Знания-Онтологии-Теории (ЗОНТ-09)

Использование Онтологий в Распределенной Системе Имитационного Моделирования Triad.Net

Миков А.И.[1], Замятина Е.Б.[2], Кубрак Е.Н.[2]

[1]*Кафедра Вычислительных Технологий, Кубанский Государственный Университет, ул. Ставропольская, д. 149, г. Краснодар, 350040, Россия.*
[2]*Кафедра Математического Обеспечения Вычислительных Систем, Пермский Государственный Университет, ул. Букирева, д. 15, г. Пермь, 614990, Россия*

alexander_mikov@mail.ru, e_zamyatina@mail.ru

Аннотация. *Системы имитационного моделирования являются одним из важнейших универсальных средств исследования больших (сложных) систем, функционирующих в реальном времени. Имитационная модель сложной системы также является сложной и требует больших затрат времени исследователя для ее построения. Она также требует больших затрат машинного времени в сеансе имитации, что обуславливает использование многопроцессорных машин или компьютерных сетей. Предлагается использовать онтологии предметных областей для повышения эффективности работы с имитационными моделями. Онтологии в сочетании с разработанным ранее языком моделирования Triad позволяют аналитику описывать лишь существенные черты модели, полагаясь на ее автоматическое доопределение средствами системы имитационного моделирования. Онтологии также позволяют ускорить отладку модели, давая информацию о вероятных ошибках и локализуя их. Знания о модели, сосредоточенные в базе знаний, позволяют эффективно управлять балансировкой нагрузки в распределенных системах имитационного моделирования. Предложенные методы реализованы в распределенной системе Triad.Net.*

Ключевые слова: имитационное моделирование, язык моделирования, распределенные системы, мультиагентные системы, онтология, доопределение, балансировка нагрузки

1 Введение

Имитационное моделирование является известным, широко распространенным, а иногда и единственным методом исследования явлений, процессов, ситуаций или сложных динамических систем. На протяжении примерно уже двух десятилетий разработчики систем имитационного моделирования (СИМ) наряду с классическими методами имитационного моделирования широко используют методы искусственного интеллекта (экспертные системы, нейронные сети, генетические алгоритмы, мультиагентный подход, онтологии).

Разработчики распределенной системы имитационного моделирования с удаленным доступом Triad.Net (разработка ведется группой преподавателей и студентов Кубанского и Пермского университетов) также успешно вводят в реализуемое ими программное обеспечение компоненты, поддерживающие методы искусственного интеллекта. В предлагаемом авторами докладе рассматриваются вопросы применения онтологического подхода. Авторы демонстрируют применение онтологий при доопределении имитационных моделей, в отладчике имитационных моделей и для автоматического определения правил, используемых для управляемой балансировки распределенной модели.

2 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 [1] .

Примерами использования онтологий моделирования могут служить управляемые онтологиями среды моделирования, а также подходы к объединению различных федератов, разрабатываемые для High Level Architecture (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 [2] 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 [3].

3 Имитационная модель в Triad

Имитационная модель в языке моделирования Triad [9] представляет собой совокупность объектов, которые действуют по определённым сценариям и обмениваются информацией друг с другом. Модель может быть представлена тройкой: $\mu = \{Str, Rout, Mes\}$ – слоями структур, рутин и сообщений.

Слой структур Str описывает набор моделируемых объектов и связей между ними, слой рутин Rout представляет собой набор алгоритмов поведения моделируемых объектов, а слой сообщений Mes даёт возможность описывать сообщения сложной структуры, передаваемые через полюсы объектов. Моделируемые объекты часто имеют иерархическую многоуровневую структуру. Имитационная модель также является иерархической. Каждый из уровней можно описать как граф с полюсами $G = \{U, V, W\}$. Здесь V – множество вершин графа, каждая вершина представляет собой моделируемый объект, который находится на конкретном уровне иерархии. W – набор дуг (ребер), связывающих вершины графа (моделируемые объекты). U – набор внешних полюсов, через которые передаются сообщения. Вершина модели может быть расшифрована подструктурой (операция расшифровки – введение новых уровней иерархии). При этом устанавливается соответствие между полюсами вершины и внешними полюсами расшифровывающего графа, таким образом, объект-граф «общается» с окружением через интерфейс, предоставленный объектом-вершиной.

