OSTIS-2016
(Open Semantic Technologies for Intelligent Systems)

УДК 004.822

ПАРАДИГМЫ ПРОГРАММИРОВАНИЯ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ СТРУКТУРНОГО СИНТЕЗА

Бикмуллина И.И.

Казанский национальный исследовательский технический университет им. А.Н. Туполева, г.Казань, Россия

elsiyar-b@yandex.ru

В статье развивается идея применения и разработки автоматизированного структурного синтеза UML моделей на основе семантических отношений предметной области. Для предлагаемого подход автоматизированного структурного синтеза информационных систем более подробно рассматриваются объекты UML с семантической точки зрения и решения принятые с целью адекватного, профессионального синтеза диаграммы классов UML.

Ключевые слова: свойства; UML; диаграммы классов; автоматизированный синтез моделей; программирование.

Введение

В современной технологии программного обеспечения (ПО) роль эксперта, рассматриваемой информационной технологией предметной области, находится на уровне консультанта. Так как современная технология разработки программ в идеале состоит из трех этапов, где автоматизированным этапом является только третий этап:

1 этап. Разработка онтологии предметной области экспертом этой области с помощью разработчика (программиста). Создаваемая система может опираться на несколько предметных областей, что влечет привлечение к разработке нескольких экспертов. В этом случае от разработчика требуется разобраться в этих разных предметных областях, найти общий язык с экспертами и формализовать данные. Это требует от разработчика владения несколькими специальностями.

2 этап. Моделирование. То есть на основе глоссария, онтологии предметной области разрабатываются диаграммы классов UML [Рамбо и др., 2002]. Моделирование диаграмм UML осуществляется разработчиком вручную или составляется им в приложении, какого - либо ПО. Однако стремление разработчиков как можно быстрее получить программный код приводит к тому, что разработчики используют моделирование диаграмм UML лишь, в крайнем случае, лишь тогда, когда проектирование системы зашло в тупик, что бывает достаточно запоздалым использованием данного этапа в технологии. Отсюда, моделирование UML моделей не используется в полной мере.

3 этап. Автоматизированная генерация программного исходного кода на основе разработанных моделей UML с помощью Rational Rose или Visual Studio.

Анализ существующей технологии разработки ПО доказывает необходимость разработки инструментов автоматизированного проектирования структурных моделей информационных систем.

В статье [Бикмуллина, 2015] представлено усовершенствование современной технологии разработки, автоматизации ИС (РИС) за счет автоматизированного проектирования диаграмм классов UML на основе семантических моделей.

Автоматизация синтеза диаграммы классов позволяет уменьшить трудоемкость проектирования структурных моделей ИС, повысить качество программного проекта за счет возможности многовариантного анализа структурных моделей, стимулировать разработчика на предварительное продумывание структуры прорабатываемой системы, изменить характер труда программиста в направлении от ручной проработки диаграмм UML к выбору оптимального варианта из множеств автоматически полученных вариантов.

Парадигмы синтеза РИС подразумевают правила синтеза моделей, синтеза объектов диаграммы классов UML. Синтез объектов диаграммы классов UML включают в себя синтез классов, синтез

отношений. Синтез классов помимо основной парадигмы образования класса с помощью свойств РИС, включает в себя определение частных случаев, таких как абстрактный класс, интерфейс. Синтез отношений подразумевает парадигмы для разработки иерархических и неиерархических отношений. При этом если на начальном этапе важность выделения парадигм не столь видна, то на этапе формирования многовариантного моделирования важность данной системы становится очевидной.

