УДК 004.8

Подход к построению архитектуры автономного интеллектуального агента для симулятора RoboCup

Суляев Роман Сергеевич
Санкт-Петербургский государственный электротехнический университет «ЛЭТИ» им. В.И. Ульянова (Ленина)
Студент

Аннотация
Статья посвящена описанию серверного компонента (rcssserver) симулятора виртуального футбола, применяемого в международном соревновании среди роботов – «RoboCup». Рассматривается подход к построению архитектуры агента в рамках данного компонента, учитывая принципы проектирования и модель поведения исходя из выбранной предметной области. Результатом является базовая архитектура агента, позволяющая при дальнейшей разработке, получить управляемую группу роботов, используемую в симуляторе RoboCup.
Ключевые слова: RoboCup, rcssserver, архитектура агентов

The approach to building the architecture of the stand-alone intelligent agent for the RoboCup simulator

Sulyaev Roman Sergeevich
Saint-Petersburg Electrotechnical University "LETI"
Student

Abstract
The article is devoted to the description of the server component (rcssserver) of the simulator of virtual football used in the international competition among robots – «RoboCup». An approach to constructing an agent architecture within the framework of this component is considered, taking into account the design principles and the behavior model based on the chosen subject area. The result is the basic architecture of the agent, allowing for further development, to obtain a managed robotic group used in the RoboCup simulator.
Keywords: RoboCup, rcssserver, architecture of agents

Задача управления группой роботов заключается в нахождении и реализации таких действий отдельного робота, которые бы приводили к оптимальному достижению групповой цели, определенной некоторыми критериями. Скоординированные и согласованные действия такой группы могут оказать значительное влияние на эффективность выполняемых процессов. Актуальность обуславливается появлением сложноустроенных систем, состоящих из различных узлов и компонентов. Такие системы можно

представить как объект управления группой. Одной из сфер применения групповой стратегии управления в условиях организованного противодействия является виртуальный футбол. Данная игра представляет собой наглядную модель, которая позволяет исследовать возможности различных алгоритмов управления противоборствующими группами роботов.

В настоящее время интерес в мире к рассматриваемой задаче стремительно возрастает, что подтверждается большим числом участников соревнований по виртуальному футболу среди роботов, проводимых в рамках чемпионатов RoboCup [1]. Однако разрабатываемые проекты в большинстве своем имеют закрытый, либо частично открытый исходный код, повышая, таким образом, порог вхождения и замедляя темпы развития подобных соревнований. Исходя из этого, целью данной статьи является знакомство с основными компонентами симулятора RoboCup и проектирование на их основе модели агента (игрока).

Футбольную составляющую соревнований RoboCup можно разделить на следующие категории (лиги):

1. Simulation League – состязание компьютерных программ.
2. Small Size League – состязание роботов футболистов диаметром не более 18см.
3. Middle Size League – состязание роботов футболистов диаметром не более 50см.
4. Standard Platform League – состязание роботов футболистов одинаковых для обеих команд.
5. Humanoid League – дизайн и программное обеспечение роботов создаётся командами-участниками самостоятельно.

Стоит отметить, что роботы, принимающие участие в подобных соревнованиях, как правильно, являются дорогостоящими, а их техническая реализация и подготовка не имеет отношения к задаче группового управления. Поэтому выбор сделан в пользу *Simulation League*, позволяющей сконцентрироваться на разработке архитектуры агентов и алгоритма взаимодействия между собой, а также с агентами команды–соперника.

RoboCup Simulation League

Simulation League имеет следующие для общепринятые правила:

1. Прямоугольное игровое поле, размерами $L \times W$, где L, W длина и ширина поля соответственно.
2. В игре участвуют две команды роботов-агентов, по N игроков с каждой стороны: $B = \{B_i\}, C = \{C_i\}$, где $i = \overline{1, N}$. Обычно количество агентов одной команды ограничивается числом 12, в которое входят: 11 полевых игроков и 1 тренер.
3. Каждый объект на игровом поле имеет ряд параметров 3–х типов:
   3.1. постоянные, к ним относятся масса объекта m, радиус r, максимальная скорость v_{max} и другие;

3.2. переменные: координаты (x, y), скорость v, ее направление φ;

3.3. параметры управления для роботов-игроков: линейное ускорение $\vec{a}$, угловая скорость ω (считается, что ее вектор ортогонален плоскости поля, а значения угловой скорости могут изменяться мгновенно)[1].

