**Федеральное государственное автономное образовательное учреждение   
высшего образования**

**КАЗАНСКИЙ (ПРИВОЛЖСКИЙ) ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ**

**ВЫСШАЯ ШКОЛА ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И   
ИНФОРМАЦИОННЫХ СИСТЕМ**

Направление подготовки: 09.04.04 - Программная инженерия

Профиль: Разработка программно-информационных систем

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА**

**РЕАЛИЗАЦИЯ ИСПОЛНИТЕЛЬНОЙ ЧАСТИ СИСТЕМЫ ПОВЫШЕНИЯ ЭФФЕКТИВНОСТИ ИТ-СЛУЖБЫ ПРЕДПРИЯТИЯ**

**Работа завершена:**

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_2017 г.

Студент группы 11-511 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ И.Р.Валеев

**Работа допущена к защите:**

Научный руководитель

Кандидат технических наук,

руководитель группы программистов

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_2017 г. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А.С.Тощев

Директор Высшей школы ИТИС

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_2017 г. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А.Ф. Хасьянов

Казань - 2017 г.

**Оглавление**

[Словарь терминов 4](#_Toc485830242)

[Введение 11](#_Toc485830243)

[1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ РАБОТЫ 13](#_Toc485830244)

[1.1. Постановка задачи дипломной работы 13](#_Toc485830245)

[1.2. Обзор существующих систем и их сравнительный анализ 14](#_Toc485830246)

[1.3. Модель теории массового обслуживания ИТ инфраструктуры 16](#_Toc485830247)

[1.4. Вывод 17](#_Toc485830248)

[2. СУЩЕСТВУЮЩИЕ МОДЕЛИ ИНТЕЛЛЕКТУАЛЬНОГО ПОИСКА 18](#_Toc485830249)

[2.1. Обзор модели IDP 18](#_Toc485830250)

[2.2. Обзор модели Menta 0.1 18](#_Toc485830251)

[2.3. Обзор модели Menta 0.2-0.3 19](#_Toc485830252)

[2.3.1. MentaController 19](#_Toc485830253)

[2.3.2. SolutionGenerator 20](#_Toc485830254)

[2.3.3. SolutionChecker 20](#_Toc485830255)

[2.3.4. ReasonerAdapter 20](#_Toc485830256)

[2.3.5. Translator 20](#_Toc485830257)

[2.3.6. Applicator 21](#_Toc485830258)

[2.3.7. KBServer 21](#_Toc485830259)

[2.4. Обзор модели TU 21](#_Toc485830260)

[2.4.1. Критик 22](#_Toc485830261)

[2.4.2. Селектор 23](#_Toc485830262)

[2.4.3. Образ мышления 23](#_Toc485830263)

[2.4.4. Уровни мышления 24](#_Toc485830264)

[2.5. Вывод 28](#_Toc485830265)

[3. ОБЗОР ПРИНЦИПА ДЕЙСТВИЯ TU 31](#_Toc485830266)

[3.1. Архитектура системы TU 31](#_Toc485830267)

[3.2. Принцип работы компонент TU 31](#_Toc485830268)

[3.3. Компоненты системы TU 33](#_Toc485830269)

[3.3.1. TU WebService 33](#_Toc485830270)

[3.3.2. MessageBus 33](#_Toc485830271)

[3.3.3. DataService 33](#_Toc485830272)

[3.3.4. CoreService 33](#_Toc485830273)

[3.3.5. KnowledgeBaseAnnotator 41](#_Toc485830274)

[3.3.6. Reasoner 42](#_Toc485830275)

[3.3.7. DataService 43](#_Toc485830276)

[3.3.8. TU Knowledge 44](#_Toc485830277)

[3.4. Реализация исполнительной части системы TU 47](#_Toc485830278)

[3.4.1. Требования к ClientAgent: 47](#_Toc485830279)

[3.4.2. Выбор технологии для реализации 48](#_Toc485830280)

[3.4.3. ClientAgent 49](#_Toc485830281)

[3.4.4. Взаимодействие компонент ClientAgent 58](#_Toc485830282)

[3.5. Вывод 61](#_Toc485830283)

[4. РЕЗУЛЬТАТЫ РАБОТЫ 64](#_Toc485830284)

[4.1. Результаты 64](#_Toc485830285)

[4.2. Тесты скорости работы системы и оценка производительности 64](#_Toc485830286)

[4.3. Вывод 67](#_Toc485830287)

[Заключение 68](#_Toc485830288)

[Список использованных источников 70](#_Toc485830289)

[Приложение 1. Исходный код 71](#_Toc485830290)

# Словарь терминов

|  |  |
| --- | --- |
| Термин | Значение |
| .Net Core | .NET Core – это модульная реализация, которая может использоваться широким набором вертикалей, начиная с дата-центров и заканчивая сенсорными устройствами, доступная с открытым исходным кодом, и поддерживаемая Microsoft на Windows, Linux и MacOS. |
| Bool | Тип данных. |
| CMD | Интерпретатор командной строки для операционных систем семейства Windows |
| Critic | Основан на определении Марвина Мински. Класс объектов, которые выступают триггерами при наступление определенного события. |
| Deliberative | Уровень рассуждений. |
| EntityFramework | Объектно-ориентированная технология Microsoft для доступа к данным. |
| FTP | Стандартный протокол, предназначенный для передачи файлов по TCP-сетям. |
| HTTP | Протокол прикладного уровня передачи данных. |
| HTTPS | Расширение протокола HTTP, для поддержки шифрования в целях повышения безопасности. |
| Instinctive | Инстинктивный уровень. |
| JRE | Минимальная реализация виртуальной машины, необходимая для исполнения Java-приложений, без компилятора и других средств разработки. |
| LINQ | Проект Microsoft по добавлению синтаксиса языка запросов, напоминающего SQL, в языки программирования платформы .NET Framework. |
| Learned | Уровень обученных реакций. |
| Link | Тип возвращаемых данных. Если метод данных не возвращает, то ничего не указывается. |
| Link DomainModel:SemanticNetwork | Описание класса, где DomainModel — сам класс, а SemanticNetwork — класс-родитель. |
| List<> | Тип данных. |
| Log4J | Библиотека журналирования Java программ. |
| Log4Net | Порт фреймворка для логирования log4j на платформу Microsoft .NET Framework. |
| MONO | Проект по созданию полноценного воплощения системы .NET Framework на базе свободного программного обеспечения. |
| OWL | Язык онтологии для интернета на основе XML/Web стандарта |
| PowerShell | Расширяемое средство автоматизации от Microsoft с открытым исходным кодом. |
| Reflective | Рефлексивный уровень. |
| Selector | Компонент, отвечающий за выборку данных из Базы Знаний. |
| Self-Conscious Reflection | Самосознательный уровень. |
| Self-Reflective | Саморефлексивный уровень. |
| SOAP | Протокол обмена структурированными сообщениям. |
| SQLite | Компактная встраиваемая реляционная база данных. |
| String | Тип данных. |
| TLC | Thinking Life Cycle. |
| TU | Сокращение от ThinkingUnderstanding. |
| ThinkingLifeCycle | Основан на определении Марвина Мински. Класс объектов, которые выступают основными объектами для запуска в приложении - рабочими процессами. |
| ThinkingUnderstanding | Система, созданная в рамках работы. Дословный перевод «Мышление-Понимание». |
| Tuple | Тип данных. |
| WSDL | Язык описания веб-сервисов. |
| WayToThink | Путь мышления. Основан на определении Марвина Мински. Класс объектов, которые модифицируют данные. |
| Wine | Программное обеспечение, позволяющее пользователям UNIX - подобных операционных систем архитектуры x86 исполнять ОС семейства Windows |
| База Знаний | База данных приложения, представленная в виде онтологии знаний. |
| БД | Сокращение от База Данных. |
| Вопросно-ответная система | Информационная система, способная принимать вопросы и отвечать на них на естественном языке, другими словами, это система с естественно-языковым интерфейсом. |
| Знания эвристические | Знания, накапливаемые интеллектуальной системой в процессе ее функционирования, а также знания, заложенные в ней априорно, но не имеющие статуса абсолютной истинности в данной проблемной области. |
| Интеллектуальная система | Система или устройство с программным обеспечением, имеющие возможность с помощью встроенного процессора настраивать свои параметры в зависимости от состояния внешней среды. |
| Искусственная нейронная сеть | Математическая модель, а также её программное или аппаратное воплощение, построенная по принципу организации и функционирования биологических нейронных сетей - сетей нервных клеток живого организма. |
| Искусственный интеллект | Раздел информатики, занимающийся вопросами имитации мышления человека с помощью компьютера. |
| Искусственный разум | Гипотетическая техническая система, способная обнаруживать свойства, идентичные разумному мышлению и поведению человека. |
| Критик | Вероятностный предикат. Датчик, который срабатывает при определенных условиях. После активации определенным событием критик с помощью своего предиката проверяет, реагирует ли система на это событие или нет. |
| Модель мышления | Модель, описывающая алгоритм обработки входных данных для принятия решения. |
| Модель мышления Марвина Мински | Модель принятия решений в основе которой лежит модель общения нейронов в головном мозге. |
| ОС | Сокращение от операционная система обеспечение. Комплекс взаимосвязанных программ, предназначенных для управления ресурсами компьютера и организации взаимодействия с пользователем. |
| ПО | Сокращение от программное обеспечение. Множество программ, используемых для управления компьютером. |
| Подсистема объяснений | Программа, позволяющая пользователю получить ответы на вопросы: - "Как было получено то или иное решение?"; обычно ответ на этот вопрос представляет собой трассировку всего процесса вывода решения с указанием использованных фрагментов базы знаний; и -"Почему было принято такое решение?"; обычно ответ на этот вопрос есть ссылка на умозаключение, непосредственно предшествовавшее полученному решению. |
| Подсистема приобретения знаний | Программа, предназначенная для корректировки и пополнения базы знаний. Подсистема приобретения знаний - в простейшем случае - интеллектуальный редактор базы знаний. Подсистема приобретения знаний - в более сложных случаях - средства для извлечения знаний: - из баз данных; - из неструктурированного текста; - из графической информации и т.д. |
| Селектор | Компонент работы с данными, производит подготовку запроса для выборки информации. |
| Система основанная на знаниях | Система искусственного интеллекта, в которой предметные знания представлены в явном виде и отделены от прочих знаний системы. |
| Теория принятия решений | Область исследования, вовлекающая понятия и методы математики, статистики, экономики, менеджмента и психологии с целью изучения закономерностей выбора людьми путей решения проблем и задач, а также способов достижения желаемого результата. |
| Экспертная система | Система имитирующая поведение человека-эксперта. Включает в себя знания об определенной слабо структурированной и трудно формализуемой узкой предметной области. В основе экспертных систем лежит один из механизмов логического вывода. |

# Введение

Искусственный интеллект становится реальностью. Мобильные приложения научились распознавать речь, социальные сети - друзей на фотографиях, автомобили все лучше и лучше ездят без водителей, а системы безопасности учатся без помощи человека автоматически определять подозрительные события. Его применение может принести пользу почти в любой деятельности не требующей широкого использования творческих способностей в работе.

Применение технологий искусственного интеллекта для автоматизации работы людей - специалистов, под которыми в настоящей работе подразумеваются агенты технической поддержки, так же возможно, однако пока не столь развито, как могло бы быть. За время развития области информационных технологий сконструировано множество различных информационных систем, которые необходимо поддерживать и обслуживать, что отнимает много времени и человеческих ресурсов. Часть этих задач способен решать искусственный интеллект.

К такой категории задач можно отнести работу электронных помощников. Если рассматривать различные заявки, поступающие от пользователей, то среди них есть множество простых в исполнении, например, «Установите мне приложение X», «Как найти статью X на вашем сайте?», «Не работает раздел Х на сайте».