Рутина представлена множествами классов 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 поведение объекта структуры. Так же устанавливается соответствие между полюсами вершины и полюсами наложенной на неё рутины. Наложение рутины возможно только на терминальную вершину модели (т.е. не расшифрованную какой-либо структурой), так как поведение расшифрованной вершины определяется поведением вершин её внутренней структуры.

Для сбора данных о ходе моделирования, для анализа и представления результатов имитационного эксперимента в модели Triad используются специальные средства – информационные процедуры и условия моделирования. Информационные процедуры и условия моделирования реализуют алгоритм исследования моделируемого объекта (процесса). 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.

4 Архитектура распределенной системы Triad.Net

Распределенная система имитации Triad.Net включает следующие компоненты: компилятор TriadCompile; ядро TriadCore; графический редактор; подсистему отладки и валидации имитационных моделей TriadDebugger; подсистему синхронизации распределенных объектов модели; подсистему балансировки TriadBalance [4,5]; подсистему организации удаленного доступа TriadEditor [6], подсистему защиты от внешних и внутренних угроз TriadSecurity [7], подсистему автоматического доопределение модели TriadBuilder [8], а также базу данных, где хранятся экземпляры элементов модели. Компоненты распределенной системы имитации представлены на рис.1.

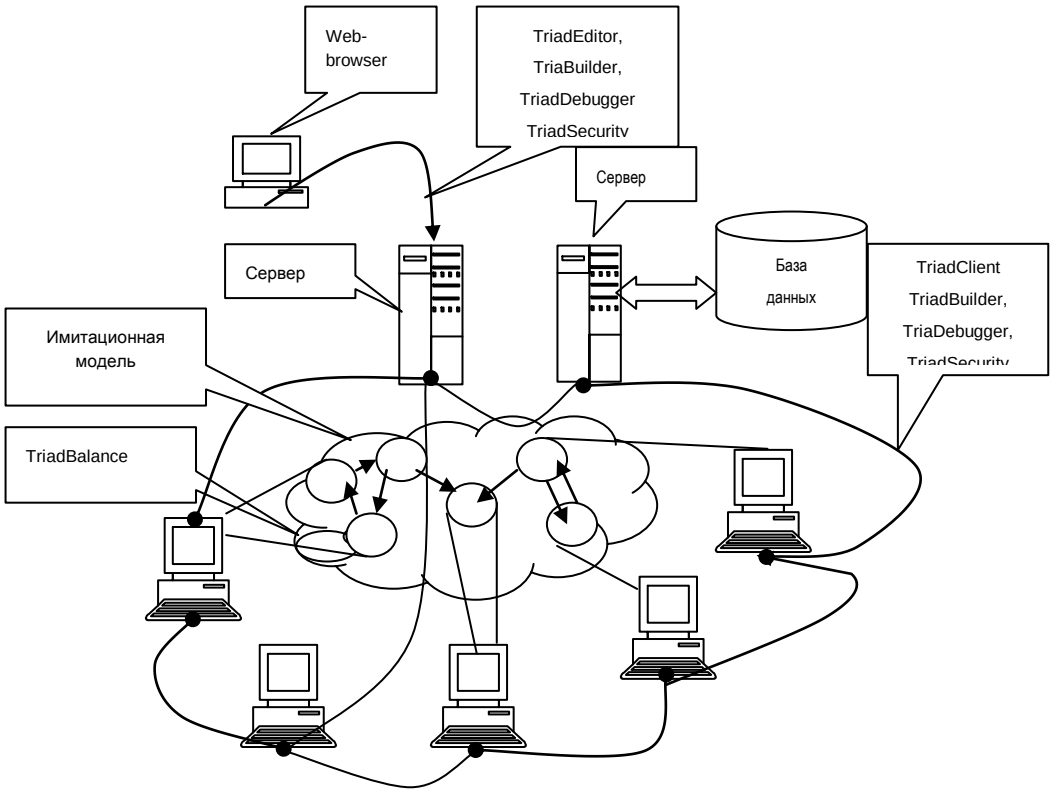

Рис.1. Архитектура системы имитационного моделирования Triad.Net

5 Онтологии в доопределении моделей

5.1 Проблема доопределения модели

При создании моделей исследователи часто сталкиваются с ситуацией необходимости анализа не полностью описанной модели. Не полностью описана может быть структура модели. Обозначим такую модель μs* = {Str*, Rout, Mes}. В модели с не полностью описанной рутиной μr* = {Str, Rout*, Mes} неизвестны в точности правила функционирования (поведение)

отдельных элементов. Например, при моделировании компьютерных сетей проектировщик не всегда может точно задать алгоритм работы маршрутизатора. На ранних стадиях проектирования достаточно довольно грубо определить его поведение. Для исследователя важно, чтобы данные передавались по сети, а конкретный алгоритм маршрутизации его на этом этапе не интересует, исследователь абстрагируется от таких деталей. Может отсутствовать и детальное описание структуры сообщений – модель μm* = {Str, Rout, Mes*}. 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 моделирования систем вида μr* = {Str, Rout*, Mes} 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.

5.2 Метод доопределения, реализованный в TriadBuilder

Автоматическое доопределение модели предполагает, что пользователь работает с частично описанной моделью μr* = {*Str*, *Rout**, *Mes*}, 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 $r_i = f(v_i)$, $i = 1..n$. Задача подсистемы автоматического доопределения – найти по определенным критериям подходящие рутины в базе экземпляров рутин и достроить модель.

В Triad.Net доопределение выполняется подсистемой доопределения модели (рис. 2).

