УДК 681.3.06

*И.Н. Парасюк, С.В. Ершов*

# ТРАНСФОРМАЦИОННЫЙ ПОДХОД К РАЗРАБОТКЕ ИНТЕЛЛЕКТУАЛЬНЫХ АГЕНТОВ НА ОСНОВЕ НЕЧЕТКИХ МОДЕЛЕЙ

Рассматривается трансформационный подход, применяемый на различных стадиях разработки интеллектуальных агентов, в основе которого находятся модели нечетких правил. Описаны трансформации, позволяющие сгенерировать нечеткие платформно-независимые модели внутриагентного управления и связанные с ними ролевые модели. Рассмотрены способы создания в среде Eclipse средств поддержки проблемно-ориентированного графового языка для спецификации нечетких моделей, не зависящих от используемой платформы. Значительное внимание уделяется архитектурным особенностям нечетких агентов, порождаемых в результате трансформации для перспективных платформ.

## Введение

Наблюдаемый в последнее время рост интереса к возможностям мультиагентных интеллектуальных систем в контексте разработки программного обеспечения ставит задачу создания методов и средств моделе-ориентированной трансформационной разработки (Model-Driven Engineering, MDE) таких систем. Хотя, многие известные методологии разработки мультиагентных систем, такие как Tropos [1] и INGENIAS [2] предлагают методы и инструменты для автоматизации трансформаций моделей, они предназначены для использования только на некоторых этапах разработки и не могут рассматриваться как систематические.

В настоящее время мультиагентные системы представляют собой один из наиболее сложных видов программного обеспечения, разработка которых затрудняется в связи с: 1) отсутствием единой метамодели, в рамках которой описывается функционирование агентов (например, рассматриваются делиберативные, реактивные и другие виды агентов); 2) существованием агентов и принятием решения агентами в условиях неопределенности, когда каждый агент обладает ограниченной информацией; 3) необходимостью рассматривать различные дополнительные аспекты описания, имеющие решающее значение для мультиагентных систем, такие как выполняемые роли, принципы и модели обмена сообщениями и координации); 4) отсутствием общепринятых “стандартных” платформ и языков реализации – несмотря на предпринимаемые попытки, такие как абстрактная архитектура FIPA [3]. Для решения вышеуказанных проблем требуются новые парадигмы концептуализации, новые модели формализации и новые архитектуры построения нечетких агентов, что, в свою очередь, требует создания новых технологий проектирования и программной реализации, опирающихся на сквозное использование моделей, представляющих различные уровни и детали представления мультиагентных систем, и на систематическое использование автоматизированных трансформаций, позволяющих сократить объем работы при переходе между моделями. Их решение возможно в рамках инженерии предметной области [4, 5], включающей в себя создание моделей, метамоделей и спецификации трансформаций для разработки нечетких агентов. За счет потраченных при этом усилий, возможна значительная экономия ресурсов и улучшение характеристик качества программного обеспечения мультиагентных систем [6] при проведении прикладной инженерии [4]. Моделе-ориентированная разработка включает в себя оба указанных вида деятельности.

В основе моделе-ориентированной разработки лежат понятия платформно-независимой и платформно-зависимой моделей (ПНМ, PIM, platform-independent и ПЗМ, PSM, platform-specific model). ПНМ определены на высшем уровне абстракции,

чем ПЗМ. Независимая от платформы модель – представление системы с независимой от платформы точки зрения [7], в то время как ПЗМ определена как вид системы с точки зрения определенной платформы. Использование этих понятий позволяет избавиться от технологических и технических деталей, которые несущественные для фундаментальной функциональности системы (или ее части).

Разделение платформно-зависимой и платформно-независимой моделей обеспечивает ряд преимуществ по сравнению с традиционным подходом. В частности, облегчается перенос программного обеспечения на другую платформу и его модификация, так как при этом можно использовать старую платформно-независимую модель и разрабатывать заново только платформно-зависимую.

Главное преимущество моделеориентированного подхода состоит в том, что с его помощью можно ускорить разработку мультиагентных систем, несмотря на то, что нужно создать несколько моделей вместо одной. Это достигается за счёт автоматизированной генерации платформно-зависимой модели по платформно-независимой с помощью специально разработанных трансформаций (выражаемых чаще всего как набор правил). Поэтому процесс перехода к платформно-зависимым моделям и программному коду мультиагентных систем, основанным на конкретной технологической платформе, может быть в значительной степени формализован с помощью таких трансформаций.

Цель настоящей работы – исследование и обоснование трансформационного подхода к разработке интеллектуальных агентов с использованием нечетких моделей, разработка трансформационной среды для спецификации моделей нечетких агентов, не зависящих от платформы реализации, и изучение архитектурных особенностей нечетких агентов, порождаемых для перспективных платформ. Перечисленные проблемы тематически вписываются в дальнейшее развитие исследований в направлении становления моделеориентированного подхода к разработке интеллектуальных программных агентов [8–12].

**Основные модели и трансформации**

Для описания процессов, в рамках которых разрабатываются нечеткие интеллектуальные агенты, нами использован Eclipse Process Framework (EPF) – инструментальная платформа управления процессами и концептуальный фреймворк для создания, адаптации, развертывания разработанных процессов [13]. EPF предоставляет готовый каталог процессов для типичных проектных ситуаций, которые могут быть адаптированы для индивидуальных целей. Два основные концепции EPF – это содержимое метода и процесс. Эти две концепции прекрасно иллюстрирует унифицированный процесс разработки программного обеспечения [14], включающий как методы, так и процессы – фазы и итерации. Содержимое метода описывает как он может быть выполнен, какие требуются навыки, пошаговое описание того, как добиться требуемых показателей. Процесс описывает сам жизненный цикл разработки и определяет последовательность осуществления выполнения работы ролями, и то как производятся и эволюционируют во времени рабочие продукты. В стандартной поставке EPF предлагаются для загрузки процессы для OpenUP, XP и Scrum. Хранение и структурирование контента определяется схемой, это развитие спецификации SPEM [15], ссылающейся на Unified Method Architecture (UMA). Такая организация контента не ограничивается традиционным производством ПО и пригодна для поддержки процессов моделеориентированной разработки сложных мультиагентных систем. Содержимое разработанных методов базируется на рабочих продуктах, видах деятельности, ролях и инструментальных средствах. Рабочие продукты производятся инструментальными средствами которые выполняют роли, и имеют рабочие продукты в качестве входов и выходов. В нашем процессе рабочие продукты представляют собой как графовые модели (например, модель внутриагентного управления), так и текстовые документы (табличное представление ро-

левой модели и код программы агента на языке программирования). Конкретные роли выполняют разработчики, вовлеченные в процесс создания мультиагентной системы. В качестве инструментальных средств рассматриваются проблемно-ориентированные редакторы моделей и наконец, сами трансформации между моделями. Такой процесс включает трансформации и виды деятельности, которые изображены на рис. 1 в виде диаграммы EPF.

Основные процессы разработки нечеткой мультиагентной системы с использованием моделе-ориентированного подхода согласуются с ядром знаний SWEBOK [16], которое является основополагающим документом, отображающем мнение многих зарубежных и отечественных специалистов в области программной инженерии и согласующееся с современными регламентированными процессами стандарта ISO/IEC 12207.