**Целью** данной дипломной работы является разработка части системы, использующей принципы искусственного интеллекта и семантического анализа входящей информации для повышения эффективности деятельности ИТ - службы предприятия, ответственной за исполнение принятого системой решения. Важными критериями таких систем являются способности обучения и использования простейшей логики [1]: мышление по аналогии; экстраполяция результата, объектирование объектов из запросов, распознавание человеческого языка и создание из него инструкций. В рамках работы над системой исследуются и используются следующие модели: IDP, Menta 0.1, Menta 0.2-0.3, Framework Thinking Understanding 1.0.

**Предметом исследования** является возможность применения модели мышления Марвина-Мински в процессе регистрации и устранения проблемных ситуаций, возникающих в ИТ - инфраструктуре предприятия. Эффективность подобного решения, сравнение быстродействия с классическим подходом к устранению проблемных ситуаций пользователя.

**Методы исследования -** теоретические методы; имитационные моделирование; теория баз знаний в области искусственного интеллекта**.**

**Задача исследования -** провести краткий анализ предметной области, анализ фреймворка Thinking Understanding, доработка функционала системы.

# ТЕОРЕТИЧЕСКАЯ ЧАСТЬ РАБОТЫ

## Постановка задачи дипломной работы

Для того чтобы создать модель системы, необходимо определить требования к ней или критерии, соответствие которым будет служить одним из доказательств состоятельности системы наряду с экспериментальными результатами. Перечисленные ниже требования сформированы, исходя из возможностей специалистов службы поддержки, а также анализа проблем, которыми они занимаются. Большинство инцидентов - тривиальные и типичные, но все они разные. Для человека проблемы «Please install Firefox» (Пожалуйста, установите Firefox) и «Please install Chrome» (Пожалуйста, установите Chrome) идентичны, но с точки зрения формализации - это не так - общее в них можно найти, взглянув на обобщение различающейся части: Firefox и Chrome являются пакетами программного обеспечения. Чтобы понять, каким требованиям должны соответствовать интеллектуальная система регистрации и анализа проблемных ситуаций в ИТ-области, нужно понять, что делает специалист службы поддержки. Такой специалист выполняет несколько основных шагов своей работы:

1. Регистрирует проблему;
2. Анализирует, ищет решение, проводя логические рассуждения и фиксируя в памяти удачное решение;
3. Решает проблему применяя решение, полученное на втором шаге.

Таким образом, для обеспечения полностью автоматического разрешения инцидентов, интеллектуальная система регистрации и анализа проблемных ситуаций в ИТ-области должна соответствовать следующим критериям:

1. Осуществлять мониторинг ИТ-инфраструктуры пользователя;
2. Регистрировать инциденты;
3. Создавать цепи обработки (Workflow) инцидента;
4. Понимать и формализовать запросы пользователя на естественном языке;
5. Искать решение, обучаться алгоритму разрешения инцидента; уметь проводить логические рассуждения (обобщение, специализация, синонимичный поиск);
6. Применять найденное решение применительно к компьютеру пользователя.

Целью данной работы была реализация исполнительной части системы повышения эффективности ИТ-службы для уже частично реализованного фреймворка принятия решения на основе искусственного интеллекта.

Из этого списка требований к системе важно выделить формализацию запросов на естественном языке. В следующем разделе работы приведены результаты анализа разработок в области формализации запросов на естественном языке.

## Обзор существующих систем и их сравнительный анализ

1. HPE OpenViews - комплексное программное решение по мониторингу ИТ-инфраструктуры [4-7] предприятия и имеет множество модулей: мониторинг; регистрация инцидентов; управление системами. При этом система не поддерживает понимание и формализацию запросов, и их автоматическое устранение.
2. Service NOW - средство автоматизации сервиса. Система предоставляет следующие возможности: регистрация инцидентов и создание цепи их обработки. Данная система также не поддерживает понимание и формализацию запросов, и автоматическое решение проблемы на основе формализации запроса. Система широко используется в ИТ- инфраструктуре Европейского Центра ядерных исследований [8, 9] для регистрации инцидентов и их решения.
3. IBM Watson [2] [3] - это вопросно-ответная система [19], которая поддерживает понимание и формализацию запросов и поиск решений. Система не поддерживает автоматическое разрешение проблемы на основе формализации запроса. Система широко используется в медицине для постановки диагнозов болезней [10-12] и реализует базовые принципы искусственного интеллекта [13-14]. Её разработка производилась целенаправленно для суперкомпьютера IBM Deep Blue [15].

Сравнительный анализ функционала существующих систем:

|  |  |  |  |
| --- | --- | --- | --- |
| **Функционал** | **HPE OpenView** | **Service NOW** | **IBM Watson** |
| Мониторинг системы | Да | Да | Да |
| Регистрация инцидентов | Да | Да | Да |
| Управление системами | Да | Нет | Нет |
| Создание цепи обработки инцидента | Да | Да | Нет |
| Понимание и формализация запросов на естественном языке | Нет | Нет | Да |
| Поиск решений | Нет | Нет | Да |
| Применение решений | Нет | Нет | Нет |
| Обучение разрешению инцидента | Нет | Нет | Да |
| Умение проводить логические рассуждения: генерализацию, специализацию, синонимичный поиск | Нет | Нет | Нет |

**Таблица 1.2.1** Сравнительный анализ существующих систем

Исходя из приведенной таблицы можно сказать, что ни одна из рассмотренных систем не имеет функционала самостоятельного решения проблем в обслуживаемой ИТ инфраструктуре без вмешательства человека. Каждая система в той или иной мере реализует те или иные функции, но системы реализующей их все полностью, пока не существует. В связи с этим, разработка решения проблемы данной работы была произведена с самого начала, не основываясь на готовых примерах систем.

## Модель теории массового обслуживания ИТ инфраструктуры

Большинство проблем, которые решает удаленная служба поддержки информационных систем носит достаточно тривиальный характер. Существуют так называемые линии поддержки, отличающиеся друг от друга уровнем знаний системы специалистами поддержки. Чем чаще встречается проблема и проще её решение - тем более близкий к пользователю уровень поддержки решает проблему. Если первый уровень (т.н. линия) поддержки не может справиться с запросом пользователя - запрос передаётся на вторую линию и т.д. Как правило линий поддержки существует 4 - от решения простых задокументированных проблем, до сложных архитектурных проблем информационной системы.

|  |  |
| --- | --- |
| **Уровень поддержки** | **Описание решаемых задач** |
| Первая линия | Решение уже известных, задокументированных задач, работа напрямую с пользователями |
| Вторая линия | Решение ранее неизвестных проблем |
| Третья линия | Решение сложных и нетривиальных задач |
| Четвертая линия | Решение архитектурных проблем инфраструктуры |

**Таблица 1.3.1** Уровни (линии) поддержки ИТ-служб

В процессе жизненного цикла поддержки системы большое количество простых в решении проблем и задач повторяется достаточно часто. Такие проблемы и задачи решаются специалистами первой линии поддержки, соответственно их решение довольно тривиально, но само их количество достаточно велико. В связи с этим, на задачи такого рода используется большой временной и человеческий ресурс. В связи с этим количество специалистов поддержки в первой линии максимально и уменьшается с номером линии поддержки.

Рисунок. 1.3.1. Соотношение количества специалистов на разных линиях поддержки

## Вывод

Как видно из графика, более 50 % проблем решаются специалистами первой линии поддержки. Часть данных задач можно автоматизировать, разгрузив первую линию поддержки, высвободив человеческие и временные ресурсы.

# СУЩЕСТВУЮЩИЕ МОДЕЛИ ИНТЕЛЛЕКТУАЛЬНОГО ПОИСКА

В рамках работы научно-исследовательской работы были рассмотрены несколько моделей интеллектуального поиска. Каждая из моделей имеет свои подходы, преимущества и недостатки.

## Обзор модели IDP

Модель IDP одна из первых, рассмотренных, моделей. (IDP-Intellectual Document Processing). Эта модель представляет собой систему, в которую загружается набор документов. Система классифицирует документы и строит дерево принятия решений. Ответ на поставленный вопрос формировался при помощи деревьев принятия решений [17]. Такая система не поддерживает обучения на естественном языке, а также требовала значительного количества документов для функционирования.

## Обзор модели Menta 0.1

Модель Menta 0.1 развитие модели IDP. Данная модель разбита на несколько модулей:

1. Обработка запросов на естественном языке;
2. Поиск решения;
3. Применение решения.

Menta 0.1 ориентирована на выполнение простых команд, например, «Добавить поле к форме». Основная функция модели представлена следующим потоком действий:

1. Получение и формализация запроса;
2. Генерация Action (специального объекта, который содержит информацию для следующего шага);
3. Изменение модели приложения в формате OWL [20] согласно Action;
4. Генерация и компиляция приложения.

Основные проблемы модели Menta 0.1:

1. Отсутствие устойчивости к ошибкам входной информации (грамматическим и содержательным), например, входной файл не имел отношения к программной системе, модель которой была представлена в базе знаний в формате OWL;
2. Система поиска решения работала только в рамках модели одной программы и только на прямых командах;
3. Отсутствовала функция обучения.

## Обзор модели Menta 0.2-0.3

Модель Menta 0.2-0.3 новая версия модели Menta 0.1, решающая некоторые из возникших проблем. Модель версии 0.2-0.3 привносит в модель генетические алгоритмы [16] в сочетании и модулями логики. В результате были сформированы основные компоненты будущей итоговой модели:

1. Критерии приёмки (Acceptance Criteria);
2. How-To для хранения решений вычислений для проверки решений;
3. Формат данных OWL;
4. Использование логических вычислений для проверки решения.

При помощи генетического алгоритма модель строила из отдельных частей новую систему и при помощи логического движка проверяла ее соответствие входным критерия приемки. Основным недостатком такого подхода оказалось отсутствие обучения и обработки естественного языка.

Так как модель Menta 0.2-0.3 можно назвать прародителем модели Thinking Understanding, которая стала результатом данной работы, необходимо вкратце рассмотреть её основные компоненты.

### MentaController

MentaController - это веб-служба, которая предоставляет интерфейс для общения с пользователем и остальными системами.

### SolutionGenerator

SolutionGenerator - модуль отвечающий за генерацию решения. На вход он получает Acceptance Criteria. Основой являетcя генетический алгоритм. Для него был выбран framework ecj. Из всех возможных классов в базе знаний, отсеянных по классификатору, составляются паросочетания. К каждому паросочетанию применяется логическое суждение на основе AcceptanceCriteria (за это отвечает модуль ReasonerAdapter). В итоге паросочетание получает оценку в виде пары Frequency, Confidence (частота, вероятность). Таким образом находится наилучшее паросочетание. Если его показатель 1,1, то решение принимается, иначе отбрасывается (на данный момент установлен жесткий показатель). SolutionGenerator включает в себя SolutionChecker, который включает в себя ReasonerAdapter.

### SolutionChecker

SolutionChecker - модуль отвечающий за проверку решения. Принимает на вход выбранные How-To, AcceptanceCriteria. Комбинирует их и передает ReasonerAdapter

### ReasonerAdapter

ReasonerAdapter - транслирует How-To в термины NARS. NARS - non-axiomatic reasoning system (система логических суждений, разработанная профессором Пеем Вонгом). Принцип действия NARS - это всевозможная комбинация фактов. Каждый факт имеет свои частоту и вероятность. Их сочетанием получается композиция данных фактов.

### Translator

Translator - транслирует объекты базы знаний (знания) в отчеты. Последние бывают следующих типов: Solution Report; UML Report; Patch. В данной версии используется первый тип отчета. Он содержит описание на выбранном языке программирования решения, найденного системой.

### Applicator

Applicator - модуль применяет решение к модели приложения, содержащейся в базе знаний. Также данная модель включает FileApplicator, который генерирует решение в виде файлов на выбранном языке программирования.

