Міністерство освіти і науки України
Запорізький державний університет

До захисту допущений
Зав. кафедрою
ММ та ІТ

(підпис)

Борю Сергій Юрійович
(прізвище, ім'я, по батькові)

(дата)

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

НА ТЕМУ: **РЕАЛІЗАЦІЯ СЕРЕДОВИЩА ІНФОРМАЦІЙНОГО ОБМІНУ У ДИНАМІЧНІХ МУЛЬТИ-АГЕНТНИХ СИСТЕМАХ**

Виконав Плаксін Сергій Леонідович

ст. групи 8226-2 С. Л. Плаксін
(шифр) (підпис і дата) (ім'я, по батькові, прізвище)

Керівник доцент В. А. Єрмолаєв
(посада) (підпис і дата) (ім'я, по батькові, прізвище)

Нормоконтролер І. О. Лєвикіна
(підпис) (ім'я, по батькові, прізвище)

Запоріжжя,
2000

ЗАВДАННЯ

2

РЕФЕРАТ

Дипломная работа: 83 с., 4 рис., 8 табл., 1 приложение, 14 источников.

Объект исследования - модели и алгоритмы координации в динамических мульти-агентских системах.

Цель работы - исследование и анализ методов координации взаимодействующих интеллектуальных программных агентов; разработка модели, архитектуры и алгоритмов координации программных интеллектуальных агентов в динамическом сообществе, ориентированном на выполнение потоков работ (заданий).

Методы исследования - теоретические методы построения программных агентов и мультиагентских систем; методы дискретной математики; методы проектирования архитектуры программного обеспечения; методы проектирования концептуальных моделей и логических схем данных; методы и технология программирования на языках высокого уровня и языках запросов к реляционным базам данных.

Данная дипломная работа посвящена исследованию и разработке моделей и алгоритмов координации интеллектуальных программых агентов в динамических сообществах, используемых для моделирования деятельности реальных и виртуальных предприятий. Главной задачей данной работы является разработка активной среды информационного обмена, обеспечивающей координацию сообщества агентов. Модель координации, предложенная в этой работе, основана на классической доске объявлений, которая и является средой обмена информацией в сообществе агентов.

АГЕНТ, МУЛЬТИ-АГЕНТНАЯ СИСТЕМА, РОЛЬ, ПОЛИТИКА, ВОЗДЕЙСТВИЕ, КООРДИНАЦИЯ АГЕНТОВ, КООРДИНАЦИОННЫЙ АГЕНТ, АРХИТЕКТУРА АГЕНТА, КОНЕЧНЫЙ АВТОМАТ, МАКРОМОДЕЛЬ.

Содержание

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ

ACL - язык взаимодействия агентов (Agent Communication Language)

AI - искусственный интеллект

DAI - распределённый искусственный интеллект

DPS - распределенные обрабатывающие системы

KIF - формат обмена знаниями (Knowledge Interchange Format)

KQML - язык запросов и манипуляцией знаниями (Knowledge Query and Manipulation Language)

MAS - мультиагентная система (multi-agent system)

БД - база данных

БДКА - база данных координационного агента

КА - координационный агент

ВВЕДЕНИЕ

В последнее время резко возросло число компьютерных систем, разработанных и реализованных на базе теоретических моделей, архитектур и программных технологий, использующих концепцию агентов и мультиагентских систем. Использование технологии агентов упрощает процесс разработки программного обеспечения. Прикладные программы, написанные как программные агенты, могут взаимодействовать с другими программными агентами, обмениваясь сообщениями на языке взаимодействия агентов. Агенты могут быть столь же простыми, как и подпрограммы, однако обычно агентами являются довольно крупные программные комплексы. Агенты применяются в областях, начиная с относительно простых персональных помощников, которые, например, фильтруют электронную почту, заканчивая сложными информационными системами, которые управляют воздушным движением, космическими кораблями и т. п.

Несмотря на такую распространённость на сегодня нет единого и общепринятого определения термину агент. Так в обзорной работе [...] приводятся следующие определения программного агента:

I **The MuBot Agent** [http://www.crystaliz.com/logicware/mubot.html]

- **Англ.** *"The term agent is used to represent two orthogonal concepts. The first is the agent's ability for autonomous execution. The second is the agent's ability to perform domain oriented reasoning."*

II **The AIMA Agent (**[Russell and Norvig 1995], page 33)

- **Англ.** *"An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through effectors."*

III **The Maes Agent** [Maes 1995, page 108]

- **Англ.** *"Autonomous agents are computational systems that inhabit some complex dynamic environment, sense and act autonomously in this environment, and by doing so realize a set of goals or tasks for which they are designed."*
- **Рус.**

IV **The KidSim Agent** [Smith, Cypher and Spohrer 1994]

- **Англ.** *"Let us define an agent as a persistent software entity dedicated to a specific purpose. 'Persistent' distinguishes agents from subroutines; agents have their own ideas about how to accomplish tasks, their own agendas. 'Special purpose' distinguishes them from entire multifunction applications; agents are typically much smaller."*
- **Рус.**

V **The Hayes-Roth Agent** [Hayes-Roth 1995]

- **Англ.** *Intelligent agents continuously perform three functions: perception of dynamic conditions in the environment; action to affect conditions in the environment; and reasoning to interpret perceptions, solve problems, draw inferences, and determine actions.*
- **Рус.**

VI **The IBM Agent** [http://activist.gpl.ibm.com:81/WhitePaper/ptc2.htm]

- **Англ.** *"Intelligent agents are software entities that carry out some set of operations on behalf of a user or another program with some degree of independence or autonomy, and in so doing, employ some knowledge or representation of the user's goals or desires."*
- **Рус.**

VII **The SodaBot Agent** [Michael Coen http://www.ai.mit.edu/people/sodabot/slideshow/total/P001.html]

- **Англ.** *"Software agents are programs that engage in dialogs [and] negotiate and coordinate transfer of information."*
- **Рус.**

VIII **The Foner Agent** [Lenny Foner - Download from ftp://media.mit.edu/pub/Foner/Papers/Julia/Agents--Julia.ps or online at http://foner.www.media.mit.edu/people/foner/Julia/ (click on "What's an agent? Crucial notions")]

- **Англ.** *Foner requires much more of an agent. His agents collaborate with their users to improve the accomplishment of the users' tasks. This requires, in addition to autonomy, that the agent dialog with the user, be trustworthy, and degrade gracefully in the face of a "communications mismatch." However, this quick paraphrase doesn't do justice to Foner's analysis*
- **Рус.**

IX **The Brustoloni Agent** [Brustoloni 1991, Franklin 1995, p. 265]

- **Англ.** *"Autonomous agents are systems capable of autonomous, purposeful action in the real world."*

Большинство исследователей в этой области придерживаются следующего определения, данного Дженнингсом, Вулдриджем [....]: Агент - объект для решения некоторой задачи (реализованный аппаратными средствами, программными средствами или их комбинацией), который обладает следующими свойствами:

- автономностью: агенты должны быть способны выполнять свои задачи без прямого вмешательства человека или других агентов, а также иметь определённую степень контроля над своими действиями и собственным внутренним состоянием;
- способностью взаимодействовать: агенты должны быть способны взаимодействовать с другими агентами или человеком для решения своих задач;

- реактивностью: агенты должны реагировать на изменения, происходящие в их среде;
- агент должен не просто реагировать на изменения в среде, его действия должны быть направлены некоторой целью.

Иногда, в дополнении к этим свойствам добавляют мобильность, рациональность, правдивость и др.

Однако при решении сложных задач, использование автономных (одиночных) агентов не всегда эффективно и поэтому используют мультиагентные системы (MAS). Использование MAS позволяет решать более сложные, реалистические и крупномасштабные проблемы, которые не поддаются решению при помощи автономных агентов. Проблема моделирования деятельности реальных предприятий является одной из сфер применения интеллектуальных информационных агентов и мультиагентных систем. В [...] представлен подход к моделированию разнообразных процессов, протекающих в реальных/виртуальных предприятиях, базирующийся на динамических сообществах интеллектуальных программных агентов, ориентированных на выполнение заданий. Агенты в таком сообществе объединяются во временный консорциум автономных, возможно, географически рассредоточенных функциональных компонент, которые объединяют свои ресурсы и опыт для решения общих задач.

Одной из главных проблем, возникающих при использовании MAS является координация агентов в сообществе - обеспечение их непротиворечивой и бесконфликтной работы. Целью данной квалификационной работы бакалавра является разработка модели и архитектуры координации программных интеллектуальных агентов в динамическом сообществе, которое строится по моделям, изложенным в [...]

Основными задачами данной дипломной работы являются:

- исследовать и проанализировать существующие методы координации взаимодействующих агентов;
- на базе аппарата построения динамических сообществ агентов, предложенного в [6], разработать модель координации, архитектуру, схему базы данных среды информационного обмена и алгоритмы макромоделей поведения координационного агента.

1 ПРОБЛЕМЫ И МЕТОДЫ КООРДИНАЦИИ В МУЛЬТИАГЕНТСКИХ СИСТЕМАХ

1.1 Мультиагентные системы - базовые понятия и характерные приложения

Управляемые компьютером среды как, например, автоматизированные фабрики, атомные электростанции, центры передачи данных, космические станции и т. д. становятся всё более сложными. Поскольку эта сложность непрерывно растёт, будет всё труднее управлять такими системами с централизованным управлением и политикой планирования, которые становятся неустойчивыми в быстро меняющихся, динамических средах.

Ещё в середине 1970-ых, исследователи в области искусственного интеллекта (AI) показали, что взаимодействие и декомпозиция общей задачи эффективно сказывается на результатах выполнения этой задачи. Кроме того, подобные эксперименты показали, что интеллектуальное, рациональное поведение не является признаком изолированных

компонентов, а, скорее, результат, который появляется при взаимодействии объектов с более простым поведением [1].

Большинство исследователей в области AI до настоящего времени имели дело с развивающимися теориями, методами, и системами, чтобы изучать и понять поведение и рассуждение одиночного познавательного объекта. Прогресс в области AI в последнее время позволил решать более сложные, реалистические и крупномасштабные проблемы. Такие проблемы - вне возможностей индивидуального агента. Возможности одиночного интеллектуального агента ограничены его знанием, его вычислительными ресурсами, и его архитектурой.

Поэтому, около двух десятилетий назад, из AI выделилось новое направление - распределённый искусственный интеллект (DAI). DAI рассматривает системы, которые состоят из множественных независимых объектов, взаимодействующих в некоторой области. Традиционно, DAI разделен на две поддисциплины: распределенные обрабатывающие системы (DPS) и мультиагентные системы (MAS).

DPS занимается информационными аспектами управления системами с несколькими подзадачами, работающими вместе для достижения общей цели. Главными проблемами DPS является декомпозиция задачи и синтез решения.

MAS - новая подобласть DAI, которая стремится обеспечить принципы для построения сложных систем, включающих множественных агентов и механизмы для координации поведения независимых агентов.

На сегодня нет общепринятого единственного определения MAS. Большинство исследователей определяют MAS следующим образом.

MAS - это слабосвязанная сеть прикладных решающих устройств, которые взаимодействуют, чтобы решить проблемы, которые являются вне индивидуальных возможностей или знания каждого прикладного решающего устройства [1, 2]. Эти прикладные решающие устройства, часто называемые агентами, являются автономными и могут быть

гетерогенными. Кроме того, MAS обладает следующими характеристиками:

- каждый агент имеет неполные способности решить проблему;
- не имеется никакого глобального системного управления;
- данные децентрализованы;
- вычисления асинхронны.

Как известно, при построении больших информационных систем наиболее мощными средствами для упрощения сложной задачи являются декомпозиция (разбиение на отдельные модули) и абстракция в смысле упрощения и обобщения. Мультиагентные системы (MAS) предлагают модульность. Если предметная область очень сложная, большая, или непредсказуемая, то единственным разумным методом решения этой проблемы должно быть выделение из предметной области ряда функционально определенных компонентов (агентов), которые специализированы для решении специфического прикладного аспекта. Эта декомпозиция позволяет каждому агенту решать только свои специфические проблемы. Когда возникают взаимозависимые проблемы, агенты в системе должны быть скоординированы, чтобы гарантировать, их непротиворечивую работу.

Первое приложение, использующее MAS, появилось к концу 1980-ых. Несмотря на небольшой период развития, уже сейчас MAS распространены во многих предметных областях, от управления производственными процессами и промышленными предприятиями, до управления и планирования воздушным движением; от управления информационными потоками и поиском информации, до моделирования деятельности предприятия.

Одним из самых первых приложений, использующих MAS, было DVMT (Distributed Vehicle Monitoring) [2]. В этом приложении набор географически распределённых агентов, контролирует транспортные

средства, проходящие по соответствующим областям, прослеживая движение транспортных средств.

В области управления производством одним из первых проектов использующих MAS был YAMS (YET ANOTHER MANUFACTURING SYSTEM) [2]. Коротко эта система может быть описана следующим образом: производственное предприятие смоделировано как иерархия функционально определённых компонентов работы. Эти компоненты работы сгруппированы в гибкие производственные системы, которые все вместе составляют предприятие. Цель YAMS - эффективно управлять промышленным процессом на предприятии. Чтобы достичь этой сложной цели в YAMS использует MAS, где каждый компонент представлен как агент. Каждый агент имеет набор задач, которые он может выполнять. Таким образом, при поступлении нового заказа на предприятие общая задача выполняется, продвигаясь по иерархической модели производства, разбиваясь на более мелкие подзадачи.

Наиболее известной системой, использующей MAS для управления процессом, является ARCHON [2, 3]. Это программная платформа для создания MAS и методологии для построения приложений на этой платформе. ARCHON применялся в нескольких приложениях управления процессом включая управление транспортированием электроэнергии (внедрено в промышленную эксплуатацию в северной Ирландии) и управление ускорителем частиц. Другие приложения управления, использующие MAS, были написаны для контроля и диагностики ошибок на атомных электростанциях, управления космическими кораблями, управления климатом.