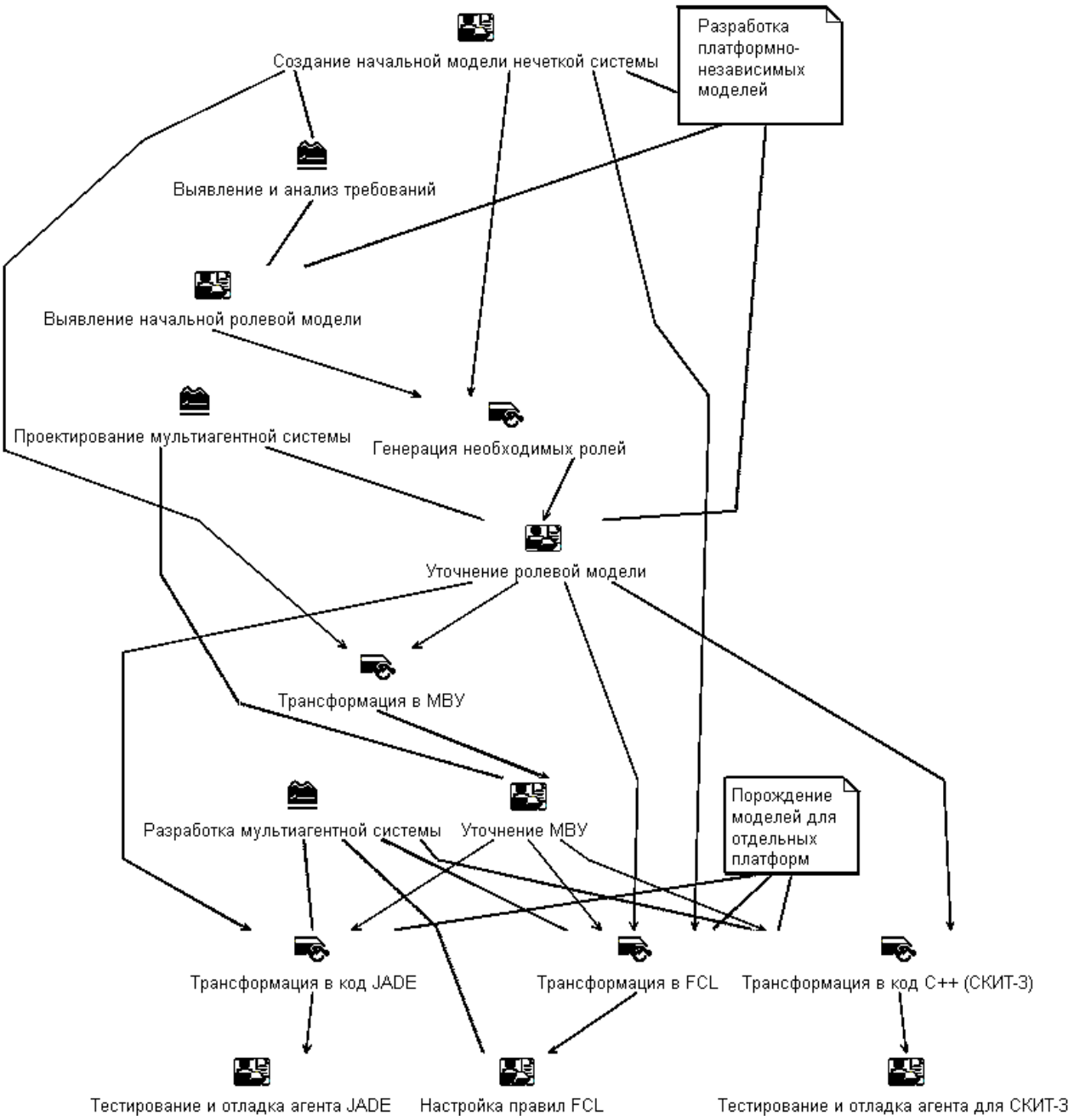



Рис. 1. Основные трансформации, виды деятельности и их взаимосвязи при разработке нечетких агентов

Основные рассматриваемые области-процессы – это разработка (выявление и анализ) требований, проектирование агентов и реализация агентов мультиагентной системы. Значение отдельных моделей интеллектуальных агентов не одинаково для различных процессов. При этом на этапах разработки требований и проектирования создаются основные платформно-независимые модели: начальная модель нечеткого агента, начальная ролевая модель, уточненная ролевая модель и модель внутриагентного управления (МВУ). Платформно-зависимые модели включают в себя модели нечеткого агента, соответствующие всем деталям текстового представления. Естественно, что такие модели являются объектом процессов конструирования и тестирования. В настоящее время такие модели разработаны для таких платформ как FCL [17], JADE [18] и для разработанной нами библиотеки классов программирования нечетких агентов в среде суперкомпьютера СКИТ-3 [19].

Основные применяемые трансформации служат как для перехода между процессами разработки, так и для преобразования между моделями, применяемыми в одном процессе (например, переход между уточненной ролевой моделью и моделью МВУ).

Моделе-ориентированный трансформационный подход включает разработку метамоделей, определяющих способы спецификации нечетких моделей. При этом, вначале выбирается единственная метамодель высшего уровня, которая не зависит от точки зрения и определяет способы построения всех остальных метамоделей (в том числе и самой себя).

Моделе-ориентированная разработка мультиагентных систем в значительной степени зависит от последовательной трансформации их моделей [20]. При этом наиболее важными являются способы преобразований моделей:

- преобразования модель-модель (M2M). Этот вид преобразования используется для преобразования одного типа графической модели (схемы, диаграммы) к другому типу графической модели. Преобразование M2M основано на исходной и целевой метамоделях и определяет преобразования элементов исходной модели в элементы целевой модели;
- преобразования модель-текст (M2T). Такие преобразования используются для трансформации визуального представления в код программы; синтаксис целевого языка определяется вместе с метамоделью графической модели.

## Разработка моделей нечетких правил и ролевых моделей агентов

Применяемые в процессах моделе-ориентированной разработки нечетких мультиагентных систем метамодели, модели и трансформации описаны на основе фреймворка Eclipse Modeling Framework [21], использующего для построения моделей метамодель высшего уровня Ecore. Она определяет, что модель состоит из экземпляров типа EClass, которые могут иметь атрибуты (экземпляры типа EAttribute) или ссылки на другие экземпляры EClass (через тип EReference). В результате, атрибуты могут быть различными экземплярами EDataType (например, целыми числами, строками, действительными числами, и т. д.). Средства описания метамоделей Ecore предназначены для поддержки времени выполнения, такой как уведомления об изменениях, средства для сохранения моделей со встроенными средствами сериализации на основе стандарта XMI (XML Metadata Interchange) [22], и эффективный интерфейс программиста для операций над объектами Ecore. Компонент Ecore Tools обеспечивает среду для операций над моделями, в том числе проверку правильности и создание генераторов.

Аналогичная технология для представления и обработки метамоделей – Meta-Object Facility (MOF) [23]. Тем не менее, метамодель EMF проще, чем метамодель MOF с точки зрения понятий, свойств и внутренней структуры, таким образом, отображение понятий EMF в понятия MOF может быть проведено относительно просто.

Исходной при трансформационной разработке агентов может служить их платформно-независимая модель нечетких правил. Для представления системы нечет-

ких правил разработана метамодель Ecore нечетких правил, содержащая все необходимые понятия для конструирования систем нечетких правил вывода, таких как системы Мамдани или системы Такаги–Сугено [24, 25]. В качестве основных классов (понятий) выбраны FuzzyRule (нечеткое правило), FuzzyVariable (нечеткая переменная), FuzzyValue (нечеткое значение), FuzzySet и LinguisticTerm (нечеткое множество и связанный с ним лингвистический терм). Каждое правило включает набор антецедентов и консеквентов, которым соответствует нечеткое значение, связанное в свою очередь с нечеткой переменной, описывающей набор допустимых лингвистических значений и их функций принадлежности. Различные способы выполнения нечетких правил учитываются с помощью конкретных классов FuzzyRuleExecutor, например, класс LarsenRuleExecutor соответствует системе Ларсена. Это позволяет расширять данную метамодель другими видами нечетких систем, описанными в литературе [24], не затрагивая при этом другие модели. Фрагмент метамодели, относящийся к представлению нечетких правил представлен на рис. 2.

Однако, задавать исходную модель правил в текстовом виде или базовыми средствами дерева объектов Ecore было бы не наглядно и затрудняло бы ее восприятие. Для ввода и редактирования таких моделей создан специальный графический редактор со средствами навигации по нечетким правилам и композиции правил замены (удаление и добавление нечетких переменных, множеств, функций принадлежности, включение переменных в антецеденты и консеквенты правил и т. д.) На рис. 3 приведен пример начальной модели нечетких правил агента, заданной на проблемно-ориентированном графовом языке, не зависящем от платформы реализации.

