

# ОБОБЩЕННАЯ ОНТОЛОГИЯ ИМИТАЦИОННОГО МОДЕЛИРОВАНИЯ ДЛЯ СОЗДАНИЯ ИМИТАЦИОННЫХ МОДЕЛЕЙ ТЕХНОЛОГИЧЕСКОГО ПРОЦЕССА С ИСПОЛЬЗОВАНИЕМ ЕСТЕСТВЕННОГО ЯЗЫКА

С. В. Рудометов, В. В. Окольников

*Институт вычислительных технологий СО РАН, 630090, Новосибирск*

УДК 519.685.1

Современный уровень развития технологий искусственного интеллекта позволяет применить их для решения таких проблем, как максимальное упрощение взаимодействия человека и компьютера, сведение его к присущим человеку типам когнитивного взаимодействия. Пример — постановка задачи для компьютера в виде фраз на естественном языке.

В докладе приводится возможный пример организации такого взаимодействия для задач имитационного моделирования. Обсуждаются ограничения этого подхода, обнаруженные в процессе его реализации в системе MTSS. Вводится понятие “дистанции программирования” как длины своеобразного “моста” между постановкой задачи и современными возможностями вычислительной техники (включая библиотеки прикладного программирования для искусственного интеллекта), которую должен “пройти” программист для того, чтобы задача в своей начальной постановке заработала на компьютере. Показано, как максимально сократить эту “дистанцию”, а в идеале — свести ее к нулю, исключив тем самым программиста из цепочки человек–постановка задачи–имитационный эксперимент–вычислительная система.

Для сокращения дистанции программирования предложена техника создания онтологии предметной области, основанная на возможностях библиотек элементарных моделей среды MTSS. Показан способ анализа фраз на естественном языке, который позволяет “собирать” имитационную модель из экземпляров элементарных моделей, связывая их друг с другом и задавая значения их атрибутов, если требуется.

**Ключевые слова:** имитационное моделирование, семантический анализ, онтология.

## Введение

В последнее время активно развивается использование технологий искусственного интеллекта, таких как семантический анализ текстов на естественных языках.

Семантический анализ текста на естественном языке имеет смысл только в контексте какой-то задачи, сам по себе он не может существовать. Примеры задач семантического анализа: машинный перевод, поиск смысловой информации в тексте (semantic web, определение эмоционального окраса текста и т.д.), управление роботами или технологическим оборудованием.

Ранее авторами было проведено исследование [1], в котором фразы естественного языка используются для конфигурирования имитационной модели, созданной в системе MTSS [2] для специализированной библиотеки имитации технологического оборудования (в нашем случае — угледобыче в пологом пласте).

Однако, при создании элементов самой библиотеки библиотечные элементы — элементарные модели (ЭМ) в терминологии MTSS, как правило, приходится конструировать из вполне определенных базовых компонентов. Задача статьи — описать эти базовые по отношению к ЭМ компоненты в виде онтологии и представить вариант ее использования в виде описания элементов библиотеки имитационных моделей технологических объектов фразами на естественном (русском, английском) языке.

## 1 Использование естественного языка

Правильно ли вообще использовать естественный язык для описания имитационной модели как компьютерной программы? С одной стороны, естественный язык, являясь продуктом эволюции человека и человеческого общества, в своей высшей точке развития приспособлен как для обмена абстрактными понятиями, так и для создания новых абстрактных понятий, возникновение которых невозможно (прежде всего из-за ограниченности мышления) в отдельно взятом мозге индивида [3]. И именно на основе естественного языка, с его развитием, возникло современное общество, которое, в свою очередь, потребовало дальнейшего развития самого языка. Усложнение общества ставило перед его частями все более сложные задачи (прежде всего различные вычисления, от элементарных площадей до сложнейших архитектурных сооружений, от простейшей навигации до сложнейших астрономических вычислений). Развитие таких задач послужило толчком к возникновению научного мировоззрения, а также к возникновению различных наук.