Мультиагентная система используется также в сложной системе управления воздушным движением OASIS [2]. В этой системе, которая проходит апробацию в Сиднейском аэропорту в Австралии, агенты используются, чтобы представить такие реальные объекты как самолёты, и различные операции по управлению воздушным движением. Агенты,

таким образом, позволяют достаточно просто моделировать реальные автономные объекты. Агенты в такой MAS распределены и обладают информацией и целями, соответствующими реальному самолёту. Например, самолёт мог бы иметь цель приземлиться на некоторой взлётно-посадочной полосе в некоторое время. Агенты в этой системе имеют BDI модель [4]. OASIS был разработан Австралийским институтом искусственного интеллекта.

WARREN [2] - финансово-портфельная MAS, которая интегрирует поиск и фильтрацию информации из интернет в контексте поддержки финансового портфеля пользователя. Система состоит из агентов, которые совместно взаимодействуют, чтобы контролировать и следить за биржевыми ценами, финансовыми новостями, финансовыми сообщениями аналитиков, и сообщениями о доходах компании и т. п. Агенты не только отвечают на допустимые запросы, но и непрерывно контролируют информационные источники в интернет для поиска необходимой информации.

Кроме того, реальные проблемы имеются в распределенных, открытых системах. Открытая система - система, в которой структура и состав самой системы способна динамически изменяться. А также ее компоненты не известны заранее, могут изменяться через какое-то время и могут состоять из гетерогенных агентов, созданных разными людьми, в разное время, при помощи различных программных инструментальных средств и методов. Возможно, наиболее ярким примеров такой системы является интернет. В подобной среде информационных источников агенты могут появляться и исчезать неожиданно. В настоящее время, агенты в интернет главным образом исполняют информационный поиск и фильтрацию. Эти задачи требуют, чтобы агенты были способны взаимодействовать друг с другом. Кроме того, способность агентов к координации совместных позволит агентам расширить круг решаемых ими задач.

Растущая популярность MAS при построении сложных систем объясняется рядом преимуществ, которыми обладает MAS перед традиционными подходами. Использование MAS при построении сложных информационных систем позволяет:

1. решить проблемы, которые являются слишком сложными для одиночного агента;
2. решить проблемы, связанные с ограниченными ресурсами или явным наличием критических параметров эффективности;
3. сохранять соответствие программного обеспечения реальному миру. Для этого система должна периодически модифицироваться. Полная перезапись такого программного обеспечения, как правило, очень дорогостоящая, а часто просто невозможна. Модифицировать аналогичную систему, включающую сообщество агентов, гораздо проще, так как необходимо изменить лишь отдельного агента/агентов.
4. решить проблемы, которые могут естественно быть расценены как сообщество автономных взаимодействующих агентов (например, управление воздушным движением, моделирование деятельности предприятия, и т. п.);
5. обеспечить решения проблем, которые эффективно используют информационные источники, которые пространственно распределены. Примерами таких областей могут быть сети датчиков, сейсмический контроль, информация, накапливающаяся из интернет.
6. обеспечить решения в ситуациях, где обработка, знания распределены. Примеры таких областей включают параллельную разработку, здравоохранение и производство.
7. увеличить эффективность системы по таким аспектам:
   - *скорость вычислений*, так как используется параллелизм;
   - *надежность*, то есть простое восстановление данных при сбоях;
   - *масштабируемость*, потому что число и возможности агентов, работающих над проблемой, может легко меняться;

- *ошибкоустойчивость,* способность системы допустить неопределенность, потому что агенты обмениваются необходимой информацией;
- *удобство сопровождения*, потому что систему, состоящую из множественных агентов проще поддержать из-за ее модульности;
- *быстрота реагирования,* потому что модульность позволяет обрабатывать аномалии в местном масштабе, не размножая их по целой системе;
- *гибкость,* потому что агенты с различными способностями могут адаптивно объединяться для решения текущих проблем;
- *многократное использование*, потому что функционально определенные агенты могут многократно использоваться в различных сообществах агентов для решения различных проблем.

1.2 Проблема координации в MAS

Однако, несмотря на все перечисленные преимущества, проектирование и формирование систем, использующих MAS трудно. Такие системы не лишены проблем, связанных с формированием традиционных распределённых параллельных систем и дополнительных трудностей, которые являются результатом наличия гибких и сложных взаимодействий между автономными проблемными компонентами. Основной проблемой при использовании MAS является обеспечение непротиворечивого и эффективного взаимодействия автономных компонентов - агентов.

Как отмечает К. Сикара [http://www.aaai.org/Pathfinder/html/multi-agent_systems.html], изучение мультиагентных систем сосредотачивается на системы, в которых много интеллектуальных агентов взаимодействуют друг с другом. Агенты, как рассматривается, являются автономными объектами, типа программ или роботов. Их взаимодействия могут быть

или кооперативными или эгоистичными. То есть агенты могут совместно преследовать общую цель (как в колония муравьёв), или они могут преследовать собственные интересы (как в свободной рыночной экономике). Поэтому необходимо реализовать каждого муравья в колонии так, чтобы заставить их всех приносить продовольствие наиболее эффективным способом, или установить правила взаимодействия так, чтобы группа эгоистичных агентов работала вместе для выполнения поставленной задачи.

Таким образом, координация - центральная проблема в MAS в частности и в распределенном искусственном интеллекте (DAI) вообще. По существу, координация - *процесс,* в котором агенты участвуют, чтобы обеспечить сообществу непротиворечивую, согласованную последовательность действий. Это означает, что действия агентов спланированы так, что они не конфликтуют друг с другом, а всё сообщество ведёт себя как единое целое. Имеется несколько причин, по которым MAS должна быть скоординирована [5]:

- *Предотвращение анархии или хаоса*: координация необходима или желательна, так как в децентрализованной MAS анархия может установиться легко;
- *Наложение глобальных ограничений*: всегда существуют глобальные ограничения, которые MAS должна удовлетворить, чтобы успешно выполнить задание;
- *Распределённые знания, ресурсы, информация*: практически всегда в MAS имеются ресурсы, работа с которыми требует координации;
- *Зависимость между действиями агентов*: цели агентов часто взаимосвязаны.
- *Эффективность*: даже когда агенты могут работать независимо, координация позволяет сэкономить время.

Продемонстрируем необходимость координации на примере, описанном в [6]. Этот пример демонстрирует принципы моделирования процессов планирования, описанные в статье [6].

В этом примере рассматривается модель структурного подразделения предприятия, функцией которого является разработка и внедрение информационных систем. Модель представлена сообществом агентов (MAS), которое представляет функциональные возможности некоторого автономного объекта (руководителя подразделения), находящегося на более высоком уровне иерархии предприятия. Членами (агентами) рассматриваемого структурного подразделения (сообщества агентов) являются организатор проекта (**PM** агент), группа администраторов баз данных (**DBA** агент), группа программистов (**PROG** агент), группа по документированию (**DOC** агент), группа тестирования (**TST** агент), группа взаимодействия с пользователями/клиентами (**LIA** агент).

Множество атомарных работ, которые выполняют агенты рассматриваемого сообщества, выглядит следующим образом [6]:

$$\{ w_1 = (\textit{'Assemble Project Proposal'}, X_1, Y_1),$$

$$w_2 = (\textit{'Choose best DB Schemata Plan Bid'}, X_2, Y_2),$$

$$w_3 = (\textit{'Choose best Software Model Plan Bid'}, X_3, Y_3),$$

$$w_4 = (\textit{'Choose best REQ Analyses Plan Bid'}, X_4, Y_4),$$

$$w_5 = (\textit{'Analyse requirements'}, X_5, Y_5),$$

$$w_6 = (\textit{'Design database schemata'}, X_6, Y_6),$$

$$w_7 = (\textit{'Design software model'}, X_7, Y_7),$$

$$w_8 = (\textit{'Program the software'}, X_8, Y_8),$$

.....

$$w_{15} = (\textit{'Perform customers training '}, X_{15}, Y_{15})\}.$$

В [6] рассматривается поведение и взаимодействие агентов при получении агентом **PM** в момент времени t_0 внешнего воздействия следующего вида:

$W_a = \{w_0 = '\text{Pr}\ opose\ \_IS\ \_Developmen\ \ t\ \_Plan\ ', X_0, Y_0\}$

с параметрами и описанием результатов в виде

$X_0 = \{$***budget*** $=<$ *figure* $>$, ***duration*** $=<$ *figure* $>$,

Proposal_Template $=<$ *file _name* $>$,

Proposal_Descr $=<$ *file _name* $>\}$;

$Y_0 = \{Possibility(\textbf{\textit{budget}}, \textbf{\textit{duration}}), Proposal(\textbf{\textit{Proposal\_Template}})\}$.

После проверки полученного воздействия на возможность выполнить, агент **PM** разбивает поступившую работу на более мелкие, атомарные работы из множества, описанного выше, и поручает их выполнение на следующем шаге моделирования $t_0 + \Delta t$ агентам из сообщества.

Так, работы w_1, w_2, w_3, w_4 он перенаправляет себе, работу w_5 - агентам **DBA** и **LIA**, работу w_6 - агентам **DBA**, **PROG** и **LIA**, работу w_7 - агентам **DBA**, **PROG** и **LIA,** работу w_{15} - агенту **LIA**.

Далее в [6], рассматривается поведение агентов на следующих шагах моделирования t_1, t_2, t_3, t_4.

Очевидно, что работы w_1, w_2, w_3, w_4 не могут быть выполнены, пока не будут известны результаты выполнения остальных работ. Это означает, что для успешного выполнения общей задачи, необходима координация действий агентов, обеспечивающая непротиворечивую последовательность выполнения работ. То есть, вначале должны быть выполнены работы w_5 - w_{15}, затем - работы w_2, w_3, w_4, и только потом - работа w_1.

Такая последовательность выполнения работ объясняется зависимостью между действиями агентов (результаты выполнения одной работы являются исходными данными для другой работы). Эффективность выполнения общей задачи также зависит и от своевременности и точности передачи промежуточных результатов.

Итак, механизм координации агентов в сообществе должен обеспечивать бесконфликтную последовательность выполнения работ агентами и среду для обмена промежуточными результатами.

1.3 Методы координации в MAS

Существует много подходов обеспечения координации в MAS. Все эти подходы можно разделить на четыре категории [5]:

• организационное структурирование;

• заключение контракта;

• планирование деятельности MAS;

• переговоры.

1.3.1 Организационное структурирование

Это самый простой метод координации, состоящий в предопределённых и долгосрочных отношениях между агентами (Durfee [5]). При этом часто используют иерархические структуры *master-slave* или *client-server*. Эта методика используется в двух вариантах:

- Главный агент планирует и распределяет задания (работы) между подчинёнными агентами. Подчинённые агенты, в конечном счете, должны сообщить главному агенту о результатах их работы. Кроме

того, в отличие от главного агента, подчинённые агенты имеют частичную автономность.

- Использование классической доски объявлений - общего информационного поля - для обеспечения координации. В этой схеме агенты используют доску для обмена информацией. Главный агент (планировщик) назначает агентам операции чтения/записи с доской. Эта схема используется Веркман в его DFI системе [5]. Этот подход может использоваться, когда задача распределена, имеется центральный агент планирования или когда задачи уже были назначены, *априорно,* агентам. Sharp Multi-Agent Kernel (SMAK) [5] тоже использует эту стратегию.

Последний вариант показывает, что координация в организационном структурировании не всегда связана с иерархией. Например, в системе DVMT [5], использующей технологию доски координация проходит среди равных агентов.

Узким местом в системах, основанных на технологии доски без прямого взаимодействия агентов, может быть большое количество агентов в сообществе, даже в случае секционированной доски. Кроме этого все агенты должны иметь общее представление о доске. Поэтом большинство систем, основанных на этой технологии, используют небольших гомогенных агентов (как, например DVMT).

Дурфи [5] считает, что такое централизованное управление, как в иерархической методике, противоречит основным предположениям о DAI. В этой методике предполагается, что, по крайней мере, один агент имеет глобальное представление всего сообщества, однако во многих областях это не реально. Если всё-таки эта методика координации используется, проектировщик должен гарантировать, что все подчинённые агенты имеют оптимальную степень детализации чтобы время, затраченное на декомпозицию общей задачи на более мелкие, не превысило времени её выполнения одним агентом.

1.3.2 Заключение контракта

В этом подходе, который предполагает децентрализованную структуру, агенты принимают две роли:

- менеджер, который делит поступившую задачу на подзадачи и ищет исполнителя, чтобы выполнить их;
- исполнитель, выполняет подзадачу. Однако исполнитель может рекурсивно стать менеджером и разбить поступившую подзадачу на ещё более мелкие задачи и поручить их другим агентам.

Поиск исполнителя выглядят таким образом (FIPA CNP [7]):

1. Менеджер объявляет задачу;
2. Исполнители оценивают эту задачу с точки зрения возможности её выполнить;
3. Менеджер получает таблицу исполнителей;
4. Оценивает полученные предложения, выбирает исполнителя и поручает выполнить ему задачу;
5. Ожидание результатов выполнения задачи.

Это полностью распределённая схема, в которой каждый агент может быть как менеджером, так и исполнителем. Этот подход применяется во многих приложениях.

Хухнс и Сингх [5] обращают внимание на то, что эта модель координации обеспечивает распределение общей задачи, и средства для самоорганизации группы агентов, и лучше всего применима когда:

• прикладная задача имеет четкий иерархический характер (природу);

• прикладная задача делима на крупные подзадачи;

• имеется минимальная взаимосвязь среди подзадач.

Преимущества этого подхода координации состоят в следующем: динамическое распределение задачи благодаря поиску оптимального исполнителя; сбалансированная загрузка агентов (занятые агенты могут не участвовать в выполнении общей задачи); и надежный механизм для распределенного управления и восстановление результатов при сбое.