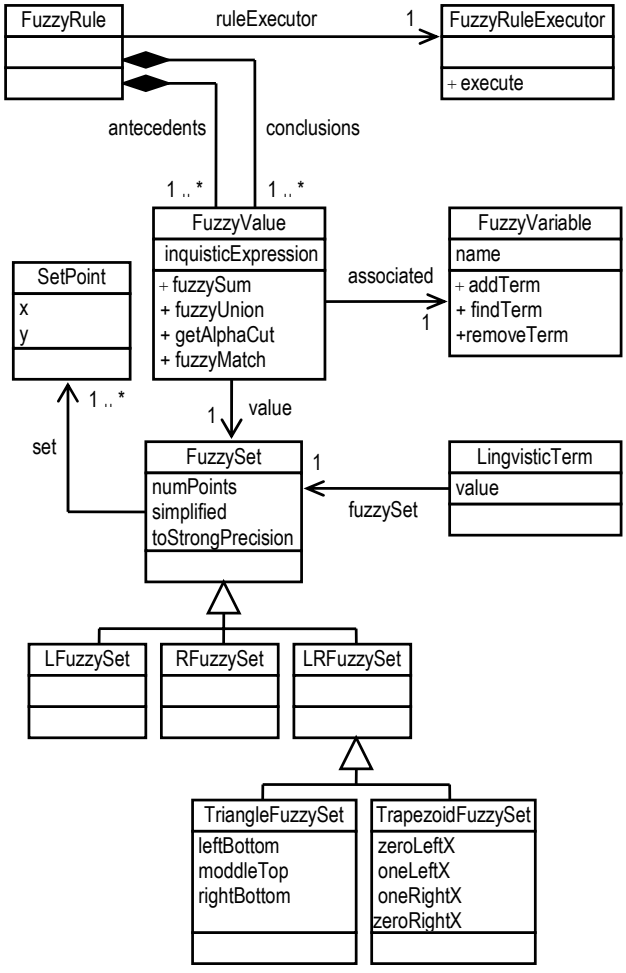



Рис. 2. Фрагмент метамодели для представления нечетких правил

Создание такого редактора производилось в рамках методологии предметно-ориентированного моделирования (domain-specific modeling), включающей систематическое использование визуальных (чаще всего основанных на графах) предметно-ориентированных языков моделирования для представления различных аспектов системы. Такие языки дают возможность специфицировать систему на более высоком уровне платформно-независимой абстракции, чем языки моделирования общего назначения, такие как UML. Процесс создания предметно-ориентированного языка состоит из трех шагов: определение абстрактного синтаксиса, определение конкретного синтаксиса и определение правил трансформации. Абстрактный синтаксис для моделирования нечетких агентов задается метамоделью в системе EMF (формат Ecore), фрагмент которой приведен выше (рис. 2). Для описания конкретного визуального синтаксиса диаграмм использовался фреймворк GMF (Graphical Modeling Framework) [26]. Построение такого синтаксиса начинается с определения модели графического определения, включающей галерею графических элементов (фигуры, надписи, линии и т. д.) в виде дерева объектов в формате Ecore. Модель инструментов включает в себя описание элементов для создания палитры, панели инструментов, а также различных меню, используемых при редактировании диаграмм. Наиболее важной из всех моделей GMF является модель отображений, связывающая элементы графического определения и инструменты с вершинами и ребрами графического редактора, и порождающая в результате модель генератора. На основе такой модели порождается полный код программы-редактора диаграмм, работающий в среде Eclipse. Аналогично, существует также возможность задать конкретный текстовый синтаксис, например, с использованием фреймворка Textual Modeling Framework [27].

Определение правил трансформаций, отображающих модели, создаваемые с помощью сгенерированного редактора, осуществлялось с использованием языка Query/View/Transformation (QVT), разработанного под руководством OMG. QVT определяет стандартный способ преобразования исходных моделей в целевые модели на основании сопоставления элементов исходной и целевой метамоделей. Хотя стандартное определение QVT предназначено для отображений над элементами MOF- совместимых метамоделей, реализация, применяемая в Eclipse, основана на метамоделях Ecore. QVT определяет не

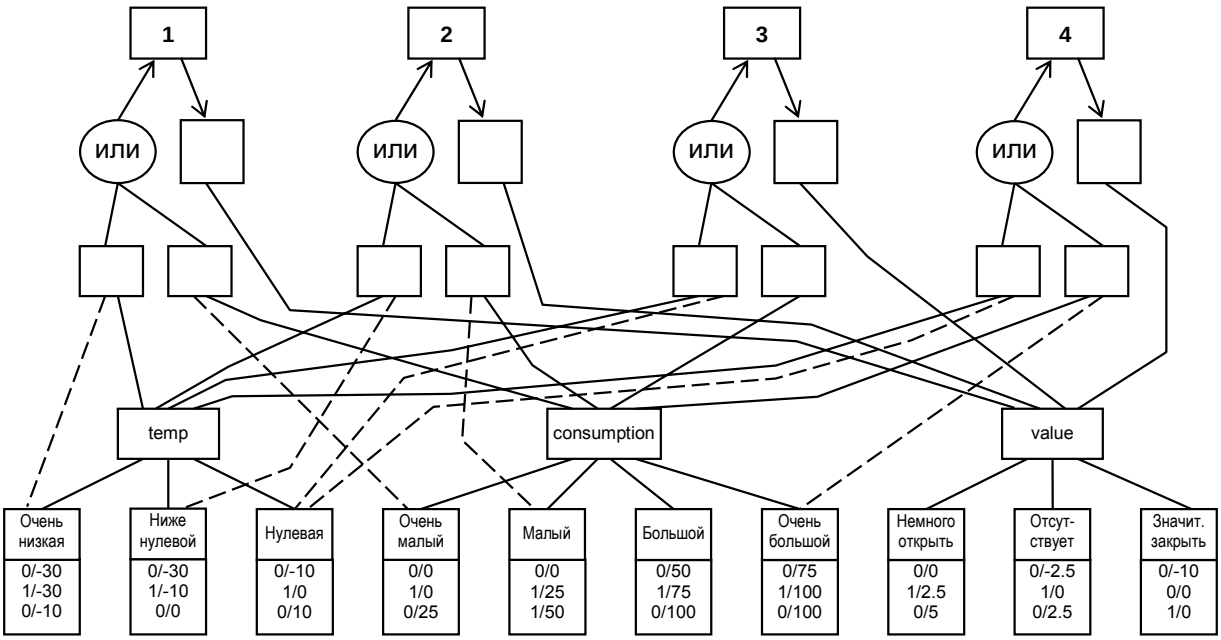



Рис. 3. Начальная модель нечеткого агента, заданная на проблемно-ориентированном графовом языке, не зависящем от платформы реализации

один, а три предметно-ориентированных языка: реляционный (декларативный) Relations, ядро Core и язык операционных отображений OperationalMappings. Relations и Core – декларативные языки на двух разных уровнях абстракции, с нормативными отображениями между ними. QVT/OperationalMappings – императивный язык, охватывающий средства QVT/Core и позволяющий задавать такие конструкции как циклы, условия и т. д. Хотя известно множество других языков, родственных QVT по лежащим в их основе понятиям (например, ATL и Tefkat), в настоящее время именно QVT/OperationalMappings обеспечивает как наибольшую практическую проработанность, так и наибольшую степень совместимости с концепциями, предлагаемыми OMG в рамках моделеориентированной архитектуры.

В процессе разработки требований также определяется их начальная ролевая модель (РМ). Предполагается что интеллектуальный агент обязывается выполнить одну или несколько ролей в течении своего жизненного цикла. Одной из особенностью разработки агентов является то, что виды деятельности агента связаны с ролью, а не с системой. Кроме того, после определения возможностей агентов и разложения их на простые деятельности, необходимо определить динамическую композицию из этих видов деятельности по каждой роли таким образом, что агент достигает поставленной цели. Метамодель ролевой модели определяет понятие *Роль*, которое ссылается на понятия:

• деятельность, которая включает два атрибута, имя (название) и функциональность (описание того, что эта деятельность делает);

• возможность, относящаяся к направлениям деятельности на которые она ссылается, достигающим высокоуровневой цели;

• протоколы и виды деятельности, которым приписываются соответствующие названия.