### KBServer

KBServer - база знаний приложения. Используется сервер nonSQL БД HypergraphDB. При реализации базы знаний был создан промежуточный модуль доступа к данным (здесь и далее - DAO, Data Access Object), данный подход широко используется в проектировании программного обеспечения. Это позволяет максимально отделить реализацию KBServer от конкретного хранилища. Класс EntityManagerFactory является входной точкой и создает объект, с помощью которого приложение осуществляет работу с базой знаний. Класс автоматически выбирает необходимые настройки для объекта. Класс EntityManager - основной класс для загрузки и хранения объектов из базы данных. Класс Configuration хранит такие параметры настройки базы данных, как физическое положение БД и максимальное количество подключений. Класс EntityTransaction используется для управления транзакциями при доступе к объектам базы данных.

Модель имеет следующие недостатки: отсутствие обучения; отсутствие обработки искусственного языка; комбинаторный взрыв алгоритма поиска и сравнения фактов.

## Обзор модели TU

Модель TU (Thinking Understanding) - это фреймворк основанный на модели мышления Марвина Мински [18]. Марвин Ли Минский - американский учёный в области искусственного интеллекта, сооснователь Лаборатории искусственного интеллекта в Массачусетском технологическом институте. Окончил Филлипсовскую академию в городке Эндовер штата Массачусетс. Защитил диссертации в Гарвардском и Принстонском университетах. Сотрудник Массачусетского технологического института с 1958 года. В 1959 году вместе с Джоном Маккарти основал в Массачусетском технологическом институте лабораторию информатики и искусственного интеллекта. Вплоть до своей смерти являлся профессором информационных искусств и наук, профессором электроники и электротехники и профессором вычислительных наук.

В 2006 году Марвин Мински опубликовал свою книгу «The emotion machine», в которой предложил свой взгляд на систему мышления и памяти человека. Основой его теории стала парадигма триплета Критик - Селектор - Образ мышления (далее T3), k-line (линия, которая связывает приобретенные знания, например, огонь - горячо) для общего сопоставления знаний. Она представляет собой связь между двумя событиями, объединяющими их в знание. На рисунке 2.4.1 представлено схематичное изображение T3:

Селектор

Критик

Рисунок 2.4.1. Критик - Селектор - Образ мышления

### Критик

Критик - это вероятностный предикат. В упрощенным виде критик является определенным переключателем: внешние обстоятельства, события или иное воздействие. Например, «услышал резкий громкий звук - обернулся на источник», «прикоснулись к горячему чайнику и одернули руку». После активации определенным событием критик с помощью своего предиката проверяет, реагирует ли система на это событие или нет. Критик активируется только тогда, когда для этого достаточно обстоятельств. Одновременно может активироваться несколько критиков. Например, при решении человеком сложной задачи, идет активация множества критиков: рассчитать значения переменных, уточнить технические детали. Кроме того, параллельно могут активироваться критики, сообщающие о необходимости отдыха. После срабатывания критик ищет сопоставленный ему селектор и возвращает его. В реализованной системе критиками являются: «Критик входящих запросов», который срабатывает при появлении запроса в системе; «Критик обработки естественного языка», который срабатывает при построении семантической сети входящего запроса.

### Селектор

Селектор является компонентом работы с данными и производит подготовку запроса для выборки информации. Селектор может вернуть запрос для активации в системе других компонентов: либо другого критика, либо другого селектора. Селектор подготавливает атрибуты запроса, занимается выбором определенных ресурсов, одним из которых является Образ мышления. В рассмотренной модели объектами типа селектор является, например, объект «Классификация проблем с прямыми инструкциями», который активируется «Критиком входящих запросов».

### Образ мышления

Образ мышления - способ решения проблемы. Образ мышления может быть простым, а может быть комплексным - способным активировать другие критики. Например, размышляя над проблемой, человек понимает, что нужно произвести полный перебор всех возможных комбинаций параметров, для получения решения проблемы, и тут он решает поискать готовое решения: возможно кто-либо уже сделал такой перебор комбинаций, и можно будет использовать его результаты. В данном случае «поиск готового решения» является критиком внутри образа мышления «поиск решения». Пример комплексного образа мышления можно описать подобной схемой:

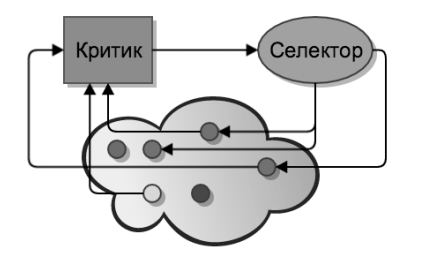


Рисунок 2.4.3.1. Критик - Селектор - Образ мышления

Примерами путей мышления в области решения проблем информационных технологий могут служить: «приобретённое знание» - система знает, как решить проблему, используя уже накопленные знания; «адаптация» - применение существующего «похожего» решения; «реформуляция» - перевод проблемы после «адаптации» в «приобретённое знание»; «зов к помощи» - путь, при помощи которого система обращается к специалисту-человеку за помощью (во время активации этого пути система переходит в режим обучения).

### Уровни мышления

Концепция уровней мышления представляет собой модель степени ментальной активности человека. Никто из людей не может похвастаться скоростью гепарда, гибкостью кошки или силой медведя. На наш взгляд, все это компенсируется возможностью изобретения образов мышления. Например, чтобы быть быстрыми, люди изобрели различные механизмы (самолеты, машины и др.). Чтобы быть сильными, они изобрели оружие.

Все изобретения человечества являются результатом взаимодействия человека с окружающим миром. Данное взаимодействие ведет людей к созданию шедевров литературы и уникальных механизмов, изобретению компьютеров и установлению рекордов. По ходу своего развития человеческий разум проходит путь от врожденных инстинктов до возможности создания фундаментальных трудов. В этом ему помогает возможность гибкого мышления: изобретение различных подходов к решению проблемы. Далее мы рассмотрим концепцию уровней мышления, следуя Марвину Мински.

#### Инстинктивный уровень

Происходят инстинктивные реакции (врожденные). Например, коленный рефлекс. Общую формулу для этого уровня можно выразить как «если ..., то сделать так». Данный уровень включает врожденные инстинкты, высший уровень - идеалы и персональные цели. На первом уровне система TU опирается на вхождение ключевых фраз в текстовое сообщение и, применяя регулярные выражения, пытается «инстинктивно» понять проблему, например, в случае шаблонных запросов, создаваемых сторонними пользователями.

#### Уровень обученных реакций

Используются накопленные знания, то есть те знания, которым человек обучается в течение жизни. Например, переходить дорогу на зеленый свет. Общую формулу для этого уровня можно описать как «если ..., то сделать так». Если TU не смогла найти решение на первом уровне, она переходит на второй уровень; здесь происходит построение семантической сети входящего запроса, активируется «критик классификации проблем».

#### Уровень рассуждений

Мышление с использованием рассуждений. Например, если перебежать дорогу на зеленый свет, то можно успеть вовремя. На данном уровне сравниваются последствия нескольких решений и выбирается оптимальное. Общую формулу для этого уровня можно выразить как «если ..., то сделать так, тогда будет так». Третий уровень в фреймворке контролирует, не найдено ли решение текущей проблемы, ставит системе новые цели. Базовая цель - «помочь пользователю». Начиная с нее, система анализирует подцели: понять запрос, понять проблему, найти решение.

#### Рефлексивный уровень

Рассуждения с учетом анализа прошлых событий. Например, «в прошлый раз я побежал на моргающий зеленый и чуть не попал под машину». В отношении Thinking Understanding Четвёртый уровень контролирует время выполнения входящего запроса и, если это время превышает определённую планку, производит перераспределение ресурсов.

#### Саморефлексивный уровень

Построение определенной модели, с помощью которой идет оценка своих поступков. Например, «мое решение не пойти на это собрание было неверным, так как я упустил столько возможностей, я был легкомысленным». На пятом уровне TU происходит инициализация контекста запросов, происходят коммуникации с пользователем.

#### Самосознательный уровень

Оценка своих поступков с точки зрения высших идеалов и оценок окружающих. Например, «а что подумают мои друзья? А как бы поступил мой герой?». Шестой уровень системы контролирует общее состояние системы, ресурсов, проблемы функционирования аппаратного комплекса и выставляет общий статус системы. Если все запросы укладываются в отведенное время, то выставляется положительный статус, иначе выставляется отрицательный статус. По общему статусу можно определить, необходимо ли внешнее вмешательство в работу системы: замена компонентов, увеличение ресурсов и прочее.

|  |
| --- |
| Ценности, Идеалы и Табу |

|  |
| --- |
| Уровень самосознательной рефлексии |
| Уровень саморефлексии |
| Уровень рефлексии |
| Уровень мышления |
| Уровень обученных реакций |
| Уровень инстинктивных реакций |

|  |
| --- |
| Инстинктивные страхи и стимулы |

**Рисунок 2.4.4.6.1** Уровни мышления модели мышления Марвина Мински

В модели Мински уровни с первого по третий составляют личность человека. В это же время, уровни со второго по пятый представляют ЭГО человека, его человеческое Я, т.е. осознание человека в общении с окружающими людьми. Уровни с третий по шестой - сверх ЭГО, т.е. его моральные принципы.

Относительно фреймворка TU важно отметить, что каждый последующий уровень контролирует предыдущие. Под контролем понимается доступ к информационным параметрам управления предыдущими уровнями. Для обмена информацией между уровнями и запросами была разработана концепция краткосрочной и долгосрочной памяти. Пути мышления работают с краткосрочной памятью и модифицируют данные в ней. После успешной обработки запроса память переписывается в долгосрочную и сохраняется уже в общей базе знаний. Таким образом, в базу знаний попадает только апробированная информация, исключая ошибки нахождения неверного решения.

Модель данных системы является семантической сетью, построенной над нереляционной базой данных, и описывает «знания» системы: решения, проблемы, пути мышления, критики, селекторы и т. д. Таким образом критики, селекторы, пути мышления могут быть «приобретенными знаниями».

Необходимо отметить возможность обучения системы. Как и концепция долгосрочной и краткосрочной памяти, обучение системы является нашим расширением модели мышления Мински. На начальном этапе система содержит базовые концепции: объект, действие. С помощью обучения в систему можно ввести новые концепции. Обучение также проводится на естественном языке. Например, «веб-браузер - это объект. Firefox - это веб-браузер». Теперь система знает две новых концепции - веб-браузер и Firefox, а также то, что Firefox - это веб-браузер.

Модель TU является более абстрактной и представляет собой верхнеуровневою архитектуру обработки запроса (мышления), ее компонентами стали лучшие части предыдущих систем

Модель созданная на основе модели мышления Мински наследует основные концепции предыдущих моделей и показала свою состоятельность на контрольных примерах.

Характерные особенности системы:

1. Acceptance Criteria;
2. Обучение, поиск и применение решения;

## Вывод

Для программной экспертной системы очень важно обладать способностью мыслить и рассуждать, например, очень важно для системы уметь проводить аналогии между уже известными разрешениями проблем и вновь появившимися. Множество запросов в ИТ службы предприятий типичны и отличаются лишь параметрами. Для экспертной системы важно уметь абстрагировать специализированные рецепты решения. К примеру, система научилась решать инцидент «Please install Firefox» («Пожалуйста, установите Firefox»). Абстрагировав данный инцидент до состояния «Please install browser» («Пожалуйста, установите браузер») или даже «Please install software» («Пожалуйста установите программу»), система сможет теми же способами попробовать разрешить новую проблему с установкой любого программного обеспечения. Модель мышления Марвина Мински подходит под для реализации системы, так как она может быть адаптирована на области разрешения проблемных ситуаций, возникающих в процессе эксплуатации ИТ-инфраструктуры предприятия. Система поддерживает основные функции: обучение, понимание инцидента, поиск решения, применение решения.

Были рассмотрены каждый из основных понятий модели мышления Марвина Мински. Важно отметить, что в своей работе Мински дает лишь рамочное описание модели с точки зрения человека.

Данная концепция была расширена и перенесена на компьютерную систему для повышения эффективности ит-службы предприятия.