Агенты в такой системе довольно пассивны и не применимы для многих прикладных задач. Наконец, такое сообщество агентов довольно интенсивно использует сеть, что может снизить все его преимущества.

1.3.3 Планирование деятельности MAS

Имеется два типа планирования деятельности MAS:

• централизованное планирование деятельности MAS;

• распределенное планирование деятельности MAS.

В централизованном планировании деятельности MAS обычно имеется координационный агент, который, получив все индивидуальные планы от остальных агентов, анализирует их, чтобы обнаружить потенциальные противоречия и конфликты во взаимодействии агентов (например, конфликт между агентами по использованию общего ресурса). Потом координационный агент пытается изменить эти индивидуальные планы и объединяет их в план MAS, в котором противоречивые взаимодействия устранены. В заключительном плане MAS, добавлены команды связи, чтобы синхронизировать взаимодействия агентов в потенциально возможных конфликтах.

В распределенном планировании деятельности MAS, идея состоит в том, чтобы обеспечить каждого агента моделью планов других агентов. Агенты взаимодействуют, чтобы создавать и модифицировать их индивидуальные планы, не конфликтующие с планами других агентов.

Координация в распределенном планировании деятельности MAS - намного сложнее, чем в централизованной форме, так как там не существует агентов, обладающих глобальным представлением обо всей распределённой системе.

1.3.4 Переговоры

Это один из самых сложных методов координации. Любая форма человеческого взаимодействия требует в некоторой степени явных или неявных переговоров. Поэтому, не удивительно, что многие исследователи переговоров как метода координации применяют именно человеческие стратегии ведения переговоров. Кроме того, при координации агентов этим подходом часто используют разные методы AI, логики, случайно основанного рассуждения, направленного ограничением поиска, и т.д.

1.4 Анализ разработок по координации взаимодействующих агентов

Таким образом, каждый из перечисленных подходов координации имеет свои преимущества и недостатки. Каждый из них идеально подходит только для некоторых специфических теоретических областей. В то же время ни один из них не удовлетворяет в полной мере требованиям реальных прикладных задач.

Следовательно, при построении MAS, отвечающей реальной задаче, в качестве модели координации необходимо, по крайней мере, использовать комбинацию, описанных выше подходов. Кроме того, очевидным является факт, что каждая такая задача будет использовать свою модель координации, и не существует универсального метода координации, идеально подходящего к любой реальной задаче.

При моделировании деятельности виртуальных и реальных предприятий, очевидно, преобладающим подходом координации будет

организационное структурирование. Этот подход достаточно хорошо может отражать иерархическую структуру предприятия. Кроме того, это один из самых простых подходов координации. Другие методы координации имеют серьёзные недостатки при решении задач подобного рода (переговоры - сложность реализации; планирование MAS - идеально подходит лишь для специфических задач, таких как планирование авиа рейсов и т. д.; заключение контракта при моделировании виртуальных предприятий напоминает организационное структурирование, так как сеть контрактов фактически предопределена изначально, то есть, нет необходимости терять время на поиск исполнителя, а сразу поручить задание).

1.5 Координационный агент

При моделировании деятельности предприятий вариант организационного структурирования, использующий общее информационное поле - классическую доску объявлений, имеет в сравнении с иерархическим вариантом некоторые преимущества. А именно, в случае, когда в сообществе нет центрального агента планирования, имеющего глобальное представление о сообществе, координация агентов, выполняющих априорно назначенные задачи, будет осуществляться именно доской объявлений. Кроме того, подчинённые агенты в иерархическом варианте координации имеют частичную автономность.

Недостатки систем, координация которых основана на технологии доски, легко устраняются при использовании *активной* доски объявлений. То есть в сообществе должен быть агент, который бы обеспечивал работу остальных агентов с доской. Другими словами этот агент инкапсулирует в себе доску объявлений и методы работы с ней - операции чтения/записи.

В этом случае отпадают проблемы, связанные с одновременным доступом, так как все операции с общим информационным полем производит один агент. Кроме того, нет необходимости каждому агенту иметь общее представление о доске и знать её структуру потому, что взаимодействие с доской аналогично взаимодействию между агентами - при помощи языков взаимодействия агентов[8] - ACL[7]/KQML[9]. Это позволяет легко добавлять/изменять агентов в сообщество, и модифицировать саму доску объявлений.

Агента, обеспечивающего работу доски, а, следовательно, и координацию сообщества, можно назвать координационным агентом (КА).

Координационная функция *активной* доски объявлений состоит в хранении промежуточных результатов работы сообщества. Другими словами доска объявлений является средой информационного обмена в сообществе агентов. Задачей этой работы является реализация такой среды.

1.6 Вывод

Итак, основным выводом из этой главы является следующее. В качестве метода координации сообщества агентов при моделировании деятельности предприятия, необходимо использовать классическую доску объявлений. Более того, необходимо использовать активную доску объявлений.

Теперь, когда определена модель координации, основными задачами следующей главы будут:

- определить функции координационного агента;
- определить архитектуру координационного агента;
- разработать организацию хранения промежуточных результатов.

2 МОДЕЛЬ КООРДИНАЦИИ И КООРДИНАЦИОННЫЙ АГЕНТ

2.1 Введение

В разделе 1.2 на примере из [6] была продемонстрирована необходимость координации сообщества агентов. Взаимодействие агентов рассматриваемого сообщества выглядит как последовательность выполнения работ. Другими словами координация работы сообщества состоит в обеспечении непротиворечивой последовательности выполнения работ. Так, в рассматриваемом примере из [6], агент **PM** не может выполнить работу w_3 раньше, чем агенты **DBA**, **PROG**, **LIA** выполнят работу w_6, то есть пока не будут известны результаты $\widetilde{Y}_6^{DBA}, \widetilde{Y}_6^{PROG}, \widetilde{Y}_6^{LIA}$.

Обмен результатами выполненных работ происходит через координационного агента - активную доску объявлений. Выполнив работу, агенты отсылают результаты координационному агенту - помещают их на доску объявлений, тем самым обеспечивая возможность выполнения других работ, параметрами которых являются эти результаты.

Агент, выполняющий работу, параметрами которой являются результаты другой работы, должен запрашивать их у координационного агента - активной доски объявлений.

Этот механизм обеспечивает правильную и непротиворечивую последовательность выполнения работ агентами и обмен промежуточными результатами между ними. Более подробное описание модели координации будет дано в следующем разделе.

2. 2 Модель координации. Функции координационного агента

Используя результаты первой главы, диакоптический подход к моделированию сообществ агентов, описанный в [6, 10, 11], построим модель координации агентов, основанную на активной доске объявлений.

Как отмечалось в разделе 1.4, использование активной доски объявлений отличается от классического подхода координации лишь тем, что все операции чтения/записи с общим полем памяти производит только один агент - координационный агент, который, по сути, инкапсулирует в себе саму доску объявлений, и обеспечивает методы работы с ней. Остальные агенты, при необходимости прочитать или записать на доску некоторую информацию, должны взаимодействовать с координационным агентом при помощи ACL/KQML сообщений.

Чтобы более подробно описать модель координации, последовательность действий агентов, взаимодействие агентов сообщества с координационным агентом, возвратимся к примеру из [6].

Для простоты рассмотрим лишь процесс выполнения агентом **PM** работы w_2 (см. Рис. 2.1).

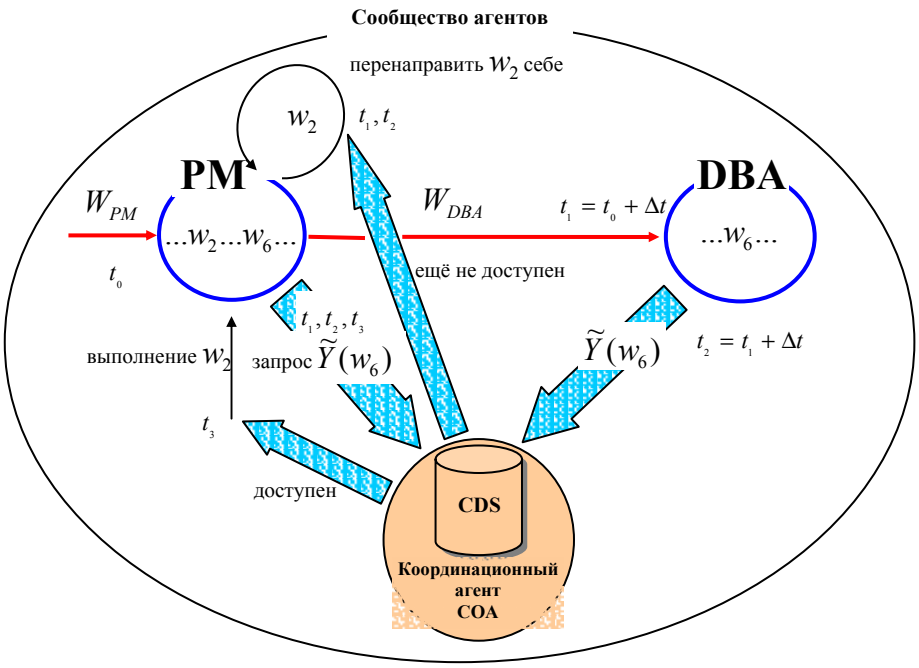

Рисунок. 2.1 - Модель координации

Работа w_2 поступает к агенту **PM** в момент времени t_1. Однако она не может быть выполнена за время $]t_1, t_1 + \Delta t]$, так как для её выполнения необходим результат работы w_6 агентом **DBA,** который нужно запросить у координационного агента - **COA**. Макромодель $F_O^{PM}(w_2)$ выполнения работы w_2 агентом **PM** может быть представлена следующим алгоритмом:

...

Avail := **Require(**"*COA*" **with** ("**ProvideResults**", $\widetilde{Y}(w_6)$ **))**

IF .NOT. Avail THEN

Redirect w_2 **TO PM** // w_2 перенаправляется **PM** на

//следующий момент времени $t_1 + \Delta t$.

ELSE

Execute(w_2 **)** //выполнение w_2 агентом **PM** (см. Рис. X)

END IF

...

В этот же момент времени t_1 к агенту **DBA** поступает работа w_6. Агент **DBA** выполняет работу w_6 в интервале времени $]t_1, t_1 + \Delta t]$ и отсылает результат $\widetilde{Y}(w_6)$ координационному агенту. Это может быть описано следующим образом:

...

$\tilde{Y}$:= **Execute(** w_6 **)** //выполнение работы w_6

Invoke("*COA*" **with** ("**AcceptResults**", $\tilde{Y}$ **))**//отправление

// $\tilde{Y}$ агенту COA

...

Реакция координационного агента **COA** на полученное от агента **DBA** воздействие типа **AcceptResults** имеет такой вид:

...

Execute("AcceptResults", DBA, $\tilde{Y}$ **)**

Реакция **COA** в ответ на запрос **PM** результатов работы w_6 , может быть описана в виде следующего алгоритма:

...

Avail := **Execute("ProvideResults",** $\widetilde{Y}(w_6)$ **))**

IF .NOT. Avail THEN

Reply("PM" with ("AcceptResults", "DUMMY"))

ELSE

Reply("PM" with ("AcceptResults", $\widetilde{Y}(w_6)$ **))**

END IF

...

Таким образом, в момент времени t_2 реакция **COA** будет отвергающей, так как результатов работы w_6 ещё нет, а в момент времени t_3 , запрос агента **PM** будет удовлетворён.

Необходимо отметить, что результаты выполненных работ, которые передаются между агентами, должны быть представлены в некотором универсальном формате, «понятном» всем агентам. Таким форматом может быть KIF (Knowledge Interchange Format) [12] - формат обмена знаниями - это формальный язык для обмена знаниями между несовместимыми компьютерными программами, написанными разными программистами, в разное время, на разных языках и т. д.

Возвращаясь к примеру из [6], напишем на языке KQML[7, 9] сообщения, обеспечивающие взаимодействие с координационным агентом.

KQML сообщение для отсылки результатов выполненной работы координационному агенту может иметь такой вид:

(**ask-one**

:sender	"DBA"
:receiver	"*COA*"
:in-reply-to	Null
:reply-with	Null
:language	KIF
:ontology	COA ontology
:contents	("**AcceptResults**", $\widetilde{Y}(w_6)$))

KQML сообщение для запроса результатов работы у координационного агента:

(**ask-one**

:sender	"PM"
:receiver	"*COA*"
:in-reply-to	Null
:reply-with	$\widetilde{Y}(w_6)$
:language	KIF
:ontology	COA ontology
:contents	("**ProvideResults**", $\widetilde{Y}(w_6)$))

KQML сообщение - возвращение результатов координационным агентом:

(**tell**

:sender	"*COA*"

:receiver "PM"
:in-reply-to Null
:reply-with Null
:language KIF
:ontology COA ontology
:contents ("**AcceptResults**", $\widetilde{Y}(w_6)$))

Более полное описание значений таких параметров KQML сообщений как language, ontology, contents будет дано в разделе 3.1.

Таким образом, основными функциями координационного агента являются получить, какое-то время хранить и отослать по запросу промежуточные результаты выполнения работ в сообществе.

2.3 Типы допустимых воздействий для координационного агента

Учитывая главную функцию координационного агента в сообществе - хранить промежуточные результаты работ, можно сказать, что основными допустимыми типами внешних воздействий, на которые должен реагировать координационный агент являются: 1) получение результата некоторой работы; 2) запрос результата некоторой работы.

Далее в этой работе получение координационным агентом результата некоторой работы будем называть *внешним воздействием первого типа*, а запрос у координационного агента результатов некоторой работы - *внешним воздействием второго типа*.