В процессе проектирования генерируются дополнительные роли агента, уточняющие начальную ролевую модель. В общей ситуации, в соответствии с шаблоном проектирования поставщик/потребитель [28] для каждой нечеткой переменной, входящей в начальную модель правил, порождаются две дополнительные роли.

Определенные ранее роли связываются с выбранными типами агентов, составляющими проектируемую мультиагентную систему и порождающими отдельные (типизированные) экземпляры агентов. При необходимости размещения агентов на одной вершине для экономии коммуникационных ресурсов их типы могут быть совмещены, что приводит к суммированию (наложению) ролей, которые они должны выполнять.

**Модели внутриагентного управления**

В процессе проектирования к уточненной ролевой модели применяются трансформации, позволяющие получить спецификации моделей внутриагентного управления – по одной для каждой сгенерированной роли. Для разработки таких трансформаций модель-модель (M2M) использован язык QVT/OperationalMappings.

Многие методологии разработки (как Tropos или INGENIAS [1, 2]) навязывают специфические ментальные модели агента. В отличие от этого, модель внутриагентного управления (МВУ) основана на нечетких схемах состояний и не делает никаких дополнительных предположений о таких аспектах как модель коммуникации, процесс вывода или ментальные состояния (убеждения-желания-намерения) агентов.

Отдельным агентам таких распределенных систем открыт доступ только к их собственным состояниям, а не к глобальному состоянию всей системы, о котором агент может делать только предположения. К тому же, любая среда передачи асинхронных сообщений между агентами не является абсолютно надежной и приводит к запаздыванию сообщений, что еще больше увеличивает степень неопределенности схем состояний агента. Поэтому работой таких агентов можно управлять только с учетом степени достижимости таких “размытых” состояний, для чего обоснованным представляется использование нечеткой математики.

Метамодель МВУ, показанная на рис. 4, определяет понятие *Модель*, которая включает нечеткие состояния, переходы и значения. Имена модели должны удовлетворять современному формату пространства имен пакетов Java или C#. Нечеткие состояния содержат следующие атрибуты: имя узла, тип узла, соответствующий типу состояния в схеме состояний, как правило, один из И, ИЛИ, НАЧАЛО, КОНЕЦ, метка узла, и деятельность, связанная с узлом.

Состояния и переходы также ссылаются на нечеткие значения, которые в свою очередь включают атрибут "значение" – в диапазоне [0,1]. Следующим понятием определенным в этой метамодели является "Переход", который, кроме ссылки на нечеткое значение, включает также следующие атрибуты:

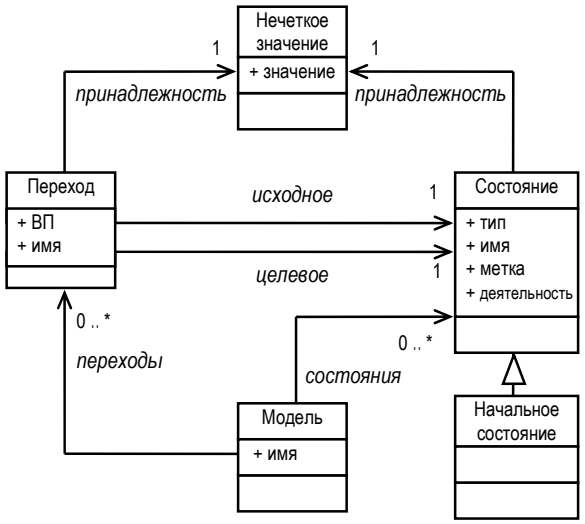



Рис. 4. Метамодель внутриагентного управления

• имя, обычно в форме <метка исходного узла> <метка целевого узла>;

• выражение перехода. Через выражения перехода (ВП) определяется основная управляющая информацию в МВУ. Это выражение содержит условия и события, которые делают переход возможным. Кроме того, может быть использовано получение или передача сообщений между агентами (в случае взаимодействующих агентов);

• источник, исходное состояние, и цель, целевое состояние.

Таким образом, каждая модель МВУ уточняется, определением условий и события приема/передачи сообщений, которые позволят переходы от одной деятельности (задачи) к другой.

МВУ задается нечеткой схемой переходов, представленной нечетким мультиграфом $G = (Q, T, \mu_G, \Sigma)$, где $Q = \{q_1, q_2, ..., q_n\}$ – множество состояний (вершин графа), где $q_1$ – начальное состояние; $T = \{t_1, t_2, ..., t_n\}$ – множество переходов (дуг) графа; $\mu_G : T \to [0,1]$ – функция принадлежности переходов данной схеме; $\Sigma : T \to A$ – функция разметки переходов событиями из некоторого конечного алфавита событий (сообщений) А.

Способ представления принадлежностей переходов фактически определяет структура нечеткой схемы переходов. Пусть такая схема включает *n* состояний, степень принадлежности ее переходов может быть представлена в виде трехмерной матрицы

$$P = \begin{bmatrix} p_{11}, p_{12}, ..., p_{1n} \\ p_{21}, p_{22}, ..., p_{2n} \\ ..... \\ p_{n1}, p_{n2}, ..., p_{nn} \end{bmatrix}, \qquad (1)$$

где $p_{ij} = [\mu_{ij}^1, \mu_{ij}^2, ..., \mu_{ij}^m]$ – вектор степени принадлежности переходов между вершинами *i* и *j*, размеченных соответственно символами $a^k$, причем $\mu_{ij}^k \in [0,1]$, символы в разметке не повторяются. Ненулевые значения $\mu_{ij}$ соответствует переходам состояний $q_j = \delta(q_i, a_{ij}^k, \mu_{ij}^k)$, при этом $\mu_{ij} = 0$ означает, что между двумя состояниями перехода нет. Правильная схема переходов должна удовлетворять следующему условию: в одном и том же векторе символы событий отличаются друг от друга ($a^k \neq a^l$ для всех $k \neq l$) и принадлежат конечному алфавиту событий А.

Определим нечеткое (размытое) состояние как вектор пар

$$Q = [< q_1, \mu(q_1) >, < q_2, \mu(q_2) >, ..., < q_n, \mu(q_n) >], \qquad (2)$$

второй компонент которых $\mu(q_i)$ – степень принадлежности агента данному состоянию. Реакцией (выполнением) схемы переходов в ответ на наступление события $a^k$ является новое состояние

$$Q_{Q\otimes P} = [< q_1, \mu_{Q\otimes P}(q_1) >, < q_2, \mu_{Q\otimes P}(q_2) >, ..., < q_n, \mu_{Q\otimes P}(q_n) >]$$

такое, что

$$\mu(q_i) = \max\{\mu(q_j) * \mu_{ji}^k\} \ (\forall < q_j, \mu(q_j) > \in Q).$$

Последовательное выполнение (поведение), задаваемое парой событий $a^k, a^l$ и схемой нечетких переходов соответствует max-* композиции нечетких отношений $P \otimes P$:

$$\mu_{ij} = \max\{\mu_{ir}^k * \mu_{rj}^l\},$$
$$(\forall \mu_{ir}^k \in p_{ir}, \mu_{rj}^l \in p_{rj}, r = 1,...n).$$

Деятельности, задаваемые схемой с *n* состояниями непосредственно представлены вектором:

$$D = [d_1, d_2, \ldots, dn],$$

где $d_j$ – деятельность, выполняемая в *j*-м состоянии. Как правило, выполняется только деятельность, степень принадлежности состояния которой – наибольшая среди всех состояний.

Таким образом, любая нечеткая схема переходов с *n* состояниями может быть представлена матрицей принадлежности *n***n* (1) и вектором состояния (2). Одной из интересных возможностей настройки нечеткой схемы переходов является возможность ее обучения на основе представленных примеров и специально разработанной эволюционной стратегии [11].

