УДК 004.896, 004.414.2

ОНТОЛОГИЧЕСКОЕ ПРОЕКТИРОВАНИЕ ПОДСИСТЕМЫ ОЦЕНКИ ОБСТАНОВКИ ИНТЕЛЛЕКТУАЛЬНЫХ АГЕНТОВ

С.В. Лебедев[1], М.Г. Пантелеев[2]

Санкт-Петербургский государственный электротехнический университет «ЛЭТИ» им. В.И. Ульянова (Ленина), Санкт-Петербург, Россия

[1]*lebedev.sv.etu@gmail.com*

[2]*mpanteleyev@gmail.com*

Аннотация

Рассматривается онтологический подход к проектированию подсистем оценки обстановки (ПОО) интеллектуальных агентов (ИА), функционирующих в динамических многоагентных средах, в том числе в условиях группового противодействия. Подход основан на выделении аспектов построения ПОО, инвариантных конкретным областям применения агентов и решаемым ими задачам. В качестве теоретического базиса процесса проектирования предложено семейство моделей, формализующих различные аспекты построения и функционирования ПОО, с учётом представления разных компонентов внутренней модели мира агента, особенностей организации процесса вычислений и этапов проектирования ПОО. Предложенные модели позволяют с единых позиций проектировать ПОО для разных ИА с использованием онтологий. Выделены два класса онтологий, поддерживающих процесс проектирования: онтология ПОО и онтологии конкретных предметных областей. Онтология ПОО обеспечивает интеграцию в инвариантный каркас программного кода, реализующего функции оценки обстановки для конкретного агента. Онтология предметной области расширяет онтологию ПОО и поддерживает автоматическую генерацию программного кода и его интеграцию в инвариантный каркас ПОО. Генерация программного кода ПОО для конкретного ИА обеспечивается предложенными отображениями онтологических классов и свойств в программные структуры. Рассмотрен прототип инструментальной платформы разработки ПОО, реализованный с использованием языков представления и обработки онтологий OWL, SPARQL, SPIN и языка программирования Java. Подход иллюстрируется построением ПОО агента, функционирующего в среде виртуального футбола.

Ключевые слова: *интеллектуальный агент, подсистема оценки обстановки, онтология проектирования, инструментальная платформа.*

Цитирование: *Лебедев, С.В.* Онтологическое проектирование подсистемы оценки обстановки интеллектуальных агентов / С.В. Лебедев, М.Г. Пантелеев // Онтология проектирования. – 2016. – Т. 6, №3(21). - С. 297-316. – DOI: 10.18287/2223-9537-2016-6-3-297-316.

Введение

В последние годы онтологии все шире используются при проектировании различных классов технических систем в качестве формализованных моделей представления концептуальных знаний об этих системах [1-3]. Одним из наиболее динамично развивающихся классов систем в области искусственного интеллекта являются в настоящее время интеллектуальные агенты (ИА) и основанные на них многоагентные системы (МАС) [4, 5]. Использованию онтологий при проектировании агентных систем различного назначения посвящено достаточно много исследований, в частности [6, 7]. Однако, вследствие широкого разнообразия и сложности таких систем, проблема их онтологического проектирования ещё далека от окончательного решения и весьма актуальна в настоящее время.

В статье рассмотрен онтологический подход к автоматизации разработки программного обеспечения *подсистем оценки обстановки* (ПОО) ИА, функционирующих в *открытых динамических многоагентных* (ОДМ) мирах.

Интеллектуального агента определим как систему, способную планировать свои действия и осуществлять автономное целенаправленное поведение в ОДМ-мирах.

В процессе проектирования агентных систем при выборе архитектуры ИА одним из ключевых аспектов является тип взаимодействия агентов в МАС: кооперация в рамках априори заданной организационной структуры; поиск партнеров в открытой многоагентой среде (включая переговоры и заключение контрактов об условиях сотрудничества); конкуренция (в том числе, в рамках аукционов); антагонистическое противодействие (например, в военных приложениях).

Одним из наиболее сложных для проектирования подклассов ИА являются агенты, предназначенные для решения целевых задач в условиях группового (командного) противодействия. При построении таких ИА в настоящее время используются делиберативные архитектуры, предполагающие формирование и поддержание в памяти агента явной символьной *модели мира* (ММ). Обобщённая делиберативная архитектура ИА представлена на рисунке 1.

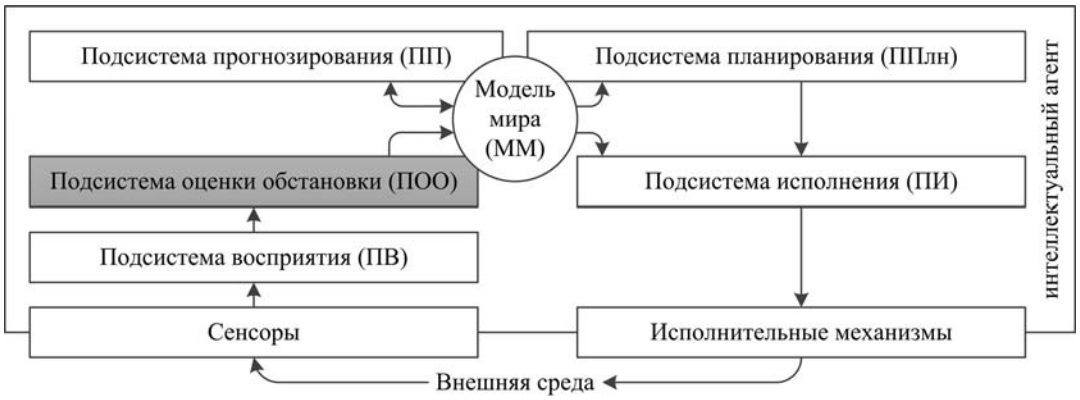

Рисунок 1 – Архитектура ИА

Важной особенностью данного класса систем является отсутствие коммуникации между агентами противодействующих групп и, как правило, ограниченная коммуникация (в некоторых случаях – отсутствие) между агентами своей группы. В таких системах агент обновляет свою ММ по результатам наблюдений за состоянием и поведением внешних объектов на основе сенсорных данных, поступающих от подсистемы восприятия.

Сенсорные данные от подсистемы восприятия являются низкоуровневыми и требуют обработки для представления ММ на более высоком уровне абстракции, необходимом для планирования агентом своих действий (принятия решений). Такая обработка выполняется ПОО.

Основными *особенностями сред функционирования* ИА рассматриваемого класса ОДМ-миров являются:

- *открытость* – состав агентов, находящихся в определённый момент времени в окружающей рассматриваемого агента среде, априори неизвестен и может меняться в процессе его функционирования как количественно, так и качественно (по типам агентов);
- *динамичность* – в результате целенаправленных действий агентов состояние мира меняется достаточно быстро и недетерминировано, т.к. намерения других агентов неизвестны (относительно этих намерений могут строиться лишь гипотезы на основе наблюдаемого поведения); в определённые моменты времени разные агенты способны в

различной степени влиять на цели и планы рассматриваемого агента, т.е. имеют для него различную прагматическую значимость;

- *реализация групповых действий в МАС* – в процессе функционирования агенты противодействующих группировок могут действовать скоординированно, динамически формируя группы различного состава на разных уровнях организационной иерархии.

Анализ указанных особенностей позволяет выделить следующие *основные задачи* ПОО агентов рассматриваемого класса, решение которых необходимо для поддержания ММ в актуальном состоянии:

- обнаружение в окружающей среде новых (вновь появившихся) агентов и фиксация выбывания отдельных агентов (например, вследствие их убытия или физического уничтожения);
- классификация типов агентов, определение (построение гипотез) по наблюдаемому поведению их намерений (распознавание планов) и, как следствие, определение (оценка) прагматической значимости этих агентов с точки зрения рассматриваемого агента;
- обнаружение фактов формирования из отдельных агентов (или групп более низкого уровня иерархии) групп скоординированно действующих агентов, изменения состава таких групп или их расформирования;
- определение (построение гипотез относительно) тактики групповых действий на разных уровнях иерархии по наблюдаемому скоординированному поведению агентов противодействующей группировки;
- формирование *ситуаций* как множества индивидуальных агентов и их групп, имеющих прагматическую значимость при планировании действий [8, 9].

1 Краткий анализ состояния проблемной области

В статье термины «оценка обстановки» и «оценка ситуации» (Situation Assessment) используются как синонимы, а содержание соответствующего понятия трактуется в смысле решения перечисленных выше задач по обновлению и поддержанию в актуальном состоянии ММ агента. Подробный анализ публикаций, рассматривающих различные аспекты этих понятий, выходит за рамки статьи. Ниже представлен краткий обзор работ, наиболее значимых и близких к предлагаемому в статье подходу.