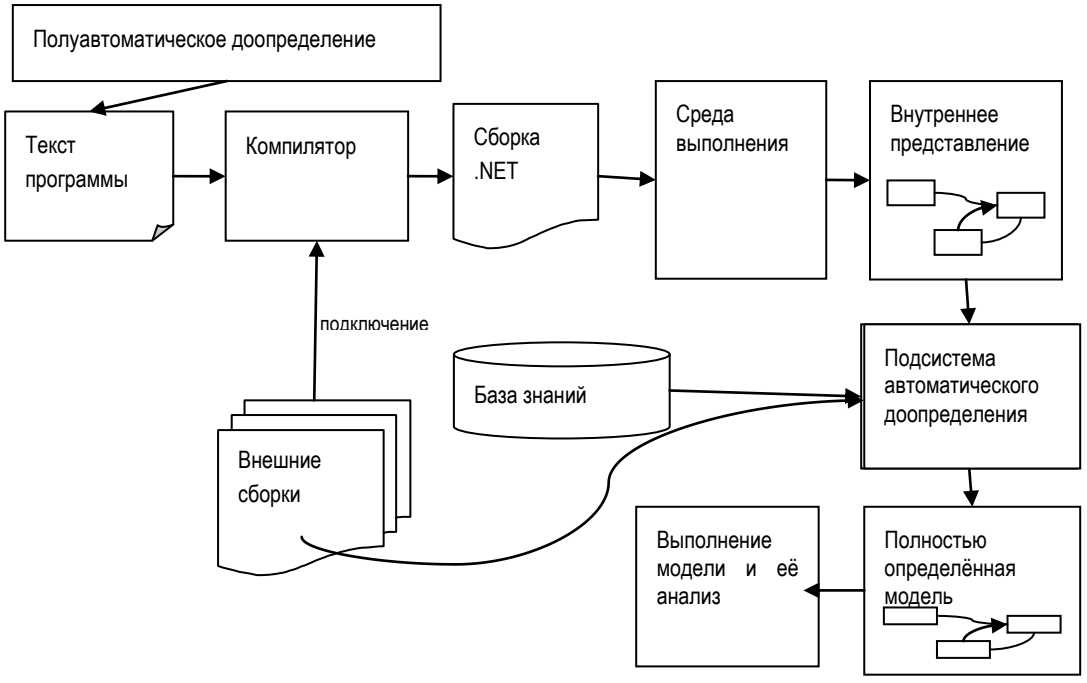

Рис.2. Структура системы доопределения моделей в Triad.Net

Рассмотрим пример моделирования компьютерной сети (рис. 3) и описание этого фрагмента на языке Triad (рис. 4). Каждая рабочая станция имеет два соседних узла: рабочую станцию и маршрутизатор. Сообщение должно быть передано от одной рабочей станции некоторой другой станции. При передаче сообщений компьютерная сеть использует маршрутизатор. Точный сценарий поведения маршрутизаторов (Router) исследователю неизвестен. Задача системы автоматического доопределения модели состоит в том, чтобы для каждой вершины типа Router подобрать подходящую рутину из базы экземпляров рутин и выполнить действия, определенные оператором наложения рутины.