## Пример разработки агентов на основе нечетких моделей

Рассмотрим процесс моделе-ориентированной разработки нечетких агентов для задачи управления подачей газа конечным потребителем. Управление подачей газа осуществляется путем частичного или полного открытия/закрытия вентиля (заслонки), от положения которого зависит давление в трубе. Задача – не допустить значительного отклонения давления. Поскольку потребление ресурса может существенно колебаться, принятие решения о положении вентиля не может быть осуществлено только на основании объема потребления. Следует учитывать также другие факторы – например, температуру воздуха, время суток. Кроме того, показания датчиков для снижения их стоимости не могут быть абсолютно точными и вносят дополнительную нечеткость.

Предположим, решение о положении вентиля принимается агентом на основании двух нечетких лингвистических переменных (температуры воздуха и текущего объема потребляемого ресурса). В таблице 1 показаны допустимые значения лингвистических переменных и соответствующих им термов, на основе которых строятся лингвистические правила, на рис. 5 – функции принадлежности нечет-

Таблица 1. Нечеткие лингвистические переменные для агента-потребителя ресурсов

| Лингвистическая переменная | Тип | Лингвистические термы |
|---|---|---|
| Температура | Входная | Очень низкая (ОН), ниже нулевой (НН), нулевая (Н), выше нулевой (ВН), очень высокая (ОВ) |
| Текущий объем потребляемого ресурса | Входная | Очень малый (ОМ), малый (М), средний (СР), большой (Б), очень большой (ОБ) |
| Изменение степени открытия вентиля | Выходная | Значительно закрыть (ЗЗ), немного закрыть (НЗ), отсутствует (О), немного открыть (НО), значительно открыть (ЗО) |

ких множеств, описывающих лингвистические термы; для простоты используются треугольные функции принадлежности. В таблице 2 представлен набор нечетких правил Мамдани, осуществляющих управление.

Для указанной системы правил разработана начальная модель нечетких правил агента, заданная на проблемно-ориентированном графовом языке, не зависящем от платформы реализации (рис. 3).

В процессе проектирования генерируются дополнительные роли агента. В общей ситуации, в соответствии с шаблоном проектирования поставщик/потребитель [28] для каждой нечеткой переменной порождаются две роли. Кроме того, поставщики входных переменных должны осуществлять их фузификацию, а потребители выходных переменных – дефузификацию, если система нечетких правил всего одна. Для вышеприведенной платформонезависимой модели в результате применения трансформации автоматически сгенерированы шесть ролей: *ПоставщикНечеткогоЗначенияТемпература* и *ПоставщикНечеткогоЗначенияПотребление*, – измерение текущей температуры и уровня потребления соответственно и их фузификация; *ПотребительНечеткогоЗначенияТемпература* и *ПотребительНечеткогоЗначенияПотребление* – сбор нечетких значений, необходимых для осуществления нечеткого вывода изменения степени открытия, *ПоставшикНечеткогоЗначения-*

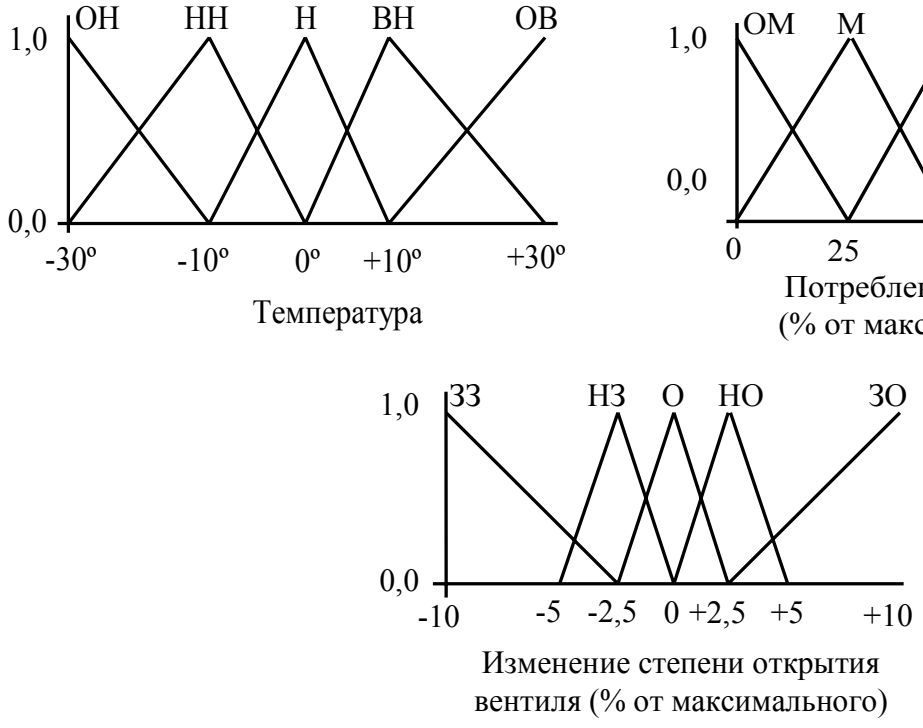



Рис. 5. Функции принадлежности для нечеткого агента

Таблица 2. База правил для агента

| Потребление ресурса (% от максимального) | Температура | | | | |
|---|---|---|---|---|---|
| | ОН | НН | Н | ВН | ОВ |
| ОМ | | | ЗЗ | | |
| М | | О | НЗ | | |
| СР | ЗО | НО | О | НЗ | ЗО |
| Б | | | НО | О | |
| ОБ | | | ЗО | | |

*ИзмененияОткрытия*, непосредственно осуществляющая нечеткий вывод и вырабатывающая необходимое нечеткое значение, *ПотребительНечеткогоЗначенияИзмененияОткрытия* – дефузификация полученного значения и его применения для закрытия/открытия вентиля. Описание уточненной ролевой модели, полученной на этапе анализа, частично приведено в табл. 3. Параметры-имена агентов-потребителей и поставщиков обозначены как FVC (fuzzy value consumer) и FVS (fuzzy value supplier) соответственно. Следует заметить, что в такой модели, все взаимодействия с координатором каталога (КК) – простые действия, так как модели взаимодействия с КК, поддерживаемые основными целевыми платформами включают соответствующие (статические) методы.

Координатор каталога – специальный агент, обеспечивающий регистрацию, и поиск предоставляемых прикладных сервисов. В качестве сервисов для порождаемых ролей рассматриваются уведомления об изменении нечеткого значения (нечеткого числа). Для регистрации используется действие *RegisterDF*, выполняемое агентом-поставщиком, а для поиска сервиса и "подписки" на уведомления – действие *SearchDF*, выполняемое агентом-потребителем. В случае успешного поиска сервиса, агент-потребитель будет получать асинхронные сообщения со значением всякий раз, когда агент-поставщик заменяет значение нечеткой величины (получаемое либо в процессе нечеткого вывода, либо вводимое пользователем, либо снимаемое с датчиков). Подобная архитектура позволяет создавать иерархические многоуровневые сети, где нечеткий вывод на каждом уровне рассматривается как совокупность ролей поставщик-потребитель.

Применим к уточненной ролевой модели трансформации, позволяющие получить спецификации моделей внутриагентного управления для каждой выбранной роли. На рис. 6 показаны полученные МВУ для ролей *ПоставщикНечеткогоЗначенияИзмененияОткрытия* (а) и *ПотребительНечеткогоЗначенияИзмененияОткрытия* (б).