TU не является программным обеспечением в общем смысле этого слова, как об этом можно подумать, это фреймворк который предоставляется по схеме “Аs a Service” для конкретных пользователей. Для данной работы взят случай, когда пользователь (ИТ-служба предприятия) поддерживает удалённую свою инфраструктуру предприятия, в этом случает TU будет предоставлен и натренирован с помощью персонала тех. поддержки, и в последствие будет вести себя как специалист тех. поддержки.

TU поддерживает следующие функции:

1. Диалог с пользователем
   1. Уточнение запроса;
   2. Подтверждение запроса;
   3. Эскалирование на персонал службы поддержки, когда система не может разрешить запрос.
2. Обучение
   1. Обучающие концепты;
   2. Обучающие разрешения проблемы.
3. Операции
   1. Поиск решения;
   2. Применение найденного решения.

ThinkingUnderstandingможет быть «натренирован» с использованием накопленных знаний специалистов в определённой области, где он используется, также может взаимодействовать со специалистами во время, когда какой-либо из запросов находится в процессе разрешения, если TU не может разрешить это сам по себе.

Три основных направления TU:

1. Общение и взаимодействие со специалистами людьми в диалоговом режиме;
2. Быть обученным и способным к обучению;
3. Нахождение наиболее возможного решения и его применение на указанной системе.

# ОБЗОР ПРИНЦИПА ДЕЙСТВИЯ TU

В данной главе рассматривается реализация модели TU: архитектура системы и программная реализация.

## Архитектура системы TU

В этом разделе работы дано краткое описание модулей, в дальнейших разделах будет дано их детальное описание, включая реализованный в рамках работы модуль исполнительной части системы.

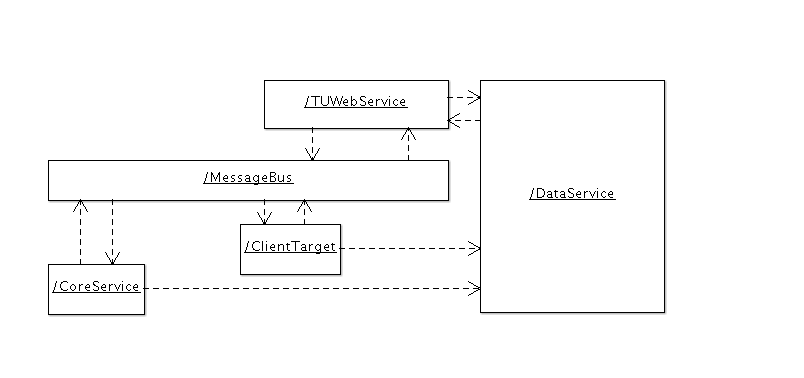
Основные модули системы описаны в таблице 3.1.1:

|  |  |
| --- | --- |
| **Модуль** | **Функционал модуля** |
| TU WebService | Основной компонент взаимодействия с внешними система, включая пользователя |
| CoreService | Ядро системы, содержащее основные классы TU |
| DataService | Компонент работы с данными |
| Reasoner | Компонент вероятностной логики |
| MessageBus | Шина данных фреймворка |
| ClientAgent | Компонент выполнения решения на целевой машине пользователя |

**Таблица 3.1.1** Основные модули фреймворка TU

## Принцип работы компонент TU

В данном разделе приведено описание алгоритма взаимодействия всех модулей системы. На рисунке 3.2.1 представлено верхнеуровневое взаимодействие основных компонентов системы:



**Рисунок 3.2.1** Диаграмма взаимодействия модулей

Основной точкой взаимодействия (с точки зрения пользователя системы) с системой является компонент TUWebService. Данный компонент построен на стандарте Web Service SOAP для универсального использования в различных внешних системах. Взаимодействие происходит по следующей схеме:

1. TUWebService получает запрос пользователя и сохраняет запрос в Базе Знаний используя DataService;

2. WebService отправляет сообщение типа Request с информацией о запросе в компонент MessageBus (шина);

3. Один из экземпляров CoreService обрабатывает запрос;

4. Компонент CoreService обрабатывает запрос и сохраняет результаты в Базе Знаний, затем он отправляет в MessageBus сообщение RequestCompleted и сообщение ActionsToExecute с указанием действий, которые необходимо исполнить;

5. WebService получает сообщение RequestCompleted c результатами выполнения запроса и уведомляет подписчиков (конечных пользователей);

6. Компонент ClientAgent получает сообщение ActionsToExecute со списком действий, которые необходимо исполнить на целевых машинах.

## Компоненты системы TU

### TU WebService

TUWebService - компонент взаимодействия с «внешней средой», который предоставляет список функций и типов, посредством вызова которых можно обработать запрос в системе.

### MessageBus

Компонент MessageBus - это шина данных, которая обрабатывает сообщения и посылает их указанным компонентам

### DataService

Компонент DataService отвечает за хранения данных, предоставляет базу знаний для приложения.

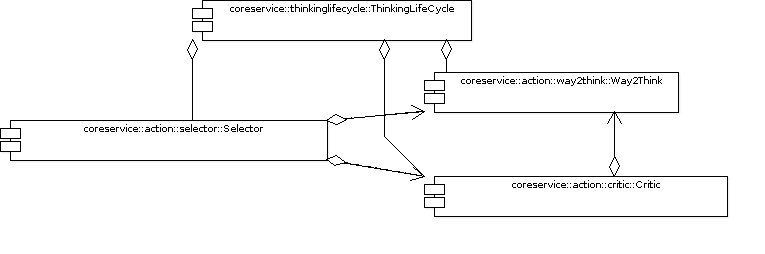
### CoreService

CoreService - ядро приложения, которое управляет основным жизненным циклом системы. CoreService состоит из нескольких компонентов:

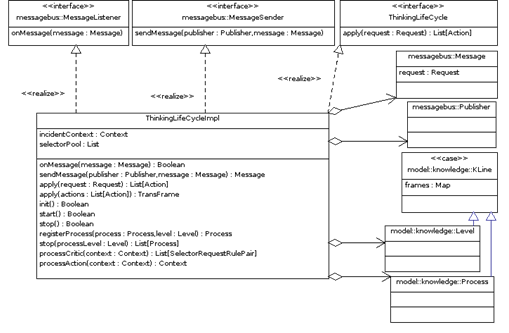
#### ThinkingLifeCycle

ThinkingLifeCycle (TLC) - данный компонент системы отвечает за управление жизненным циклом системы: потоками, событиями приложения. Он запускает исполнение Критиков (Critic), Селекторов (Selector), Образов мышления (Way2Think), осуществляет обмен данных между компонентами. Компонент построен на фреймворке Akka Concurrency, который позволяет разрабатывать приложения, работающие параллельно [21].

Компонент ThinkingLifeCycle запускает и останавливает внутренние процессы в системе; компонент Selector получается ресурсы используя Critic запрос; Critic это основной аналитический модуль кратковременной памяти; Way2Think компонент меняет содержание кратковременной памяти. В следующих главах будут рассмотрены каждая компонента будет описана более подробно.



**Рисунок 3.3.4.1.1** Диаграмма основных компонентов TU



**Рисунок 3.3.4.1.2** Диаграмма интерфейса компонента ThinkingLifeCycle

Компонент ThinkingLifeCycle имеет 6 уровней мышления описанных ранее в главе 2.4.4:

1. Self-Conscious Reflection;
2. Self-Reflective Thinking;
3. Reflective;
4. Deliberative;
5. Learned;
6. Instinctive.

#### Selector

Selector - это компонент, который ответственен за получение списка действий и ресурсов из базы знаний, согласно входным параметрам. Данный компонент используется для получения правильных действий из сервера базы знаний в зависимости от переданных параметров.

|  |
| --- |
| <<interface>>  Selector |
| apply(request : Request)  apply(goal : Goal)  apply(criticResult : SelectorRequest)  apply(criticResult : SelectorRequestRulePair)  apply(criticResults : List[SelectorRequestRulePair])  apply(criteria : SemanticNetwork)  start()  stop() |

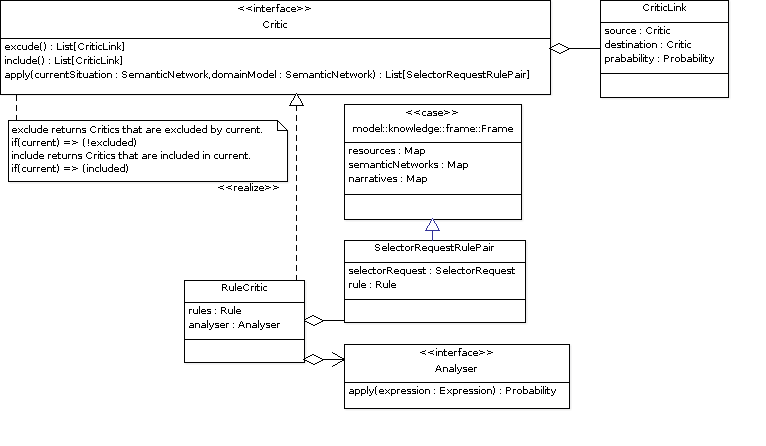
**Рисунок 3.3.4.2.1** Диаграмма интерфейса компонента Selector

|  |  |
| --- | --- |
| **Метод** | **Описание** |
| apply(request : Request) : Action | Данный метод на основе запроса пользователя получает из Базы знаний необходимые Critic. |
| apply(goal: Goal) : Action | Данный метод на основе цели системы получает из Базы знаний необходимые Critic. |
| apply(criticResult : ActionProbabilityRule) : Action | Данный метод на основе работы Critic получает из Базы знаний необходимые Action. |

**Таблица 3.3.4.2.1** Описание методов класса (компонента) Selector

#### Critic

Critic является основным компонентом для анализа в T3. Critic используется для классификации входной информации, рефлексии, самоанализа и служит определенным вероятностным переключателем. Например, компоненты контроля времени, контроля эмоционального состояния системы - это тоже Critic. Анализ включает в себя принятие решений. Подсистема принятия решений выделена в отдельный компонент, названный Reasoner. Critic используется для выбора компонента Way2Think, классификации введенной информации, рефлексии, самоанализа и так далее. Ниже приведена диаграмма интерфейса компонента Critic:



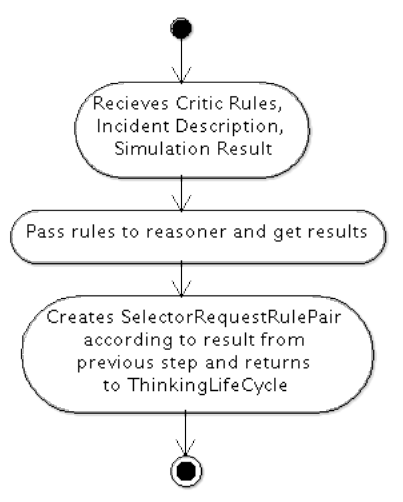
**Рисунок 3.3.4.3.1** Диаграмма интерфейса компонента Critic

В фреймворке существует три типа Critic, для различных целей:

Manager - простой тип критика, который работает как триггер, например, Goal, который запускает необходимый WayToThink. Именно он используется для решения проблемы пользователя.

Control - контролирующий Critic, который ждёт определенного события (срабатывает на определенное событие). Например, заканчивается отведенное на решение проблемы время.

Analyzer – анализатор, обрабатывающий и выявляющий тип инцидента. Например, прямые инструкции, проблема с желаемым состоянием, выбор наиболее вероятного действия.



**Рисунок 3.3.4.3.1** Алгоритм действий компонента Critic

|  |  |
| --- | --- |
| **Метод** | **Описание** |
| exclude() : List[CriticLink] | Возвращает список CriticLink, которые при срабатывание данного Critic будут игнорироваться с определенной вероятностью, после срабатывания Critic будет посчитана суммарная вероятность активации. После чего система решит, какой Critic был вероятнее всего активирован |
| include() : List[Critic] | Возвращает список объектов класса CriticLink, которые при срабатывание данного Critic будут включаться с определенной вероятностью |
| apply(currentSituation: SemanticNetwork, DomainModel : SemanticNetwork) : List[SelectorRequestRulePair] | Запускает Critic, после чего вернется список Selector с определенной вероятностью, после чего TLC их активирует |

**Таблица 3.3.4.3.1** Описание методов класса (компонента) Critic

Critic действуют на разных уровнях мышления. Далее приведено описание Critic с привязкой к уровням мышления.

