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 МОДЕЛІ КОМУНІКАЦІЇ ПРОГРАМНОГО АГЕНТУ У МУЛЬТІ-АГЕНТНИХ СИСТЕМАХ**

Виконав Михайличенко Володимир Володимирович

ст. групи 8226-2 _____ В. В. Михайличенко
(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

2

РЕФЕРАТ

Дипломная работа: 45 с., 4 рис., 2 приложения, 12 источников.

Объект исследования - мультиагентные системы.

Цель работы - исследовать и провести анализ методов коммуникации взаимодействующих агентов. Разработать и реализовать модель коммуникации сообщества программых агентов при моделировании деятельности реальных предприятий.

Методы исследования - описательный, семантический, сравнительный.

Данная дипломная работа ориентирована на коммуникацию интеллектуальных программых агентов используемых при моделировании деятельности реального предприятия. Задачей данной работы является реализация библиотеки функцию на языке программирования C++, обеспечивающей работу с KQML-сообщениями.

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

СОДЕРЖАНИЕ

ПЕРЕЧЕНЬ ПРИНЯТЫХ 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

ВВЕДЕНИЕ

В настоящее время исследователи уделяют пристальное внимание проблемам и методам, связанным с моделированием виртуальных предприятий. Повышенное внимание к этим проблемам объясняется как ростом процессов распределения и виртуализации производственных и бизнес процессов (хорошим примером может служить стремительное развитие интернет-магазинов, других средств оказания реальных услуг виртуальными методами, например, электронной коммерции и др.), так и необходимостью придания существующим моделям автоматизированного управления реальными предприятиями недостающего в них динамизма.

Большинство современных систем, которые применяются при разработке виртуальных предприятий и интернет-комерции, строятся с использованием концепции и технологии мультиагентных систем. Технология мультиагентных систем заключается в том, что такая система состоит из нескольких полностью автономных программных агентов. Одной из основных проблем, возникающих при этом, является проблема организации взаимодействия агентов. От выбора модели коммуникации программных агентов зависит, насколько эффективно будет работать вся система в целом.

Настоящая квалификационная работа бакалавра посвящена алгоритмической и программной реализации модели коммуникации интеллектуальных программных агентов в динамических сообществах, ориентированых на выполнение потоков работ. Актуальность работы обусловлена растущей популярностью мультиагентных систем при проектировании сложных информационных систем.

Основными зазачами, решаемыми в данной работе, являются: сделан

- обзор и анализ существующих моделей коммуникации программных агентов;
- обзор и сравнительный анализ языков коммуникации программных агентов: KQML и FIPA ACL, выбор базового языка коммуникации для данной работы;
- анализ и уточнение модели коммуникации, основанной на использовании параметрических обратных связей при получении и обработке результатов;
- сравнительный анализ существующего программного обеспечения для построения программных агентов и мультиагентных систем;
- разработка алгоритмы выполнения базовых коммуникативных актов для использованной модели коммуникации;
- реализация библиотеки функций написанная на языке программирования C++, которая обеспечивает работу с сообщениями интеллектуальных программных агентов.

1 МУЛЬТИАГЕНТНЫЕ СИСТЕМЫ И МЕТОДЫ КОММУНИКАЦИИ

1.1 Развитие мультиагентных систем. Основные понятия

Системы управления, использующие компьютерную обработку данных, такие как, системы управления крупными предприятиями, системы безопасности, системы управления космическими аппаратами с каждым днем становятся все более сложными. В связи с этим возникла необходимость разработать новую технологию построения таких систем, потому что, системы с централизованным управлением и политикой планирования уже не в состоянии адекватно работать в средах с быстро меняющимися показателями.

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 [2]. Традиционно, DAI разделен на две поддисциплины: распределенная система обработки (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]. 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;
- вычисление асинхронно.

Как известно, при построении больших информационных систем наиболее мощными средствами для обработки сложности являются модульность и абстракция. 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 [3].

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) [4]. 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). 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.

Кроме того, реальные проблемы имеются в распределенных, открытых системах. Открытая система - система, в которой структура 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 возможности требуют, чтобы агенты были способны взаимодействовать друг с другом. Кроме того, эти возможности позволят агентам увеличивать множество решаемых ими задач.

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

Итак, использование MAS при построении сложных информационных систем позволяет:

- решить проблемы, которые являются слишком большими для централизованного агента;
- решить проблемы, связанные с ограниченными ресурсами или явного наличия критических параметров эффективности;
- сохранять темп деловой пригодности и непротиворечивости. 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.

- решить проблемы, которые могут естественно быть расценены как сообщество автономных взаимодействующих агентов (например, управление воздушным движением, моделирование деятельности предприятия, и т. п.);
- обеспечить решения проблем, которые эффективно используют информационные источники, которые пространственно распределены. Примерами таких областей могут быть сети датчиков, сейсмический контроль, информация, накапливающаяся из интернет.
- обеспечить решения в ситуациях, где обработка, знания распределены. Примеры таких областей включают параллельную разработку, здравоохранение и производство.
- 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 необходимая информация обменивается среди агентов;
    - *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 Коммуникация и ее роль МАС

Взаимодействие - одна из наиболее важных характеристик агентов. Агенты взаимодействуют, чтобы совместно использовать информацию, выполнять задачи, достигать общих целей. Genesereth и Ketchpel [5] утверждают, что способность связываться на языке взаимодействия агентов (ACL) - единственная характеристика, которая отличает агентов от другого программного обеспечения. Механизмы связи налагают дополнительные ограничения на внутреннюю архитектуру агента, которая должна быть способной эффективно их использовать [5, 6].

