ОНТОЛОГИЧЕСКИЙ МЕТОД ДООПРЕДЕЛЕНИЯ ИМИТАЦИОННОЙ МОДЕЛИ

Александр Миков, Елена Замятина, Евгений Кубрак

***Аннотация**: В докладе представлена подсистема доопределения имитационной модели. Подсистема является компонентом системы имитации и автоматизированного проектирования вычислительных систем Triad.Net. В Triad.Net на ранних стадиях проектирования пользователь может опустить некоторые детали при описании модели проектируемого объекта. Тем не менее, ему необходимо получить оценки (пусть и приближенные) функционирования этого объекта. Подсистема доопределения по определенным критериям осуществляет поиск фрагмента программного кода, позволяющего доопределить модель. Различают автоматическое и полуавтоматическое доопределение модели. В статье приводится архитектура подсистемы доопределения, описаны алгоритмы автоматического доопределения, основанные на онтологическом подходе, указаны особенности полуавтоматического доопределения.*

***Ключевые слова**: Имитационное моделирование, системы автоматизированного проектирования, автоматическая генерация моделей, онтологии, OWL.*

***ACM Classification Keywords**: I.6 Simulation and Modeling I.6.2 Simulation Languages; J.6 Computer-aided Engineering; I.2 Artificial Intelligence I.2.5 Programming Languages and Software - Expert system tools and techniques*

***Conference**: The paper is selected from XIVth International Conference "Knowledge-Dialogue-Solution" KDS 2008, Varna, Bulgaria, June-July 2008*

Введение

Известно, что имитационное моделирование является одним из наиболее часто используемых методов при исследовании сложных систем и, в частности, при проектировании ВС.

Исследователи сложных систем часто сталкиваются с ситуацией необходимости анализа не полностью описанной модели. Обычно это выражается в том, что неизвестны в точности правила функционирования (поведение) отдельных элементов. Например, при моделировании компьютерных сетей проектировщик не всегда может точно определить алгоритм работы маршрутизатора. На ранних стадиях проектирования достаточно довольно грубо определить его поведение. Для исследователя важно, чтобы данные передавались по сети, а конкретный алгоритм работы маршрутизатора его временно не интересует, исследователь абстрагируется от этих деталей.

Ясно, что в таких условиях имитационное моделирование не даст абсолютно точной картины процессов, происходящих в сложной системе, тем не менее, приближенный результат может быть получен. Трудность заключается в том, что программная система имитационного моделирования не может провести сеанс имитации в отсутствие хотя бы одной процедуры, описывающей функционирование элементов моделируемой сложной системы. Требуется замещение отсутствующих процедур какими-либо имеющимися «подходящими» библиотечными процедурами.

В работе рассматривается подход, базирующийся на знаниях, представленных в виде онтологии моделируемой предметной области. Отсутствующие процедуры выбираются из библиотеки на основе информации о конкретных моделируемых элементах системы, представленной в виде семантического типа – особой метки, присваиваемой элементу модели для обозначения его функции, а также некоторых дополнительных ограничений. Выбор процедур подходящих семантических типов основывается на онтологической информации о моделируемой предметной области. Дополнительные ограничения позволяют более точно выбрать библиотечную процедуру.

Онтологический подход в имитационном моделировании

Известно, что онтология – это описание типов сущностей, существующих в предметной области, их свойств и отношений. Каждая предметная область (некая часть реального мира) может быть описана с помощью онтологий. Онтологии создаются и используются во множестве областей знаний, в том числе, известны примеры их успешного применения в имитационном моделировании. Однако создание онтологий для моделирования является достаточно сложной задачей, поскольку этот метод используют для исследования самых разнообразных систем, относящихся к различным предметным областям (химическим, физическим, транспортным и т.д.). Кроме того, методы имитационного моделирования основаны на математических, вероятностных и статистических расчетах, и, таким образом, онтологии для этих областей должны служить основой для всех остальных. Онтологии используют на различных этапах имитационного моделирования, начиная с этапа сбора информации о моделируемой системе и заканчивая этапом валидации модели [Fishwick 2004].

