ТЕХНОЛОГИЯ РАЗРАБОТКИ ПРИКЛАДНЫХ МНОГОАГЕНТНЫХ СИСТЕМ В ИНСТРУМЕНТАЛЬНОЙ СРЕДЕ MASDK

В. И. Городецкий[1], О. В. Карсаев[2], В. Г. Конюший[3], В. В. Самойлов[4], Е. В. Маньков[5], А. В. Малышев[6]

Санкт-Петербургский институт информатики и автоматизации РАН

СПИИРАН, 14-я линия ВО, д. 39, Санкт-Петербург, 199178

[1]<gor@iias.spb.su>, [2]<ok@iias.spb.su>, [3]<kvg@iias.spb.su>, [4]<samovl@iias.spb.su>, [5]<eman@iias.spb.su>, [6]<A.Malyshev@iias.spb.su>

УДК 681.3

Городецкий В. И., Карсаев О. В., Конюший В. Г., Самойлов В. В., Маньков Е. В., Малышев А.В. **Технология разработки прикладных многоагентных систем в инструментальной среде MASDK** // Труды СПИИРАН. Вып. 3, т. 1. — СПб.: Наука, 2006.

Аннотация. *Приводится описание инструментальной среды, которая предназначена для поддержки технологии разработки прикладных многоагентных систем, построенной на основе методологии Gaia. Инструментальная среда реализует графический стиль разработки и поддерживает полный цикл создания прикладных многоагентных систем.* — Библ. 26 назв.

UDC 681.3

Gorodetsky V. I., Karsaev O. V., Konyushiy V. G., Samoylov V. V., Mankov E. V., Malyshev A. V. **Technology of multi agent system development in MASDK environment** // SPIIRAS Proceedings. Issue 3, vol. 1. — SPb.: Nauka, 2006.

Abstract. *The paper presents a software development environment intended for support of the multi-agent technology based on Gaia methodology. The aforementioned environment implements graphical development style and support the whole life cycle of multi-agent applications development.* — Bibl. 26 items.

1. Введение

Многоагентные системы в настоящее время рассматриваются в числе наиболее перспективных подходов к разработке сложных интеллектуальных распределенных информационных систем. Примерами проблемных областей, в которых использование многоагентных систем рассматривается наиболее перспективным, являются логистика, транспортные и телекоммуникационные сети, мониторинг бизнес-процессов, мониторинг и прогнозирование состояния окружающей среды, ликвидация последствий природных и техногенных катастроф, электронная коммерция и т.д.

В результате наблюдается быстро растущий интерес к исследованиям в области многоагентных систем как с научной, так и с практической стороны. Однако, несмотря на большие силы, вовлеченные в последнее время в исследования по многоагентным системам (МАС), технология разработки этих систем еще не достигла такого уровня зрелости, который позволил бы рассматривать многоагентные технологии в качестве стандартного подхода для разработки индустриальных приложений. Одна из основных причин такого уровня развития МАС состоит в том, что в настоящее время пока еще не существует достаточно развитых инструментальных средств, применимых для разработки прикладных МАС в широком спектре предметных областей.

Текущий этап развития многоагентных технологий характеризуется наличием большого разнообразия создаваемых и развиваемых технологических

решений (формальных языков описания многоагентных систем, средств моделирования многоагентных систем, и т.д.), а также инструментальных сред. В частности, достаточно полный список работ в данном направлении можно найти в [25, 26]. В числе наиболее известных и популярных к настоящему времени инструментальных сред рассматриваются такие разработки, как AgentBuilder [21], Jack [19], JADE [4], ZEUS [10], FIPA-OS [14], agentTool [12], и некоторые другие. Основной целью инструментальных сред в отличие от отдельных технологических решений является обеспечение полного цикл разработки прикладных систем, начиная с этапа анализа предметной области, включая этапы проектирования, непосредственно разработки, верификации и заканчивая этапами развертывания и сопровождения.

В качестве основы для разработки таких инструментальных сред, как правило, рассматривается технология, разработанная в рамках объектно-ориентированного подхода. До настоящего времени эта технология «де-факто» рассматривается в качестве стандартного подхода для разработки индустриального программного обеспечения [7]. В основе этой технологии в качестве формального языка для спецификации объектно-ориентированных программ используется язык UML (unified modeling language). В настоящее время предпринимаются достаточно активные усилия по развитию этого языка применительно для спецификации многоагентных систем. В частности, одним из наиболее известных в этом направлении проектов является проект по разработке языка Agent UML ([1, 3]). Этот проект выполняется совместно организациями FIPA и OMG, являющимися общепризнанными лидерами в области исследования многоагентных технологий (FIPA) и в области объектно-ориентрованных технологий (OMG). Другим примером в этом направлении может служить язык AML [2], разрабатываемый компанией Whitestein. В связи с тем что агентно-ориентированный подход в определенной степени рассматривается в качестве развития объектно-ориентированного подхода, существует достаточно много работ, рассматривающих специфику каждого из этих подходов и приводящих их сравнительный анализ. В частности, достаточно подробное описание специфических особенностей, присущих разработке многоагентных систем, можно найти в работах ([23, 24]). Анализ различий между подходами можно найти в работе [22].

Наряду с разработкой инструментальных сред и отдельных технологических решений активные исследования ведутся в направлении развития различных методологий для разработки прикладных многоагентных систем. К числу наиболее популярных методологий можно отнести следующие методологии: Gaia [22], MESSAGE [8], MaSE [13], Prometheus [20], Adelfe [5], Tropos [15], PASSI [9]. Методологии, с одной стороны, обобщают наиболее удачные концептуальные и проектные решения, достигнутые в процессе разработки тех или иных инструментальных сред и/или прикладных многоагентных систем, и с другой стороны, рассматриваются в качестве основы для дальнейшего развития существующих и разработки новых инструментальных сред. Сравнительный анализ использования указанных выше и других методологий в различных инструментальных средах можно найти в работах [6, 11].

В качестве одной из наиболее важных задач, которая рассматривается во всех перечисленных выше методологиях, является задача разработки конструктивных взаимосвязей между решениями, достигаемыми на различных этапах разработки приложений. В этом отношении результаты, достигнутые в объектно-ориентированном подходе, в определенном смысле опять-таки могут

Труды СПИИРАН. 2006. Вып. 3. Т. 1. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2006. Issue 3. V. 1. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

служить основой для сравнения и развития подходящих решений с учетом специфики многоагентных систем. В частности, в методологии Gaia [22] рассматриваются два этапа: этап анализа предметной области и этап проектирования прикладной системы. При этом целью этапа анализа является достижение понимания системы и ее структуры без описания каких-либо деталей разработки. Этот этап предполагает разработку абстрактных решений и понятий, связанных с описанием организации системы, в том числе: выявление и описание задач, решаемых системой, описание ролей и их взаимодействия. Целью этапа проектирования является трансформация абстрактных моделей, описанных на этапе анализа, в модели более низкого уровня абстракции, которые затрагивают уже описание деталей разработки. В частности, на этом этапе выполняется описание моделей агентов и моделей сервисов.

В данной статье приводится описание инструментальной среды MASDK (Multi Agent System Development Kit), предназначенной для разработки прикладных многоагентных систем, которая является объектом исследований и развивается в Санкт-Петербургском институте информатики уже в течение шести лет. Первые две версии этой среды [16] использовались на практике для разработки прототипов прикладных систем в различных предметных областях [17, 18] и для исследований основных возможностей многоагентной технологии. Опыт, накопленный в процессе исследований, а также анализ современных тенденций и результатов, как в области технологий, так и в области методологий, послужили основой для формирования требований к разработке третий версии данной среды. В частности, в качестве основы для разработки новой версии инструментальной среды используется методология Gaia [22], а также ряд положений, рассматриваемых в проекте разработки языка AUML [1].

Основное внимание в статье уделяется описанию технологии разработки прикладных многоагентных систем в среде MASDK. При этом статья организована следующим образом. Во втором пункте приводится общее описание среды и архитектуры агентов, генерируемых с ее помощью. Общее описание технологии, состоящей из десяти этапов, составляет содержание третьего пункта. Все последующие пункты статьи содержат детальное описание каждого из десяти этапов технологии.

2. Общее описание среды и архитектуры генерируемых агентов

В состав инструментальной среды MASDK входит следующий набор компонент (рис. 1).

1) *Описание проекта МАС* выполняется на специально разработанном языке AFW (Agent Framework), являющимся подмножеством XML. Данный язык играет ту же роль, что и язык UML в объектно-ориентированном подходе.