Имитационные модели, как программы, исполняемые на компьютерах, создаются с использованием языков программирования. Эти языки, в свою очередь, являются продуктом именно научного мировоззрения, наследуя его основные черты и задачи. А именно, языки программирования:

- формальные. Это первое с чем сталкивается любой студент: для написания программы ему недостаточно использовать всего одну фразу (например, “Смеркалось”). Ему требуется описать весь процесс, с использованием строгих знаний, опираясь на существующие (в системе программирования или моделирования) классы и объекты, и если их нет — сначала создать и их. Например, так: “Сумерки — часть суток после заката и перед восходом Солнца, в течение которого Солнце находится за горизонтом и невидимо, но видны признаки заката, обусловленные рассеянием солнечного света в верхних слоях атмосферы Земли”. Другими словами, формализованное описание никогда не опирается на некоторое промежуточное субъективное знание, например, на личный опыт индивида. Оно уже, как правило, содержит в себе достаточно полную информацию. Именно поэтому формализованное знание может быть транслировано в компьютерный код. Причем все сказанное относится не только к описательной формализации, но и к попыткам управления компьютерной программой (в нашем случае — имитационной моделью) с помощью языковых команд: команды управления также должны быть формальными.
- используются для написания компьютерных программ, каждая из которых предназначена для решения какой-то конкретной задачи. Основные проблемы, которые решаются с применением программ, в точности совпадают с проблемами науки, в частности предсказания (“что будет если”, задачи типа what-if). Невозможно представить программу, написанную ряды программы.

Система программирования MTSS позволяет строить большие имитационные модели из малых, библиотечных, элементарных имитационных моделей (ЭМ), созданных, верифицированных и валидированных заранее. Способ построения и исполнения такой модели — визуальный. Пользователь соединяет на экране компьютера экземпляры ЭМ, подобно частям конструктора, затем задает значения свойств каждого экземпляра ЭМ, визуально контролируя процесс построения и затем исполнения имитационной модели.

Таким образом, в системе имитационного моделирования MTSS естественный язык может использоваться только:

1. В формальном виде, опираясь на сущности, определенные ранее в самой системе имитационного моделирования и в ее библиотечных элементах, как это было сделано ранее для “пологого пласта”
2. Для описания связей и взаимодействия между объектами модели (экземплярами ЭМ)
3. Для описания алгоритма работы ЭМ
4. Для управления исполнением модели (командный язык)

## 2 Базовая онтология

Попробуем оттолкнуться от иерархии в живой природе, а именно от того, как устроена нервная система живого организма. С точки зрения кибернетики и нейронных сетей, нам потребуются:

- рецепторы (входные сигналы)

- эффекторы (выходные сигналы)
- обработчик информации от рецепторов, с возможностью выдачи сигналов в эффекторы. Иногда он называется мозгом, но это не всегда так. Иногда обработчика информации от рецепторов (мозга) нет как такового, а есть только прямые связи между рецепторами и эффекторами, определенные эволюцией (пример — гидра).

Каждый субъект в формализме DEVS [4] (на котором построен MTSS) имеет входные и выходные порты. При построении имитационной модели субъекты (в MTSS — экземпляры ЭМ) соединяются своими выходами с входами других субъектов. Этим они сильно напоминают нейроны в мозгу живых существ [5], в широком смысле этого понятия (то есть, включая сюда рецепторы и эффекторы в виде входных и выходных портов). Но использование просто абстрактного субъекта, в виде его “нейросетевого” наполнения, затруднительно по причине сложности получаемой модели и необходимости “обучения” такой модели. Хотелось бы все-таки наполнить такой субъект каким-то конкретным, достаточно сложным содержанием, совершив тем самым “эволюционный скачок” на требуемый уровень. Такой скачок, если исходить из кибернетического подхода, должен сопровождаться выработкой специализированных компонент (зон в мозге или нейронной сети).