Фаруди и Грэхэм [3] выделяют три типа взаимодействия, которые могут использоваться в МАС:

- Непрямой (косвенный) обмен сообщениями (indirect message passing);
- Прямое взаимодействие, используя API или Remote Procedure Call (RPC)
- Используя общедоступную память, например доску объявлений

Они утверждают, что взаимодействие агентов при помощи обмена сообщениями будет более гибким, чем общедоступная память (с проблемами параллельного доступа и семафорами) и RPC (который ограничивает гибкость во время выполнения), потому что оно позволяет

агентам свободно связываться и быть распределёнными. Кроме того, этот вид взаимодействия обеспечивает гибкость линий связи.

1.2.1 Средства коммуникации агентов

К средствам взаимодействия агентов относятся языки взаимодействия агентов (FIPA ACL, KQML). Язык взаимодействия агентов обеспечивает обмен знаниями и информацией между агентами. FIPA ACL в отличии от таких средств как RPC, RMI, CORBA, обеспечивающих обмен информацией между приложениями, имеет более сложную семантику [6]. Кроме того, ACL обладает следующими преимуществами по сравнению с CORBA[4]:

- FIPA ACL управляет суждениями, правилами, действиями, а не семантически не связанными объектами.
- Сообщения FIPA ACL описывает ожидаемое состояние, а не процедуру или метод.

Однако ACL не охватывают полный спектр объектов, которыми могли бы обмениваться агенты, например планы, цели, опыт, стратегии.

На техническом уровне, при использовании ACL, агенты транспортируют сообщения по сети, используя протоколы низшего уровня, например SMTP, TCP/IP, POP3, или HTTP.

Язык взаимодействия агентов (ACL) должен позволять передавать информацию любого вида между различными агентами. Имеются два подхода к проектированию языков взаимодействия агентов:

- ***Процедурный***, включает обмен процедурными директивами/командами. Это может быть реализовано используя такие языки программирования как Java или Tcl.

- ***Декларативный***, где связь основана на декларативных инструкциях, типа определений, предположений, знаний, и т.п.

Из-за ограничений на процедурные подходы (например, такие сценарии трудно координировать, объединить), декларативные языки были предпочтены для создания языков взаимодействия агентов. Одни из наиболее популярных декларативных языков - KQML со своими диалектами и FIPA ACL [7].

1.2.1.1 KQML

KQML[8] - высокоуровневый, яык обмена информацией между агентами независимый от синтаксиса содержимого и онтологии. Другими словами KQML не зависит от механизма транспортировки (например TCP/IP, SMTP, IIOP), от содержательных языков (например KIF, SQL, STEP, Prolog)[8] и от онтологии.

Концептуально в KQML-сообщении можно выделить три уровня [8]: уровень связи (который описывает параметры связи низкого уровня, тип отправителя, получателя и идентификаторы связи), уровень сообщения (который содержит собственно сообщение и протокол интерпретации) и содержательный уровень (который содержит информацию, имеющую отношение к запрошенному действию).

KQML-сообщение представляет собой непозиционированый список ключевых слов и их значений. На рисунке 1.1 приведён пример KQML-сообщения.

```
(register
    :sender       Agent1
    :receiver     Agent2
    :language AnyLanguage
    :content (
                Select result (
                AgentID = #00086A
                ProcessID = #00450P
                WorkID = #009W
                Customer = #00080A
                Executor = #00081A
                Parameters ="1.25, 0.0"
                                        )
                            )
              )
```

Рисунок 1.1 – Пример KQML-сообщения

Подобное сообщение вполне может получить агент в реально существующей МАС. В этом сообщении агент Agent1 запрашивает у агента Agent2 результаты некоторой работы. Параметры запрашиваемого результата описаны в пункте content на некотором семантическом языке агента (AnyLanguage).

Как уже отмечалось выше, KQML налагает стандартный синтаксис на все сообщения. Этот стандартный синтаксис состоит из расширяемого набора ключевых слов, которые имеют четкую семантику. Семантика протокола связи зависит от цели сообщения. Например, сообщение, которое используется при назначении работы, имеет отличную семантику, чем сообщение, используемое в контексте распределения ресурсов. Сообщения одного и того же типа должны использовать общую

семантику. Агенты могут поддерживать лишь некоторое подмножество полного набора типов сообщений. Кроме того, они могут поддерживать некоторый свой, нестандартный тип сообщений.

1.2.1.2 FIPA ACL

Другим наиболее прменимым языком взаимодействия агентов является FIPA ACL.

FIPA ACL как и KQML основан на теории речевых актов[9]: сообщения предназначены для выполнения некоторых действий на основании содержимого сообщения [9]. FIPA спецификация ACL состоит из набора типов сообщений и описания их прагматики. FIPA ACL подобен KQML. Его синтаксис идентичен синтаксису KQML, если бы не различные имена для некоторых зарезервированных примитивов. KQML и FIPA ACL идентичны относительно основных концепций и принципов. Они отличаются, прежде всего, их семантическими структурами [9].

Пример сообщения написанного на FIPA ACL приведен на рис. 1.2.

1.3 Обзор программного обеспечения для реализации коммуникации в мультиагентных системах

В данный момент проблемой построения мультиагентных систем и в частности построением агентов в отдельности занимаются все больше и больше лабораторий, рабочих групп и университетов. Одним из лидеров в этой области Стенфордский университет. Сотрудники этого университета разработали пакет для построения программных агентов: JATLite.

```
(request
   :sender Agent1
   :receiver Agent2
   :content (
            Select result (
                        AgentID = #00086A
                        ProcessID = #00450P
                        WorkID = #009W
                        Customer = #00080A
                        Executor = #00081A
                        Parameters ="1.25, 0.0"
                        )
              )
   :language AnyLanguage)
)
```

Рисунок 1.2 – Пример FIPA ACL-сообщения

Но в то же время существуют и другие пакеты: Magenta, набор библиотек для языка программирования C++.