2) *Интегрированная система графических редакторов* диаграмм. С их помощью выполняется описание компонент разрабатываемой МАС. В качестве прообраза данной компоненты среды MASDK можно рассматривать систему графических редакторов над языком UML. Например, систему редакторов, разработанных в среде Rational Rose.

3) Библиотека классов, реализующая инвариантную метамодель агентов и именуемая далее как *основа агента*.

Труды СПИИРАН. 2006. Вып. 3. Т. 1. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2006. Issue 3. V. 1. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

4) *Генератор агентов*, выполняющий автоматическую генерацию исходного и исполняемого кодов программных агентов на основе описания прикладной системы на языке AFW. Настоящая версия среды MASDK выполняет генерацию программного кода на языке C++.

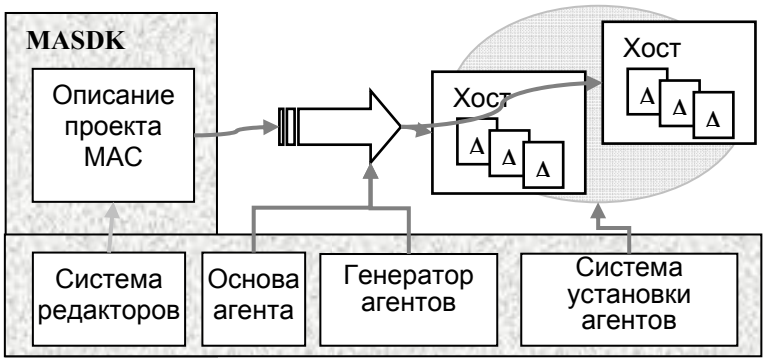

Рис. 1. Общее описание среды MASDK.

5) *Система установки агентов*, обеспечивающая развертывание агентов в сети. В состав этой системы входит компонента, именуемая *порталом*. Эта компонента устанавливается на компьютеры сети, на которых предполагается развертывание агентов прикладной системы, и используется как на этапе установки агентов, так и в процессе их функционирования в качестве образования коммуникационной платформы для агентов.

Обобщенная структура агентов, разрабатываемых с помощью среды, представлена на рис. 2. Она состоит из следующих компонент. *Основа агента* является готовой компонентой, входящей в состав среды (рис. 1), и используется для генерации всех прикладных агентов. Основа агента играет роль «движка» агента и реализует необходимый набор инвариантных, проблемно-независимых функций. К их числу относятся такие функции, как: а) запуск и остановка агента, б) отправка и получение сообщений, в) запуск и выполнение сервисов, включая контроль прерываний при выполнении сервисов, г) обеспечение доступа к базе данных агента и внешним компонентам, а также другие функции инвариантного типа. Компоненты *Модель поведения агента*, *Ментальная модель агента* и *Сервисы* содержат проблемно-ориентированное описание агента, которое выполняется на языке AFW. При генерации программных агентов исходное описание этих компонент на языке AFW преобразуется в программный код. Сервисы описывают функциональность, которой должны обладать агенты для исполнения соответствующих им ролей в составе МАС. Описание сервисов выполняется в виде машин состояний. Ментальная модель описывает данные и знания, которые используются при выполнении сервисов и которыми обмениваются агенты в процессе взаимодействия. Модель поведения агента описывает сценарии исполнения сервисов и события, инициирующие запуск сервисов, а также выполнение действий агентов во внешней среде. В частности, модель поведения описывает связи между сервисами агента и коммуникационными актами, с помощью которых описываются схемы взаимодействия агентов. *Сервисные функции* являются описанием атомарных сценариев действий агентов, исполняемых в отдельных состояниях машин состояний при выполнении соответствующего

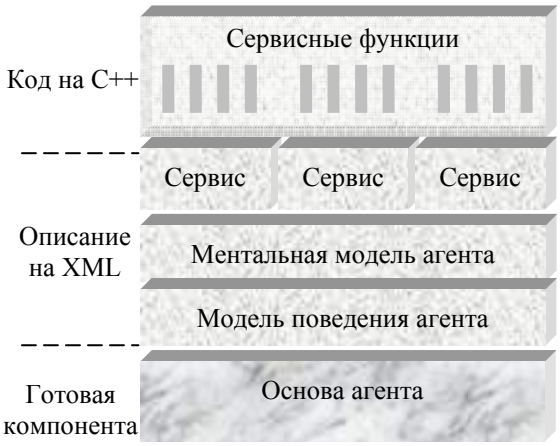

Рис. 2. Структура агента.

Труды СПИИРАН. 2006. Вып. 3. Т. 1. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2006. Issue 3. V. 1. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

сервиса. Описание сервисных функций выполняется на объектно-ориентированном языке программирования, в текущей версии на языке C++. Функционирование агента может предусматривать использование внешних компонент. В связи с этим сервисные функции, в частности, описывают необходимые варианты использования агентом внешних компонент. При этом агент, реализуя сервисные функции, может инициировать работу внешних компонент, получать результаты работы внешних компонент в виде данных, хранить их в своей ментальной модели, использовать для принятия решений и обмениваться ими с другими агентами.

В связи с описанием агентов в среде MASDK используются понятия классов агентов. Агенты одного класса имеют общее функциональное описание в рамках структуры, отображенной на рис. 2, а именно — характеризуются общей *Моделью поведения агентов*, обладают общим набором *Сервисов* и *Сервисных функций* и используют одни и те же внешние компоненты. Агенты каждого класса различаются только содержанием данных и знаний, хранящихся в их *Ментальных моделях*.

3. Общее описание технологии разработки прикладных МАС в инструментальной среде

Методология разработки прикладных МАС в среде MASDK (рис. 3) состоит из десяти этапов, выполняемых в определенной последовательности. Последовательность выполнения этапов определяется взаимозависимостью результатов. Решения, описанные на одних этапах, являются исходными данными для выполнения последующих этапов. На более высоком уровне методология может быть описана в виде пяти стадий. Далее в этом параграфе приводится общее описание методологии разработки прикладных МАС в среде MASDK на уровне стадий. Более детальное описание каждого из этапов составляет содержание последующих пунктов статьи.

На первой стадии выполняется проектирование прикладной МАС. Достижение данной цели предполагает выполнение двух этапов: анализа предметной области и описания онтологии проблемной области. На этапе анализа предметной области выполняется выявление и описание задач, ролей, распределение задач по ролям, описываются модели взаимодействия ролей. На этапе разработки онтологии проблемной области выполняется описание понятий и отношений между ними. При этом в процессе разработки онтологии может рассматриваться решение двух задач. Первая задача состоит в описании тех понятий предметной области, которые должны составлять основу взаимодействия агентов. Эти понятия являются элементами сообщений, которыми агенты будут обмениваться в процессе переговоров. Вторая задача также в определенной степени может рассматриваться как анализ предметной области, но на уровне формализации понятий. В отличие от этого в рамках этапа анализа предметной области эта задача рассматривается с точки зрения выявления и концептуального описания функциональности разрабатываемой МАС.

Конечной задачей этапа анализа предметной области является задача идентификации классов агентов и сопоставления им ролей. Распределение ролей между классами агентов «автоматически» влечет сопоставление классам агентов задач, решение которых они должны будут обеспечивать в процессе функционирования прикладной МАС.

Труды СПИИРАН. 2006. Вып. 3. Т. 1. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2006. Issue 3. V. 1. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

Решения, разработанные на первой стадии, являются исходными данными для второй стадии — стадии проектирования идентифицированных классов агентов. Содержанием задач, решаемых на трех этапах второй стадии, является описание соответствующих трех компонент, образующих структуру агентов (рис. 2), а именно: модели поведения агента, моделей сервисов и ментальной модели. На основании множества задач, сопоставленных классам агентов, выполняется проектирование соответствующих сервисов классов агентов. При разработке метамодели поведения класса агентов выполняется описание событий и сценариев, которые определяют порядок использование сервисов классами агентов.

Рис. 3. Методология разработки прикладной МАС в MASDK.

Описание ментальных моделей выполняется на этапе описания частных онтологий классов агентов и на второй стадии сводиться только к спецификации ее структуры. Содержательное наполнение ментальной модели выполняется на четвертой стадии при описании агентов. В связи с этим следует отметить, что агенты каждого класса имеют общую структуру ментальной модели, но обладают различным содержательным наполнением.

