Санкт-Петербургский Государственный Университет

Математико-механический факультет

Кафедра системного программирования

Обзор подходов к созданию визуальных языков программирования

Курсовая работа студентки 361 группы
Гудошниковой Анны Андреевны

Научный руководитель: ст.преп. Ю.В.Литвинов

Санкт-Петербург
2012

Содержание

Введение.....3
Основные понятия.....5
Классификационная модель.....7
DSM-платформы.....15
Результаты.....25
Список литературы.....26

Введение

Процесс разработки программного обеспечения всегда требовал больших затрат, в частности, на глубинное изучение предметной области, для которого нужны большие человеческие ресурсы и время. Сейчас большое внимание уделяется визуальному программированию, которое позволяет упростить создание программы путем представления отдельных ее аспектов в виде диаграм. Это позволяет не расссматривать всю предметную область целиком, а сконцентрироваться на какой-то отдельной ее части.

Для реализации такого способа разработки программного обеспечения существуют Computer-Aided Software Engineering (CASE) системы. Они предоставляют средства генерации кода и используются в качестве инструментов для разработки, анализа и проектирования программного обеспечения. В CASE-системах используются различные визуальные языки. На этих языках строятся различные модели для описания компонент программного обеспечения.

Для создания таких CASE-систем в некоторых случаях применяются metaCASE-системы. Такие системы позволяют автоматически генерировать CASE-системы, задавая описание предметно-ориентированного языка, который будет использоваться в CASE-системе.

Язык, с помощью которого описывается предметно-ориентированный, или просто визуальный язык, называется метаязыком. На метаязыке строятся различные модели для описания визуального языка. Такие модели называются метамоделями.

Примером metaCASE-системы может служить система QReal - разработка научно-исследовательской группы кафедры системного программирования СПбГУ. QReal предоставляет средства, с помощью которых можно сгенерировать код визуальных редакторов по описаниям их метамоделей.

На практике процесс создания визуального языка вызывает огромные трудности, потому что не хватает знаний о том, как это делается. Приходится читать различную документацию, что занимает очень много времени.

В настоящее время в QReal не поддерживаются конкретные методологии создания визуального языка. Таким образом, процесс создания визуального языка занимает много времени.

Поэтому важной задачей является рассмотрение различных подходов к тому, как создавать визуальные языки, а также исследование опыта разработки визуальных языков и вариантов их реализации в различных metaCASE-системах. Таким образом, моя цель:

- рассмотреть различные методологии и процессы, как последовательность действий при создании визуального языка
- изучить некоторые metaCASE-системы на предмет того, как они поддерживают соответствующие методологии.

Основные понятия

Для создания визуального языка надо определить его синтаксис и семантику. Синтаксис языка - это взаимосвязи знаков языка, которые определяют правила построения из них текстов. Семантика языка - это значение его знаков, их проекции в предметную область.

Выделяют три вида синтаксиса: абстрактный, конкретный и служебный.

Абстрактный синтаксис - это структура визуальных спецификаций, в рамках которой тщательно классифицированы все графические символы, определены все их атрибуты и связи друг с другом.

Конкретный синтаксис задает правила изображения символов языка, из которых строятся визуальные модели.

И наконец, служебный синтаксис задает способ хранения визуальных моделей. Это формат хранения визуальных спецификаций, выполненных с помощью данного языка. Одним из самых распространенных средств служебного синтаксиса является XMI (XML MetaData Interchange).

При визульном моделировании ПО используются следующие уровни абстракции:

- предметная область
- модель
- метамодель
- метаметамодель.

Как говорилось во введении, модель - это упрощенное описание предметной области. Метамодель - это описание языка, при помощи которого строятся модели. Метаметамодель - это описание языка, на котором строятся метамодели. На рис. 1 (URL:

http://www.omg.org/spec/UML/2.4.1/Infrastructure/PDF) приведена схема уровней абстракции. Предметная область для приведенного ниже примера - это приложение, работающее с видео-файлами. Описание компонент этого приложения происходит с помощью различных моделей (M1 - User model). Чтобы создать модели, нужен язык. На следующем уровне абстракции находится описание этого языка, т.е. его метамодель (M2 - UML). В данном случае метамодель языка UML. Чтобы описать метамодель также нужен язык. Его метамодель, она называется метаметамодель, находится на следующем уровне абстракции, M3 - MOF. В данном случае метамодель языка MOF является метаметамоделью.