DEVS-формализм позволяет создавать модели элементов исходной моделируемой системы, которые получаются после декомпозиции (важного и необходимого шага при построении имитационной модели). В MTSS после декомпозиции выделяются, как правило:

- классы элементов исходной системы (ЭМ в MTSS). Классы определяют, прежде всего, алгоритм функционирования, а также наборы входных сигналов, выходных сигналов, и свойств (параметров) класса. Свойство класса имеет признак типа, имени, набора (интервала) значений. Классы в MTSS можно объединить в библиотеку(и).
- объекты этих классов (субъекты в DEVS), с конкретными значениями параметров класса, и конкретными соединениями с другими объектами этого или других классов. Таким образом определяется топология конкретной имитационной модели. В MTSS объекты (экземпляры ЭМ) создаются и настраиваются визуально, специалистом в предметной области, при проведении моделирования.

Целью работы является показать, что для создания классов, моделирующих элементы исходной системы, можно использовать некоторые базовые, “протоклассы”, путем их соответствующей настройки.

### 3 Устройство субъекта

DEVS-формализм является развитием дискретно-событийного подхода. DEVS описывает поведение системы на двух уровнях. В своей основе (первый уровень) система в нотации DEVS-формализма является дискретно-событийной имитационной моделью. Система входных и выходных портов, а также способ соединения простых субъектов для образования имитационной модели является DEVS-формализмом (второй уровень).

Более формально, имитационная модель представляется в виде набора

$$M = \{s_j^i, i \in Class, j \in Inst^i\} \quad (1)$$

экземпляров субъектов разных классов. ( $Class$  — множество классов объектов,  $Inst^i$  — множество экземпляров объектов класса  $i$ . Другими словами, множество  $M$  является набором экземпляров субъектов разных классов. Верхний индекс — номер класса, нижний индекс — номер экземпляра). Классы определяются на основе декомпозиции имитируемой системы.  $s_j^i$  могут соединяться друг с другом.

Каждый субъект в нотации DEVS представляется в виде набора своих важных свойств:

$$s_j^i = \{S, ta, \delta_{int}, X, \delta_{ext}, Y, \lambda\} \quad (2)$$

Здесь  $S$  — фиксированный набор состояний (как определено в Discrete Event Simulation, или сокращенно DES),  $ta$  (или time advance) — время следующего исполнения субъекта,  $\delta_{int}$  — набор входных портов,  $X$  — функция переключения состояний, как это определено для DES,  $\delta_{ext}$  — набор выходных портов,  $Y$  — функция реакции на события во входных портах,  $\lambda$  — функция отображения внутреннего состояния из множества  $S$  в набор выходных сигналов.

## 4 Онтологическое представление

Из представления (1) прямо следует, что это же определение является и онтологией имитационной модели. А именно, как следует из определения онтологии [6]

“Онтология — концептуальная схема, состоящая из

1. структуры данных, содержащей все релевантные классы объектов,
2. связи между классами
3. отношений (теорем, ограничений), принятых в данной предметной области”

Ограничивая область знаний результатами декомпозиции исходной системы, мы можем заключить, что (1) определяет искомую онтологию.

Таким образом, для формального описания имитационной модели на естественном языке нужно пройти следующие этапы:

*Этап 1.* Описать каждый класс, определенный после декомпозиции исходной системы. Для этого определить элементы этого класса:

$$s^i = \{S, X, Y, \delta_{int}, \delta_{ext}, \lambda, P, R\}, i \in Class, \quad (3)$$

здесь множества  $S, X, Y, \delta_{int}, \delta_{ext}, \lambda$  те же, что и в (2).

Множество  $P$  — это множество переменных, которые может определить сам создатель модели (очевидно, что это может быть не только программист, но и специалист в предметной области) для того, чтобы увеличить базу элементов онтологии, на которую могут опираться фразы на естественном языке. Другими словами,  $P$  — это множество определений.

Множество  $R$  называется множеством правил. Проще всего представить элементы данного множества как глаголы. Правила могут оперировать со всеми доступными данными состояния модели, включая множество  $P$ . Все описанные классы объединяются в библиотеку.

*Этап 2.* Используя библиотеку, порожденную на этапе 1, создать имитационную модель, как это уже делается в MTSS.