1.3.1 JATLite

JATLite (Java Agent Template, Lite) – это пакет программ, написанных на языке программирования Java, который позволяет быстро, просто и удобно создавать новых программных агентов, которые могут взаимодействовать посредством сети Интернет. JATLite позволят агентам

выполнять следующие функции: регистрироваться в Agent Message Router, используя имя агента и пароль, посылать и получать сообщения по сети Интернет, пересылать и получать файлы, взаимодействовать с программами, которые работают на компьютере агента или на любом другом компьютере в сети [10].

JATLite особенно удобно использовать для построения агентов которые используют в качестве коммуникационного языка KQML. В качестве транспортного протокола JATLite использует один из открытых протоколов: TCP/IP, SMTP или FTP. Однако разработчик может использовать и любой другой протокол для передачи данных.

JATLite содержит набор шаблонов агентов с помощью которых довольно легко строить приложения основанные на технологии агентов. Легкость построения приложений обеспечивается тем, что программист использует предопределенные классы в языке программирования JAVA. Это удобно тем что, если разработчик не использует какие либо классы, то JATLite, после компилляции, просто не включит эти классы в исполняемый код.

Недостатком JATLite является то, что эта система изначально не предназначена для построения интеллектуальных агентов.

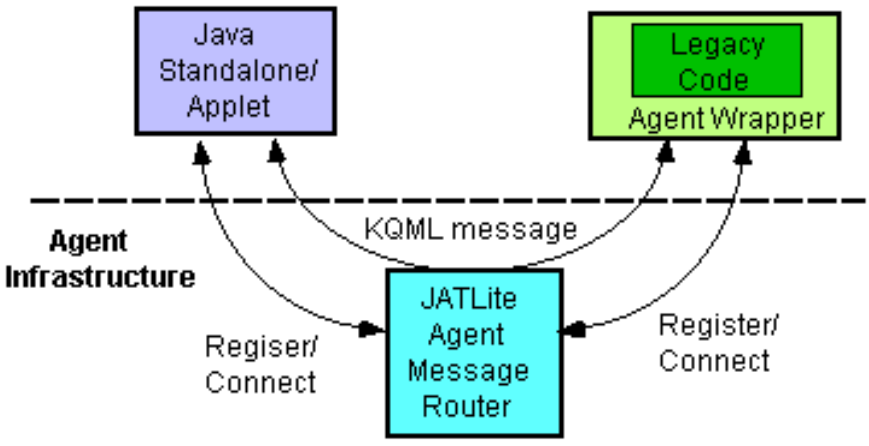

Рисунок 1.3 – Схема приложения написанного при помощи системы JATLite (приводится по [10])

Стандартное приложение, написанное при помощи системы JATLite представляет собой приложение, схему которого иллюстрирует рисунок 1.3. JATLite позволяет агентам взаимодействовать с Agent Message Router, который осуществляет буферизацию сообщений а так же следит за присутствием агентов в сети, отслеживая их IP адреса.

JATLite обеспечивает стандартное программное обеспечение для коммуникации между агентами. Общепринятыми стандартами для обмена сообщениями между агентами являются KQML и FIPA ACL.

JATLite был протестирован при построении мультиагентной системы задачей которой являлся анализ, координация и управление производственным процессом.

JATLite легко интегрируется с уже существующими приложениями написанными на языке программирования JAVA, однако он также легко интегрируется в программы написанные на C++ и Лисп.

Возможности предоставляемые системой JATLite:

- Модульная конструкция, с системой уровней, которые могут быть заменены без изменения всего кода агента
- Использование в качестве низкоуровневого протокола передачи данных TCP/IP, который поддерживается в большинстве операционных систем (Unix, Windows, Mac OS и т.д.), однако можно легко добавить протоколы более высокого уровня (например, e-mail)
- В качестве языка общения между агентами используется KQML с встроенным синтаксическим анализатором сообщений. Передаваемые данные могут быть описаны на SQL, Express, KIF
- Использование Message Router для обеспечения регистрации и связи агентов, с использованием имени и пароля агента

- Обеспечение хранения и организация очереди сообщений для мобильных агентов
- Поддержка автономных агентов написаных на JAVA, C++, а так же аплетов, используя WWW-браузер (Netscape Communicator, Internet Explorer)
- Встроенная поддержка FTP-протокола (протокола передачи файлов)

Архитектура JATLite организована в виде уровней, в следствии чего разработчик при построении системы может выбирать тот уровень который ему необходим, например, разработчик хочет использовать в своей системе протокол TCP/IP, но не хочет использовать KQML, тогда он может ограничится только абстрактным и базовым уровнями (Рисунок 1.4).