Примерами использования онтологий моделирования могут служить управляемые онтологиями среды моделирования, а также подходы к объединению различных федератов, разрабатываемые для HLA. Подход, разрабатываемый для HLA, использует онтологии для описания требований, которым должны удовлетворять интерфейсы федератов для успешного взаимодействия в федерации, а так же для разработки этих требований, с учётом знаний о моделируемой предметной области.

В работе [Liang, 2003] представлена онтология портов, рассматриваемая как средство автоматизации построения моделей из компонентов. Порты описывают интерфейс, определяющий границы компонентов или подсистем в конфигурации системы. Система представлена как конфигурация подсистем или компонентов, соединенных друг с другом через четко определенные интерфейсы. Онтологии успешно применяются и в других работах по имитационному моделированию [Benjamin, 2006].

Прежде, чем представить архитектуру подсистемы доопределения модели, описать критерии выбора подходящих библиотечных процедур, реализующих функционирование некоторого элемента модели, опишем особенности представления имитационной модели в Triad.Net.

Имитационная модель в Triad

Имитационная модель в Triad [Mikov, 1995] представляет собой совокупность объектов, которые действуют по определённым сценариям и обмениваются информацией друг с другом и может быть представлена тройкой: μ = {Str, Rout, Mes}.{Str), (Rout), (Mes) – это слой структур , рутин и сообщений соответственно.

Слой структур предназначен для описания моделируемых объектов и связей между ними, слой рутин представляет собой набор алгоритмов поведения моделируемых объектов, а слой сообщений даёт возможность описывать сообщения сложной структуры. Моделируемые объекты часто имеют иерархическую структуру. Имитационная модель также является иерархической. Каждый из уровней можно описать как граф с полюсами P = {U,V,W}, где V – множество вершин графа, каждая вершина представляет собой моделируемый объект, который находится на конкретном уровне иерархии. W – набор дуг, связывающих вершины графа (моделируемые объекты). U – набор внешних полюсов. Внутренние полюса используют для передачи сообщений на одном уровне иерархии. Их разделяют на входные In(V) и выходные Out(V). Набор внешних полюсов служит для передачи информации объектам, находящимся на различных (смежных) уровнях иерархии. Структура модели представляется иерархически, для этого вводится операция расшифровки объекта структурой. При выполнении этой операции некоторой вершине v структуры ставится в соответствие граф, описывающий внутреннюю структуру объекта системы, представленного вершиной v. Устанавливается так же соответствие между полюсами вершины и внешними полюсами расшифровывающего графа, таким образом, граф «общается» с окружением через интерфейс, предоставленный вершиной.

Рутина представлена множествами событий (E), состояний Q, моментов времени. Каждое состояние определяется набором значений локальных переменных (множество Var) каждой конкретной рутины. Поведение объекта системы, представляемого некоторой вершиной структуры, определяется путём наложения на неё соответствующей рутины. Операция наложения рутины во многом схожа с операцией расшифровки объекта. Рутина таким же образом представляет внутреннее устройство объекта структуры. Так же устанавливается соответствие между полюсами вершины и полюсами наложенной на неё рутины. Сообщения, приходящие на входные полюса вершины, мгновенно передаются на соответствующие полюса рутины, и наоборот: сообщения исходящие через выходные полюса рутины, передаются через соответствующие полюса вершины. Наложение рутины возможно только на терминальную вершину модели (т.е. не расшифрованную структурой), т.к. поведение расшифрованной вершины определяется поведением вершин её внутренней структуры.

Для сбора статистических данных о ходе моделирования, для анализа и представления результатов имитационного эксперимента в Triad используют специальные средства – информационные процедуры и условия моделирования. Информационные процедуры и условия моделирования реализуют алгоритм исследования. Алгоритм исследования отделён от модели. Пользователь имеет возможность изменить алгоритм исследования в ходе моделирования, при этом модель остаётся неизменной, нет необходимости вносить в неё какие-либо изменения, чтобы указать алгоритму исследования те элементы модели, за поведением которых надо вести наблюдение. Управление имитационным экспериментом осуществляется в условиях моделирования. В условиях моделирования указывают условия завершения моделирования, определяют набор информационных процедур, которые осуществляют сбор информации об имитационной модели.

