Framework_FAQ (дата обращения: 11.12.2012).

17. *Gronback R.C.* Eclipse Modeling Project: A Domain-Specific Language (DSL) Toolkit. – Reading : Addison-Wesley, 2009. – 706 p.
18. *Kelly S.* Comparison of Eclipse EMF/GEF and MetaEdit+ for DSM [Электронный ресурс]. URL: http://www.softmetaware.com/oopsla2004/kelly.pdf (дата обращения: 11.12.2012).
19. *Mazanek S.* Visual Languages. MetaEdit+ [Электронный ресурс]. URL: http://visual-languages.blogspot.com/2007/11/metaedit.html, свободный (дата обращения: 11.12.2012).
20. *Ozgur T.* Comparison of Microsoft DSL Tools and Eclipse Modeling Frameworks for Domain-Specific Modeling In the context of the Model-Driven Development [Электронный ресурс]. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.118.6383&rep=rep1&type=pdf (дата обращения: 11.12.2012).
21. *Stevens P.* Bidirectional Model Transformations in QVT: Semantic Issues and Open Questions // Lecture Notes in Computer Science. – 2007. – Vol. 4735/2007. – P. 1-15.
22. *Sukhov A.O., Lyadova L.N.* MetaLanguage: a Tool for Creating Visual Domain-Specific Modeling Languages // Proceedings of the 6th Spring/Summer Young Researchers' Colloquium on Software Engineering, SYRCoSE 2012. – Perm, 2012. – P. 42-53.
23. *Tolvanen J., Rossi M.* MetaEdit+: defining and using domain-specific modeling languages and code generators // Proceeding OOPSLA '03 Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications. – New York, 2003. – P. 92-93.