Основой для автоматического синтеза структурной модели являются правила принятия решения построения элементов структурных моделей. Каждое правило принятия решений определяет возможные структурные сочетания компонентов и атрибутов. Семантические отношения между составляющими этой модели весьма разнообразны, но они могут быть описаны и представлены с помощью семантических структур. При этом задача РИС сводится к тому, чтобы получить изделие требуемой спецификации. В объектно-ориентированной технологии создания программных изделий спецификация РИС задается системой диаграмм. Система диаграмм – это свойства и основа изготовления РИС. Таким образом, необходимым условием существования РИС является спецификация РИС в виде системы диаграмм. Природа специфики в понимании свойств РИС основывается на разнообразии вариантов абстрагирования, использующихся в разных науках. Отличия типов исследуемых свойств, способов моделирования данных типов во многом определяется дифференциацией наук.

1. Свойства разрабатываемой информационной системы

Процесс РИС - это придание объектам моделирования желаемых свойств, адекватно отражающих необходимые черты моделируемой предметной области.

Свойства РИС описывается в зависимости от особенностей ИС, требований к ней, рассматриваемых как техническое средство, инструмент решения поставленных задач. Особенность – это черта, характеризующая объект с точки зрения его связи с другими объектами: возможности превращения в другой объект под давлением другого объекта, возможности влияния на другой объект. Также программное изделие должно иметь комплекс особенностей. Одной из таких особенностей является та операционная среда, благодаря которой программное изделие превращается в инструмент решения задач автоматизации. Особенность РИС как технического средства - это отношение данного РИС к другим объектам (естественным и искусственным), с которыми программное изделие взаимодействует. Таким образом, особенность – это характерная черта чего-то по отношению к другим предметам. В этом смысле прикладная программа обладает набором свойств, которые позволяют решать возложенную на нее прикладную задачу. Вычислительная среда, наоборот представляет собой лишь особенность, так как черта, не зависит от прикладной задачи.

В автоматизированных системах современная обработка данных в основном работает с терминами. Для создания интеллектуальных автоматизированных систем, в том числе и систем автоматизации проектирования РИС, нужно от терминов перейти к понятиям. Для этого нужно включить в обработку явным образом выписанные смыслы(все), которые обозначены конкретными терминами. Понятием является языковая модель, четко определяющая смысл. Понятие в отличие от термина (зафиксированного понятия, где смысл зафиксирован) не статично во времени, его смысл меняется достаточно динамично, но при этом сохраняя значительную стабильность. Любой научный термин нагружен базовым списком свойств (многие из которых лишь подразумеваются). И когда в математике говорится «множество», то неявно предполагается, что неважна последовательность его элементов и его элементы не повторяются и т.п. В моделировании деталей машин также имеется много примеров: «консоль», «вал», втулка» и т.п. Смысл этих терминов известен.

Было принято решение, что понятия (ConceptsUM) в диаграмме классов [Бикмуллина, 2015] будут формализоваться по принципу:

$$Concepts_{UML} = <Concepts, Relations, Initial\_Concept>, \quad (1)$$

где понятия диаграммы классов состоят из множества понятия (*Concepts*), множества отношений (*Relations*) различного вида (иерархические, неиерархические) между данными понятиями и начального понятия (*Initial_Concept*) в описании класса понятий. Рассмотрим вышеизложенные компоненты понятия диаграммы класса более подробно:

$Concepts = \{Concept_{i=1}\}^{conceptscount}$ – конечное непустое множество понятий (включая *Initial_Concept*), при этом $Concept_i$ состоит из:

$$Concept_i = <Concept_{Value}, Concept_{Kind}, Concept_{TermType}>, \quad (2)$$

где $Concept_{Value}$- значение понятия;
$Concept_{Kind}$- тип понятия;
$Concept_{TermType}$ - тип терминального понятия.

Итак, понятие включает в себя:

1.Значение понятия включает в себя:

$$Concept_{Value}=<Concept_{ValueType}, Concept_{ValueValue}>, \quad (3)$$

где значение понятия описывается своим типом $Concept_{ValueType}$ и значением $Concept_{ValueValue}$.

$Concept_{ValueType} \in$ {«Строковое», «Целое», «Вещественное», «Логическое», «Бинарные данные»}.

