Казанский (Приволжский) Федеральный Университет

На правах рукописи

УДК xxx.xxx

Тощев Александр Сергеевич

Разработка эффективного подхода обработки производственных задач прикладного характера в области обслуживания программного обеспечения и информационной инфраструктуры предприятия на основе стохастического поиска, вероятностно-логических рассуждений и машинного обучения

Специальность 05.13.01 —
«Системный анализ, управление и обработка информации (по отраслям)»

Диссертация на соискание учёной степени
Кандидат технических наук

Научный руководитель:
уч. степень, уч. звание
Елизаров А.М.

Казань — 2015

Оглавление

Введение

В настоящее время в области IT набрало большую популярность системы удаленной поддержки информационной инфраструктуры, так называемый «Аутсорсинг». Ввиду развития рынка компаниям становится невыгодно держать свой штат службы поддержки, и они отдают свою инфраструктуру сторонней компании. Ввиду возросшей интенсивности данного бизнеса возникла потребность автоматизации работы. В данном контексте рассматривается автоматизация обработки инцидентов, начиная с разбора инцидентов на естественном языке и заканчивая поиском решения и применением решения. Главными требованиями к системе являются

1. Обработка естественного языка
2. Возможность обучения
3. Общение с специалистом
4. Проведение логических рассуждений: аналогия, дедукция, индукция
5. Умения абстрагировать решение и экстраполировать его на другие решения

На данный момент многие компании ведут разработку подобных систем. Примером такой системы является набирающая популярность система IBM Watson [1]. Подобный класс система также называют вопросно-ответными системами. Другим примером является система Wolfram Alpha [2]. В данной работе был сделан акцент на попытку создания мыслящий системы на основе модели мышления Марвина Мински [3].

Целью данной работы является создание архитектуры и реализация базового прототипа программного комплекса обеспечивающего разбор и формализацию входного запроса пользователя и поиск решения данной проблемы.

Для достижения поставленной цели необходимо было решить следующие задачи:

1. Исследовать целевую область
2. Вычислить возможность автоматизации целевой области
3. Исследовать модель мышления Марвина Мински
4. Адаптировать модель для прикладной реализации
5. Создать архитектуру приложения на основе модели

6. Реализовать прототип на основе архитектуры

Основные положения, выносимые на защиту:

1. Возможность автоматизации области предоставления удаленной поддержки информационной инфраструктуры
2. Прикладное применение модели мышления Марвина Мински для решения задачи автоматизации
3. Возможность программной реализации модели мышления Марвина Мински
4. Экстраполяция программной системы для других областей

Научная новизна:

1. Впервые была представлена реализация модели мышления Мински на практике
2. Была представлена новая модель данных для модели мышления
3. Было выполнено оригинальное исследование модели мышления …

Научная и практическая значимость …

Степень достоверности полученных результатов обеспечивается результатами выполнения тестов на контрольных примерах. Результаты находятся в соответствии с результатами, полученными другими авторами и экспертными системами

Апробация работы Основные результаты работы докладывались на:

– RCDL-2014
– AINL-2013
– WCIT-2012
– AMSTA-2015

Словарь терминов

Личный вклад. Автор принимал активное участие в разработке архитектуры приложения, реализации прототипа, проработки теории, тестировании.

Публикации. Основные результаты по теме диссертации изложены в XX печатных изданиях [**?**, **?**, **?**, **?**, 4], X из которых изданы в журналах, рекомендованных ВАК [**?**, **?**, **?**], XX — в тезисах докладов [**?**, 4].

Объем и структура работы. Диссертация состоит из введения, четырех глав, заключения и двух приложений. Полный объем диссертации составляет XXX страница с XX рисунками и XX таблицами. Список литературы содержит XXX наименований.

Таблица 1: Глоссарий

Термин	Значения
База Знаний	База данных приложения, представленная в виде онтологии знаний
WayToThink	Путь мышления. Основан на определении Марвина Мински [3]. Класс объектов, которые модифицируют данные
Critic	Критик. Основан на определении Марвина Мински [3]. Класс объектов, которые выступают триггерами при наступление определенного события
ThinkingLifeCycle	TLC. Основан на определении Марвина Мински [3]. Класс объектов, которые выступают основными объектами для запуска в приложении - рабочими процессами

Глава 1

ПОСТАНОВКА ЗАДАЧИ ПОЛУЧЕНИЯ, АНАЛИЗА И ОБРАБОТКИ ЭКСПЕРТНОЙ ИНФОРМАЦИИ

1.1. Возникновение области

В настоящее время в области IT набрало большую популярность системы удаленной поддержки информационной инфраструктуры, так называемый «Аутсорсинг». Ввиду развития рынка компаниям становится невыгодно держать свой штат службы поддержки, и они отдают свою инфраструктуру сторонней компании. Большинство проблем, которые решает удаленная служба поддержки носят весьма тривиальный характер :

- Установить приложение
- Переустановить приложение
- Решить проблему с доступом к тому или иному ресурсу

Данные проблемы решают специалисты технической поддержки. Обычно техническая поддержка делится на несколько линий:

1. Первая линия. Решение уже известных, задокументированных проблем, работа напрямую с пользователем
2. Вторая линия. Решение ранее неизвестных проблем
3. Третья линия. Решение сложных и нетривиальных проблем
4. Четвертая линия. Решение архитектурных проблем инфраструктуры

Каждая линия поддержки представлена специалистами. В среднем команда, обслуживающая одного заказчика насчитывает 60 человек. Процентное соотношение специалистов разных линий поддержки отображено на Диаграмме 1.1

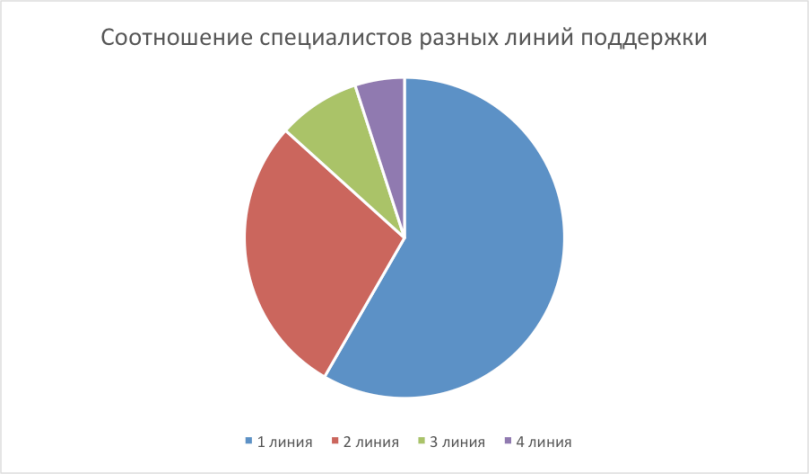

Рисунок 1.1: Диаграмма состава команд

Работа специалиста 1 линии поддержки состоит из множества рутинных и простых задач. На Диаграмме 1.2 показано соотношение разных типов проблем, встречающихся во время работы поддержки

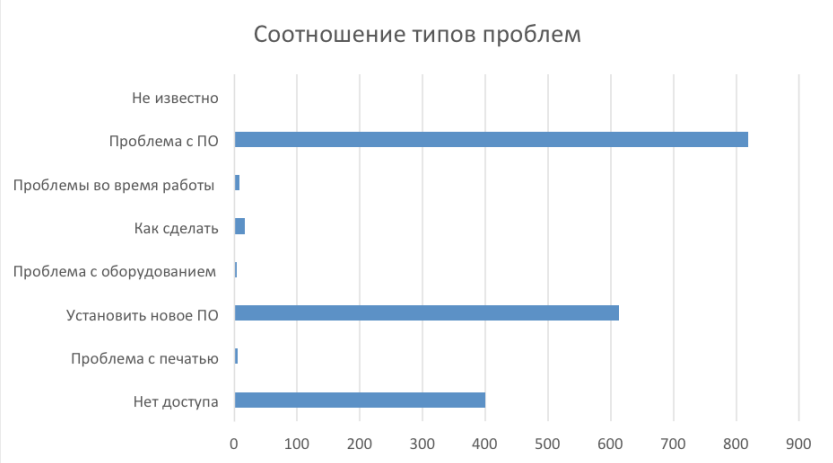

Рисунок 1.2: Диаграмма соотношений типов проблем

Решение части задач может быть автоматизировано, а специалисты получат дополнительное время на решение более интересных задач. Проблема заключается в автоматизации решения рутинных задач в области удаленной поддержки инфраструктуры.

1.2. Прогноз развития области

Основной тенденцией в развитии области удаленной поддержки инфраструктуры является попытки удешевить и улучшить стоимость предоставления услуг.

Компании, работающие на этом рынке вкладывают большие деньги в автоматизацию. Кроме того современное развитие науки и техники, а точнее вычислительных мощностей позволяет автоматизацию даже самых наукоемких процессов.

Дальнейшим развитием области является замена человеческих специалистов на автоматические

Таблица 1.1: Категории инцидентов

Категория	Описание
Проблема с ПО	Проблема при запуске ПО на компьютере. Решается переустановкой
Проблемы во время работы	Проблема с функционированием программного обеспечения
Как сделать	Запрос на инструкцию по работе с тем или иным компонентом рабочей станции
Проблема с оборудованием	Неполадки на уровне оборудования
Установить новое ПО	Требование установки нового программного обеспечения
Проблема с печатью	Установка принтера в систему
Нет доступа	Нет доступа к общим ресурсам

системы. Многие ведущие компании ведут разработки в этом направлении. Например, компания HP. Данная компания имеет свою системы по регистрации подобных инцидентов и сейчас ведется работа над автоматизацией системы.

1.3. Методологии, используемые в области IT аутсорсинга: ITIL и ITSM

В области IT аутсорсинга есть несколько готовых стандартов ведения работ. Одним из таких стандартов является библиотека ITIL. Данные стандарт описывает лучшие практики организации работ в области IT аутсорсинга. Используемый в библиотеки подход соответствует стандартам ISO 9000 (ГОСТ Р ИСО 9000) . Наличие стандартов в области диктует унифицированность постановки проблем, а также унифицированность алгоритмов решения. Такие предпосылки говорят о возможности частично или в некоторых случаях полной автоматизации решения проблем.

1.4. Постановка задачи

Задачами данного исследования являются:

– Изучение возможности автоматизации области удаленной поддержки инфраструктуры путем анализа области

– Выработка критериев и сравнительный анализ существующих решений в области

– Создание программного комплекса (фреймворка) для автоматизации поддержки удаленной инфраструктуры

– Подсчет статистических результатов работы комплекса

Глава 2

МЕТОДЫ И КОМПЛЕКСЫ ОБРАБОТКИ ЕСТЕСТВЕННОГО ЯЗЫКА

2.1. Обработка Эталонных Текстов

В данном разделе проводится обзор обработчиков естественного языка. За основу были взяты инциденты из выгрузки систем поддержки ОАО ”ICL КПО-ВС”.
Ввиду специфики области основным языком был выбран английский язык. Был сформирован список из типичных эталонных фраз, на которых тестировались обработчики естественного языка. Фразы были выявлены путем анализа существующих отчетов об инцидентах. Примерами инцидентов являются следующие инциденты:
Инцидент 1 *User had received wrong application. User has ordered Wordfinder Business Economical for her service tag 7Q4TC3J, there is completed order in LOT with number ITCOORD-18125. However she received wrong version, she received Wordfinder Tehcnical instead of Business Economical. Please assist.*
Инцидент 2 *Laptop – user has almost full C: but when he looks in the properties of the files and folders on C: they are only 40GB and he has a 55GB drive.*
Инцидент 3 *User cannot find Produkt Manageron start menu. Please reinstall.*
Инцидент 4 *User needs to have pdf 995 re-installed please.*

Во время анализа были использованы следующие обработчики естественного языка:

1. Open NLP [5]
2. RelEx [6]
3. StanfordParser [7]

Результат работы вычислялся при помощи метрик, представленных в Таблице2.1.
Результаты приведены на сводной диаграмме Рисунок 2.1

Таблица 2.1: Таблица метрик

Метрика	Описание	Формула
Аккуратность	Понимание текста обработчиком	$Ac = \frac{1-x}{y}$ где x- количество нераспознанных слов, y количество распознанных
Успешно обработанные	Успешно обработанные инциденты	$P = \frac{x}{100}$ где x успешно обработанные
Не успешно обработанные	Неуспешно обработанные инциденты	$N = \frac{y}{100}$ где y неуспешные инциденты
Результативность	Общая результативность обработчика	$R = \frac{P}{N}$
Общий бал	Общая оценка обработчика	$T = Ac + R$

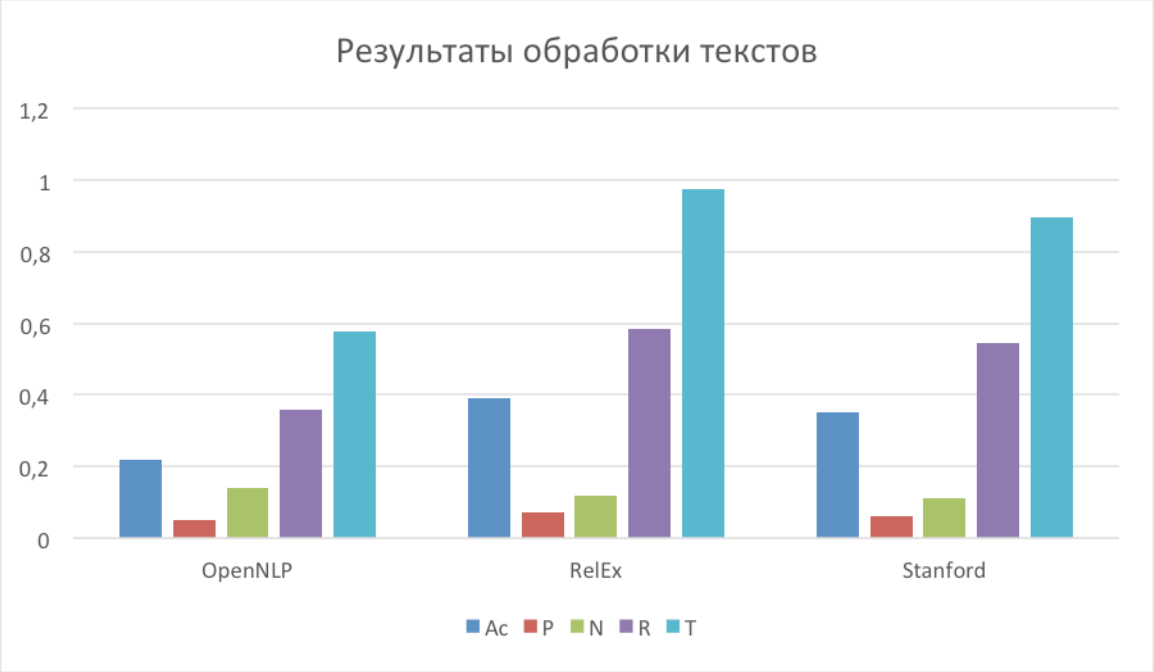

Рисунок 2.1: Результаты обработки текстов

Из диаграммы видно, что наилучшее результаты показывает обработчик RelEx [6]. После анализа необработанных инцидентов было выявлено нескольких проблем у всех обработчиков:

1. Невозможности корректировки простых грамматических ошибок, связанных с пропущенными пробелами или неверным форматированием. Ошибки первого типа.

2. Ошибки неверной интерпретации слов в предложении. Например, слово please интерпретировалось как глагол, хотя является по смыслу «формой вежливости». Ошибки второго типа.

2.2. Обработка текстов с ошибками

По результатам прошлого раздела было решено выбрать в качестве обработчика естественного языка RelEx, но были выявлены некоторые проблемы. Было принято решение исправить данные проблемы при помощи предварительной обработки текста. Предварительная обработка текста была разбита на несколько фаз:

1. Комплексная корректировка ошибок

2. Обработка при помощи внутренней базы знаний

Для того, чтобы избавится от орфографический, синтаксических ошибок используется составной корректировщик. Данный компонент имеет модульную структуру и применяет корректировку последовательно.

Для данного компонента были написаны модули корректировки:

– Google API

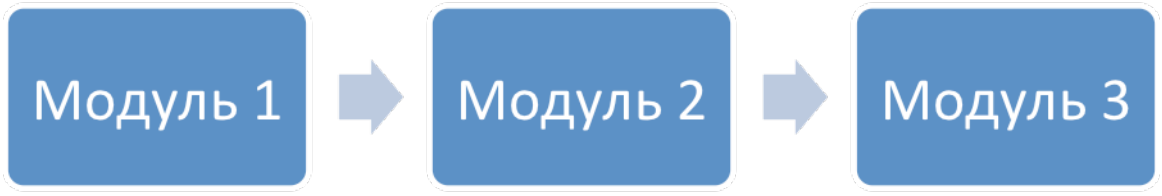

Рисунок 2.2: Архитектура предварительной обработки текста

– After The Deadline

Таким образом удалось исправить большинство ошибок, связанный с синтаксисом, грамматикой, орфографии. Также удалось исправить ошибки неверного написания: лишних пробелов, пропущенных запятых, пропущенных точек. По-прежнему остается проблема обработки неверной интерпретации слов в тексте.