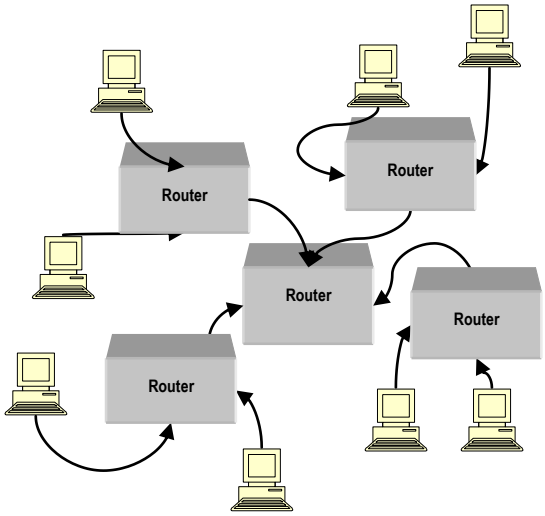

Рис 3. Фрагмент компьютерной сети

```
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.4. Описание фрагмента компьютерной сети на языке Triad

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. 4. 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.5).

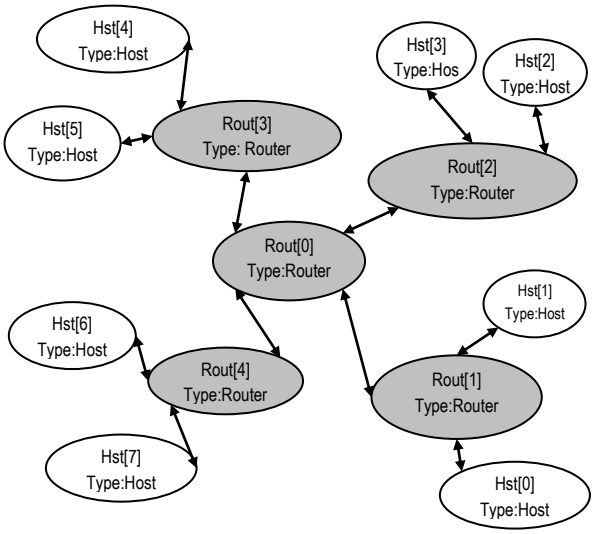

Рис 5. Внутреннее представление фрагмента модели, описанной программой на рис 4

Семантические типы определяют смысловую нагрузку того или иного объекта модели. Для поиска экземпляра рутины используется база знаний, представленная в виде онтологий. В этих онтологиях описываются семантические типы, отношения наследования между ними, а так же множества соответствующих этим типам экземпляров рутин, и семантической информации, необходимой для проверки условий доопределения. Семантические 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 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.

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 T_1 соответствует семантическому типу T_2 , если T_1 является суперклассом T_2 (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:

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

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

Пусть $D(L_i)$, $D(L_o)$ – мощности множеств входных и выходных полюсов вершины, участвующих в отношениях L_i и L_o 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:

$$|In(v)| \geq |D(L_i)| \quad (3)$$

$$|Out(v)| \geq |D(L_o)| \quad (4)$$

Использование неравенства 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* = {*U, V, W*} - 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*'} - граф окружения *v*, если выполняются следующие условия:

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

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

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

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

Условие (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*. Условие (7) указывает, что такое подмножество включается для каждой из смежных с *v* вершин. Условие (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. Условие (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 сохранённой 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.

5.3 Представление знаний и базовая онтология

Для представления семантических знаний, необходимых для доопределения моделей, были выбраны онтологии, для представления онтологий – язык OWL, 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 Notation3 (N3).

Онтологическое описание слоя структур системы Triad.Net необходимо для проверки семантических условий доопределения модели. В качестве такого описания используют базовую онтологию, импортируемую 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.6.

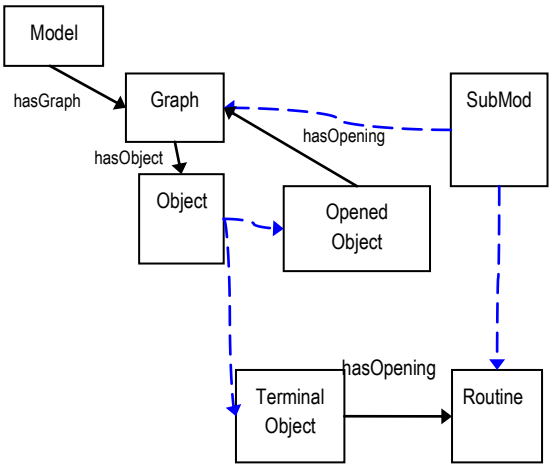

Рис.6. Фрагмент базовой онтологии

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.

6 Онтологический подход к выявлению и локализации ошибок

Построенная модель должна быть проверена на валидность, т.е. определение соответствия поведения построенной модели функционированию моделируемого объекта. Для определения валидности модели в подсистеме TriadDebugger системы Triad.Net используется механизм информационных процедур. При использовании информационных процедур возможно выявление многочисленных типов ошибок, в том числе: некорректное определение временных задержек; определение неверных путей передачи данных; семантическая некорректность обмена сигналами; ошибки управления функционированием модели; запрещенные состояния модели; семантическая некорректность смены состояний моделируемой системы.