Значение понятия есть непустая последовательность символов, которая идентифицирует определенный класс объектов описываемой предметной области, множество значений (сорт) или представляет в ней конкретное (константное) значение:

$$Concept_{Value\_Value} \in String \cup Integer \cup \cup Real \cup Boolean. \quad (4)$$

String – множество строк. *Integer* –множество целых чисел. *Real* – множество вещественных чисел. *Boolean*– множество {«true», «false»}.

Значением понятия может быть последовательность символов, интерпретируемых как строковая константа ($Concept_{ValueType}$ = «Строковое»), целое число из множества целых чисел ($Concept_{ValueType}$ = «Целое»), вещественное число из множества вещественных чисел ($Concept_{ValueType}$ = «Вещественное»), элемент множества {«true», «false»} ($Concept_{ValueType}$ = «Логическое»), ($Concept_{ValueType}$ = «Бинарные данные»).

2. Тип понятия включает в себя:

$Concept_{Kind} \in$ {«Терминальное», «Нетерминальное»}. То есть понятие может быть терминальным ($Concept_{Kind}$ = «Терминальное») или нетерминальным ($Concept_{Kind}$ = «Нетерминальное»).

3. Тип терминального понятия состоит из:

$Concept_{TermType} \in$ {«Идентификатор», «Константа», «Сорт», «Не определено»}. Терминальное понятие может быть идентификатором ($Concept_{TermType}$=«Идентификатор»), константой ($Concept_{TermType}$=«Константа»), или обозначать некоторый сорт ($Concept_{TermType}$=«Сорт»).

Если понятие $Concept_i$ является нетерминальным, то $Concept_{TermType}$ = «Неопределено».

Также на понятия накладываются ограничения, которые могут отсутствовать, если текстовое представление информации не требуется.

Описание синтаксических ограничений:

$$Syntax\_Restrictions = <Lexica, Syntax>, \quad (5)$$

где *Lexica* - лексические ограничения;
Syntax - синтаксические ограничения.

Описание лексических ограничений:

$$Lexica = \{<Lexem\_Type_i, Definition_i>\}_{i=1}, \quad (6)$$

где $Lexem\_Type_i \in$ {«Идентификатор», «Целое число», «Вещественное число», «Строковая константа», «Ограничитель строковой константы»};
$Definition_i$ – строка символов, включающая метасимволы и представляющая конкретное лексическое ограничение для вида лексемы.

Синтаксические ограничения задаются для нетерминальных понятий:

$$Syntax = \{<Concept_i, Definition_i>\}, \quad (7)$$

где $Concept_i \in Concepts \setminus Terminal\_Concepts$;
$Definition_i$ – строка символов, включающая метасимволы и представляющая конкретное синтаксическое ограничение для конкретного понятия.

Далее возникает необходимость в предоставлении связок, используемых для формулировки сочетаемости понятий:

1) «И»,

$$P(x)=P_1(x) \Lambda P_2(x) \Lambda \dots \Lambda P_n(x). \quad (8)$$

Таким образом, например, фраза «БПЛА (беспилотный летательный аппарат) может быстро доставить небольшую посылку в любую точку города, минуя пробки. Кроме того, с его помощью можно доставлять посылки в труднодоступные места» будет записано как «*БПЛА(x)= беспилотный летательный аппарат(x) Λ быстро доставляющий небольшую посылку в любую точку города(x) Λ минующий пробки(x) Λ доставляющий посылки в труднодоступные места(x)».*

2) «НЕ»,

$$P(x) = \neg P_n(x). \quad (9)$$

Отсюда, например, фраза «Беспилотный летательный аппарат — это летательный аппарат без экипажа на борту» будет записано как «*беспилотный летательный аппарат(x)= летательный(x) Λ аппарат(x) Λ ¬ экипажа на борту(x)».*

3) «ИЛИ»,

$$P(x) = P_1(x) \vee P_2(x) \vee \dots \vee P_n(x). \quad (10)$$

Например, фраза «Беспилотный летательный аппарат— это любое удаленно управляемое или вовсе самостоятельное (интеллектуальное) средство» будет записано как «*беспилотный летательный аппарат(x) = (самостоятельное(x) Λ интеллектуальное(x) ∨ удаленно управляемое(x)) Λ средство(x)*».