## 5 Дистанция программирования

Введем понятие **дистанции программирования** как некоторую несистемную характеристику, которая может показать количество или уровень трудозатрат, которые должен приложить программист (как посредник между компьютером и человеком) для реализации какой-либо идеи в виде компьютерной программы.

Эта характеристика оценивалась по-разному. Когда-то это было количество строк, затем — количество часов. На данный момент наиболее полной характеристикой является понятие **story point**, или **sp** [7], или т.н. единица производства. Story point позволяет создать наиболее точный план производства. То есть это как раз та самая характеристика, которая может показать искомый уровень трудозатрат.

Интересно, от чего зависит величина story point, и из чего состоит эта единица. Story point это:

1. Количество работы,
2. Сложность работы,
3. Любые риски и неочевидности, которые могут возникнуть в процессе работы.

Не вдаваясь в более глубокие детали, можно заметить, что, снижая значения во всех трех пунктах, можно добиться того, что программист вообще окажется не нужен.

То есть, требуется снизить не только количество работы, но и минимизировать сложность работы, а также убрать или “управлять” рисками. Все это может быть достигнуто автоматизацией.

На самом деле, такой подход не является новым. Как раз именно таким способом программист исключается из цепочки пользователь—компьютер в различных специализированных системах. Но такие системы (как например оригинальная MTSS) основаны на покрытии предметной области. Чем больше такое покрытие, тем большие возможности для имитационных экспериментов получает конечный пользователь. Такой подход имеет только одно ограничение — если пользователю все-таки чего-то не хватает, опять нужен программист.

Ниже показан пример онтологии, которая позволяет задавать имитационные эксперименты на естественном языке.

## 6 Пример описания библиотеки на естественном языке

Наиболее очевидным и как следствие — популярным способом синтаксического анализа фраз на естественном языке является т.н. “грамматика составляющих”. Эта грамматика пытается разбить предложение на более мелкие структуры, затем их — на еще более мелкие структуры, пока не дойдет до отдельных слов или минимальных синтаксических структур. Каждая такая структура отмечается своим признаком, обозначающим ее роль в предложении (POS-tagging). Из такого определения следует, что результат синтаксического разбора исходного предложения в частности зависит от порядка слов в предложении. Этот подход успешно применяется для языков, в которых правилами жестко задан порядок слов (например — английский язык).

Другой возможный подход — грамматика зависимостей. В отличие от грамматики составляющих, результатом синтаксического разбора предложения в данной грамматике будет набор зависимостей между частями предложения (в частности, это подразумевает, что для предложения уже выполнен синтаксический разбор с использованием грамматики составляющих). Грамматика зависимостей выделяет в предложении значимые части (субъект, объект), и зависимости между ними. В данном случае порядок слов не так важен, например, как в русском языке.

В системе MTSS мы попытались использовать грамматику зависимостей, потому что ее результат (зависимости) наиболее естественно может быть отображен на вызовы алгоритмических функций с определенными наборами параметров. Покажем это на примере.

Для предложения “Проходческо-очистной комбайн фронтального действия движется от точки врезки под углом А к горизонту вверх, а после достижения кровли пласта — под углом залегания пласта В — вглубь” после последовательного анализа обоими грамматиками выделяются следующие сущности:

- субъект: “Проходческо-очистной комбайн фронтального действия” (далее просто “комбайн”)
- объекты: непосредственно из фразы: “точка врезки”, “пласт”, “кровля пласта”, “угол залегания пласта”, “горизонт”.
- алгоритмы: *движется1* (<субъект> **комбайн**, <точка> **от** (точка врезки, скорость, угол(уголА, горизонт)), <точка> **до**(не определено), <направление> **направление**(вперед, угол(А к горизонту))), *движется2* (<субъект> **комбайн**, <точка> **от**(не определено), <точка> **до**(не определено, <направление> **направление**(вперед, угол(В залегания пласта))). Здесь значения, заключенные в угловые скобки < и >, обозначают стереотипы данных и служат для описания типов параметров
- как следствие из предыдущего пункта: требуется определить набор вычисляемых свойств на основании значений переменных, представленных объектами.
- условия: начало(алгоритмы: *движется1*), достижение кровли пласта (алгоритмы: *движется2*).