Необходимо отметить особенность имитационных моделей в Triad: модель не является статической. В Triad определены операции над моделями в каждом из трёх слоёв [Mikov 1995]. Это операции в слое структуре: добавление и удаление вершины, добавление и удаление полюсов, добавление и удаление дуг, рёбер, объединение графов (модель представлена в виде графа), пересечение графов и т.д. В слое рутин – это добавление и удаление событий из графа событий. В слое сообщений – добавление и удаление типов и т.д. Кроме того, в языке Triad определен оператор наложения слоя сообщений. Заменив слой сообщений новым, исследователь имеет возможность (не изменяя модели) провести эксперимент с той же моделью, но с другими правилами преобразования данных.

Подсистема доопределения имитационной модели в Triad.Net

Как уже было сказано ранее, на начальных этапах проектирования исследователь может описать модель частично, опустив описание поведения какого либо элемента модели (μ_r* = {STR,ROUT*,MES}), не указав информационные потоки, воздействующие на модель (μ_s* = {STR*,ROUT*,MES}), не определив правила преобразования сигналов в слое сообщений (μ_m* = {STR,ROUT*,MES}). Однако для запуска модели и последующего её анализа все эти элементы должны быть так или иначе (пусть приближенно) описаны. В Triad.Net доопределение выполняется подсистемой доопределения модели. На рис.1. представлен процесс обработки имитационной модели.

Следует различать автоматическое и полуавтоматическое доопределение модели.

Полуавтоматическое доопределение моделей предполагает использование условий моделирования и погружение в среду моделирования. При полуавтоматическом доопределении исследователь вводит в часть initial условий моделирования: (а) операторы наложения рутин на вершину; (б) операторы наложения слоя сообщений; (в) операторы расшифровки вершины подструктурой; (г) операторы, реализующие операции над структурой модели (добавление и удаление вершин, дуг, входов и выходов и т.д.). При выполнении процесса имитации по оператору simulate симулятор вначале выполняет доопределение модели (при обработке операторов, записанных в части initial условий моделирования). Полуавтоматическое доопределение позволяет только на один акт имитации изменить воздействие информационных потоков (расшифровка вершины подмоделью, наложение рутины на вершину) или

условия преобразования сигналов (наложение слоя сообщений). Для того, чтобы провести исследования с другими информационными потоками или с другими правилами преобразования сигналов надо запустить процесс моделирования с другими условиями моделирования: simulate M on condition of simulation New_Condition (M.N1.a,M.N2.b) M.N1.a,M.N2.b – фактические параметры – переменные модели, за которыми ведется наблюдение в системе моделирования Triad.Net.