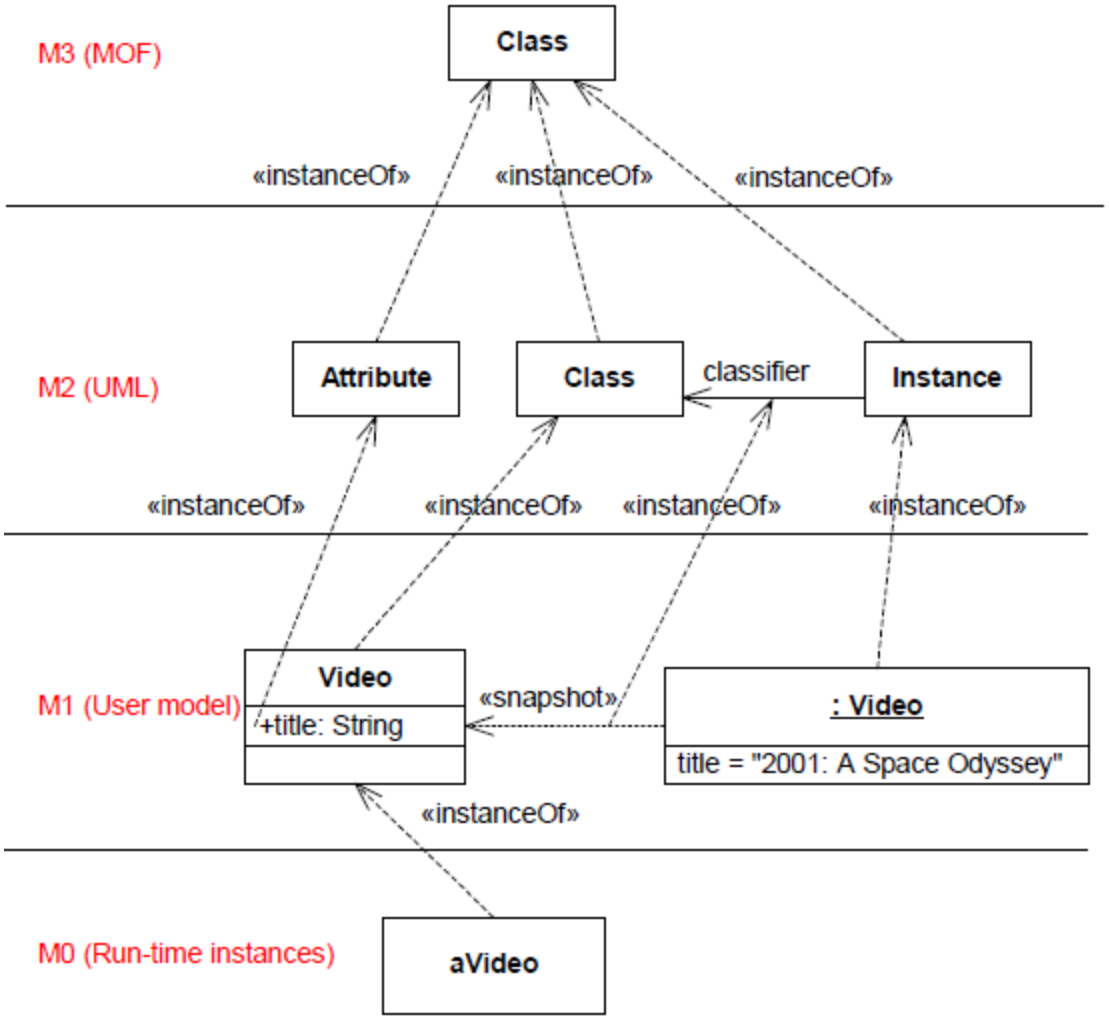

Рис. 1

Классификационная модель

Описание графических объектов и правил построения допустимых визуальных предложений является важным шагом при создании визуального языка. Существует большое количество различных топологий визуальных языков, каждая из которых со своими особыми графическими и структурными характеристиками. Поэтому существует необходимость иметь модели и инструменты, чтобы объединить шаги проектирования для различных типов визуальных языков. Для этой цели приводится формальная модель классов визуальных языков, созданная группой итальянских ученых из университета Салерно[1]. Каждый класс характеризуется семейством визуальных языков, который основан на природе графических объектов и правил построения.

Каждая из metaCASE-систем основана на особом методе описания визуального языка. В настоящее время выделяют три главных подхода для описания визуальных языков: грамматический, логический и алгебраический[3].

Грамматический подход основан на грамматических формализмах, использующихся в описаниях строковых языков. Грамматические формализмы для описания визуальных языков отличаются от грамматических формализмов для описания строковых языков тем, что для визуальных языков нужно рассматривать множества и мультимножества, а для строковых - последовательности символов, а также для визуальных языков нужно описывать геометрические отношения между рассматриваемыми объектами.

Логический подход использует математическую логику первого порядка или другие формы математической логики, которые часто возникают из искусственного интеллекта.

Эти подходы обычно базируются на пространственных логиках, которые аксиоматизируют различные возможные топологические (геометрические) отношения между объектами. Одно из преимуществ логического подхода - это то, что один и тот же формализм может быть использован для описания и синтаксиса, и семантики диаграмм.

Алгебраический подход состоит из композиции функций построения, которые формируют сложные графические изображения из более простых графических элементов. Процесс парсинга может быть рассмотрен аналогично процессу нахождения функциональной последовательности, которая задает изображение. Семантика может быть задана параллельными алгебраическими спецификациями для диаграмм.

Эти три подхода тесно связаны. Грамматические формализмы могут быть рассмотрены как легко вычислительно разрешимые части логического и алгебраического подходов. Например, контекстно-свободные грамматики по существу изоморфны определению логических формул и систем уравнений, в которых уравнения упорядочены.

Грамматический подход

Такой подход использует два метода для представления визуального предложения: основанный на отношениях и основанный на атрибутах. Первый описывает предложение как множество графических объектов и также описывает отношения над этими графическими объектами. Во втором методе предложения понимаются как множество атрибутов графических объектов.

Определим модель классов визуальных языков. Каждый класс характеризуется семейством визуальных языков в терминах синтаксических атрибутов собственных графических объектов и пространственными отношениями, которые могут быть использованы для построения визуальных предложений. Например, графический объект в семействе языка графов потоков управления - прямоугольник. Его синтаксические атрибуты определяют особые точки на стороне прямоугольника. Отношения - это внутренние соединения, которые используются, чтобы связывать графические объекты посредством присоединяющих точек, и эти отношения визуализированы посредством

ломанных линий. Другой пример. Графический объект в семействе иконических языков - это иконка. Ее синтаксические атрибуты - кооррдинаты центра изображения. В качестве отношений выступают пространственные соединения. Эти два примера отражают два метода, которые обычно используют для построения визуальных предложений: с помощью соединения графических объектов и с помощью компоновки этих графических объектов в пространстве. Поэтому классы визуальных языков подразделяются на две категории: основанные на соединениях и основанные на геометрии.