Проектирование классов агентов, выполняемое на второй стадии, не предполагает написания программного кода. Проектирование выполняется с помощью соответствующих графических редакторов диаграмм, в процессе использования которых формируется формальное описание классов агентов на языке AFW (рис. 1). Разработка программного кода составляет основную задачу для третьей стадии разработки и относится, в основном, только к описанию сервисных функций. Разделение между стадиями проектирования и разработки программного кода в рамках среды MASDK позволяет выполнять необходимую верификацию проекта прикладной системы еще до начала кодирования, что

Труды СПИИРАН. 2006. Вып. 3. Т. 1. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2006. Issue 3. V. 1. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

позволяет избежать ошибок в процессе разработки на более поздних этапах и тем самым сокращает трудозатраты на разработку прикладной системы в целом.

Разработка программного кода, выполняемая на третьей стадии, относится к описанию сервисных функций. Сервисы классов агентов описываются в виде машин состояний, при этом сервисные функции соответствуют состояниям машин состояний. Таким образом, содержательное описание сервисных функций, требующих написания программного кода, формируется на этапе проектирования классов агентов при описании их сервисов в виде машин состояний. При разработке сервисных функций описываются методы, которые предназначаются для таких классов задач, как: формирование исходящих сообщений, изменение состояния ментальной модели агента, реализация программного интерфейса с внешними компонентами агента, и другие. Кроме описания сервисных функций третья стадия разработки включает этап генерации программного кода классов агентов. Эта функция выполняется автоматически.

Четвертая стадия делится на два этапа. Первый этап, описание агентов, предполагает описание регистрационных данных, необходимых для их установки. Второй этап, описание начальных ментальных моделей агентов, выполняется на основании спецификаций ментальных моделей классов агентов, разработанных на второй стадии.

Развертывание агентов в сети составляет содержание единственной задачи, рассматриваемой на последней стадии разработки.

4. Проектирование прикладной МАС

4.1. Анализ предметной области

Анализ предметной области в соответствии с методологией Gaia [22] сводится к описанию организации прикладной МАС, что в частности предполагает решение следующих задач:

- описание моделей ролей,
- описание моделей взаимодействия ролей, и
- идентификацию классов агентов и сопоставление им ролей.

Описание моделей ролей предполагает выявление и идентификацию ролей. Выявление ролей основывается на анализе а) физических объектов и/или субъектов рассматриваемой проблемной области, и б) задач, решаемых этими объектами/субъектами. Идентифицированные роли описываются в верхней части редактора мета модели прикладной МАС (рис. 4). В частности, в данном примере идентифицированы и описаны две роли: *Manager* и *Executor*.

В нижней части этого же редактора описывается общая схема модели взаимодействия ролей на уровне имен протоколов. В частности, в примере на рис. 4 рассматриваются три протокола: *Acquaintance*, *Planning* и *Scheduling*. Описание протокола на этом уровне предполагает указание а) ролей — участников протокола взаимодействия и б) инициатора протокола, который выделяется с помощью треугольника. В данном примере обе роли принимают участие в каждом из трех протоколов взаимодействия, причем во всех трех протоколах инициатором является роль *Manager*.

Модели взаимодействия ролей, по сути, описывают сценарий (последовательность) решения подзадач, распределенных между ролями. В частности, в

Труды СПИИРАН. 2006. Вып. 3. Т. 1. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2006. Issue 3. V. 1. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

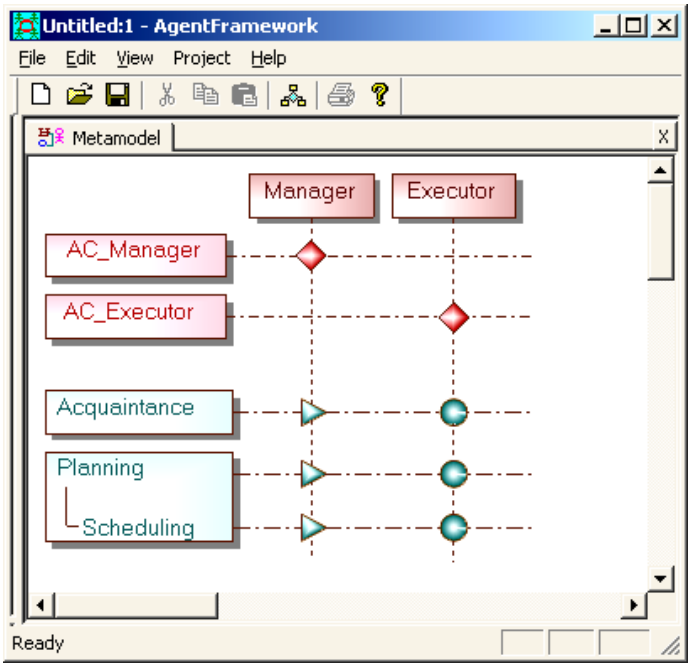

Рис. 4. Редактор организации прикладной МАС.

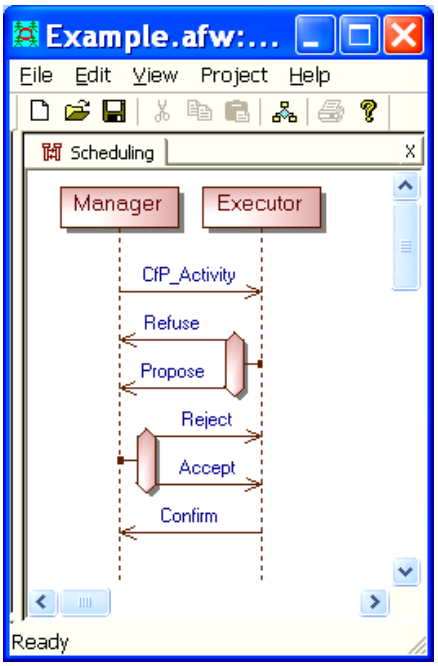

Рис. 5. Редактор протоколов.

связи с этим редактор описания мета модели прикладной МАС предоставляет возможность изменять порядок отображения множества протоколов, а также указывать вложенность протоколов. Например, протокол *Scheduling* является вложенным по отношению к протоколу *Planning*.

Более детально каждый из протоколов взаимодействия описывается с помощью редактора, представленного на рис. 5. В данном случае приведен пример детального описания протокола *Scheduling*, в основе которого используется схема протокола сети контрактов (CNP — contract net protocol). Схема протокола описывает сценарий коммуникационных актов, с помощью которых агенты обмениваются соответствующими сообщениями. При этом, событие, инициирующее выполнение подзадачи, как правило, связано с выполнением соответствующего коммуникационного акта. Например, в приведенном примере перед исполнителем роли *Manager* стоит задача назначить работу и согласовать время ее исполнения. Исполнитель роли *Manager* посылает всем агентам, играющим роль *Executor,* сообщение *CfP_Activity* с описанием требований в отношении выполнении данной работы. Исполнитель роли *Executor,* получив такое сообщение, анализирует принятые ранее обязательства и оценивает свои возможности выполнить предложенную работу. На основе этого исполнитель роли *Executor* возвращает роли *Manager* либо отказ (коммуникационный акт *Refuse*), либо предложение с описанием условий выполнения работы (коммуникационный акт *Propose*). Далее, исполнитель роли *Manager* анализирует полученные предложения, выбирает из них наиболее подходящее предложение и посылает в адрес исполнителя роли *Executor*, автора выбранного предложения, сообщение *Accept* об утверждении предложения. В адрес всех остальных участников переговоров посылается сообщение *Reject* об отклонении предложения. В последнем коммуникационном акте протокола *Confirm* исполнитель роли *Executor* извещает исполнителя роли *Manager* о принятии обязательства выполнить работу в соответствии с оговоренными условиями.

Труды СПИИРАН. 2006. Вып. 3. Т. 1. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2006. Issue 3. V. 1. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

Третья задача — задача идентификации классов агентов сводится к определению необходимого множества классов агентов и сопоставлению им ролей. В частности, в примере на рис. 4 идентифицированы два класса агентов: *AC_Manager*, которому сопоставлена роль *Manager*, и *AC_Executor,* которому сопоставлена роль *Executor.* Такой вариант идентификации классов агентов, когда для исполнения каждой роли вводится соответствующий класс агентов, является на практике наиболее типичным, но не обязательным. В частности, одному классу агентов может быть сопоставлено несколько ролей. В целом, задача идентификации классов агентов связана с оценкой «рациональности» организации МАС. Одним из основных показателей такой оценки является структура и количество коммуникационных актов разрабатываемой МАС. В связи с этим, задача идентификации классов агентов решается на основе анализа выявленных и описанных моделей ролей и предполагает анализ следующих факторов.