Для корректировка ошибок второго типа был использовано инъекция в работу обработчика RelEx. В ввиду OpenSource природы проекта, модульности был подменен модуль извлечения и обработки слов в предложения. Стандартный процесс обработки был разбит на «предобработку» и «обработку». Стадия «обработки» включала в себя алгоритм работы такой же как был до этого в модули, на стадии «предобработки» управление передается модулю основному приложению, который проверяет данное слово на предмет его вхождения во внутреннюю Базу Знаний и если таковое имеется, то приложение передает соответствующие корректировки в модуль

2.3. Сравнение средств обработки русского и английского языка

Средства обработки естественного языка принято относить в большому классу средств NLP – Natural Language Processing. Для английского языка существует множество открытых средств обработки естественного языка, для русского языка найти их гораздо сложнее. Рассмотрим архитектуру средств обработки естественного языка на примере OpenCog RelEx.
OpenCog RelEx использует результаты обработки Link Grammar [8]. Link Grammar поддерживает

множество языков: английский, русский, турецкий, немецкий и т.д. RelEx использует вывод LG и преобразует его в формат связей. **Пример 1**. User is unable to start KDP web, please reinstall Java.
Результат

```
_obj(start, KBP)
pos(start, verb)
inflection-TAG(start, .v)
tense(start, present)
pos([web], WORD)
noun_number(KBP, singular)
definite-FLAG(KBP, T)
pos(KBP, noun)
_advmod(reinstall, please)
pos(reinstall, verb)
inflection-TAG(reinstall, .v)
tense(reinstall, present)
pos(please, adv)
inflection-TAG(please, .e)
noun_number(Java, singular)
definite-FLAG(Java, T)
pos(Java, noun)
pos(., punctuation)
_obj(,, Java)
pos(,, verb)
tense(,, infinitive)
HYP(,, T)
_to-do(unable, ,)
pos(unable, adj)
inflection-TAG(unable, .a)
tense(unable, present)
pos(to, prep)
inflection-TAG(to, .r)
pos(be, verb)
inflection-TAG(be, .v)
_predadj(User, unable)
noun_number(User, singular)
definite-FLAG(User, T)
pos(User, noun)
```

Возьмем разбор слова start. В результате мы получаем несколько отношений:

– pos(start, verb) - start глагол

– tense(start, present) - время настоящее

– inflection-TAG(start, .v) - метод обозначения на схеме (индекс)

На их основе можно формализовать приложение на естественном языке. Остальные парсера пока не поддерживают русский язык. Существуют открытые проекты, но они еще недостаточно развиты.

2.4. Вывод по главе

В данный главе был выполнен основной анализ обработчиков естественного языка. Ввиду развитости и доступности было решено использовать OpenCog RelEx.

Глава 3

АНАЛИЗ ТЕКУЩИХ РЕШЕНИЙ ПОЛУЧЕНИЯ, АНАЛИЗА И ОБРАБОТКИ ЭКСПЕРТНОЙ ИНФОРМАЦИИ В ОБЛАСТИ ОБСЛУЖИВАНИЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ И ИНФОРМАЦИОННОЙ ИНФРАСТРУКТУРЫ

3.1. Обзор решений

HP OpenView [9] является комплексным программным решением по мониторингу ИТ инфраструктуры предприятия. Система имеет множество модулей. Данная система охватывает широкий спектр возможностей:

- Мониторинг
- Регистрация инцидентов
- Управление системами

Система не поддерживает:

- Понимание и формализация запросов
- Автоматическое исправление проблемы на основе формализации запроса

ServiceNOW Средства автоматизации сервиса. Предоставляет следующие возможности:

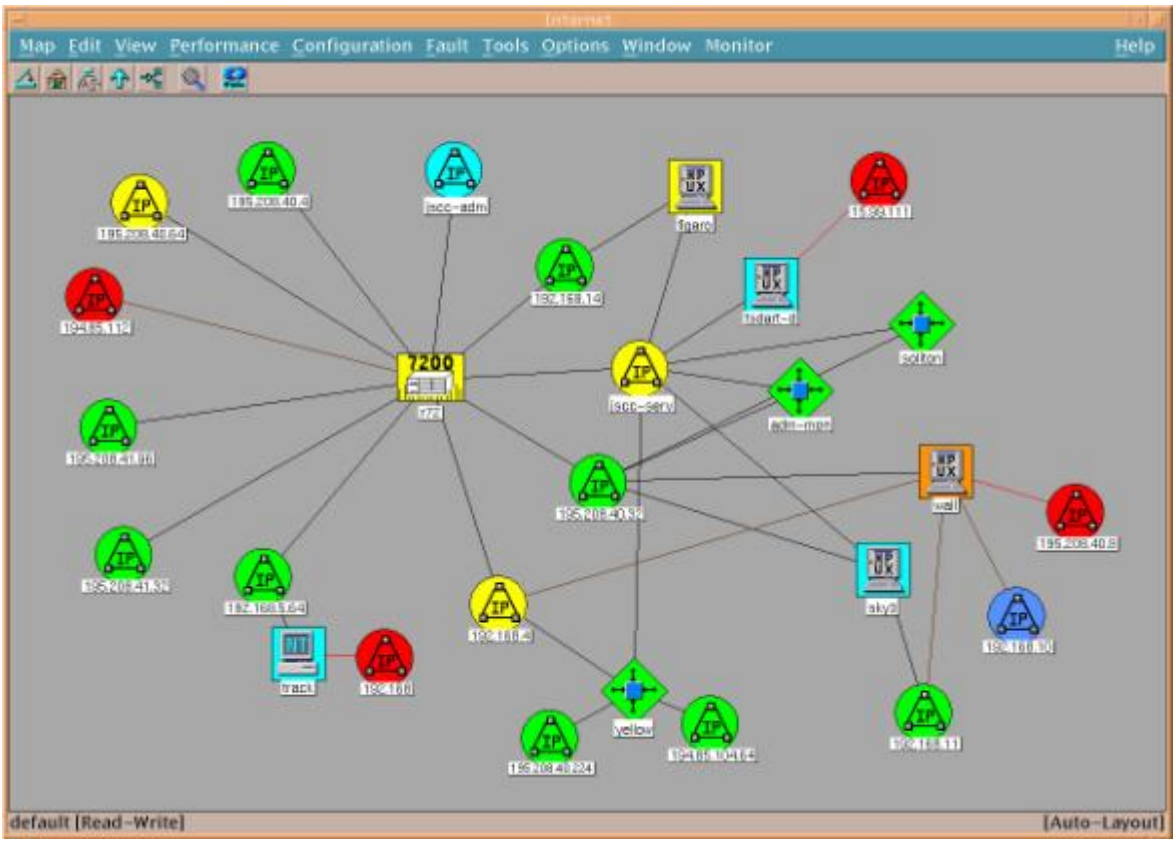

Рисунок 3.1: HP OpenView

– Регистрация инцидентов

– Создание цепи обработки инцидента

Система не поддерживает:

– Понимание и формализация запросов

– Автоматическое исправление проблемы на основе формализации запроса

Прочие системы Кроме того существуют дополнительные способы автоматизации

– Обработка инцидентов посредством регулярных выражений. В таком решение нет гибкости, так как обработка идет путем поиска ключевых слов вне контекста

– Обработка инцидентов при помощи скриптов. Автоматизирует лишь рутинные операции

3.2. Требования к системе

Основными требованиями к системе является следующие:

– Понимание входящий информации на естественном языке

– Формализация инцидента в контексте

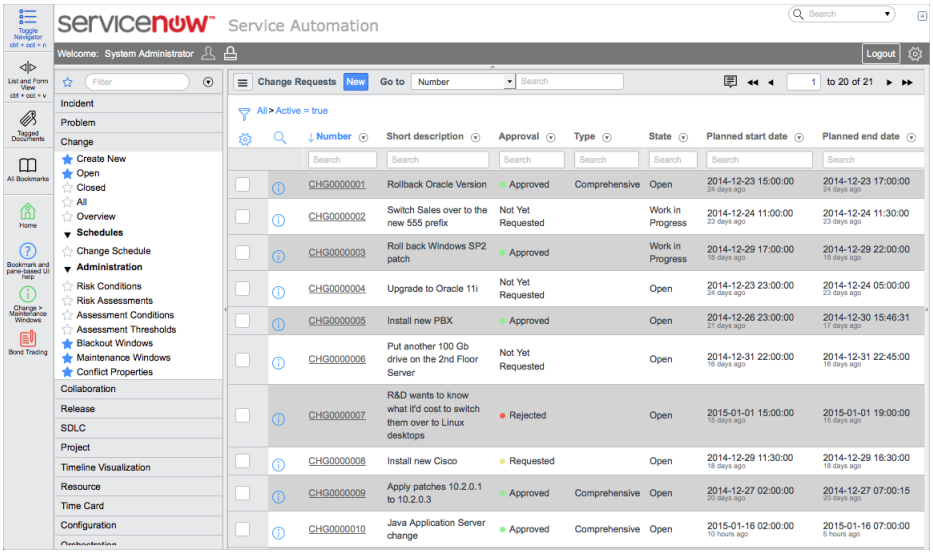

Рисунок 3.2: Service NOW

– Поиск решение инцидента

– Обучение решению инцидента

– Умение проводить логические рассуждения: генерализацию, специализацию, синонимичный поиск

– Умение мыслить

Требования к системе формировались исходя из возможностей специалистов поддержки, а также анализа проблем, которыми они занимаются. Большинство инцидентов тривиальные и типичные, но все они разные. Для человека проблема ”Please insall Firefox” и ”Please install Chrome” идентичные, но с точки зрения формализации - нет. Общее в них можно найти взглянув на генерализацию различающейся части. Firefox и Chrome являются пакетами программного обеспечения.

Глава 4

ТЕОРЕТИЧЕСКОЕ ОБОСНОВАНИЕ СИСТЕМЫ

4.1. Модели мышления

В работе было рассмотрено несколько моделей мышления:

- Модель мышления Питера Норвина [10]
- Модель мышления Марвина Мински [3]
- Модель мышления на базе нейронных сетей

Модель Питера Норвига носит более абстрактный характер и тяжела для реализации в выбранной области. Модель мышления на базе нейронных сетей требует больших вычислительных мощностей. Модель Мински наиболее подходит для целевой области - автоматического решения инцидентов.

4.2. Модель мышления Марвина Мински

4.2.1. Крити-Селектор-Путь мышления

В 2006 году Марвин Мински опубликовал свою книгу ”The emotion machine” [3], в которой предложил свой взгляд на систему мышления и памяти человека. В основу теории легла парадигма триплета Критик-Селектор-Путь мышления, k-line для сопоставления знаний. На рисунке 4.1 представлена схематичное изображение Критика-Селектора-Пути мышления

Критик представляет собой определенный триггер: внешние обстоятельства, события или иное воздействие. Например, включился свет и зрачки сузились. Обожглись и одернули руку. Критик активируется только когда для этого достаточно обстоятельств. Одновременно могут активироваться несколько критиков. Например, человек решает сложную задачу. Идет активация

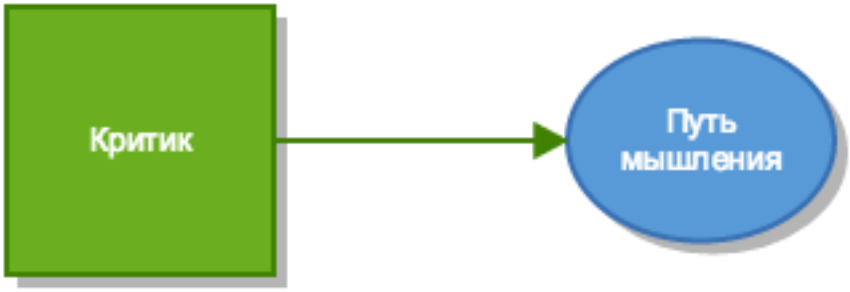

Рисунок 4.1: Критик-Селектор-Путь мышления

множество критиков: считать, технические детали, кроме того параллельно может активироваться критик переработки, сообщающей о необходимости отдыха.
Селектор занимается выбором определенных ресурсов, которыми также являются Пути мышления.
Путь мышления это способ решения проблемы. Путь мышления также может активировать следующий критик.

На рисунке 4.2 представления расширенная модель работы триплета Критик-Селектор-Путь мышления. Критик активирует селектор, который активирует путь мышления (синий круг). Путь мышления в свою очередь может активировать критик или же совершить определенные действия. Например, зажегся зеленый свет светофора, значит можно переходить дорогу.
Если активировалось много критиков, значит проблему нужно уточнить, так как степень неопределенности слишком высока. Если проблема очень похожа, то можно судить по аналогии.

4.2.2. Уровни мышления

Концепция уровней мышления представляет собой степень ментальной активности человека. Никто из людей не может похвастаться скоростью гепарда, гибкости кошки, силой медведя. Но наш вид все это компенсирует возможностью изобретения путей мышления. Например, чтобы быть быстрыми мы изобрели самолеты, машины. Чтобы быть сильными, мы изобрели оружие. Что же делает это возможным? Безусловно результатом всего является взаимодействие человека с окружающим миром. Именно данное взаимодействие заставляет людей изобретать что-то новое, создавать шедевры литературы и летать в космос. Но как же мы всего этого добиваемся начиная от, инстинктивного одергивая руки до создания Теории всего [11]. Далее мы рассмотрим концепцию уровней мышления.

1. Инстинктивный уровень
2. Уровень обученных реакций
3. Уровень рассуждений

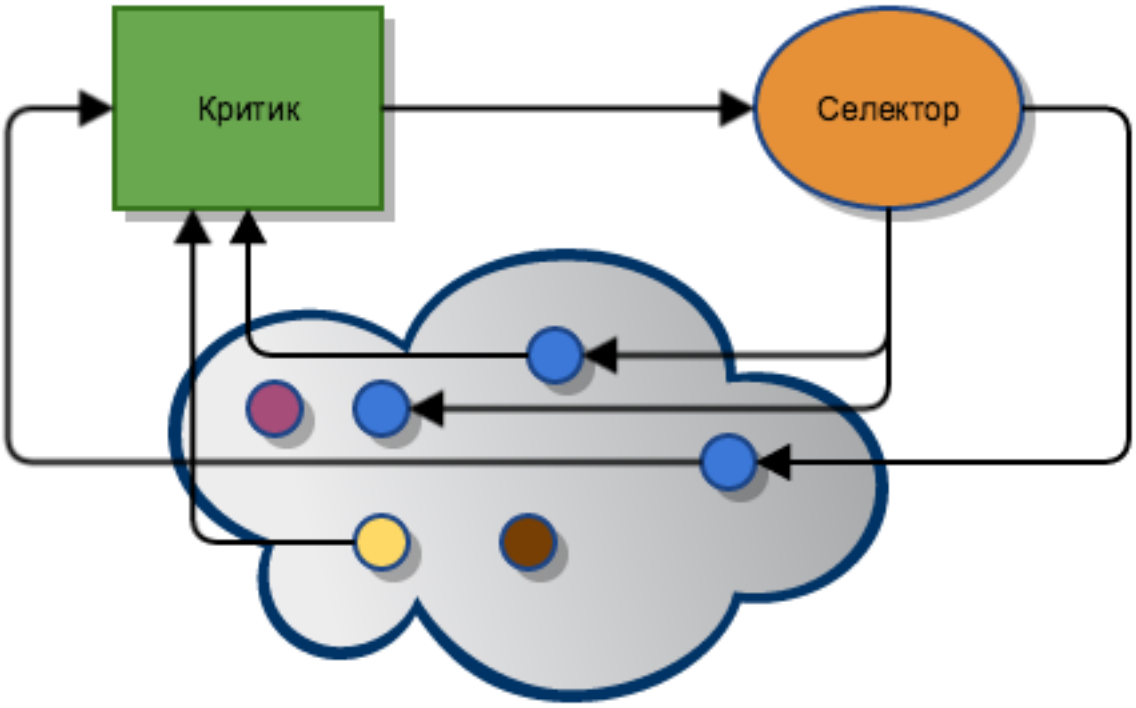

Рисунок 4.2: Критик-Селектор-Путь мышления в разрезе ресурсов

4. Рефлексивный уровень
5. Саморефлексивный уровень
6. Самосознательный уровень

Инстинктивный уровень. На данном уровне происходят инстинктивные реакции (врожденные). Например, боязнь обжечься. Не прыгать под машину. Общую формулу для этого уровня можно выразить как ”Если ..., то сделать так”.

Уровень обученных реакций. На данной уровне происходит мышление обученных реакций, то есть тех реакций, которыми человек обучается в течение жизни. Например, переходить дорогу на зеленых свет. Общую формулу для этого уровня можно выразить как ”Если ..., то сделать так”.

Уровень рассуждений. На данной уровне происходит мышление с использованием рассуждений. Если я сделаю так, то будет ... 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”.

Рефлексивный уровень. На данном уровне происходит рассуждение с учетом анализа прошлых событий. Например, прошлый раз я побежал на моргающий зеленый и чуть не попал под машину.

Саморефлексивный уровень. На данном уровне происходит оценка себя. Строится определен-

ная модель 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 **легкомысленный**.
Самосознательный уровень. Самозонательный уровень на данный момент характерен только для человека. На данном уровне идет оценка поступков человека с точки зрения высших идеалов и внешних оценок. Например, а что подумают мои друзья? А как бы поступил мой герой?
Деление на данные уровне носит условный характер. Например уровень 5 и 6 можно объединить. Но по словам Марвина Мински принцип бритвы Оккама успешно применяется в физики, но в психологии он не должен применяться также легко.
На рисунке 4.3 представлена схематичное изображение уровней мышления. 1-3 уровни составляют личность человека. 2-5 представляют ЭГО человека (Человеческое Я) - осознание человека в общении с окружающими. 3-6 представляют собой сверх ЭГО человека (сверх Я) - его моральные установки.