1. Уровень обученных реакций;
   1. PreprocessManager - предобработка информации;
   2. Классификаторы инцидентов: Прямые инструкции, Проблема с желаемым состоянием, Проблема без желаемого состояния;
   3. SolutionCompletenessManager - связывается с пользователем и проверяет устраивает ли его найденное решение.
2. Уровень рассуждений;
   1. Выбор наиболее вероятного Selector по Rule. Данный Critic после проверки правил, выбирает из них правило с большей вероятностью.
3. Рефлексивный уровень;
   1. Менеджер целей. Установка целей.
4. Саморефлексивный уровень;
   1. ProcessingManager - запускает выполнение запроса;
   2. TimeControl - контроль времени исполнения запроса;
   3. DoNotUnderstandManager - активируется, когда необходимо уточнение пользователя для продолжения работы.
5. Самосознательный уровень;
   1. EmotionalStateManager - контроль общего состояния системы.

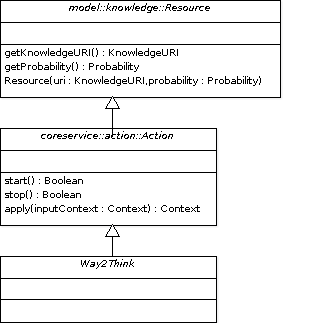
Основным примером работы компонента может служить классификация инцидентов. Например, у нас есть Critic для прямых инструкций DirectInstruction, есть для ситуации с желаемым состоянием DesiredState. Пусть на входе будет запрос вида: install antivirus (установить антивирус). DesiredState найдет здесь действие - install, но не найдет желаемого состояния, то есть вероятность его выполнения будет 60%. DirectInstruction будет искать действие и объект, которые присутствуют в запросе, его вероятность будет 100%, его ThinkingLifeCycle будет активирован как наиболее вероятный. Это был является простым примером работы компонента, но на самом деле механизм работы гораздо гибче к запросам: он поддерживает включения, исключения, составные правила и логику. В данной главе был описан важный компонент системы, который определяет алгоритм ее работы, в этом компоненте скрыта основная возможность системы думать и искать решение проблемы пользователя, согласно внешним обстоятельствам и наборам правил.

#### Way2Think

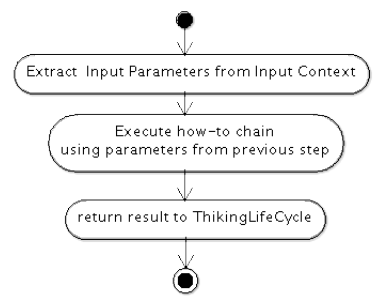
Way2Think является основным операционным компонентом T3. Основными задачами данного компонента являются: обновление, преобразование, сохранение данных и коммуникация с пользователем, иными словами, все, что в той или иной форме изменяет операционный контекст данных системы. Также широко используется для обновления, трансформирования и добавления знаний в систему. WayToThink также используется как описание алгоритма разрешения проблемы пользователя, то есть описывает последовательность действий, необходимых для решения проблемной ситуации

В зависимости от типа WayToThink активируется та или иная часть последовательность действий, шагов. В общем виде последовательность имеет подобный вид: получить данные, обработать, вернуть данные. Например, в случае WayToThinkSimulate второй шаг имеет вид «поиск связи между текущим контекстом проблемы и проблемами, уже сохранёнными в базе знаний системы».

В данной главе был описан основной компонент модификации данных WayToThink. Нужно отметить, что данный компонент также является важной частью модели TU. Он проектировался для универсального применения во всех случаях, когда необходимо действие над данными. Например, если нужно использовать скрипт, который был написан на языке интерпретации Bash, или PowerShell, и ввести его в систему, можно разбить каждый шаг и вызов на отдельную часть и сделать сложный WayToThink с ветвлениями и циклами.



**Рисунок 3.3.4.4.1** Диаграмма интерфейса компонента Way2Think



**Рисунок 3.3.4.4.2** Работа компонента WayToThink в режиме описания решения проблемы (HowTo)

### KnowledgeBaseAnnotator

Данный компонент устанавливает связи между терминами во входной фразе и базой знаний и также является WayToThink.

Описание работы компонента

1. Получен Термин;
2. Поиск в локальной базе знаний;
3. Если совпадение не найдено идет запрос во внешнюю базу знаний;
4. Внешняя база возвращает список синонимов;
5. Компонент ищет по синонимам во внутренней базе знаний;
6. Если поиск успешен, то создается связь между входящем термином, синонимом и концепцией в базе знаний.

Например, входящий запрос содержит термин «program» , база знаний содержит термин «computer software». Идет запрос во внешние базы знаний, найдено «computer software, program» Будет добавлена связь-аналогия в база знаний «program - computer software».

### Reasoner

Компонент Reasoner осуществляет логические вычисления для системы, например, для обработки правил в компоненте Critic.

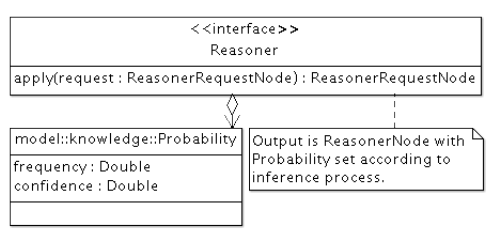
|  |  |
| --- | --- |
| **Метод** | **Описание** |
| apply(request : ReasonerRequestNode) : ReasonerRequestNode | Данный метод проводит обработку правил и считает вероятность (Probability) и уверенность (Confidence) |

**Таблица 3.3.7.1** Описание методов класса (компонента) Critic

В качестве реализации в системе используется два движка логического вычисления PLN и NARS. Главным применением данного компонента являются правила для компоненты Critic. Логика правил обрабатывается при помощи этого компонента. Этот результат используется для определения вероятности активации данного или другого Critic. Такое использование дает простор и гибкость в построении свода правил.

|  |
| --- |
| <<interface>>  KnowledgeBaseServer |
| save(obj)  remove(obj)  select(obj)  link(objs : List<Resource>,linkName : String)  selectLinkedObject(obj,linkName : String) : List<Resource>  addLinkedObject(parent, toLink,linkName: String)  saveRequest(obj : Request) : Request  selectRequest(obj : Request) : Request  saveBusinessObject(obj : RefObject) : RefObject  selectBusinessObject(obj : RefObject) : RefObject |

**Рисунок 3.3.7.1** Диаграмма интерфейса компонента Reasoner



**Рисунок 3.3.7.2** Интерфейс компонента Reasoner

### DataService

Компонент DataService отвечает за хранение данных в системе. База знаний построена на графах. В базе знаний используется два типа объектов Object - объект базы знаний, BusiessObject - объект для WebService (User, Request). BusinessObject является кортежем для интеграции с внешними системами. У объекта есть ID, который уникально удостоверяет его в рамках системы.

|  |  |
| --- | --- |
| **Метод** | **Описание** |
| save(obj : Resource) | Данный метод позволяет сохранить ресурс в базу знаний |
| remove(obj : Resource) | Данный метод позволяет удалить объект |
| select(obj : Resource): Resource | Данный метод позволяет выбрать объект |
| link(obj : List, linkName : String) | Данный метод позволяет сделать ссылку между 2-мя объектами |
| selectLinkedObject(obj : Resource, linkName : String): Link | Данный метод позволяет выбрать все объекты, которые имеют связь под названием linkName с объектом obj |
| addLinkedObject(parent : Resource, toLink : Resource, linkName : String) | Данный метод позволяет создать ссылку linkName с объектом |
| saveRequest(obj : Request) | Данный метод позволяет получить запрос из Базы Знаний |
| selectRequest(obj : RefObject) | Данный метод позволяет получить запрос из Базы Знаний |
| saveBusinessObject(obj : RefObject) : RefObject | Данный метод позволяет сохранить объект в базу |

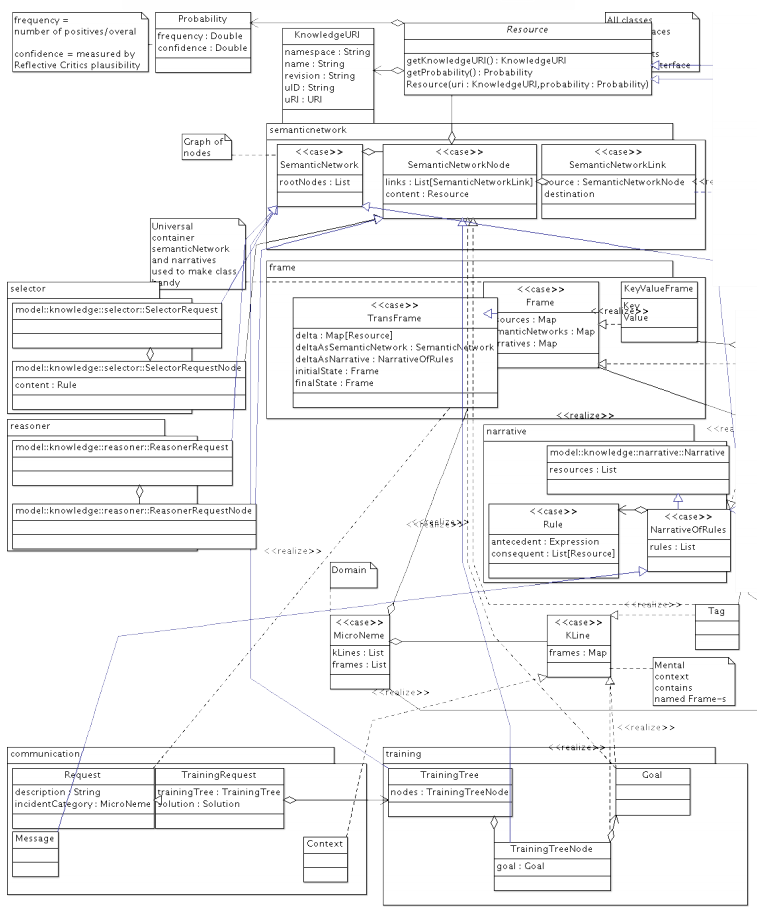
**Таблица 3.3.6.1** Описание методов класса (компонента) Critic

### TU Knowledge

Важной частью системы является реализация хранения данных на основе модели TU. Для работы системы была разработана схема данных, которая сочетает в себе OWL и графовую базу данных. Язык OWL, традиционно использующийся для структурирования информации в Вебе, обрел широкое использование во многих схемах данных, так как давал возможность дополнительного расширенного описания взаимосвязи между данными.

|  |  |
| --- | --- |
| **Класс** | **Описание** |
| Knowledge | Базовый класс всех объектов модели. Содержит в себе URI, по которому уникально идентифицируется. Поддерживает версионность. Свойствами данного объекта обладают все объекты системы. Также содержит Probability (Вероятность) и Confidence (Уверенность) поля. Например, когда в результате работу WayToThink получается Knowledge он имеет Confidence 0, так как он только что был сгенерирован, когда его проверит Critic на его состоятельность при помощи определенных в Critic правил, то он поставит ему не 0 Confidence |
| Narrative | Список слов исходного запроса |
| Rule | Правило. Класс, описывающий правила в системы. Например, правило по которому сработает Critic. |
| AnnotatedNarrative | Слова исходного запроса и их сопоставление на концепции в Базе Знаний |
| SemanticNetwork | Граф из SemanticNetworkNode и SemanticNetworkLink |
| SemanticNetworkNode | Узел графа SemanticNetwork, содержит в себе ссылки на другие узлы, а также ссылку на Knowledge |
| SemanticNetworkLink | Ссылка в графе SemanticNetworkLink |
| Frame | Коллекция объектов Knowledge, с возможностью назначения специального атрибута (тега) для семантической группировки |
| TransFrame | Коллекция Frame, содержащая два состояния одного фрейма: до и после |
| Goal | Цель |
| Tag | То же, что и цель, но используется для меток |
| PreliminaryAnnotation | SemanticNetwork входного запроса |
| KnowledgeBase annotation | SemanticNetwork с сопоставлением концепциям Базы Знаний |
| Domain model | SemanticNetwork доменной модели |
| Situation model | SemanticNetwork, часть DomainModel, созданной для обработки текущего запроса |
| Incident | SemanticNetwork входного запроса к системе |
| K-Line | Связь между объектами. Например, когда в систему поступает запрос она создает K-Line между Conversation, Narrative |
| Conversation | SemanticNetwork, контекст инцидента |
| InboundRequest | SemanticNetwork входного запроса |
| TrainingRequest | SemanticNetwork входного запроса для обучений |

**Таблица 3.3.7.1** Описание классов TU Knowledge



**Рисунок 3.3.7.2** UML схема данных TU

## Реализация исполнительной части системы TU

Исполнительная часть фреймворка TU была реализована в рамках данной работы. Данный модуль должен устанавливаться на клиентские машины, с которых выполняются запросы к TU. Решение проблемы, найденное системой TU, исполняется модулем ClientAgent.