Для организации игрового процесса в Simulation League используются следующие компоненты:

1. Сервер (rcssserver) – основный компонент симуляции матча, взаимодействие с агентами (игроками) происходит через UDP/IP (протокол пользовательских датаграмм). С его помощью каждый игрок отправляет запрос для регистрации своих действий, сервер, получая информацию, обеспечивает робота−агента необходимыми данными, такими, как: местоположение и расстояние видимых объектов, физические кондиции и другое. Сервер позволяет общаться игрокам между собой и тренером, а также получать сообщения от судьи. Следует понимать, что сервер работает с дискретными временными интервалами (циклами). Таким образом, низкая производительность игрока, обеспечивающая несвоевременный обмен данными с сервером в заданный цикл, влияет на производительность команды в целом [2]. Структура компонента представлена на рисунке 1.
2. Монитор (rcssmonitor) – компонент визуализации, позволяющий помимо просмотра матча, управлять игровым процессом: останавливать, возобновлять матч, выдавать дисциплинарные наказания для нарушивших правила в виде карточек.
3. Видеоплеер (rcsslogplayer) представляющий собой видеоплеер с возможностью воспроизводить уже проведенные футбольные матчи. Компонент необходим для анализа, выявления сильных и слабых сторон команды.

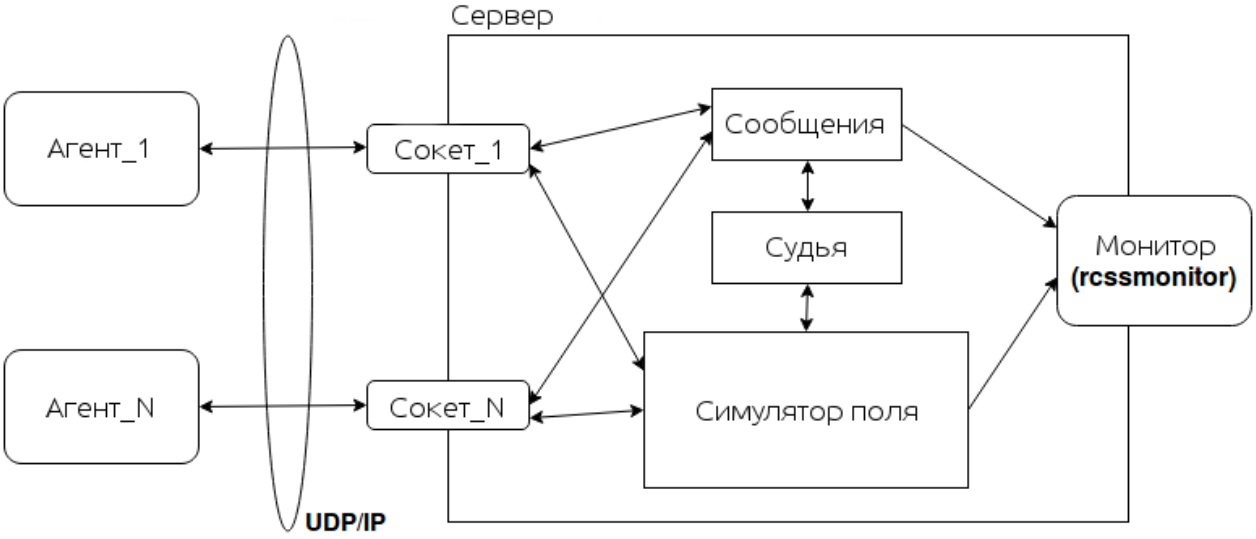

Рисунок 1 – Структура компонента rcssserver

Агент представляет собой виртуального робота и имеет три группы датчиков, позволяющих получать ясное представление об окружающей среде:

1. Звуковой датчик (*Aural Sensor Model*) позволяет принимать команды от партнеров по команде, тренера, рефери.
2. Визуальный датчик (*Vision Sensor Model*) необходим для получения информации, связанной с расстоянием и направлением видимых объектов на игровом поле относительно выбранного агента.
3. Физический датчик (*Body Sensor Model*) определяет основные физические параметры: выносливость, угол поворота головы, вектор направления скорости и другое. Дополнительно датчик показывает количество выполненных активных действий: нанесенных ударов, пойманных мячей, совершенных рывков и т.д.