В терминах [6, 10, 11] внешнее воздействие первого типа будет директивой, так как от координационного агента требуется лишь принять результат; внешнее воздействие второго типа будет детерминированным запросом с детерминированной реакцией, так как координационный агент

должен возвратить запрашиваемый результат или сообщить, что такого результата ещё нет.

Очевидным является тот факт, что эффективность работы всего сообщества зависит от того, насколько быстро координационный агент будет реагировать на внешние воздействия. Другими словами критическим параметром работы координационного агента является время реакции. Этот вывод позволяет наложить некоторые ограничения на архитектуру самого координационного агента; на макромодели, описывающие реакцию агента на внешние воздействия; на организацию хранения данных - результатов работ. Поэтому следующие разделы второй главы будут посвящены этим проблемам.

2.4 Архитектура координационного агента

Как указывают Дженнингс и Вудридж [3], существует несколько подходов к определению архитектуры агента. Некоторые понимают под архитектурой агента специфическую методологию для формирования агентов. Эта методология позволяет разбить агента на набор модулей и описать их взаимодействие. Общий набор модулей и их взаимодействие должны обеспечить ответ на вопрос, каким образом внешнее воздействие и текущее внутреннее состояние агента определяют действия агента и будущее внутреннее состояние. Другие же понимает под архитектурой агента совокупность модулей программного обеспечения (или аппаратных средств), обычно изображаемых на схеме прямоугольниками со стрелками, указывающими направление потока данных и передачу управления между модулями.

Различные подходы в построении архитектуры агента могут быть классифицированы по принципу, который лежит в основе выбора агентом следующего действия. В соответствии с этим архитектуры интеллектуальных агентов делятся на три вида: делиберативная,

реактивная и гибридная. Если поведение агента базируется на некоторых рассуждениях, основанных на явном описании его среды целей, планов, то такую архитектуру называют делиберативной. Если же агент ведёт себя ситуативно, то есть действия агента предварительно запрограммированы, то говорят о реактивной архитектуре. В случае, когда при выборе дальнейшего поведения используется комбинация этих подходов, такую архитектуру называют гибридной.

Делиберативная архитектура – это классический вид архитектуры интеллектуальных агентов в области AI. Агенты с такой архитектурой применяются в системах планирования. Цель таких систем состоит в том, чтобы сгенерировать правильную последовательность действий - план, после выполнения которого будет достигнуто желаемое состояние. Существенным ограничением использования этого типа архитектуры является то, что такой агент должен обладать полным описанием состояния среды, а также эффекты действий агента должны быть известны заранее. Планировщики, которые чередуют процессы планирования и выполнения плана, достаточно гибки, так как могут модифицировать свои планы в ответ на изменения в среде. Однако методы планирования, используемые в таких системах достаточно сложны, и процесс генерации плана очень медленный, поэтому этот вид архитектуры не применяется в быстро меняющихся динамических средах.

Альтернативным типом архитектуры агентов является реактивная архитектура. Агенты с такой архитектурой принимают решение о следующих действиях на основе ограниченной информации о среде. Если в делиберативной архитектуре агенту необходимо выполнить оптимальное или правильное действие, то в реактивной архитектуре от агента требуется быстрая реакция на изменения в среде. Обычно архитектуры реактивного типа формируются из относительно простых механизмов, например, конечных автоматов.

Что касается смешанных систем, имеющих гибридную архитектуру, то они строятся на основе комбинации традиционного делиберативного и альтернативного реактивного подходов. Подобные системы, как правило, состоят из двух (или более) подсистем. Одна, из которых содержит делиберативную модель и принимает решения относительно долгосрочных целей, планов; другая – обладает реактивной моделью и реагирует на события, происходящие в среде не применяя сложных рассуждений. Часто реактивный компонент преобладает над делиберативным, это даёт возможность быстро реагировать в динамических средах.

Выбор типа архитектуры для конкретного агента зависит от роли агента в сообществе, от характеристик среды, в которой находится агент.

Учитывая специфическую роль координационного агента - обеспечивать координацию сообщества агентов - можно утверждать, что он должен иметь реактивную архитектуру. Действительно, основными задачами этого агента является обеспечение хранения полученных результатов работ и быстрая реакция на запрос некоторого результата. Выполнение подобного рода задач не требует каких-либо делиберативных действий (рассуждений), требуется лишь быстрая реакция на полученное воздействие. Кроме того, координационный агент получает достаточно небольшое количество типов допустимых воздействий, реакция на которые достаточно однообразна и может быть неявно внедрена в структуру самого агента. Поэтому координационный агент должен иметь реактивную архитектуру.

Подобная архитектура должна отражать поведение координационного агента при получении внешнего воздействия.

Принимая во внимания всё выше сказанное и используя результаты, достигнутые в [6, 10, 11], можно утверждать, что архитектура координационного агента будет содержать в себе следующие блоки (см. Рис. 2.2):

- блок взаимодействия - связи (Communication)

- блок верификации - проверки (Verification)
- блок выполнения (Macromodel Execution)
- блок накопленных знаний и опыта (Knowledge, Experience)

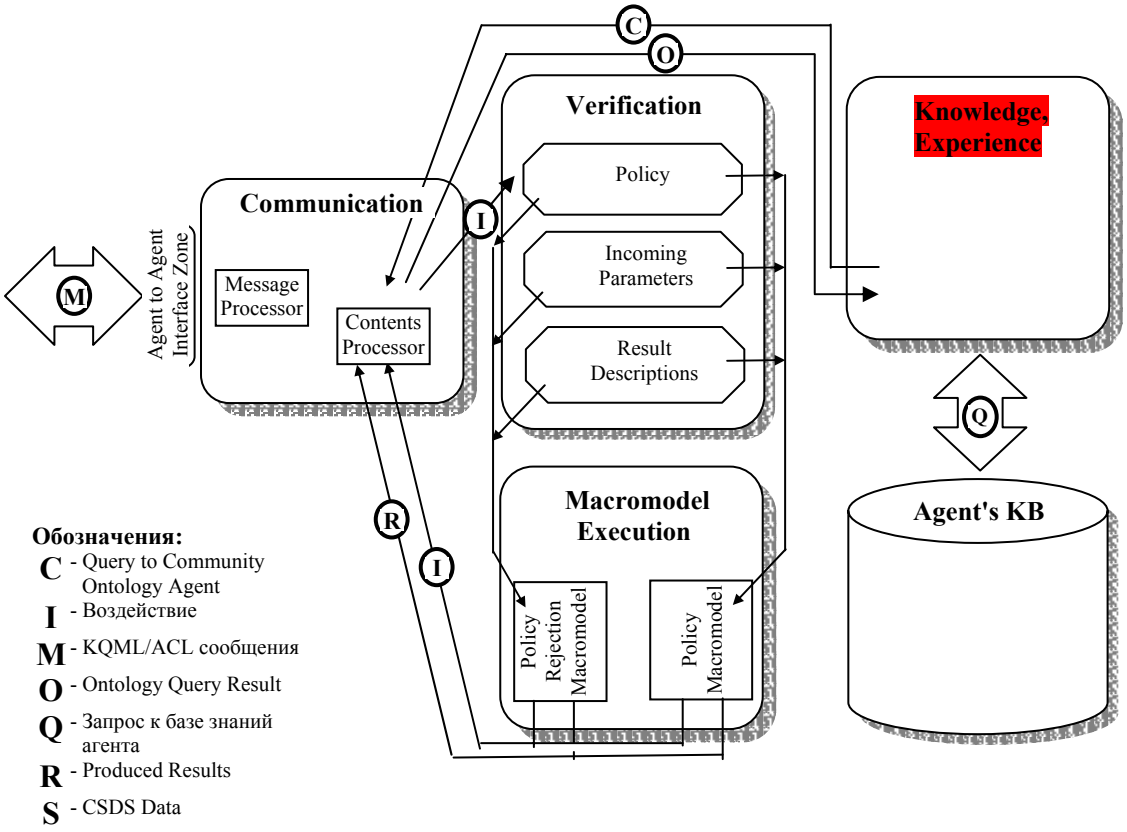

Рисунок 2.2 - Общая архитектура агента

Блок взаимодействия (Communication) отвечает за взаимодействие агента с внешним миром - с другими агентами. Кроме того, в функции этого блока входит преобразование поступившего сообщения в воздействие и, наоборот, преобразование результатов работы в сообщение.

Блок верификации (проверки) отражает работу конечных автоматов, описанных в [10]. Он проверяет полномочия агента реагировать на это воздействие, соответствие полученных параметров политике и текущему состоянию, и, наконец, формальное соответствие вектора результатов политике и состоянию агента.

Блок выполнения выполняет воздействие, прошедшее через все стадии проверки блока верификации, или генерирует отвергающую реакцию в противном случае.

Блок накопленных знаний и опыта содержит в себе информацию о предыдущей "жизни" агента, о предыдущих реакциях на воздействия. Вся эта информация хранится в базе знаний агента.

Главный поток информации и передачи управления между блоками архитектуры агента изображён на рисунке 2.2 при помощи стрелок.

Процесс смены внутренних состояний агента в этой архитектуре описан в [10].

2.5 Организация хранения результатов

2.5.1 БДКА

Как уже отмечалось выше, основной задачей координационного агента является хранение промежуточных результатов. Наиболее естественным и правильным было бы предположить, что координационный агент должен быть снабжён базой данных координационного агента, в которой он и будет хранить поступившие от других агентов результаты выполненных работ. Использование базы данных для хранения промежуточных результатов объясняется простотой реализации (так как предметная область небольшая), надёжностью хранения, удобством работы (использую SQL запросы).

2.5.2 Идентификация результата в БДКА

Выясним, какую дополнительную информацию кроме самих результатов и названия выполненной работы необходимо хранить в базе данных координационного агента, для того чтобы однозначно идентифицировать каждый поступивший результат.

Итак, для однозначной идентификации результатов выполненной работы необходимы:

- ***название работы***
- ***результаты работы***

Но если учесть что:

- Сообщество агентов может одновременно работать над выполнением нескольких общих задач (процессов), и работа с одним и тем же названием может выполняться в разных процессах, то необходимо добавить ***идентификатор процесса.***
- Работа с одним и тем же названием может выполняться в одном процессе несколько раз с разными параметрами, то необходимо добавить ***параметры выполнения работы.***
- Работа с одним и тем же названием может выполняться разными агентами с разными параметрами, то необходимо добавить ***исполнителя.***
- Работа с одним и тем же названием может поступать от разных агентов с разными (или одинаковыми, но в один и тот же момент времени) параметрами, то необходимо добавить ***заказчика.***

Кроме того, необходимыми для координационного агента могут оказаться такие параметры как:

- ***время хранения,*** для того чтобы знать какое время хранить полученный результат. По умолчанию результаты хранятся до завершения процесса, к которому они относятся. Возможно, некоторые результаты понадобится хранить дольше (например, итоговые результаты).
- ***список агентов, имеющих доступ к результату,*** для того чтобы обеспечить защиту результатов.
- ***время поступления результата,*** для служебного использования.

Итак, для однозначной идентификации результата в базе данных необходимо хранить следующую информацию:

- ***идентификатор процесса;***
- ***название работы;***
- ***результаты;***
- ***параметры выполнения работы;***
- ***исполнитель;***
- ***заказчик;***
- ***время хранения;***
- ***список агентов;***
- ***время поступления.***

2.5.3. ER-схема БДКА

Очевидно, что основным объектом в рассматриваемой предметной области является результат выполненной работы, следовательно, необходимо выделить сущность "**РЕЗУЛЬТАТ**". Эта сущность будет иметь следующие свойства: ***идентификатор результата, идентификатор процесса, идентификатор работы, параметры выполнения работы, результат работы, время хранения результата, время поступления результата.*** Идентификатор результата является ключевым полем и используется координационным агентом лишь в служебных целях. Целесообразно также в рассматриваемой предметной области выделить ещё один объект – агент. Сущность - "**АГЕНТ**" будет обладать свойствами ***идентификатор агента*** и ***имя агента.*** Отличие между именем агента и его идентификатором состоит в том, что имя агента, как отмечено в [14], это совокупность пар атрибут-значение, которые однозначно идентифицируют агента, а ключевое поле идентификатор агента - это служебный параметр, используемый координационным агентом. Возможно в будущем сущность "**АГЕНТ**" будет расширена за счёт добавления некоторых новых свойств.

Между объектами РЕЗУЛЬТАТ и АГЕНТ можно выделить три вида связи (см. Рис. 2.3): ***"Заказал выполнение", "Выполнил работу", "Имеет доступ к результатам"***.