В приведенной выше модели (рис.4) целесообразно определить, не превышает ли временной интервал при посылке сообщения от одной рабочей станции к другой некоторого предельного значения; верно ли был передан сигнал от одной рабочей станции другой, получен ли нужный сигнал на конкретном входе рабочей станции и т.д.

Для определения некорректных временных задержек система Triad.Net располагает стандартными информационными процедурами, которые использует отладчик имитационной модели, к примеру: more_interval(t)[e1,e2] (если временной интервал между двумя событиями e1(например, событие посылки сообщения) и e2(событие приема сигнала) превышает допустимое значение t, то отладчик сигнализирует об ошибке). Стандартные процедуры фиксируют моменты времени выполнения конкретного события, изменения значений переменных: all_events(e_i), all_changes(var_i). Процедура value_in_out(val1, val2) [p1, p2, t] определяет правильность преобразования сигнала, поступившего на входной полюс p1 и выработанного на выходном полюсе p2. Значения сигналов определяются фактическими настроечными параметрами val1, val2. Кроме того, процедура фиксирует интервал времени, необходимый для преобразования сигнала. Полюса p1, p2 могут принадлежать разным вершинам.

Следует отметить, что информационные процедуры являются единственным средством, которые в один и тот же момент времени имеют доступ к разным рутинам. Таким образом, с помощью информационных процедур можно определить запрещенные состояния в модели.

Для локализации ошибок используется экспертный компонент, включающий базу знаний, в которой хранятся правила локализации ошибок. Предположим, что на выходном полюсе рабочей станции (рис.4) получен сигнал val1, причем значение его не соответствует

ожидаемому. Для того, чтобы понять, где произошла ошибка, необходимо проверить цепочку передачи сигнала, т.е. значения сигналов на всех полюсах вершин, через которые идет сигнал. Кроме того, ошибка может быть локализована при проверке результатов выполнения цепочки событий, которые выполняются при преобразовании сигналов.

Для правил, которые используются для локализации ошибок, построены онтологии. При обнаружении ошибки конкретного типа выполняется запрос к онтологиям и из базы знаний извлекаются соответствующие правила.

7 Онтологический подход к балансировке нагрузки

Имитационная модель в Triad.Net является распределенной, т.е. объекты имитационной модели во время выполнения могут располагаться на различных вычислительных узлах. Для синхронизации объектов в системном времени используют специальные алгоритмы (консервативный и оптимистический). Использование ресурсов нескольких вычислительных узлов позволяет оптимизировать время выполнения имитационного эксперимента и повышает его надежность.