Сервер с определенной в конфигурации периодичностью передает информацию агенту, которую он должен правильно интерпретировать, что позволит игроку адекватно оценить ситуацию на поле. Помимо прослушивания агент может отправлять сообщения на сервер. Это может говорить о подключении к серверу: (init team_name [version] [goalie]) или совершенному активному действию: (kick power direction), (dash power), (catch direction) и т.д.

Преимуществом компонента rcssserver над универсальными средами моделирования является ориентированность на симуляцию футбольного матча, а именно добавление погрешности в некоторые основополагающие модели действий [3].

Архитектура агента

Под архитектурой агента, в рамках выбранной предметной области, понимается совокупность классов и методов, позволяющая управлять группой роботов, выполняя поставленную задачу. Для реализации глобальной цели (забить мячей больше, чем пропустить) необходимо понять, с чем придется взаимодействовать во время матча, на рисунке 2 представлена UML диаграмма моделируемых сервером объектов.

Из диаграммы можно сделать вывод, что все объекты: игроки, мяч, линии поля, координатные флаги имеют направление и дистанцию относительно выбранного агента. Выделяют 2 категории объектов:

1. Неподвижные объекты, к которым относятся координатные флаги, выставленные по периметру игрового поля и имеющие свои абсолютные координаты. Применяются, в основном, для определения положения игрока и выбора направления движения.
2. Мобильные объекты: мяч и игроки, имеющие скорость и характеристики, изменяющиеся со временем, такие как направление и расстояние относительно текущего игрока.

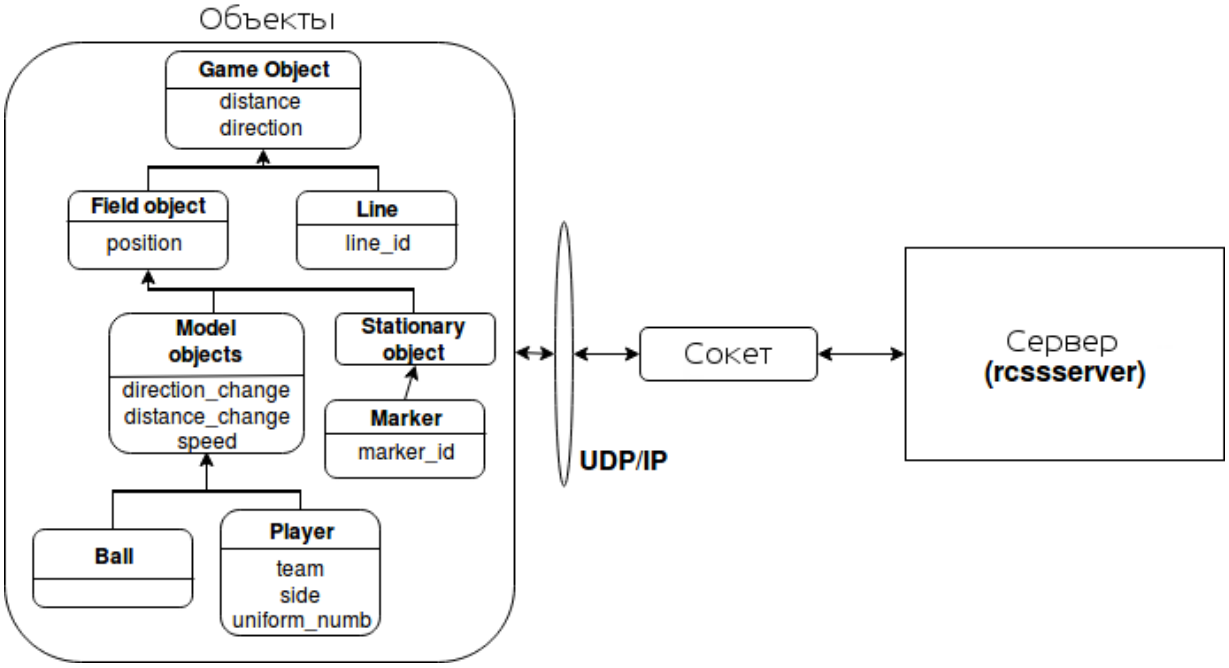

Рисунок 2 – Моделируемые объекты во время матча

Для взаимодействия с сервером каждый агент должен подключиться к серверу по протоколу UDP/IP (порт 6000) и передать инициализирующее сообщение. После чего агенту будет выдан уникальный, в рамках сессии, порт, позволяющий взаимодействовать с сервером: получать и передавать сообщения. Стоит отметить, что архитектура разрабатывается для отдельного агента, но никак не для команды – для добавления на сервер 11 игроков необходимо продублировать такое же количество подключений к серверу. Поэтому для разработки архитектуры агента в этой статье используется принцип объектно–ориентированного программирования (ООП). Подобная организация кода удобна тем, что позволяет гибко задавать роли игрокам, наследуясь от базового класса.