1) Классы, основанные на соединениях (connection-based classes)

Здесь визуальные предложения формируют множество графических объектов, связанных между собой. Синтаксическими атрибутами графического объекта являются обособленные соединяющие точки или даже множества соединяющих точек. В последнем случае, множества могут быть дискретными или непрерывными (такие как линии, области). Такие множества называются соединяющими областями. Каждому соединению дается уникальный идентификатор. Этот идентификатор является значением соединяющей точки или области. В частности, соединяющая точка является частным случаем соединяющей области, которая содержит одну точку.

Соединения могут быть визуализированы различными способами. Рассмотрим только два способа: перекрытие и связывание. В первом случае, соединения визуализированы посредством перекрытия соединяющих областей. Во втором случае - посредством явных связываний между соединяющими областями. На рис. 2 представлены эти два типа соединения. Случаи a) и b) показывают перекрытие, на рис a) показано перекрытие соединяющих точек двух линий. Вместе они образуют литеру V. На рис b) показано перекрытие соединяющих областей. Случаи c) и d) показывают второй тип соединения - связывание. На рис. c) показано как два графических объекта блок-схемы соединены связью, которая связывает две соединяющих точки, а на рис. d) показано, как две соединяющих области (множество точек, образующие окружность) связаны посредством связи, образуя простой граф.

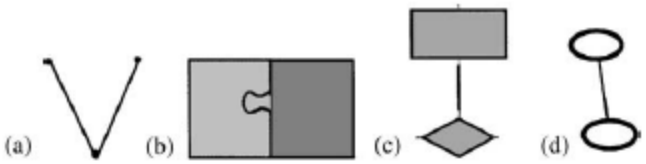

Рис. 2

Выбор типа соединения зависит от характеристик изучаемого визуального языка. Например, графы с помеченными ребрами и ориентированные графы могут быть представлены как множества вершин и ребер, где окрестность вершины (соединяющая область) может быть перекрыта одной или двумя конечными вершинами ребра (соединяющая точка). Выбор такого представления означает, что ребра несут собственную информацию, такую как метки и направления. По этой причине их нужно представить как графические объекты. А непомеченные неориентированные графы могут быть явно представлены как множества вершин, соединенных посредством ломаных, представляющих отношения между вершинами.

a) Класс сетей (plex class)

Визуальные языки в этом классе имеют структуру графа с ограничением, что каждый терминальный графический объект может иметь только фиксированное число соединений. На рис. 3 a) изображены графические объекты языка логических схем. На рис. 3 b) изображена логическая схема, где графические объекты соединены посредством связей.

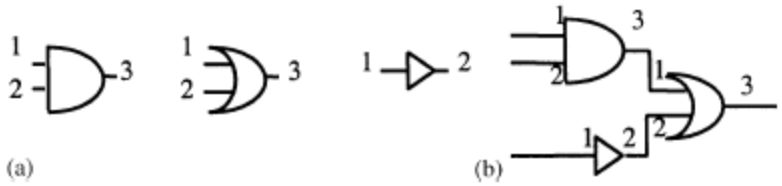

Рис. 3

b) Класс графов (class graph)

Визуальные языки в этом классе имеют структуру графа без ограничений. На рис.4 изображена простая решетка. Графический объект - это вершина, которая имеет две соединяющих области: верхняя полуокружность и нижняя полуокружность (выделена

жирным). Между двумя вершинами существует связь, которая соединяет нижнюю полуокружность и верхнюю полуокружность двух вершин соответственно.

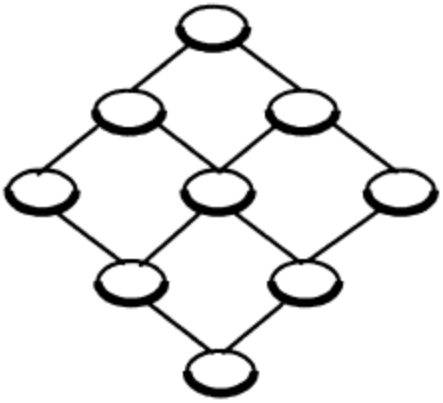

Рис. 4

2) Классы, основанные на геометрии (geometric-based classes)

Визуальное предложение в языках из этих классов описывается как множество графических объектов, пространственно расположенных на координатной плоскости. Используются пространственные правила построения для построения такого рода визуальных предложений. Отношения между графическими объектами могут быть выражены через пространственное включение, смежность, перекрытие. Синтаксические атрибуты графического объекта - это координаты множества точек, достаточных, чтобы связать объекты согласно пространственным правилам. Синтаксические атрибуты автоматически конкретизируются, когда графический объект располагают на координатной плоскости. Стоит заметить, что в качестве синтаксических атрибутов могут быть рассмотрены и другие характеристики (например, круг может быть синтаксически описан координатами центра и размерами радиуса), но использование координат точек на координатной плоскости дает определение классов, основанных на геометрии, независимо от исключительной формы графических объектов.

Как пример языка, который относится к классам, оснаванных на геометрии, можно рассмотреть диаграммы Венна (рис. 5)

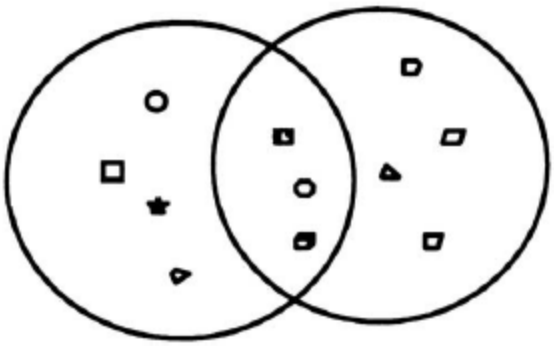

Рис.5

a) Класс строк (class string)

В этот класс входят строковые языки. Этот класс может быть рассмотрен как проекция геометрического класса на линейный случай. Графические объекты этого класса - буквы.

b) Класс иконок (iconic class)

Графическими объектами этого класса являются иконки. Под иконкой понимается изображение внутри прямоугольника фиксированного размера. Отношение между объектами определяется их пространственным расположением друг относительно друга. На рис. 6 изображены два иконических предложения. Первое описывает операцию “удалить строку”, второе - “показать текстовый файл”. В обоих примерах используется пространственное соединение “шаг вправо”.

