УДК 004.434

ЯЗЫК ОПИСАНИЯ ДИАГРАММ

Ф. А. Новиков,
канд. физ.-мат. наук, доцент
Санкт-Петербургский государственный политехнический университет
К. Б. Степанян,
архитектор проектов
ЗАО «Astrosoft Development»

Предлагается язык описания диаграмм, позволяющий формально определить графический синтаксис (нотацию) диаграмм заданного типа и связать нотацию с семантикой, заданной в форме набора классов. Язык может быть применен для автоматической генерации визуализаторов и графических редакторов диаграмм различных типов.

A Diagram Definition Language (DiaDeL) is proposed. The paper contains textual specifications for graphical notation and its binding to semantics. The semantics is defined as a set of classes. DiaDeL could be used for automatic generation of various diagram visualizers and editors.

Введение

В современной компьютерной индустрии для представления информации широко используются графические конструкции (схемы, диаграммы и т. п.). Особенно большое значение в последнее время приобрели различные диаграммы, используемые при моделировании, проектировании и реализации программного обеспечения. Разнообразие применяемых при этом визуальных образов довольно велико. В данной работе рассматриваются вопросы описания плоских (двумерных) графоподобных диаграмм, которые чаще всего применяются при разработке программ. Неформально графоподобная диаграмма — это диаграмма, состоящая из графических конструкций, которые можно разбить на два множества — множество фигур (вершин) и множество линий, соединяющих эти фигуры (ребер). Формальное определение подобных конструкций можно найти в работе [1].

Изначально подобные диаграммы предназначались только для рисования и чтения человеком. С развитием современных технологий они стали использоваться в целях автоматизации производства. Но инструментальное программное обеспечение, применяемое при работе с диаграммами, должно «понимать», что представлено на диаграмме. Для этого требуется строго формализовать нотацию и семантику всех графических конструкций, используемых на диаграммах данного типа. Если это условие соблюдено, диаграмму можно тем или иным способом интерпретировать, вследствие чего класс диаграмм данного типа становится визуальным (графическим) языком (например, UML, SDL, MSC). Вопрос о способе формализации описания семантики и нотации графического языка (употребляются также термины абстрактный и конкретный синтаксис соответственно) является одним из центральных в этой статье.

Семантика специфицирует смысловую нагрузку, а нотация — графическое представление конструкций визуального языка. Нотация зависит от семантики: правила построения графических конструкций на диаграмме определяются для семантических элементов языка. Для отображения диаграммы, описанной на визуальном языке, необходима как семантика (что отображаем), так и нотация (как отображаем). Но для иного использования диаграммы, например для генерации кода, достаточно иметь только семантическую составляющую.

В наиболее типичном случае использование конкретного типа диаграмм поддерживается специализированным графическим редактором. Последний выполняет две основные функции: поддерживает соответствие между семантикой и нотацией, визуально отображая изменения в семантической части диаграммы, и поддерживает соответствие между нотацией и семантикой, транслируя визуальные манипуляции с диаграммой в изменения ее смысла. Важно подчеркнуть, что в общем случае эти соответствия не являются взаимнооднозначными: заданный смысл диаграммы может быть изображен различными графиче-

скими конструкциями и, наоборот, разные визуальные манипуляции с изображением диаграммы могут иметь один и тот же смысл.

Важной особенностью современной практики применения диаграмм при разработке программного обеспечения является их «непостоянство». Различные стандарты, инструменты крупных производителей и даже отдельные разработчики изображают и понимают немного по-разному диаграммы с одним и тем же названием. Дело в том, что использование диаграмм в программировании еще не устоялось до такой же степени, как применение чертежей в технике. Со временем унификация, наверное, произойдет, но сейчас для эффективного использования диаграмм при разработке необходимо иметь возможность менять как нотацию, так и семантику буквально «на ходу».

Таким образом, в настоящее время актуальными являются задачи создания адекватных средств описания семантики и нотации диаграмм различных типов и разработки гибких инструментальных средств поддержки работы с диаграммами в разнообразных областях применения.

Обзор существующих решений

Вопросу описания диаграмм и генерации графических редакторов посвящено множество статей и разработок. Все они в качестве результата генерируют редактор диаграмм по заданной спецификации. Ниже рассмотрены некоторые разработки, наиболее примечательные с точки зрения авторов.