### Требования к ClientAgent:

При классическом решении проблемы пользователя ИТ-службой используется два основных способ. Первый способ - это консультации от какой-либо линии поддержки пользователю - в таком случае пользователь решает свою проблему с помощью инструкций предоставленных ИТ-службой. Второй способ – удалённое подключение специалиста к компьютеру пользователя. Второй способ наиболее универсален и надёжен по сравнению с первым способом, т.к. исключается возможность ошибки пользователя.

К модулю ClientAgent были разработаны следующие требования:

1. Удобное логирование модуля для отладки и изучения исключительных ситуаций, которые могут возникнуть в ходе обращения пользователя к TU и решения проблемы;
2. Решение, принятое TU должно транслироваться на компьютер пользователя и исполнятся для удовлетворения его запроса;
3. После применения решения должна производится проверка решения;
4. ClientAgent должен локально хранить все произведенные с компьютером действия;
5. ClientAgent должен иметь возможность гибкой настройки по своим основным параметрам;
6. ClientAgent должен иметь возможность производить широкий спектр изменений над целевой операционной системой, такие как выполнение команд, управление файловой системой, управление установленными программами, управление питанием компьютера, архивирование, конфигурирование и прочее.
7. ClientAgent должен поддерживать автоматическое обновление, в случае, если на сервере TU появляется обновленная версия приложения;
8. ClientAgent должен быть устойчив к ошибкам и исключительным ситуациям;
9. ClientAgent должен поддерживать выполнение на операционных системах Unix и Windows;
10. ClientAgent должен запускаться при старте операционной системы.

### Выбор технологии для реализации

Фреймворка Thinking Understanding реализован с использованием функционального языка Scala. Scala — мультипарадигмальный язык программирования, спроектированный кратким и типобезопасным для простого и быстрого создания компонентного программного обеспечения, сочетающий возможности функционального и объектно-ориентированного программирования. Основной платформой для эксплуатации системы был выбран Linux Debian. Связано это, прежде всего, с тем, что ряд компонентов был написан на С++ [22] и использует библиотеки, доступные только на Linux.

Однако ClientAgent – компонент исполняемый на клиентской системе. Большинство систем с которых предполагается получение запросов пользователей работает под управлением операционных систем семейства Windows. По данной причине среди технологий, подходящих для реализации ClientAgent рассматривалось три фреймворка/языка программирования: Scala, Java, C#.

Вариант использования Scala [23] был отброшен по причине необходимости установки исполняемой среды Scala на систему пользователя. Также среди перечисленных технологий, Scala обладает наименьшим комьюнити и количеством сторонних библиотек.

Использование Java [24] было отвергнуто по причине возможности отсутствия JRE [25] на целевой системе пользователя, т.к. стандартная сборка Windows не подразумевает установки Java Runtime Environment. В случае если на целевой машине установлена JRE, её версия может сильно отличаться от версии 1.5 до версии 1.8.

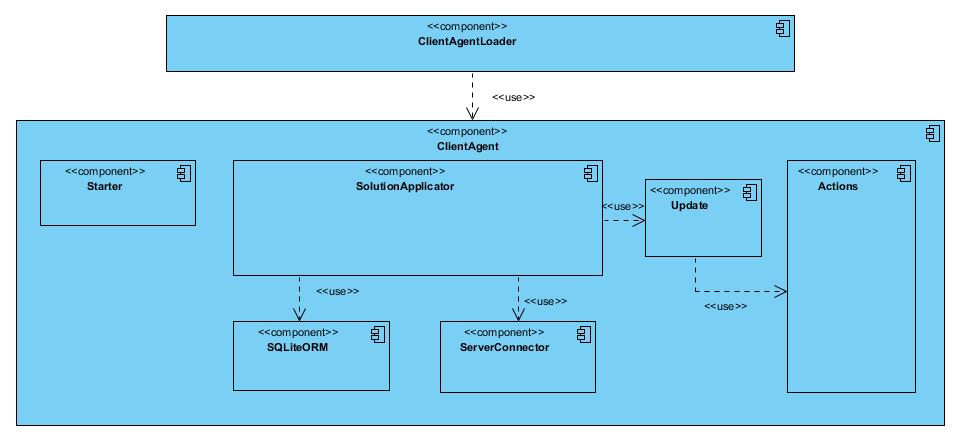
Использование C# [26] как языка программирования оправданно по причине того, что на ОС Windows начиная с версии Windows 7 по умолчанию установлен .Net framework достаточный для исполнения всего необходимого функционала, описанного в требованиях к ClientAgent. Код написанный на C# может также исполняться на OS семейства Linux посредством Wine или на фреймворке MONO. С момента открытия исходного кода .Net Core Microsoft , исполнение кода, написанного под .Net возможно на ОС MacOS и Linux без сторонних библиотек, таких как MONO и окружений типа Wine.

### ClientAgent

ClientAgent - клиентский компонент для выполнения команд на машинах клиента. Для работы клиента необходима непосредственная установка клиента на машины пользователей. Компонент состоит из нескольких модулей:

|  |  |
| --- | --- |
| **Модуль** | **Функционал модуля** |
| ClientAgentLoader | Компонент ответственный за старт и обновление ClientAgent |
| ClientAgent | Ядро клиента, приложение |
| Logger | Компонент логирования приложения |
| SQLiteORM | Компонент работы с данными |
| HowTo | Компонент выполнения решения из TU |
| Actions | Действия, выполняемые с операционной системой |
| ActionsToExecute | Список Actions выполняемый над целевой системой пользователя |
| Starter | Компонент старта приложения |
| ServerConnector | Компонент связи с сервером TU |
| SolutionApplicator | Компонент выполнения решения из TU на целевой машине пользователя |

**Таблица 3.4.3.1** Основные модули компоненты ClientAgent



**Рисунок 3.4.3.1** UML диаграмма компонентов ClientAgent

Более детальное описание каждого модуля дано в следующих главах отдельно.

#### ClientAgentLoader

Данный модуль является загрузчиком клиента ClientAgent. Он запускается при старте операционной системы и выполняет следующие действия:

1. Проверяет связь с сервером ThinkingUnderstanding. В случае потери связи загрузчик останавливает свою работу, до следующей проверки связи, по причине того, что работа ClientAgent не имеет смысла без связи с сервером TU;
2. Проверяет версию и целостность клиента ClientAgent. Для выполнения данной процедуры Loader делает запрос к серверу TU.

По запросу клиента сервер присылает словарь из имён файлов и их хеш сумм. Целостность и версия файлов проверяется сверкой файлов по хеш-функции MD5;

* 1. В случае, если версия клиента устарела, либо целостность файлов нарушена файлы перекачиваются с сервера. Для защиты от потери файлов и прочих исключительных ситуаций новая версия клиента скачивается zip файлом защищённым паролем. После скачивания архива он разархивируется;
  2. В случае, если версия клиента актуальна Loader запускает экземпляр службы ClientAgent.

1. В случае, если версия клиента актуальна Loader запускает экземпляр службы ClientAgent.

Модуль ClientAgentLoader реализован с использованием .Net Framework версии 2.0 для возможности запуска на не обновлённых системах. Он является отдельной исполняемой частью решения и игнорируется при проверке хеш-сумм клиента.

|  |  |
| --- | --- |
| **Метод** | **Описание** |
| CheckServerConnection() : Bool | Данный метод проверяет связь с сервером |
| GetFilesSums() : List<Tuple<String, String>> | Данный метод получает с сервера набор имён файлов и их хеш-суммы |
| CheckClient(List<Tuple<String, String>>) : Bool | Данный метод проверяет сборку клиента на версию файла и целостность |
| GetUpdate(String) : FileStream | Данный метод проверяет связь с сервером |
| RunClient() : Bool | Данный метод проверяет связь с сервером |

**Таблица 3.4.3.1.1** Описание методов класса (компонента) ClientAgentLoader

#### ClientAgent

Данный модуль является второй исполняемой частью клиента после модуля ClientAgentLoader. В данном модуле реализована исполнительная часть системы повышения эффективности ИТ-службы предприятия. Он был реализован отдельной частью системы для возможности обновления и использует .Net Framework версии 3.5. В качестве конфигурационного модуля используется стандартный для .Net Framework механизм .config файлов и namespace Properties.Settings.

В данный модуль включены модули Logger, SQLiteORM, HowTo, Actions, Starter, ServerConnector, SolutionApplicator, их подробное описание приведено в следующих главах работы.

#### Logger

Модуль Logger используется для логирования работы ClientAgent. Он является фасадом над используемой ClientLoader библиотекой Apache log4net. Log4net является портом фреймворка log4j на платформу Microsoft .Net и считается стандартом при выборе способа логирования в крупных приложениях. Фреймворк схож с log4j, но при этом обладает преимуществами платформы .Net. Возможности настройки библиотеки очень широки и позволяют настраивать логирование по уровням в любую удобную форму.

В Logger было реализовано несколько уровней логирования, настраиваемых через конфигурацию ClientAgent

|  |  |
| --- | --- |
| **Метод** | **Описание** |
| Init(): Bool | Данный метод инициализирует механизм логирования |
| Log(String) | Данный метод логирует сообщение с уровнем логирования выставленным в конфигурации приложения |
| Debug(String) | Данный метод логирует сообщение с уровнем логирования Debug, необходимым для отладки приложения |
| Error(String) | Данный метод логирует сообщение с уровнем логирования Error, в случае возникновения ошибки в работе приложения |
| Fatal(String) | Данный метод логирует сообщение с уровнем логирования Fatal, в случае фатальной для работы приложения ошибки |
| Info(String) | Данный метод логирует сообщение с уровнем логирования Info, необходимым для информационных сообщений работы клиента |
| Warn(String) | Данный метод логирует сообщение с уровнем логирования Warn, необходимым для предупреждения о проблеме в работе приложения |

**Таблица 3.4.3.3.1** Описание методов класса (компонента) Logger

#### SQLite

В качестве локального хранилища данных была использована реляционная база данных SQLite, как компактное, не требующее дополнительно установленного на компьютере пользователя хранилище данных. Движок SQLite не требует серверной или клиентской части, а является набором файлов, хранящих данные и библиотекой для доступа к ним. Такой подход уменьшает накладные расходы, время отклика и упрощает программу. SQLite, как и log4net можно назвать стандартом для приложений подобного рода.

#### SQLiteORM

Для работы с данными приложения, хранящимися в SQLite, был использован Entity Framework, являющийся частью фреймворка .Net Framework. ADO.NET Entity Framework - это объектно-ориентированная технология доступа к данным, предоставляющая возможность работы с объектами, хранящимися в базе данных как посредством SQL запросов, так и посредством LINQ запросов. Используя некоторые особенности языка, LINQ позволяет использовать SQL-подобный синтаксис непосредственно в коде программы, написанной, на языке C#.