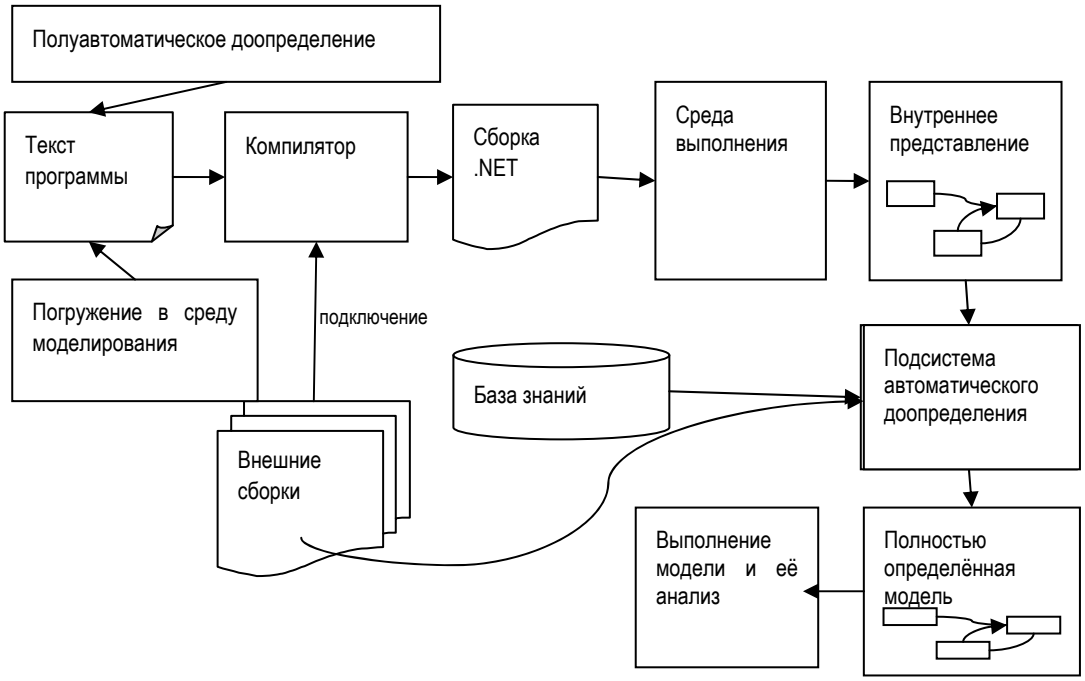

Рис.1. Структура системы доопределения моделей в Triad.Net

Погружение в среду моделирования позволяет оперативно изменить информационные потоки, поступающие на внешние полюса имитационной модели и выполняется с помощью оператора расшифровки вершины v графом, который представляет собой внутреннюю структуру моделируемого объекта, представленного вершиной v.

Автоматическое доопределение модели предполагает, что пользователь работает с частично описанной моделью μ_r^*, в которой не определены алгоритмы поведения некоторых элементов. Во время компиляции компоненты подсистемы автоматического доопределения моделей выявляют вершины v_i, для которых исследователем не определены рутины $ri = f(v_i)$, i=1..n. Задача подсистемы автоматического доопределения– найти по определенным критериям подходящие рутины в базе экземпляров рутин и достроить модель.

Подсистема автоматического доопределения модели

Рассмотрим пример моделирования компьютерной сети, представленной на рис. 2. и описание этого фрагмента на языке Triad. Каждая рабочая станция имеет два соседних узла: рабочую станцию и маршрутизатор. Сообщение должно быть передано от одной рабочей станции другой (не соседней). При передаче сообщений компьютерная сеть использует маршрутизатор. Точный сценарий поведения маршрутизаторов (Router) исследователю неизвестно. Задача системы автоматического доопределения модели состоит в том, чтобы для каждой вершины Router подобрать подходящую рутину из базы экземпляров рутин и выполнить действия, определенные оператором наложения рутины.

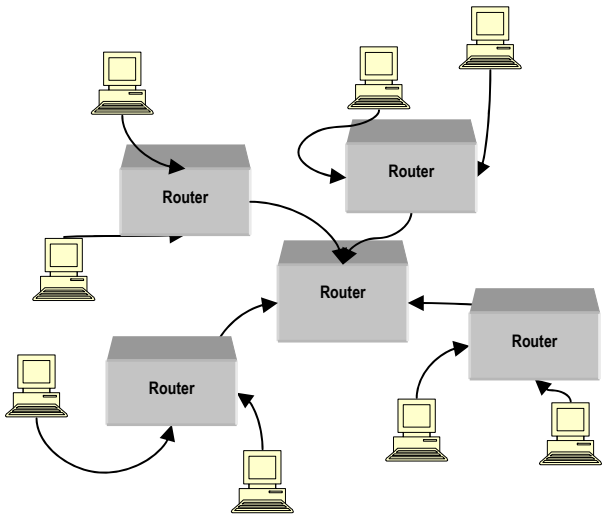

Рис.2. Фрагмент компьютерной сети

```
Type Router,Host; integer i;
   M:=dStar(Rout[5]<Pol[4]>);
   M:=M+node Hst[8]<Pol>;
   M.Rout[0]=>Router;
   for i:=1 by 1 to 4 do
       M.Rout[i]=>Router;
       M:=M+edge(Rout[i].Pol[1]—Hst[2*i-2]);
       M:=M+edge(Rout[i].Pol[2]—Hst[2*i-1]);
   endf;
   for i:=0 by 1 to 7 do
       M.Hst[i]=>Host;
   endf;
```

Рис.3. Описание фрагмента компьютерной сети на языке Triad с использованием семантических типов

Автоматическое доопределение в Triad выполняется на основании дополнительной семантической информации. Семантическая информация включает такое понятие как семантический тип. Семантический тип вводится для того, чтобы сгруппировать ряд объектов по некоторому смысловому, структурному, поведенческому типам. Так, для обозначения множества процессорных устройств при моделировании вычислительных систем, может быть введён семантический тип Процессор, для обозначения элементов памяти – тип ***МодульПамяти***. При моделировании систем массового обслуживания уместны будут семантические типы Очередь, ***ГенераторЗаявок*** и т.п. Для того, чтобы причислить объект к тому или иному семантическому типу, в тексте программы употребляется специальный оператор: <имя объекта> => <имя типа>. Семантические типы объявляются специальным оператором type <имя типа>. На рис. 3. приведен фрагмент программы, использующей семантические типы. В результате выполнения этой программы будет построена структура, описывающая небольшую сеть, терминальным вершинам которой будет присвоен семантический тип Host, а промежуточным – Router (Рис.4).