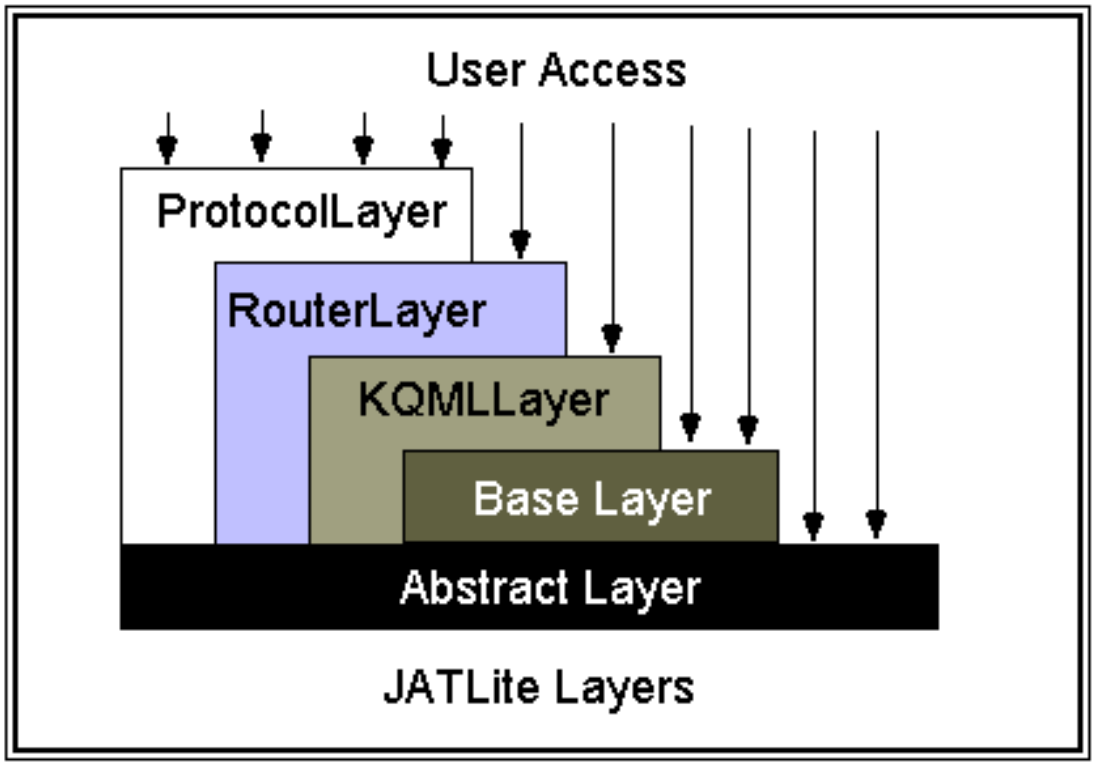

Рисунок 1.4 – Уровни системы JATLite (приводится по [10])

Описание уровней системы JATLite:

- Абстрактный уровень содержит в себе набор базовых абстрактных классов необходимых для работы агента. Хотя считается что протоколом низкого уровня является TCP/IP, расширяя и изменяя классы абстрактного уровня можно добавлять и другие протоколы, например, UDP
- Основной уровень обеспечивает основные методы коммуникации используя TCP/IP и абстрактный уровень. На этом уровне не существует ни каких ограничений на язык сообщений агентов. Расширяя и изменяя классы этого уровня можно добавить, например работы с файлами (весь поток сообщений будут писаться в файл), обеспечить агента способностью одновременно принимать более одного сообщения и т. д.
- KQML уровень обеспечивает хранение и синтаксический анализ KQML-сообщений
- Уровень маршрутизации обеспечивает хранение и организацию очереди сообщений, а так же их маршрутизацию. Все агенты в системе построенной при помощи JATLite посылают и получают сообщения через Message Router, который, в случае краха агента, получает и хранит все сообщения которые были посланы этому агенту до тех пор пока агент снова не будет запущен.
- Уровень протокола отвечает за реализацию протоколов передачи данных высокого уровня (SMTP, POP3, HTTP, FTP и т. д.)

1.4.1 Magenta

Magenta (версия написанная на языке программирования C++) – это библиотека функций котороая обеспечивает коммуникацию агентов

работающих в разнородных системах. Magenta обеспечивает коммуникацию по стандарту ACL (это значит, что она поддерживает KIF и KQML). Хотя разработчик может библиотеку функций для построения модели коммуникации один к одному, предпочтительнее использовать ее в системах с координационным агентом [10].

Однако необходимо подчеркнуть, что данная библиотека функций поддерживает и другие модели коммуникации: клиент/сервер и один-к-одному. Другой отличительной особенностью является то, что разработчик может сам расширять и изменять данную библиотеку по своему усмотрению, обеспечивая тем самым ее гибкость для построения систем различного назначения.

1.4 Вывод

В этой главе была рассмотрена история развития мультиагентных систем, сделан обзор существующего пргограммного обеспечения с помощью которых строятся мультиагентные системы.

Определены задачи и роль коммуникации в мультиагентных системах.

Рассмотрено два языка обмена сообщениями между агентами: KQML и FIPA ACL. Для построения модели коммуникации, мы выбрали KQML, как наиболее удобный для нас.

Теперь когда определена роль коммуникации и выбран язык обмена сообщениями между агентами, задачей следующей главы будет:

- Выбрать наиболее оптимальную модель программного агента
- Рассмотреть коммуникацию агентов на уровне коммуникативных актов
- Выбрать язык с помощью которого будет описываться содержимое сообщений

2 МОДЕЛЬ ВЫПОЛНЕНИЯ КОММУНИКАТИВНЫХ АКТОВ

2.1 Архитектура программного агента

Как говорится в работе Дженнингса и Вудриджа [5], существует несколько понятий архитектуры агента. Некоторые понимают под архитектурой агента специфическую методологию для формирования агентов. Эта методология позволяет разбить агента на набор модулей и описать взаимодействие между модулями. 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, обычно изображаемых на схеме прямоугольниками и стрелками, указывающими направление потока данных и передачу управления между модулями.