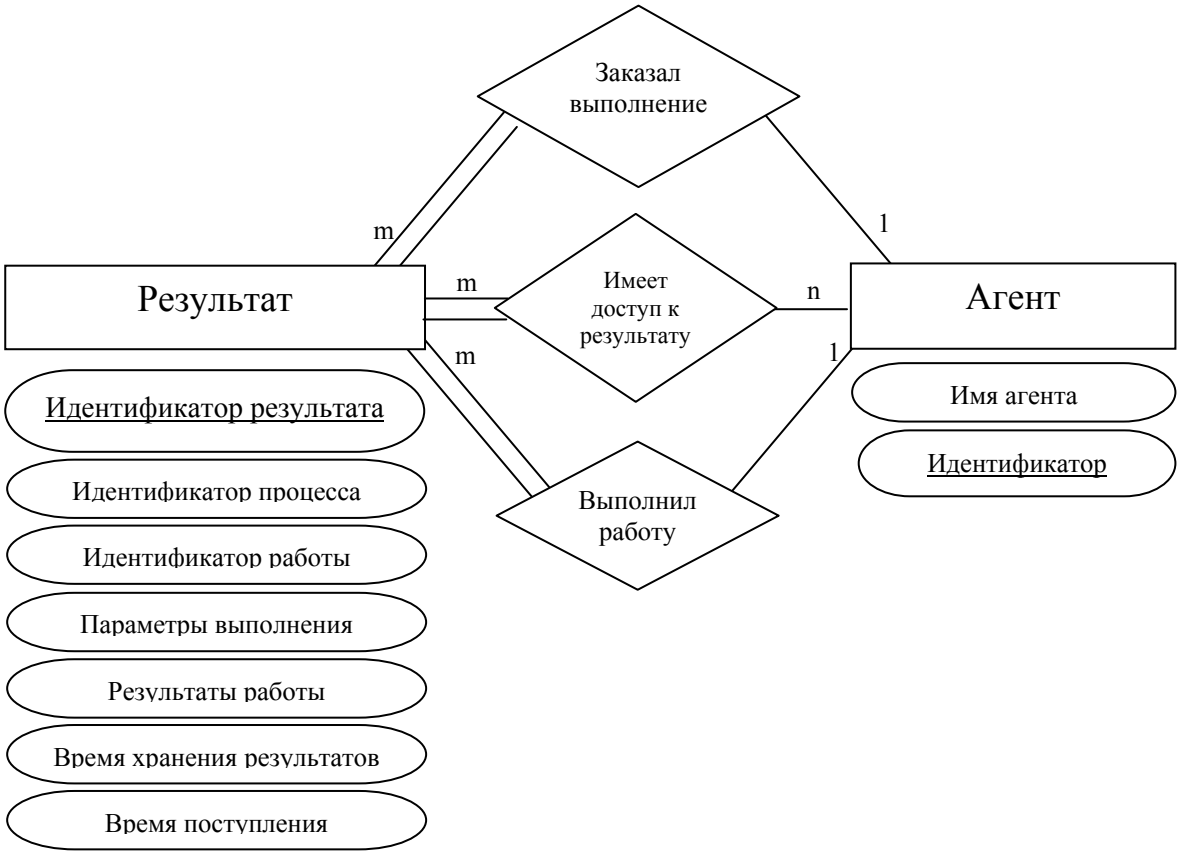

Рисунок 2.3 - ER-схема базы данных координационного

Связь "***Заказал выполнение***" имеет тип один ко многим. Действительно, один агент может заказать получение нескольких результатов, в то же время один результат может быть заказан только одним агентом. Со стороны сущности "**РЕЗУЛЬТАТ**" связь обязательная, так как получение каждого результата должно быть кем-то заказано.

Связь ***"Выполнил работу"*** также имеет тип один ко многим, потому что один агент может выполнить несколько работ, но одна работа может быть выполнена только одним агентом. Со стороны сущности "**РЕЗУЛЬТАТ**" связь обязательная, так как каждый результат должен быть кем-то выполнен.

Связь ***"Имеет доступ к результатам"*** имеет тип многие ко многим, так как каждый агент может иметь доступ к нескольким результатам, и, наоборот, к любому результату может иметь доступ несколько агентов. Причём, со стороны сущности "**РЕЗУЛЬТАТ**" связь обязательна, так как к любому результату должен иметь доступ хотя бы один агент, иначе для кого этот результат был получен.

2.6 БДКА в реляционной модели данных

Имея ER-схему базы данных координационного агента, можно осуществить переход в реляционную модель данных. Схема данных в этой модели будет состоять из трёх отношений: **"РЕЗУЛЬТАТ", "АГЕНТ", "ДОСТУП".**

Отношение **"РЕЗУЛЬТАТ"** будет иметь такую схему:

РЕЗУЛЬТАТ {идентификатор результата, идентификатор процесса, идентификатор работы, идентификатор заказчика, идентификатор исполнителя, параметры выполнения работы, результат, время хранения результата, время поступления результата}

или, что то же самое:

Result {ResultID, ProcessID, WorkID, CustomerOfWork, ExecutorOfWork, ParametersOfPerformanceOfWork, ResultsOfWork, StorageTimeOfData, TimeOfReceipt }.

Отношение **"АГЕНТ"** будет иметь такую схему:

АГЕНТ {идентификатор агента, имя агента}

или, что то же самое:

Agent {AgentID, AgentName}.

Отношение **"ДОСТУП"** будет иметь такую схему:

ДОСТУП {идентификатор результата, идентификатор агента}

или, что то же самое:

Access {<u>ResultID</u>, <u>AgentID</u>}.

Графически это можно изобразить следующим образом (типы данных из MS SQL server).

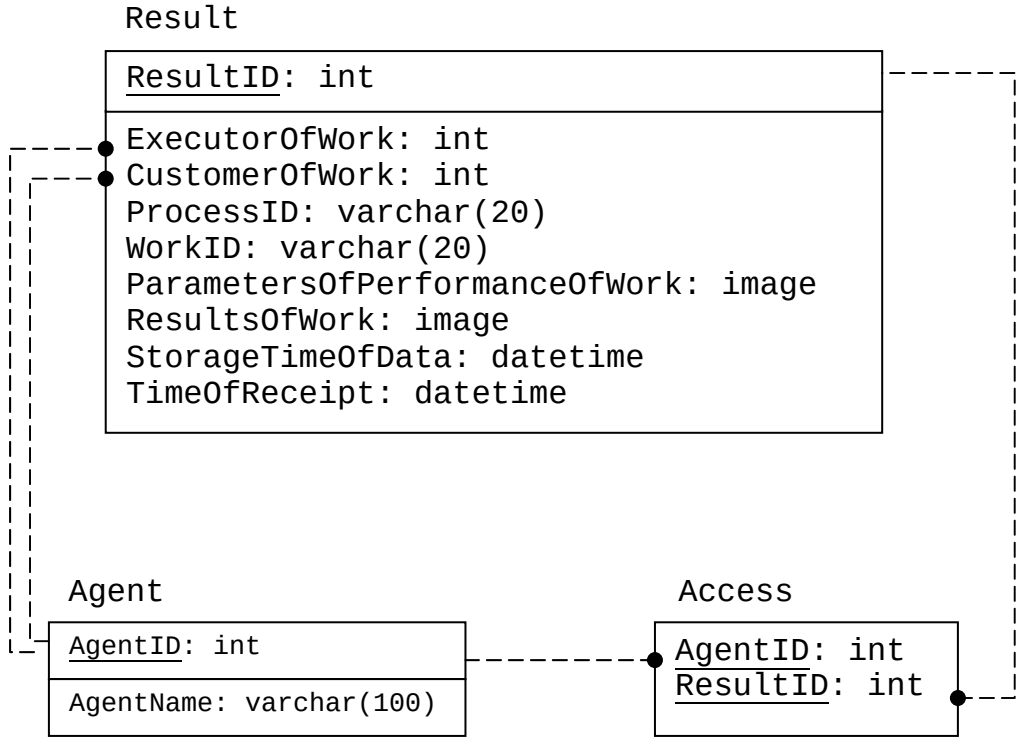

Рисунок 2.4 - БДКА в реляционной модели

2.7 Описание БДКА в MS SQL Server

Для создания базы данных координационного агента с реляционной моделью, показанной на рисунке 2.4, в MS SQL Server необходимо выполнить следующие запросы.

```
CREATE TABLE Agent (
    AgentID          int NOT NULL,
    AgentName        varchar(100) NOT NULL
)
```

```
ALTER TABLE Agent
    ADD PRIMARY KEY (AgentID)

CREATE TABLE Result (
    ResultID            int NOT NULL,
    ExecutorOfWork      int NOT NULL,
    CustomerOfWork      int NOT NULL,
    ProcessID           varchar(20) NOT NULL,
    WorkID              varchar(20) NOT NULL,
    ParametersOfPerfomanseOfWork ntext NOT NULL,
    ResultOfWork        ntext NOT NULL,
    StorageTimeOfData   datetime NULL,
    TimeOfReceipt       datetime NULL
)

ALTER TABLE Result
    ADD PRIMARY KEY (ResultID)

CREATE TABLE Access (
    ResultID            int NOT NULL,
    AgentID             int NOT NULL
)

ALTER TABLE Access
    ADD PRIMARY KEY (ResultID)

ALTER TABLE Result
    ADD FOREIGN KEY (ExecutorOfWork)
                REFERENCES Agent

ALTER TABLE Result
```

```
    ADD FOREIGN KEY (CustomerOfWork)
                REFERENCES Agent

ALTER TABLE Access
    ADD FOREIGN KEY (ResultID)
                REFERENCES Result

ALTER TABLE Access
    ADD FOREIGN KEY (AgentID)
                REFERENCES Agent
```

Полное описание БДКА для MS SQL Server дано в Приложении А.

2.8 Вывод

Результатами второй главы являются: описание модели координации сообщества с координационным агентом, выделение функций координационного агента в сообществе и типов внешних воздействий для координационного агента, предложение реактивной архитектуры координационного агента, разработка базы данных координационного агента для хранения промежуточных результатов работы сообщества.

Для более полного описания самого координационного агента, его реакции на внешние воздействия в качестве задач, которые необходимо решить в третьей главе можно выделить следующие:

- написать сообщения, обеспечивающие взаимодействие с координационным агентом на KQML с использованием KIF;
- написать алгоритмы работы с БДКА;
- написать макромодели координационного агента;
- написать общий алгоритм работы координационного агента при получении внешнего воздействия.

3 АЛГОРИТМЫ И МАКРОМОДЕЛИ КООРДИНАЦИОННОГО АГЕНТА

3.1 Языковая среда взаимодействия с координационным агентом: KQML и KIF

Теперь, когда известны типы внешних воздействий для координационного агента и схема базы данных координационного агента, можно уточнить и расширить KQML-сообщения взаимодействия с координационным агентом, приведённые в разделе 2.2.

Основными параметрами KQML-сообщения являются:

- sender - отправитель - агент, отправляющий сообщение;
- receiver - получатель - агент, которому адресовано сообщение;
- in-reply-to - идентификатор сообщения, ответом на которое является данное сообщение;
- reply-with - идентификатор, который нужно использовать в сообщении-ответе на это сообщение в качестве значения параметра in-reply-to;
- language - язык, на котором написано содержимое сообщения в наших примерах мы используем KIF - формат обмена знаниями;
- ontology - идентификатор онтологии, используемой в содержимом сообщения;
- contents - собственно содержимое сообщения.

Что касается значения параметра language в KQML-сообщениях, то мы будем использовать KIF, как наиболее универсальный формат обмена знаниями. Именно в этом формате будет написано значение параметра contents.

Под онтологией понимается формальное описание некоторой предметной области, например в формате KIF. Так как это достаточно непростая задача даже для такой небольшой предметной области как

активная доска объявлений, описание онтологии координационного агента не является задачей данной работы.

В этой работе мы будем называть онтологию, описывающую взаимодействие с координационным агентом "COA ontology", и формально записывать сообщения в KIF, использующие эту онтологию, оставляя вопрос детального описания этой онтологии открытым.

KQML-сообщение, соответствующее внешнему воздействию первого типа, можно записать следующим образом:

(ask-one

:sender **"DBA"**

:receiver **"COA"**

:in-reply-to Null

:reply-with Null

:language KIF

:ontology COA ontology

:contents (AcceptResults

(and

(= ProcessID #n_1Qxxx…x)

(= WorkID #n_2Qxxx…x)

(= Customer #n_3Qxxx…x)

(= Executor #n_4Qxxx…x)

(= Parameters #n_5Qxxx…x)

(= ResultsWork #n_6Qxxx…x)

(= StorageTime #n_7Qxxx…x)

(= ListAgents #n_8Qxxx…x)

)

)

)

Этим сообщением агент **DBA** отсылает результаты выполненной работы координационному агенту **COA**. Содержимое сообщения -

значение параметра contents, написано на языке KIF, с использованием некоторой онтологии координационного агента - COA ontology.

В содержимом этого сообщения указываются параметры выполненной работы: идентификатор процесса (ProcessID), идентификатор работы (WorkID), заказчик (Customer), исполнитель (Executor), параметры выполнения работы (Parameters), результаты работы (ResultsWork), время хранения (StorageTime), список агентов, имеющих доступ к этим результатам (ListAgents).

Запись $\#n_iQxxx\ldots x$, используемая в содержимом этого сообщения с точки зрения синтаксиса формата KIF означает последовательность символов длиной n_i (# и Q - обязательные ключевые символы, не входящие в последовательность).

Так как внешнее воздействие первого типа является директивой, то координационный агент не должен отсылать ответ (кроме, возможно, сообщения о том, что результат успешно принят).

KQML-сообщение, соответствующее внешнему воздействию второго типа, можно записать следующим образом:

(**ask-one**

:sender "PM"

:receiver **"COA"**

:in-reply-to Null

:reply-with $\widetilde{Y}(w_6)$

:language KIF

:ontology COA ontology

:contents (ProvideResults

(and

(= AgentID $\#n_1Qxxx\ldots x$)

(= ProcessID $\#n_2Qxxx\ldots x$)

(= WorkID $\#n_3Qxxx\ldots x$)

(= Customer $\#n_4Qxxx\ldots x$)

(= Executor #n_5Qxxx…x)

(= Parameters #n_6Qxxx…x)

)

)

Этим сообщением **PM**, запрашивает у координационного агента **COA**, результаты некоторой работы параметры этой работы описаны в содержимом сообщения. Смысл этих параметров такой же, как и в предыдущем сообщении.

Так как внешнее воздействие второго типа является детерминированным запросом с детерминированной реакцией, то координационный агент должен реагировать на это воздействие - отослать запрашиваемый результат. Это можно сделать при помощи следующего KQML-сообщения.

(**tell**

:sender	**"COA"**
:receiver	**"PM"**
:in-reply-to	$\widetilde{Y}(w_6)$
:reply-with	Null
:language	KIF
:ontology	COA ontology
:contents	(AcceptResults
	(= ResultsWork #n_6Qxxx…x)
	)

)

Семантика параметров этого сообщения аналогична предыдущим сообщениям.