Рисунок 4.3: Уровни мышления

4.2.3. K-line

Концепция K-line была первый раз упомянута Марвином Мински в 1987 году в журнале Cognitive Science. В книге ”The Society of Mind” [12] Марвин Мински раскрывает концепцию K-line. Полностью концепция описана позже в книге ”The Emotion Machine” [3]. K-line представляет собой связь между двумя событиями, объединяющими их в знание. Например, объединение Пути мышления, найденного решения и активированной проблемы. Данная линия объединяет то как мы думали, решение. На Рисунке **??** показана K-line, которая объединяет пути мышления, решение и

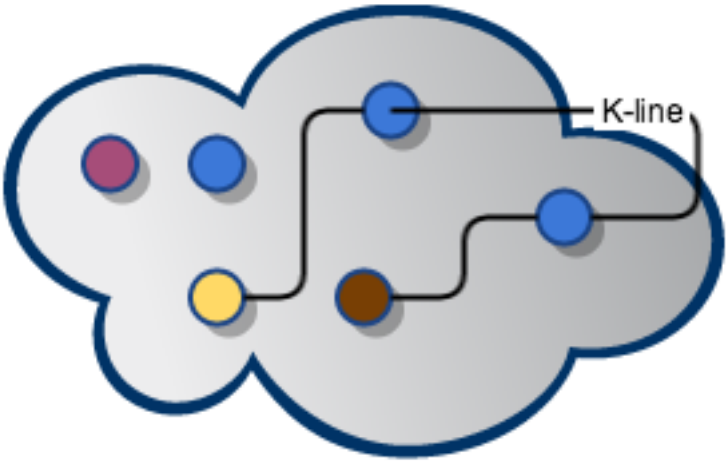

Рисунок 4.4: K-line

другие Критики. Данная концепция позволяет ”запоминать” удачные решения.

4.3. Выводы по главе

Для программной экспертной системы очень важно иметь способность мышления и рассуждения. Например, очень важно для системы уметь действовать по аналогии. Так как множество запросов типичны и отличаются частностями. Например, пожалуйста, установить Office, Antivirus и т.д.

Также для экспертной системы важно уметь абстрагировать специализированные рецепты решения. Например, система научилась решать инцидент ”Please install Firefox”. Абстрагировав данный инцидент до степени ”Please install browser” система сможет теми же способами попробовать решить новый инцидент.

После рассмотрения нескольких моделей была выбрана модель мышления Марвина Мински, так как данная модель наиболее точно ложится на целевую область решения инцидентов в области IT.

Глава 5

Результаты работы

5.1. Архитектура системы

Архитектура системы представляет собой модульную систему. Основными компонентами системы являются:

1. TU webservice
2. CoreService
3. DataService
4. Reasoner
5. ClientAgent
6. MessageBus

Система может работать в 2-х режимах: режим обучения и режим запроса. Вариант использования для режима обучения представлен на Рисунке 5.1. Главными действующими лицами является специалист технической поддержки (TSS), в общем случае это базовый класс Пользователь (User). Данный вариант использования имеет несколько ветвей:

– communication:Train - обучение посредством коммуникации с системой специалиста технической поддержки.

– communication:ProvidesSolution - в случае коммуникации в режиме обучения специалист технической поддержки должен предоставить не только сам запрос, который будет формализован системой, но также решение данного запроса. Система формализует запрос, формализует решение и создаст между ними связи

– communication:ProvideRequest - специалист технической поддержки вводит в систему запрос

– communication:MonitorsSolution - специалист технической поддержки смотрит как применяется решение, если находится проблема, то решение корректируется в CorrectSystemSolutions

Второй вариант использования это основной кейс. Главными действующими лицами системы является заказчик (Customer), в общем случае это базовый класс Пользователь (User). Он также имеет несколько ветвей:

– ProvideRequest - заказчик вводит запрос в систему. Это может быть либо команда ProvideDirectInstruction, либо описание проблемы ProvideProblemDescription.

– communication:ProvideClarificationResponse - в случае, если система не может формализовать запрос, либо нашлось множество решений, то система запрашивает пользователя детали

– communication:ProvideConfirmationResponse - в случае, когда система нашла решение, она запрашивает пользователя подтверждение о том, что искомое решение решило его проблему

5.1.1. Компоненты системы

На Рисунке 5.2 представлена диаграмма компонентов системы. Взаимодействие компонентов системы показано на рисунке 5.3. Пользователь взаимодействует с системой посредством компонента WebService 5.1.2. Взаимодействие происходит по следующем схеме:

1. WebService получает запрос пользователя. Сохраняет запрос в Базу Знаний (Базу данных) 1.
2. WebService отправляет сообщение типа Request с информацией о запросе в компонент MessageBus (шина).
3. Один из экземпляров CoreService компонента обрабатывает запрос.
4. Компонент CoreService обрабатывает запрос и сохраняет результаты в Базу Знаний, затем он отправляет в MessageBus сообщение RequestCompleted и сообщение ActionsToExecute с действиями, которые необходимо исполнить
5. WebService получает сообщение RequestCompleted с результатами выполнения запроса и уведомляет подписчиков (конечных пользователей)
6. Компонент ClientAgent получает сообщение ActionsToExecute со списком действий, которые необходимо исполнить на целевых машинах

5.1.2. Компонент WebService

Данный компонент обрабатывает запросы пользователей. Запрос пользователя представляется объектом Request, который содержит информацию о пользователя, а также ссылку на метод, который будет вызван, когда запрос будет обработан. Вся работа происходит в компоненте CoreService. На Рисунке 5.4 представлен интерфейс компонента.

Таблица 5.1: Описание методов

Метод	Описание
createRequest(request:Request):[RefObject]	Создает запрос от пользователя. В качестве параметра в метод передается SubscriptionID, по которому идет проверка запроса.
subscribe(user:User,subscription:Subscription)	Создает подписку для пользователя.
unsubscribe(user:RefObject,subscription:RefObject)	Убирает подписку пользователя.
updateSubscription(user:RefObject,subscription:Subscription)	Обновляет подписку пользователя.
getSubscription(subscriptionID:RefObject):List<Request>	Возвращает подписку.
findRequests(user:RefObject)	Возвращает запросы пользователя.
createUser(user:User):RefObject	Создает пользователя.
updateUser(user:User)	Обновляет информацию о пользователе.
removeUser(user:RefObject)	Удаляет информацию о пользователе.
findRequest(request:RefObject):Request	Возвращает запрос по ссылке.

Подробное описание классов представлено в А. Основной алгоритм работы компонента:

1. Пользователь создает запрос, используя метод WebService.createRequest
2. Система сохраняет запрос в Базу Знаний и начинает его обработку
3. Когда изменяется статус запрос request.state система оповещает подписчиков на этот запрос

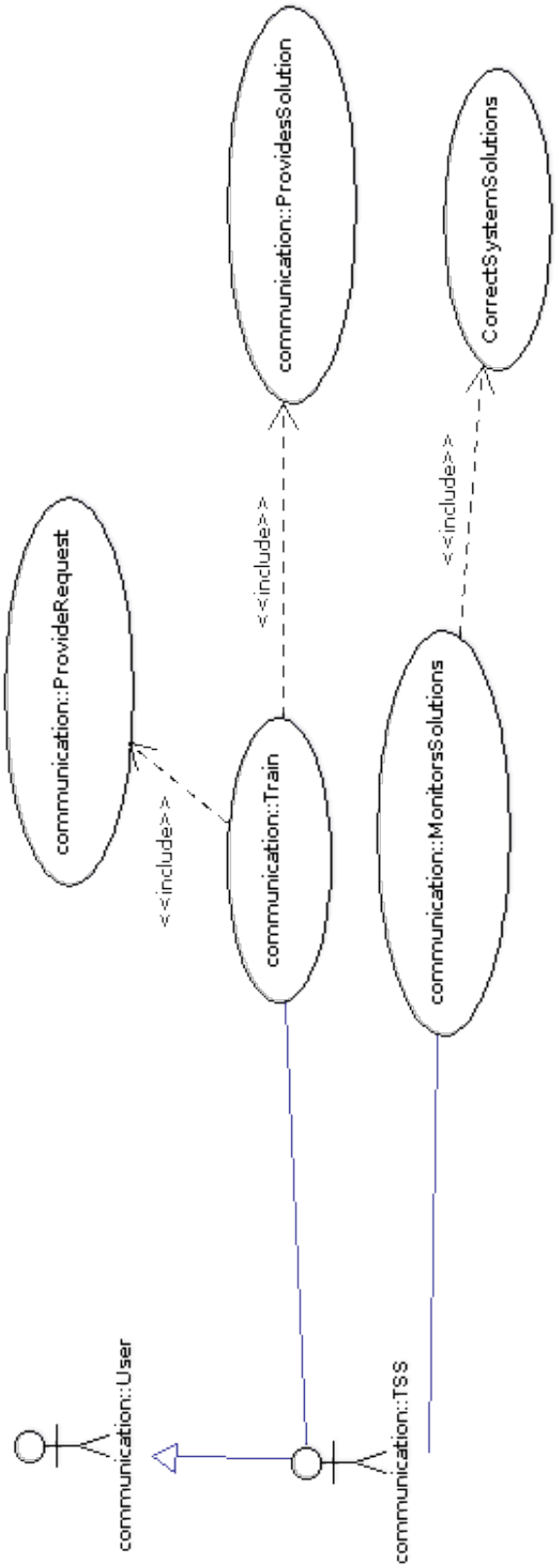

Рисунок 5.1: Вариант использования. Обучение.

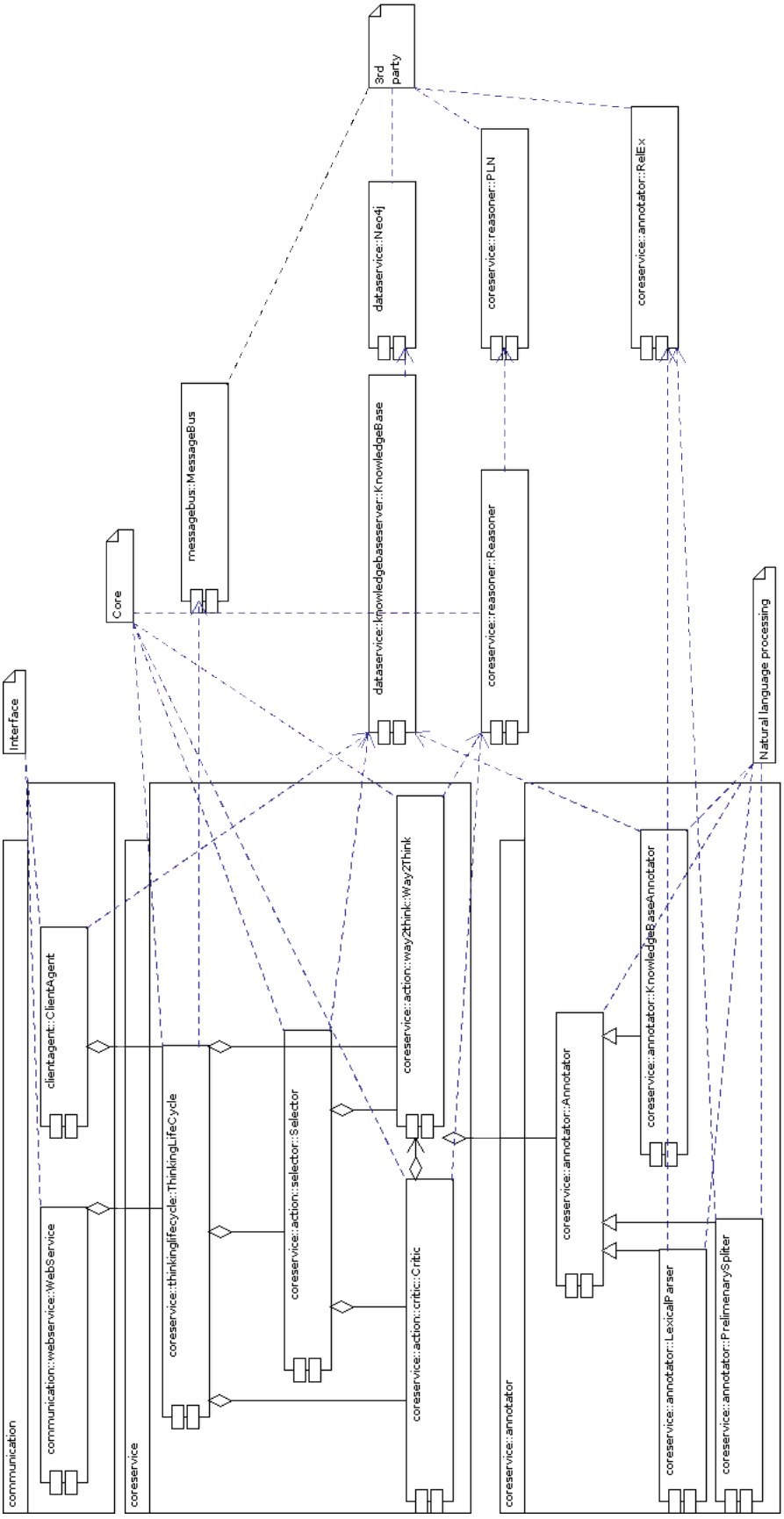

Рисунок 5.2: Диграмма компонентов

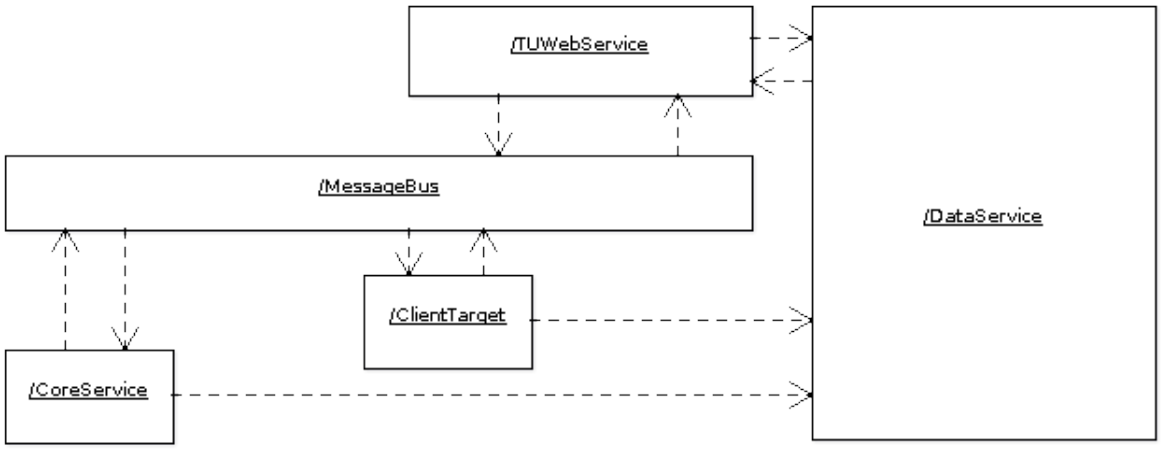

Рисунок 5.3: Диграмма взаимодействия компонентов

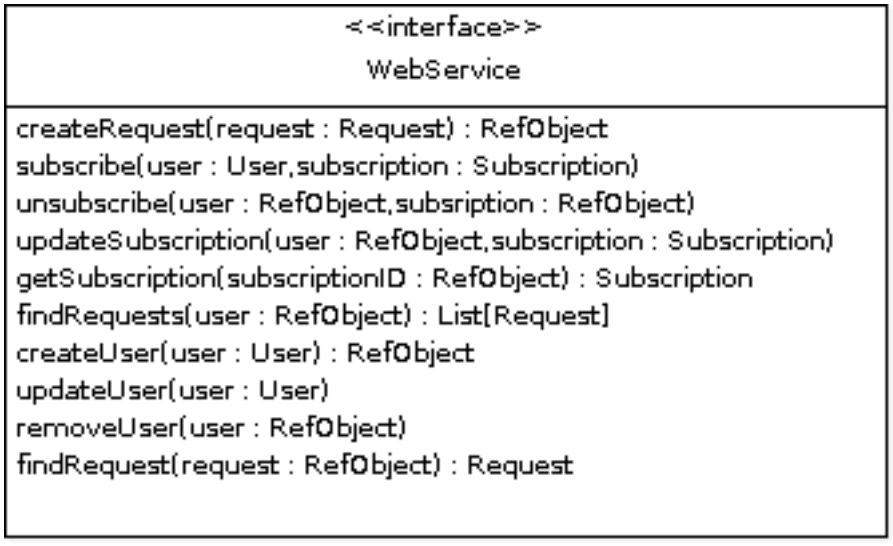

Рисунок 5.4: Интерфейс компонента WebService

5.1.3. Компонент CoreService.ThinkingLifeCycle

Это основной компонент системы, ответственный непосредственно за выполнение запросов. Данный компонент управляет потоками, событиями приложения. Он запускает исполнение Критиков, Селекторов, Путей мышления, осуществляет обмен данных между компонентами. Компонент построен на фреймворке Akka Concurrency, который позволяет разрабатывать приложения, которые могут работать параллельно [13].
В данном компоненте реализовано шесть уровней мышления.