Рис. 6

c) Класс прямоугольников (class Box)

Графическими объектами этого класса являются ограниченные прямоугольники.

Главное отличие предыдущего класса от этого - это то, что для класса прямоугольников допустимы ограниченные прямоугольники произвольного размера. Это приводит к различным определениям синтаксических атрибутов и отношений внутри этих двух классов. На рис. 7 показано простое коробочное визуальное предложение.

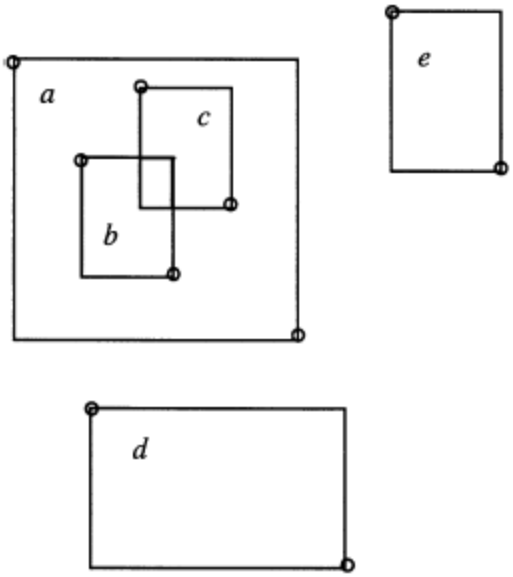

Рис. 7

Также можно двумерные арифметические выражения смоделировать с помощью этого класса. Пример такого моделирования можно увидеть на рис. 8.

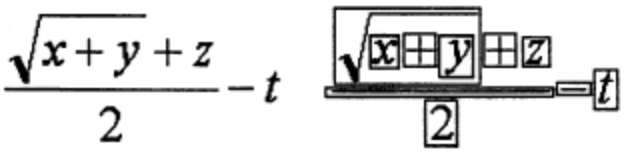

Рис.8

Иерархия классов

Классы визуальных языков, описанные выше, могут быть представлены в некоторой иерархии, которая выражает отношения между ними. На рис. 9 представлена данная иерархия. Абстрактные классы выделены курсивом. Таким образом, наиболее общий класс, основанный на соединениях, - это класс графов, а наиболее общий класс,

основанный на геометрии, - класс прямоугольников. Класс сетей - это спецификация класса графов, где соединяющие области содержат изолированные точки. Класс иконок описывает более общий класс прямоугольников, а класс строк - спецификация иконического класса в линейном случае.

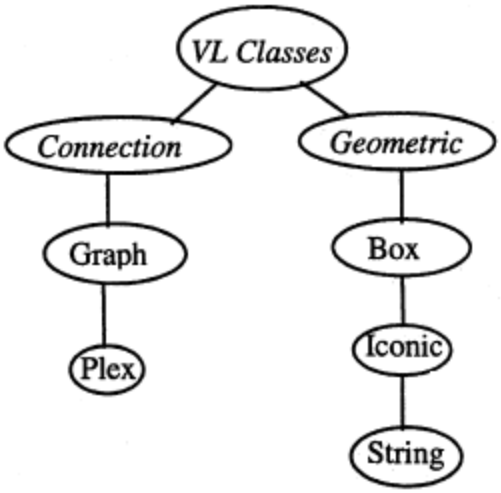

Рис. 9

Гибридные классы

Визуальный язык может быть построен на основе базисных классов, которые были определены выше. Интегрируя определенные характеристики из этих классов, можно построить новый класс визуальных языков. Т.е. графические объекты этого нового визуального языка могут одновременно иметь свойства, которые имеют графические объекты визуальных языков из разных базисных классов. Внедрение такой парадигмы моделирования мотивируется необходимостью поддерживать определение сложных визуальных языков, включающих много практических визуальных языков таких как: SDL и MSC - для моделирования телекоммуникационных систем, BPML и IDEF - для моделирования бизнесс-процессов на разных уровнях и многие другие.

Применение модели классов

Перед тем, как создать визуальный язык, необходимо определить, к какому классу

он принадлежит. Точность выбора зависит от того, как естественно выбранный класс описывает язык, и от особых нужд проектировщика визуального языка.

Понятие класса визуального языка встроено в VLCC (Visual Language Compiler-Compiler), который наследует понятия и методы традиционных генераторов компиляторов, таких как YACC, и расширяет их на визуальную область. VLCC обеспечивает создателя визуального языка вспомогательными инструментами, чтобы помочь ему в процессе определения графических объектов, синтаксиса и семантики проектируемого языка. Инструменты для определения визуального языка являются параметрами по отношению к классу визуального языка. Проектировщик может выбрать подходящий класс для создаваемого визуального языка и использовать соответствующий редактор грамматики для этого класса. Для гибридных визуальных языков проектировщик может выбрать больше, чем один класс.

DSM-платформы

Существует много различных средств, которые позволяют использовать визуальные языки, созданные специально для конкретной предметной области. Такие языки называются предметно-ориентированными языками (DSL, Domain Specific Language), а их использование называется предметно-ориентированным моделированием (DSM, Domain-Specific Modeling). Каждая DSM-платформа поддерживает конкретную методологию создания предметно-ориентированных языков. Рассмотрим некоторые из них.