Круг проблем, связанных с оценкой обстановки в сложных динамических средах, достаточно широк и исследуется в рамках концепций *ситуационной осведомленности* (Situation Awareness) и *слияния данных* (*Data Fusion*) [10].

Согласно наиболее известному определению, предложенному Эндсли (Endsley), ситуационная осведомлённость понимается как «восприятие элементов среды в рамках некоторого пространства и времени, осознание её смысла и проецирование её состояния на ближайшее будущее» [11]. В соответствии с данным определением в модели ситуационной осведомлённости выделены три уровня: *восприятие элементов среды* (их состояния, свойств и динамики); *осмысление текущей ситуации* на основе синтеза всех элементов уровня 1 (включая фокусировку внимания, интеграцию различных частей информации и определение их значимости для целей агента); *проекция будущих состояний*, т.е. способность прогнозировать события и динамику будущих ситуаций.

Среди моделей слияния данных наибольшее распространение получила *JDL Data Fusion Model* [12], разработанная Группой слияния данных (Data Fusion Group) Объединенной дирекции лабораторий (Joint Directors of Laboratories) Министерства Обороны США. Подходы к оценке ситуаций с использованием модели JDL рассмотрены, в частности, в [13, 14]. Представленные в этих работах модели достаточно абстрактны, не ориентированы на практиче-

ские вопросы проектирования ИА, в частности, на повышение эффективности программной реализации агентов.

Наиболее близким к рассмотренному в настоящей статье подходу, является онтолого-ориентированное построение систем оценки ситуации с использованием модели Эндсли и ситуационной теории [15]. Предложенная в этой работе модель позиционируется как общий формализованный теоретический каркас, позволяющий с единых позиций проектировать ПОО для разных областей применения. В [16, 17] рассмотрено использование модели Эндсли в системах поддержки операторов крупных центров данных. Основные идеи: повторное использование артефактов, имеющих точки расширения (за счёт применения ядерных онтологий и мета-сущностей), широкая поддержка пространственно-временных отношений, формальное описание развития отношений и ситуаций. Применение известных подходов при построении ПОО агентов, функционирующих в ОДМ-мирах, ограничено недостаточно гибкими возможностями управления процессом вычислений. Это обусловлено недостаточным уровнем гранулярности таких моделей и, как следствие, отсутствием возможности адаптации вычислительного процесса к доступному времени принятия решений.

2 Уточнение постановки задачи

Рассмотрим функционирование ПОО в составе базовой архитектуры ИА при решении выделенных выше её основных задач на концептуальном уровне, абстрагируясь от особенностей конкретных агентов и средств функционирования.

Сенсоры и средства низкоуровневой обработки поступающих от них сигналов и изображений в рассматриваемой архитектуре ИА отнесены к подсистеме восприятия (ПВ). Учитывая практическую направленность настоящей работы на создание инструментальной платформы, обеспечивающей быстрое прототипирование программного обеспечения данного класса систем, примем объектно-ориентированное представление ММ агента (см. рисунок 2).

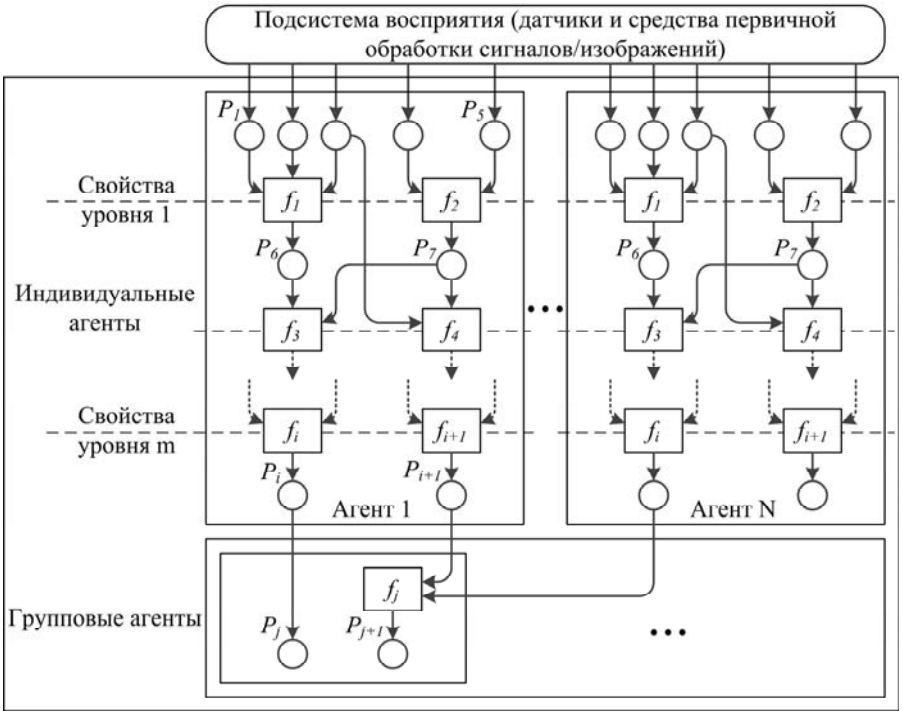

Рисунок 2 – Объектное представление внутренней модели мира ИА и уровней обработки данных

В структуре ММ выделим уровень индивидуальных агентов и, в общем случае, произвольное число уровней иерархии групп агентов (на рисунке 2 показан один такой уровень). Индивидуальные агенты и их группы в объектном представлении ММ характеризуются определённым набором свойств, состав которых зависит от типа соответствующих сущностей (на рисунке 2 показаны кружками).

Введём понятие *уровня вычисления свойства* (далее для краткости – *уровня свойства*). К первому уровню отнесём свойства индивидуальных агентов, значения которых поступают непосредственно от ПВ (на рисунке 2 – P_1-P_5). Свойство относится к k-му уровню, если для вычисления его значения используется хотя бы одно свойство (k – 1)-го уровня. Значения свойств в общем случае вычисляются некоторыми функциями f_i, зависящими от семантики свойства и метода его вычисления в конкретном приложении.

Поскольку состав агентов в открытой среде может динамически меняться, предусмотрены функции порождения нового агента в ММ (конструктор) и удаления агента из ММ (деструктор).

Порождение в ММ новых сущностей на уровне групп агентов и их удаление (расформирование) определяется на основе выделения соответствующих отношений между индивидуальными агентами (или группами более низкого уровня иерархии). Например, групповая сущность «линия нападения» в виртуальном футболе [18] выделяется (формируется в ММ), если соответствующие игроки в течение определённого интервала времени находятся ближе всего к воротам соперника и действуют примерно на одной линии.

Приведённое на рисунке 2 объектное представление ММ инвариантно предметным областям (ПрО). Выделенные уровни индивидуальных агентов, их групп, свойств и отношений, а также абстрактных функций для их вычисления, составляют абстракцию ПОО, не зависящую от среды функционирования агента и решаемых им задач. С точки зрения инструментария разработки программного обеспечения ИА данному объектному представлению соответствует инвариантный каркас ПОО.

Проектирование агентов для конкретных приложений предполагает конкретизацию данной абстракции путём задания конкретных *типов объектов*, функционирующих в среде, *состав свойств* для каждого типа, *функций для вычисления значений свойств* индивидуальных агентов и *отношений* между агентами, *позволяющих выделить* скоординированно действующие *группы* и т.п. Эти сущности на этапе проектирования могут быть взяты из онтологий соответствующих ПрО.

Исходя из изложенного, задача проектирования ПОО для ИА, функционирующих в различных средах, в статье ставится и решается в двух аспектах. *В теоретическом плане* необходимо разработать множество моделей, составляющих основу процесса проектирования ПОО, включая создание средств быстрого прототипирования ИА. В *практическом плане* рассматриваются вопросы создания инструментальной платформы для автоматизации проектирования и разработки ИА.

Традиционными подходами к решению задачи автоматизации проектирования программного обеспечения являются разработка на основе моделей и повторное использование кода. Использование онтологий в рамках модельно-ориентированной парадигмы [19] является перспективным направлением, т.к. позволяет строить (в том числе, используя графические редакторы), визуализировать и автоматически выводить элементы моделей.

3 Базовые формальные модели онтологического проектирования ПОО ИА

Общая концепция предлагаемого подхода к автоматизации проектирования и разработки программного обеспечения ПОО ИА иллюстрируется рисунком 3.

Рисунок 3 - Концепция автоматизации проектирования ПОО

Рассмотрим базовые формальные модели, лежащие в основе предлагаемого подхода.

На самом общем уровне решаемая проблема декомпозируется на две базовые задачи: построение формализованного описания рассматриваемой ПрО; проектирование ПОО агента, предназначенного для решения определённых задач в данной ПрО.

Соответственно, базовая *модель проблемной области* содержит два компонента: < *ММ*, *МПр* >, где *ММ* – метамодель ММ; *МПр* – модель проектирования.