Описанное выше подключение реализовано в классе **Socket**, экземпляр которого передается в класс **Agent** (базовый для всех игроков). После успешной инициализации сервером агент сможет принимать сообщения от сервера, выделенные в три категории, представленные ранее. Пример такого потока сообщений представлен на рисунке 3.

```
(sense_body 0 (view_mode high normal) (stamina 8000 1) (speed 0 0) (head_angle 0) (kick 0)
(dash 0) (turn 0) (say 0) (turn_neck 0) (catch 0) (move 1) (change_view 0) (arm (movable 0)
 (expires 0) (target 0 0) (count 0)) (focus (target none) (count 0)) (tackle (expires 0) (c
ount 0)))
(see 0 ((f c) 49.9 0) ((f c t) 60.3 -34) ((f c b) 60.3 34) ((f r t) 107.8 -18) ((f r b) 107
.8 18) ((f g r b) 102.5 4) ((g r) 102.5 0) ((f g r t) 102.5 -4) ((G) 2.5 180) ((f p r b) 88
.2 13) ((f p r c) 85.6 0) ((f p r t) 88.2 -13) ((f p l c) 14 0 0 0) ((f t 0) 63.4 -38) ((f
t r 10) 71.5 -33) ((f t r 20) 79.8 -29) ((f t r 30) 89.1 -26) ((f t r 40) 98.5 -23) ((f t r
 50) 107.8 -21) ((f t l 10) 55.7 -44) ((f b 0) 63.4 38) ((f b r 10) 71.5 33) ((f b r 20) 79
.8 29) ((f b r 30) 89.1 26) ((f b r 40) 98.5 23) ((f b r 50) 107.8 21) ((f b l 10) 55.7 44)
 ((f r 0) 107.8 0) ((f r t 10) 107.8 -5) ((f r t 20) 108.9 -11) ((f r t 30) 111.1 -16) ((f
r b 10) 107.8 5) ((f r b 20) 108.9 11) ((f r b 30) 111.1 16) ((b) 49.4 0) ((l r) 102.5 90))
```

Рисунок 3 – Принимаемые сообщения от сервера

Следующей возможностью класса Agent является обработка потоков сообщений через экземпляр класса **MessageHandler**. Подготовленная информация хранится в свойствах класса **Playground** – основного класса для всех математических расчетов (нахождение абсолютных координат игрока, описание поведения в определенной ситуации, задание модели для удара, рывка и т.д.). Через экземпляр Playground в классе Agent описывается нестандартное поведение агентов в различных игровых режимах, принимаются решения относительно ударов, паса, дриблинга, длинны передач, темпа матча, направления нападения и защиты. Также класс Agent, используя Playground, позволяет отправлять сообщения определенного типа серверу. В свою очередь, Playground для этого использует экземпляр класса **ActionHandler**, в котором описаны типы сообщений, зависящие от ситуации на игровом поле. Также классу ActionHandler необходим метод, позволяющий выбирать первое сообщение определенного типа из очереди, готовящееся к отправке. Такое ограничение предусмотрено сервером: сообщения активных действий (catch, kick, dash, move, turn, turn_neck) можно отправлять только раз в цикл.

В целом, Agent описывает общее (базовое) поведение агентов и запускает методы ролевых классов (AgentGK, AgentDF, AgentMD, AgentST) – вратаря, защитника, полузащитника, нападающего соответственно.

Также реализованы классы, хранящие неизменяющуюся информацию. Примерами могут служить **PlayModes** (игровые режимы), **RefereeMessages** (типы сообщений судьи), **FieldObjects** (объекты игрового поля). Они используются основными классами для построений тактического схемы или проверки типа сообщений.

В качестве основного языка разработки использовался мультипарадигменный язык программирования JavaScript [4]. Для взаимодействия с сервером применялась популярная программная платформа Node.js, а также библиотека TypeScript – для реализации основных классов и методов агента. Пример инициализации агентов приведен на рисунке 4.