3.2 Алгоритмы работы с БДКА

При получении внешних воздействий первого и второго типов координационный агент должен обращаться к БДКА. Так, при получении внешнего воздействия первого типа координационный агент должен добавить полученный результат в БДКА, а при получении внешнего воздействия второго типа - прочитать запрашиваемый результат из БДКА. Целью этого раздела является написание запросов для добавления нового результата в БДКА и поиск в БДКА некоторого результата.

3.2.1 Алгоритмы добавления результата в БДКА

При получении внешнего воздействия первого типа макромодель, описывающая реакцию координационного агента должна выполнить следующие действия.

1. Если одного или нескольких агентов, имена которых содержатся в параметрах воздействия - Customer, Executor, ListAgents нет в БДКА, то необходимо добавить информацию о этих агентах в таблицу Agent БДКА при помощи такого запроса (для каждого из них необходимо сгенерировать идентификатор AgentID):

   (ЗАПРОС 1 на добавление агента)

   INSERT INTO Agent (AgentID, AgentName) VALUES (AgentID1, AgentName1)

2. Добавить полученный результат в таблицу Result при помощи запроса (для этого результата необходимо сгенерировать идентификатор ResultID):

   (ЗАПРОС 2 на добавления результата)

   INSERT INTO Result (ResultID, ExecutorOfWork, CustomerOfWork, ProcessID, WorkID, ParametersOfPerformanceOfWork,

ResultsOfWork, StorageTimeOfData,
TimeOfReceipt)
VALUES (ResultID1, ExecutorOfWork1,
CustomerOfWork1, ProcessID1, WorkID1,
ParametersOfPerformanceOfWork1,
ResultsOfWork1, StorageTimeOfData1,
TimeOfReceipt1)

3. Добавить информацию о доступе к новому результату в таблицу Access при помощи запроса:

   (ЗАПРОС 3 на добавление в таблицу Access)

   INSERT INTO Access (AgentID, ResultID) VALUES (AgentID1,
   ResultID1)

3.2.2 Алгоритм получения результата из БДКА

При получении внешнего воздействия второго типа макромодель, описывающая реакцию координационного агента должна выполнить следующие действия:

1. Определить идентификатор агента, запрашивающего результат, и идентификаторы заказчика и исполнителя запрашиваемого результата при помощи запроса:

   (ЗАПРОС 4 на определение AgentID)

   SELECT Agent.AgentID
   FROM Agent
   WHERE Agent.AgentName = Name

2. Определить идентификатор запрашиваемого результата при помощи запроса:

   (ЗАПРОС 5 на определение ResultID)

   SELECT Result.ResultID
   FROM Result

WHERE (Result.ExecutorOfWork = ExecutorID) AND
(Result.CustomerOfWork = CbstomerID) AND
(Result.ProcessID = ProcessID1) AND
(Result.WorkID = WorkID1) AND
(Result.ParametersOfPerformanceOfWork = Parameters)

3. Проверить, имеет ли агент доступ к запрашиваемому результату при помощи запроса:
(ЗАПРОС 6 на проверку доступа)
SELECT COUNT(*)
FROM Access
WHERE (Accesss.AgentID = AgentID1) AND (Access.ResultID = ResultID1)

4. Если агент не имеет доступа к запрашиваемому результату, то отослать ему соответствующее сообщение, иначе - получить из таблицы Result запрашиваемый результат при помощи запроса:
(ЗАПРОС 7 на получение результата)
SELECT Result.ResultOfWork
FROM Result
WHERE Result.ResultID = ResultID1

3.3 Общий алгоритм работы КА в сообществе при получении воздействия

Роль агента - множество политик, выполняемых агентом в сообществе. Координационный агент выполняет в сообществе всего две политики, которые можно назвать так: "принять результат", "отдать результат". Используя обозначения из [10], роль координационного агента можно записать следующим образом:

$$F^{COA} = \{\text{"принять результат", "отдать результат"}\}$$

(или $F^{COA} = \{\text{" AcceptResu lt" , " ProvideRes ult" }\}$).

Множество системных параметров координационного агента может быть записано в виде:

$$X_{COA} = \{\text{AgentName, ProcessID, WorkID,}$$
$$\text{Customer, Executor, Parameters,}$$
$$\text{ResultsWork, StorageTime, ListAgents}\}.$$

Учитывая важную функцию координационного агента в сообществе - координировать сообщество агентов, необходимо отметить, что нормальная работа всего сообщества зависит от стабильной работы координационного агента. Поэтому множество внутренних состояний координационного агента должно выглядеть следующим образом:

$$S^{COA} = \{s_1, s_2\}$$

где s_1 - нормальная работа координационного агента;

s_2 - аварийная работа координационного агента.

В состоянии s_1 координационный агент выполняет все свои политики в сообществе - реагирует на все допустимые воздействия. В состоянии s_2 координационный агент не может правильно реагировать на допустимые воздействия. Это может возникнуть, например, по причине ошибки работы с БДКА, с базой знаний КА, или в следствии других объективных причин.

Рассмотрим поведение координационного агента, при получении внешнего воздействия. В соответствии с архитектурой координационного агента, описанной в разделе 2.4, процесс обработки внешнего воздействия, можно разбить на четыре этапа. Первый этап обработки внешнего

воздействия соответствует работе блока комуникации координационного агента, второй этап - работе блока проверки, третий этап - работе блока выполнения, и на четвёртом этапе происходит возвращение результатов выполненного воздействия - работа блока комуникации.

Этап 1. Получение сообщения.

Шаг 1. Процессор сообщений координационного агента получает KQML-сообщения, соответствующие внешним воздействиям первого или второго типов. Примеры таких сообщений приведены в разделе 3.1.

Шаг 2. Процессор содержимого координационного агента преобразует полученное сообщение в воздействие.

Приведём схему преобразования процессором содержимого KQML-сообщений, соответствующих допустимым внешним воздействиям первого и второго рода, в воздействия.

KQML-сообщение, приведённое в разделе 3.1, соответствующее внешнему воздействию первого типа, может быть преобразовано процессором содержимого в воздействие вида:

$$\widetilde{Y} = a(f, X, Y)$$

где:

$a(f, X, Y)$ - воздействие;

$f =$ **"принять результат"** (или $f =$ **"AcceptResults"**) - политика. $f \subseteq F^{COA}$;

$X = (\text{ProcessID}, \text{WorkID}, \text{Customer}, \text{Executor}, \text{Parameters}, \text{ResultsWork}, \text{StorageTime}, \text{ListAgents})$ - параметры выполнения воздействия;

$Y = \varnothing$ - описание запрашиваемых результатов;

$\widetilde{Y} = \varnothing$ - реакция (результаты).

$Y = \widetilde{Y} = \emptyset$, так как в терминах [6, 10, 11] внешнее воздействие первого типа является директивой.

KQML-сообщение, приведённое в разделе 3.1, соответствующее внешнему воздействию второго типа, может быть преобразовано процессором содержимого в воздействие вида:

$$\widetilde{Y} = a(f, X, Y)$$

где:

$a(f, X, Y)$ - воздействие;

$f =$ **"отдать результат"** (или $f =$ **"ProvideResults"**) - политика. $f \subseteq F^{COA}$;

$X =$ (**AgentName, ProcessID, WorkID, Customer, Executor, Parameters**) - параметры выполнения воздействия;

$Y =$ **(ResultOfWork)** - описание запрашиваемых результатов;

$\widetilde{Y} =$ **(ResultOfWork)** - реакция (результаты).

После того как процессор содержимого преобразует полученное KQML-сообщение в воздействие, переходим к следующему шагу первого этапа.

Шаг 3. Процессор содержимого координационного агента передаёт полученное воздействие блоку проверки, то есть начинается второй этап обработки внешнего воздействия.

Этап 2. Обработка воздействия в блоке проверки.

Шаг 1. Прохождение воздействия через конечные автоматы F_a, F_X, F_Y, описанные в [10].

$F_a = F_a(a, F, s_i) = \{\aleph_a, \wp_a(s_i), \Re_a, \delta_a(s_i)\}$ - конечный автомат, осуществляющий авторизацию политики внешнего воздействия a в состоянии $s_i \in S$.

Элементами этого конечного автомата являются:

$\aleph_a^{COA} = \{$**"AcceptResult","ProvideResult"**$\}$ - входной алфавит;

$\wp_a^{COA}(s_i)$ - множество состояний конечного автомата ($\wp_a^{COA}(s_1) = \{\Re\}$, $\wp_a^{COA}(s_2) = \{$**E**$\}$);

$\Re_a^{COA} = \{\Re\}$ - множество разрешающих состояний;

$\delta_a(s_i)$ - функции переходов.

Таблица переходов конечного автомата F_a^{COA} для состояния s_1 выглядит следующим образом:

Таблица 3.1

	"AcceptResult"	"ProvideResult"	
$\Re$	$\Re$	$\Re$	1

Очевидно, что если координационный агент находится в состоянии s_2 то невозможно правильное выполнение ни одной из политик. Поэтому таблица переходов конечного автомата F_a^{COA} для состояния s_2 выглядит следующим образом:

Таблица 3.2

	"AcceptResult"	"ProvideResult"	
E	**E**	**E**	0

Таким образом, конечный автомат F_a^{COA} авторизует политику внешнего воздействия только тогда, когда координационный агент

находится в состоянии s_1 и эта политика принадлежит роли F^{COA} координационного агента в сообществе.

Если конечный автомат F_a^{COA} перешёл в состояние **E** (политика внешнего воздействия не авторизована), то проверка внешнего воздействия завершена и управление переходит к третьему этапу.

После того, как политика внешнего воздействия авторизована конечным автоматом F_a^{COA}, внешнее воздействие проверяется конечным автоматом F_X^{COA} на соответствие параметров X полученного воздействия a множеству системных параметров X_{COA} и ограничениям состояния s_i координационного агента. Другими словами работа конечного автомата F_X^{COA} состоит в сравнении параметров полученного воздействия с их описанием, которое хранится в базе знаний координационного агента.

Формально этот конечный автомат записывается следующим образом:

$$F_X^{COA} = F_X(X, X_{COA}, s_i) = \{\aleph_X, \wp_X(X_{COA}, s_i), \Re_X, \delta_X(X_{COA}, s_i)\}.$$

Входной алфавит конечного автомата F_X^{COA} имеет вид:

$$\aleph_X^{COA} = \{\text{AgentName, ProcessID, WorkID,}$$
$$\text{Customer, Executor, Parameters,}$$
$$\text{ResultsWork, StorageTime, ListAgents}\}.$$

$\wp_X^{COA}(X_{COA}, s_i)$ - множество состояний конечного автомата.

$\Re_a^{COA} = \{\Re\}$ - множество разрешающих состояний;

Таблица переходов конечного автомата F_X^{COA} для состояния s_1 выглядит следующим образом:

Таблица 3.3

	AgentName	ProcessID	WorkID	Customer	Executor	Parameters	ResultsWork	StorageTime	ListAgents	
1	2	7	**E**	**E**	**E**	**E**	**E**	**E**	**E**	0
2	**E**	3	**E**	**E**	**E**	**E**	**E**	**E**	**E**	0
3	**E**	**E**	4	**E**	**E**	**E**	**E**	**E**	**E**	0
4	**E**	**E**	**E**	5	**E**	**E**	**E**	**E**	**E**	0
5	**E**	**E**	**E**	**E**	6	**E**	**E**	**E**	**E**	0
6	**E**	**E**	**E**	**E**	**E**	ℜ	**E**	**E**	**E**	0
7	**E**	**E**	8	**E**	**E**	**E**	**E**	**E**	**E**	0
8	**E**	**E**	**E**	9	**E**	**E**	**E**	**E**	**E**	0
9	**E**	**E**	**E**	**E**	10	**E**	**E**	**E**	**E**	0
10	**E**	**E**	**E**	**E**	**E**	11	**E**	**E**	**E**	0
11	**E**	**E**	**E**	**E**	**E**	**E**	12	**E**	**E**	0
12	**E**	**E**	**E**	**E**	**E**	**E**	**E**	13	**E**	0
13	**E**	**E**	**E**	**E**	**E**	**E**	**E**	**E**	ℜ	0
ℜ	ℜ	ℜ	ℜ	ℜ	ℜ	ℜ	ℜ	ℜ	ℜ	1
E	**E**	**E**	**E**	**E**	**E**	**E**	**E**	**E**	**E**	0

Очевидно, что если координационный агент находился в состоянии s_2, то процедура проверки прервется ещё до конечного автомата F_X^{COA}. Если же координационный агент перешёл в состояние s_2 непосредственно перед работой конечного автомата F_X^{COA}, то этот конечный автомат должен перейти в отвергающее состояние **E**. Поэтому таблицу переходов конечного автомата F_X^{COA} для состояния s_2 можно изобразить так:

Таблица 3.4

	AgentName	ProcessID	WorkID	Customer	Executor	Parameters	ResultsWork	StorageTime	ListAgents	
E	**E**	**E**	**E**	**E**	**E**	**E**	**E**	**E**	**E**	0

Если в процессе проверки конечный автомат F_X^{COA} перешёл в состояние **E** , то проверка внешнего воздействия завершена и управление переходит к третьему этапу.

Если в процессе проверки конечный автомат F_X^{COA} перешёл в состояние $\Re$, это означает что параметры X полученного воздействия a соответствуют множеству системных параметров X_{COA} и ограничениям состояния s_i координационного агента, после чего внешнее воздействие проверяется на соответствие описания ожидаемых результатов Y тем результатам, которые могут быть получены при выполнении координационным агентом в состоянии s_i политики f .

Формально этот конечный автомат записывается следующим образом:

$$F_Y = F_Y(Y, f, s_i) = \{\aleph_Y(f), \wp_Y(f, s_i), \Re_Y(f), \delta_Y(f, s_i)\}.$$

По сути, работа конечного автомата F_Y состоит в проверке возможности представить результаты выполнения воздействия в виде Y .