- Физическая распределенность субъектов/объектов предметной области в пространстве. Если роли принадлежат удаленным объектам/субъектам проблемной области, то им должны сопоставляться различные классы агентов.
- Содержание задач, соответствующих различным ролям, и совокупность методов, используемых для решения задач. Сопоставительный анализ задач и методов может определять «схожесть» и/или «различие» моделей ролей. Это может служить основанием для принятия решений при идентификации классов агентов. В частности, если две различные роли не имеют пространственной удаленности, но при этом имеют «относительно схожие» модели, то эти обстоятельства могут рассматриваться в качестве обоснования для сопоставления этих ролей одному классу агентов.

4.2. Описание онтологий проблемной области

Описание онтологии проблемной области наряду с этапом анализа проблемной области является начальным этапом разработки прикладной МАС. При этом если анализ проблемной области содержательно сводится к выявлению задач и определению функциональности прикладной МАС, то разработка онтологии соотносится с уровнем понятий и сводится к:

- описанию понятий проблемной области и
- описанию отношений между понятиями.

Описание понятий в данном случае выполняется в стиле традиционного объектно-ориентированного проектирования, что предполагает описание атрибутов для каждого из понятий с учетом отношений наследования между ними. Разработанный в среде MASDK редактор для описания понятий и отношений между ними представлен на рис. 6.

Использование онтологии понятий для разработки прикладной МАС определяет дополнительную специфику в отношении понятий и отношений между ними. В соответствии с этой спецификой все понятия онтологии могут быть разделены на два класса: понятия онтологии, которые: а) используются и б) не используются в переговорах агентов. В случае использования онтологии для разработки многоагентных систем описание понятий первого класса является обязательным. В частности, понятия онтологии, относящиеся к этому классу, используются в редакторе протоколов (рис. 5) при детальной спецификации

Труды СПИИРАН. 2006. Вып. 3. Т. 1. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2006. Issue 3. V. 1. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

коммуникационных актов. Описание понятий, относящихся ко второму классу, имеет вспомогательный характер. Эти понятия в основном используются на этапе описания частных онтологий классов агентов в качестве наследуемых классов. При этом понятия частных онтологий являются производными от понятий общей онтологии и являются их потомками.

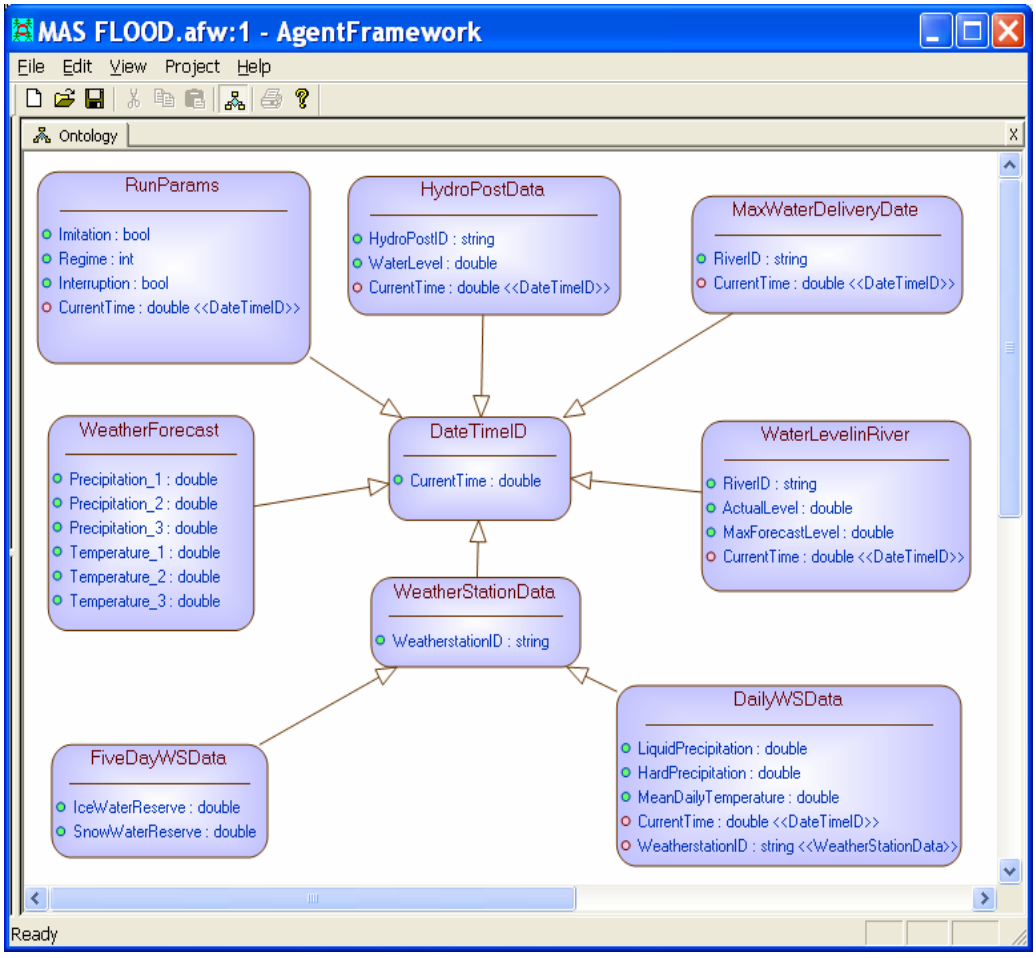

Рис. 6. Редактор онтологии понятий предметной области.

5. Проектирование классов агентов

5.1. Описание мета моделей поведения классов агентов

Описание мета моделей поведения классов агентов предполагает решение следующих подзадач:

- идентификация сервисов классов агентов,
- спецификация сценариев использования сервисов, и
- идентификация внешних компонент и описание сценариев их использования классом агентов.

Исходными данными для решения этих задач являются решения, полученные на этапе анализа предметной области, а именно: список идентифицированных классов агентов и описание схемы взаимодействия ролей. В частности, при идентификации классов агентов выполняется сопоставление им ролей. С одной стороны, это автоматически влечет сопоставление классам агентов

Труды СПИИРАН. 2006. Вып. 3. Т. 1. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2006. Issue 3. V. 1. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

списка задач, которые они должны решать в связи с назначенной им ролью. С другой стороны, это определяет для классов агентов множество протоколов взаимодействия, в которых они принимают участие. В связи с этим следует заметить, что при описании протоколов взаимодействия формируется верхний уровень описания сценария решения задач в целом. Специфика протоколов состоит в том, что с их помощью определяется та часть сценария решения задач, которая определяет взаимодействие распределенных сущностей (агентов).

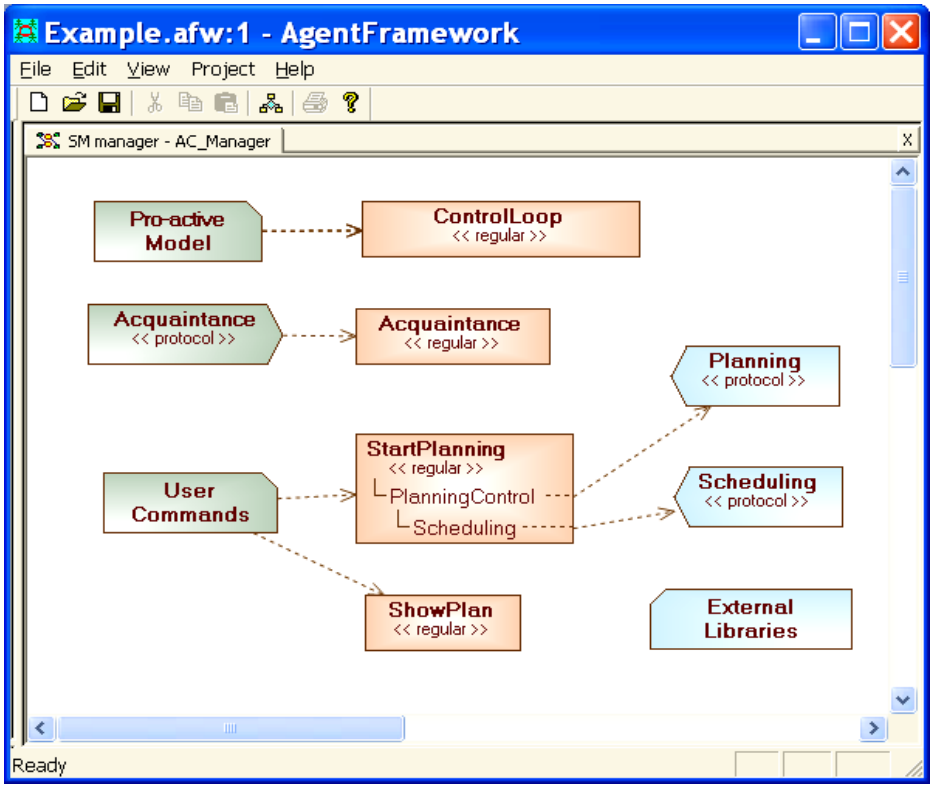

Рис. 7. Редактор метамоделей поведения классов агентов.