1. Instinctive - Инстинктивный уровень
2. Learned - Уровень обученных реакций
3. Deliberative - Уровень рассуждений
4. Reflective - Рефлексивный уровень
5. Self-Reflective Thinking - Саморефлексивный уровень
6. Self-Conscious Reflection - Самосознательный уровень

На уровне Instinctive идет обработка сгенерированных по шаблону инцидентов. Объект, который используется для обработки использует паттерн Akka [13]. На рисунке 5.5 представлена диграмма классов компонента.

Описание методов класса

onMessage(message : Message)
Данный метод вызывается при получении сообщения от шины. После этого происходит обработка запроса, вычисляется список действий, которые нужно выполнить. После этого запускается исполнение этих действий. На рисунке 5.6 представлена диаграмма действий для этого метода.

sendMessage(publisher: Publisher, message: Message): Boolean
Данный метод используется для создания и отправки сообщения в шину. На рисунке 5.7 представлена диаграмма действий для этого метода.

apply(request : Request) : List[Action]
Данный метод используется для запуска обработки входящего запроса. Для запроса создается контекст, если такой уже не был создан. После этого вызывается следующий компонент системы Selector, который выбирает необходимые ресурсы из базы. На рисунке 5.8 представлена диаграмма действий для этого метода. Важно отметить, что метод apply является специальном методом в контексте функционального языка программирования Scala. По умолчанию при применение класса к параметрам исполняется функция apply [14].

apply(actions : List[Action]) : TransFrame
Данный метод запускает обработку действий. Все действия разделяются на Critic (триггеры действий, которые в итоге должны перейти в WayToThink через Selector) и WayToThink (пути мыш-

ления, непосредственно обработчики данных, классы, которые производят изменения данных) На рисунке 5.9 представлена диаграмма действий для этого метода.

processWay2Think(inputContext: Context, outputContext: Context): TransFrame

Данный метод запускает обработку WayToThink 1. Данный метод создает входной контекст (InputContext), заполняет его параметрами, создает выходной контекст OutputContext. Затем он запускает обработку данных во входном контексте. На рисунке 5.10 представлена диаграмма действий для этого метода.

processCritic(context: Context):List[SelectorRequestRulePair]

Данный метод запускает обработку Critic 1. На рисунке 5.11 представлена диаграмма действий для этого метода.

init(): Boolean

Данный метод инициализирует экземпляр класса ThinkingLifeCycle. Во время инициализации происходит Базы Знаний 1, подключения к Шине данных. На рисунке 5.12 представлена диаграмма действий для этого метода.

start(): Boolean

Данный метод является оберткой для поддержки Akka Concurrency. Он вызывает метод init.

stop(): Boolean

Данный метод является оберткой для поддержки Akka Concurrency. Он останавливает работу экземпляра класса: останавливается сессия к шине данных, останавливается подключение к Базе Знаний.

registerProcess(process : Process,level : Level) : Process

Данный метод регистрирует процесс в пуле. В качестве параметра принимается Level (уровень приоритета процесса).

stop(processLevel : Level) : List[Process]

Данный метод регистрирует останавливает процесс. В качестве параметра принимается ссылка на процесс. На рисунке 5.13 представлена диаграмма действий для этого метода.

Описание работы компонента

Запуск и остановка

1. Когда приложение стартует оно инициализирует ThinkingLifeCycle, который активирует набор критиков, базируясь на текущей цели системы. Например, цель-классифицировать инцидент, активируется набор критиков: разобрать, проверить, найти категорию.
2. Когда приложение останавливается - оно останавливает все объекты класса и подклассов Actions (Critics, WayToThink), Selectors и ThinkingLifeCycle.

Коммуникация происходит посредством сообщений, отправленных через MessageBus (Шину Данных) 1 JMS [15]. *Взаимодействие компонента с другими компонентами*

1. Критик возвращает список Селекторов (SelectorRequestRule)
   (a) ThinkingLifeCycle запускает обработку компонента Selector
   (b) Selector возвращает список Action B из базы знаний
   (c) ThinkingLifecycle параллельно запускает возвращенные Action
      i. Если Action это Critic
      ii. ThinkingLifeCycle создает InputContext (входной контекст приложения) и копирует туда все данные из Context (контекста) инцидента
      iii. Если Action это Critic с ссылками ReturnToSameSelector ThinkingLifeCycle ждет результаты и отправляет список SelectorRequestRule, возвращенные Critic новому Selector. Иными словами Critic может вернуть новый Selector. В данном случае нам нужно провести операцию Join для всех потоков [16]. В иных же случаях все Action запускаются в параллельных потоках.
      i. Если Action это WayToThink
      ii. ThinkingLifeCycle создает InputContext (входной контекст приложения) и копирует туда все данные из Context (контекста) возвращенный Selector
      iii. TLC 1 запускает WayToThink
      iv. TLC сохраняет параметры в OutputContext
      v. TLC сохраняет итоговый результат работы и возвращает его