Исполнение модели заключается в задании набора начальных алгоритмов, последовательном, пошаговом исполнении текущего набора алгоритмов и затем — проверке условий. Условия могут переключить набор активных алгоритмов.

Как видно из этого примера, мы можем определить сложное движение субъекта имитационного моделирования, основываясь на простых алгоритмах. В данном примере это только один алгоритм — а именно функция “*движется*”

В базовой онтологии, реализованной в MTSS, записано, что для того чтобы использовать этот алгоритм, ему, как функции, требуется задать, кроме субъекта, параметры “точка начала движения”, “точка окончания движения” или “направление движения” (если не определена “точка окончания движения”). Как именно вычислять нужные для онтологической функции параметры?

К сожалению, это и есть та самая “дистанция программирования”, которую еще требуется преодолеть для реализации полностью автоматического построения модели с использованием естественного языка. Но можно заметить, насколько она сократилась по сравнению с созданием полной модели, с самого начала.

## 7 Что дальше?

Идеальным было бы в обозримом будущем перейти в создании имитационных моделей от формального языка к неформальному, который, как указано в [3], требует субъективного фильтра для правильной трактовки информации. Другими словами, “поговорить” с компьютером о создании какой-нибудь имитационной модели. Для этого в подходе, используемом MTSS, есть только некоторые части из списка:

- базовые элементы (как определено в данной статье)
- механизм накопления знаний (использование формализованных внешних источников)
- механизм создания абстракций
- механизм управления абстракциями

Ведь целью введения языков программирования было не создать отдельный мир вычислительных машин, в котором все взаимодействия формализованы. Сначала была необходимость хоть как-то “заставить” работать механический (а затем — электрический и наконец электронный) аппарат чтобы производить все более сложные вычисления. Затем, после того как компьютеры стали эффективно решать вычислительные задачи, их приспособили как инструмент в информационной эре, для сбора, поиска и обмена информацией (что мы сегодня и наблюдаем). И теперь, как представляется авторам, настала наконец пора сделать следующий “метасистемный” переход — перейти к неформальному общению с компьютерами, для которого у них есть очень большой массив накопленной информации. Вопрос в том — а кому и зачем нужен компьютер, который может поговорить на отвлеченные темы? Наша система предлагает использовать такое общение для создания имитационных моделей.

## Список литературы

- [1] Рудометов С. В., Окольников В.В. Использование онтологии технологических систем добычи угля для задания имитационных моделей на естественном языке. “Распределенные информационно — вычислительные ресурсы” (DICR-2014). 2–5 декабря 2014, Новосибирск.
- [2] Рудометов С. В. Визуально — интерактивная система имитационного моделирования технологических систем. Вестник СибГУТИ. 2011. No 3. С. 14–27.
- [3] Турчин В.Ф. Феномен науки: Кибернетический подход к эволюции. Изд. 2-е — М.: ЭТС. — 2000. — 368 с.
- [4] Bernard Zeigler (1987). “Hierarchical, modular discrete-event modelling in an object-oriented environment”. Simulation. 49 (5): pp 219–230.
- [5] Саймон Хайкин. Нейронные сети. Полный курс. 2-е изд. Пер. с англ. — М.: Издательский дом “Вильямс”, 2006. — 1104 с.: ил. — Парал. тит. англ.
- [6] Лапшин В. А. Онтологии в компьютерных системах. — М.: Научный мир, 2010.
- [7] Кеннет С. Рубин. Основы Scrum. Практическое руководство по гибкой разработке ПО. Пер. с англ. — М.: Издательский дом “Вильямс”, 2016.— 537 с.

*Виктор Васильевич Окольников — д.т.н., ведущий науч. сотр.  
Института вычислительных технологий СО РАН;  
e-mail: okoln@mail.ru;*

*Сергей Валерьевич Рудометов — к.т.н., н.с.  
Института вычислительных технологий СО РАН;  
e-mail: rsw@inbox.ru.*

*Дата поступления — 30 мая 2017 г.*