3.1 Метамодель модели мира

Анализ особенностей взаимодействия агента со средой позволяет выделить три обобщённых компоненты, независящие от ПрО. С одной стороны, в процессе функционирования агент на основе воспринимаемых данных и априорных знаний должен формировать в своей памяти (и в каждом цикле обновлять) внутреннюю модель мира (*ВнММ*). Эта модель содержит знания агента об объектах внешней среды, их свойствах и отношениях, представленные на различных уровнях абстракции.

С другой стороны, ИА, обладая свойством *проактивности*, в каждый момент времени хранит в памяти свои текущие планы, варианты возможных будущих действий, оценки их полезности и другую информацию, влияющую на оценку текущей ситуации. Соответствующий компонент *ММ* назовём *проактивным компонентом* (*ПК*).

ПК задаёт контекст, определяющий сущности внешней среды (объекты, свойства, отношения), которые являются прагматически значимыми для агента в данный момент времени, т.е. могут влиять на его решения. Совокупность этих сущностей соответствует понятию *ситуации* и образует верхний уровень представлений агента о текущем состоянии мира. Именно этот уровень ПОО используется для идентификации классов проблемных ситуаций. Соответствующий компонент представлений агента назовём *ситуационной моделью мира* (*СММ*).

Таким образом, метамодель включает три рассмотренных выше компонента:

$$ММ = < ВнММ, ПК, СММ >.$$

3.2 Модель проектирования

Проектирование ПОО предполагает решение двух взаимосвязанных задач: выбор формализованного представления *ММ* и реализацию процесса вычисления её элементов в процессе решения задачи оценки обстановки. Для описания этих задач будем использовать соответственно *модель представления* (*МП*) и *модель вычисления* (*МВ*). Для формализации связи между этими моделями введём в рассмотрение *модель вычислимости* (*МВч*).

Таким образом, модель проектирования может быть представлена кортежем:

$$МПр = < МП, МВч, МВ >.$$

3.2.1 Модель представления

МП включает совокупность элементов, используемых для внутреннего представления воспринимаемой среды:

(1) $$МП = E = O \bigcup R,$$

где E – множество элементов; O – множество объектов; R – множество отношений между объектами.

Без потери общности можно ограничить рассмотрение бинарными отношениями:

(2) $$r(o_1,o_2).$$

Элемент представляет собой совокупность свойств: $e = \{p_1,\ldots,p_n\}$. Объект представлен множеством свойств и отношений с другими объектами:

$$o = \{p_1,\ldots,p_n; r_1,\ldots,r_m\}.$$

В зависимости от особенностей среды функционирования и физической организации подсистемы восприятия агента, объекты и их свойства могут быть либо непосредственно воспринимаемыми, либо вычисляемыми. Например, в виртуальном футболе непосредственно воспринимаемыми объектами являются игроки, мяч, линии и флаги [20], а непосредственно воспринимаемыми свойствами – координаты и скорости движения игроков в полярной системе координат. Примером вычисляемого свойства является «владение мячом», а вычисляемого объекта – групповой объект «линия нападения». Вычисляемыми отношениями являются: «находится в одной линии», «быть партнером для паса» и т. п.

3.2.2 Модель вычисления

МВ описывает компоненты (модули) вычислительного процесса, позволяющие установить факт существования элементов модели мира на основе воспринятых, ранее вычисленных и априорно известных элементов. Таким образом, множество элементов *МП* (см. формулу (1)) можно определить так:

$$E = \{e_{known}\} \bigcup \{e_{percepted}\} \bigcup \{e_{calculated}\},$$

где $\{e_{known}\}$ – множество априорно известных элементов, $\{e_{percepted}\}$ – множество воспринимаемых элементов, $\{e_{calculated}\}$ – множество вычисляемых элементов.

Вычисление базируется на допущении наличия между объектами отношений, заданных формулой (2). Если установлено существование части элементов этой конфигурации (например, объект o_1 воспринят и представлен в *ММ* агента), то может быть вычислен факт существования других элементов – объект o_2 и отношение r. Таким образом, вычислительные зависимости определены через предметные отношения.

В соответствии с принятым допущением введём в рассмотрение две базовые *функции вычисления* (ФВ):

(3) $$f_a(\dot{o}_1, \dot{o}_2, r(o_1, o_2)) = \dot{r}(\dot{o}_1, \dot{o}_2),$$

(4) $$f_c(\dot{o}_1, r(o_1, o_2)) = < \dot{r}(\dot{o}_1, \dot{o}_2), \dot{o}_2 > .$$

Здесь f_a – *ассоциирующая функция*, связывает два известных объекта некоторым отношением; f_c – *порождающая функция*, создаёт новый объект по известному объекту и отношению, связывающему данные объекты. Символы без точки соответствуют априорно известным элементам, символы с точкой – воспринятым или вычисленным элементам. Отметим, что типы отношений включены в отображения.

Примером ассоциирующей функции является вычисление отношения «владеть мячом» на основе воспринятой информации об игроке и мяче. Вычисление же объекта «владелец мяча» реализуется порождающей функцией на основе той же информации.

С учётом вышеизложенного, в реализации оценки обстановки можно выделить два ключевых аспекта:

- установление факта существования объектов – *расширение ММ*;
- установление отношений между существующими объектами – *связывание ММ*.

Процесс вычисления в ПОО состоит из множества функций вычисления и, в соответствии с требованиями к ИА, должен обладать свойствами гранулярности и иерархичности. Иерархичность представлена деревом вычислительных зависимостей, в котором вычисляемое значение зависит от известных (воспринятых и/или ранее вычисленных) значений.

Свойства гранулярности и иерархичности позволяют реализовать оценку обстановки как процесс с итеративным уточнением результата в пределах выделенного времени. Одним из фрагментов такой иерархии в виртуальном футболе может быть, например, определение владельца мяча на основе воспринятого множества игроков и мяча.

В соответствии с принципом иерархичности и гранулярности процесс делится на уровни вычисления (далее просто уровни) и узлы вычисления (далее узлы):

$$MB =< l_1, l_2, ..., l_n >,\ l_i \in L,$$

где l_i – i-ый уровень; L – множество уровней.

Каждый уровень может быть представлен кортежем:

$$l_i = <\{e\}_i,\ \{cn\}_i,\ g_i>,$$

где $\{e\}_i$ – множество элементов *ММ* i-ого уровня; $\{cn\}_i$ – множество узлов i-ого уровня, каждый из которых реализует функцию вычисления; g_i – функция вычисления $(i + 1)$-ого уровня, такая что $g_i \circ f_i = l_{i+1}$. Таким образом, уровни, в общем случае, порождаются динамически на основе значений функций вычисления нижележащего уровня.

Например, в виртуальном футболе элементами первого уровня являются непосредственно воспринимаемые элементы: линии, флаги, ворота, игроки, мяч. При этом один из узлов может реализовывать порождающую функцию вычисления владельца мяча. Вычисленный владелец мяча будет являться элементом второго уровня иерархии.

Множество $\{e\}_i$ элементов каждого i-го уровня представляет собой фрагмент *ММ*. Таким образом, можно записать:

$$MM = \{e\}_1 \cup \{e\}_2 \cup ... \cup \{e\}_n : \{e\}_i \cap \{e\}_j = \varnothing,\ i \neq j.$$

При этом множество элементов каждого уровня формируется динамически в результате восприятия (для 1-го уровня) или вычисления (для остальных уровней).

Каждый узел может быть представлен кортежем:

$$cn =< \{e\}_{pr},\ \{e\}^{in}_{pr},\ \{e\}^{out}_{pr},\ pr,\ l_i >,$$

где pr – процедура вычисления элементов, реализующая функцию вычисления; $\{e\}_{pr}$ – обязательное множество аргументов процедуры pr, такое что $\{e\}_{pr} \subseteq \{e\}_i$; $\{e\}^{in}_{pr}$ – необязательно множество аргументов процедуры pr, такое что $\{e\}^{in}_{pr} \subseteq \{e\}_1 \cup \{e\}_2 \cup ... \cup \{e\}_{i-1}$; $\{e\}^{out}_{pr}$ – множество результирующих элементов процедуры pr, такое что $\{e\}^{out}_{pr} \subseteq \{e\}_{i+1}$; l_i – i-ый уровень, к которому относится данный узел.

Для рассматриваемого в данном примере узла множество обязательных аргументов процедуры включает игроков и мяч, элементом результирующего множества является «владелец мяча».

Структура вычислительного процесса, соответствующая описанной модели, представлена рисунке 4.