Существуют три вида архитектуры программных агентов:

- Делиберативная
- Реактивная
- Гибридная

Если логика работы агента базируется на рассуждениях, целях и планах агента, то такую архитектуру называют делиберативной. Если же действия агента заранее запрограммированы, то архитектуру такого агента называют реактивной. В случае, когда при выборе дальнейшего поведения используется комбинация этих подходов, такую архитектуру называют гибридной.

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

Принимая во внимания всё выше сказанное и используя результаты, достигнутые в [6, 10, 11], можно утверждать, что архитектура программного агента будет содержать в себе следующие блоки (см. Рис. 2.1):

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

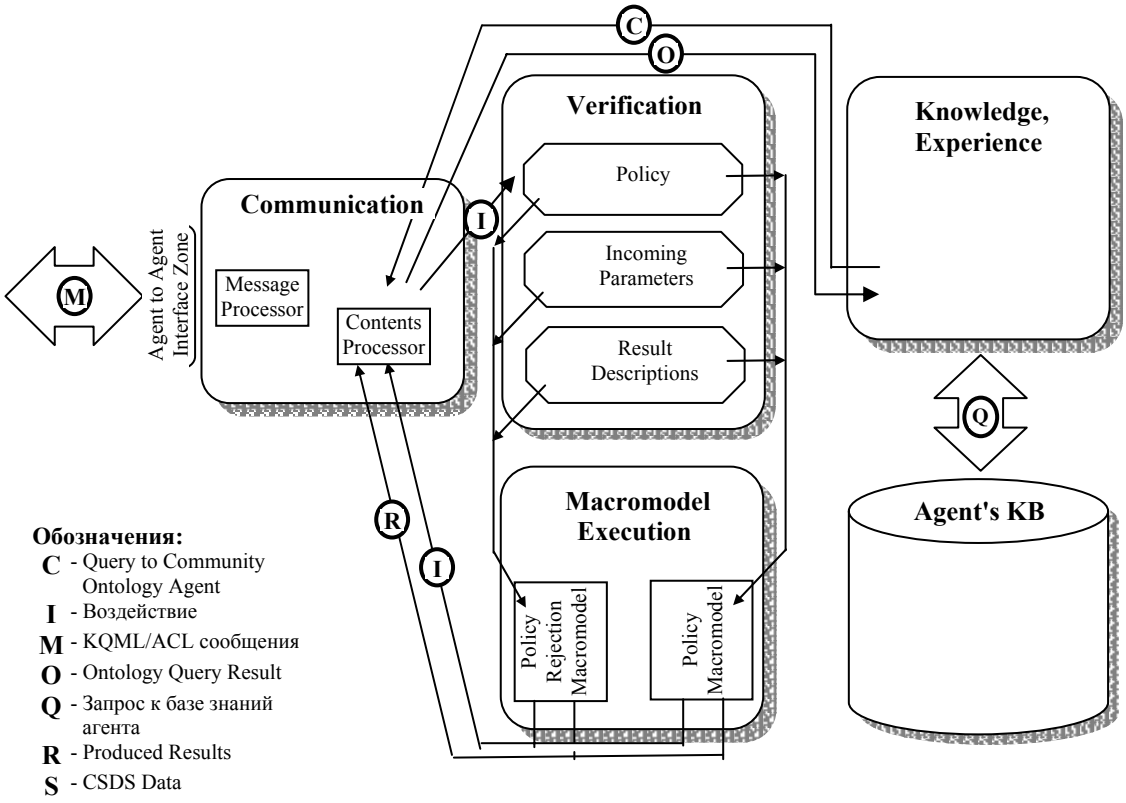

Рисунок 2.1 – Общая структура программного агента
(приводится по [5])

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 [13]. 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 2.1 при помощи стрелок.

2.1 Базовые типы коммуникативных актов

Если не вникать в проблему семантического соответствия коммуникации агентов, то можно выделить следующие типы связей:

- *директива* – безоговорочное выполнение действия подчиненным агентом (рисунок 2.2).
- *детерминированный запрос с детерминированной реакцией* – посылая сообщение агент хочет, что бы ему возвратили какие-либо результаты (рисунок 2.3).
- *детерминированный запрос с оптимизацией результата* – после получения результатов запроса, агент, перед использованием результатов, должен их сначала оптимизировать (рисунок 2.4).

- *недетерминированный запрос с оптимизацией результата* – используется, когда заранее не известен агент получатель сообщения. После того как сообщение будет послано всем агентам внутри системы и получены от каждого такого агента результаты, будет выбран лучший (рисунок 2.5).