Входной алфавит этого конечного автомата соответствующий политике $f_1 =$**"AcceptResults"** является пустым множеством, так как эта политика не требует возвращения какого-либо результата. Поэтому не имеет смысла говорить о конечном автомате F_Y^{COA} для политики f_1.

Результатом выполнения внешнего воздействия второго рода должны быть запрашиваемые результаты. Для простоты будем считать,

что описание ожидаемого результата для внешнего воздействия второго рода должно быть всегда описано в виде ResultWork. Поэтому входной алфавит конечного автомата F_Y^{COA} соответствующий политике $f_2 = "\text{ProvideResults}"$ можно записать в виде:

$$\aleph_Y^{COA}(f_2) = \{\text{ResultsWork}\}.$$

Множество состояний конечного автомата для политики f_2 состоит из одного состояния $\Re$, которое и является разрешающим.

Таблица переходов конечного автомата F_Y^{COA} для политики f_2 и состояния s_1 записывается в виде:

Таблица 3.5

	ResultWork	
$\Re$	$\Re$	1

Если координационный агент находился в состоянии s_2, то процедура проверки прервется ещё до конечного автомата F_Y^{COA}. Если же координационный агент перешёл в состояние s_2 непосредственно перед работой конечного автомата F_Y^{COA}, то этот конечный автомат должен перейти в отвергающее состояние **E**. Поэтому таблицу переходов конечного автомата F_Y^{COA} для состояния s_2 можно изобразить так:

Таблица 3.6

	ResultWork	
E	**E**	0

После того, как внешнее воздействие прошло все конечные автоматы в блоке проверки, управление передаётся в блок выполнения - третьему этапу процесса обработки внешнего воздействия.

Этап 3. Выполнение.

Шаг 1. Если на втором этапе обработки внешнее воздействие было отклонено (один из конечных автоматов F_a, F_X, F_Y перешёл в состояние **E**), то в блоке выполнения координационного агента выполняется отвергающая макромодель, которая генерирует ответное воздействие - информационное сообщение с указанием причин, по которым входное воздействие было отклонено (координационный агент находится в состоянии S_2 - аварийной работы, политика воздействия не авторизована, неверно заданы входные параметры или неправильно описаны ожидаемые результаты). После чего переходим к четвёртому этапу.

Шаг 2. Если все конечные автоматы F_a, F_X, F_Y перешли в разрешающее состояние $\Re$ (внешнее воздействие можно выполнить), то в блоке выполнения вызывается макромодель, реализующая политику воздействия с данными параметрами (макромодели координационного агента, соответствующие внешним воздействиям первого и второго типов приведены в разделе 3.4). При выполнении внешнего воздействия первого типа обработка внешнего воздействия на этом завершается. Если же выполнялось внешнее воздействие второго типа, то результат выполнения передаётся процессору содержимого. После чего переходим к четвёртому этапу.

Этап 4. Отправление результатов.

Шаг 1. Процессор содержимого преобразует результат выполнения макромодели в формат KIF и формирует ответное KQML-сообщение и передаёт его процессору сообщений.

Шаг 2. Процессор сообщений отсылает ответное KQML-сообщение. Обработка внешнего воздействия завершена.

3.4 Макромодели координационного агента

3.4.1 Макромодель обработки внешнего воздействия первого типа

Параметры:

Таблица 3.7

Имя	Тип в С	Описание
ProcessID	Char*	Идентификатор процесса
WorkID	Char*	Идентификатор работы
Customer	Char*	Имя агента, заказавшего выполнение этой работы
Executor	Char*	Имя агента выполнившего работу
Parameters	Char*	Параметры выполнения работы
ResultsWork	Char*	Результат работы
StorageTime		Время хранения данных
ListAgents	Char*	Список агентов, которые могут использовать этот результат

Возвращаемого значения нет.

Используемые функции:

CurrentTime() - возвращает текущее время.

ReturnAgentID(AgentName) - функция, возвращаемая идентификатор агента, имя которого AgentName. Если такого агента нет в БДКА, то он добавляется. Эта функция использует запросы 1 и 4 из разделов 3.2.1 и 3.2.2.

NewResultID() -генерирует новый уникальный идентификатор результата.

AccessToResult(ResultID, ListAgents) - изменяет таблицу "Access" в БДКА , разрешая агентам из списка ListAgents доступ к результату с идентификатором ResultID. При этом используется запрос 3 из раздела 3.2.1.

AddResultToDB (ResultID, ProcessID, WorkID, CustomerID, ExecutorID, Parameters, ResultsWork, StorageTime, TimeReceipt) - добавляет новый результат в БД. При этом используется запрос 2 из раздела 3.2.1.

Собственно макромодель:

AcceptResults (ProcessID, WorkID, Customer, Executor, Parameters,
ResultsWork, StorageTime, ListAgents)
{
TimeReceipt = CurrentTime();
// получить идентификаторы агентов
CustomerID = ReturnAgentID(Customer)
ExecutorID = ReturnAgentID(Executor)

// получить идентификатор нового результата
ResultID = NewResultID()

// добавить результат в БД
AddResultToDB (ResultID, ProcessID, WorkID, CustomerID, ExecutorID,
Parameters, ResultsWork, StorageTime, TimeReceipt)

// добавить в БД информацию о доступе к результату
AccessToResult(ResultID, ListAgents)

}

3.4.2 Макромодель обработки внешнего воздействия первого типа

Параметры:

Таблица 3.8

Имя	Тип в С	Описание
AgentName	Char*	Имя агента, запрашивающего результаты

ProcessID	Char*	Идентификатор процесса
WorkID	Char*	Идентификатор работы
Customer	Char*	Имя агента, заказавшего выполнение работы
Executor	Char*	Имя агента выполнившего работу
Parameters	Char*	Параметры выполнения работы

Тип возвращаемого значения символьная строка (char*).

Используемые функции:

ReturnAgentID(AgentName) - функция, возвращаемая идентификатор агента, имя которого AgentName. Эта функция использует запрос 4 из раздела 3.2.2.

ReturnResultID(ProcessID, WorkID, Customer, Executor, Parameters) - функция, которая возвращает идентификатор запрашиваемого результата если он существует в БДКА, и NULL - в противном случае. При этом используется запрос 5 из раздела 3.2.2.

CheckOfAccess(ResultID, AgentID) - функция, которая возвращает TRUE если агент, идентификатор которого AgentID, имеет доступ к результату, идентификатор которого ResultID, и FALSE в противном случае. При этом используется запрос 6 из раздела 3.2.2.

ResultOfWork(ResultID) - функция, которая возвращает результат, идентификатор которого ResultID. При этом используется запрос 7 из раздела 3.2.2.

Собственно макромодель:

```
ProvideResults (AgenName, ProcessID, WorkID, Customer, Executor,
                Parameters)
{
// получить идентификаторы агентов
```

```
AgentID = ReturnAgentID(AgentName)
ExecutorID = ReturnAgentID(Executor)
CustomerID = ReturnAgentID(Customer)

// получить идентификатор запрашиваемого результата
ResultID = ReturnResultID(ProcessID, WorkID, CustomerID, ExecutorID,
                                    Parameters)
IF (ResultID == NULL) THEN
        RETURN "Запрашиваемого результата ещё нет, или неверно заданы
                параметры"
END IF

// проверка доступа
IF ( CheckOfAccess(ResultID, AgentID) == FALSE ) THEN
        RETURN "The access is forbidden"
END IF

// возвратить результат
RETURN  ResultOfWork(ResultID)
}
```

3.5 Вывод

В третьей главе были достигнуты практические результаты, касающиеся реализации координационного агента. В частности были написаны KQML-сообщения в формате KIF, которые обеспечивают взаимодействие с координационным агентом, написаны алгоритмы работы с базой данных координационного агента, разработанной во второй главе. Также в третьей главе разработан алгоритм, описывающий реакцию координационного агента на внешнее воздействие. Этот алгоритм с

небольшими изменениями может быть применён для любого агента, имеющего реактивную архитектуру предложенную в разделе 2.4. Кроме того, в этой главе были описаны макромодели координационного агента, описывающие реакцию координационного агента на внешние воздействия первого и второго типов.

ВЫВОДЫ

Данная дипломная работа посвящена разработке математической модели, архитектуры и алгоритмов координации взаимодействующих сообществ программных агентов в динамике. Актуальность работы обусловлена растущей популярностью подходов на базе концепции мультиагентских систем для проектирования и реализации сложных интеллектуальных программных систем. Особенно интересным представляется применение технологии агентов при моделировании деятельности реальных предприятий.

В процессе выполнения дипломной работы были решены следующие задачи и получены следующие результаты:

- сделан обзор и анализ существующих методов координации взаимодействующих агентов. В результате анализа сделан вывод о преимуществах использования классической доски объявлений, для координации сообщества агентов. Кроме того, в качестве механизма координации было предложено использовать активную доску объявлений - координационного агента;
- разработана модель координации взаимодействий в сообществе агентов при помощи координационного агента;
- выделены функции координационного агента в сообществе и типы внешних воздействий для координационного агента;
- сделан выбор типа архитектуры координационного агента и предложена реактивная архитектура координационного агента;
- предложена организация хранения промежуточных результатов работы сообщества, разработана схема базы данных координационного агента для хранения промежуточных результатов работы сообщества. База данных координационного агента описана в реляционной модели данных и реализована в MS SQL Server;

- разработаны алгоритмы работы с базой данных координационного агента;
- разработаны KQML-сообщения с использованием формата обмена знаниями KIF, обеспечивающие взаимодействие с координационным агентом;
- разработан алгоритм, описывающий реакцию координационного агента на внешнее воздействие. Этот алгоритм с небольшими изменениями может быть применён для любого агента, имеющего реактивную архитектуру предложенную в этой работе;
- разработаны алгоритмы макромоделей координационного агента, представляющие собой реакции (поведение) координационного агента в ответ на внешние воздействия выделенных в работе первого и второго типов.

Результаты проведенных в процессе выполнения дипломной работы исследований представляются новыми, актуальными и могут быть использованы при реализации таких сложных программных интеллектуальных систем, как динамические сообщества интеллектуальных программных агентов. Областью применения таких интеллектуальных программных систем является моделирование бизнес процессов и автоматизация деятельности реальных и виртуальных предприятий.

ПЕРЕЧЕНЬ ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

1. Roberto A. Flores-Mendez: "Towards a Standardization of Multi-Agent System Frameworks"
2. Katia P. Sycara: "Multiagent systems", AI Magazine, Vol. 10, No. 2, 1998, pp. 79-93.
3. Mike Wooldridge and Nick Jennings: "Intelligent Agents: Theory and Practice", Knowledge Engineering Review, v10n2, June 1995.
4. Rao, A. S. and Georgeff, M. P. Modeling rational agents within a BDI-architecture. In Fikes, R. and Sandewall, E., editors, Proceedings of Knowledge Representation and Reasoning (KR&R-91), 1991, pages 473–484. Morgan Kaufmann Publishers: San Mateo, CA.
5. Hyacinth Nwana, Lyndon Lee, and Nick Jennings: "Coordination in Software Agent Systems", BT Technology Journal, 14(4), 1996, pp 79-88.
6. **S. U. BORUE, V. A. ERMOLAYEV, V. A. TOLOK: APPLICATION OF DIAKOPTICAL MAS FRAMEWORK TO PLANNING PROCESS MODELLING // IN: "PROBLEMS OF PROGRAMMING" SCIENTIFIC JOURNAL №1-2, 2000, ISBN 966-02-1244-5, SPECIAL ISSUE: THE PROC. OF THE 2-ND INTL. SCIENTIFIC - PRACTICAL CONFERENCE ON PROGRAMMING (UKRPROG'2000), KIEV, 23-26 MAY 2000, P. 488-500 (**
7. "Agent Communication Language", FIPA Spec 2 - 1999
8. Yannis Labrou, Tim Finin, and Yun Peng: "Agent Communication Languages: The Current Landscape",
9. Labrou, Y. and Finin, T. A Proposal for a new KQML Specification. TR CS-97-03, Computer Science and Electrical Engineering Department, University of Maryland Baltimore County, Baltimore, February 1997.
10. С. Ю. Борю, В. А. Ермолаев, В. А. Толок: "Диакоптический подход к моделированию процессов в многофункциональных информационных системах". // "Artificial Intelligence" - a theoretical journal №2, 1999, ISSN

1561-5359, spec. issue: Proceedings of International Conference "Knowledge-Dialog-Solution" - KDS'99. Katciveli, 13-18.1999, pp.211-219

11. V. A. Ermolayev, S. U. Borue, V. A. Tolok, N. G. Keberle: "Use of Diakoptics and Finite Automata for Modelling Virtual Information Space Agent Societies", Вестник ЗГУ №1, 2000

12. M. R. Genesereth Knowledge Interchange Format. Specification. - represents the consensus of the X3T2 Ad Hoc Group on KIF as of March 1995.

13. Ferguson, I. A. TouringMachines: An Architecture for Dynamic, Rational, Mobile Agents. PhD thesis, Clare Hall, University of Cambridge, UK. (Also available as Technical Report No. 273, 1992, University of Cambridge Computer Laboratory).