Однако гетерогенность вычислительной системы (ряд вычислительных узлов являются перегруженными, в то время как другие простаивают), на которой выполняется моделирование и гетерогенность имитационной модели могут свести выигрыш от использования ресурсов нескольких компьютеров (процессоров) к минимуму. Гетерогенность имитационной модели выражается в том, что часть логических процессов модели все время выполняют вычисления, в то время как другие большее время пребывают в ожидании, часть процессов интенсивно обмениваются сообщениями друг с другом, в то время как другие посылают и принимают сообщения достаточно редко.

Для решения этой проблемы рекомендуется выполнять балансировку вычислительной нагрузки на узлах компьютерной сети (многопроцессорной ЭВМ) с помощью подсистемы TriadBalance. Сбалансированность предполагает равномерное распределение вычислительной нагрузки на узлы ВС и нагрузки по передаче данных на каналы связи.

Различают статическую и динамическую балансировки. Статическая балансировка выполняется до начала имитационного эксперимента. При распределении логических процессов по вычислительным узлам (по рабочим станциям в сети или по процессорам многопроцессорной ВС) учитывается опыт предыдущих имитационных прогонов.

Динамическая балансировка выполняется во время имитационного прогона. При возникновении дисбаланса необходимо восстановить равномерную нагрузку на вычислительные узлы. В этом случае логический процесс должен быть перенесен с загруженного вычислительного узла на менее загруженный. При интенсивной работе линий связи требуется разгрузить их, а именно, перенести интенсивно обменивающиеся сообщениями логические процессы на один узел (или на другие узлы, соединенные линией связи с более мощной пропускной способностью). Перенос логического процесса называется «миграцией».

Динамическая система балансировки TriadBalance является мультиагентной, она состоит из совокупности агентов разных типов: агента-датчика вычислительного узла; агента-датчика имитационной модели; агента анализа; агента миграции; агента распределения.

Агенты каждого типа действуют по своему сценарию для достижения цели, а вместе они реализуют балансировку распределенной имитационной модели.

Агент-датчик вычислительного узла постоянно собирает информацию о нагрузке вычислительного узла, о состоянии линий связи, о наличии свободной памяти. Агент-датчик вычислительного узла взаимодействует с агентом анализа. Агент-датчик имитационной модели ведет постоянное наблюдение за объектами имитационной модели во время имитационного прогона, регистрируя интенсивность обмена между объектами, частоту выполнения тех или иных событий, частоту изменения переменных и т.д. Агент-датчик имитационной модели использует механизм информационных процедур. Агент-датчик имитационной модели взаимодействует с агентом распределения.

Агент анализа, взаимодействуя с агентом - датчиком вычислительного узла (этот агент является реактивным), принимает решение о необходимости перераспределения нагрузки. Агент анализа является когнитивным агентом и, принимая решения, использует правила из

базы знаний экспертной системы. Агент распределения получает информацию от агента анализа. Цель агента распределения – выявить порцию нагрузки (выбрать объекты имитационной модели) на вычислительном узле, которую следует передать другим узлам, чтобы избежать дисбаланса, и определить целевой узел, на который следует перенести нагрузку. Этот агент также является когнитивным. Агент миграции выполняет перенос нагрузки (он является реактивным).