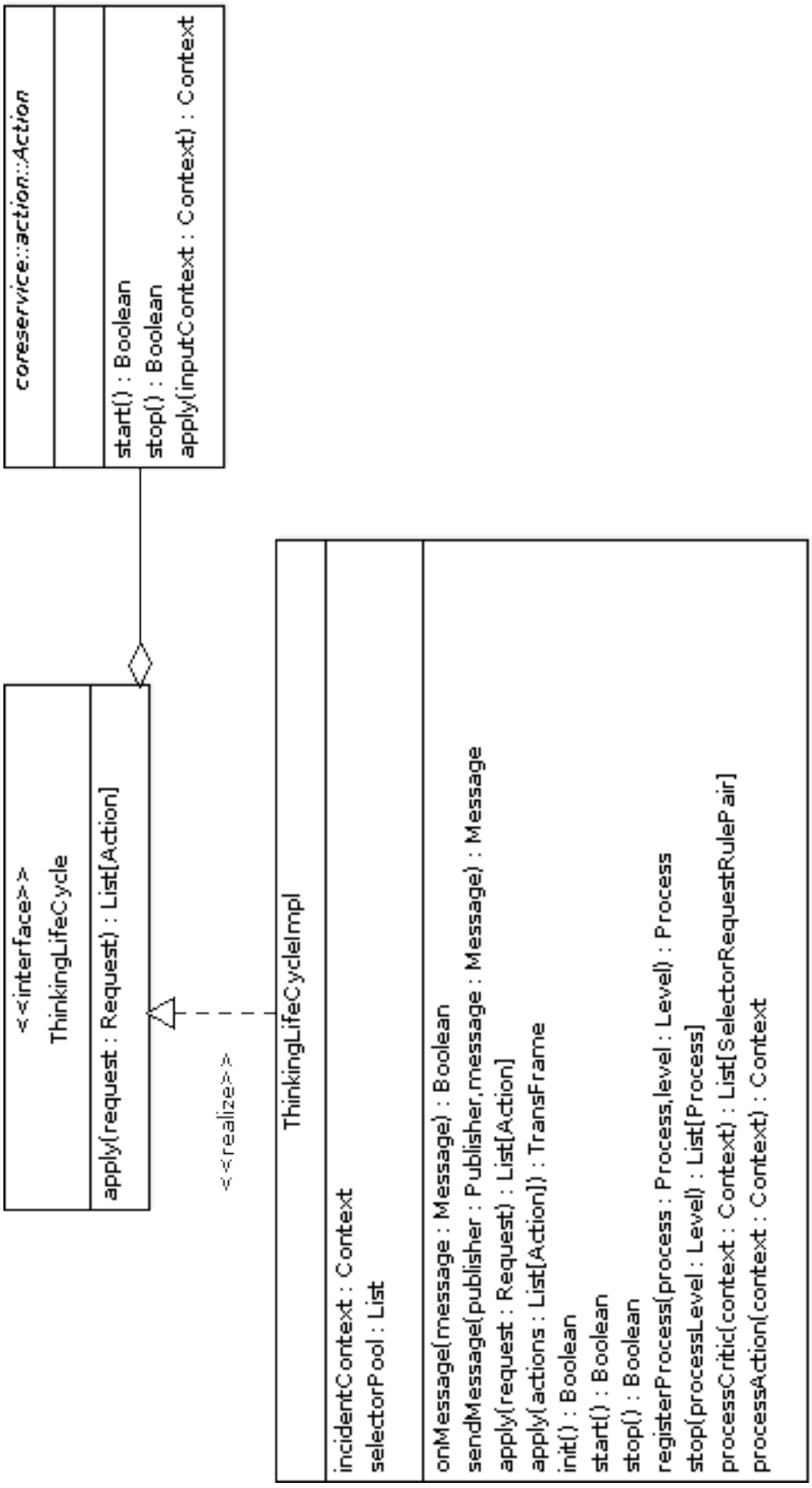

Рисунок 5.5: Диаграмма классов ThinkingLifeCycle

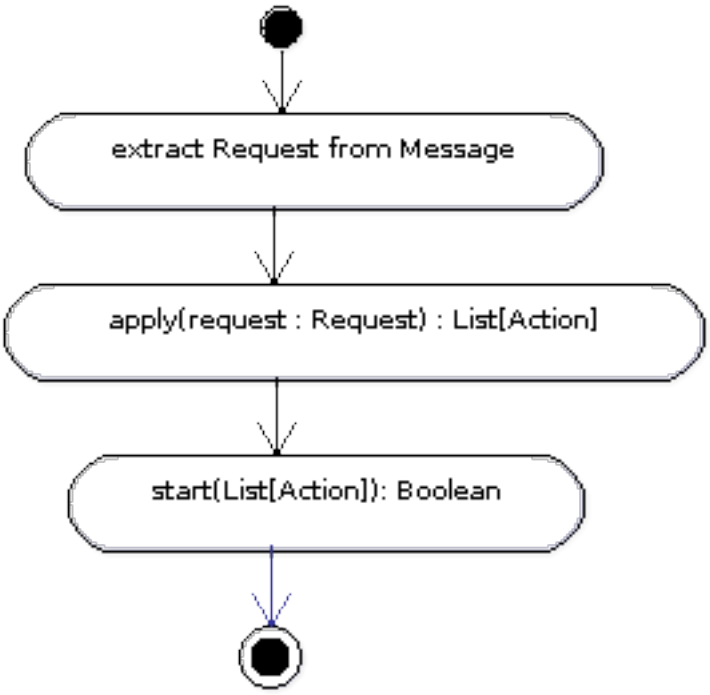

Рисунок 5.6: Диаграмма действий nnMessage

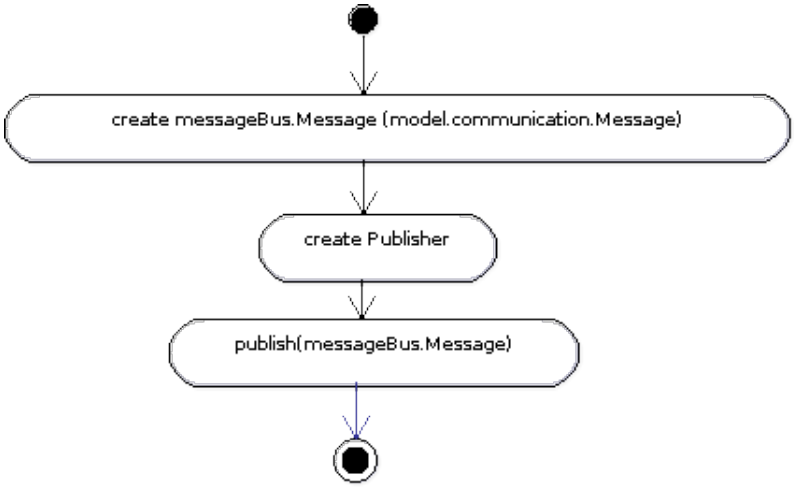

Рисунок 5.7: Диаграмма действий sendMessage

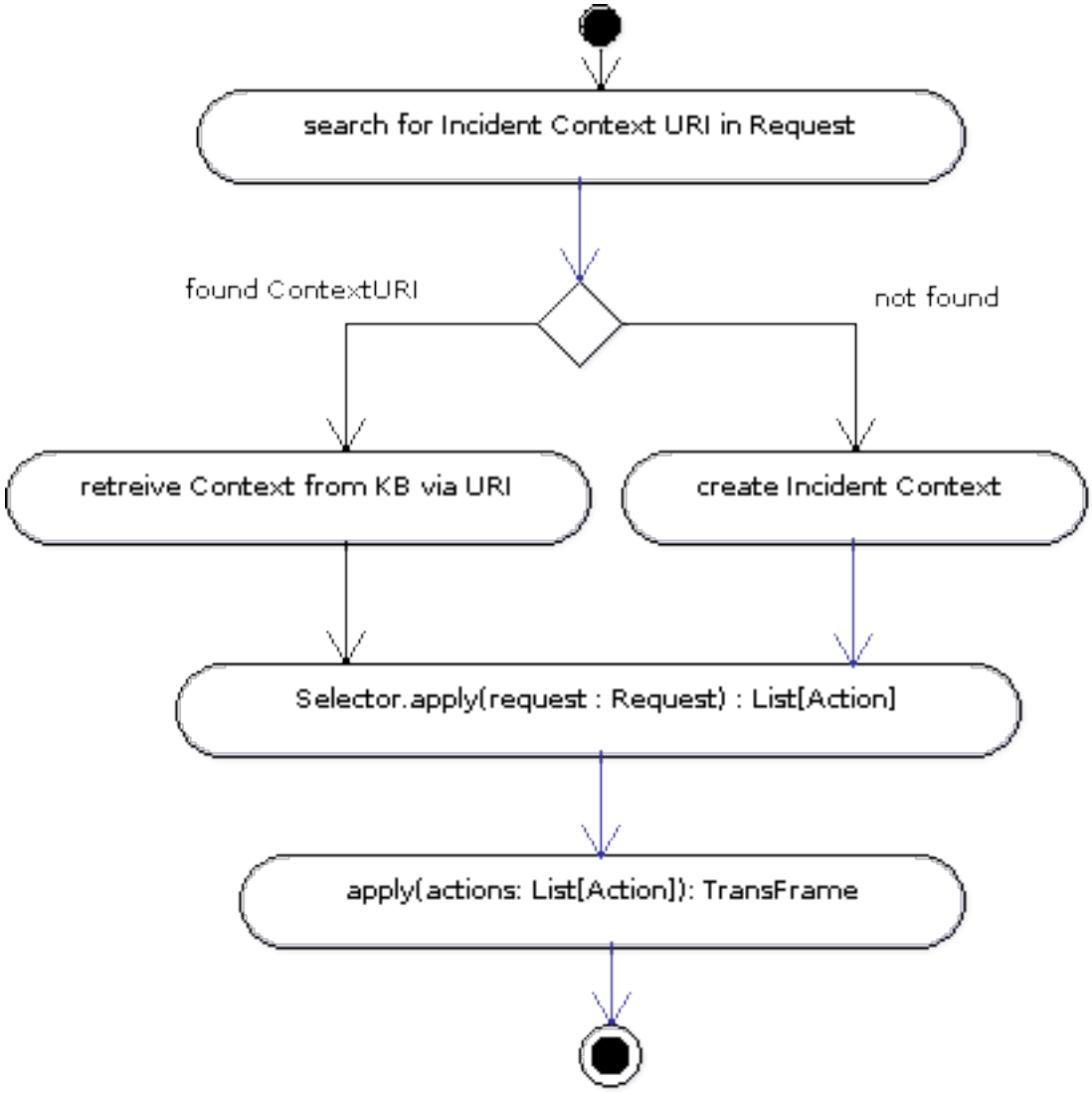

Рисунок 5.8: Диаграмма действий apply

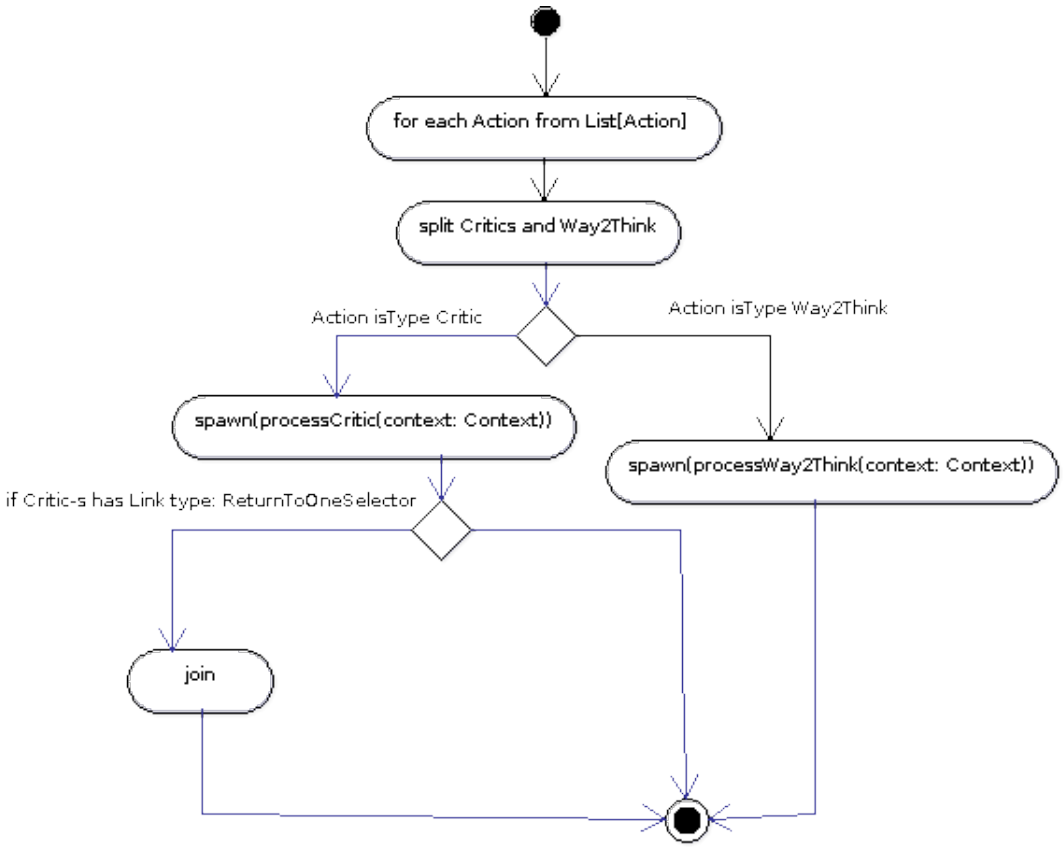

Рисунок 5.9: Диаграмма действий apply

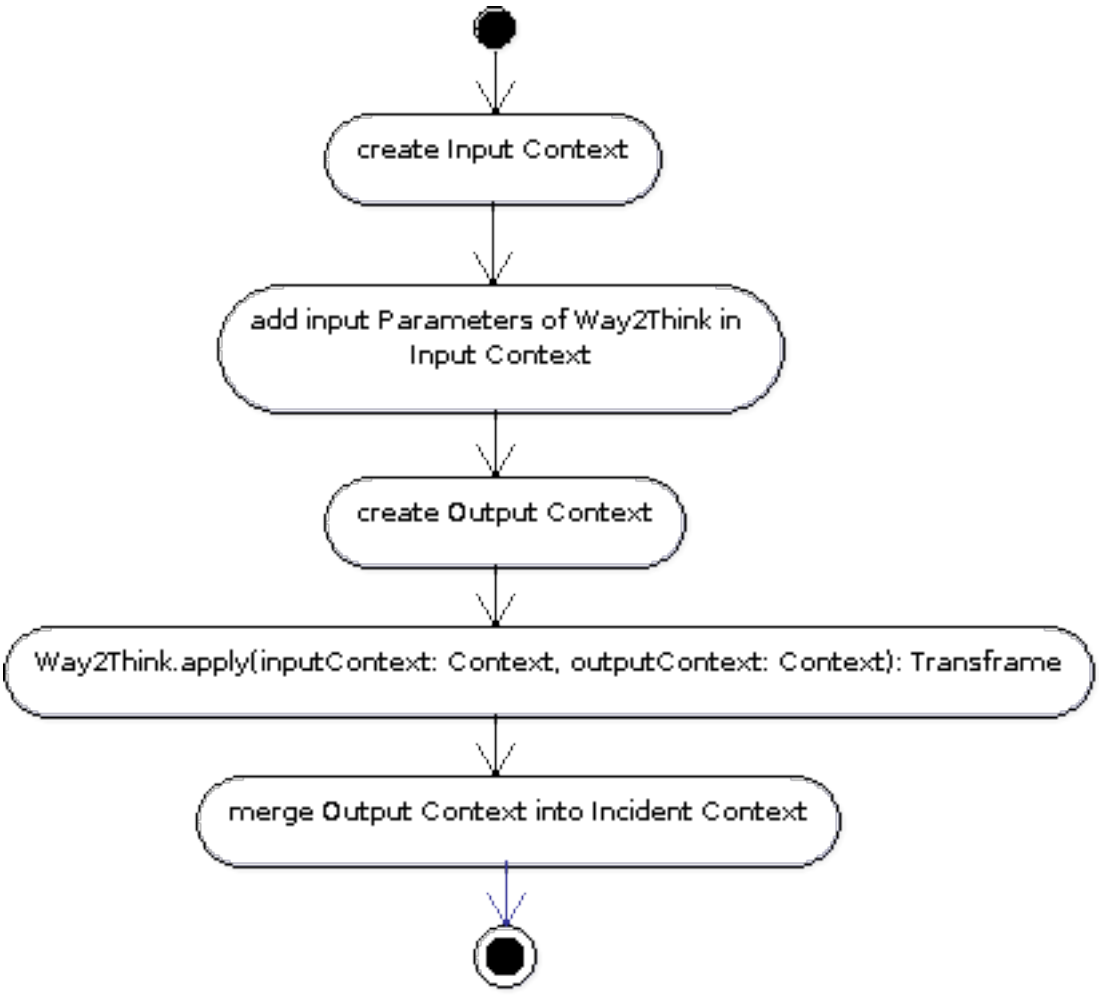

Рисунок 5.10: Диаграмма действий processWay2Think

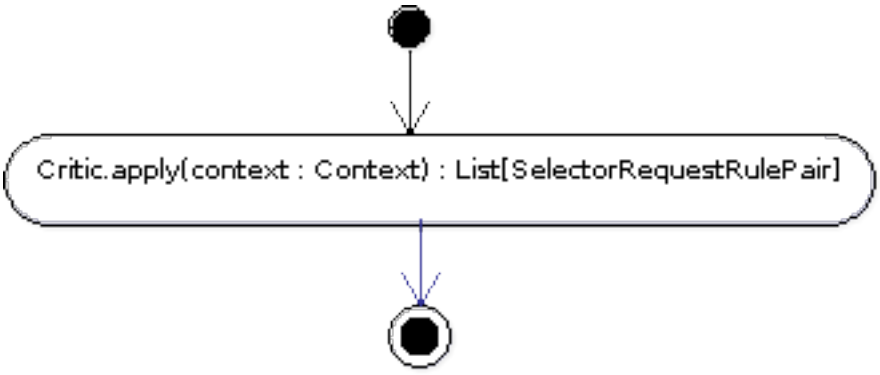

Рисунок 5.11: Диаграмма действий processCritic

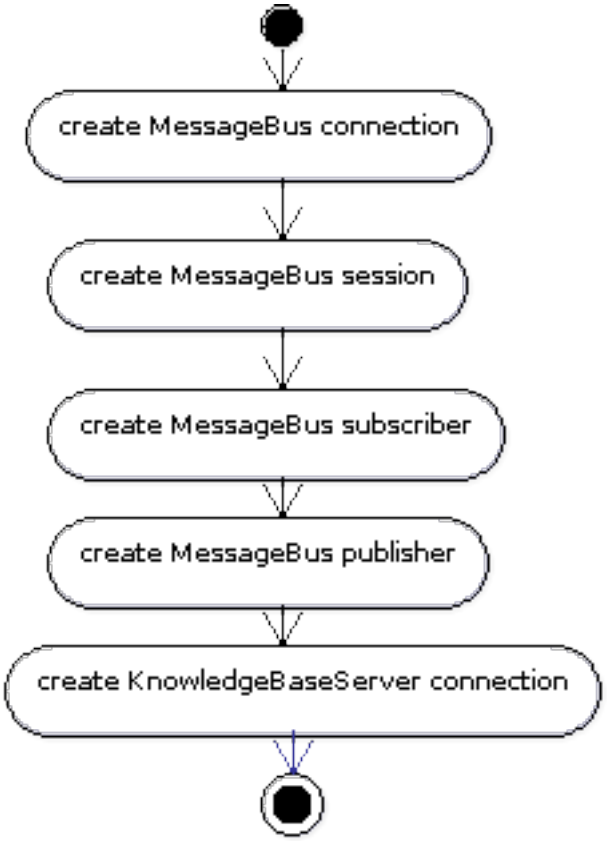

Рисунок 5.12: Диаграмма действий init

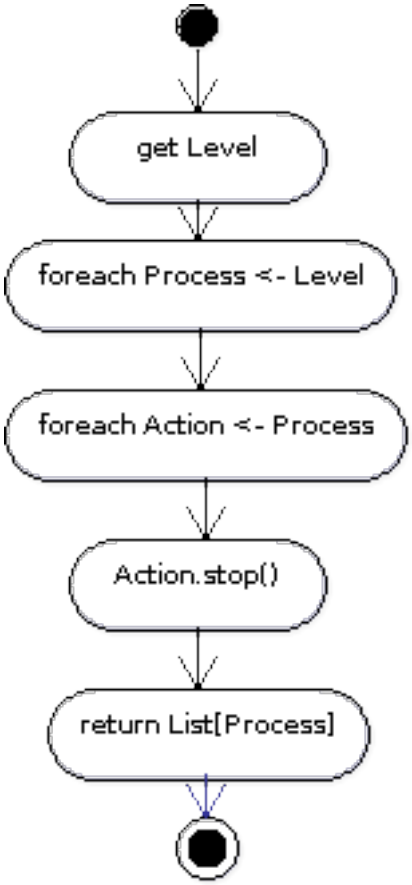

Рисунок 5.13: Диаграмма действий stop

5.1.4. Компонент CoreService.Selector

Selector это компонент, который ответственен за получение списка действий из Базы знаний, согласно входным параметрам. **Входной критерий**. TLC запускает Selector с параметрами. **Выходной критерий**. Selector получает список Action: WayToThink или Critic.

Описание методов класса

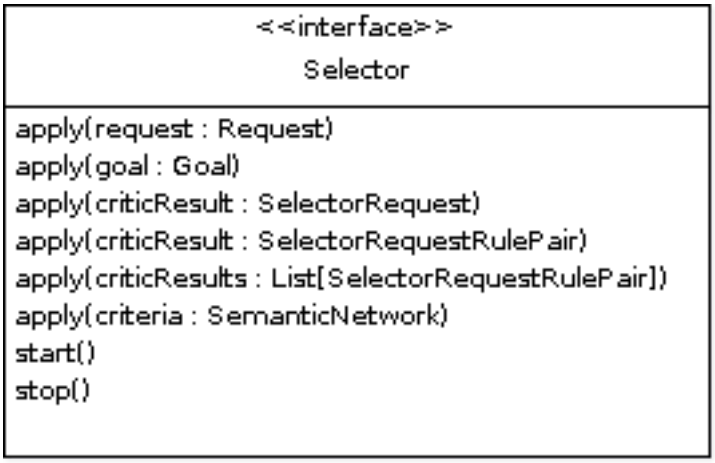

Рисунок 5.14: Интерфейс компонента Selector

На Рисунке 5.14 показан интерфейс компонента.

apply(request : Request) : Action

Данный метод на основе запроса пользователя получает из Базы знаний необходимые Critic 5.1.5. На рисунке 5.15 представлена диаграмма действий для этого метода.

apply(goal: Goal): Action

Данный метод на основе цели системы получает из Базы знаний необходимые Critic 5.1.5. На рисунке 5.16 представлена диаграмма действий для этого метода.

apply(criticResult : ActionProbabilityRule) : Action

Данный метод на основе работы Critic получает из Базы знаний необходимые Action **??**. На рисунке 5.17 представлена диаграмма действий для этого метода.

Описание работы компонента

Действия при классификации инцидента

1. TLC 5.1.3 запускает входящие Critic 5.1.5 параллельно
2. Когда Critic возвращает результат работы в виде ActionProbabilityRuleTriple, TLC запускает Selector с этим параметром

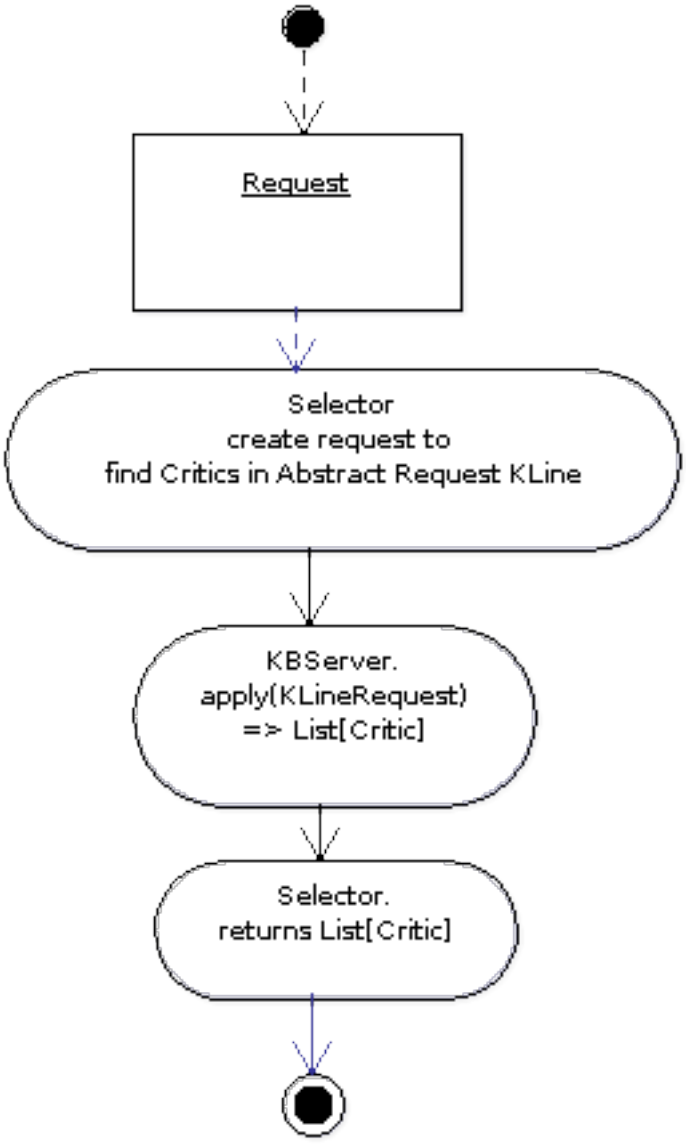

Рисунок 5.15: Диаграмма действий Selector.apply

3. Selector запускает GetMostProbableWay2Think, который возвращает наиболее вероятный WayToThink

4. В некоторых случаях Selector может вернуть менее вероятный вариант, если на Refelective уровне мышления сработал Critic, который посчитал, что данное решение некорректно или же пользователь признал его таким

На Рисунке 5.18 представлена диаграмма действий выбора наиболее вероятного WayToThink. На Рисунке 5.19 представлена диаграмма действий классификации инцидента. TLC 5.1.3 получает цель Классифицировать инцидент, затем Selector по этой цели возвращает необходимые Critic. Затем TLC запускает обработку Critic в разных потоках (параллельно). В данном случае рассматривает 3 Critic.

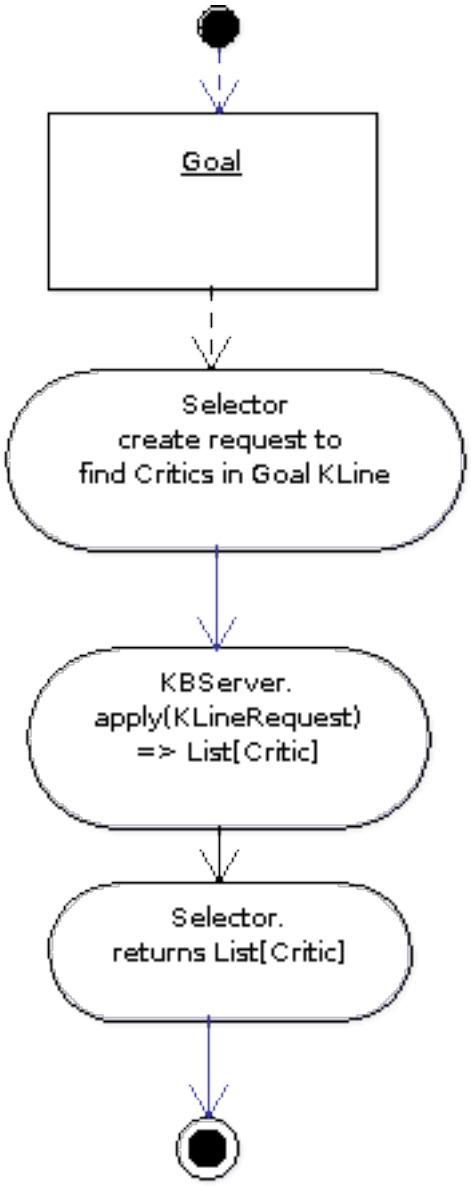

Рисунок 5.16: Диаграмма действий Selector.apply

– DirectInstruction - прямые инструкции, данный Critic возвращает WayToThink Simulate 5.1.6, который ищет связь между концепциями в запросе и концепциями в Базе Знаний.

– ProblemWithDesiredState - проблема с ожидаемым результатом, данный Critic возвращает Simulate+Reformulate WayToThink, которые ищут сопоставление концепциями в Базе Знаний и пытается преобразовать запрос к DirectInstruction запросу (прямым инструкциям).

– ProblemWithoutDesiredState - проблема без ожидаемого результата. Данный Critic возвращает Simulate+Reformulate+InferDesiredState, который пытается преобразовать проблему к ProblemWithDesiredState.

Затем TLC собирает результаты выполнения всех Critic и запускает их по новой, пока не будет достигнута изначальная цель.

5.1.5. Компонент CoreService.Critics

Critic является основным компонентом для анализа в триплете Critic->Selector->WayToTHink. Critic используется для классификации входной информации, рефлексии, само-анализа и т.д. и служит своеобразным триггером.