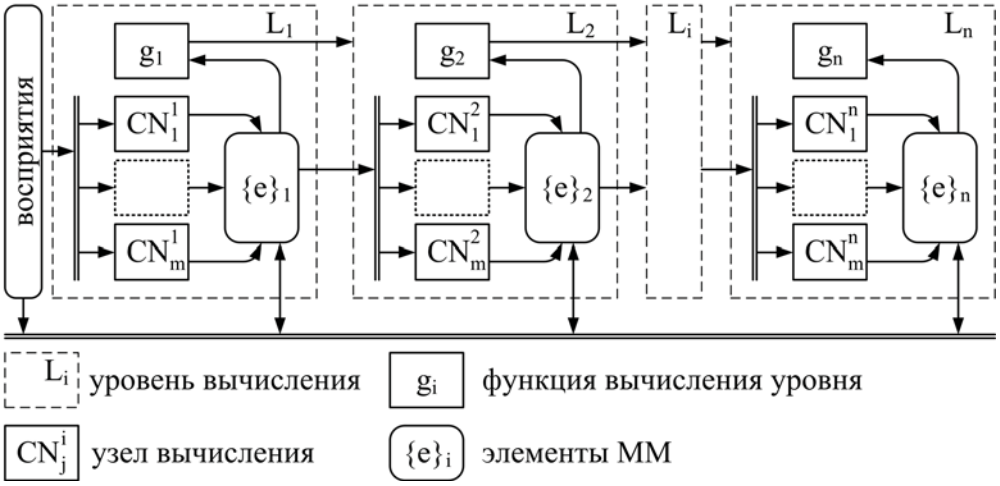

Рисунок 4 – Структура вычислительного процесса, соответствующая МВ

Важным свойством предлагаемой модели является динамическое порождение уровней и узлов, что обеспечивает гибкость в организации процесса вычислений и позволяет выделять ресурсы только по мере необходимости.

3.2.3 Модель вычислимости

МВч представляет собой отображение *МП* в *МВ*, в основе которого лежит использование функций вычисления, как связующего звена между элементами указанных моделей. Данная модель позволяет переходить от *МП* к *МВ* на основе описываемых далее отображений.

Отображение *МП* в *МВч* задаётся через ассоциирующие и порождающие вычислительные отношения:

$$r_a, r_c \in R_f : Domain \to Range .$$

Введём отображение предметных отношений в функции вычисления:

(5) $$R \to R_f .$$

Тогда функции вычисления (3) и (4) могут быть записаны в виде:

- для $r \to r_a$, $f_a : < Domain, Range > \to r_a$;
- для $r \to r_c$, $f_c : Domain \to < Range, r_c >$.

Для представления отображения *МВч* в *МВ* введём понятие *звена вычисления* (далее просто звено) *CU:*

$$CU = < \{e_b\}, \{e_d\}, f > ,$$

где $f \in F_a \bigcup F_c, \{e_b\}$ – множество базовых элементов звена; $\{e_d\}$ – множество производных элементов звена.

Каждое вычислительное отношение может быть преобразовано в звено следующим образом:

- для $r \to r_a$: $\{e_b\} = Range \bigcup Domain$, $\{e_d\} = r$;
- для $r \to r_c$: $\{e_b\} = Domain$, $\{e_d\} = r \bigcup Range$.

Отображение $CU \to CN$, множества звеньев вычисления *CU* во множество узлов вычисления *CN*, должно удовлетворять следующим условиям:

(6)
$$\begin{cases} \{e\}_{pr} \cup \{e\}_{pr}^{in} = \{e_b\}, \\ \{e\}_{pr}^{out} = \{e_d\}, \\ pr \text{ является реализацией } f, \\ l_i \text{ устанавливается динамически.} \end{cases}$$

Рассмотренные отображения обеспечивают реализацию принципа соответствия одного представления множеству вариантов организации вычислений. Отображение (5) позволяет связать с одним предметным отношением вычислительные отношения двух разных типов – ассоциирующее или порождающее. Отображение (6) позволяет динамически определять уровень, на котором в ПОО будет реализована данная функция и, кроме того, сопоставлять одной функции вычисления несколько программных реализаций (отличающихся, например, языком программирования).

3.3 Подход к проектированию

Предлагаемый подход к проектированию ПОО предполагает дополнение рассмотренных базовых моделей моделями конкретной ПрО, для которой проектируются ИА. Собственно процесс проектирования ПОО включает последовательные отображения, заданные формулами (5) и (6), и состоит из трёх этапов:

1) построение *МП* для *ВнММ*, *ПК* и *СММ*, путём расширения базовой *МП* объектами и отношениями выбранной ПрО;
2) отображение полученных *МП* в *МВч* в соответствии с отображением, представленным формулой (5);
3) отображение полученной *МВч*, а соответственно и *МП*, в *МВ* в соответствии с отображением, представленным формулой (6).

3.4 Пример

Проиллюстрируем предлагаемый подход к проектированию ПОО на примере агента, функционирующего в среде виртуального футбола. В этой среде состояние многоагентного мира моделирует сервер, обновляя это состояние на основе команд, получаемых в каждом такте от всех игроков, и генерируя им неполные и неточные сенсорные данные.

Подсистема восприятия агента в каждом такте получает от сервера следующие данные: координаты статических (линии, флаги, ворота) и динамических (другие игроки, мяч) объектов в полярных координатах относительно себя. Статические объекты используются в качестве навигационных привязок для определения собственных координат.

Необходимо построить модель ПОО агента, обеспечивающую:

- вычисление собственных абсолютных координат агента на основе относительных координат навигационных объектов;
- определение владельца мяча на основе абсолютных координат мяча и игроков.

В соответствии с предложенным подходом для решения задачи проектирования ПОО агента разработаны:

- инвариантные онтологии и программный каркас ПОО, реализующие представленные модели;
- онтология и программная структура, соответствующая ПрО; каждый из этих артефактов в качестве основы использует соответствующую онтологию;
- способ автоматизированной интеграции онтологии и программного кода.

При решении поставленной задачи использовались следующие языки и инструменты:

- Java – основной язык программирования при реализации платформы проектирования;

- OWL [21], SPIN [22], SPARQL [23] – языки представления и обработки онтологических знаний;
- TopBraid Composer Free Edition [24] – среда разработки, поддерживающая работу со всеми указанными выше языками.

При разработке сквозного сценария проектирования для апробации предлагаемого подхода были приняты следующие упрощающие допущения, не влияющие на общность подхода:

- формируемая на основе онтологической модели структура вычислений в ПОО *является статической* (на уровне модели это означает отождествление понятий узла и звена);
- рассматриваются *достаточно простые конфигурации отношений* между объектами;
- *проактивный контекст* оценки обстановки предполагается статичным и *не учитывается* (*СММ* и *ПК* исключены из рассмотрения).

3.4.1 Онтология ПОО

В инвариантной онтологии ПОО (core.rdf) формализованы модели проектирования – *МП*, *МВч*, *МВ*. На рисунке 5 представлена иерархия классов данной онтологии.

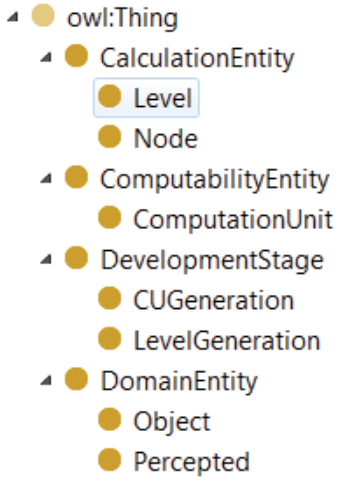

Рисунок 5 - Иерархия классов онтологии ПОО

Базовыми онтологическими классами *МПр* являются: *DomainEntity*, *ComputabilityEntity*, *CalculationEntity*.

Класс *DomainEntity* является суперклассом всех классов, используемых для построения *МП*. Класс *Percepted* соответствует элементам, поступающим от ПВ. Отношения между объектами записываются с помощью стандартной конструкции языка OWL – *Object Property*.

Для класса *ComputabilityEntity* определён подкласс *ComputationUnit*, используемый для представления звена.

Класс *CalculationEntity* имеет подклассы, представляющие элементы процесса вычисления – узлы и уровни вычисления.

Кроме рассмотренных классов в онтологии определён ряд свойств, представленных ниже в формате *имя свойства* (*область определений* → *область значений*):

- *hasBase*(*ComputationUnit* → *DomainEntity*) – описывает базовые элементы звена;
- *hasDerivative*(*ComputationUnit* → *DomainEntity*) – описывает производные элементы звена;
- *computable* и его дочерние свойства *associative* и *generative* – представляют ассоциирующие и порождающие вычислительные отношения (для этих свойств области определения и значения не заданы);
- *level*(*ComputationUnit* → *Level*) – описывает уровень вычисления данного звена;
- *number*(*Level* → *xsd:integer*) – номер («высота») уровня вычисления.