MetaEdit+

MetaEdit+(URL: http://metacase.com/) является средой для создания и использования предметно-ориентированных языков, которая была разработана в рамках научно-исследовательского проекта MetaPHOR в University of Jyväskylä как расширение среды MetaEdit, которая была создана ранее в рамках проекта SYTI в сотрудничестве с компанией MetaCase в период с конца 1980-х - начала 1990-х годов.

Система MetaEdit+ состоит из двух продуктов: Method Workbench и MetaEdit+ Modeler, первый из них предназначен для проектирования различных визуальных языков, второй является многопользовательской средой, предоставляющей конечным пользователям средства для работы с визуальными языками, как созданными с помощью Method Workbench, так и с входящими в стандартную поставку системы.

Method WorkBench предлагает описывать визуальный язык с помощью языка метамоделирования GOPPRR. С его помощью описываются абстрактный синтаксис и конкретный синтаксис, а также семантика визуального языка. Основные понятия, используемые в этом языке, таковы:

1) Граф (Graph). Под графом понимают создаваемый язык моделирования, т.е. граф – это совокупность объектов, связей, ролей объектов. Примером графа может служить диаграмма потоков данных, диаграмма прецедентов UML и т.п.
2) Объект (Object). Под объектом обычно понимают объекты предметной области, которые визуализируются с помощью данного визуального языка, объекты выражают основные понятия языка моделирования. Это элементы, которые связываются вместе и часто переиспользуются, такие как Process, Message, State, Button. Примером объекта является: класс в диаграммах классов UML, сущность в диаграммах ERD.
3) Отношения (relationships)/Связи. Связи используются для соединения двух и более объектов. В MetaEdit+ поддерживаются следующие виды связей уровня метамодели:
    - Ассоциация. Эта связь соединяет два или более равноправных объектов графа;

- Наследование. Эта связь позволяет создавать новый объект на основе уже существующего объекта;
- Агрегация. Она является частным случаем ассоциации, при котором один из объектов является главным, а другой зависимым;
- Декомпозиция. Эта связь позволяет для любого объекта графа задать уточняющий граф;
- Детализация. Она позволяет объектам, связям и ролям ссылаться на элементы других графов.

4) Роль (Role). Роль определяет, каким образом объект участвует в связи. Так объекты, участвующие в связи «наследование», могут иметь одну из двух ролей: «родитель», «потомок».

5) Порт (Port). Порт - это конструкция языка, задающая семантические и синтаксические ограничения для соединения связи и объекта. Т.е. эти ограничения могут задавать возможные сценарии соединения диаграммных объектов между собой, порты также могут иметь свои наборы свойств и символы для отображений.

6) Свойство (Property). Под свойством понимают атрибут, который характеризует любой из раньше определенных понятий языка. Это атрибуты какого-либо объекта графа. Тип свойства может быть любым: строковым, целочисленным, логическим, ссылкой на файл и т.п.

Эти основные конструкции называются метатипами. Зная основные понятия, требуется создать новый визуальный язык. Для определения языка моделирования в Method WorkBench нужно определить понятия, накладываемые на них правила и ограничения, задать нотацию и описать генератор. На первых двух шагах по сути создается абстрактный синтаксис нового языка. В данном контексте абстрактный синтаксис языка, или метамодель языка - это графоподобная структура, состоящая из объектов, обладающих

некоторыми свойствами и связанными друг с другом ориентированными отношениями (при этом эти связи могут проходить через определенные порты). Накладываемые на метамодель ограничения и правила определяют свойства связей между объектами (кратность, порты, ориентацию), взаимные зависимости, число экземпляров и т.д. На третьем шаге для всех элементов метамодели языка рисуется нотация.

Опишем подробнее первые два шага.

1. Создание нового типа графа.

Определить подграф можно двумя способами: первый - как отдельный подграф, второй - как интеграция нескольких пографов. Если создаем отдельный подграф, то тип создаваемой метамодели Metamodel[GOPPRR]. Если мы создаем целый язык (т.е. интеграция нескольких подграфов), то тип создаваемого графа должен быть Metamodel for multiple graphs[GOPPRR] (об этом подробнее в п.4 - интеграция отдельных графов). Затем определяем атрибуты подграфа (имя, свойства).

2. Добавление нового объекта в метамодель.

Создаем Object[GOPPRR] или Object Set[GOPPRR] (понятие Object Set используется для описания коллекции объектов в связывании. Это упрощает проектирование метамодели в том смысле, что не нужно описывать связи для каждого объекта в отдельности). Определяем атрибуты и свойства объекта. Также есть возможность переиспользовать уже существующие свойства.

3. Создание связей между объектами.

Для определения связей в Method WorkBench есть инструмент Binding[GOPPRR]. Определяем тип отношения, его имя и свойства. Для связывания надо определить по крайней мере две роли. Существует также возможность переиспользовать уже определенные типы ролей. Определяется направление роли.

4. Интеграция отдельных графов или нескольких языков вместе.

Аналогично созданию отдельной метамодели, создаем Metamodel for multiple graphs[GOPPRR]. Далее, перед разработчиком стоит выбор: или он хочет интегрировать уже существующие подграфы, или создать здесь новые метамодели по уже расписанному сценарию, которые впоследствии будут интегрироваться. Дальше происходит создание объектов в вершинном графе, где переиспользуются уже определенные объекты в подграфе. Любые изменения с объектом в одном месте отразятся и в другом месте, где объект переиспользуется. Затем, происходит создание декомпозиционного отношения между объектом внутри подграфа и самим подграфом. Эта структура позволяет создавать и поддерживать иерархию подграфов.

Таким образом, пройдя данные 4 шага, проектировщик задал визуальный язык, определив его понятия, свойства, связи и правила.

Далее, происходит третий шаг моделирования - задание графической нотации каждого понятия в созданном языке, для чего используются соответствующие инструменты Method WorkBench.