Решение трех перечисленных выше задач, рассматриваемых на этапе описания модели поведения класса агентов, в среде MASDK выполняется с помощью редактора, отображенного на рис. 7. В основе концепции данного редактора используются следующие абстрактные понятия.

- *Сервис* — атомарный сценарий поведения агента, в процессе исполнения которого агент может выполнять следующие классы действий:
  - Использовать и модифицировать данные ментальной модели агента.
  - Обеспечивать участие агента в протоколах взаимодействия.
- *Внешняя компонента* — внешний программный модуль, который может использоваться агентом. Внешние компоненты также могут рассматриваться как ресурсы класса агентов. Методы внешних компонент вызываются при выполнении сервисов.
- *Входное событие* — событие во внешнем мире агента, возникновение которого предполагает ту или иную реакцию агента. Классами входных событий являются:
  - *Входное сообщение* — сообщение другого агента, поступившее в адрес данного агента;

Труды СПИИРАН. 2006. Вып. 3. Т. 1. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2006. Issue 3. V. 1. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

- *Команда пользователя.* Агент может иметь интерфейс с пользователем, что, в частности, предполагает возможность поступление команд (сообщений) агенту со стороны пользователя;
- *Событие среды.* Агент может иметь интерфейс с окружающей средой, реализованный, например, с помощью сенсорных устройств.

➢ *Про-активное правило*. Поведение агента может быть инициировано не только входными событиями. В частности, поведение агентов может инициироваться, например, на основе анализа текущего состояния ментальной модели, или в определенные моменты времени. Такие события описываются с помощью про-активных правил, которые представляются в виде конструкции *When … If … Then …*

➢ *Выходное событие* — событие во внешнем мире, инициируемое агентом в результате выполнения сервисов. Классами выходных событий являются:
- *Выходное сообщение* — сообщение данного агента в адрес другого агента;
- *Сообщение пользователю* — сообщение агента пользователю, которое, в частности, может предполагать ответную реакцию со стороны пользователя;
- *Действие в среде*. Агент может иметь интерфейс с активными физическими устройствами, реализованными в окружающей среде.

Между перечисленными абстрактными понятиями могут устанавливаться направленные отношения, с помощью которых можно описывать сценарии поведения агента. Классами направленных отношений являются:

➢ *«Входное событие → Сервис»*: возникновение входного события предполагает вызов сервиса агента.

➢ *«Сервис → Сервис»*: при выполнение одного сервиса осуществляется вызов другого сервиса. Возможны два типа вызовов сервиса:
- *Синхронный вызов*. В этом случае вызываемый сервис является вложенным по отношению к исходному сервису. При таком вызове выполнение основного сервиса прерывается на время выполнения вложенного сервиса и продолжается после его завершения;
- *Асинхронный вызов*. В этом случае после момента вызова сервиса оба сервиса выполняются независимо.

➢ *«Сервис → Выходное событие»*: в результате или в процессе выполнения сервиса агент инициировал возникновение выходного события;

➢ *«Сервис → Внешняя компонента»*: в процессе выполнения сервиса осуществляется вызов метода или методов внешней компоненты.

Перечисленные абстрактные понятия в рассматриваемом редакторе отображаются с помощью следующей графической нотации (рис. 7). Множество идентифицированных сервисов представляется в центральной части редактора в виде древовидной структуры. При этом в узлах этой структуры отображаются имена идентифицированных сервисов. Связь между узлами структуры, сервисами, обозначает следующее. Сервис, отображенный на более высоком уровне в структуре, в процессе выполнения вызывает или может вызвать другой сервис. Один и тот же сервис может вызываться более чем одним другим сервисом. В соответствии с этим один и тот же сервис может быть представлен в различных узлах структуры.

Входные события и про-активные правила отображаются слева от древовидной структуры с помощью следующих типов фигур.

Труды СПИИРАН. 2006. Вып. 3. Т. 1. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2006. Issue 3. V. 1. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

- *Input message*. Фигуры данного типа обозначают протоколы взаимодействия, описанные в метамодели МАС (рис. 4), в которых агенты данного класса принимают участие и не являются инициаторами протоколов. Внутри каждой фигуры отображается названия протокола. Связь между фигурой, обозначающий протокол взаимодействия, и сервисом означает, что этот сервис поддерживает участие класса агентов в этом протоколе. В частности, все входящие и исходящие по отношению к классу агентов сообщения указанного протокола взаимодействия, обрабатываются указанным сервисом.
- *Pro-active model*, *User commands*, *Events.* Фигуры данного типа обозначают соответственно множество правил про-активного поведения, множество команд пользователя и множество событий окружающей среды, описанные в модели поведения класса агентов. Направленные связи между этими фигурами и сервисами указывают, какие сервисы инициируются при срабатывании правил про-активного поведения, какие при поступлении команд от пользователя и какие при возникновении событий в окружающей среде.

Выходные события отображаются справа от блока, описывающего сервисы, с помощью следующих типов фигур.

- *Output message*. Фигуры данного типа по содержанию аналогичны фигурам типа *Input message* с той лишь разницей, что они обозначают протоколы взаимодействия, описанные в метамодели МАС (рис. 4), в которых агенты данного класса принимают участие и являются инициаторами протоколов.
- *User interface*, *Actions*. Фигуры данного типа обозначают соответственно множество типов сообщений пользователю и множество действий в окружающей среде, описанные в модели поведения класса агентов. Направленные связи между этими фигурами и сервисами указывают, какие сервисы инициируют сообщения пользователю, а какие — действия в окружающей среде.

Рассматриваемый редактор используется для графического представления модели поведения класса агентов. Детальная спецификация всех описанных компонент выполняется с помощью группы вспомогательных редакторов следующего уровня. В частности, связь между сервисами и внешними компонентами описывается только во вспомогательном редакторе и не представляется на диаграмме.

С методологической точки зрения решение задач на этапе описания модели поведения классов агентов можно выполнять в следующей последовательности.

1. Определить набор внешних компонент, необходимых классу агентов для обеспечения решения задач в соответствии с назначенной ему ролью. Следует отметить, что при проектировании классов агентов наряду с проектированием новых внешних компонент могут также использоваться и готовые компоненты.
2. Идентифицировать множество входных и выходных событий за исключением входных и выходных сообщений. Последние рассматриваются в качестве исходных данных для данного этапа и формируются в результате выполнения этапа анализа предметной области.

Труды СПИИРАН. 2006. Вып. 3. Т. 1. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2006. Issue 3. V. 1. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

3. Определить ситуации, в которых класс агентов может инициировать проактивное поведение и описать эти ситуации с помощью правил *When … If … Then….*
4. На основании результатов, полученных на шагах 1–3 идентифицировать необходимое множество сервисов для описания функциональности класса агентов и решения задач, соответствующих классу агентов. Описать идентифицированные сервисы в редакторе с указанием направленных связей между сервисами.
5. Описать с помощью соответствующих редакторов множества входных и выходных событий, множество про-активных правил с указанием направленных отношений между элементами этих множеств и идентифицированными сервисами.

В результате выполнения перечисленных шагов в рассматриваемом редакторе (рис. 7) формируется отображение модели поведения класса агентов в графической нотации, описанной выше.

5.2. Описание сервисов классов агентов

Выполнение этого этапа предполагает описание всех сервисов, идентифицированных на этапе описания моделей поведения классов агентов. Описание сервисов выполняется в виде машин состояний. Пример представления сервиса в виде машины состояний приведен на рис. 8. Использование машин состояний для описания сервисов классов агентов главным образом обусловлено тем, что конструкции машин состояний позволяют описывать прерывания выполнения сервисов и состояния ожиданий, в которых агенты могут находиться во время выполнения протоколов взаимодействия.