Семантические типы определяют смысловую нагрузку того или иного объекта модели. Для поиска экземпляра рутины используют базу знаний, представленной в виде онтологий. В этих онтологиях описывают семантические типы, отношения наследования между ними, а так же множества соответствующих этим типам экземпляров рутин, и семантической информации, необходимой для проверки условий доопределения. Семантических типы представлены в виде иерархии классов онтологии. Использование такого подхода предполагает, что дочерние семантические типы будут описывать понятия, конкретизирующие понятия, соответствующие родительским типам. Например, при моделировании вычислительных систем, на верхние уровни иерархии будут помещены семантические типы, соответствующие наиболее базовым понятиям, например Устройство. Дочерние типы будут представлять более конкретные понятия моделируемой предметной области: Устройства разделятся на ***Процессор, МодульПамяти*** и т.д., процессоры могут быть классифицированы в зависимости от их архитектуры. Таким образом, семантический тип представляется в базе знаний как класс объектов, являющийся подклассом общего типа Object, соответствующего всему множеству вершин. При определении семантических типов возможно указание нескольких семантических типов для одного объекта.

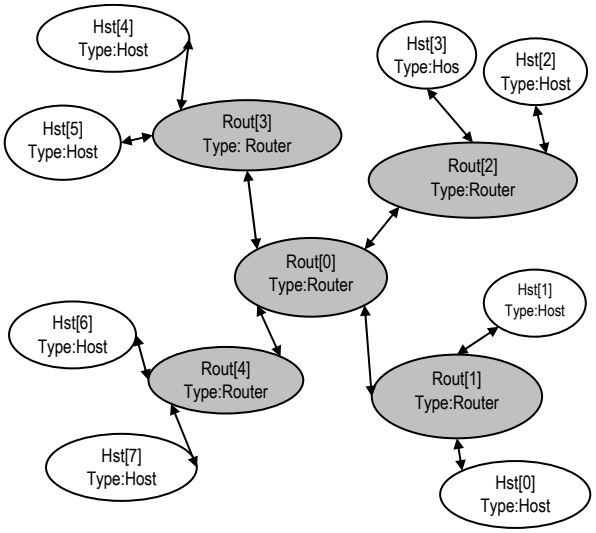

Рис.4. Внутреннее представление фрагмента модели, описанной программой на рис.3.

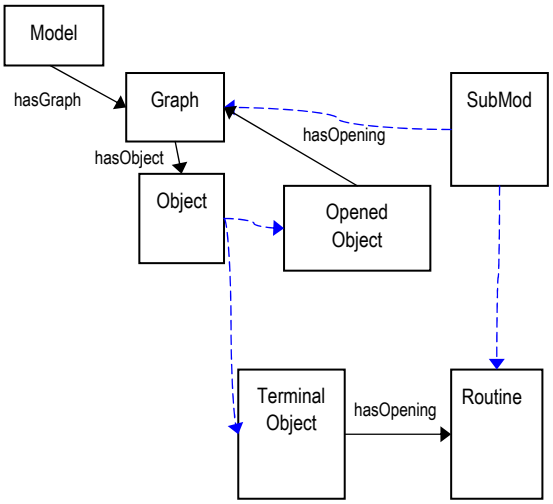

Рис.5. Фрагмент базовой онтологии

В системе Triad.Net выделены условия доопределения терминальной вершины экземпляром рутины: условия специализации, условия конфигурации и условия декомпозиции:

-- ***Условия специализации***. Пусть v – терминальная вершина, а r – экземпляр рутины, который соответствует этой вершине. Введём функцию equtype(v,r), определяющую выполнение условия специализации. Эта функция будет считаться истинной, если семантический тип Type(v), приписанный вершине, соответствует семантическому типу Type(r), соответствующему экземпляру рутины r, найденному в базе знаний. Семантический тип T1 соответствует семантическому типу T2, если T1 является суперклассом T2 (т.е. $T_2 \subset T_1$). Таким образом, условие специализации выполняется, если найденный экземпляр рутины соответствует семантическому типу вершины, или более частному типу.