Также из выше приведенного примера видно, что приоритет у «И» выше чем у «ИЛИ», то есть между ними взаимоотношения такие же, как между умножением и сложением или логическим & и ∨. Правила структурной сочетаемости понятий:

1) «горизонтальное объединение» – объединение подряд идущих понятий в словосочетание.

2) «целое-часть», $y : x_1 \dots x_n$. Где слева в формуле находится «целое» в виде термина - определяемое понятие, а справа находятся понятия – «части» данного «целого». При этом возможен вариант как строгого включения частей (как неотъемлемые компоненты одного целого), то есть обязательного присутствия и в последствие включения всех частей в целое $y : x_1 \Lambda x_2 \Lambda \dots \Lambda x_n$, либо не строгого включения (компоненты достаточно самостоятельны, чтобы существовать

отдельно от целого) $y : x_1 \vee x_2 \Lambda x_3 \vee \ldots \vee x_n$. Так, например, высказывание «Хвостовое оперение состоит из двух частей: киль и стабилизатор» будет записано как «*хвостовое оперение*» : «*киль*» Λ «*стабилизатор*».

Характеристика понятия «целое / часть» отражает способность понятия структурно присоединять другие понятия или, наоборот, присоединяться к господствующему компоненту сочетания [Барков, 2003]. В работе части в обоих случаях будут относиться к атрибутам, а целое к названию класса. И применяется данное правило для фиксации структурных свойств объекта, а также его характеристик. Однако на практике правила образования структур «целое-часть» недостаточно, так как правила структурной сочетаемости имеют более богатый набор средств интересующие нас сочетаемости. Также возможно пересечение соседних фраз, когда окончание одной является уже началом следующей.

3) Установление тождественных понятий. “Поиск в модели экстенсионально тождественных или интенсионально тождественных элементов (например, с целью поиска тех элементов, являющихся в сечении кругом, являющимися телами вращения)”[Барков, 2003].

4) «Зависимость» - характеристика зависимости понятия, обозначает обязательность совместного использования структурных элементов изделия[Барков, 2003]. Данное правило набирает силу при определении наиболее устойчивых словосочетаниях, например, «*искуственный интеллект*» или когда к понятию привязывается его характеристика, доопределяющая рассматриваемое понятие, например, «*воздушный шар*», «*кусок металла*». Также когда объекты конкретной предметной области друг без друга являются бесполезными элементами, например словосочетание, «*болт, гайка*».

5) «Компонентность», обозначает отнесение параметров рассматриваемого понятия, его характеристик(например, «*размер моста*», так «*размер*» становится атрибутом понятия «*мост*») к его атрибутам, то есть к атрибутным свойством изделия, например, «габаритный размер»[Барков, 2003].

6) «Наследовательность», означает, что у понятий есть что-то общее («*воздухоплавательный аппарат*» и «*аэростат*») и между ними есть смысловая ассоциация «целое-часть», например, «*машина*» и «*легковая машина*», «*грузовая машина*».

Для построения интеллектуальных систем необходимо содержание понятия определить как описание его свойств. Для выделения объема понятия необходимо процедурным путем вычислить свойства рассматриваемых объектов. Итак, свойство является описанием, характеризующим данную предметную область и позволяющим точно определить программное изделие при необходимой степени декомпозиции. При отсутствии этой черты программное изделие превращается в другое изделие.