Среди всех работ можно выделить множество, в котором нотация диаграммы задается в виде начального алфавита символов (или графических примитивов) и порождающей графической грамматики. К указанному классу работ относятся системы CIDER [2], TIGER [3] и DiaGen [4]. Большим плюсом такого подхода является возможность описывать диаграммы, которые не подпадают под определение графоподобных (например, диаграммы Нейси–Шнейдермана). В то же время, используя подобный способ задания нотации диаграммы, затруднительно описывать сложные конструкции (например, составное состояние в диаграмме состояний или класс в диаграмме классов UML). Две первые системы не позволяют описывать подобные конструкции и редактировать их в сгенерированном редакторе. Причиной данного ограничения авторы считают невозможность указать с помощью используемых в системах грамматик отношение включения одной графической конструкции в другую при описании нотации.

Другим недостатком названных систем является использование своих собственных уникальных семантических моделей. Следовательно, при необходимости использовать внешнюю систему с заданной семантической моделью встает вопрос об интеграции сгенерированного редактора и внешней системы. Подобная интеграция подразумевает трансформацию семантической модели, что является трудной и не всегда выполнимой задачей.

Одной из основ системы ***CIDER*** является метафора *«интеллектуальная диаграмма»*. По мнению авторов системы, редактор должен допускать синтаксически некорректные конструкции для удобства редактирования. Поэтому сгенерированные редакторы оперируют конструкциями не в семантических терминах, а в терминах графических примитивов, что имеет свои недостатки. Например, для удаления сложной графической конструкции необходимо удалить по порядку все ее примитивные составляющие.

Система ***TIGER*** является новой версией проекта GenGED [5] и, как следствие, основана на тех же идеях. Главным нововведением является то, что грамматика задается в графическом виде. Такой подход позволяет на диаграмме за одну операцию создавать конструкции, состоящие более чем из одного семантического элемента. Другой особенностью является наличие атрибутов у правил грамматики. Один из них является действием, которое описывает правило. Допустимо четыре типа действий: создание, редактирование, движение и удаление.

Данная система реализована как подключаемый модуль системы Eclipse. Генерируемые редакторы диаграмм также являются подключаемыми модулями для Eclipse. Это придает им гибкость с точки зрения интеграции с другими модулями Eclipse.

В данной статье мы остановимся только на одной отличительной особенности системы ***DiaGen***, которая связана со спецификацией нотации для диаграммы. Подход для описания диаграммы, используемый в системе DiaGen и в предлагаемом нами языке DiaDeL, имеет под собой одинаковую основу, а именно: разделение диаграммы на абстрактный (семантическая модель) и конкретный (нотация) синтаксис. Отличие от систем CIDER, TIGER и GenGED заключается в том, что кроме спецификации графических примитивов, в DiaGen пользователь также задает графические отношения (например, «внутри» или «касается»), что придает системе большую гибкость и свободу в описании нотации.

Особого внимания заслуживает проект ***GMF*** [6]. Система GMF является подключаемым модулем среды Eclipse и представляет собой мощный инструмент, позволяющий автоматически генерировать редакторы для диаграмм. Подобно подходу, реализованному как в системе DiaGen, так и в предлагаемом языке DiaDeL, GMF разделяет описание семантики и нотации. Спецификация для генерации редактора в среде GMF состоит из следующих частей: описание семантической модели; описание нотации; описание инструментов для работы с сущностями диаграммы; связывание трех описаний воедино. Описание семантической модели и сама семантическая модель должны быть ре-

ализованы на основе библиотеки EMF, что позволяет визуализировать уже существующие модели. Это очевидное преимущество одновременно является существенным недостатком. По сути, GMF может быть использована только в рамках системы Eclipse и не может быть рассмотрена как независимая и сколько-нибудь универсальная система отображения диаграмм.

Основные концепции

Данная статья посвящена языку описания диаграмм DiaDeL (Diagram Definition Language), который позволяет специфицировать нотацию графоподобной диаграммы и связать ее с уже существующей семантикой.

Семантическая модель. Язык DiaDeL основан на предположении, что семантика каждой отдельной диаграммы задана в виде набора конкретных программных объектов определенных классов. Набор объектов, соответствующих конкретной диаграмме, называется ее семантической моделью, а набор всех возможных классов объектов, которые могут появляться на диаграммах данного типа, называется метамоделью. Фактически метамодель описывает абстрактный синтаксис визуального языка.

Подобный подход является гибким в смысле разделения методов компьютерной обработки информации и ее графического представления для пользователя. Зачастую семантическая метамодель задана стандартом, а методы обработки семантических моделей уже существуют (реализованы в каком-либо инструменте), в то время как графическое представление либо определено недостаточно формально, либо у пользователей возникает требование динамически менять графическое представление в соответствии со своими предпочтениями. В таких случаях использование существующих решений (таких, например, как Microsoft Visio) для построения и отображения диаграмм подразумевает интеграцию с заданной метамоделью и методами ее обработки, что не всегда возможно и требует значительных трудозатрат почти всегда, когда допустимо. Также затруднительно использовать генераторы редакторов диаграмм, поскольку они самостоятельно генерируют абстрактный синтаксис визуального языка, который может быть не связан с заданной метамоделью, что потребует программирования трансформаций. Предлагаемый язык имеет целью преодоление этих трудностей.