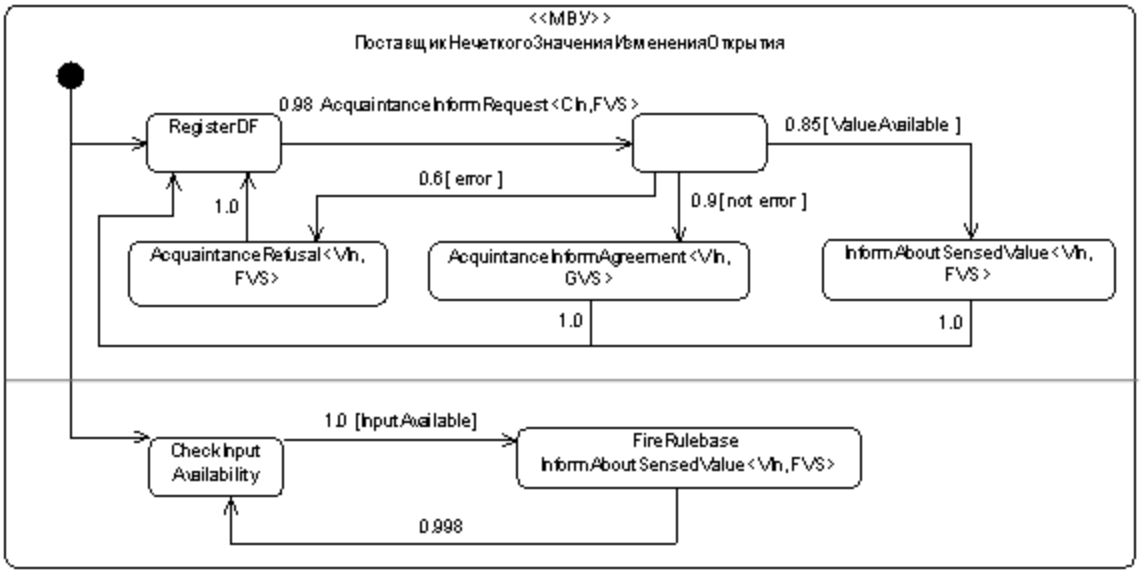
Рассмотрим, например, ситуацию, когда агент, выполняющий роль *ПотребительНечеткогоЗначенияИзмененияОткрытия* находится в начальном (четком) состоянии. Выполнение МВУ на рис. 6 в ситуации когда поступает сообщение *InformAboutSensedValue* приводит к размытому состоянию в котором принадлежность состояния обозначенного *Apply-Output* составляет 0.6, принадлежность всех остальных состояний равняется нулю. Последовательные сообщения *Acquitance-InformAgreement* (запоздавшее сообщение об установлении "подписки" на поставляемое значение, пришедшее в неверном порядке) и *InformAboutSensedValue* порождают размытое состояние, при котором наибольшую степень принадлежности имеет состояние *Reset* (0.39), что свидетельствует в пользу возможной ошибки и приводит к необходимости переустановить взаимодействие с агентом-поставщиком. Учет подобной неопределенности в случае четкой модели МВУ либо невозможен в принципе, либо приводит к значительному увеличению количества состояний, отражающих степень неопределенности при реакции на приходящие сообщения.

В рассматриваемом примере соответствующая модель включает четыре типа агентов (рис. 7): *ПоставщикТемпературы*, *ПоставщикПотребления*, выполняющие похожие роли *ПоставщикНечеткогоЗначения*; *ПоставшикИзменениеОткрытия*, выполняющего роли *ПоставщикНечеткогоЗначенияИзмененияОткрытия*, *ПотребительНечеткогоЗначенияТемпература* и *ПотребительНечеткогоЗначенияПотребление*, и *РегуляторВентиля*, выполяющий одну важную роль – *ПотребительНечеткогоЗначенияИзмененияОткрытия*.

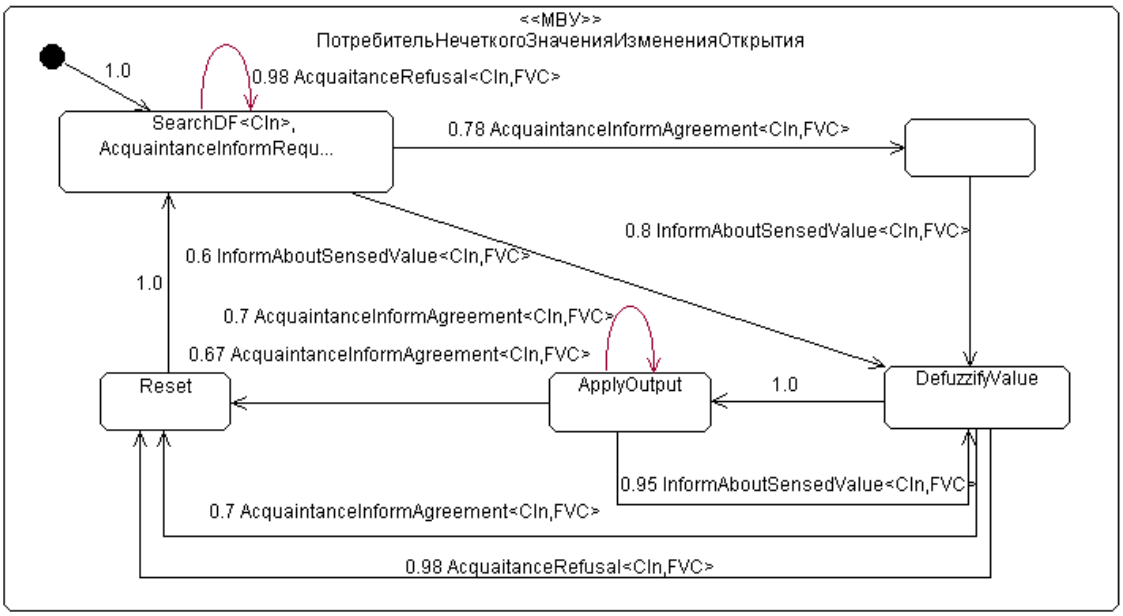
Взаимодействие между агентами различных типов приведено в табл. 4. При этом *Взаимодействует* означает что тип агента, указанный в строке посылает сообщения агенту, тип которого обозначен в столбце таблицы. *ПодписанНа* обозначает, что тип агента в строке ожидает уведомлений об изменившихся нечетких значениях типа агента в столбце, поставщиком которых он является.

Таблица 3. Описание отдельных ролей нечеткого агента

| | |
|---|---|
| Схема роли | ПоставщикНечеткогоЗначенияТемпература (ПоставщикНечеткогоЗначения-Потребление) |
| Описание: | восприятие входного четкого значения (температуры, уровня потребления), а также извещение об этом всех подписавшихся агентов. Успешная регистрация в КК будет проведена в течение всего времени жизни агента |
| Протоколы и действия: | RegisterDF, ReadCrispValue, FuzzifyValue, SensorValueAcquitanceRequest<T, FVS>, RegisterAcquitanceRequest, SensorValueNotification<T, FVS> |
| Доступ: | читает измеренное sensedCrispValue<T> // температура (уровень потребления);<br>поставленное consumerAID // идентификатор подписавшегося агента;<br>поставленное conceptTName<br>Порождает фуззифицированное sensedFuzzyValue<T> // нечеткое значение температуры (уровня потребления) |
| Схема роли | ПоставшикНечеткогоЗначенияИзмененияОткрытия |
| Описание: | восприятие значения CIn, необходимого, чтобы установить положение вентиля, а также передача извещения об этом всем подписавшимся агентам. Успешная регистрация в КК будет проведена в течение всего времени жизни агента. Система нечетких правил срабатывает только при получении всех входных значений. Система правил срабатывает снова, если изменяется хотя бы одно входное значение |
| Протоколы и действия: | RegisterDF, CheckInputAvailability, FireRuleBase, SensorValueAcquitanceRequest<CIn, FVS>, RegisterAcquitanceRequest, SensorValueNotification<CIn, FVS> |
| Доступ: | читает поставленное consumerAID // идентификатор подписавшегося агента поставленное conceptTName. Порождает нечеткое sensedFuzzyValue<T> – нечеткое значение изменения степени открытия, которое необходимо обеспечить |
| Схема роли | ПотребительНечеткогоЗначенияИзмененияОткрытия |
| Описание: | обрабатывает нечеткое значение CIn с целью его дефузификации для обеспечения необходимого положения вентиля, также установление взаимодействия с соответствующими агентами-поставщиками. Необходимо, чтобы был осуществлен успешный поиск понятия CIn в онтологии КК. Обрабатывает значение sensedFuzzyValue<CIn> когда оно – не пустое |
| Протоколы и действия: | SearchDF<CIn>, DefuzyfyValue, ApplyOutput, SensorValueAcquitanceRequest<CIn, FVS>, SensorValueNotification<CIn, FVS> |
| Доступ: | читает поставленное sensedFuzzyValue<CIn> // нечеткое выходное значение. Порождает и применяет дефузифицированное sensedCrispValue<CIn> // выходное значение, которое необходимо обеспечить |




а




б

Рис. 6. Модели внутриагентного управления для ролей: а – ПоставщикНечеткогоЗначения-ИзмененияОткрытия; б – ПотребительНечеткогоЗначенияИзмененияОткрытия

### Генерация кода мультиагентной системы

Порождение кода мультиагентной системы состоит из двух трансформаций – выполнения ряда преобразований модель-модель с целью получения платформно-специфической модели нечетких агентов и запуска последующих трансформаций “модель-код” для получения текста программы целевого агента.