#### Action

Данный класс является родительским абстрактным классом для всех возможных действий, выполняемых с системой пользователя, на которой выполняется ClientAgent. Все действия, выполняемые с системой логируются на случай, если решение, найденное TU, не решит проблему пользователя. Родительский класс имеет поля количества попыток выполнения действия, флаг продолжения в случае, если шаг не был закончен успешно.

|  |  |
| --- | --- |
| **Метод** | **Описание** |
| Launch() : Bool | Данный метод запускает действие над операционной системой пользователя |
| CheckResult() : Bool | Данный метод проверяет успешно ли выполнен Action |

**Таблица 3.4.3.8.1** Описание методов класса (компонента) Action

##### Delete

Данный класс является наследником класса Action и используется для удаления файлов. В качестве параметра конструктора класса принимает путь до файла или директории.

##### Run

Данный класс является наследником класса Action и используется для выполнения системных вызовов. В качестве параметра класс принимает команду необходимую для исполнения. Может выполнять команды в окружении CMD или PowerShell.

##### Download

Данный класс является наследником класса Action и используется для скачивания сетевых ресурсов на машину клиента. В качестве параметра класс принимает ссылку на объект. Поддерживает протоколы http, https, ftp, smb.

##### Upload

Данный класс является наследником класса Action и используется для выгрузки сетевых ресурсов на машину клиента. В качестве параметра класс принимает ссылку на объект. Поддерживает протоколы http, https, ftp, smb.

##### Copy

Данный класс является наследником класса Action и используется для копирования файлов. В качестве параметров конструктора класс принимает путь до файла или директории и путь, куда следует скопировать файл, файлы или директорию.

##### Move

Данный класс является наследником класса Action и используется для перемещения файлов. В качестве параметров конструктора класс принимает путь до файла или директории и путь, куда следует переместить файл, файлы или директорию.

##### XMLConfig

Данный класс является наследником класса Action и используется для изменения XML конфигураций. В качестве параметров конструктора класс принимает путь до конфигурационного файла, ноду необходимую для изменения, новое значение ноды, тип изменения.

##### Zipper

Данный класс является наследником класса Action и управления zip архивами. В качестве параметров конструктора класс принимает тип операции (архивирование или разархивирование) путь до файла или директории и путь, куда следует переместить архив, или содержимое архива.

#### ServerConnector

Компонент ServerConnector осуществляет связь с сервером TU. Для связи используется WSDL описанный при реализации TU.

|  |  |
| --- | --- |
| **Метод** | **Описание** |
| Init(String): Bool | Данный метод инициирует ServerConnector |
| CheckConnection() : Bool | Данный метод проверяет наличие связи с сервером TU |
| GetActionsToExecute(String) : List<ActionsToExecute> | Данный метод проверяет наличие действий в очереди TU предназначенных для решения проблем пользователя |

**Таблица 3.4.3.10.1** Описание методов класса (компонента) ServerConnector

#### ActionsToExecute

Данный класс описывает список действий (Actions) необходимых для выполнения над системой пользователя, обратившегося к TU. ActionsToExecute генерируется при помощи ServerConnector при выполнении запроса к серверу, посредством метода GetActionsToExecute.

|  |  |
| --- | --- |
| **Метод** | **Описание** |
| Start() : Bool | Данный метод запускает выполнение цепочки действий Actions для решения проблемы пользователя |
| Stop() : Bool | Данный метод останавливает выполнение цепочки |

**Таблица 3.4.3.6.1** Описание методов класса (компонента) ActionsToExecute

#### Starter

Модуль Starter является инициализатором компонент приложения. Его функционал используется при старте модуля ClientAgent и подготавливает приложение к работе. Во время работы Starter выполняется инициализация Logger, проверка связи с сервером через ServerConnector, инициализация SQLiteORM и запуск SolutionApplicator выполняющий основной цикл приложения.

|  |  |
| --- | --- |
| **Метод** | **Описание** |
| InitСonfig() : Bool | Данный метод инициализирует конфигурационную систему приложения |
| InitLogger() : Bool | Данный метод инициализирует модуль Logger |
| InitSQLiteORM() : Bool | Данный метод инициализирует модуль SQLiteORM |
| InitServerConnectro() : Bool | Данный метод инициализирует модуль ServerConnector |
| StartWorkFlow() : Bool | Данный метод инициализирует основной поток приложения инициализируя и запуская модуль SolutionApplicator |

**Таблица 3.4.3.9.1** Описание методов класса (компонента) Starter

#### SolutionApplicator

Данный класс представляет собой главный цикл приложения, в котором реализована основная логика ClientAgent. После вызова метода StartWorkFlow модуля Starter инициируется экземпляр класса SolutionApplicator. Используя ServerConnector и SQLiteORM, SolutionApplicator проверяет наличие действий в очереди TU предназначенных для решения проблем пользователя. В случае если такие действия есть, то SolutionApplicator получает объект типа ActionsToExecute, хранящий в себе список действий Action необходимых для решения проблемы пользователя. Далее вызывается метод Start класса ActionsToExecute, который по очереди вызывает метод Launch каждого Action отдельно. После выполнения всех Action из ActionsToExecute SolutionApplicator «засыпает» на некоторое время до следующей проверки. Одновременно может быть запущено несколько экземпляров SolutionApplicator.

|  |  |
| --- | --- |
| **Метод** | **Описание** |
| Init(Int32): Bool | Данный метод инициализирует SolutionApplicator |
| Start() : Bool | Данный метод запускает цикл SolutionApplicator |
| Stop() : Bool | Данный метод останавливает цикл SolutionApplicator |

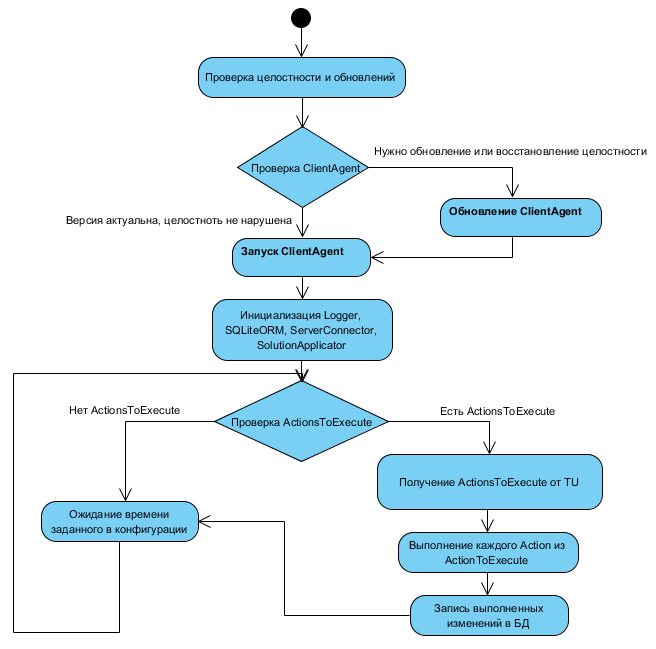
**Таблица 3.4.3.11.1** Описание методов класса (компонента) Logger

### Взаимодействие компонент ClientAgent

В данной главе приведен рабочий процесс компонент модуля ClientAgent:

1. При запуске операционной системы модуль ClientAgentLoader проверяет соединение с сервером TU и целостность файлов ClientAgent. Если проверка выявила необходимость обновления ClientAgentLoader обновляет ClientAgent;
2. После успешной проверки ClientAgent запускается модуль Starter, который инициализирует настройки приложения, настройки логирования Logger, модуль базы данных SQLiteORM, модуль связи с сервером ServerConnector.
3. Модуль Starter инициализирует модуль SolutionApplicator
4. SolutionApplicator проверяет с помощью ServerConnector наличие действий предназначенных клиенту для решения проблем;
5. В случае если ActionsToExecute найдены – SolutionApplicator применяет каждый Action в ActionsToExecute для применения решения проблемы пользователя;
6. После применения цепочки Action из ActionsToExecute, SolutionApplicator ждёт время, заданное в конфигурации приложения после чего, возвращается к пункту 4.

Диаграмма деятельности описывающая процесс работы ClientAgent изображена на картинке 3.4.4.1.



**Рисунок 3.4.4.1** Диаграмма деятельности ClientAgent



**Рисунок 3.4.4.2** Диаграмма классов ClientAgent

## Вывод

В данной главе приведено детальное описание всех компонентов и подкомпонентов фреймворка TU. Для лучшего понимания представлены описание механизма взаимодействия компонентов и общий сценарий использования системы:

1. Поступает запрос от пользователя: «User had received wrong application. User has ordered Google Chrome Ent. version However, she received wrong version, she received Google Chrome Canary instead of Ent. version. Please assist» («Пользователь получил неверное приложение. Пользователь заказал приложение «Google Chrome Ent. Версию», но получила неверную версию, - «Canary». Пожалуйста, помогите»);
2. Компонент GoalManger (Менеджер целей) устанавливает цель системы HelpUser (Помочь пользователю);
3. Главный компонент ThinkingLifeCycle TLC активирует набор компонентов Critic (Критик), привязанный к данной цели (HelpUser);
4. Активируется компонент PreliminaryAnnorator (Предварительный обработчик), который разбирает запрос, проводя орфографическую коррекцию и предварительный разбор;
5. Активируется компонент PreliminaryAnnorator (Предварительный обработчик), который разбирает запрос, проводя орфографическую коррекцию и предварительный разбор;
6. Компонент KnowledgeBaseAnnotator (разбор при помощи накопленных знаний) создает семантическую сеть и ссылки на нее;
7. Компонент Critic (Критик), привязанный к цели HelpUser на Рефлексивном уровне, запускает WayToThink (Образ мышления)
8. Компонент Critic на Рефлексивном уровне выбирает WayToThink KnowingHow (Поиск рецепта решения);
   1. Запускаются параллельно все компоненты класса Critic, которые привязаны к цели ResolveIncident (Решить проблему), в данном случае это DirectInstruction (прямые инструкции), ProblemWithDesiredState (проблемы с желаемым состоянием), ProblemWithoutDesiredState (проблема без желаемого состояния);
   2. Компонент Selector (Селектор) выбирает среди всех результатов наиболее вероятный результат работы. В данном случае им будет Problem Description with desired state (Проблема с желаемым состоянием);
   3. Компонент KnowingHow сохраняет варианты выбора Selector;
   4. Компонент Simulation (Моделирование) WayToThink с параметрами «создать модель текущий ситуации» создает: концепцию существующей ситуации (CurrentState), концепцию пользователя, концепцию программного обеспечения;
   5. Компонент Reformulation WayToThink (Компонент дополнения), используя результаты предыдущего шага, синтезирует артефакты, которых не хватает, чтобы получить из CurrentState DesiredState (Желаемое состояние), так как он не указан явно. WayToThink запускает Critic размышления, чтобы найти корень проблемы. Он находит CurrentState (настоящее состояние) - Wordfinder Tehcnical и DesiredState (состояние, которое нужно пользователю) - Wordfinder Business Economical;
   6. Рефлексивные Critic оценивают состояние системы - на каком шаге она находится, и если цель не достигнута, то запускают другой WayToThink, например, DirectInstruction;
   7. Компонент Critic Solution Generator (Компонент генерации решения) запускает KnowingHow WayToThink, ExtensiveSearch (Поиск решения);
   8. Компонент Selector выбирает наиболее вероятный образ мышления. В данном случае это будет ExtensiveSearch, который будет находить решения, позволяющие привести систему в необходимое пользователю состояние (DesiredState), если сделать это невозможно, то система инициирует коммуникацию с пользователем.
   9. В случае если выбранный образ мышления решает проблему пользователя в очередь ActionsToExecute добавляются действия, которые необходимо выполнить на системе пользователя.
   10. Компонент ClientAgent получает ActionsToExecute и применяет сгенерированные Actions на целевой ОС.
9. Рефлексивный Critic проверяет состояние системы. Если Цель достигнута, то пользователю посылается ответ, информирующий об этом. На данном шаге активируются компоненты класса Critic на cамосознательном уровне, которые сохраняют информацию о затратах на решение.