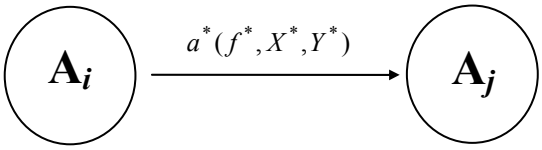

Рисунок 2.2 – Директива

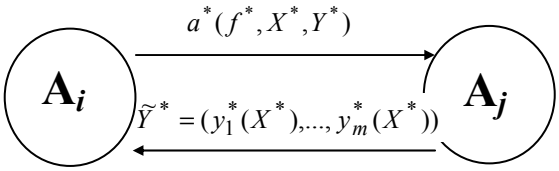

Рисунок 2.3 - детерминированный запрос с детерминированной реакцией

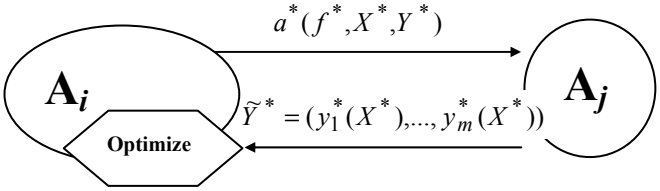

Рисунок 2.4 - детерминированный запрос с оптимизацией результата

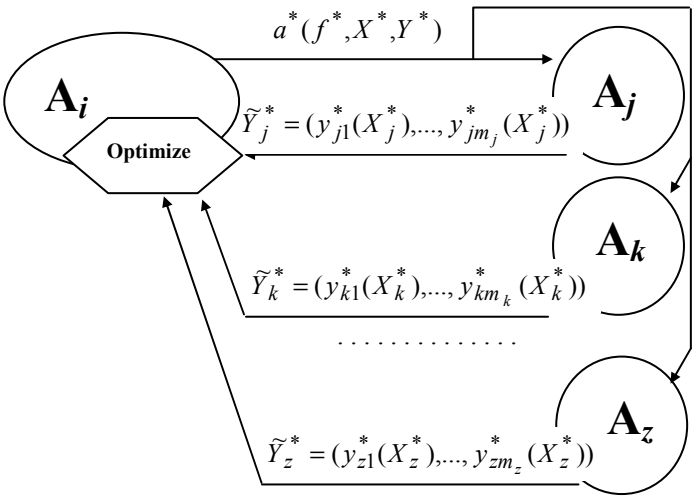

Рисунок 2.5 - недетерминированный запрос с оптимизацией результата

Если представить внешнее воздействие на сообщество и реакцию агента на это воздействие в виде следующего соотношения:

$$\tilde{Y} = a(f, X, Y) \qquad (2.1)$$

где: $a(f, X, Y)$ - воздействие, f - политика, $X = (x_1, ..., x_n)$ - параметры,

$Y = (y_1(X), ... y_m(X))$ - описание запрашиваемых результатов,

$\tilde{Y} = (\tilde{y}_1(X), ..., \tilde{y}_k(X))$ - реакция (результаты)

и, кроме того, взаимодействие агентов внутри сообщества, в виде:

$$\tilde{Y}^* = a^*(f^*, X^*, Y^*) \qquad (2.2)$$

то:

- для воздействия вида «директива» будет справедливо:

$$X^* \subseteq X, \quad Y^* = \tilde{Y}^* = \emptyset, \quad f^* \subseteq f; \qquad (2.3)$$

- для воздействия вида «детерминированный запрос с детерминированной реакцией» будет справедливо:

$$X^* \subseteq X, \quad Y^* = \tilde{Y}^*, \quad f^* \subseteq f; \qquad (2.4)$$

- для воздействия вида «детерминированный запрос с оптимизацией результата» будет справедливо:

$$X^* \subseteq X, \quad Y^* \subseteq \tilde{Y}^*, \quad f^* \subseteq f; \qquad (2.5)$$

- для воздействия вида «недетерминированный запрос с оптимизацией результата» будет справедливо:

$$X^* = X, \qquad Y^* = \tilde{Y}^*, \qquad f^* \equiv f; \tag{2.6}$$

Выше приведенные уравнения позволяют описать любое воздействие на программного агента в мультиагентной системе.

2.3 Язык описания содержимого сообщений KIF

Knowledge Interchange Format (KIF) – машинно-ориентированный язык, предназначенный для обмена знаниями между гетерогенными системами.

KIF не предназначен для использования в качестве конечного языка при ведении диалога с пользователем, хотя иногда он используется для этих целей. Он также не предназначен для хранения знаний внутри компьютерных систем. Если система использует для своей работы знания, описанные на KIF, то она перед тем как работать с ними конвертирует знания во внутренний формат системы, а уже при передачи знаний другим системам или компонентам внутри данной системы, конвертирует знания из внутреннего формата в KIF.

По своей логике и принципу использования KIF похож на язык Postscript. Однако, он имеет несколько особенностей:

- KIF имеет декларативную семантику. Можно понять смысл выражения написанного на KIF без использования интерпретатора, этим он отличается от таких языков программирования как Emycin and Prolog.
- KIF позволяет описать любое предложение с использованием исчисления предикатов.

На основании всего выше перечисленного мы выбрали KIF в качестве языка описания контекста сообщений, посылаемыми программными агентами в мультиагентной системе.

Описание синтаксиса языка KIF приведено в Приложении А.

2.4 Вывод

Результатами второй главы являются: описание модели коммуникации мультиагентной системы, выделение четырех базовых типов коммуникативных актов, с помощью которых возможно реализовать более сложные коммуникативные акты, предложение реактивной архитектуры программного агента, выбран язык написания содержимого сообщений.

В качестве задач для следующей главы можно выделить следующее:

- Написать алгоритм выделения данных из KQML-сообщения
- Написать алгоритм преобразования данных в KQML-сообщение для последующей отправки его другому агенту
- Реализовать выше названные алгоритмы в виде библиотеки функций на языке программирования C++

3 АЛГОРИТМЫ И ПРОГРАММЫ ВЫПОЛНЕНИЯ КОММУНИКАТИВНЫХ АКТОВ

Практической частью дипломной работы являлось разработка библиотеки функций написанных на языке программирования C++, которая бы обеспечивала агента возможностью работать с KQML-сообщениями.

Используя данную библиотеку функций можно осуществлять следующие преобразования:

- Формирование KQML-сообщения с требуемым содержанием.
- Выделения из KQML-сообщения данных необходимых агенту.