14. "Agent Naming", FIPA Spec 17 -1999

ПРИЛОЖЕНИЕ А
Описание БДКА для MS SQL Server

```
CREATE TABLE Agent (
    AgentID            int NOT NULL,
    AgentName          varchar(100) NOT NULL
)
go

ALTER TABLE Agent
    ADD PRIMARY KEY (AgentID)
go

CREATE TABLE Result (
    ResultID           int NOT NULL,
    ExecutorOfWork     int NOT NULL,
    CustomerOfWork     int NOT NULL,
    ProcessID          varchar(20) NOT NULL,
    WorkID             varchar(20) NOT NULL,
    ParametersOfPerfomanseOfWork ntext NOT NULL,
    ResultOfWork       ntext NOT NULL,
    StorageTimeOfData  datetime NULL,
    TimeOfReceipt      datetime NULL
)
go

ALTER TABLE Result
    ADD PRIMARY KEY (ResultID)
go

CREATE TABLE Access (
    ResultID           int NOT NULL,
    AgentID            int NOT NULL
)
```

```
go

ALTER TABLE Access
    ADD PRIMARY KEY (ResultID)
go

ALTER TABLE Result
    ADD FOREIGN KEY (ExecutorOfWork)
                REFERENCES Agent
go

ALTER TABLE Result
    ADD FOREIGN KEY (CustomerOfWork)
                REFERENCES Agent
go

ALTER TABLE Access
    ADD FOREIGN KEY (ResultID)
                REFERENCES Result
go

ALTER TABLE Access
    ADD FOREIGN KEY (AgentID)
                REFERENCES Agent
go

create trigger tD_Agent on Agent for DELETE as
/* ERwin Builtin Tue Jun 06 12:26:10 2000 */
/* DELETE trigger on Agent */
begin
  declare  @errno   int,
           @errmsg  varchar(255)
    /* ERwin Builtin Tue Jun 06 12:26:10 2000 */
    /* Agent R/12 Result ON PARENT DELETE RESTRICT */
    if exists (
```

```
  select * from deleted,Result
  where
    /*  %JoinFKPK(Result,deleted," = "," and") */
    Result.ExecutorOfWork = deleted.AgentID
)
begin
  select @errno  = 30001,
        @errmsg = 'Cannot DELETE Agent because Result exists.'
  goto error
end

/* ERwin Builtin Tue Jun 06 12:26:10 2000 */
/* Agent R/11 Result ON PARENT DELETE RESTRICT */
if exists (
  select * from deleted,Result
  where
    /*  %JoinFKPK(Result,deleted," = "," and") */
    Result.CustomerOfWork = deleted.AgentID
)
begin
  select @errno  = 30001,
        @errmsg = 'Cannot DELETE Agent because Result exists.'
  goto error
end

/* ERwin Builtin Tue Jun 06 12:26:10 2000 */
/* Agent R/9 Access ON PARENT DELETE RESTRICT */
if exists (
  select * from deleted,Access
  where
    /*  %JoinFKPK(Access,deleted," = "," and") */
    Access.AgentID = deleted.AgentID
)
begin
  select @errno  = 30001,
```

```
        @errmsg = 'Cannot DELETE Agent because Access exists.'
    goto error
  end


  /* ERwin Builtin Tue Jun 06 12:26:10 2000 */
  return
error:
  raiserror @errno @errmsg
  rollback transaction
end
go

create trigger tU_Agent on Agent for UPDATE as
/* ERwin Builtin Tue Jun 06 12:26:10 2000 */
/* UPDATE trigger on Agent */
begin
 declare  @numrows int,
        @nullcnt int,
        @validcnt int,
        @insAgentID int,
        @errno   int,
        @errmsg  varchar(255)

 select @numrows = @@rowcount
 /* ERwin Builtin Tue Jun 06 12:26:10 2000 */
 /* Agent R/12 Result ON PARENT UPDATE RESTRICT */
 if
  /* %ParentPK(" or",update) */
  update(AgentID)
 begin
  if exists (
    select * from deleted,Result
    where
     /*  %JoinFKPK(Result,deleted," = "," and") */
```

```
    Result.ExecutorOfWork = deleted.AgentID
  )
  begin
    select @errno  = 30005,
          @errmsg = 'Cannot UPDATE Agent because Result exists.'
    goto error
  end
end

/* ERwin Builtin Tue Jun 06 12:26:10 2000 */
/* Agent R/11 Result ON PARENT UPDATE RESTRICT */
if
  /* %ParentPK(" or",update) */
  update(AgentID)
begin
  if exists (
    select * from deleted,Result
    where
      /*  %JoinFKPK(Result,deleted," = "," and") */
      Result.CustomerOfWork = deleted.AgentID
  )
  begin
    select @errno  = 30005,
          @errmsg = 'Cannot UPDATE Agent because Result exists.'
    goto error
  end
end

/* ERwin Builtin Tue Jun 06 12:26:10 2000 */
/* Agent R/9 Access ON PARENT UPDATE RESTRICT */
if
  /* %ParentPK(" or",update) */
  update(AgentID)
begin
  if exists (
```

```
    select * from deleted,Access
    where
      /*  %JoinFKPK(Access,deleted," = "," and") */
      Access.AgentID = deleted.AgentID
  )
  begin
    select @errno  = 30005,
          @errmsg = 'Cannot UPDATE Agent because Access exists.'
    goto error
  end
 end


 /* ERwin Builtin Tue Jun 06 12:26:10 2000 */
 return
error:
   raiserror @errno @errmsg
   rollback transaction
end
go

create trigger tD_Result on Result for DELETE as
/* ERwin Builtin Tue Jun 06 12:26:10 2000 */
/* DELETE trigger on Result */
begin
 declare  @errno   int,
         @errmsg  varchar(255)
   /* ERwin Builtin Tue Jun 06 12:26:10 2000 */
   /* Result R/10 Access ON PARENT DELETE RESTRICT */
   if exists (
    select * from deleted,Access
    where
      /*  %JoinFKPK(Access,deleted," = "," and") */
      Access.ResultID = deleted.ResultID
   )
```

```
    begin
      select @errno  = 30001,
            @errmsg = 'Cannot DELETE Result because Access exists.'
      goto error
    end


    /* ERwin Builtin Tue Jun 06 12:26:10 2000 */
    return
error:
    raiserror @errno @errmsg
    rollback transaction
end
go

create trigger tI_Result on Result for INSERT as
/* ERwin Builtin Tue Jun 06 12:26:10 2000 */
/* INSERT trigger on Result */
begin
  declare  @numrows int,
           @nullcnt int,
           @validcnt int,
           @errno   int,
           @errmsg  varchar(255)

  select @numrows = @@rowcount
  /* ERwin Builtin Tue Jun 06 12:26:10 2000 */
  /* Agent R/12 Result ON CHILD INSERT RESTRICT */
  if
    /* %ChildFK(" or",update) */
    update(ExecutorOfWork)
  begin
    select @nullcnt = 0
    select @validcnt = count(*)
      from inserted,Agent
```

```
      where
       /* %JoinFKPK(inserted,Agent) */
       inserted.ExecutorOfWork = Agent.AgentID
  /* %NotnullFK(inserted," is null","select @nullcnt = count(*) from inserted where"," and")
*/

  if @validcnt + @nullcnt != @numrows
  begin
    select @errno  = 30002,
          @errmsg = 'Cannot INSERT Result because Agent does not exist.'
    goto error
  end
 end

 /* ERwin Builtin Tue Jun 06 12:26:10 2000 */
 /* Agent R/11 Result ON CHILD INSERT RESTRICT */
 if
  /* %ChildFK(" or",update) */
  update(CustomerOfWork)
 begin
  select @nullcnt = 0
  select @validcnt = count(*)
   from inserted,Agent
    where
     /* %JoinFKPK(inserted,Agent) */
     inserted.CustomerOfWork = Agent.AgentID
  /* %NotnullFK(inserted," is null","select @nullcnt = count(*) from inserted where"," and")
*/

  if @validcnt + @nullcnt != @numrows
  begin
    select @errno  = 30002,
          @errmsg = 'Cannot INSERT Result because Agent does not exist.'
    goto error
  end
```

```
  end


  /* ERwin Builtin Tue Jun 06 12:26:10 2000 */
  return
error:
    raiserror @errno @errmsg
    rollback transaction
end
go

create trigger tU_Result on Result for UPDATE as
/* ERwin Builtin Tue Jun 06 12:26:10 2000 */
/* UPDATE trigger on Result */
begin
  declare  @numrows int,
           @nullcnt int,
           @validcnt int,
           @insResultID int,
           @errno   int,
           @errmsg  varchar(255)

  select @numrows = @@rowcount
  /* ERwin Builtin Tue Jun 06 12:26:10 2000 */
  /* Result R/10 Access ON PARENT UPDATE RESTRICT */
  if
    /* %ParentPK(" or",update) */
    update(ResultID)
  begin
    if exists (
      select * from deleted,Access
      where
        /*  %JoinFKPK(Access,deleted," = "," and") */
        Access.ResultID = deleted.ResultID
    )
```

```
  begin
    select @errno  = 30005,
          @errmsg = 'Cannot UPDATE Result because Access exists.'
    goto error
  end
end

/* ERwin Builtin Tue Jun 06 12:26:10 2000 */
/* Agent R/12 Result ON CHILD UPDATE RESTRICT */
if
  /* %ChildFK(" or",update) */
  update(ExecutorOfWork)
begin
  select @nullcnt = 0
  select @validcnt = count(*)
    from inserted,Agent
      where
        /* %JoinFKPK(inserted,Agent) */
        inserted.ExecutorOfWork = Agent.AgentID
  /* %NotnullFK(inserted," is null","select @nullcnt = count(*) from inserted where"," and")
*/

  if @validcnt + @nullcnt != @numrows
  begin
    select @errno  = 30007,
          @errmsg = 'Cannot UPDATE Result because Agent does not exist.'
    goto error
  end
end

/* ERwin Builtin Tue Jun 06 12:26:10 2000 */
/* Agent R/11 Result ON CHILD UPDATE RESTRICT */
if
  /* %ChildFK(" or",update) */
  update(CustomerOfWork)
```

```
  begin
    select @nullcnt = 0
    select @validcnt = count(*)
      from inserted,Agent
        where
          /* %JoinFKPK(inserted,Agent) */
          inserted.CustomerOfWork = Agent.AgentID
    /* %NotnullFK(inserted," is null","select @nullcnt = count(*) from inserted where"," and")
*/

    if @validcnt + @nullcnt != @numrows
    begin
      select @errno  = 30007,
            @errmsg = 'Cannot UPDATE Result because Agent does not exist.'
      goto error
    end
  end


  /* ERwin Builtin Tue Jun 06 12:26:10 2000 */
  return
error:
    raiserror @errno @errmsg
    rollback transaction
end
go

create trigger tI_Access on Access for INSERT as
/* ERwin Builtin Tue Jun 06 12:26:10 2000 */
/* INSERT trigger on Access */
begin
  declare  @numrows int,
           @nullcnt int,
           @validcnt int,
           @errno   int,
```

```
        @errmsg  varchar(255)

  select @numrows = @@rowcount
  /* ERwin Builtin Tue Jun 06 12:26:10 2000 */
  /* Result R/10 Access ON CHILD INSERT RESTRICT */
  if
    /* %ChildFK(" or",update) */
    update(ResultID)
  begin
    select @nullcnt = 0
    select @validcnt = count(*)
      from inserted,Result
        where
          /* %JoinFKPK(inserted,Result) */
          inserted.ResultID = Result.ResultID
    /* %NotnullFK(inserted," is null","select @nullcnt = count(*) from inserted where"," and")
*/

    if @validcnt + @nullcnt != @numrows
    begin
      select @errno  = 30002,
            @errmsg = 'Cannot INSERT Access because Result does not exist.'
      goto error
    end
  end

  /* ERwin Builtin Tue Jun 06 12:26:10 2000 */
  /* Agent R/9 Access ON CHILD INSERT RESTRICT */
  if
    /* %ChildFK(" or",update) */
    update(AgentID)
  begin
    select @nullcnt = 0
    select @validcnt = count(*)
      from inserted,Agent
```

```
        where
         /* %JoinFKPK(inserted,Agent) */
         inserted.AgentID = Agent.AgentID
    /* %NotnullFK(inserted," is null","select @nullcnt = count(*) from inserted where"," and")
*/

    if @validcnt + @nullcnt != @numrows
    begin
      select @errno  = 30002,
            @errmsg = 'Cannot INSERT Access because Agent does not exist.'
      goto error
    end
  end


  /* ERwin Builtin Tue Jun 06 12:26:10 2000 */
  return
error:
    raiserror @errno @errmsg
    rollback transaction
end
go

create trigger tU_Access on Access for UPDATE as
/* ERwin Builtin Tue Jun 06 12:26:10 2000 */
/* UPDATE trigger on Access */
begin
  declare  @numrows int,
           @nullcnt int,
           @validcnt int,
           @insResultID int,
           @errno   int,
           @errmsg  varchar(255)

  select @numrows = @@rowcount
```

```
/* ERwin Builtin Tue Jun 06 12:26:10 2000 */
/* Result R/10 Access ON CHILD UPDATE RESTRICT */
if
  /* %ChildFK(" or",update) */
  update(ResultID)
begin
  select @nullcnt = 0
  select @validcnt = count(*)
    from inserted,Result
      where
        /* %JoinFKPK(inserted,Result) */
        inserted.ResultID = Result.ResultID
  /* %NotnullFK(inserted," is null","select @nullcnt = count(*) from inserted where"," and")
*/

  if @validcnt + @nullcnt != @numrows
  begin
    select @errno  = 30007,
          @errmsg = 'Cannot UPDATE Access because Result does not exist.'
    goto error
  end
end

/* ERwin Builtin Tue Jun 06 12:26:10 2000 */
/* Agent R/9 Access ON CHILD UPDATE RESTRICT */
if
  /* %ChildFK(" or",update) */
  update(AgentID)
begin
  select @nullcnt = 0
  select @validcnt = count(*)
    from inserted,Agent
      where
        /* %JoinFKPK(inserted,Agent) */
        inserted.AgentID = Agent.AgentID
```

```
   /* %NotnullFK(inserted," is null","select @nullcnt = count(*) from inserted where"," and")
*/

   if @validcnt + @nullcnt != @numrows
   begin
     select @errno  = 30007,
           @errmsg = 'Cannot UPDATE Access because Agent does not exist.'
     goto error
   end
  end


  /* ERwin Builtin Tue Jun 06 12:26:10 2000 */
  return
error:
   raiserror @errno @errmsg
   rollback transaction
end
go
```