-- ***Условия конфигурации***. Условие конфигурации предполагает проверку количества входных и выходных полюсов вершины и экземпляра рутины. При наложении рутины r на вершину v определяются отношения:

$$L_i : In(v) \to In(r) \tag{1}$$

$$L_o : Out(v) \to Out(r) \tag{2}$$

Отметим, что эти отображения не являются функциональными. Они задают множество связанных пар $(p_1; p_2)$, таких что $p_1 \in v; p_2 \in Pol(r)$, при этом каждый полюс может входить в любое количество пар, или не участвовать в отношении вообще. Пусть D(Li/o) – мощности множеств входных и выходных полюсов вершины, участвующих в отношениях Li и Lo соответственно. В зависимости от этих величин, вершина на которую предполагается наложить рутину должна иметь определённые количества входных и выходных полюсов, а именно, необходимо выполнение следующих условий:

$$\left|In(v)\right| \geq \left|D(L_i)\right| \tag{3}$$

$$\left|Out(v)\right| \geq \left|D(L_o)\right| \tag{4}$$

Использование нестрогого равенства позволяет накладывать экземпляры рутин на вершины, имеющие избыточное количество полюсов. При этом часть «лишних» входов и выходов вершины останется «висячими», т.е. не расшифрованными полюсами рутины, а сообщения, приходящие на эти полюса, не будут обрабатываться рутиной.

-- ***Условия декомпозиции***. Для определения условий декомпозиции, введём понятие графа окружения вершины v. Пусть $G = \{V; W; U\}$ - граф, которому принадлежит вершина ($v \in V$). Отношение S определяет смежность вершин в графе G, т.е.:

$$\forall v_1, v_2 \in V : (v_1; v_2) \in S \leftrightarrow \exists p_1 \in v_1, \exists p_2 \in v_2 : ((p_1; p_2) \in W \vee (p_2; p_1) \in W)$$

Функция Sub(w,v) определяет множество полюсов вершины w, соединённых с полюсами v:

$$Sub(w, v) = \{p \in w \mid \exists p_0 \in v : (p; p_0) \in W \vee (p_0; p) \in W\} \quad (5)$$

Тогда граф $GG(v) = \{V'; W'; \varnothing\}$ - граф окружения v, если выполняются следующие условия:

$$V' = \{v\} \cup \{w' \mid w' = Sub(w, v); (w; v) \in S\} \quad (6)$$

$$\forall w : (w; v) \in S \rightarrow Sub(w, v) \in V' \quad (7)$$

$$\forall p_1, p_2 : [(p_1; p_2) \in W] \wedge [p_1 \in v \vee p_2 \in v] \rightarrow (p_1; p_2) \in W' \quad (8)$$

$$W' \subset W \quad (9)$$

(6) ограничивает множество вершин графа окружения только самой вершиной v, и подмножествами вершин, смежных с ней. При этом учитываются только те полюса смежных вершин, которые, согласно (5), непосредственно связаны с полюсами v.(7) указывает, что такое подмножество включается для каждой из смежных с v вершин.(8) говорит, что каждая дуга, входящая или исходящая из полюсов v, будет включена в граф окружения. Из (6) и (7), очевидно, что все соответствующие полюса будут включены в граф. (9) утверждает, что никаких дополнительных дуг в графе окружения нет, т.е. включаются только дуги, соответствующие условию (8). Выполнение условия декомпозиции определяется значением функции iso(GG'(r),GG(v)), которое является истинным, если в графе окружения вершины v, есть изоморфный графу GG'(r) подграф. Функция проверяет также выполнение некоторых дополнительных требований, которым должен удовлетворять граф окружения. Граф GG' должен храниться в базе знаний вместе с экземпляром рутины. Таким образом, работа системы доопределения модели заключается в том, чтобы по сохранённым в базе знаний информации об условиях специализации, конфигурации и декомпозиции, для всех выявленных компилятором вершин без сценария поведения, найти соответствующие экземпляры рутины из базы знаний.