Основным назначением языка DiaDeL является:

- описание нотации (графического синтаксиса) диаграмм и
- описание связи нотации с существующей семантикой.

Нотация диаграммы задается в виде описания графических конструкций и графических отношений между ними. Связывание нотации с семантикой является ключевым моментом в описании диаграммы. Для этого элементам из семантической модели, которые должны быть представлены визуально, сопоставляются графические конструкции.

Семантическая метамодель должна соответствовать ряду требований для возможности ее использования в DiaDeL-описании.

1. Метамодель должна представлять собой набор классов или интерфейсов.

2. Среди классов метамодели должен быть выделен корневой класс, который соответствует диаграмме в целом. В любой конкретной семантической модели диаграммы объект этого класса должен быть единственным.

3. Классы метамодели должны быть связаны отношениями ассоциации таким образом, чтобы от корневого элемента можно было осуществить навигацию к любому другому элементу.

4. Один элемент семантической модели может быть представлен с помощью одной и только одной графической конструкции. Одна и та же графическая конструкция может являться представлением нескольких разных элементов модели.

5. Элементы семантической модели должны предоставлять возможность извлечения информации, необходимой для построения диаграммы.

Указанные ограничения являются довольно сильными, но, к счастью, большая часть семантических метамоделей, построенных для различных типов диаграмм, им удовлетворяет. Например, для известных метамоделей диалектов UML ограничения 1 и 3–5 выполнены. Если ограничение 2 не выполнено, то нетрудно доработать метамодель, добавив соответствующий корневой класс.

Способы использования языка. Возможны несколько вариантов использования языка DiaDeL.

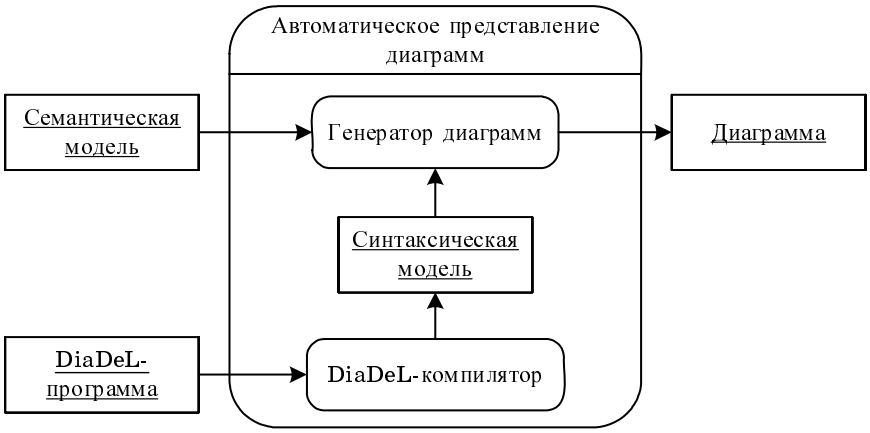
1. Построение системы автоматического представления диаграмм – системы, которая отображает диаграмму, используя описание на языке DiaDeL и экземпляр семантической модели (рис. 1).

2. Построение синтезатора визуализаторов диаграмм – системы, которая генерирует код визуализатора диаграмм с описанной на DiaDeL нотацией и для указанной семантической модели (рис. 2).

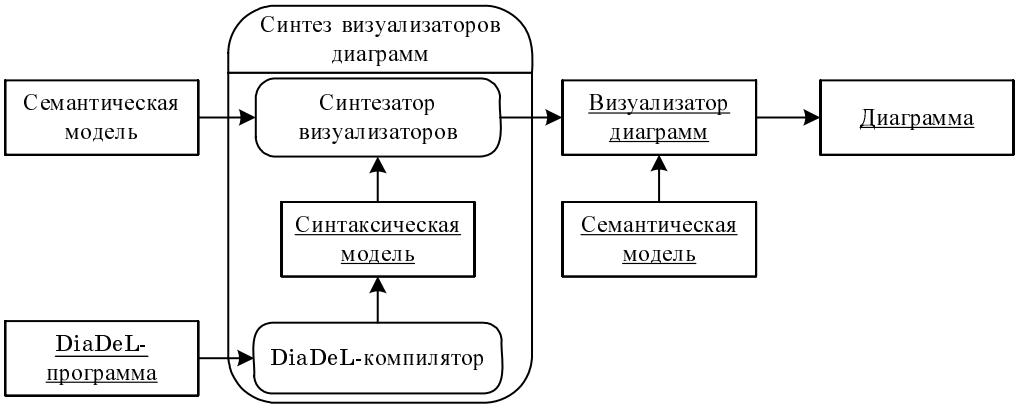
Система автоматического представления диаграмм состоит из двух компонент: компилятора языка DiaDeL и генератора диаграмм. Компилятор производит компиляцию описания диаграммы из конкретного синтаксиса в синтаксическую модель. Генератор диаграмм, используя скомпилированную синтаксическую модель и экземпляр семантической модели, генерирует экземпляр диаграммы.