Описание сервисов в виде машин состояний, по сути, является способом графического описания алгоритмов (сценариев поведения), реализуемых агентами класса при выполнении соответствующих сервисов.

Описание машин состояний сводится к описанию множества состояний и переходов между ними. Состояния машин состояний соответствуют сервисным функциям. Таким образом, описание сервисов в виде машин состояний идентифицирует необходимое множество сервисных функций. Наименования состояний являются именами сервисных функций.

Для описания сервисных функций в среде используется понятие классов сервисных функций. В частности, в текущей версии среды используется один базовый класс сервисных функций и шесть классов специфических сервисных функций, предназначенных для:

- обработки получения сообщений;
- управления состоянием, связанным с ожиданием получения сообщений;
- отправки сообщений;
- вызова сервисов;
- решения задачи классификации объектов.

Использование классов специфических сервисных функций обеспечивает достижение двух целей. Первая из них состоит в поддержке процесса проектирования сервисов классов агентов, а вторая — в описании часто используемых типовых действий агентов в виде готовых решений на уровне программного кода.

Труды СПИИРАН. 2006. Вып. 3. Т. 1. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2006. Issue 3. V. 1. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

Поддержка процесса проектирования сервисов классов агентов состоит в следующем. Решения, полученные на этапе описания моделей поведения классов агентов, являются исходными данными для рассматриваемого этапа, этапа описания сервисов. В частности, на этапе описания моделей поведения классов агентов кроме описания содержательных требований в отношении идентифицированных сервисов могут быть также указаны связи между сервисами и протоколами взаимодействия. Таким образом, при проектировании таких сервисов часть требуемых сервисных функций может быть предопределена. Например, если для сервиса указано, что он используется для поддержки участия агентов класса в некотором протоколе взаимодействия, то в описание сервиса должны быть обязательно включены сервисные функции, предназначенные для обработки получения и отправки сообщений в соответствии со сценарием коммуникационных актов этого протокола. Такие состояния добавляются в описание соответствующих машин состояний автоматически.

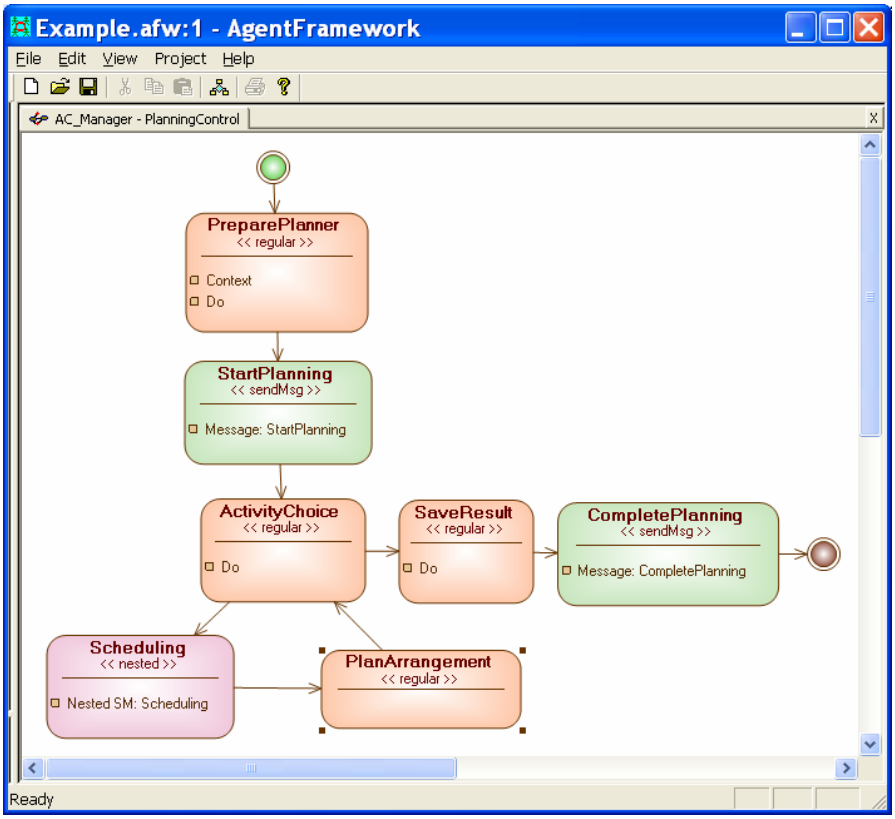

Рис. 8. Редактор машин состояний.

5.3. Описание частных онтологий классов агентов

Наряду с общей онтологией предметной области для классов агентов описываются частные онтологии. Понятия и отношения частных онтологий описывают структуры ментальных моделей классов агентов и используются для описания переменных сервисных функций.

Частные онтологии классов агентов могут наследовать понятия онтологии предметной области, а также включать новые понятия и отношения, необходимые только для описания классов агентов.

Труды СПИИРАН. 2006. Вып. 3. Т. 1. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2006. Issue 3. V. 1. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

Для описания частных онтологий понятий классов агентов используется тот же самый редактор, что и для описания онтологии предметной области (рис. 6), но реализующий дополнительные функции. Предназначение данных функций связано с необходимостью описания для классов агентов:

- структур хранилищ ментальных моделей, и
- компонент, реализующих механизм доступа агентов к хранилищам метальных моделей в процессе их функционирования.

В качестве типа хранилищ ментальных моделей может быть выбрана либо реляционная база данных, либо XML файл. При этом для решения двух указанных задач дополнительные функции позволяют описывать соответствие между понятиями и атрибутами онтологии и элементами структур хранилищ ментальных моделей. В частности, если в качестве типа хранилища выбрана реляционная база данных, то описывается соответствие между именами понятий онтологии и именами таблиц базы данных, а также — между атрибутами понятий и полями соответствующих таблиц базы данных.

6. Разработка программного кода классов агентов

6.1. Описание сервисных функций

Описание сервисных функций в среде выполняется на объектно-ориентированном языке программирования, в данном случае на языке C++. В связи с этим следует отметить, что в предыдущей версии инструментальной среды была исследована возможность описания сервисных функций на специально разработанном для этого языке скриптов. С одной стороны, это давало определенные преимущества на этапе разработки прикладных МАС. С другой стороны, использование такого языка предполагало исполнение сервисов агентами в режиме интерпретации, что существенно снижало временные показатели работы агентов. Это обстоятельство послужило основанием для решения об использовании в текущей версии инструментальной среды традиционного языка программирования для описания сервисных функций.

Описание сервисных функций выполняется с помощью специальных редакторов. Как было отмечено в пункте 7, в данной версии среды рассматривается шесть классов сервисных функций. Для описания каждого класса сервисных функций используется соответствующий редактор. На рис. 9 отображен редактор, который используется для описания базового класса сервисных функций. Редакторы, используемые для остальных классов сервисных функций, разработаны на основе данного редактора с учетом их дополнительных специфических возможностей.

Описание сервисных функций базового класса предполагает описание следующих фрагментов программного кода (рис. 9).

a. Список переменных, используемых при описании сервисной функции, специфицируется в поле *Context*. Список переменных составляется на основе понятий частной онтологии класса агентов.
b. Описание *основного сценария* сервисной функции выполняется в поле *Do*.
c. Описание *вспомогательного сценария* сервисной функции выполняется в поле *Exit*.
d. Описание условий переходов в другие состояния выполняется в таблице *Transition*.

Труды СПИИРАН. 2006. Вып. 3. Т. 1. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2006. Issue 3. V. 1. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

Описание основного сценария сервисной функции может содержать описание действий класса агентов, вследствие которых после выполнения основной части сервисной функции может возникнуть прерывание выполнения сервисной функции. После прерывания продолжение выполнения сервисной функции предполагает выполнение вспомогательного сценария, если он описан.

6.2. Генерация кода классов агентов

Генерация исходного и исполняемого кода классов агентов в соответствии со структурой, отображенной на рис. 2, выполняется генератором программных агентов (рис. 1). Генерация программного кода каждого класса агентов сводится к выполнению следующей последовательности функций.