Method WorkBench позволяет моделировать не только статические структуры, но и поведение. Например, для декларативного (информационно-логического) языка моделирования семантика статична и задается метамоделью. Для выполняемого языка моделирования семантика определяется поведением вычислительной модели. И в том и в другом случае в Method WorkBench семантика использования разрабатываемого языка моделирования задается с помощью конкретного представления моделей, описываемых на нем.

С помощью редактора генераторов можно задать любое представление для модели как экземпляра метамодели языка (шаг 4): в виде документа XML, программы на известном языке программирования и т.д. Генераторы создаются с помощью предопределенных шаблонов, обращений к интерфейсу метамодели и скриптового языка описания генераторов MERL.

Созданный язык затем используется для создания и редактирования моделей в MetaEdit+ Modeler. Таким образом, в основе определяемого языка лежит метамодель GOPPRR (модель для создания метамодели языка моделирования), а операционная семантика задается в генераторе.

Согласно классификационной модели, определенной выше, создаваемые языки в Method WorkBench принадлежат к множеству классов, основанных на соединениях. В частности, они принадлежат классу графов. Таким образом подход к созданию визуального языка в инструменте Method WorkBench можно описать следующим образом: на основе метаязыка GOPPRR задаем абстрактный синтаксис языка в виде графа. Затем определяем его конретный синтаксис, используя соответсвующие инструменты. Определяется семантика. Method WorkBench позволяет создавать динамически настраиваемые языки. Т.е. модификация метамоделей может производиться на любом этапе создания визуального языка. При этом после внесения изменений в метамодель система автоматически внесет все необходимые изменения в модели, созданные с помощью этой модели.

Пример метамодели для языка, который описывает диаграммы UseCase из пакета UML, можно увидеть на рис.10 (URL:http://www.metacase.com/support/50/manuals/Graphical%20Metamodeling.pdf).

На рис. 11 показана модель на языке UseCase диаграмм.

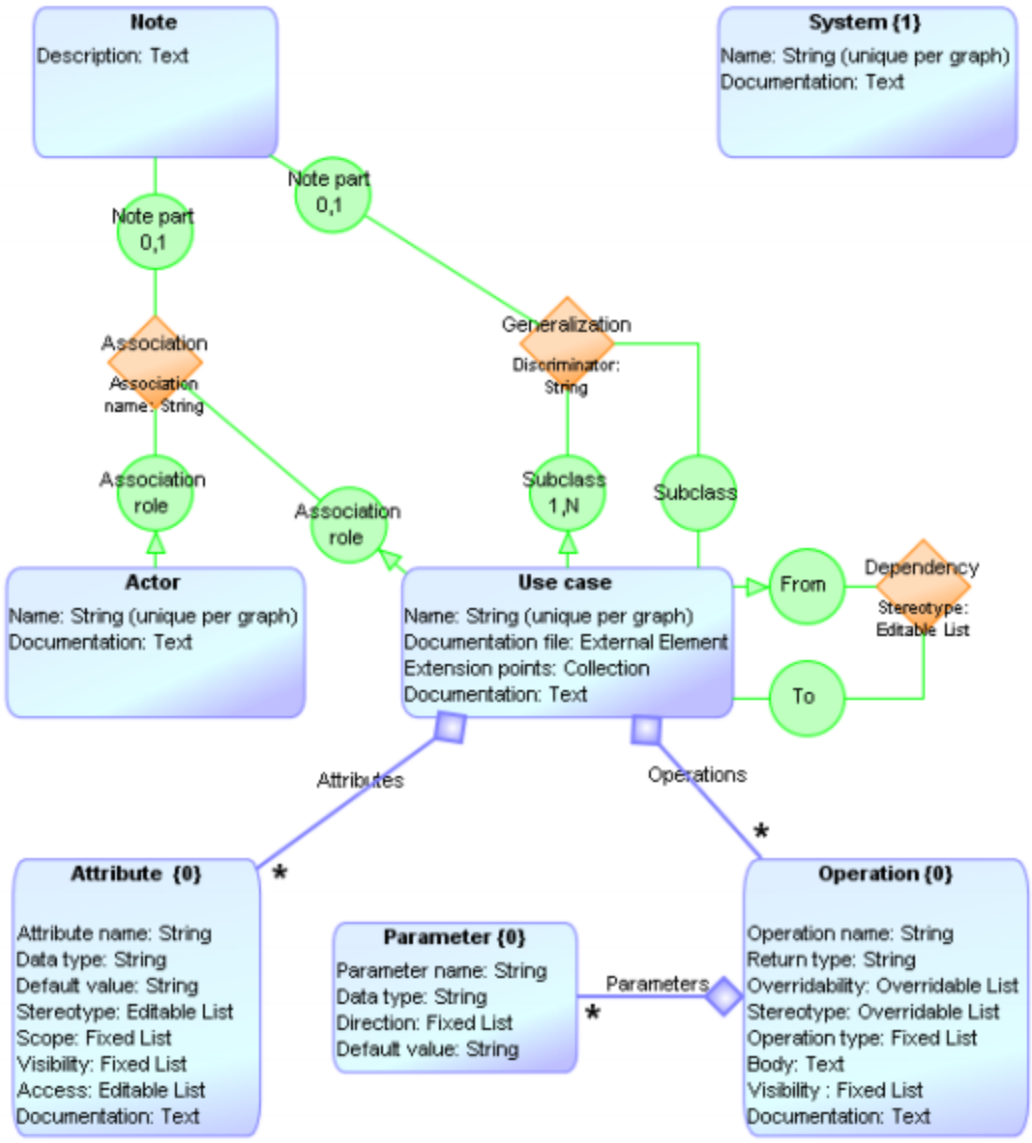
Note
Description: Text
System {1}
Name: String (unique per graph)
Documentation: Text
Note part
0,1
Note part
0,1
Association
Association
name: String
Generalization
Discriminator:
String
Association
role
Association
role
Subclass
1,N
Subclass
Actor
Name: String (unique per graph)
Documentation: Text
Use case
Name: String (unique per graph)
Documentation file: External Element
Extension points: Collection
Documentation: Text
From
Dependency
Stereotype:
Editable List
To
Attributes
Operations
*
*
Attribute {0}
Attribute name: String
Data type: String
Default value: String
Stereotype: Editable List
Scope: Fixed List
Visibility: Fixed List
Access: Editable List
Documentation: Text
Parameter {0}
Parameter name: String
Data type: String
Direction: Fixed List
Default value: String
Parameters
*
Operation {0}
Operation name: String
Return type: String
Overridability: Overridable List
Stereotype: Overridable List
Operation type: Fixed List
Body: Text
Visibility : Fixed List
Documentation: Text