Входной критерий

TLC 5.1.3 запускает Critic согласно Goal C (Цель) или Request.

Выходной критерий

Critic генерирует SelectorRequest 5.1.4. На входе Critic принимает

- Загруженные из базы правила для работы Critic (CriticRules)
- DomainModel:SemanticNetwork - доменная модель, представляющая собой семантическую сеть
- Описание инцидента, представляющая собой семантическую сеть

На выходе Critic предоставляет следующую информацию:

- SelectorRequest 5.1.4 - запрос на выбор Selector из базы знаний
- CriticRules - правило, которое сработало для активации. Данное правило является логическим предикатом

На Рисунку 5.20 представлена диаграмма работы Critic. Основные типы Critic

1. Manager - простой тип критика, который работает как триггер Goal C, чтобы запустить необходимый WayToThink.
2. Control - контролирующий Critic, который ждет определенного события (срабатывает на определенное событие). Например, заканчивается, отведенное на решение время.
3. Analyser - анализатор, обрабатывает и выявляет тип инцидента. Например, прямые инструкции, проблема с желаемым состоянием, наиболее вероятное действие

Основные критики

1. Уровень обученных реакций
   (a) PreprocessManager - предобработка информации
   (b) Классификаторы инцидентов: Прямые инструкции, Проблема с желаемым состоянием, Проблема без желаемого состояния

(c) SolutionCompletenessManager - связывается с пользователем и проверяет устраивает ли его найденное решение

2. Уровень рассуждений

(a) Выбор наиболее вероятного Selector по Rule. Данный Critic после проверки правил, выбирает из них правило с большей вероятностью

3. Рефлексивный уровень

(a) Менеджер целей. Установка целей

4. Саморефлексивный уровень

(a) ProcessingManager - запускает выполнение запроса

(b) TimeControl - контроль времени исполнения запроса

(c) DoNotUnderstandManager - активируется, когда необходимо уточнение пользователя для продолжения работы

5. Самосознательный уровень

(a) EmotionalStateManager - контроль общего состояния системы

5.1.6. Компонент CoreService.WayToThink

WayToThink является основным операционным компонентом триплета Critic->Selector->WayToThink. Основными задачами данного компонента являются: обновление, преобразование, сохранение данных и коммуникация с пользователем.

Входной критерий

Запуска из компонента ThinkingLufeCycle 5.1.3. Входными данными является InputContext, который содержит параметры WayToThink.

Выходной критерий

WayToThink завершил работу. На выходе возвращается измененные данные в ходе работе.

На Рисунке 5.21 представлен интерфейс компонента.

В общем виде компонент описывает последовательность действий. В системе используется два больших класса WaytToThink простой и составной (сложный). Простые WayToThink являются встроенными в систему, остальные являются комбинацией компонентов: Critic 5.1.5, Selector 5.1.4, WayToThink 5.1.6.

Встроенные типы:

1. Создать контекст

2. Установить общий статус системы

3. Установить цель системы
4. Разделить фразу на слова и предложения
5. Найти связи между входной информацией и базой знаний
6. Извлечь связи
7. Установить контакт с пользователями
8. Сохранить наиболее вероятное решение
9. Перефразировать (Reformulate)
10. Смоделировать (Simulate)
11. Найти решение
12. Остановить работу

WayToThink также используется как Решение (HowTo) D , то есть описывает последовательность действий, необходимых для решения проблемы.

5.1.7. Компонент CoreService.PreliminaryAnnotator

Данный компонент проводит предварительную подготовку текста: грамматическую и орфографическую коррекцию текста, а также разделение на предложения. На Рисунке 5.23 представлен интерфейс компонента.

5.1.8. Компонент CoreService.KnowledgeBaseAnnotator

Данный компонент устанавливает связи между терминами во входной фразе и базой знаний.
Входными критериями является список фраз.
Выходными критериями является список ссылок на внутренние термины.
Поток действий:

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 внутренний 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 аналогия в база знаний program->computer software.

5.1.9. Компонент DataService

Данный компонент отвечает за хранение данных в системе. База знаний построена на графах. На Рисунке 5.24 представлен интерфейс компонента. В базе знаний используется два типа объектов Object - объект базы знаний, BusiessObject - объект для Web Service (User, Request). BusinessObject является кортежем для интеграции с внешними системами. У объекта есть ID, который уникально удостоверяет его в рамках системы.

Список методов:

1. save(obj:Resource):Resource - сохранить ресурс в базу знаний
2. remove(obj:Resource) - удалить объект
3. select(obj:Resource):Resource - выбрать объект
4. link(obj:List<Resource>,linkName:String) - сделать ссылку между 2-мя объектами
5. selectLinkedObject(obj:Resource,linkName:String):Link<Resource> - выбрать все объекты, которые имеют связь под названием linkName с объектом obj.
6. addLinkedObject(parent:Resource,toLink:Resource,linkName:String) - создать ссылку linkName с объектом
7. saveRequest(obj:Request) - сохранить запрос в базу
8. selectRequest(obj:RefObject) - получить запрос из Базы Знаний
9. saveBusinessObject(obj:RefObject):RefObject - сохранить объект в базу
10. selectBusinessObject(obj:RefObject):RefObject - полуить объект из базы знаний

5.1.10. Компонент ClientAgent

Данный компонент предназначен для выполнения решений на конечной машине. Данный компонент должен поддерживать обработку D. В случае проблем компонент также должен обращаться за помощью к специалисту.