2. Роль отношений в моделируемой диаграмме классов

Задачей в исследовании является построение системы бинарных отношений *Relation.*

Relations = $\{Relation_{i=0}\}^{count}$– конечное, возможно пустое, множество отношений. Каждое отношение $Relation_i$ является направленным бинарным отношением, имеющим, возможно, спецификатор множественности и связывающим два понятия. Отношение описывается следующим образом:

$$Relation_i=<Relation_{EndSp},Begin\_Concept,End\_Concept>, \quad (11)$$

где $Relation_{EndSp}$ - спецификатор множественности;

Begin_Concept - понятие-начало отношения;

End_Concept - понятие-конец отношения.

Рассмотрим более подробно:

1.Спецификатор множественности (кардинальность) является характеристикой отношения, определяющий способ порождения понятия-экземпляра вместе с отношением к нему по понятию-концу данного отношения:

$Relation_{EndSp} \in$ {«Конкретность», «Единственность», «Множество»}, где

1) $Relation_{EndSp}$= «Конкретность» (пустой спецификатор) означает, что может быть порождено только одно понятие-экземпляр и его значение (имя) должно совпадать со значением (именем) его понятия-прототипа (*End_Concept*).

2) $Relation_{EndSp}$= «Единственность» означает, что по *End_Concept* может быть порождено только одно понятие- экземпляр, но его значение (имя) обязательно должно быть задано при порождении. Порожденное понятие при этом является сущностью из класса объектов или элементом из множества значений, идентифицируемого понятием-прототипом (*End_Concept*) (строка, целое значение, вещественное значение, логическое значение).

3) $Relation_{EndSp}$= «Множество» означает, что по *End_Concept* может быть порождено любое количество (но, по крайней мере, одно) понятий-экземпляров и их значения (имена) обязательно должны быть заданы при порождении. Каждое порожденное понятие при этом является сущностью из класса объектов или элементом из множества значений, идентифицируемого понятием-прототипом (*End_Concept*).

2. *Begin_Concept* – понятие, из которого исходит – понятие-начало отношения.

$$Begin\_Concept \in Concepts \setminus Terminal\_Concepts, \quad (12)$$

где *Terminal_Concepts* – множество терминальных понятий, $Terminal\_Concepts \subset Concepts$. Понятием- началом отношения может быть любое нетерминальное понятие.

3. *End_Concept* – понятие, в которое входит – понятие-конец отношения:

$$End\_Concept \in Concepts \backslash \{Initial\_Concept\}, \quad (13)$$

где *Initial_Concept* – начальное понятие в описании класса понятий, оно единственно, и через него не может быть выражено ни одно другое понятие из описываемого класса понятий:

$$Initial\_Concept \in Concepts. \quad (14)$$

Понятием-концом отношения может быть любое понятие за исключением *начального понятия*.

Бинарные отношения синтезируются учитывая следующую классификацию отношений определенную на основе анализа:

1) виды отношений взаимодействия классов:
   1.1) иерархические
      а) ассоциация:

$$R_{AS}(O_4)\ \{C_i(O_4)\ \text{x}\ C_j(O_4)\} \quad (15)$$

«используется для», «находится в процессе», «быть результатом», «быть полученным при», «служить параметрами», «быть полученным с помощью», «использоваться для» [Глоба и др., 2013].

      i) один к одному,
      ii) один ко многим,
      iii) многие к одному.

      б) наследования (обобщения),

$R_n(O_4)=a_i,r_i|\ A_{Cm}(O_4) \longrightarrow a_i,\ r_i|\ A_{Ck}(O_4)$[Глоба и др., 2013]. (16)

Класс и наследование атрибутов и отношений его подклассами:

$A(C_1),R(C_1)\longrightarrow A(C_{11}),R(C_{11}),A(C_1),R(C_1)\longrightarrow \longrightarrow A(C_{12}),R(C_{12}),A(C_1),R(C_1)\longrightarrow A(C_{13}),R(C_{13})$ [Глоба и др., 2013]. (17)