1) Генерация исходного программного кода компонент *Модель поведения агента, Ментальная модель агента, Сервисы* (рис. 2) на языке C++ на основе их исходного описания на языке XML. Генерация кода выполняется автоматически на основе использования XSLT технологии.
2) Компиляция кода, описывающего класс агентов. При этом используются:
   - код проблемно-ориентированных компонент, сгенерированный в результате выполнения функции 1;
   - код, описывающий сервисные функции;
   - готовый код, описывающий основу агентов.
3) Генерация хранилища данных агентов на основе описания ментальной модели класса агентов.

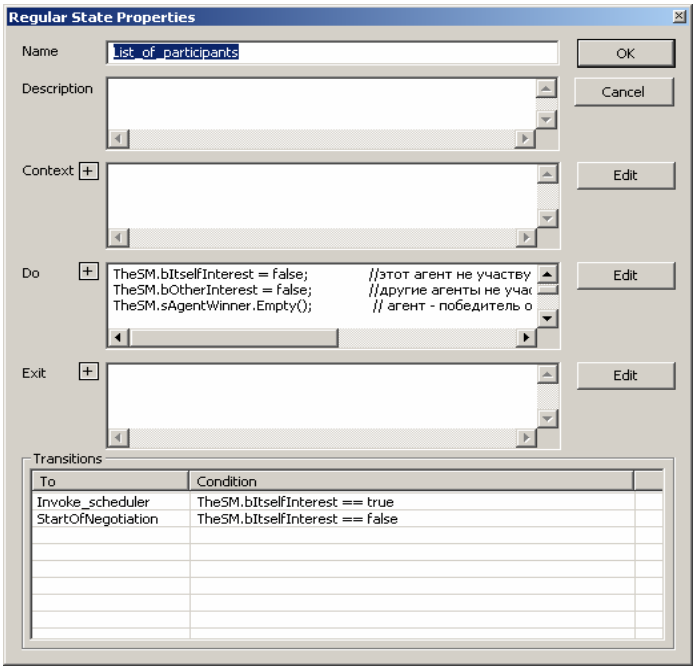

Рис.9. Редактор сервисных функций общего класса.

В результате выполнения указанной последовательности функций для каждого класса агентов формируются исполняемый программный код и хранилище ментальной модели.

7. Описание агентов и развертывание МАС

Две последние стадии разработки прикладной МАС, на которых выполняются этапы описания агентов, описания начальных ментальных моделей агентов и развертывания МАС, предусматривают выполнение задач, которые свойственны как процессу разработки прикладной МАС, так и процессу сопровождения прикладной МАС в ходе ее использования. В частности, в процессе разработки прикладной МАС описание агентов и их развертывания требуется для верификации и отладки системы. В процессе сопровождения прикладной МАС

Труды СПИИРАН. 2006. Вып. 3. Т. 1. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2006. Issue 3. V. 1. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

возможно изменение конфигурации множества агентов, изменение их начальных ментальных моделей, изменение их мест расположения. В связи с этим, редактор конфигурации агентов (рис. 10), с помощью которого выполняются перечисленные выше три этапа, обладает следующей особенностью. С одной стороны, наряду со всеми рассмотренными выше редакторами он является элементом инструментальной среды MASDK. С другой стороны, этот же редактор используется в качестве основного редактора вспомогательной системы MASDK Lite, которая используется как средство сопровождения прикладных МАС, разрабатываемых с помощью среды MASDK.

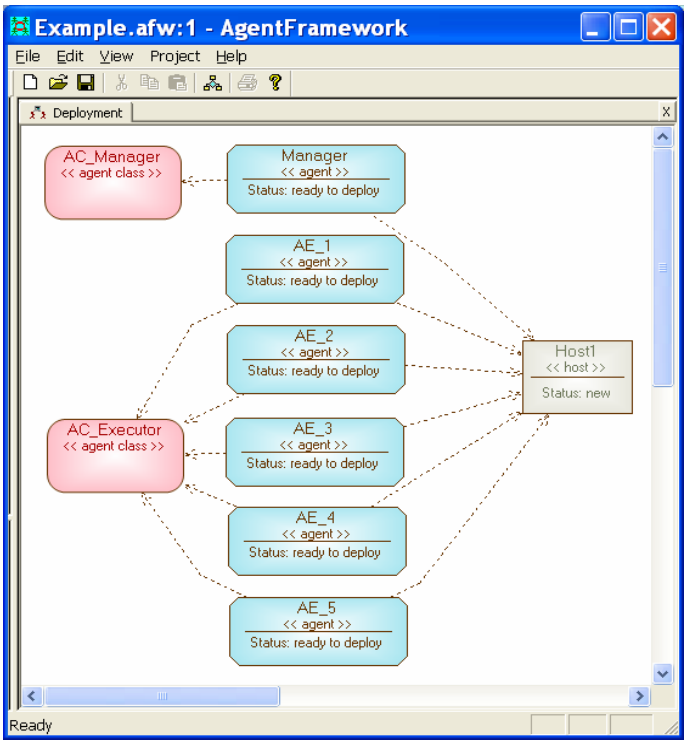

Рис.10. Редактор конфигурации агентов.

7.1. Описание агентов

Описание агентов сводится к тому, что для каждого из них необходимо:

- указать имя агента,
- указать класс агента,
- указать место размещение агента в сети, и
- описать начальную ментальную модель агента.

Последняя из перечисленных функций обладает определенной спецификой, и поэтому рассматривается в качестве отдельного этапа разработки/сопровождения прикладной МАС. Описание агентов системы отображается в графическом редакторе конфигурации агентов (рис. 10). В частности, в примере, приведенном на рис. 10, в конфигурации описано шесть агентов, один из которых является агентом класса AC_Manager, а остальные являются агентами класса AC_Executor. Также, в данном описании конфигурации указано, что все шесть агентов должны быть размещены на одном хосте. Размещение агентов на одном хосте, как правило, используется при верификации и отладке системы. При использовании прикладной МАС расположение агентов выполняется в соответствии с требованиями пользователя.

7.2. Описание начальных ментальных моделей агентов

Под начальными ментальными моделями агентов понимаются те данные и знания, которыми агенты должны обладать до момента начала их функционирования. Далее, в процессе функционирования агенты меняют свои ментальные модели в результате выполнения тех или иных функций в соответствии с рассматриваемыми задачами.

Труды СПИИРАН. 2006. Вып. 3. Т. 1. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2006. Issue 3. V. 1. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

В отношении описания начальных ментальных моделей агентов могут рассматриваться различные подходы. Например, в системе AgentBuilder [21] реализован подход, когда описание начальных ментальных моделей формируется при описании онтологии предметной области с помощью встроенного редактора. По сути, этот редактор является незначительным расширением редактора онтологии. Такой подход является эффективным в случае описание достаточно «простых» ментальных моделей. Другой возможный подход связан с использованием внешних достаточно развитых «стандартных» редакторов и последующим экспортированием данных из этих редакторов в хранилища ментальных моделей агентов. В частности, в качестве таких редакторов могут использоваться такие редакторы, как Protégé, OntoEditor и другие. Недостаток стандартных редакторов связан с их универсальностью и проявляется при описании начальных ментальных моделей в случае сложно структурированных онтологий предметных областей. В этом случае используется третий подход, предполагающий разработку специфических редакторов начальных ментальных моделей агентов, предназначенных для конкретных приложений.

В соответствии со спецификой онтологий прикладных систем, разрабатываемых в MASDK, в текущей версии нет встроенного редактора начальных ментальных моделей, а используются два других подхода, предполагающих либо использование внешних редакторов, либо разработку специфических редакторов ментальных моделей с учетом поддержки функции экспорта данных.

7.3. Развертывание прикладной МАС

Развертывание прикладной МАС выполняется с помощью специальной компоненты инструментальной среды — *Системы установки агентов* (рис. 1). В состав данной системы входит компонента *Портал*, которая должна быть предварительно установлена на тех узлах сети, где предполагается устанавливать агентов.

Развертывание прикладной МАС состоит в установке каждого агента прикладной МАС по адресу, указанному в его описании. Процесс установки агента выполняется *Системой установки агентов* и состоит из выполнения следующей последовательности функций.

1. Копирование по указанному адресу агента исполняемого кода соответствующего класса агентов.
2. Генерация для агента шаблона хранилища ментальной модели соответствующего класса агентов.
3. Загрузка начальной ментальной модели агента в его хранилище ментальной модели.
4. Копирование по указанному адресу хранилища ментальной модели агента, содержащего описание его начальной ментальной модели.
5. Регистрация агента на порталах, обеспечивающих коммуникационную функцию агентов прикладной системы.