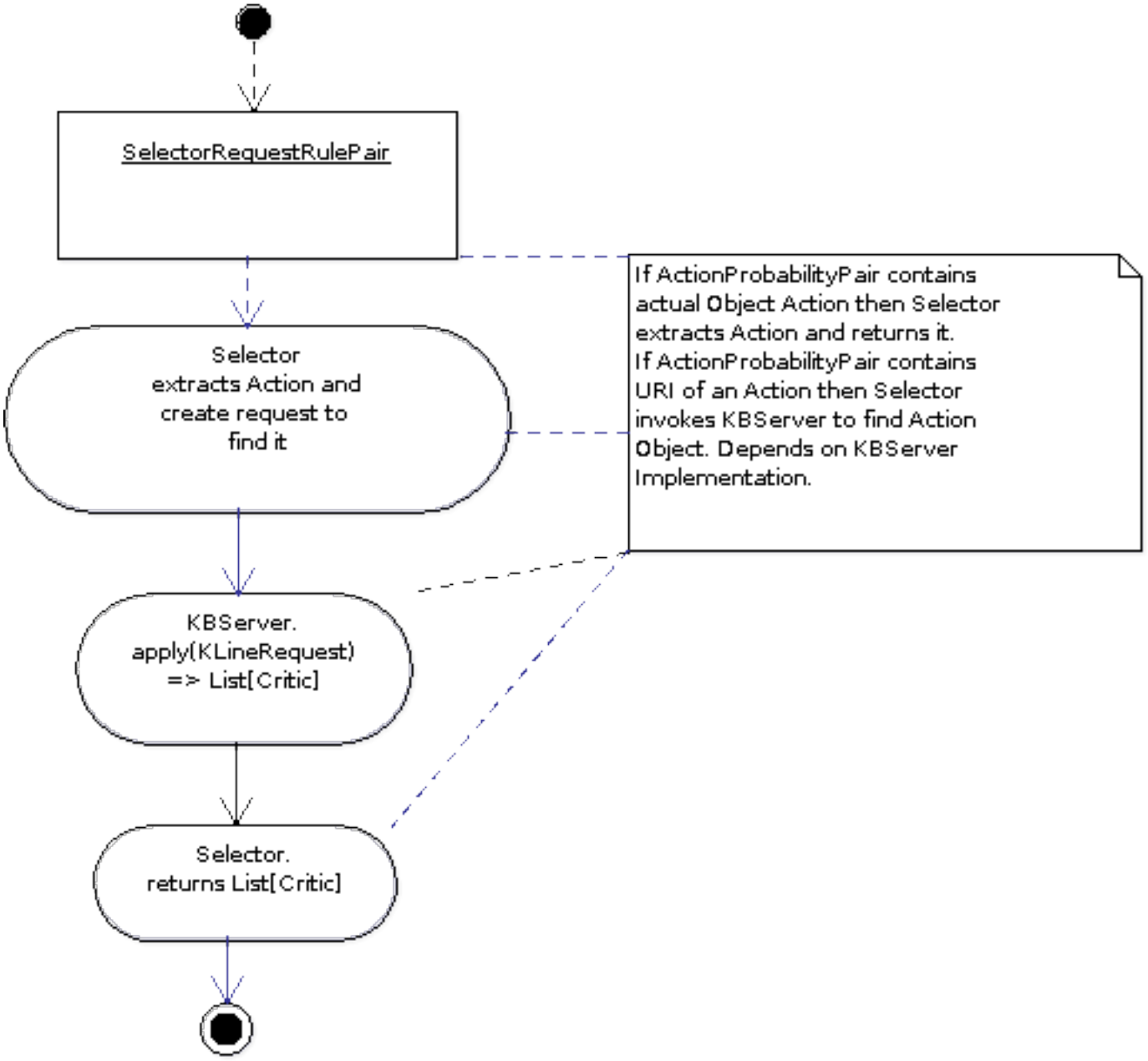

Рисунок 5.17: Диаграмма действий Selector.apply

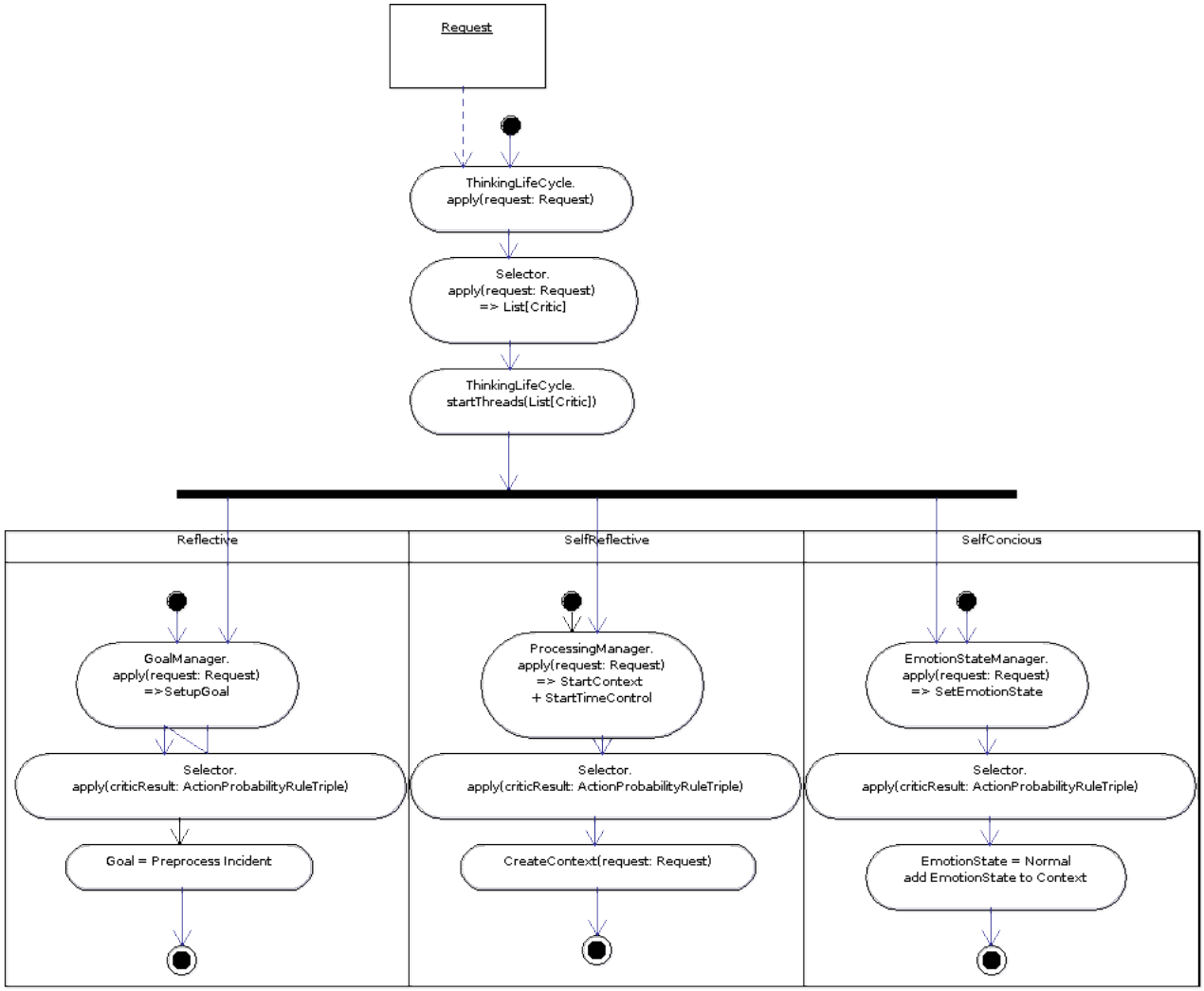

Рисунок 5.18: Диаграмма действий выбора WayToThink

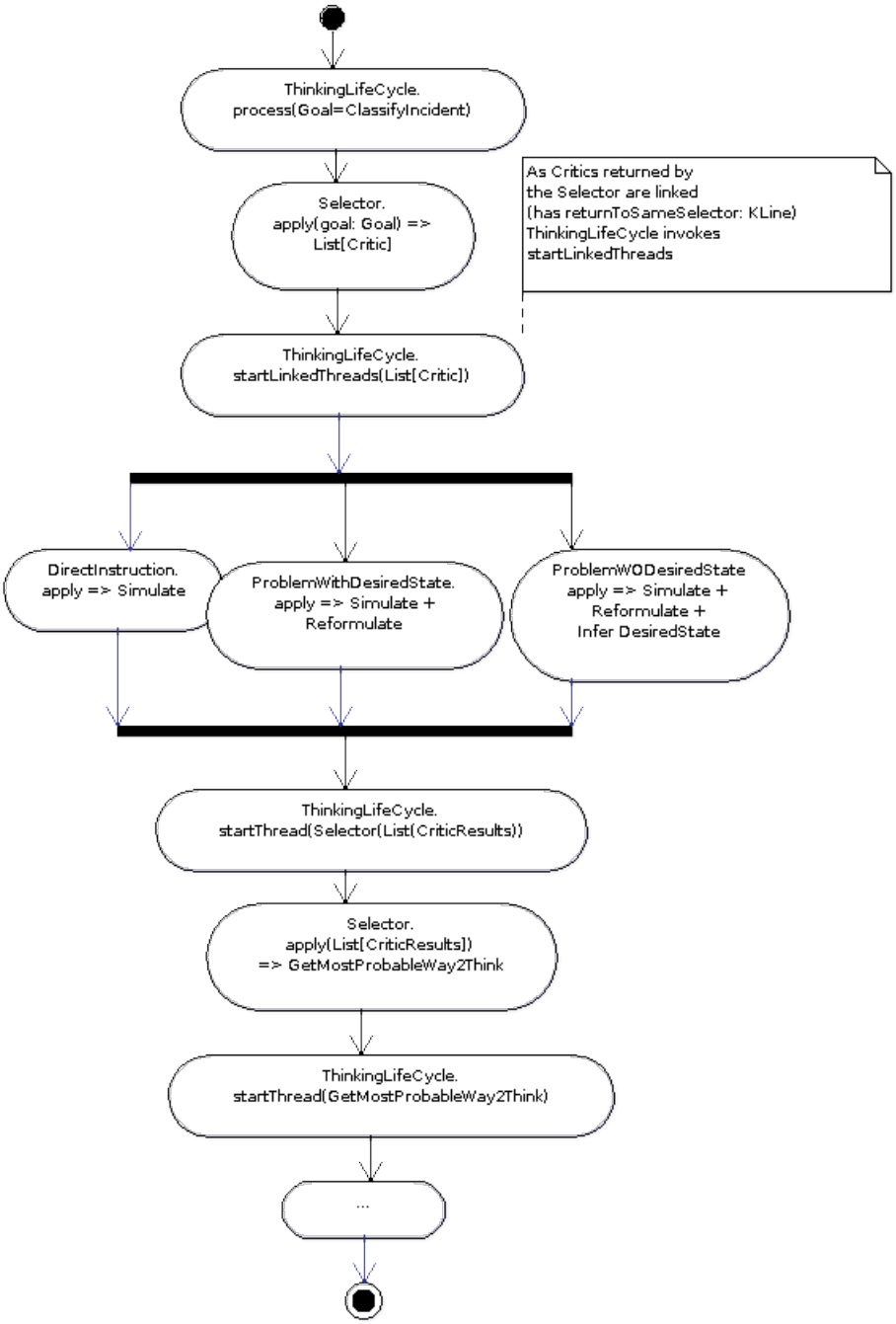

Рисунок 5.19: Диаграмма действий классификации инцидента

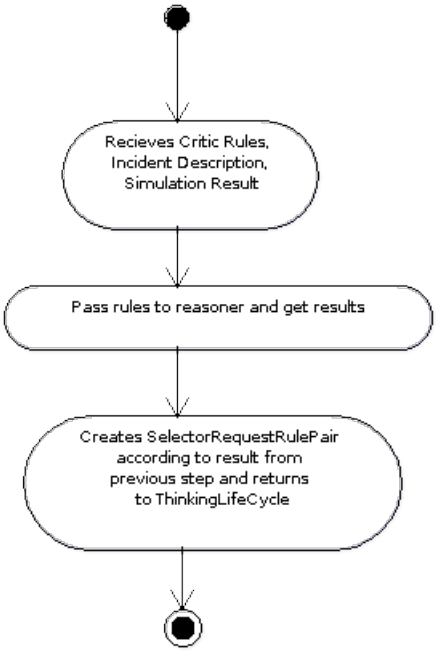

Рисунок 5.20: Диаграмма действий работы Critic

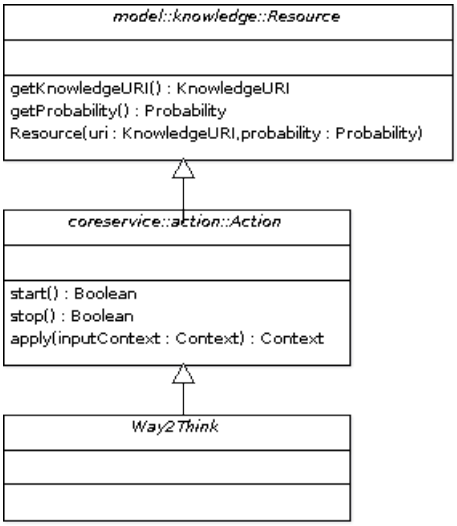

Рисунок 5.21: Интерфейс компонента WayToThink

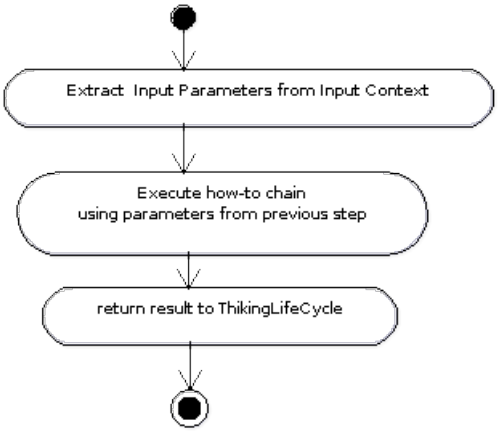

Рисунок 5.22: Работа компонента WayToThink в режиме HowTo

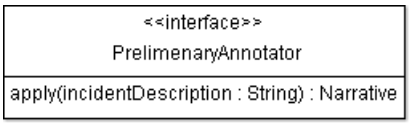

Рисунок 5.23: Интерфейс компонента PrelimenaryAnnotator

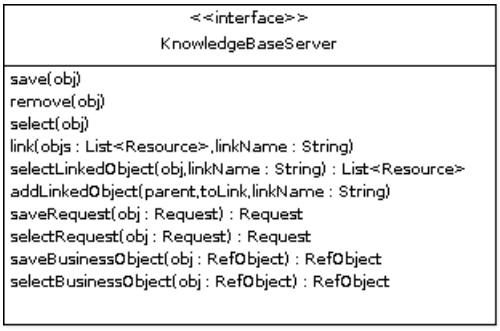

Рисунок 5.24: Интерфейс компонента KnowledgeBaseServer

5.2. Прототип

В прототипе были реализованы 4 уровня мышления. Основной рабочий поток приложения описан следующим алгоритмом:

1. Поступает запрос от пользователя
   User had received wrong application. User has ordered Wordfinder Business Economical. However she received wrong version, she received Wordfinder Tehcnical instead of Business Economical. Please assist.
2. GoalManger устанавливает цель системы HelpUser
3. Активируется набор Critic, привязанный к данной цели
4. PreliminaryAnnorator разбирает фразу
5. KnowledgeBaseAnnotator создает семантическую сеть и ссылки на нее
6. Critic на Рефликсивном уровне запускает WayToThink ProblemSolving с целью: ResolveIncident
7. Critic на Рефликсивном уровне выбирает WayToThink KnowingHow
   (a) Запускаются параллельно все Critic, которые привязаны к IncidentClassification Critic, который привязан к ResolveIncident цели, в данном случае это DirectInstruction, ProblemWithDesiredState, ProblemWithoutDesiredState 5.1.3
   (b) Selector выбирает наиболее вероятный результат работы среди всех результатов компонентов. В данном случае будет результат работы Problem Description with desired state.
   (c) KnowingHow сохраняет варианты выбора Selector.
   (d) Simulation WayToThink с параметрами Создать модель текущий ситуации создает модель CurrentSituation. User, Software
   (e) Reformulation WayToThink, используя результаты предыдущего шага синтезирует артефакты, которых не хватает, чтобы получить CurrentState и DesiredState. DesiredState не указан явно. WayToThink запускает Critic размышления, чтобы найти корень проблемы. Critic размышления находит CurrentState- Wordfinder Tehcnical, DesiredState-Wordfinder Business Economical
   (f) Рефлексивные Critic оценивают состояние системы - на каком шаге она находится, и если цель не достигнута, то запускают другой WayToThink, который был возвращен, например, DirectInstruction.
   (g) Critic генерации решения запускает KnowingHow D WayToThink, ExtensiveSearch.

(h) Selector выбирает наиболее вероятный путь мышления. В данном случае ExtensiveSearch, который будет находить решения, позволяющие привести систему в необходимое состояние (DesiredState). Если он не сможет, то он инициирует коммуникацию с пользователем.

8. Рефлексивный Critic проверяет состояние системы. Если Цель достигнута, то пользователю посылается ответ.

9. Само Сознательные Critic активируется на данном шаге и сохраняют информацию о затратах на решение.

5.2.1. UML диаграмма действий приложения

На Рисунке 5.25 представлена UML диаграма действий системы.

5.3. Испытание прототипа

5.4. Выводы по главе

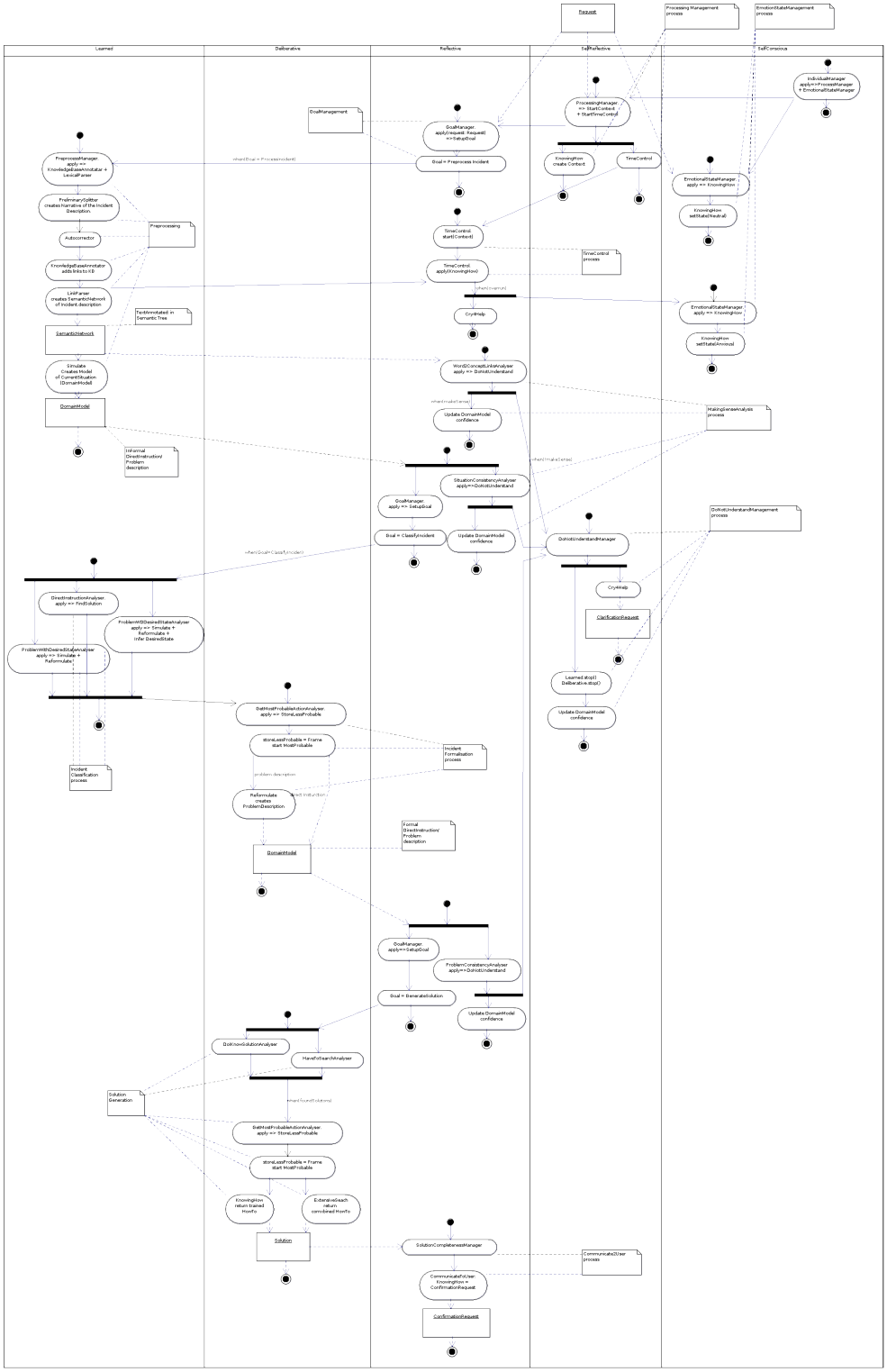

Рисунок 5.25: Диаграмма действий LifecycleActivity

Заключение

Основные результаты работы заключаются в следующем.

1. На основе анализа …
2. Численные исследования показали, что …
3. Математическое моделирование показало …
4. Для выполнения поставленных задач был создан …

И какая-нибудь заключающая фраза.

Список литературы

1. *Wikipedia*. IBM Watson. — web. — 2014. https://ru.wikipedia.org/wiki/IBM_Watson.
2. *Wolfram*. Wolfram Alpha. — web. — 2014. — 00. http://www.wolframalpha.com/.
3. *Minsky Marvin*. The Emotion Machine. — Simon & Schuster, 2006.
4. *Лермонтов Михаил Юрьевич*. Собрание сочинений: в 4 т. — М.: Терра-Кн. клуб, 2009. — 4 т.
5. *Foundation Apache Software*. Apache OpenNLP. — web. — 2012. — 04. https://opennlp.apache.org/.
6. *Goetzel Ben*. OpenCog RelEx. — web. — 2012. — 04. http://wiki.opencog.org/w/RelEx.
7. *Вебер П., Вилльямс Д.* Введение в обработку информации / Под ред. Т. Зителло. — Upper Saddle River, New Jersey 07458: Прентис Холл, 2009. — 581 с.
8. *Гринберг Д.* Надежный алгоритм обработки для грамматики // *Технический отчет Университета Карнеги Мелон CMU-CS-95-125*. — 1995. — 1 июля.
9. *Соколов А. Н., Сердобинцев К. С.* HP OpenView System Administration Handbook: Network Node Manager, Customer Views, Service Information Portal, HP OpenView Operations / Ed. by X. Шутзе. — Астрахань: Издательство, 2004. — 688 pp.
10. *Stuart Russell Peter Norvig*. Artificial Intelligence. A Modern approach. — Pearson, 2010.
11. *Хокинг С.* ТЕОРИЯ ВСЕГО. — Москва: Амфора, 2009. — 160 с.
12. *Minsky Marvin*. The Society of Mind. — Simon & Schuster, 1988.
13. *Вайтт Д.* Akka Concurrency / Под ред. К. Роланд. — Артима, 2013. — 521 с.
14. *Одерски М., Спун Л., Веннерс Б.* Programming in Scala, Second Edition. A comprehensive step-by-step guide. — Артима, 2010. — 883 с.
15. *Робинсон С.* WebSphere Application Server 7.0 Administration Guide. — PACKT publishing, 2009. — 344 с.

16. *Гойтз Б., Пейерлс Т., Блох Д.* Java Concurrency in Practice. — Addison-Wesley Professional; 1 edition, 2006. — 384 с.

Список рисунков

Список таблиц

Приложение А

Приложение А. Интерфейсная модель

Интерфейсная модель содержит классы и интерфейсы для взаимодействия с пользователем.

RefObject

Представляет собой общий объект, который сохраняется в Базе Знаний. (Базовый класс для всех остальных классов и объектов)

– ObjectID- уникальный в пределах класса ключ

– Reference- уникальный в пределах всех баз знаний ключ

– Name-имя объекта

Request

Объект для хранения запроса пользователя.

– SubscriptionID - идентификатор подписки

– RequestText - запрос пользователя в виде текста

– Solution - ссылка на решение запроса

– State - статус запроса (например, Поиск Решения)

– FormalizedRequest - ссылка на формализованный запрос

Subscription

Информация о подписке пользователя на события

– Endpoints - списко UserEndpoint, которые будут использоваться для обратной связи с пользователем

UserEndpoint

– Type - тип точки связи с пользователем (например, веб-сервис)

– Address - адрес точки связи с пользователем

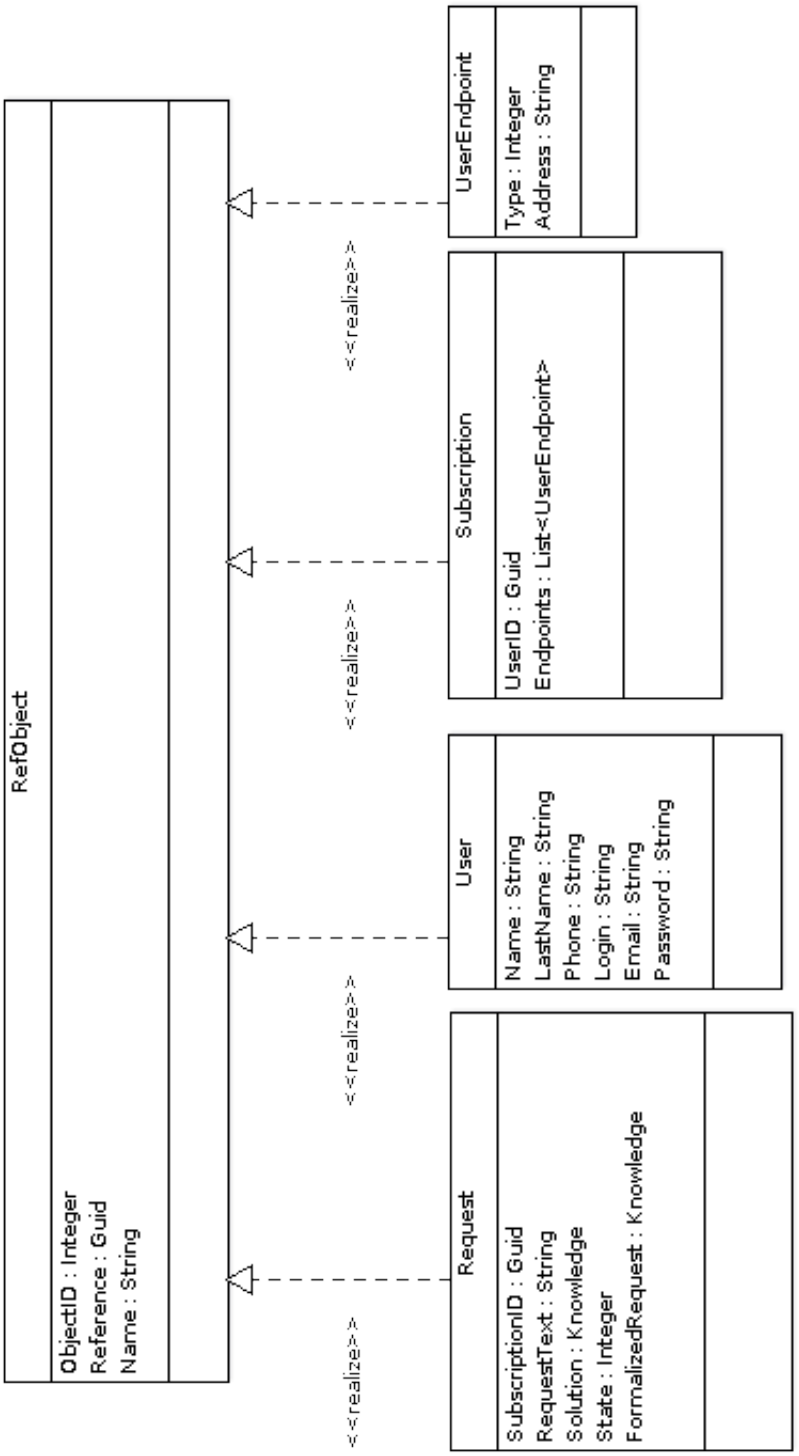

Рисунок А.1: Диаграмма классов интерфейсной модели

Приложение B

Приложение B. Action

Action является базовым классом для WayToThink или Critic.