3.4.2 Автоматизация процесса проектирования

Процесс проектирования предполагает расширение базовой онтологии элементами ПрО. В рамках рассматриваемого примера разработана онтология ПрО (soccer.rdf), в которую импортирована базовая онтология ПОО.

Процесс проектирования в соответствии с п. 3.3 включает:

1) определение объектов и отношений (свойств) ПрО;
2) отображение предметных отношений в вычислительные отношения;
3) формирование звеньев и структуры процесса вычисления.

3.4.3 Онтология виртуального футбола

В результате первых двух этапов процесса проектирования разработана онтология ПрО, фрагмент которой представлен на рисунке 6. Кроме классов, определён ряд отношений (свойств), перечисленных ниже в функциональном формате (ассоциативные отношения отмечены признаком «a», порождающие – признаком «g»):

- *beBase4Ego*(*BodyState → Ego; g*) – порождает внутреннее представление собственного состояния агента-футболиста на основе восприятия;
- *seeLandmark*(*Ego → Flag, Goal, Line; a*) – фиксирует множество видимых агентом статических объектов, которые могут использоваться для навигации на футбольном поле;
- *identifiedAs*(*Player → IdentifiedPlayer; g*) – порождает внутреннее представление об игроке на основе соответствующего восприятия (в виртуальном футболе данная операция включает идентификацию игрока и отнесение его к определённой команде);
- *subClassOfEgo*(*seeLandmark → EgoAbsolute; g*) – порождает внутреннее представление абсолютных координат, вычисляемых на основе видимых навигационных объектов;
- *ownsBall*(*EgoAbsolute, IdentifiedPlayer → Ball; a*) – определяет владельца мяча.

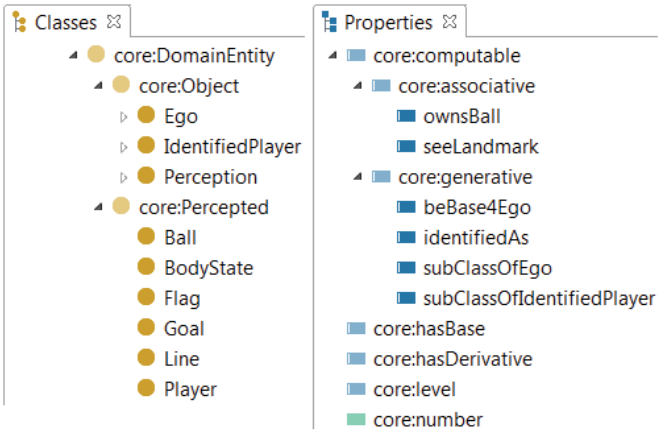

Рисунок 6 - Фрагменты онтологии предметной области (классы и свойства)

Отображение предметных отношений в вычислительные выполняется через наследование соответствующего ассоциирующего или порождающего свойства (см. рисунок 6).

3.4.4 Автоматизация процесса проектирования

На заключительном этапе проектирования в качестве средства автоматизации используется редактор TopBraid Composer Free Edition, поддерживающий язык SPIN. Данный язык позволяет связывать SPARQL-запросы с онтологическими классами. Запросы выполняются встроенным модулем вывода на экземплярах соответствующих классов. В частности, для порождения новых сущностей (классов, экземпляров, свойств) используются SPARQL-запросы *CONSTRUCT*. Другой особенностью языка SPIN является возможность определения пользовательских функций, представляющих собой запросы на языке SPARQL, которые можно вызывать из других SPARQL-запросов.

Запуск правил на экземплярах класса позволил построить небольшую иерархию классов-этапов процесса проектирования (см. рисунок 5) – *DevelopmentStages*: *CUGeneration* и *LevelGeneration*. Создание экземпляров данных классов позволяет запускать правила, порождающие элементы онтологии на соответствующем этапе проектирования.

Класс *CUGeneration* включает правила порождения звеньев:

- звеньев первого уровня, играющих роль объектов-посредников с ПВ (такие звенья-посредники не имеют базовых элементов);
- звеньев, реализующих ассоциирующие или порождающие функции.

Кроме того, класс *CUGeneration* включает правило для привязки звеньев-посредников к первому уровню.

Класс *LevelGeneration* включает правило порождения первого уровня и правило привязки звена к уровню, которое является наиболее сложным. При построении правила привязки используются понятия звена-реципиента и звена-донора: базовые элементы первого являются производными элементами второго. Правило реализует следующий алгоритм.

1) фиксируются базовые элементы реципиента, при этом реципиент не должен быть привязан к какому-либо уровню.
2) ищутся доноры, у которых в качестве производных элементов выступают базовые элементы. Эти доноры должны быть привязаны к некоторому уровню.
3) выбирается донор, привязанный к наиболее высокому уровню, по которому определяется уровень реципиента. Все базовые элементы должны быть поддержаны донорами.

Соответствующее правило представлено на рисунке 7.

```
CONSTRUCT {
 ?recipient :level ?nextLevel .
}
WHERE {
 ?recipient a :ComputationUnit .
 MINUS {
   {
     SELECT DISTINCT ?recipient
     WHERE {
      {
        ?recipient a :ComputationUnit .
        BIND (:CountBases(?recipient) AS ?cb) .
        BIND (:CountDonors(?recipient) AS ?cd) .
        FILTER (((?cd = 0) || (?cd = 0)) || (?cb != ?cd)) .
       }.
     }
   } .
 } .
 MINUS {
   ?recipient :level ?anyLevel .
 } .
 BIND (:FindMaxLevel(?recipient) AS ?level) .
 BIND (:FindNextLevel(?level) AS ?nextLevel) .
}
```

Рисунок 7 – Правило связывания звена вычисления и уровня вычисления

Данное правило использует следующие пользовательские функции:

- *FindMaxLevel* – реализует поиск максимального уровня, к которому привязан донор;
- *FindNextLevel* – вычисляет уровень реципиента;
- *CountBases* – подсчитывает количество базовых элементов звена;
- *CountDonors* – подсчитывает количество доноров звена.

Класс *ComputationUnit* также содержит правило для порождения нового уровня: при добавлении нового звена на самый высокий среди имеющихся уровень, порождается следующий уровень.

На рисунке 8 представлены звенья, их взаимозависимость и принадлежность к уровню, полученные автоматически в результате обработки SPARQL-запроса.

Логика порождаемой иерархии звена заключается в следующем. Часть восприятий переводится во внутреннее представление, что позволяет, например, сохранять историю. Это относится к состоянию агента (*cd_beBas4Ego*) и видимым игрокам (*cd_idetifiedAs*). Навигационные объекты связываются с внутренним представлением агента (*cd_seeLandmark*), что позволят в явном виде использовать полученное отношение для вычисления абсолютных координат (*cd_subClassOfEgo*). В соответствии с протоколом работы футбольного сервера воспринятое собственное состояние не включает абсолютных координат. Но эти координаты могут быть вычислены на основе расстояния и угла относительно навигационных объектов. Абсолютные координаты навигационных объектов относятся к априорным знаниям. Последний шаг – это вычисление владельца мяча

(*cd_ownBall*), исходя из собственного положения, положения видимых игроков и положения мяча.

donor	recipient	level
	cd_perception	level_1
cd_perception	cd_beBase4Ego	level_2
cd_perception	cd_identifiedAs	level_2
cd_beBase4Ego	cd_seeLandmark	level_3
cd_perception	cd_seeLandmark	level_3
cd_identifiedAs	cd_subClassOfIdentifiedPlayer	level_3
cd_seeLandmark	cd_subClassOfEgo	level_4
cd_perception	cd_ownsBall	level_5
cd_subClassOfEgo	cd_ownsBall	level_5
cd_subClassOfIdentifiedPlayer	cd_ownsBall	level_5

Рисунок 8 – Вычислительные зависимости между звеньями вычисления и их привязка к уровням вычисления

На рисунке 9 с некоторыми упрощениями представлена схема автоматизированного порождения элементов *MB*. В ручном режиме описываются классы, отношения и определяется тип отношений. После этого автоматически с использованием запроса CONSTRUCT выводятся элементы процесса вычислений.