Синтезатор визуализаторов диаграмм работает по аналогичной схеме, только вместо генератора диаграмм в ней находится синтезатор визуализаторов, который на вход получает саму семантическую модель, а на выходе строит синтезированный код визуализатора.

Приведенные способы использования предлагаемого языка DiaDeL не исчерпывают всех воз-



■ ***Рис. 1.*** *Схема работы системы автоматического представления диаграмм*



■ ***Рис. 2.*** *Схема работы системы, синтезирующей визуализаторы диаграмм*

можностей, здесь приведены два наиболее важных, с точки зрения авторов. Описание остальных способов использования DiaDeL мы оставляем за рамками статьи.

Особенности языка. Предлагаемый язык разработан с учетом имеющихся решений данной проблемы и нацелен на преодоление определенных недостатков этих решений. Описанная выше архитектура языка базируется на принятии некоторых основных положений, каждое из которых имеет как свои плюсы, так и минусы. Опишем их подробнее.

1. Ограниченный набор графических конструкций. Язык DiaDeL позволяет описывать сравнительно небольшое количество графических конструкций, но уровень этих конструкций достаточно высок. Набор выбранных графических примитивов был определен путем анализа диаграмм тех типов, которые чаще всего используются при разработке программного обеспечения в настоящее время. Например, в число основных конструкций включена «рамка» – конструкция, которую трудно отнести к традиционным графическим примитивам, но которая часто используется в современных визуальных языках. Заметим, что в более ранних языках, например в граф-схемах алгоритмов, подобные конструкции не применялись. Достоинством данного решения является удобство описания самых распространенных типов диаграмм, недостатком – отсутствие универсальности.

2. Простой текстовый синтаксис. Для языка DiaDeL определен текстовый синтаксис в традиционном стиле (см. раздел «Описание языка» статьи). Альтернативными способами описания типов диаграмм могли бы быть, например, графические грамматики, визуальные примеры, манипулирование классами модели с помощью различных конструкторов. Выбранное средство (формальный текстовый язык) является наиболее привычным для основной категории пользователей DiaDeL – разработчиков программного обеспечения и в то же время одним из самых простых в реализации. Недостатки текстовых языков общеизвестны, прежде всего, они сравнительно многословны и требуют предварительного изучения перед использованием. Заметим, что в языке DiaDeL используется единый синтаксис как для описания нотации, так и для описания связи нотации с семантикой, что отличает DiaDeL от, например, системы DiaGen.

3. Динамическая связь нотации и семантики. Важным допущением, положенным в основу языка DiaDeL, является предположение о том, что средство описания связи семантики и нотации (семантические мосты, см. раздел «Семантические мосты») имеет «двустороннее действие». Мы предполагаем, что заданная извне семантическая модель обеспечивает не только получение информации, достаточной для отображения диаграммы, но и сама модель позволяет себя менять извне теми же средствами. Это, очевидно, открывает путь к применению «живых» диаграмм, графические манипуляции с которыми отражаются в семантической модели, что является несомненным достоинством. С другой стороны, если используемая семантическая модель не поддерживает «двустороннее» действие, описание диаграммы на языке DiaDeL невозможно «оживить», поскольку DiaDeL *не* является языком программирования семантических моделей.

Независимость от платформы и языка программирования. Синтаксис языка DiaDeL спроектирован независимо от какого-либо языка программирования и платформы, на которой строится визуализатор диаграмм, описанных на DiaDeL. Это делает возможным построение визуализатора с использованием различных средств и платформ, таких как Java, Eclipse, .NET и т. д. Это выгодно отличает его от большинства существующих решений, которые «привязаны» к конкретной платформе или языку программирования. Например, упомянутые выше решения TIGER и GMF жестко привязаны к Eclipse. Вследствие чего описание диаграммы возможно задать только в среде Eclipse или программно, но используя библиотеки Eclipse. Часть описания диаграммы в системе DiaGen производится на Java, что привязывает систему к целевому языку при генерации редактора.