Подробным образом были рассмотрены каждый из основных в работе компонентов ThinkingUnderstanding. Также детальным образом был рассмотрен исполнительный механизм части фреймворка TU, реализованный в рамках данной работы.

# РЕЗУЛЬТАТЫ РАБОТЫ

## Результаты

Система была протестирована на выгрузках проблем информационных систем, которые были предоставлены ОАО «АйСиЭл КПО ВС».

## Тесты скорости работы системы и оценка производительности

Для верификации экспертной системы поддержки принятия решений TU была выбрана область поддержки информационной инфраструктуры предприятия. На выборке, предоставленной «АйСиЭл» запускался программный комплекс и анализировались результаты. Удалось добиться 95% успешных инцидентов. Успешным считался инцидент, который был успешно сопоставлен концепциям в базе знаний, результатом успешной обработки инцидента считалось найденное решение, если же решения не было, то проверялось правильное понимание системой всех концепций, так как решения не было в базе знаний. Запуск работы системы производился при помощи автоматизированных тестов. Проверка данных также осуществлялась при помощи этой технологии. Система также может функционировать в режиме диалога и в консольном варианте, в этом режиме видно взаимодействие с пользователем.

Система показала свою жизнеспособность на модельных данных. Были проведены тесты в сравнении с работой человеческого специалиста. Был выбран контрольный список инцидентов. Сравнивался поиск решения для инцидентов. В приведенной далее таблице видны результаты сравнения. Тесты были выполнены на компьютере Intel Core i5 2200 MHz, 8GB RAM, 160 GB HDD, Windows7.

|  |  |  |
| --- | --- | --- |
| Инцидент | TSS1 (.мс) | TU(.мс) |
| Please install Firefox (Установите Firefox) | 9000 | 859 |
| User miss Internet Explorer 8 (У пользователя нет Internet Explorer 8) | 10000 | 10589 |
| User needs document portal update (Пользователю требуется обновление документов) | 15000 | 16543 |
| Add new alias Host name on host that alias is wanted to: hrportal.lalala.biz IP address on host that alias is wanted to: 127.150.132.22 Wanted Alias: webadviser.lalala.net (Добавьте, пожалуйста, новую ссылку на hrportal.lalala.biz через 322.223.333.22) | 10000 | 18432 |
| Outlook Web Access (CCC) — 403 — Forbidden: Access is denied (Нет доступа к Outlook Web Access (CCC)) | 15000 | 10342 |
| PP2C — Cisco IP communicator. Please see if you can fix the problem with the ip phone, it’s stuck on configuring ip + sometimes Server error rejected: Security etc (PP2C — коммуникатор Cisco IP. Пожалуйста, помогите исправить проблему с ИП-телефоном, он застревает во время кон- фигурирования и иногда показывает ошибку «Безопасность») | 13000 | 12343 |

**Таблица 4.2.1** Описание методов класса (компонента) Logger

Основное время при работе специалиста технической поддержки тратилось на коммуникацию с пользователем и манипуляции с файлами установки, скриптами, настройками. Также явным преимуществом фреймворка ThinkingUnderstanding перед классической моделью поддержки является постоянная работа сервиса, в отличие от службы поддержки, в которой работают живые люди, имеющие распорядок дня, обеденное время, не исключающие множество так называемых «человеческих факторов».

Основной проблемой для системы составляют инциденты с большой неоднозначностью, например, «I should have Internet Explorer, but Firefox was installed». Здесь непонятно, нужен ли пользователю браузер Firefox или нет. В этом случае система должна выявить проблему о необходимости пользователю Internet Explorer. Другой пример, который трудно однозначно решить, используя классические подходы: I install Internet Explorer previously, but i need Chrome. Здесь есть следующие наборы концепций: i, install, Internet Explorer; i, need, Chrome. Используя регулярные выражения, однозначно решить не удастся, но, используя интеллектуальное решение, эту проблему решить можно. В рамках TU сработает более вероятный Critic, который определит проблему «need Chrome», базируясь на наличии концепции «previously». В таблице 4.2.2 приведены результаты работы системы в разрезе категорий инцидентов.

|  |  |
| --- | --- |
| Класс проблемы | % успешных |
| Проблема с ПО | 64% |
| Проблемы во время работы | 10% |
| Как сделать | 10% |
| Проблема с оборудованием | 0% |
| Установить новое ПО | 95% |
| Проблема с печатью | 80% |
| Нет доступа | 60% |

**Таблица 4.2.2** Описание экспериментальных данных

Ожидаемым был результат в 51% (доля разрешенных проблем, поставленных пользователями), но и достигнутый результат в ~30% можно считать приемлемым, так как он значительно увеличивает эффективность разрешения проблемных запросов.

## Вывод

В главе рассмотрены экспериментальные данные, которые были использованы для верификации системы. На основе экспериментов была посчитана скорость работы системы в сравнении со специалистом технической поддержки. Были приведены сложные для решения примеры входных запросов пользователя и дан их разбор.

# Заключение

По результатам анализа всех моделей построения интеллектуальных вопросно - ответных систем была выбрана та из них, которая базируется на модели мышления Марвина Мински. Модель TU позволяет выбрать наиболее оптимальное сочетание возможности прикладной реализации и учета универсальной гибкости человеческого мышления. Иными словами, модель не копирует процесс человеческого мышления, а дает возможную его интерпретацию, тем самым позволяя реализовать ее в условиях ограниченных вычислительных ресурсов.

В результате выполнения выпускной квалификационной работы был проанализирован фреймворк ThinkingUnderstanding, спроектирована архитектура исполнительной части фреймворка - ClientAgent и проведены испытания реализованной системы.

Процесс выполнения выпускной квалификационной работы включал в себя:

1. Обзор существующих систем повышения эффективности ИТ-службы предприятия;
2. Анализ модели теории массового обслуживания ИТ инфраструктуры;
3. Анализ фреймворка ThinkingUnderstanding;
4. Разработка требований к исполнительной части системы повышения эффективности ИТ-службы предприятия;
5. Разработка архитектуры ClientAgent с учётом поставленных требований;
6. Реализация модуля ClientAgent согласно разработанной архитектуре;
7. Проверка работоспособности решения, проведение тестов, замеры скорости работы на контрольных примерах.

В рамках работы использовались и обрабатывались данные, собранные во время работы команд ОАО «АйСиЭл КПО ВС» над поддержкой информационной структуры предприятий - заказчиков.

# Список использованных источников

1. **Social big data: Recent achievements and new challenges** [Журнал] / авт. G.A Bello-Orgaz, J.J. Jung, D.A. Camacho. - [б.м.] : Information Fusion, 2015 г.. - Т. 28.
2. **Problem-oriented patient record summary: An early report on a Watson application** [Конференция] / авт. M. Devarakonda, D. Zhang, C.H. Tsou // 2014 IEEE 16th International Conference on e-Health Networking. - [б.м.] : Applications and Services, 2014
3. **TOP 10 MACHINE LEARNING APIS** [В Интернете] / авт. Wagner J // T SPEECH, IBM WATSON, Google prediction. - 2015 г.. - <http://goo.gl/FQ9G6G>.
4. **Clause-based approach to extracting problem phrases from user reviews** [Журнал] / авт. V. Ivanov, E. Tutubalina. - [б.м.] : Communications in Computer and Information Science, 2014 г.. - Т. 436.
5. **HP OpenView System Administration Handbook** [Книга] / авт. A. Sokolov, К. Serdobincev. - [б.м.] : Astrahan, 2004.
6. **Automation of incidents’ recording process in the network of the mobileradio communication of standard GSM-900/1800** [Конференция] / авт. A. Tsvetkov, O. Ponomareva, M. Yurina // International Crimean Conference Microwave and Telecommunication Technology. - Crimea. : [б.н.], 2014. - Т. 1.
7. **Trustworthy and resilient monitoring system for cloud infrastructures** [Конференция] / авт. S. Padhy, D. Kreutz, A. Casimiro, M. Pasin // 12th International Middleware Conference. - Middleware : [б.н.], 2011. - Т. 1.
8. **Migration of the CERN IT data centre support system to servicenow** [Журнал] / авт. R.A. Alonso, G. Arneodo, O. Barring. - [б.м.] : Journal of Physics, 2013 г.. - Т. 513. - стр. 7-23.
9. **Open Science Grid (OSG) ticket synchronization: Keeping your home field advantage in a distributed environment** [Журнал] / авт. K. Gross,S. Hayashi, S. Teige, R. Quick. - [б.м.] : Journal of Physics, 2012 г.. - Т. 396. - стр. 1-80.
10. **Relation extraction and scoring in DeepQA** [Журнал] / авт. C. Wang, A. Kalyanpur, J. Fan // IBM journal of Research and Development.. - 2012 г.. - Т. 56. - стр. 1-12.
11. **Watson: Beyond jeopardy!** [Журнал] / авт. D. Ferrucci, A. Levas, S. Bagchi. - [б.м.] : Artificial Intelligence, 2013 г.. - Т. 10.1016. - стр. 93-105.
12. **Understanding and summarization** [Журнал] / авт. Alterman R.. - 1991 г.. - Т. 5. - стр. 239-254.
13. **John McCarthy — Father of artificial intelligence** [Журнал] / авт. Rajaraman V // Resonance. - 2014 г.. - Т. 19. - стр. 198-207.
14. **Detecting friendly, flirtatious, awkward, and assertive speech in speed-dates** [Журнал] / авт. D. Jurafsky, J. Martin. - [б.м.] : Computer Speech and Language, 2013 г.. - Т. 27. - стр. 89-115.
15. **Deep Blue** [Журнал] / авт. M. Campbell, Hoane Jr., F.-H. A.Joseph, Hsu. - [б.м.] : Artificial Intelligence, 2002 г.. - Т. 134. - стр. 57-83.
16. **Artificial Intelligence** [Журнал] / авт. S. Russel, P. Norvig. - New York, USA, Pearson : A Modern approach, 2010 г.. - стр. 1152.
17. **Bias of importance measures for multi-valued attributes and solutions** [Конференция] / авт. H. Deng, G. Runger, E. Tuv // Proceedings of the 21st International Conference on Artificial Neural Networks (ICANN). - 2011. - стр. 293-300.
18. **The Emotion Machine** [Книга] / авт. Minsky M.. - [б.м.] : Simon & Schuster, 2007.
19. **A visual recommender tool in a collaborative learning experience** [Журнал] / авт. A. Anaya, M. Luque, M. Peinado. - [б.м.] : Expert Systems with Applications, 2016 г.. - Т. 45. - стр. 248-259.
20. **Owl: Representing Information Using the Web Ontology Language** [Книга] / авт. Lesly L.. - Blumington, Liberty drive 1663 : Tredford Publishing, 2005.
21. **Akka Concurrency** [Книга] / авт. D. White ; Ed. by K. Rolland. - [б.м.] : Artima, 2013.
22. **C++ Language Reference** [В Интернете]. - Microsoft. - <https://msdn.microsoft.com/en-us/library/3bstk3k5.aspx>.
23. **Scala Lang** [В Интернете]. - Мартин Одерски, 2003 г.. - <https://www.scala-lang.org/>.
24. **Java technology** [В Интернете]. - Oracle, 1995 г.. - <http://java.com>.
25. **Java SE Runtime Environment** [В Интернете]. - ORACLE. - <http://www.oracle.com/technetwork/java/javase/downloads/jre8-downloads-2133155.html>.
26. **C# | Microsoft Docs** [В Интернете]. - Microsoft, 2016 г.. - <https://docs.microsoft.com/ru-ru/dotnet/csharp/csharp>.

# Приложение 1. Исходный код

Ввиду большого объема исходный код фреймворка можно скачать по адресу

1. <https://github.com/tu-team/2>
2. https://drive.google.com/open?id=0B30tiEV73jYPRXBYNTFFSk1wb2M