Для тестирования возможностей нечеткой системы одного агента, подходящей целевой платформой также является MATLAB/Simulink [25]. Такая среда позволяет моделировать всю систему отдельного агента с помощью композиции функциональных блоков системы нечеткого логического вывода. Однако, MATLAB/Simulink – не предназначена для разработки мультиагентных систем и не обладает средствами для разработки таких важных аспектов как распределенность, коммуникация и др.

Поскольку нашей задачей является

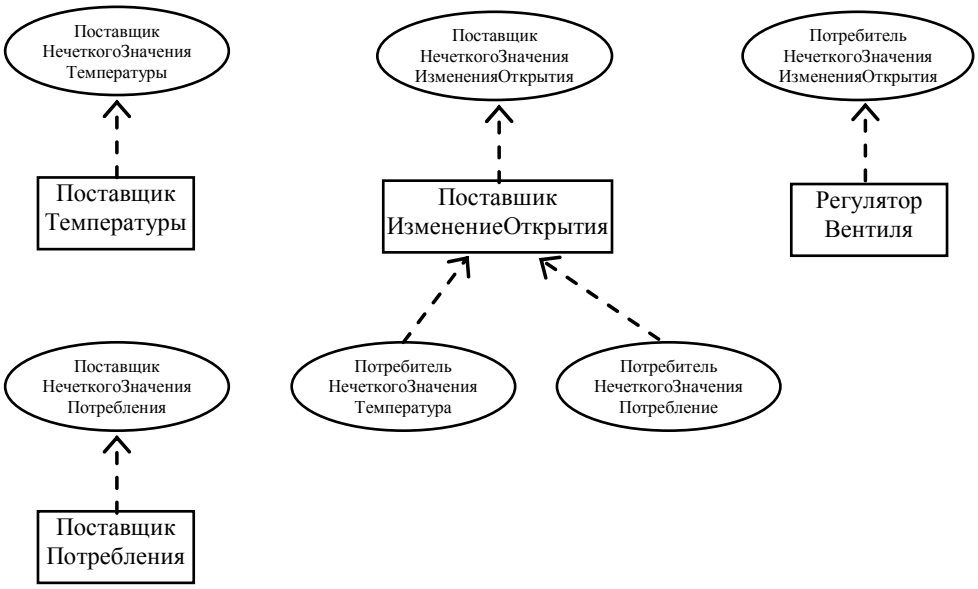



Рис. 7. Модель типов агентов

Таблица 4. Взаимодействие между типами нечетких агентов

| Типы нечетких агентов | Поставщик-Температуры | Поставщик Потребления | Поставщик-Изменение-Открытия | Регулятор Вентиля |
|---|---|---|---|---|
| ПоставщикТемпературы | ----- | ----- | Взаимодействует | ----- |
| ПоставщикПотребления | ----- | ----- | Взаимодействует | ----- |
| ПоставщикИзменениеОткрытия | Взаимодействует ПодписанНа | Взаимодействует ПодписанНа | ----- | Взаимодействует |
| РегуляторВентиля | ----- | ----- | Взаимодействует ПодписанНа | ----- |

трансформация нечетких агентов в целевую платформу, поддерживающую нечеткий логический вывод, приведем непосредственно реализацию системы правил, полученных на языке FCL (Fuzzy Control Language, Язык нечеткого управления) в результате трансформации платформонезависимой модели (рис. 8).

FCL – единственно стандартизированный предметно-ориентированной язык для спецификации систем нечеткого логического вывода [17]. Хотя FCL не позволяет описывать императивные операторы, ограничиваясь (как и MATLAB) системами логического вывода Мамдани и Такаги–Сугено [24, 25], его основное преимущество – наглядность и простота, а также наличие переносимых реализаций (в виде интерпретаторов) на языках Java и C++. FCL обеспечивает несколько методов фузификации и дефузификации переменных и спецификацию лингвистических термов. Однако, его нельзя рассматривать как единственно возможную целевую платформу по следующим причинам: 1) унификация и упрощение языка приводит к ограничению способов логического вывода и отсутствию набора базовых функций принадлежности, которые задаются не параметрически, а путем кусочно-линейной аппроксимации; 2) язык предназначен только для написания нечетких контроллеров (систем управления) и не позволяет описывать иерархические системы характерные для сложных интеллектуальных систем.

Поскольку FCL не обеспечивает всей функциональности, необходимой для функционирования агентов, порождается код на языке C++ (или на языке Java) для требуемой целевой платформы, который содержит вызовы интерпретатора FCL. В настоящее время известно много платформ для поддержки функционирования мульти-

```
FUNCTION_BLOCK УправлениеПодачейРесурса
VAR_INPUT
        temp : REAL;
        consumption : REAL;
END_VAR
VAR_OUTPUT
        valve : REAL;
END_VAR

FUZZIFY temp
        TERM ОН := (-30, 1) (-10, 0);
        TERM НН := (-30, 0) (-10, 1) (0, 0); … // другие термы
END_FUZZIFY
FUZZIFY consumption
        TERM ОБ := (75, 0) (100, 1);
        TERM Б:= (50, 0) (75, 1) (100, 0); … // другие термы
END_FUZZIFY
DEFUZZIFY valve
        TERM ЗЗ := (-10, 1) (-2.5, 0);
        TERM НЗ := (-5, 0) (-2.5, 1) (0, 0); … // другие термы
        ACCU : MAX; METHOD : COGS; DEFAULT := 0;
END_DEFUZZIFY

RULEBLOCK No1
AND : MIN;
RULE 1 : IF temp IS ЗЗ AND consumption IS ОМ THEN valve IS ЗЗ;
RULE 2 : IF temp IS НН AND consumption IS М THEN valve IS О;
RULE 3 : IF temp IS Н AND consumption IS Б THEN valve IS НО;
RULE 4 : IF temp IS Н AND consumption IS ОБ THEN valve IS ЗО;
. . . … // другие правила
END_RULEBLOCK
END_FUNCTION_BLOCK
```

Рис. 8. Реализация системы правил системы нечеткого вывода для агента, управляющего подачей ресурса

агентных систем – таких как Jason, 3APL, JACK и других [29, 30]. Следует заметить, что большинство этих платформ поддерживает только архитектуру Убеждение-Желание-Намерение (Belief-Desire-Intention, BDI), которая воплощает один из основных видов делиберативных (“разумных” агентов) и содержит явно представленную структуру данных, соответствующую этим трем указанным свойствам рассуждений, применяемых агентом при решении задач [8, 29, 30].

Привлекательность этого BDI-подхода снижается, поскольку был предложен ряд других моделей, позволяющих устранить проблемы BDI-агентов, обзор которых приведен в [8]. Поэтому, в качестве целевых платформ выполнения сгенерированных мультиагентных систем используется кластерная система СКИТ-3 [19] и платформа JADE [18]. Последняя представляет собой фреймворк для программирования распределенных открытых мультиагентных систем, взаимодействующих в соответствии со стандартами, разработанными FIPA (Foundation for Intelligent Physical Agents, Организацией интеллектуальных физических агентов) [3].