Описание языка

Язык DiaDel имеет вполне традиционный синтаксис. Ниже приведен фрагмент формальной порождающей грамматики языка DiaDeL в стандартной форме Бекуса—Наура. Терминальные конструкции набраны полужирным шрифтом, пустая цепочка обозначена ε.

```
Program ::= Diagram DiagramDefs
Diagram ::= diagram id { DiagramDefBody }
DiagramDefs ::= EntityDef DiagramDefs | ε
EntityDef ::= Figure | Line | Text   | Decoration
                                     | Frame | SemBridge
Figure ::= figure id { FigureDefBody }
Line ::= line id { LineDefBody }
Text ::= text id { TextDefBody }
Decoration ::= decoration id { DecorationDefBody }
Frame ::= frame id { FrameDefBody }
SemBridge ::= bridge id { BridgeDefBody }
```

Формальное описание дальнейших синтаксических деталей опущено для экономии места. Далее основные конструкции языка рассмотрены на примерах.

Конструкция diagram. В описании нотации конкретного типа диаграмм должна присутствовать одна и только одна конструкция с типом **diagram** (диаграмма). Данная конструкция описывает тип диаграммы в целом и задает для нее допустимые графические конструкции.

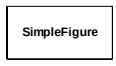
Пример описания диаграммы классов UML:

```
diagram classdiag {
        contain = {class, interface, generalization,
                   implementation, association,
                   dependency};
}
```

Конструкция figure. Фигура — основной элемент для конструирования диаграмм. В графоподобной диаграмме фигура представляет вершину (сущность отображаемой модели), размером и положением которой управляет пользователь. Фигура бывает простой, составной и контейнером. Все они описываются с помощью одного ключевого слова **figure**.

Пример описания простой фигуры class, представленной прямоугольником с текстом внутри фигуры:

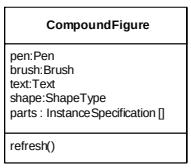
```
figure class {
    // геометрический примитив
    shape = rectangle;
    // свойства пера для рисования примитива
    pen.style = solid;
    pen.color = black;
    pen.width = 1;
    // свойства кисти для рисования
       примитива
    brush.style = solid;
    brush.color = white;
    // расположение текста относительно примитива
    text_position = inside;
}
```



Составная фигура и контейнер являются сложными фигурами. Отличие от простой фигуры заключается в определении отношения включения в сложную фигуру других конструкций. Для составной фигуры отношение включения определяет части фигуры, из которых она состоит (свойство **parts**). Экземпляры этих конструкций создаются вместе с созданием составной фигуры.

Пример составной фигуры compound_class, которая включает в себя три части (ns, fs, os), расположенные вертикально друг под другом так, чтобы заполнить всю область родительской фигуры (этот пример соответствует нотации классификаторов на диаграммах классов UML):

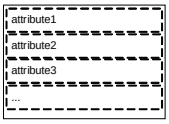
```
figure compound_class {
    ...
    // части составной фигуры
    parts = {ns : name_sec, // имя
         fs : atributes_sec,// атрибуты
         os : operations_sec};// операции
    // расположение частей
    layout = vertical;
}
```



Конструкция **diagram** должна содержать декларацию для каждой части: name_sec, attributes_sec, operations_sec, т. е. типы частей должны быть объявлены.

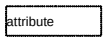
Секция атрибутов attributes_sec, в свою очередь, является контейнером. Для контейнера отношение включения определяет конструкции, которые могут быть отображены внутри контейнера (свойство **contain**). Список экземпляров этих конструкций формируется динамически во время построения и обновления диаграммы. Таким образом, два различных экземпляра одной фигуры-контейнера могут включать различные наборы экземпляров графических конструкций:

```
figure attributes_sec {
    shape = rectangle;
    pen.style = solid;
    pen.color = black;
    pen.width = 1;
    contain = {attribute};
    layout = vertical;
}
```



Конструкция text. В рассматриваемом примере секция атрибутов является контейнером для атрибутов, которые представляются в виде текста. Конструкция для представления текста объявляется с помощью ключевого слова **text**. Она позволяет определить шрифт (свойство **font**), перенос слов (свойство **word_wrap**) и выравнивание для отображаемого текста (свойства **hor_align** и **vert_ align**):

```
text attribute {
        // шрифт и его свойства отображения
        font.name = "Arial";
        font.color = black;
        font.size = 10;
        // слова не переносятся
        word_wrap = false;
        // выравниевание слева по горизонтали
        hor_align = left;
        // по центру по вертикали
        vert_align = center;
}
```



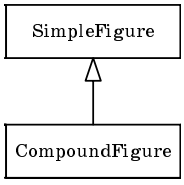
Конструкция line. Линии на диаграмме представляют семантические отношения между сущностями модели. Поэтому у непротиворечивой диаграммы все концы линий инцидентны двумерным графическим конструкциям.