Рис. 10

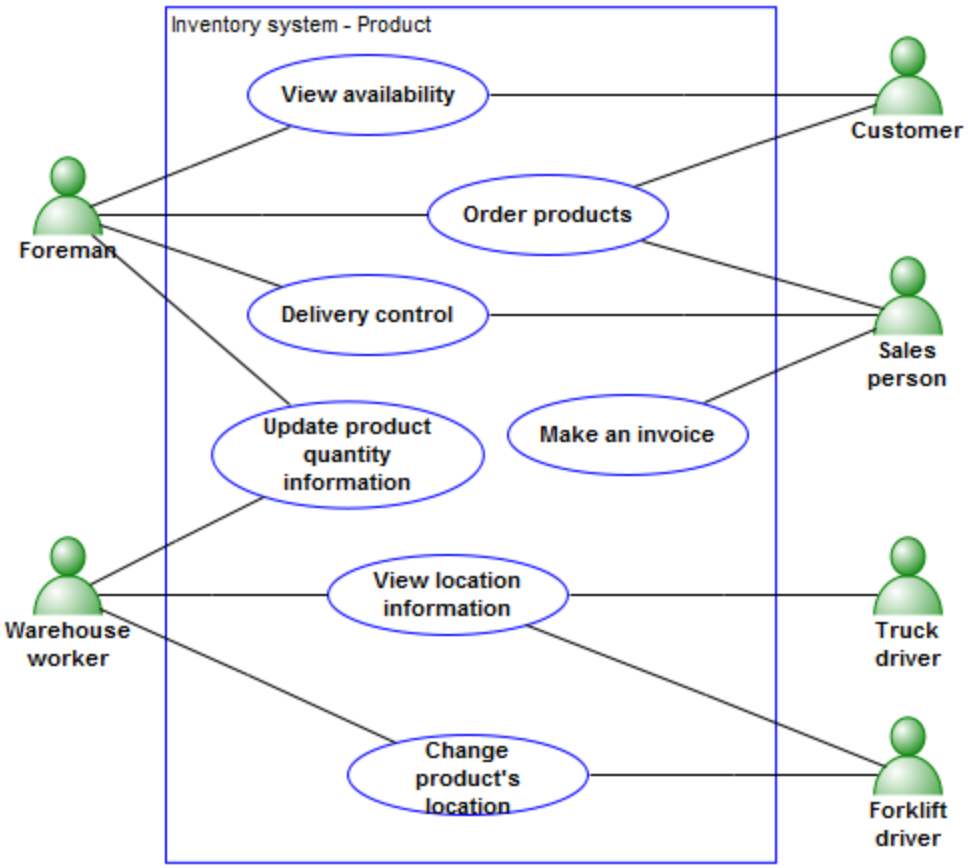

Рис. 11

eMOFLON

eMOFLON (URL: http://www.moflon.org) является metaCASE-инструментом, который поддерживает процесс моделирования ПО, обеспечивая метамоделирование и систему трансформации моделей. Этот инструмент больше подходит для изменения моделей и интеграции, но разработку визуальных языков он тоже поддерживает. Здесь семантика визуального языка разбивается на два типа: статическая и динамическая.

Статическая семантика определяет правила и ограничения объектов создаваемого языка (например, кратность), в то время как динамическая семантика определяет правила для динамического поведения системы. Статическая семантика выражена в метамодели.

В качестве метаязыка eMOFLON использует язык метамоделировая Ecore, метаязык из Eclipse GMP. Он определяет такие сущности как EClass и EReference. EClass определяет объекты языка, EReference - связи между ними. У каждого объекта, определенного как EClass, существуют атрибуты (например, имя, размер и т.д. - для конкретного объекта набор атрибутов свой). Для каждого такого атрибута определются типы (например, EString, EInt и др.). У связи EReference существует источник (source), от какого объекта она проводится, и цель (target), куда такая связь проводится. Также для связи существут свойства, такие как кратность, видимость, направленность и др. Определив все это мы задали абстрактный синтаксис визуального языка и его статическую семантику.

Далее определяется динамическая семантика. Ее можно определить через преобразование моделей, но также можно задать с помощью операций прямо в метамодели. Для операции также задаются свойства. Здесь можно провести аналогию с программированием на объектно-ориентированном языке. Мы также обозначаем тип возвращаемого значения, параметры. Т.е. операции выступают в роли методов в ООП. Определив динамическую семантику мы задали визуальный язык.

Пример метамодели в metaCASE-инструменте eMOFLON можно увидеть на рис.12.

На рис. 13 приведена модель на языке, который описан с помощью определенной на рис. 12 метамодели. Матемодель задает модель так называемой коробки для изучения языков. В коробке есть секции, в секциях карточки. На карточке написана та или иная фраза, на обороте - перевод. Модель на рис.12 показывает процесс угадывания перевода на карточке. Если угадали - переводим карточку в следующую секцию, нет - приписываем штраф, и оставляем в этой же секции.