Рисунок 4 – Инициализация агентов

Сравнение команд–участников

Для сравнения были выбраны команды–победители, участвующие в RoboCup Simulation League. Ограничение состоит в неполном открытом доступе к ресурсам команд. Ниже приведен список команд, соответствующих заявленным критериям:

1. UvA TriLearn [5] – команда из Амстердама, победитель RoboCup Simulation League 2003. Исходный код открыт частично, отсутствует командный модуль принятия решений. Присутствуют механизмы взаимодействия с сервером и построения модели агента. Проект реализован на языке программирования C++ [6].
2. Brainstormers [7] – проект из университета Osnabrück, Германия. Победитель RoboCup Simulation League 2005. Как и в случае с предыдущей командой разработчики убрали из открытого доступа модуль принятий решений, оставив модули для одиночных агентов. Проект реализован на языке программирования C++ [6].
3. CM United [8] – команда–победитель RoboCup 1999, США.

В качестве критериев сравнения выбраны следующие:

- Полноценное использование возможностей rcssserver – сервер реализует множество методов для отслеживания характеристик агентов, помогая тем самым получить адекватную оценку действиям игрока на поле. Полноценное использование возможностей сервера позволяет получить более отчетливое представление об агенте и команде в целом.
- Гибкие настройки агентов – данный критерий позволяет изменять параметры компонентов из терминала запуска или графического интерфейса, не анализируя код при этом. Гибкость достигается за счет возможности подобной настройки во время симуляции игрового процесса.
- Двусторонняя симуляция агентов – использование аналогичных роботов–агентов при реализации двух противоборствующих команд. Такая возможность позволит правильно локализовать и устранить уязвимости команды, для которой генерируется однотипный соперник. Подходит для тренировки и оттачивания алгоритма и архитектуры агентов.

Во всех командах отсутствует возможность гибкой настройки агентов, что ограничивает пользователей, использующих подобную архитектуру во время разработки и тестирования. Стоит также отметить, что команда Brainstormers реализована исключительно под семейство Linux-подобных систем, такой принцип ведет к ограничению мультиплатформенности. Но, несмотря на это, команда Brainstormers лучше других соответствует критериям, предъявляемым к разрабатываемому компоненту в данной статье. Результаты наглядно представлены в таблице 1.

Таблица 1 – Сравнительная таблица команд–участников

Критерий	UvA TriLearn	Brainstormers	CM United
Полноценное использование rcssserver	+	+	-
Гибкие настройки агентов	-	-	-
Двусторонняя симуляция агентов	-	+	-

Заключение

В результате данной работы была спроектирована и реализована архитектура агента, которая позволяет взаимодействовать с сервером для решения задачи управления группой роботов. Для достижения цели были рассмотрены основные компоненты соревнования RoboCup. Дальнейшее развитие данной задачи предполагает реализацию интеллектуальной модели поведения агентов на основе конечных автоматов, которые будут использовать архитектуру, представленную в статье.

Библиографический список

1. Каляев И.А., Гайдук А.Р., Капустян С.Г. Модели и алгоритмы коллективного управления в группах роботов, М.: ФИЗМАТЛИТ 2009, 280 с.
2. Chen M., Dorer K. RoboCup Soccer Server, 2003. 150 с.
3. Беляев С.А., Матросов В.В. Опыт создания среды имитационного моделирования // III Всероссийская научно-практическая конференция «Теоретические и прикладные проблемы развития и совершенствования автоматизированных систем управления военного назначения» (22.11.2017 г., СПб). Сборник тезисов. СПб.: Военно-космическая академия имени А.Ф. Можайского. 2017. С.232.
4. Беляев С.А. Разработка игр на языке JavaScript. Учебное пособие. СПб.: Изд-во Лань, 2016. 128 с.
5. UvA TriLearn // jellekok.nl. URL: http://www.jellekok.nl/index.php?cat=robocup (дата обращения 12.12.2017)
6. Bradfield P. Creating agents for the RoboCup Soccer Simulator using evolutionary techniques, 2007, 70 с.
7. Brainstormers // sourceforge.net/projects/bsrelease. URL: https://sourceforge.net/projects/bsrelease/files/ (дата обращения 13.12.2017)
8. CM United. URL: http://www.cs.cmu.edu/~mmv/papers/LNAI97-robot.pdf (дата обращения 12.12.2017)