Представление знаний и базовая онтология

Для представления семантических знаний, необходимых для доопределения моделей, были выбраны онтологии, для представления онтологий – язык OWL[Dean, 2002], поскольку существует большое количество инструментальных средств работы с онтологиями OWL, поддерживающих возможность публиковать созданные онтологии в сети Internet и объединять информацию из различных источников, как локальных, так и находящихся в глобальной сети. Для работы с онтологиями используется инструментарий Jena OWL API. На данном этапе разработки онтологии сохраняются в виде текстовых файлов в формате представления Notation 3 (N3).

Семантические знания, которые необходимы для проверки условий автоматического доопределения, необходимо онтологическое описание слоя структур системы Triad.Net. В качестве такого описания используется базовая онтология, импортируемая всеми создаваемыми онтологиями. В этой онтологии определены следующие классы: Model (класс, описывающий множество моделей языка Triad), SubMod (класс, описывающий множество всех экземпляров рутин и структур), Graph (класс, описывающий множество структур моделей, является подклассом SubMod), Routine (множество экземпляров рутин, является подклассом SubMod), Object (множество всех вершин структуры модели, является суперклассом для всех семантических типов) и т.д. Фрагмент базовой онтологии представлен на рис.5. Для работы с самими онтологиями реализован класс OntoManager, поддерживающий загрузку нескольких онтологий, создание и сохранение онтологий. В классе TypeManager описаны функции, используемые классом, сохраняющим экземпляры рутин и классом доопределения для работы с семантическими типами вершин и экземпляров рутин.

Заключение

Подсистема доопределения модели позволяет автоматизировать процесс генерации моделей. Особенно это полезно в условиях неопределенности, когда исследователю неизвестны некоторые характеристики имитационной модели. В Triad.Net можно выполнить полуавтоматическое и автоматическое доопределение модели. Автоматическое доопределение выполняют на основе онтологий. Онтологический подход дает возможность доопределить модель, описанную пользователем лишь частично, а именно, с помощью онтологий найти в базе знаний фрагмент программы, который заменит рутину, описывающую наиболее точно сценарий поведения некоторого элемента модели.

Благодарности

Работа выполнена при финансовой поддержке грантов РФФИ 08-07-90005-Бел_а и 08-07-90006-Бел_а.

Библиографический список

[Fishwick, 2004] Fishwick P.A.. Ontologies For Modeling And Simulation: Issues And Approaches /Paul A. Fishwick, John A. Miller // Proceedings of the 2004 Winter Simulation Conference. pp. 259-264

[Benjamin, 2006]Benjamin P., Patki M., Mayer R.. Using Ontologies For Simulation Modeling. Proceedings of the 2006 Winter Simulation Conference/ L. F. Perrone, F. P. Wieland, J. Liu, B. G. Lawson, D. M. Nicol, and R. M. Fujimoto, eds. – pp.1161-1167

[Mikov, 1995] Mikov A.I. Simulation and Design of Hardware and Software with Triad// Proc.2nd Intl.Conf. on Electronic Hardware Description Languages, Las Vegas, USA, 1995. pp. 15-20.

[Dean, 2002] Dean M., Connolly D., van Harmelen F., et al. 2002. Web Ontology Language (OWL) Reference Version 1.0. W3C. //www.w3.org/TR/2002/owl-ref/

[Liang, 2003] Liang Vei-Chung. A Port Ontology For Automated Model Composition / Vei-Chung Liang, Christiaan J.J. Paredis // Proceedings of the 2003 Winter Simulation Conference, - pp. 613-622

Сведения об авторах

Александр Миков *– АНО «Институт компьютинга», директор; Россия, г. Краснодар, ул. Аксайская, 40/1-28; e-mail: alexander_mikov@mail.ru*

Елена Замятина *– Пермский государственный университет, доцент кафедры математического обеспечения вычислительных систем, Россия, г. Пермь, 614017, ул.Тургенева, .33–40; e-mail: e_zamyatina@mail.ru*

Евгений Кубрак *– Пермский государственный университет, выпускник кафедры математического обеспечения вычислительных систем, Россия, г. Пермь, ул. Леонова, .20-5; e-mail: g_brick@mail.ru*