Линия описывается с помощью ключевого слова **line**. Описание линии содержит декларацию пар конструкций, которые линия соединяет на диаграмме (свойство **links**). Для каждой пары указывается допустимая кратность этой линии.

Пример линии generalization (отношение обобщения):

```
line generalization {
    links = {(class, class, 1),
             (interface, interface, 1)};

    attachments[100%] =
        (:gen_dec)[100%,50%];
}
```



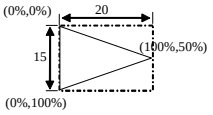
Представленная линия может соединять два экземпляра конструкции class или interface. Между ними может быть только один экземпляр линии generalization. Треугольник на конце является декорацией, прикрепленной к линии (свойство **attachments**, подробнее см. ниже).

Конструкция decoration. Для отображения графических элементов с фиксированными размерами используется конструкция декорация. Конструкция декорация очень похожа на простую фигуру, но имеет ряд отличительных черт: размер декорации декларируется в описании и не может быть изменен при отображении диаграммы; декорация не может иметь прикреплений, но ее экземпляры могут быть прикреплены к другим конструкциям (см. ниже описание отношения прикрепления); декорация может быть представлена картинкой.

Основное назначение данной конструкции – украшение других конструкций. Но она также может представлять сущность семантической модели (например, интерфейс на диаграмме классов UML, отображаемый в виде кружка).

Пример описания декорации для линии, представляющей отношение обобщения:

```
decoration gen_dec {
    minsize = (20,15);
    shape = polygon((0%,0%),
            (0%,100%), (100%,50%));
}
```



Поскольку декорация имеет фиксированный размер в пикселях, он должен быть задан в описании (свойство **minsize**). Данный пример также иллюстрирует применение полигона как графического примитива. В скобках через запятую указываются узловые точки, положение которых задается относительно размеров конструкции. Положение (0 % ,0 %) – это левый верхний угол конструкции, положение (100 % , 100 %) – правый нижний.

Отношение прикрепления. Для создания комплексных конструкций используется отношение прикрепления, декларирующее, что к родительской конструкции крепится экземпляр другой конструкции. Термин «крепится» несет следующий смысл: положение экземпляра на диаграмме однозначно определяется с помощью положения родительской конструкции и свойств прикрепления, причем прикрепленный экземпляр рисуется поверх родительской конструкции. В роли родительской конструкции могут выступать фигуры, рамки и линии. К ним можно прикреплять экземпляры декораций и текстовых конструкций.

Декларация прикрепления находится в родительской конструкции и имеет следующий синтаксис:

```
attachments[<МТочка>]=(<имя экз>:
                        <имя констр>)[<СТочка>];
```

где <имя экз>:<имя констр> – объявление экземпляра конструкции, который прикрепляется (имя экземпляра может быть опущено); <МТочка> — точка родительской фигуры, к которой соверша-

ется прикрепление; <СТочка> – точка экземпляра, которой он крепится.

При отображении экземпляр размещается таким образом, чтобы указанные точки совпадали. Способ задания точек такой же, как и в случае описания полигона. Следует также отметить, что указание точек для линий и двумерных конструкций отличается. Поскольку линия – это одномерная конструкция, то для указания ее <МТочки> используется только одна координата, которая указывает положение вдоль линии.

Пример конструкции описания прикрепления для линии generalization:

```
line generalization {
    ...
    attachments[100%] =
        (:gen_dec)[100%,50%];
}
```

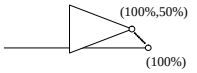


Иллюстрация справа показывает, как должна располагаться декорация относительно линии в приведенном примере. При повороте линии прикрепленные к ней декорации должны поворачиваться соответствующим образом.

Конструкция frame. Для обрамления и группирования элементов диаграммы используется графическая конструкция рамка (ключевое слово **frame**). По существу, рамка является фигурой, и к ней применимы все те же правила, но со следующими отличиями: расположение рамок на диаграмме не подчиняется правилам расположения других конструкций; рамка рисуется поверх всех других двумерных конструкций, но не линий (для того чтобы она не перекрывала обрамляемые конструкции при отображении, ее делают прозрачной); отношение включения в рамку других конструкций допустимо только для секций.

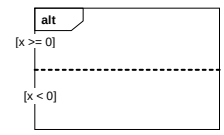
Секция – это специальный тип простой фигуры, который может быть включен в рамку. Рамка может не включать секции (аналогично простой фигуре), включать фиксированное число секций (аналогично составной фигуре, свойство **parts**) и включать изменяемое число секций (аналогично контейнеру, свойство **contain**). Секция для рамки объявляется с использованием ключевого слова **section** и полностью идентична простой фигуре.