3.1 Структуры данных

Для хранения KQML-сообщения используется следующая структура данных:

```
struct Tkqml
{
char sender[50];
char receiver[50];
char replay_with[50];
char language[50];
char content[50];
};
```

В данной структуре поле *sender* содержит имя агента который посылает сообщение, *receiver* – имя агента получателя. Поле *replay_with* необходимо для идентификации сообщения, после того как агент номер

один послал сообщение агенту номер 2 с определенным значением поля *replay_with*, агент номер два должен ответить на сообщение агента номер два сообщением с точно таким же значением поля *replay_with*. В поле *language* хранится идентификатор языка на котором написано содержание сообщения, хранящегося в поле *content*.

3.2 Алгоритмы

В данной главе будут приведены алгоритмы функций, реализованных в библиотеке функций. Ниже приведен алгоритм функции, которая формирует KQML-сообщение используя данный контекст:

Шаг 1. Получить входные данные (поля структуры Tkqml: sender, receiver, reply_with, content, languages).

Шаг 2. Сформировать KQML-сообщение.

Шаг 3. Отдать сформированное KQML-сообщение, в качестве выходных данных функции.

Алгоритм функции выделения контекста из KQML-сообщения:

Шаг 1. Получить KQML-сообщение.

Шаг 2. Найти в KQML-сообщении поле sender и сохранить его значение.

Шаг 3. Если в KQML-сообщении такого поля не оказалось, то в качестве значания использовать NULL.

Шаг 4. Найти в KQML-сообщении поле receiver и сохранить его значение.

Шаг 5. Если в KQML-сообщении такого поля не оказалось, то в качестве значания использовать NULL.

Шаг 6. Найти в KQML-сообщении поле content и сохранить его значение.

Шаг 7. Если в KQML-сообщении такого поля не оказалось, то в качестве значания использовать NULL.

Шаг 8. Найти в KQML-сообщении поле language и сохранить его значение.

Шаг 9. Найти в KQML-сообщении поле reply_with и сохранить его значение.

Шаг 10. Если в KQML-сообщении такого поля не оказалось, то в качестве значания использовать NULL.

Шаг 11. Заполнить структуру типа Tkqml и отдать ее в качестве выходных данных функции.

3.3 Описание библиотеки функций

Библиотека содержит в себе две функции, которые осуществляют преобразование данных в KQML-сообщение и наоборот.

Функция *kqmltokif* выделяет из полученного KQML-сообщения данные. Функция *kiftokqml* осуществляет обратное преобразование.

Описание функций:

- *Tkqml *kqmltokif(char *msg)* – в качестве параметра данной функции передается текст KQML-сообщения, а результат выполнения помещается в переменную типа *Tkqml*.
- *char *kiftokqml(Tkqml *msg)* – в качестве параметра передается переменная типа *Tkqml*, где указывается имя агента, который посылает сообщение, имя агента получателя, и содержание сообщения написанное на языке KIF.

3.4 Пример программы, использующей данную библиотеку функций

Пример программы которая использую данную библиотеку хранящуюся в файле с именем “*lib.h*” выполняет сначала преобразование KQML-сообщения, выделяя из него данные, а затем осуществляет преобразование данных в KQML-сообщение:

```
#include “lib.h”
void main()
{
Tkqml *kqml1;
char *msg1="(sender: agent1
            reciver: agent2
            replay-with: test
            language: kif
            content: (code msg on kif))";
char *msg2;
kqml1=kqmltokif(msg1);
msg2=kiftokqml(kqml1);
};
```

Полный текст библиотеки функций приведен в Приложении Б.

3.5 Вывод

Разработанные алгоритмы и библиотека программ на языке C++ реализует все базовые коммуникативные акты модели коммуникации интеллектуальных программных агентов, рассмотренной в главе 2.

Разработанная библиотека функций может быть в дальнейшем использована для программной реализации интерфейсов коммуникации агент - агент в сообществах интеллектуальных программных агентов, реализуемых на базе подхода, предложенного в [9].

ВЫВОДЫ

Данная квалификационная работа посвящена алгоритмической и программной реализации модели коммуникации интеллектуальных программных агентов в динамических сообществах, ориентированых на выполнение потоков работ. Актуальность работы обусловлена растущей популярностью мультиагентных систем при проектировании сложных информационных систем. Особенно интересным представляется применение технологии агентов при моделировании деятельности реальных предприятий, где одной из основных задач является построение и реализация оптимальной модели коммуникации.

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

- сделан обзор и анализ существующих моделей коммуникации программных агентов;
- сделан обзор и сравнительный анализ языков коммуникации программных агентов: KQML и FIPA ACL, в качестве базового языка коммуникации для данной работы выбран язык KQML.
- в результате сделанного обзора была выбрана модель коммуникации основанная на использовании параметрических обратных связей при получении и обработке результатов;
- произведен обзор существующего программного обеспечения для построения программных агентов и мультиагентных систем.
- разработаны алгоритмы выполнения базовых коммуникативных актов для использованной модели коммуникации;
- Реализована библиотека функций написанная на языке программирования C++, которая обеспечивает работу с сообщениями программных агентов написанных на языке KQML.

ПЕРЕЧЕНЬ ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Katia P. Sycara: "Multiagent systems", *AI Magazine*, Vol. 10, No. 2, 1998, pp. 79-93.
2. Mike Wooldridge and Nick Jennings: "Intelligent Agents: Theory and Practice", Knowledge Engineering Review, v10n2, June 1995.
3. 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.
4. Borue, S. U., Ermolayev, V. A. Tolok V. A. : "Application of diakoptical MAS framework to planning process modelling" Submitted to UkrPROG'2000, May 23-26, 2000, Kiev, Ukraine.
5. "Agent Communication Language", FIPA Spec 2 - 1999
6. Yannis Labrou, Tim Finin, and Yun Peng: "Agent Communication Languages: The Current Landscape",
7. 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.
8. 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
9. 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

10. http://java.stanford.edu/index2.htm

11. http://logic.stanford.edu/software/magenta/magentacpp.html

12. J. R. Searle: Speech Act, Cambridge University Press, 1969.

ПРИЛОЖЕНИЕ А

Синтаксис языка KIF

Символы.