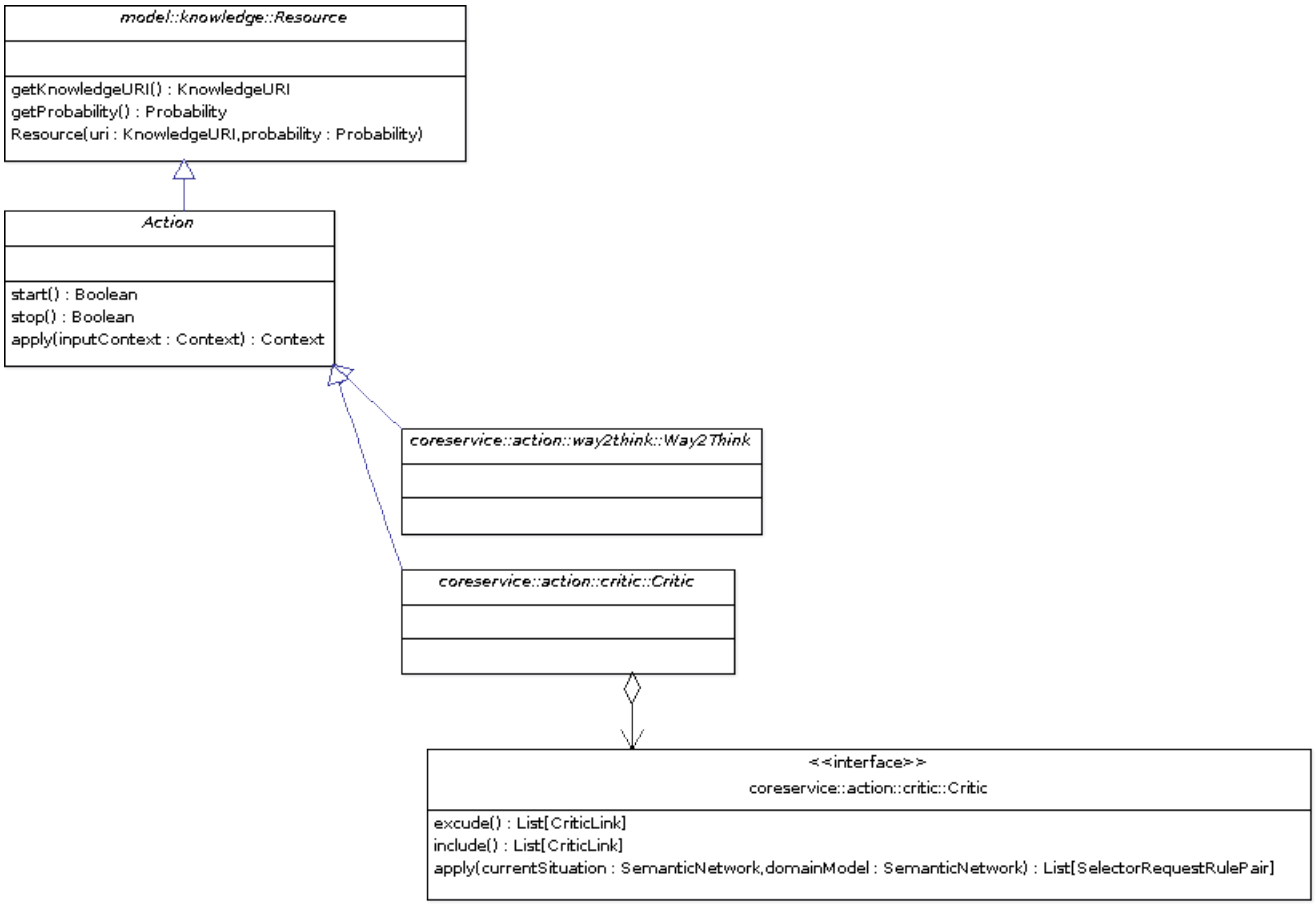

Рисунок B.1: Диаграмма классов Action

Приложение C

Приложение C. Цели

Goal (Цель) является набором вероятностных предикатов и последовательностью How-To необходимых для того, чтобы достичь цель. Goal и How-To тесно связаны. На Рисунке C.1 показан состав Goal. Goal состоит из:

1. Parameters - параметры, которые используются предикатами для выполнения
2. Precondition - условия, которые должны быть выполнены до выполнения проверок цели
3. Entry criteria - входной критерий, предикат, который определяет, что цель активировалась
4. Exite criteria - условия, когда цель считается выполненной
5. PostCondition - дополнительные условия для выхода
6. HowTo - набор решения. Список путей решения

Типы предикатов

В решение используется 3 типа логических предикатов: and, or, not. Представление Goal в SemanticNetwork показано на диаграмме C.2.

Иерархия целей имеет высшую цель: Помочь пользователю. Далее вниз по иерархии идут подцели: Решить инцидент, Понять тип инцидента, Найти решение инцидента и т.д.

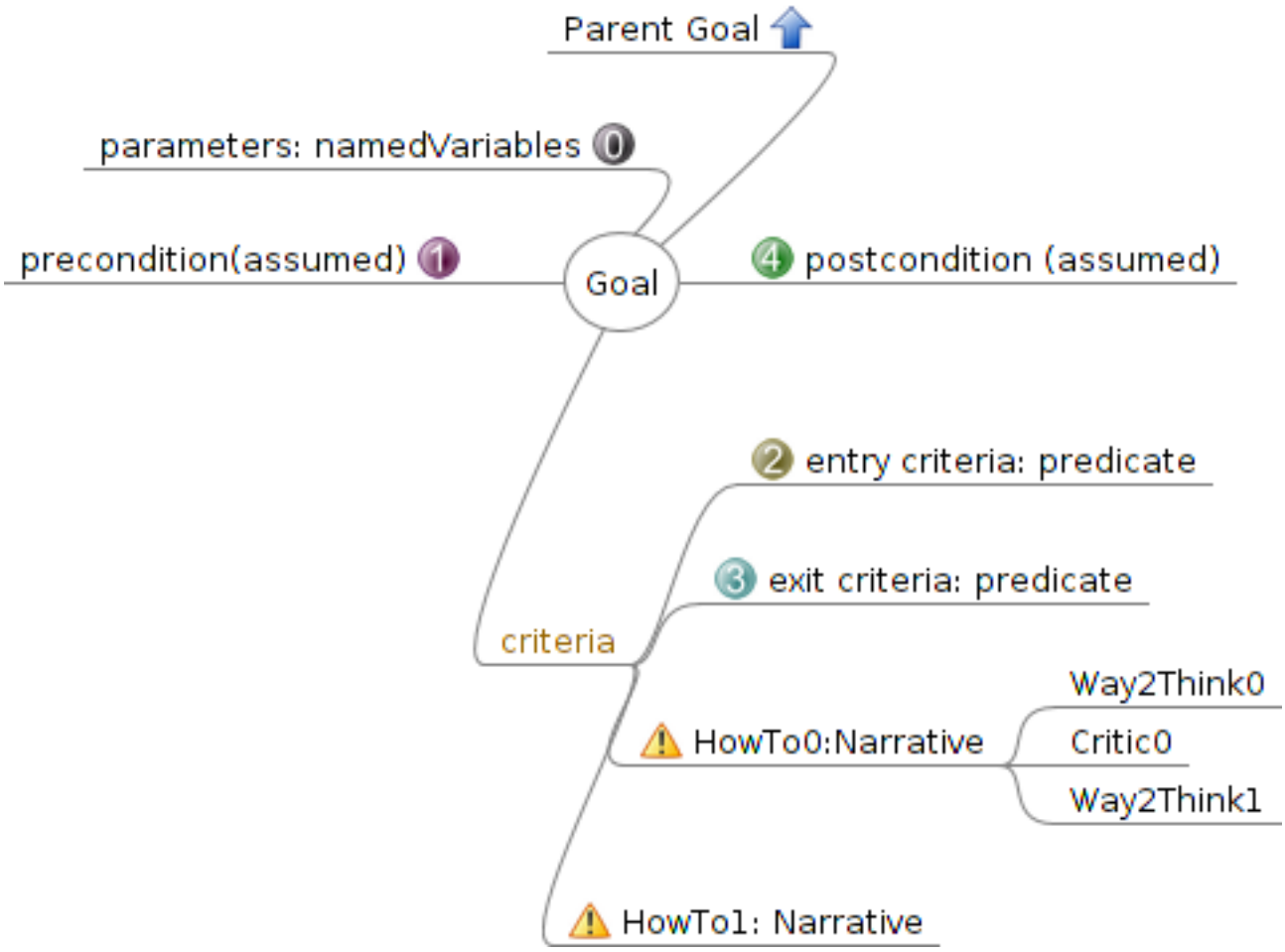

Рисунок С.1: Диаграмма классов Goal

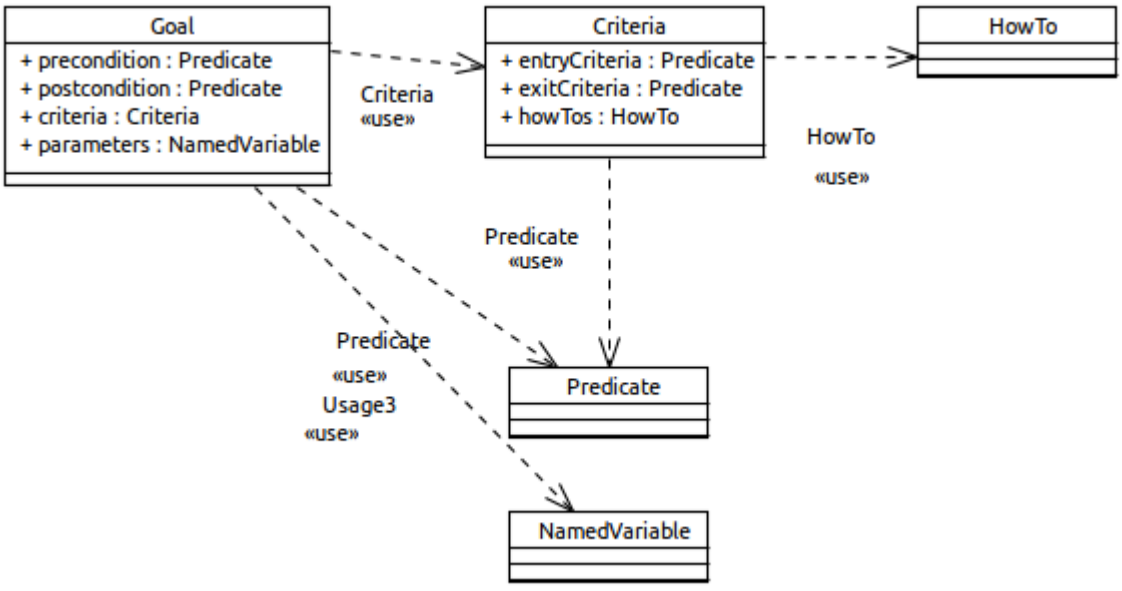

Рисунок С.2: Диаграмма места Goal в SemanticNetwork (Семантической сети)

Приложение D

Приложение D. Рецепты решений

Рецепты решений представляют собой последовательность действий выполняемых для решения проблемы, описанной в инциденте. Было разработано два типа HowTo: ValueHowTo-содержит в себе простое значение; FunctionalHowTo- содержит в себе функцию.
FunctionalHowTo состоит из следующих частей:

1. FunctionalBody - тело функции, описывающий содержание функции
2. InputParameters - входные параметры функции
3. OutputParameters - выходные параметры

Комбинация FunctionaHowTo и ValueHowTo является Рецептом Решения. Например, решение проблемы неработающего сегмента кластера в формате для специалиста технической поддержки.

- Войти на сервер U1
- Запустить утилиту 12 для Windows Servers
- Открыть вкладку 1
- Перейти на All Managed Server, найти нужный Server из правой панели, открыть свойства сервера
- Нажать на Backup Exec Services
- Выберите проблемный сегмент кластера
- Нажмите Restart all Services
- Подождите и проверьте статус

Преобразованный в формат HowTo данный рецепт решения будет выглядеть как

login:howto Parameters:[
Key:’ScriptName’, Value:’LogonScript.bat’, Key:’Description’, Value:’Logon to server’]
InputParameters:[Key:’ServerName’, Value:’U1’, Key:’UserName’, Value:’MyUser’]
OutputParameters:[Key:’SessionID’, Value:’SSSE12’,
]
launch:howto Parameters:[
Key:’ScriptName’, Value:’LaunchScript.bat’, Key:’Description’, Value:’Launch the application’]
InputParameters:[Key:’ExecName’, Value:’Utility12.exe’,]
OutputParameters:[Key:’SessionID’, Value:’SSSE12’,
]

Приложение Е

test

Е.1. Подраздел приложения

Вот размещается длинная таблица:

Параметр	Умолч.	Тип	Описание
&INP			
kick	1	int	0: инициализация без шума ($p_s = const$) 1: генерация белого шума 2: генерация белого шума симметрично относительно экватора
mars	0	int	1: инициализация модели для планеты Марс
kick	1	int	0: инициализация без шума ($p_s = const$) 1: генерация белого шума 2: генерация белого шума симметрично относительно экватора
mars	0	int	1: инициализация модели для планеты Марс
kick	1	int	0: инициализация без шума ($p_s = const$) 1: генерация белого шума 2: генерация белого шума симметрично относительно экватора
mars	0	int	1: инициализация модели для планеты Марс
kick	1	int	0: инициализация без шума ($p_s = const$) 1: генерация белого шума 2: генерация белого шума симметрично относительно экватора
mars	0	int	1: инициализация модели для планеты Марс
kick	1	int	0: инициализация без шума ($p_s = const$) 1: генерация белого шума 2: генерация белого шума симметрично относительно экватора
mars	0	int	1: инициализация модели для планеты Марс
kick	1	int	0: инициализация без шума ($p_s = const$) 1: генерация белого шума 2: генерация белого шума симметрично относительно экватора
mars	0	int	1: инициализация модели для планеты Марс
kick	1	int	0: инициализация без шума ($p_s = const$) 1: генерация белого шума 2: генерация белого шума симметрично относительно

продолжение следует

(продолжение)			
Параметр	Умолч.	Тип	Описание
			экватора
mars	0	int	1: инициализация модели для планеты Марс
kick	1	int	0: инициализация без шума ($p_s = const$) 1: генерация белого шума 2: генерация белого шума симметрично относительно экватора
mars	0	int	1: инициализация модели для планеты Марс
kick	1	int	0: инициализация без шума ($p_s = const$) 1: генерация белого шума 2: генерация белого шума симметрично относительно экватора
mars	0	int	1: инициализация модели для планеты Марс
kick	1	int	0: инициализация без шума ($p_s = const$) 1: генерация белого шума 2: генерация белого шума симметрично относительно экватора
mars	0	int	1: инициализация модели для планеты Марс
kick	1	int	0: инициализация без шума ($p_s = const$) 1: генерация белого шума 2: генерация белого шума симметрично относительно экватора
mars	0	int	1: инициализация модели для планеты Марс
kick	1	int	0: инициализация без шума ($p_s = const$) 1: генерация белого шума 2: генерация белого шума симметрично относительно экватора
mars	0	int	1: инициализация модели для планеты Марс
kick	1	int	0: инициализация без шума ($p_s = const$) 1: генерация белого шума 2: генерация белого шума симметрично относительно экватора
mars	0	int	1: инициализация модели для планеты Марс
kick	1	int	0: инициализация без шума ($p_s = const$) 1: генерация белого шума 2: генерация белого шума симметрично относительно экватора
mars	0	int	1: инициализация модели для планеты Марс
kick	1	int	0: инициализация без шума ($p_s = const$) 1: генерация белого шума 2: генерация белого шума симметрично относительно экватора
mars	0	int	1: инициализация модели для планеты Марс
&SURFPAR			
kick	1	int	0: инициализация без шума ($p_s = const$) 1: генерация белого шума 2: генерация белого шума симметрично относительно экватора
mars	0	int	1: инициализация модели для планеты Марс
kick	1	int	0: инициализация без шума ($p_s = const$) 1: генерация белого шума 2: генерация белого шума симметрично относительно экватора
mars	0	int	1: инициализация модели для планеты Марс
			продолжение следует

(продолжение)			
Параметр	Умолч.	Тип	Описание
kick	1	int	0: инициализация без шума ($p_s = const$) 1: генерация белого шума 2: генерация белого шума симметрично относительно экватора
mars	0	int	1: инициализация модели для планеты Марс
kick	1	int	0: инициализация без шума ($p_s = const$) 1: генерация белого шума 2: генерация белого шума симметрично относительно экватора
mars	0	int	1: инициализация модели для планеты Марс
kick	1	int	0: инициализация без шума ($p_s = const$) 1: генерация белого шума 2: генерация белого шума симметрично относительно экватора
mars	0	int	1: инициализация модели для планеты Марс
kick	1	int	0: инициализация без шума ($p_s = const$) 1: генерация белого шума 2: генерация белого шума симметрично относительно экватора
mars	0	int	1: инициализация модели для планеты Марс
kick	1	int	0: инициализация без шума ($p_s = const$) 1: генерация белого шума 2: генерация белого шума симметрично относительно экватора
mars	0	int	1: инициализация модели для планеты Марс
kick	1	int	0: инициализация без шума ($p_s = const$) 1: генерация белого шума 2: генерация белого шума симметрично относительно экватора
mars	0	int	1: инициализация модели для планеты Марс
kick	1	int	0: инициализация без шума ($p_s = const$) 1: генерация белого шума 2: генерация белого шума симметрично относительно экватора
mars	0	int	1: инициализация модели для планеты Марс

Е.2. Ещё один подраздел приложения

Нужно больше подразделов приложения!

Е.3. Очередной подраздел приложения

Нужно больше подразделов приложения!

Е.4. И ещё один подраздел приложения

Нужно больше подразделов приложения!