Пример описания рамки для представления составных шагов взаимодействия на диаграмме последовательности UML 2.0:

```
frame fragment {
      shape = rectangle;
      pen.style = solid;
      pen.color = black;
      pen.width = 1;
      brush.style = none;

      attachment[0%,0%] =
          (name:alt_dec)[0%,0%];

      layout = vertical;
      contain = {operand};
}
```




```
section operand {
      shape = polyline((0%,0%), (100%,0%));
      pen.style = dash;
      brush.style = none;

      attachment[0%,30%]=
          (cnd:cond_txt)[50%,50%];
}
```

В верхнем левом углу она имеет прикрепленную декорацию для обозначения типа составного шага (на рисунке это рамка с меткой **alt**). Рамка состоит из неопределенного числа секций, разбивающих ее горизонтально. Секции представляют операнды. Каждая секция имеет прикрепленный к ней текст.

Семантические мосты. Для связывания графических конструкций с элементами семантической модели используются семантические мосты. Для одной графической конструкции может быть объявлено несколько семантических мостов, т. е. одна конструкция может представлять на диаграмме разные семантические элементы. Но семантический элемент может быть представлен на диаграмме только одной графической конструкцией.

Декларация семантического моста имеет следующий синтаксис:

```
bridge <имя конструкции> {
    <декларация связей с семантическим элементом>
    <управление отображением>
}
```

Тело декларации моста состоит из двух частей: декларации связей с семантическим элементом и секции управления отображением (см. ниже). Декларация связей является перечислением свойств или методов семантического элемента, которые предоставляют информацию, необходимую для построения экземпляра графической конструкции. Очевидно, что для построения экземпляров разных типов необходима различная по содержанию информация. Например, для отображения экземпляра фигуры контейнера необходимо иметь информацию об элементах, которые включены в нее. А для отображения линии необходимо иметь информацию о ее полюсах.

Дадим описание всех допустимых вариантов декларации связи с указанием, к каким типам конструкций они применимы.

model – объявление элемента (класса или интерфейса) семантической модели, с которой связывается конструкция данным мостом. Эта связь должна быть описана в любом мосту вне зависимости от типа конструкции, для которого он декларируется.

properties – объявление множества свойств элемента, изменение которых влечет обновление диаграммы на экране. Эта связь может быть описана в любом мосту, но не является обязательной.

edges – объявление свойства семантического элемента, используемого для извлечения отношений этого элемента. Данное свойство применимо ко всем двумерным графическим конструкциям,

т. е. к фигуре (любого типа), декорации, рамке и секции.

start, end – объявление свойств семантического отношения, используемых для получения его полюсов. Данные объявления допустимы только в мостах для линий.

children – объявление свойства семантического элемента, используемого для извлечения информации о других элементах, содержащихся в данном. Данная декларация допустима только для диаграммы, контейнера, рамки.

inside – объявление свойства семантического элемента, используемого для извлечения информации о элементах, которые должны быть обрамлены рамкой (или ее секцией). Данное объявление допустимо только для рамок или секций.

Корректное описание типа диаграммы на языке DiaDeL содержит семантические мосты не только для графических конструкций, но и мост для конструкции **diagram**. Он используется для определения элементов, которые должны быть отображены на диаграмме.

Пример семантического моста для диаграммы классов UML:

```
bridge <classdiag> {
        model := org.eclipse.uml2.uml.Package;
        children := model.getPackagedElements();
}
```

Управление отображением. Семантический мост также может содержать блок управления отображением. Этот блок позволяет изменять атрибуты представления графической конструкции в зависимости от состояния семантического элемента. Данная возможность особенно удобна, когда графическая конструкция используется для отображения нескольких семантических элементов.

Блок управления отображением вводится ключевым словом **refresh** и состоит из набора операторов. Допустимы два типа операторов: присваивание и условный оператор. Операторы выполняются каждый раз, когда строится или обновляется представление диаграммы.

Пример семантического моста для составной фигуры compound_class:

```
bridge <compound_class> {
        model := org.eclipse.uml2.uml.Class;
        edges := model.getRelations();
        refresh() {
                ns.text.value = model.getName();
                if (model.isAbstract())
                        ns.text.font.italic = true;
        }
}
```

Блок управления отображением содержит два оператора. Первый оператор извлекает из модели имя отображаемого элемента (имя класса) и присваивает его значение тексту части, отображающей имя. Второй оператор условный: в случае, если фигура представляет абстрактный класс, то текст в секции имени отображается курсивом.

Следует обратить внимание, что, используя ключевое слово **model**, допустимо обращаться к свойствам и методам семантического элемента (в примере **model**.getName() и **model**.isAbstract()). Обращение к свойствам или частям графической конструкции выполняется без каких-либо префиксных ключевых слов (ns.**text.value** — обращение к значению текста секции имени фигуры class).