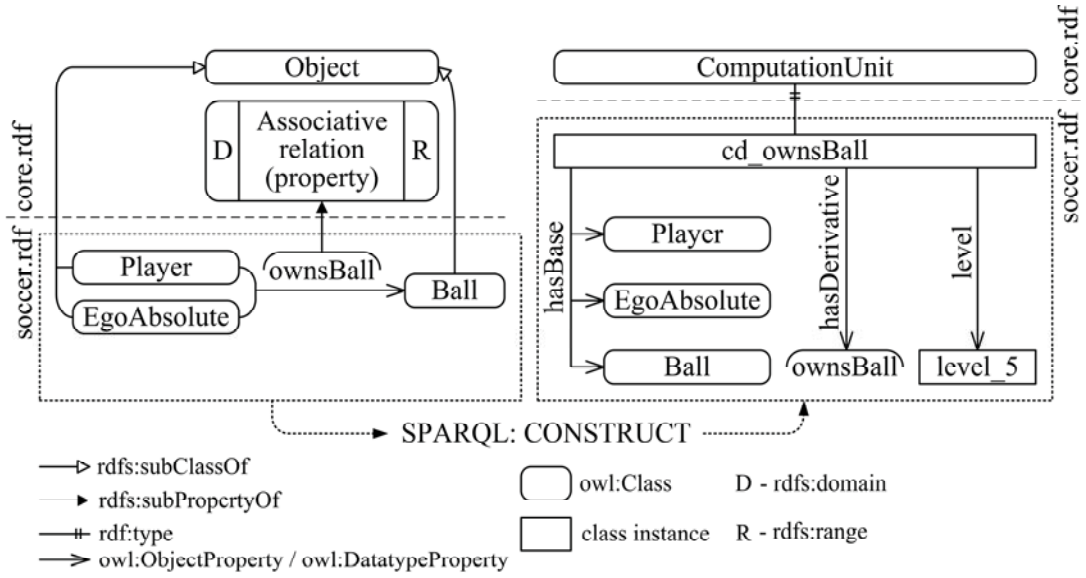

Рисунок 9 – Автоматизация генерации элементов *MB* на уровне онтологии

3.4.5 Программный каркас

Программный каркас реализован на языке Java версии 1.8. В настоящее время каркас включает две подсистемы – восприятия и оценки обстановки. Подсистема восприятия предназначена для взаимодействия с сервером виртуального футбола и работает в отдельном вычислительном потоке. Данная подсистема реализует обмен сообщениями и преобразование полученных текстовых сообщений из текстового формата S-expression в объекты.

Главный класс приложения инициализирует каждую из подсистем, после чего запускает их в бесконечном цикле. Цикл начинается с ожидания сенсорных данных. После их получения они передаются ПОО (push-подход). После этого управление передается ПОО.

Программный каркас ПОО представлен тремя базовыми классами:

- *LevelHolder* – класс-контейнер, содержащий все уровни обработки информации в ПОО;
- *LightweightLevel* – класс-контейнер уровней, содержащий все узлы данного уровня;
- *LightweightNode* – абстрактный класс, представляющий узел.

Классы *LevelHolder* и *LightweightLevel* реализуют метод *process()*, в теле которого вызываются одноименные методы внутренних объектов.

Класс *LightweightNode* также содержит абстрактный метод *process()*, который должен быть реализован пользователем в соответствии с логикой ПрО (например, вычисление абсолютных координат объекта в виртуальном футболе). Кроме того, в классе *LightweightNode* определён абстрактный метод *pullBases*() для получения данных от узлов-поставщиков (доноров), т.е. используется pull-подход.

Процесс вычисления организован следующим образом (см. рисунок 10). При получении управления каждый узел (кроме узла 1-го уровня) пытается получить информацию от узлов-доноров. Если информация доступна, выполняются необходимые вычисления и результат сохраняется в полях объекта, доступных узлам-потребителям вышележащих уровней. Для упрощения взаимодействия программных компонентов между узлами всегда передаются списки объектов.

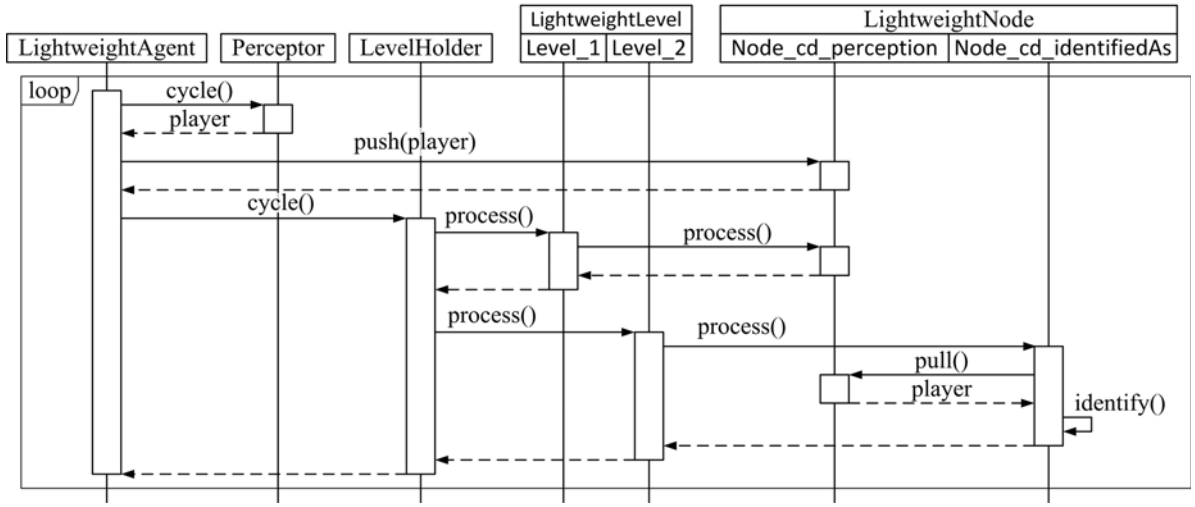

Рисунок 10 – Фрагмент цикла вычисления

3.4.6 Генерация программного кода

Для генерации программного кода использовалась библиотека CodeModel [25], позволяющая компоновать программный код Java-классов. Библиотека представляет собой набор вызовов, каждый из которых создаёт заданную структуру программного класса (поля, методы, конструкторы). Над этими функциями был надстроен функционал преобразования онтологических данных в программный код, реализованный в классе *Translator*. Для доступа к онтологии использовались средства TopBraid API [26] и Jena [27].

Для получения данных из онтологии использовались следующие SPARQL запросы:

- получение всех звеньев с номерами уровней, к которым они привязаны, для генерации уровней и узлов;
- получение базовых и производных элементов для заданного звена для генерации полей класса узла, представляющих собой списки базовых и порождаемых элементов;
- получение узлов доноров совместно с уровнями, к которым они привязаны, для уточнения конструктора класс узла таким образом, чтобы в него передавались уровни, включающие узлы-доноры (при инициализации извлекаются по имени класса узла-

донора). Узлы-доноры используются в методе *pullBases()*, который также порождается автоматически. Также добавляются поля класса, представляющие узлы-доноры.

Перед выполнением запросов необходимо запустить логический вывод для формирования структуры вычислительного процесса в рамках онтологии.

Кроме порождения классов для уровней и узлов порождается класс *LevelNodeInitializer*, инициализирующий структуры процесса вычисления: создание экземпляров классов уровней и узлов, с передачей последним уровней, от которых они зависят.

После порождения кода требуется его интеграция в каркас и определение методов *process()* каждого из узлов вычисления. В будущем эти методы могут быть специфицированы на этапе создания онтологии, если соответствующие функции будут доступны из готовых библиотек.

3.4.7 Результаты апробации подхода

Разработанный прототип продемонстрировал возможность построения онтологии выбранной ПрО на базе инвариантной онтологии и возможность генерации программного кода по построенной онтологии ПрО. Достоинствами предлагаемого подхода являются:

- Возможность использования графических средств для построения онтологии ПрО. Выбранный редактор имеет встроенные средства, позволяющие на этапе построения онтологии просматривать структуру формируемого в ПОО вычислительного процесса и контролировать её правильность уже на ранних этапах проектирования.
- Поддержка автоматической генерации онтологических сущностей (в реализованном прототипе автоматически сгенерировано 55 троек).
- Поддержка автоматической генерации программного кода. В реализованном прототипе было автоматически сгенерировано 12 классов, общий объём сгенерированного кода составил 291строку.

Вместе с тем, разработанный прототип выявил ряд требующих решения проблем.

- Поддержка на онтологическом и программном уровне групповых объектов, порождаемых на основе множества других (более простых) объектов, связанных специфическим отношением. Примером является определение тактического построения команды на основе ранее вычисленных линий обороны, полузащиты и нападения. При этом для каждого из специфических отношений будет создано своё звено, тогда как по логике вычисления звено должно быть одно. Т.е. вместо одного звена, вычисляющего формацию на основе трёх объектов, будет создано три отдельных звена, каждое из которых будет вычислять формацию на основе одной линии.
- Переход от отношений-свойств к отношениям-классам, обусловленный, например, необходимостью расширения отношений специфическими свойствами.
- Реализация динамического свойства МВ (динамическое порождение уровней и узлов). В качестве базовых вариантов можно назвать централизованное порождение и распределённое порождение, на каждом уровне вычисления отдельно.
- В рамках изложенного подхода предлагается переходить от предметных отношений к вычислительным зависимостям. С этим переходом связана проблема вычисления на уровне свойств отдельного объекта, когда одни свойства объекта вычисляются на основе других свойств и отношений, в которых участвует данный объект. В предложенном примере это решено вводом специфического отношения наследования, например, *subClassOfEgo*, область определения которого включает свойство *seeLandmark*. Иным вариантом может стать выделение подмодели, формализующей зависимости между свойствами и отношениями одного класса.