В процессе поддержания прикладной МАС система установки агентов обеспечивает возможность добавления новых агентов, а также удаления ранее установленных агентов.

Труды СПИИРАН. 2006. Вып. 3. Т. 1. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2006. Issue 3. V. 1. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

8. Заключение

Инструментальная среда MASDK, описанная в данной статье, обладает рядом свойств, позволяющих существенно сокращать трудоемкость разработки прикладных МАС. Среди них наиболее важными являются следующие свойства.

1. Методология разработки прикладных МАС, положенная в основу создания инструментальной среды MASDK, является развитием одной из наиболее известных методологий разработки многоагентных систем — методологии Gaia. При этом, развитие сводится главным образом к тому, что на основании результатов двух начальных этапов, этапов анализа проблемной области и дизайна прикладной МАС, поддерживается весь последующий жизненный цикл разработки вплоть до развертывания и сопровождения прикладной системы в сети.
2. Использование в среде формального языка для описания прикладных МАС, позволяет иметь формальное описание взаимосвязей решений, разрабатываемых на различных этапах. В целом это обеспечивает контроль целостности прикладной МАС в процессе разработки и служит основой для работы вспомогательной мастер-компоненты.
3. Процесс разработки прикладных МАС в среде выполняется с помощью системы интегрированных графических редакторов диаграмм, что обеспечивает достижение двух основных целей. Первая цель состоит в том, что графическое проектирование полностью заменяет написание части программного кода прикладной системы. В процессе графического проектирования генерируется формальное описание решений и моделей верхнего уровня, разрабатываемых на этапе проектирования. При этом, часть программного кода, связанного с описанием этих решений и моделей формируется автоматически. Вторая цель состоит в верификации проекта прикладной системы до момента разработки тех компонент, чье описание в виде программного кода остается более рациональным решением. В среде MASDK такими компонентами прикладных систем являются сервисные функции. Следует отметить, что подобное разделение этапов проектирования и написания программного кода уже стало стандартным решением в случае объектно-ориентированного подхода. В связи с этим следует отметить, что реализация такого же подхода в случае агентно-ориентированного проектирования и программирования является в настоящее время одной из наиболее актуальных проблем в области развития многоагентных технологий.
4. В состав среды включены библиотеки, содержащие описание готовых решений на уровне программного кода, которые позволяют сокращать трудоемкость, связанную с описанием программного кода агентов, а также практически полностью исключают необходимость программирования компонент, связанных с реализацией специфики агентно-ориентированного подхода.

Труды СПИИРАН. 2006. Вып. 3. Т. 1. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2006. Issue 3. V. 1. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

Литература

1. Agent UML [Электронный ресурс] // <http://www.auml.org> (по состоянию на 03.04.2006).
2. Agent Modeling Language. Language Specification. Version 0.9. [Электронный ресурс] // <http://www.whitestein.com> (по состоянию на 03.04.2006).
3. *Bauer B., Muller J., Odell J.* Agent UML: Formalism for Specifying Multiagent Interaction // Agent-Oriented Software Engineering / *Ciancarini P., Wooldridge M.* (Eds.). Berlin: Springer-Verlag, 2001. P. 91–103.
4. *Bellifemine F., Caire G., Trucco T., Rimassa G.* Jade Programmer's Guide // JADE 2.5. 2002. [Электронный ресурс] / <http://sharon.cselt.it/projects/jade/> (по состоянию на 03.04.2006).
5. *Bernon C., Gleizes M. P., Peyruqueou S., Picard G.* Adelfe, a Methodology for Adaptive Multi-Agent Systems Engineering // Third International Workshop "Engineering Societies in the Agents World" (ESAW). Madrid, 2002.
6. *Bitting E., Carter J., Ghorbani A. A*. Multiagent Systems Development Kits: An Evaluation // Proceedings of the 1st Annual Conference on Communication Networks & Services Research. Moncton, Canada, 2003. P. 80–92.
7. *Booch G*. Object-Oriented Analysis and Design, 2nd ed. Addison-Wesley: Reading. MA., 1994. 608 p.
8. *Caire G., Leal F., Chainho P., Evans R., Garijo F., Gomez J., Pavon J., Kearney P., Stark J., Massonet P.* Agent-oriented analysis using MESSAGE/UML // Second International Workshop on Agent-Oriented Software Engineering (AOSE) / *Wooldridge M., Ciancarini P., Weiss G.* (Eds.). 2001. P. 101–108.
9. *Cossentino M., Sabatucci L., Sorace S., Chella A*. Patterns Reuse in the PASSI Methodology // Fourth International Workshop Engineering Societies in the Agents World (ESAW'03). London, UK, 2003. P. 294–310.
10. *Collis J., Ndumu D*. Zeus Technical Manual [Электронный ресурс] // <http://labs.bt.com/projects/agents/zeus/techmanual/TOC.html> (по состоянию на 17.04.2006)
11. *Dam K. H., Winikoff M*. Comparing Agent-Oriented Methodologies // Proceedings of the Fifth International Conference-Workshop on Agent-Oriented Information Systems (AAAMAS-03). Melbourne, Australia. 2003.
12. *DeLoach S., Wood M*. Developing Multiagent Systems with AgentTool // Intelligent Agents VII. Agent Theories Architectures and Languages, 7th International Workshop, LNCS. Vol. 1986 / Castelfranchi, C., Lesperance Y. (Eds.) Springer Verlag, 2001.
13. *DeLoach S. A., Wood M. F., Sparkman C. H*. Multiagent Systems Engineering // International Journal of Software Engineering and Knowledge Engineering. 2001. Vol. 11(3). P. 231–258.
14. FIPA-OS: A component-based Toolkit Enabling Rapid Development of FIPA Compliant Agents [Электронный ресурс] // <http://fipa-os.sourceforge.net/> (по состоянию на 03.04.2006).
15. *Giunchiglia F., Mylopoulos J., Perini A.* The Tropos Software Development Methodology: Processes, Models and Diagrams // Third International Workshop on Agent-Oriented Software Engineering. 2002.
16. *Gorodetsky V., Karsaev O., Kotenko I., Khabalov A*. Software Development Kit for Multi-agent Systems: Design and Implementation // From Theory to Practice in Multi-agent Systems. Lecture Notes in Artificial Intelligence. 2002. Vol. 2296 / Dunin-Keplicz B., Navareski E. (Eds.). P. 121–130.
17. *Gorodetsky V., Karsaev O., Konyushiy V*. Multi-Agent System for Resource Allocation and Schedulling // Proceedings of The 3rd International Central and Eastern European Conference on Multiagent Systems (CEEMAS'03). LNAI. 2003. Vol. 2691. P. 226–235.
18. *Gorodetsky V., Karsaev O., Samoilov V*. Multi-agent Technology for Distributed Data Mining and Classification // Proceedings of the IEEE Conference Intelligent Agent Technology (IAT-03). Halifax, Canada, 2003. P. 438–441.
19. Jack. Jack intelligent agents. Agent Oriented Software. Ltd., Australia [Электронный ресурс] // <http://www.agent-software.com.au> (по состоянию на 03.04.2006).
20. *Padgham L., Winikoff M*. Prometheus: A pragmatic Methodology for Engineering Intelligent Agents // Proceedings of the OOPSLA 2002 Workshop on Agent-Oriented Methodologies. Seattle, 2002. P. 97–108.

21. AgentBuilder: An Integrated Toolkit for Constructing Intelligent Software Agents. Revision 1.3. Reticular Systems Inc. [Электронный ресурс] // <http://www.agentbuilder.com/> (по состоянию на 03.04.2006)
22. *Wooldridge M., Jennings N. R., Kinny D*. The Gaia Methodology for Agent-Oriented Analysis and Design // Journal of Autonomous Agents and Multi-Agent Systems. 2000. Vol. 3, no. 3. P. 285–312.
23. *Woldridge M*. Agent-based Software Engineering // IEEE Proc. Software Eng. 1997. Vol. 144(1). P. 26–37.
24. *Woldridge M., Jennings N. R*. Pitfalls of Agent-oriented Development. // Proc. Second Int. Conf. On Autonomous Agents (Agents 98). Minneapolis/St Paul. MN, 1998. P. 385–391.
25. AgentBuilder. [Электронный ресурс] <http://www.agentbuilder.com/AgentTools/index.html> (по состоянию на 03.04.2006).
26. AgentLink. [Электронный ресурс] <http://www.agentlink.org/resources/agent-software.php> (по состоянию на 03.04.2006).