Поскольку, система СКИТ не содержит специальной платформы для поддержки взаимодействия агентов, была создана специальная библиотека на языке C++, включающая ряд шаблонов, классов и методов для организации поведения агентов – циклического, одноразового и многократного, а также для обмена различными типами асинхронних сообщений с использованием MPI в качестве носителя. Трансформация МВУ в код агента для целевой платформы – автоматизированное преобразование “модель-текст”, создающее класс агента и несколько классов *Behaviour* (Поведение) для каждой модели МВУ. Особенностью функционирования

порождаемых агентов для СКИТ является то, что в отличие от JADE не используется язык ACL (агенты на кластере не взаимодействуют с другими открытыми платформами), вместо этого генерируется код, выполняющий маршалинг/демаршалинг основных типов сообщений. Также, связывание сервисов с агентами-поставщиками выполняется во время компиляции, а не динамически, что позволяет сократить накладные расходы на обмен сообщениями.

Для генерации текстовых файлов программ-агентов разработаны преобразования "модель-текст" на основе текстовых шаблонов платформы XPand, запускаемых в контексте компонентов EMFT Workflow. Преимущество Xpand то, что он является независимым от исходной модели. Это означает, что любая из программ-парсеров может быть использована для общих моделей программного обеспечения, таких как MOF или EMF. Такая трансформация генерирует методы поведения агента для получения и отправки сообщений и композитные поведения нечеткой схемы переходов, которые координируют выполнение простых поведений.

Однако, средства представленной выше метамодели находятся достаточно далеко от средств программирования мультиагентной целевой платформы, такой как JADE. Поэтому для интеграции нечетких правил в среду выполнения указанных платформ используются преобразования, отображающее нечеткие правила в модель на основе JEM (Java EMF Model), позволяющей представить основные конструкции классов языка Java и платформы J2EE. Аналогичная промежуточная модель разработана и для C++. По сравнению с непосредственной генерацией кода на основе исходных моделей, использование промежуточных моделей значительно упрощает создание текстовых шаблонов.

## Выводы

Таким образом, нами изложен трансформационный подход, применяемый на различных стадиях разработки интеллектуальных агентов, в основе которого находятся модели нечетких правил. Следуя моделе-ориентированному подходу, разработаны основные преобразования, позволяющие сгенерировать нечеткие платформно-независимые модели внутриагентного управления и связанные с ними ролевые модели для распределенных интеллектуальных систем.

Для спецификации нечетких моделей, не зависящих от рассматриваемой платформы реализации, создан проблемно-ориентированный графовый язык и средства его инструментальной поддержки на основе среды Eclipse. Хотя нечеткие агенты, порождаемые в результате трансформации для таких перспективных платформ, как FCL, JADE и среды суперкомпьютера СКИТ-3, имеют ряд отличительных особенностей, трансформационный подход позволяет как учесть все эти вариативные особенности реализации, так и обеспечить перенос разработанных моделей мультиагентных систем на вновь разрабатываемые агентные платформы и даже среды, не имевшие таких платформ (как в случае системы СКИТ).

Важно, что предложенный трансформационный подход охватывает основные фазы разработки мультиагентных систем (от выявления требований до реализации), переход одной фазы в другую осуществляется с помощью преобразований нечетких моделей. В результате, модели каждой из фаз обогащают информацией, постепенно приводящей к реализации адаптивных интеллектуальных агентов.

1. *Perini A., Susi A.* Automating model transformations in agent-oriented modeling // Agent-oriented software engineering VI. Lecture notes in computer science. – Springer, 2006. – P. 167–178.
2. *Pavon J., Gomez-Sanz J.* Agent-oriented software engineering with INGENIAS // Multi-agent system and applications III. Lecture notes in computer science. – Springer, 2003. – P. 8–38.
3. *FIPA* abstract architecture. – http://www.fipa.org/specs/fipa00001/
4. *Чарнецки К., Айзенкер У.* Порождающее программирование: методы, инструменты, применение. – СПб.: Питер, 2005. – 731 с.
5. *Лаврищева Е.М., Петрухин В.А.* Методы и средства инженерии программного обес-

печения. – М.: Московский физико-технический институт, 2006. – 304 с.
6. *Основы* инженерии качества программных систем / Ф.И. Андон, Г.И. Коваль, Т.М. Коротун, Е.М. Лаврищева, В.Ю. Суслов // 2-е изд. – Киев: Академпериодика, 2007. – 672 с.
7. *Kleppe A., Warmer J., Bast W.* MDA Explained. The model driven architecture: practice and promise. – New York: Addison-Wesley Professional, 2003. – 192 p.
8. *Парасюк И.Н.*, *Ершов С.В*. Нечеткие модели мультиагентных систем в распределенной среде // Проблеми програмування. – 2010. – № 2–3. – С. 330–339.
9. *Парасюк И.Н., Ершов С.В*. Моделеориентированная архитектура нечетких мультиагентных систем // Компьютерная математика. – 2010. – № 2. – С. 62–74.
10. *Ершов С.В*. Принципы построения нечетких мультиагентных систем в распределенной среде // Там же. – 2009. – № 2. – С. 54–61.
11. *Ершов С.В*. Трансформационный подход к разработке адаптивных интеллектуальных агентов на основе нечетких схем переходов // Компьютерная математика. – 2011. – № 1. – С. 69–78.
12. *Парасюк И.Н., Ершов С.В*. Теоретико-категорная формализация при проектировании нечетких интеллектуальных систем // Кибернетика и системный анализ. – 2009. – № 6. – С.149–154.
13. *Eclipse* Process Framework Project (EPF). – http://www.eclipse.org/epf/
14. *Якобсон А., Буч Г., Рамбо Дж.* Унифицированный процесс разработки программного обеспечения. – СПб.: Питер, 2002. – 496 с.
15. *Software* and Systems Process Engineering Metamodel specification (SPEM) Version 2.0. - http://www.omg.org/spec/SPEM/2.0/
16. *SWEBOK*. – http://www.swebok.org.
17. *International* Electrotechnical Commission (IEC). Technical Committee No. 65: Industrial Process Measurement and Control. IEC 1131 – Programmable Controllers Part 7 – Fuzzy Control Programming. – http://www.fuzzytech.com/binaries/ieccd1.pdf
18. *Bellifemine F., Caire G., Greenwood D.* Developing Multi-Agent Systems with JADE. – Chichester: John Wiley & Sons Inc, 2007. – 303 p.
19. *Суперкомпьютеры* ИК НАН Украины. – http://icybcluster.org.ua.
20. *Sendall S., Kozaczynski W.* Model transformation: the heart and soul of model-driven software development // IEEE Software. – 2003. – Vol. 20, № 5. – P. 42–45.
21. *Eclipse* Modeling Framework. – http://wiki.eclipse.org/EMF/
22. *MOF* 2.0/XMI Mapping. – http://www.omg.org/spec/XMI/2.1.1/
23. *OMG* MetaObject Facility. – http://www.omg.org/mof/
24. *Рутковская Д.*, *Пилиньский М.*, *Рутковский Л.* Нейронные сети, генетические алгоритмы и нечеткие системы. – М.: Горячая линия – Телеком, 2004. – 452 с.
25. *Леоненков А.В.* Нечеткое моделирование в среде MATLAB и fuzzyTECH. – СПб.: БХВ, 2005. – 736 с.
26. *Graphical* Modeling Framework. – http://www.eclipse.org/modeling/gmp/
27. *Eclipse* Modeling TMF. – http://www.eclipse.org/modeling/tmf/
28. *Application* Design Patterns: Producer/Consumer. – http://zone.ni.com/devzone/cda/tut/p/id/3023/
29. *Sterling L., Taveter K.* The Art of Agent-Oriented Modeling. – Cambridge: MIT Press, 2010. – 367 p.
30. *Wooldridge M.J.* An Introduction to Multiagent Systems. – Cambridge: MIT Press, 2002. – 366 p.

Получено 03.02.2011

***Об авторах:***

*Парасюк Иван Николаевич,*
член-корреспондент НАН Украины,
профессор, заведующий отделом,

*Ершов Сергей Владимирович*,
кандидат физико-математических наук,
старший научный сотрудник.

***Место работы авторов:***

Институт кибернетики
имени В.М. Глушкова НАН Украины,
03187, Киев-187,
Проспект Академика Глушкова, 40.
Тел.: (044) 526 5168