Эта проблема может быть рассмотрена в контексте более общей проблемы – общеприменимости принципа порождения вычислительных зависимостей на основе предметных отношений;

- Сложные задачи трудно решить на едином уровне концептуализации, когда эффективней ввести различные интерпретации для сущностей, например, как пространственно-временных и структурно-организационных. Под каждую такую интерпретацию необходимо строить свою онтологию и обеспечивать их взаимное отображение.

4 Будущие направления исследований

Перспективы практического развития данной работы связаны с реализацией полнофункциональной среды проектирования, поддерживающей сквозные сценарии проектирования ПОО для ИА различного назначения. Эти сценарии должны включать генерацию программных классов объектов и отношений по онтологическим сущностям, а также реализацию и подключение специфических функций вычисления. Механизм подключения предполагается реализовать через специфицирование библиотек и функций в онтологии ПрО. В теоретическом плане требуют дальнейшего анализа проблемы, сформулированные в п. 3.4.7.

Конечной целью проекта является создание интегрированной системы проектирования ИА РВ, обеспечивающей совместное проектирование программного обеспечения всех подсистем ИА, включая ПОО и подсистемы планирования.

Заключение

Разработан и апробирован онтологический подход к автоматизации проектирования и разработки ПОО ИА, основанный на повторном использовании кода и модельно-ориентированном проектировании. Онтологии позволяют применять методы автоматической генерации программного кода, который затем интегрируется в инвариантный вычислительный каркас ПОО.

В процессе апробации подхода разработаны инвариантные предметной области базовая онтология и программный каркас ПОО. При проектировании агента для среды виртуального футбола построена доменная онтология и апробирована технология автоматической генерации программного кода.

Результаты апробации подтвердили работоспособность и эффективность предложенного онтологического подхода к автоматизации проектирования программного обеспечения ПОО ИА. Вместе с тем, апробация позволила уточнить проблемы, требующие дальнейшей разработки.

Список источников

[1] ***Боргест, Н.М.*** Онтология проектирования: теоретические основы. Ч. 1. Понятия и принципы / Н.М. Боргест // уч. пособ. – Самара: Изд-во Самар. гос. аэрокосм. ун-та, 2010. – 88 с.

[2] ***Мордвинов, В.А.*** Онтология моделирования и проектирования семантических информационных систем и порталов / В.А. Мордвинов // Справочное пособие. – М.: МИРЭА, 2005. – 237 с.

[3] ***Скобелев П.О.*** Онтологии деятельности для ситуационного управления предприятиями в реальном времени / П.О. Скобелев // Онтология проектирования. – 2012.– № 1(3). – С. 6–38.

[4] ***Russell S. J., Norvig P.*** Artificial Intelligence. A modern approach // Artificial Intelligence. – 3rd edition. – Prentice-Hall, 2010. – 1152 p.

[5] ***Пузанков, Д.В.*** Интеллектуальные агенты, многоагентные системы и семантический Web: концепции, технологии, приложения / Д.В. Пузанков, В.И. Мирошников, М.Г. Пантелеев, А.В. Серегин – СПб.: Изд-во «Технолит», 2008. – 292 с.

[6] ***Ed. Hadzic, M., Chang, E. J., Wongthongtham, P.*** Ontology-Based Multi-Agent Systems // Studies in Computational Intelligence. – Berlin: Springer-Verlag, 2009. – Т. 219. – XIV. – 274 p.

[7] ***Лахин, О.И.*** Подход к разработке прототипа интеллектуальной системы поддержки принятия согласованных решений при проектировании малоразмерных космических аппаратов на основе мультиагентных технологий / О.И. Лахин, Е.В. Симонова, П.О. Скобелев // Информационно-управляющие системы. – 2015. – № 2. – С. 43-48.

[8] ***Пантелеев, М.Г.*** Концепция построения интеллектуальных агентов реального времени на основе модели опережающего итеративного планирования / М.Г. Пантелеев // Труды 12-ой национальной конф. по искусственному интеллекту с международным участием КИИ-2012. – Белгород: Изд-во БГТУ, 2012. – Т 3. – С.25-33.

[9] ***Пантелеев, М.Г.*** Формальная модель опережающего итеративного планирования действий интеллектуальных агентов реального времени / М.Г. Пантелеев // //Труды 14-ой национальной конф. по искусственному интеллекту с международным участием КИИ-2014. – Казань: Физматлит, 2014. – Т 1. – С.323-334.

[10] ***Foo P.H., Ng G.W.*** High-level Information Fusion: An Overview // J. Adv. Inf. Fusion. – 2013. – V. 8. – №. 1. – pp. 33-72.

[11] ***Endsley M.R.*** Toward a theory of situation awareness in dynamic systems // Human Factors: The Journal of the Human Factors and Ergonomics Society. – 1995. – V. 37. – №. 1. – pp. 32-64.

[12] ***Bowman C.L., Steinberg A.N.*** Systems Engineering Approach for Implementing Data Fusion Systems/ Llinas J., Hall D. L., Liggins M. E. (ed.). Handbook of Multisensor Data Fusion: Theory and Practice. – CRC Press, 2009. – pp. 561-596.

[13] ***Steinberg A.N., Bowman C.L.*** Revisions to the JDL Data Fusion Model / Llinas J., Hall D. L., Liggins M. E. (ed.). Handbook of Multisensor Data Fusion: Theory and Practice. – CRC Press, 2009. – pp. 44-67.

[14] ***Steinberg, A.N.*** Foundations of Situation and Threat Assessment / Llinas J., Hall D. L., Liggins M. E. (ed.). Handbook of Multisensor Data Fusion: Theory and Practice. – CRC Press, 2009. – pp. 437-501.

[15] ***Kokar, M.M., Matheus, C.J., Baclawski K.*** Ontology-based situation awareness //Information fusion. – 2009. – V.10. – № 1. – pp. 83-98.

[16] ***Baumgartner N., Gottesheim W., Mitsch S., Retschitzegger W., Schwinger W.*** BeAware!—situation awareness, the ontology-driven way // Data & Knowledge Engineering. – Elsevier, 2010. – V. 69. – №. 11. – pp. 1181-1193.

[17] ***Baumgartner N., Mitschb S., Muller A., Retschitzegger W., Salfinger A., Schwinger W.*** A tour of BeAware–A situation awareness framework for control centers //Information Fusion. – 2014. – V. 20. – pp. 155-173.

[18] RoboCup Soccer Server for Soccer Server Version 7.07. Users Manual / Mao Chen, Klaus Dorer, Ehsan Foroughi, Fredrik Heintz, ZhanXiang Huang, Spiros Kapetanakis, Kostas Kostiadis, Johan Kummeneje, Jan Murray, Itsuki Noda, Oliver Obst, Pat Riley, Timo Steffens, Yi Wang and Xiang Yin. February 11, 2003. – 140 p. https://sourceforge.net/projects/sserver/files/rcssmanual/9-20030211/manual-20030211.pdf/download (Valid on 26.02.2016).

[19] ***Pan J. Z., Staab S., Abmann U., Ebert J., Zhao Y.*** Ontology-driven software development. – Berlin: Springer-Verlag, 2013. – 344 p.

[20] Robocup Soccer Server. User manual. – URL: https://sourceforge.net/projects/sserver/files/rcssmanual/9-20030211/manual-20030211.pdf/download (дата обращения 26.02.2016).

[21] OWL 2 Web Ontology Language Primer (Second edition). – URL: https://www.w3.org/TR/owl2-primer (дата обращения 26.02.2016).

[22] SPIN – Overview and Motivation. – URL: https://www.w3.org/Submission/spin-overview (дата обращения 26.02.2016).

[23] SPARQL 1.1 Query Language. – URL: https://www.w3.org/TR/sparql11-query (дата обращения 26.02.2016).

[24] TopBraid Composer. Getting Started Guide. – URL: http://www.topquadrant.com/resources/products/docs/TBC-Getting-Started-Guide.pdf (дата обращения 26.02.2016).

[25] CodeModel project. – URL: https://codemodel.java.net (дата обращения 26.02.2016).

[26] The TopBraid SPIN API. – URL: http://topbraid.org/spin/api (дата обращения 26.02.2016).

[27] Apache Jena. – URL:https://jena.apache.org (дата обращения 26.02.2016).