Язык, определенный в инструменте eMOFLON, согласно классификационной модели, также принадлежит к классам, основанным на соединениях. Также его можно отнести к классу графов, как и в MetaEdit+.

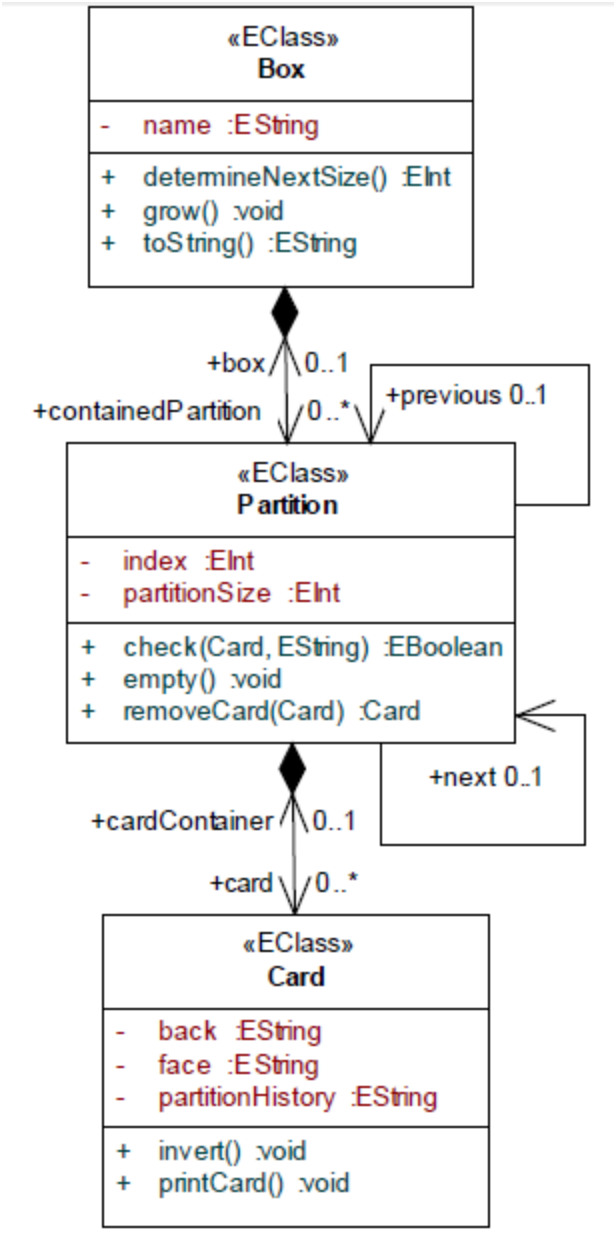
«EClass»
Box
- name :EString
+ determineNextSize() :EInt
+ grow() :void
+ toString() :EString
+box
0..1
+containedPartition
0..*
+previous 0..1
«EClass»
Partition
- index :EInt
- partitionSize :EInt
+ check(Card, EString) :EBoolean
+ empty() :void
+ removeCard(Card) :Card
+next 0..1
+cardContainer
0..1
+card
0..*
«EClass»
Card
- back :EString
- face :EString
- partitionHistory :EString
+ invert() :void
+ printCard() :void

Рис. 12

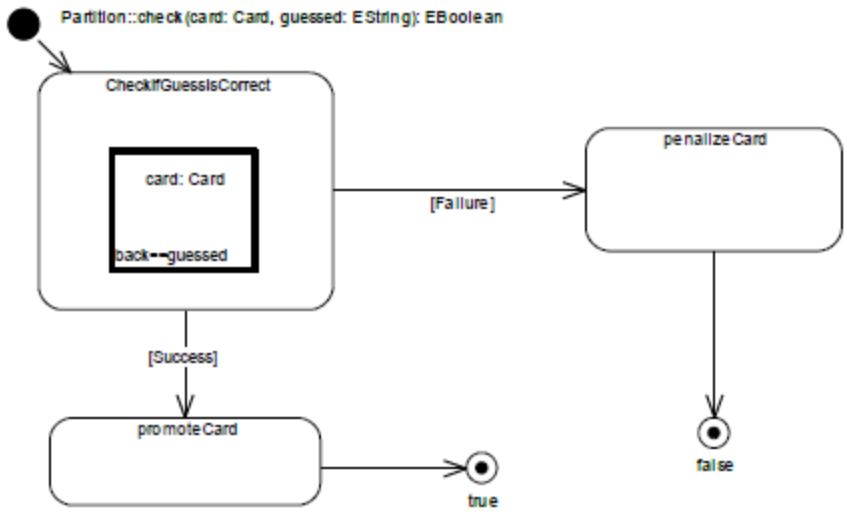
Partition::check(card: Card, guessed: EString): EBoolean
CheckIfGuessIsCorrect
card: Card
back=guessed
[Failure]
penalizeCard
[Success]
promoteCard
true
false

Рис. 13

Результаты

В рамках курсовой были получены следующие результаты:

- изучены два metaCASE-инструмента: MetaEdit+ и eMOFLON;
- реализованы метамодели нескольких пробных языков в указанных выше инструментах;
- написаны обзор по созданию визуального языка в инструменте MetaEdit+, обзор методов классификации визуальных языков
- упорядочены полученные знания и написан отчет

Список литературы

[1] G. Costagliola, A. Delucia, S. Orefice and G. Polese, A Classification Framework to Support the Design of Visual Languages, Journal of Visual Languages and Computing (2002) 13.

[2] An Introduction to Metamodelling and Graph Transformations with eMOFLON, http://www.moflon.org (дата обращения 10.05.2013)

[3] Kim Marriott, Bernd Meyer, Kent B. Wittenburg, A Survey of Visual Language Specification and Recognition.

[4] Graphical Metamodeling, http://metacase.com/ (дата обращения 23.03.2013)