A B CDE FG H IJK L MNO PQ R ST U VW X Y Z

a b cde fg h ijk l mno pq r st u vw x y z

0 1 234 56 7 89( ) []{ }< >

= + -*/ \& ^ ~'` " _@# $% : ;, . !?

Структуры KIF.

<word> ::= примитивный объект

<expression> ::= <word> | (<expression>*)

<indvar> ::= слово начинающееся с символа ?

<seqvar> ::= слово начинающееся с символа @

<termop> ::= listof | setof | quote | if | cond | the | setofall | kappa | lambda

<sentop> ::= = | /= | not | and | or | => | <= | <=> | forall | exists

<ruleop> ::= =>> | <<= | consis

<defop> ::= defobject | defunction | defrelation | := | :=> | :&

<objconst> ::= слово описывающее объект

<funconst> ::= слово описывающее функцию

<relconst> ::= слово описывающее отношение

<logconst> ::= слово описывающее логическое выражение

Переменные, операторы и константы.

<variable> ::= <indvar> | <seqvar>

<operator> ::= <termop> | <sentop> | <ruleop> | <defop>

<constant> ::= <objconst> | <funconst> | <relconst> | <logconst>

Объекты, функции и отношения.

<term> ::= <indvar> | <objconst> | <funconst> | <relconst>|
<funterm> | <listterm> | <setterm> |
<quoterm> | <logterm> | <quanterm>
<listterm> ::= (listof <term>* [<seqvar>])
<setterm> ::= (setof <term>* [<seqvar>])
<funterm> ::= (<funconst> <term>* [<seqvar>])
<quoterm> ::= (quote <expression>)
<logterm> ::= (if <sentence> <term> [<term>]) | (cond (<sentence>
<term>) ... (<sentence> <term>))
<quanterm> ::= (the <term> <sentence>)| (setofall <term> <sentence>)|
(kappa (<indvar>* [<seqvar>]) <sentence>*)| (lambda (<indvar>*
[<seqvar>]s) <term>)

ПРИЛОЖЕНИЕ Б

Исходный текст библиотеки функций для работы с KQML-сообщениями

```
// Подключение внешних библиотек необходимых для работы
#include <stdio.h>
#include <conio.h>
#include <string.h>
#include <stdlib.h>

// Описание структуры данных предназначенной для хранения
// KQML-сообщения
struct Tkqml
{
char sender[50];          // имя агента посылающего сообщение
char reciver[50];         // имя агента получателя сообщения
char replay_with[50];     // тема сообщения
char language[50];        // язык на котором написанно содержание
                          // сообщения
char content[50];         // содержание сообщения
};

// Объявление функций для работы с KQML-сообщениями
Tkqml *kqmltokif(char *msg);        // Функция выделения
                                    // содержимого из
                                    // KQML-сообщения
```

```
char *kiftokqml(Tkqml *msg);        // Функция преобразования
                                    // содержимого в
                                    // KQML-сообщение


Tkqml *kqmltokif(char *msg)
{
int n=1;
Tkqml *kqml=new Tkqml;
while (msg[n]!=' ')
        n++;
n++;
strcpy(kqml->sender, "");
while (msg[n]!=' ')
        {
        strncat(kqml->sender, &msg[n], 1);
        n++;
        }
strcat(kqml->sender, "\0");
n++;
while (msg[n]!=' ')
        n++;
n++;
strcpy(kqml->reciver, "");
while (msg[n]!=' ')
        {
        strncat(kqml->reciver, &msg[n], 1);
        n++;
        }
strcat(kqml->reciver, "\0");
```

```
n++;
while (msg[n]!=' ')
        n++;
n++;
strcpy(kqml->replay_with, "");
while (msg[n]!=' ')
        {
        strncat(kqml->replay_with, &msg[n], 1);
        n++;
        }
strcat(kqml->replay_with, "\0");
n++;
while (msg[n]!=' ')
        n++;
n++;
strcpy(kqml->language, "");
while (msg[n]!=' ')
        {
        strncat(kqml->language, &msg[n], 1);
        n++;
        }
strcat(kqml->language, "\0");
n++;
while (msg[n]!=' ')
        n++;
n++;
strcpy(kqml->content, "");
while (msg[n]!='\0')
        {
        if (msg[n+2]=='\0')
```

```
                {
                strncat(kqml->content, &msg[n+1], 1);
                break;
                }
        strncat(kqml->content, &msg[n], 1);
        n++;
        }
strcat(kqml->content, "\0");
return (kqml);
}


//----------------------------

char *kiftokqml(Tkqml *msg)
{
char *kqml= (char *) malloc(200);
strcpy(kqml, "(");
strcat(kqml, "sender: ");
strcat(kqml, msg->sender);
strcat(kqml, " reciver: ");
strcat(kqml, msg->reciver);
strcat(kqml, " replay-with: ");
strcat(kqml, msg->replay_with);
strcat(kqml, " language: ");
strcat(kqml, msg->language);
strcat(kqml, " content: ");
strcat(kqml, msg->content);
strcat(kqml, ")\0");
return (kqml);}
```