Правила, по которым действуют агенты, извлекаются из базы знаний. Правила могут быть определены для конкретного вида моделей, для конкретных топологий вычислительных систем и моделей. Предлагается использовать онтологию имитационных моделей и различные механизмы привязки конкретной модели к этой классификации. Выбирая походящее действие по балансировке из базы правил с учетом онтологии имитационных моделей, мы получим большую гибкость и адекватность механизмов балансировки. Как и в предыдущих случаях, программное обеспечение Triad.Net должно обеспечивать работу с внешними онтологиями.

8 Заключение

Сочетание методов имитационного моделирования с методами, относящимися к представлению знаний, дает значительный эффект в работе исследователя. В работе на примере распределенной системы имитационного моделирования Triad.Net показано применение онтологий предметных областей при реализации подсистемы доопределения модели. С помощью онтологий в базе знаний автоматически отыскивается фрагмент программы, описывающий наиболее точно сценарий поведения некоторого элемента модели. В реализации интеллектуального отладчика онтологии позволяют выбрать в базе знаний правила для локализации ошибки в конкретной ситуации. Подсистема балансировки нагрузки в распределенной системе функционирует на основе выбора правил когнитивных агентов для исследуемых имитационных моделей.

9 Благодарности

Авторы благодарят Российский Фонд Фундаментальных Исследований за финансовую поддержку работы (гранты РФФИ 08-07-90005 Бел_а и 08-07-90006 Бел_а).

Литература

[1] Fishwick P.A., Miller J.A.: Ontologies for Modeling and Simulation: Issues and Approaches. In: *Proceedings of*, Winter Simulation Conference, p. 259-264, 2004.

[2] Benjamin P., Patki M., Mayer R.: Using Ontologies for Simulation Modeling. In: *Proceedings of*, Winter Simulation Conference, p.1161-1167, 2006.

[3] Liang V.-C., Paredis C.J.: A Port Ontology For Automated Model Composition. In: *Proceedings of*, Winter Simulation Conference, p. 613-622, 2003.

[4] Миков А.И., Замятина Е.Б., Осмехин К.А.: Динамическое распределение объектов имитационной модели, основанное на знаниях. In: *Proceedings of*, XIII International Conference “Knowedge-Dialogue-Solution” KDS, Vol.2., p.618-624, ITHEA, Sofia, 2007.

[5] Миков А.И., Замятина Е.Б., Козлов А.А.: Оптимизация параллельных вычислений с применением мультиагентной балансировки.//Труды конференции ПАВТ-2009, с. 599-604, Нижний Новгород, Россия, 2009.

[6] Mikov A., Zamyatina E., Firsov A.: Software for Remote Parallel Simulation. International Journal «Information Theories & Applications», Vol.14, № 4, p. 389-395, 2007.

[7] Миков А.И., Замятина Е.Б., Панов М.П.: Мультиагентная система защиты имитационной модели с удаленным доступом.//Труды Международных научно-технических конференций «Интеллектуальные системы» (AIS’07) и «Интеллектуальные САПР» (CAD-2007), Т1, с.323-330, -М.: Физматлит, 2007.

[8] Mikov A., Zamyatina E., Kubrak E.: Implementation of simulation process under incomplete knowledge using domain ontology. In: *Proceedings of*, 6-th EUROSIM Congress on Modeling and Simulation. Vol.2. Book of full papers, p. 1-7, University of Ljubljana, Slovenia, 2007.

[9] Mikov A.I.: Simulation and Design of Hardware and Software with Triad. In: *Proceedings of*, 2nd Intl. Conf. on Electronic Hardware Description Languages, p. 15-20, Las Vegas, USA, 1995.