   1.2)структурные отношения («часть-целое»)

$C_1 \subset C_{11} \Lambda\ C_{12} \Lambda\ C_{13}$ [Глоба и др., 2013]. (18)

      а) агрегации,
      б) композиции.
      в) «класс-данные» (класс – атрибуты, класс - функции)- «входит в состав», «является характеристикой».

$$R_{CD}(O_4) = C_j(O_4) \subseteq D_i(O_4). \quad (19)$$

Отношение данного вида (формула 19) реализовано для всех классов данной онтологии $C_1(O_4) \subseteq D$, $A_{C1} \subseteq A_D$ [Глоба и др., 2013], то есть $A \subseteq A_D$, $C_1 \subseteq A$

   1.3) неиерархические
      а) реализация,
      б) зависимость.

2) виды отношения доступности (разновидности методов доступа):
   2.1) без ограничений (public),
   2.2) закрытое (private),
   2.3) защищенное (protected).

3) для языков с ООП
   3.1) конструктор (C++, C#),
   3.2) деструктор (для языков C++)

4) виды классов
   4.1) абстрактное
   4.2) интерфейс
   4.3) шаблон
   4.4) утилита

Где O_4 – онтология; C_1 – класс, *A*-атрибуты, *R*-отношения, *D* – вид атрибута, $O_4 = \{C, A, R, D\}$.

Перечисленные отношения составляют основу диаграммы классов.

Иерархическая модель – это абстрактных данных ранжированная модель. Из выше представленной классификации отношение наследования означает наличие сущностей предка и потомка, а также передача предком потомку свои методы, структурные части.

Отношение наследования помимо выше сказанного может создавать иерархию, в которой существует общая сущность, может модифицироваться, конкретизоваться на более частные, конкретные сущности. Отношение агрегации и композиции являются одними из представителей иерархии целое – часть. В агрегации при этом части существуют самостоятельно, а в композиции части опекаются разбиваемой сущностью.

Отличие иерархических и неиерархических отношений заключается в существовании в неиерархическом отношении понятия «братья». То есть для описания неиерархического понятия возникает необходимость расширения описания иерархического понятия. Особенности неиерархических отношений:

1) в ряде случаев рассматриваемые отношения обладают иерархическими свойствами и вовсе не несут неиерархическую нагрузку. Например, одно из отношений зависимости может использоваться с такой же целью как иерархическое структурное отношение агрегации. Таким образом, возможен вариант, когда неиерархическое отношение воспринимается, как иерархическое и обладает лишь иерархическими свойствами;
2) чаще всего неиерархические отношения строятся на базе иерархических отношений, то есть отношение, связывающее два класса, обладает как иерархическими свойствами, так и неиерархическими.

Из выше представленных данных и на основе анализа вытекает, что описание неиерархического понятия базируется на иерархическом понятии. В рассмотренном в статье [Бикмуллина, 2015] примере (о нахождении среднего значения и дисперсии) одним из возможных вариантов выборки видов отношений является:

– наследования (от среднего значения к дисперсии),
– агрегации {(файл, запись), (МЭД, файл), (массив, запись)} или
– композиции {(МЭД, массив), (МЭД, файл), (массив, запись)} или

– сервер-клиент {(файл, МЭД), (запись, массив)}.

Этот этап в данной статье рассмотрен не достаточно подробно, чтобы представить всю систему парадигм, однако цель данной статьи заключалась в представлении общей картины данной системы.

Заключение

В работе более подробно рассматриваются объекты UML с семантической точки зрения и решения принятые с целью адекватного, профессионального синтеза моделей диаграммы классов UML и в дальнейшем включаемые в усовершенствование современной технологии РИС за счет автоматизированного проектирования диаграмм классов UML на основе семантических моделей.

Библиографический список

[Рамбо и др., 2002] Рамбо Дж., Якобсон А., Буч Г. UML: специальный справочник– СПб.: Питер, 2002. - 656 с.