ONTOLOGY-DRIVEN DESIGN APPROACH TO DEVELOPMENT OF SITUATION AWARENESS SUBSYSTEM OF INTELLIGENT AGENTS

S.L. Lebedev[1], M.G. Panteleyev[2]

Saint Petersburg Electrotechnical University "LETI", St. Petersburg, Russia
[1]*lebedev.sv.etu@gmail.com*, [2]*mpanteleyev@gmail.com*

Abstract

An ontology-driven design approach to development of situation awareness subsystem (SAS) of intelligent agents is discussed. It is supposed that agents act in multi-agent competitive environments. The approach is based on the separation of domain independent and domain specific aspects. A set of domain independent models is proposed as a theoretical underpinning of the approach. Those models formalize different aspects of design and development of SAS including internal agent's representation of an environment, situation awareness calculation stages and SAS design stages. The models are used to build ontological and program components, used for SAS design and development. Two classes of ontologies are defined: SAS ontology and ontologies of specific domains. SAS ontology is used to provide domain specific program code integration into the domain independent SAS framework. Domain specific ontology is used to represent domain specific concepts on the base of SAS ontology concepts. SAS development process includes the following stages: domain specific ontology construction, automatized generation of domain specific code, integration of domain specific code into the SAS framework and implementation of domain specific functions within the framework. Automatized code generation is based on the proposed mappings of ontological classes and properties to the program structures. A toolset is proposed for building domain specific SAS. The toolset utilizes such languages from the Semantic Web stack as OWL, SPARQL and SPIN. The toolset is written in Java language. The approach is illustrated and evaluated on the simplified virtual soccer scenario.

Key words: *intelligent agent, situation assessment, ontology-driven development, toolset.*

Citation: *Lebedev SL, Panteleyev MG.* Ontology-driven design approach to development of situation awareness subsystem of intelligent agents [In Russian]. *Ontology of designing*. 2016; 6(21): 297-316. DOI: 10.18287/2223-9537-2016-6-3-297-316.

References

[1] *Borgest NM.* Ontology of Designing: theoretical foundations. Part 1. Concepts and definitions: textbook [In Russian]. – Samara: Samara airspace university publ., 2010. — 88 p.

[2] *Mordvinov VA*. Ontology of modeling and designing of semantic information systems and portals. Reference guide [In Russian]. – Moscow: MIREA, 2005. – 237 p.

[3] *Skobelev PO.* An activity ontology for enterprise real-time situation management [In Russian]. Ontology of Designing. 2012; 1(3): 6–38.

[4] *Russell SJ, Norvig P*. Artificial Intelligence. A modern approach. – 3rd edition. – Prentice-Hall, 2010. – 1152 p.

[5] *Puzankov DV, Miroshnikov VI, Panteleev MG, Seregin AB*. Intelligent agents, multiagent systems and Semantic Web: concepts, technologies and applications [In Russian]. – SPb.: "Tekhnolit" publ., 2008. – 292 p.

[6] *Hadzic EdM, Chang EJ, Wongthongtham P*. Ontology-Based Multi-Agent Systems. Studies in Computational Intelligence. – Berlin: Springer-Verlag, 2009. – V. 219. – XIV. – 274 p.

[7] *Lahin OI, Simonova IB, Skobelev PO*. Prototype intelligent system of coordinated decision-making support for small spacecraft design on the basis of multi-agent technologies [In Russian]. Information and Control Systems. – 2015; 2: 43-48.

[8] *Panteleev MG*. A conception of a real-time intelligent agents development on the basis of proactive iterative planning [In Russian]. Proc. of the 12 National Conference on Artificial Intelligence with International Participation KII-2012. – Belgorod: BGTU, 2012. – V 3. – pp. 25-33.

[9] *Panteleev MG.* A formal model of proactive iterative action planning for real-time intelligent agents [In Russian]. Proc. of the 14 National Conference on Artificial Intelligence with International Participation KII-2014. – Kazan: Fizmatlit, 2014. – V 1. – pp.323-334.

[10] *Foo PH, Ng GW*. High-level Information Fusion: An Overview. J. Adv. Inf. Fusion. – 2013; 8(1): 33-72.
[11] *Endsley MR*. Toward a theory of situation awareness in dynamic systems. Human Factors: The Journal of the Human Factors and Ergonomics Society. – 1995; 37(1): 32-64.
[12] *Bowman CL, Steinberg AN*. Systems Engineering Approach for Implementing Data Fusion Systems. Llinas J., Hall D. L., Liggins M. E. (ed.). Handbook of Multisensor Data Fusion: Theory and Practice. – CRC Press, 2009. – pp. 561-596.
[13] *Steinberg AN, Bowman CL*. Revisions to the JDL Data Fusion Model. Llinas J., Hall D. L., Liggins M. E. (ed.). Handbook of Multisensor Data Fusion: Theory and Practice. – CRC Press, 2009. – pp. 44-67.
[14] *Steinberg AN*. Foundations of Situation and Threat Assessment. Llinas J., Hall D. L., Liggins M. E. (ed.). Handbook of Multisensor Data Fusion: Theory and Practice. – CRC Press, 2009. – pp. 437-501.
[15] *Kokar MM, Matheus CJ, Baclawski K*. Ontology-based situation awareness. Information fusion. – 2009. – V. 10. – № 1. – pp. 83-98.
[16] *Baumgartner N., Gottesheim W., Mitsch S., Retschitzegger W., Schwinger W*. BeAware!—situation awareness, the ontology-driven way. Data & Knowledge Engineering. – Elsevier, 2010; 69(11): 1181-1193.
[17] *Baumgartner N, Mitschb S, Muller A, Retschitzegger W, Salfinger A, Schwinger W*. A tour of BeAware–A situation awareness framework for control centers. Information Fusion. – 2014. – V. 20. – pp. 155-173.
[18] RoboCup Soccer Server for Soccer Server Version 7.07. Users Manual / Mao Chen, Klaus Dorer, Ehsan Foroughi, Fredrik Heintz, ZhanXiang Huang, Spiros Kapetanakis, Kostas Kostiadis, Johan Kummeneje, Jan Murray, Itsuki Noda, Oliver Obst, Pat Riley, Timo Steffens, Yi Wang and Xiang Yin. February 11, 2003. – 140 p. http://www.robocup.org/robocup-soccer/simulation/ (Valid on 26.02.2016).
[19] *Pan J. Z., Staab S., Abmann U., Ebert J., Zhao Y*. Ontology-driven software development. – Springer-Verlag, 2013. – 344 p.
[20] Robocup Soccer Server. User manual.
Source: ⟨https://sourceforge.net/projects/sserver/files/rcssmanual/9-20030211/manual-20030211.pdf/download⟩.
[21] OWL 2 Web Ontology Language Primer (Second edition). Source: ⟨https://www.w3.org/TR/owl2-primer⟩.
[22] SPIN – Overview and Motivation. Source: ⟨https://www.w3.org/Submission/spin-overview⟩.
[23] SPARQL 1.1 Query Language. Source: ⟨https://www.w3.org/TR/sparql11-query⟩.
[24] TopBraid Composer. Getting Started Guide.
Source: ⟨http://www.topquadrant.com/resources/products/docs/TBC-Getting-Started-Guide.pdf⟩.
[25] CodeModel project. Source: ⟨https://codemodel.java.net⟩.
[26] The TopBraid SPIN AP. Source: ⟨http://topbraid.org/spin/api⟩.
[27] Apache Jena. Source: ⟨URL:https://jena.apache.org⟩.

Сведения об авторах

Пантелеев Михаил Георгиевич, 1960 г. рождения. Окончил Пензенский политехнический институт в 1982 г. Кандидат техн. наук (1988), доцент кафедры вычислительной техники СПбГЭТУ «ЛЭТИ». В списке научных трудов более 100 работ в области искусственного интеллекта, многоагентных систем и технологий семантического Web.
Panteleyev Michail Georgievich (b. 1960) graduated from Penza Polytechnical Institute in 1982. Ph.D (1988), Associate Professor, Saint Petersburg Electrotechnical University "LETI" (Department of Computer Science and Engineering). He is co-author of more than 100 publications in the field of artificial intelligence, multi-agent systems and semantic web technologies.

Лебедев Сергей Вячеславович, 1984 г. рождения. Окончил Санкт-Петербургский государственный электротехнический университет «ЛЭТИ» им. В.И. Ульянова (Ленина) (СПбГЭТУ ЛЭТИ) в 2010 году. Ассистент кафедры Вычислительной техники СПбГЭТУ ЛЭТИ. В списке научных трудов более 10 работ в области многоагентных систем и технологий семантического Web.
Lebedev Sergey Vyacheslavovich (b. 1984) graduated from Saint Petersburg Electrotechnical University "LETI" in 2010. He is an assistant at Saint Petersburg Electrotechnical University "LETI" (Department of Computer Science and Engineering). He is co-author of more than 10 publications in the field of multi-agent systems and semantic web technologies.