Особым случаем является объявление семантического моста для конструкции, которая является частью составной фигуры. В этом случае связь с семантическим элементом выводится из связи родительской фигуры, для чего используется ключевое слово **parent**.

Пример семантического моста для секции attributes_sec составной фигуры class:

```
bridge <attributes_sec> {
        model := parent.model;
        children := model.getOwnedAttributes();
}
```

Пример описания диаграммы классов UML

Приведем пример описания упрощенной нотации для диаграммы классов UML (рассматриваются только классы и обобщения) и ее связь с существующей метамоделью UML, реализованной в виде библиотеки Eclipse.

```
// объявление диаграммы
diagram classdiag {
      contain = {class, generalization}; // содержащиеся
                                                конструкции
}
// объявление фигуры класса
figure class {
      shape = rectangle; // графический примитив –
                                              прямоугольник
}
// объявление линии обобщения
line generalization {
      links = {(class, class, 1)}; // допустимые полюса
                                                и кратность
      attachments[100%] = (:gen_dec)[100%,50%]; //
                                    прикрепление декорации
}
// объявление декорации в виде замкнутого треугольника
                                        для линии обобщения
decoration gen_dec {
      minsize = (20,15); // размер конструкции
      shape = polygon((0%,0%), // графический примитив –
                                                  ломаная
          (0%,100%), (100%,50%));
}
// объявление моста для диаграммы
bridge <classdiag> {
      model := org.eclipse.uml2.uml.Package;
      children := model.getPackagedElements(); //
                                        элементы диаграммы
}
// объявление моста для фигуры класса
bridge <class> {
      model := org.eclipse.uml2.uml.Class;
      edges := model.getGeneralizations(); //
                                    отображаемые отношения
      refresh() {
            text.value = model.getName(); // извлечение
                                                имени класса
            if (model.isAbstract()) // если класс
                                                абстрактный
```

```
            text.font.italic = true; // отображать курсивом
        }
}
// объявление моста для линии обобщения
bridge <generalization> {
      model := org.eclipse.uml2.uml.Generalization;
      start := model.getSpecific(); // начало линии
      end := model.getGeneral();      // конец линии
}
```

Заключение

В статье представлен язык описания диаграмм DiaDeL [7]. Предложенный язык позволяет описать нотацию для диаграммы при условии, что семантика диаграммы (абстрактный синтаксис) задана изначально. Это дает возможность визуализировать уже существующую информацию в виде диаграмм и является выгодным отличием от большинства подобных решений (DiaGen, TIGER, CIDER). На семантику накладывается ряд ограничений, но они более слабые, чем те, которые требует GMF. Более того, DiaDeL не накладывает ограничений на язык и платформу семантической модели.

Язык DiaDeL включает в себя описание представления в виде графических конструкций и связывание графических элементов с элементами семантической модели. Представлением элемента допустимо управлять динамически. Это придает языку гибкость в смысле описания визуального представления, которое должно динамически меняться в зависимости от состояния семантического элемента. Язык DiaDeL предоставляет возможность описывать сложные конструкции (такие как составные фигуры, контейнеры и рамки), что не предусматривается в системах CIDER, TIGER и GenGED.

Авторы уверены, что использование языка DiaDeL позволит расширить область практической применимости диаграмм различных типов, используемых при разработке программного обеспечения.

Литература

1. **Жоголев Е. А.** Графические редакторы и графические грамматики // Программирование. 2001. № 3. С. 30–42.
2. **Chok S. S., Marriott K.** Automatic Generation of Intelligent Diagram Editors // ACM Transactions on Computer-Human Interaction. September 2003. Vol. 10. N 3. P. 244–276.
3. **Ehrig K., Ermel C., Hänsgen S., Taentzer G.** Generation of visual editors as eclipse plug-ins // ACM Intern. Conf. on Automated Software Engineering: Proc. 20th IEEE // IEEE Computer Society. Long Beach, California, USA. 2005.
4. **Minas M., Viehstaedt G.** DiaGen: A Generator for Diagram Editors Providing Direct Manipulation and Execution of Diagrams: Proc. IEEE Symp. on Visual Languages. September 5–9. Darmstadt, Germany. 1995. P. 203–210.
5. **Bardohl R.** GenGED – Visual Definition of Visual Languages based on Algebraic Graph Transformation: Дис. ... д-ра. Технический университет Берлина. 1999.
6. **Eclipse Consortium.** Eclipse Graphical Modeling Framework (GMF) – Version 1.0.2. http://www.eclipse.org/gmf. 2006.
7. **Степанян К. Б.** Язык описания диаграмм // Научно-технические ведомости СПбГПУ. 2006. № 6-1. С. 36–41.