[Бикмуллина, 2015] Бикмуллина И.И. Технология автоматизированного синтеза информационных систем с помощью семантических моделей предметной области/ И.И. Бикмуллина// OSIS, 2015, С. 445-450

[Барков, 2003] Барков И.А. Теория конструкторской семантики/ И.А. Барков// ИжГТУ, Ижевск, 2003, С. 30

[Глоба и др., 2013] Глоба Л.С., Терновой М.Ю., Новогрудская Р.Л. Метод организации слабосвязанных информационных ресурсов на порталах знаний / Л.С. Глоба, М.Ю. Терновой, Р.Л. Новогрудская// Минск, OSIS, 2013, С. 49-54

PROGRAMMING PARADIGMS INFORMATION TECHNOLOGY STRUCTURAL SYNTHESIS

Bikmullina I.I.

Kazan National Research Technical University named after A. N. Tupolev (KAI), Kazan, Russia

elsiyar-b@yandex.ru

The paper develops the idea of the application and development of computer-aided structural synthesis of UML models based on semantic relations of the subject area. The proposed approach for automated structural synthesis of information systems described in more detail the UML objects from the semantic point of view and decisions taken for adequate, professional synthesis UML class diagrams.

Introduction

In modern software technology the role of the expert under consideration of the information technology subject area, is at the level of a consultant. As modern technology development programs ideally consists of three steps, where the automated stage is the third stage:

Stage 1. Development of domain ontology by expert this field by using a developer (programmer). The system can rely on a few subject areas, which implies the involvement of several experts. In this case, the designer has to understand these different subject areas, to find a common language with experts and to formalize the data. This requires the developer to own several specialties.

Stage 2. Modeling. That is based on the Glossary of ontology developed the UML class diagram [Rumbaugh et al., 2002]. Modeling language UML diagrams is provided by the developer manually or composed them in the app any software. However, the desire of developers as soon as possible to get the code leads to the fact that developers use modeling diagrams UML only in the extreme case where system design is at an impasse, which is quite late by this stage in the technology. Hence, modelling with UML models is not fully used.

Stage 3. Automated generation of source code based on the developed models using UML rational rose or visual Studio.

Analysis of the existing technologies of software development proves the need to develop tools for automated design of structural models of information systems(IS).

In the article [Bikmullina, 2015] presents the improvement of modern technology development, IS automation through computer-aided design of class diagrams of UML based semantic models.

Automation of synthesis of a class diagram makes it possible to reduce the complexity of designing structural IS models to improve the quality of a software project by allowing multivariate analysis of structural models, to stimulate developer's preliminary thinking patterns are working on the system, change the nature of the work of the programmer from manual study of the UML diagrams to the choice of the optimal variant of the sets automatically received options.

The paradigm of synthesis of the development, automation of IS rules imply synthesis models, synthesis of objects of the UML class diagram. The synthesis of objects class diagrams UML class diagrams include the synthesis, the synthesis of relations. The fusion classes in addition to the basic paradigm of education class using properties of the design, automation of IS involves identifying particular cases, such as abstract class, interface. The synthesis of the relationship implies the paradigm for the development of hierarchical and non-hierarchical relations. If at the initial stage the importance of distinguishing paradigms are not so visible, at the stage of formation of the multivariate modeling, the importance of this system becomes obvious.

The basis for automatic synthesis of structural models are the rules the decision of constructing the elements of structural models. Each rule of a decision specifies the possible structural combinations of components and attributes. Semantic relations between the components of this model are very diverse, but they can be described and represented using semantic structures.The development task of automation IS is to get the product to the required specification.

Conclusion

The paper presents more detail the UML objects from the semantic point of view and decisions taken adequate, professional synthesis models UML class diagrams in the future be included in the improvement of modern technology development, IS automation through computer-aided design of UML class diagrams on the basis of semantic models.