

**РУКОВОДСТВО РАЗРАБОТЧИКА ЭЛЕКТРОННОГО
СРЕДСТВА УЧЕБНОГО НАЗНАЧЕНИЯ
«ИНТЕЛЛЕКТУАЛЬНАЯ СПРАВОЧНАЯ СИСТЕМА ПО
ПЛАНИМЕТРИИ»**

МИНСК 2014

1 Назначение системы

Интеллектуальная справочная система по геометрии предназначена для информационного обслуживания пользователя, путём ответа на вопросы предметной области планиметрии и осуществления решения задач из этой же области.

Данная система предназначена как для школьников, учащихся СУЗов, ВУЗов, преподавателей геометрии, так и для людей, интересующихся геометрией.

При изучении геометрии учащиеся могут испытывать трудности. Причём зачастую с этими трудностями учащиеся сталкиваются дома при выполнении домашних заданий, подготовке к контрольным работам. Им необходима консультация, помощь в решении задач, объяснение способа решения задачи проверка своих знаний по пройденному материалу.

Преподаватели могут использовать данную систему в качестве вспомогательного материала на уроках (занятиях и т.д).

Основой для реализации системы по геометрии является открытая семантическая технология проектирования интеллектуальных систем OSTIS. Больше информации о технологии можно найти на сайте проекта OSTIS <http://ostis.net> и сайте интеллектуальной Метасистемы OSTIS <http://ims.ostis.net>.

2 Основные возможности системы

К основным возможностям системы относятся следующие:

- Осуществление навигации по базе знаний системы путем:
 - осуществления перехода от одной семантической окрестности элемента семантической сети к другой по ссылке
 - использования поисковых операций в пункте меню графического интерфейса системы
- Осуществление решения задач из предметной области геометрии на основе имеющихся у системы знаний, включая объяснение способа решения
- Предоставление возможности создания геометрических чертежей, понимаемых системой при помощи специализированного редактора.

3 Системные требования (программная и аппаратная составляющая)

Система поставляется как web-приложение, которое может быть развернуто как на локальном сервере, так и на удаленном ресурсе.

Требования к серверной части:

	Минимальные	Рекомендуемые
Программные требования	ОС: ОС на базе Linux	ОС: Ubuntu 14.04 LTS
Аппаратные требования	Процессор: 1 ядро 385 МГц ОЗУ: Свободное место на HDD: 10 Гб Доступ к Internet	Процессор: 2 ядра по 1 ГГц ОЗУ: Свободное место на HDD: 20 Гб Доступ к Internet

Требования к клиентской части:

Web-браузер с поддержкой последних стандартов HTML и JavaScript, например Google Chrome

4 Установка системы

4.1 Процесс установки и сборки системы

Для установки системы на сервере должна быть предустановлена операционная система, указанная в системных требованиях.

Далее для работы необходимо установить систему контроля версий Git. Это можно сделать при помощи команды

```
sudo apt-get install git
```

После этого необходимо загрузить установочные скрипты в какую-либо папку на сервере при помощи команды

```
git clone https://github.com/geometryostis/geometry.ostis.git
```

Для установки и сборки системы достаточно перейти в нужный каталог и выполнить скрипт сборки:

```
cd geometry.ostis/scripts
```

```
./prepare.sh
```

Далее при установке нужно руководствоваться сообщениями, которые скрипт будет выводить в терминал.

После успешной установки системы для ее запуска необходимо выполнить следующие команды в том же каталоге

```
./build_kb.sh – сборка базы знаний из исходников
```

```
./run_sctp.sh & – запуск sctp-сервера
```

```
./run_scweb.sh & - запуск серверной части web-интерфейса
```

Для остановки системы нужно выполнить команды

```
pkill python
```

```
pkill sctp-server
```

4.2 Структура, программные компоненты, их назначение, структура каталогов

Программная реализация системы представлена следующим набором базовых компонентов:

- 1) Исходные тексты базы знаний собственно системы по планиметрии (включая тексты программ и спецификации поисковых агентов)

```
geometry.ostis/kb
```

- 2) Исходные тексты базы знаний ядра технологии OSTIS

```
geometry.ostis/ims.ostis.kb
```

- 3) Хранилище содержимого базы знаний и платформенно-зависимые реализации базовых агентов обработки

```
geometry.ostis/sc-machine
```

- 4) Интерпретатор scp-программ

```
geometry.ostis/scp-machine
```

- 5) Геометрический интерфейс

```
geometry.ostis/geometry.drawings
```

- 6) Скрипты установки

```
geometry.ostis/scripts
```

5 Архитектура системы

5.1 Компоненты системы

Интеллектуальная система в целом рассматривается нами как многоагентная система, ориентированная на обработку семантического пространства, представленного семантической сетью, принадлежащей SC-коду. Назовем это sc-моделью интеллектуальных систем.

Абстрактная sc-модель интеллектуальной системы – это динамическая система, состоящая

- (1) из абстрактной структурно перестраиваемой ассоциативной sc-памяти, в которой хранятся и обрабатываются тексты SC-кода и переработка информации в которой сводится к изменению конфигурации связей между sc-элементами
- (2) из коллектива внутренних sc-агентов – самостоятельных субъектов, каждый из которых способен выполнять соответствующей ему класс действий, направленных на изменение состояния sc-памяти. Каждое такое действие инициируется возникновением в sc-памяти ситуации или события определенного вида. Результатом каждого такого действия является генерация (создание) новых ситуаций или событий в sc-памяти. Частью такого результата является регистрация факта успешно или неуспешно выполненного действия (это часть языковых средств синхронизации действий sc-агентов)
- (3) из коллектива рецепторных sc-агентов (интеллектуальных датчиков), каждый из которых реагирует на определенные ситуации или события во внешней среде и отображает (описывает) их в sc-памяти
- (4) из коллектива эффекторных sc-агентов, каждый из которых выполняет действие соответствующего класса, направленные на изменения состояния внешней среды. Каждое действие инициируется соответствующей командой, появившейся в sc-памяти.

5.2 Принципы взаимосвязи и взаимодействия компонентов

В основе рассматриваемой системы лежит принцип интеграции различных компонентов на общей формальной основе. Рассмотрим основные принципы взаимодействия любых агентов обработки знаний в рамках системы.

В предлагаемом подходе к построению машин обработки знаний сама машина рассматривается в неклассическом варианте. В данном случае машина обработки знаний представляет собой графодинамическую sc-машину (память в качестве модели представления знаний использует семантическую сеть с теоретико-множественной интерпретацией), состоящую из двух частей:

- графодинамическая sc-память;
- система sc-операций (sc-агентов).

Система операций является агентно-ориентированной и представляет собой коллектив sc-операций, условием инициирования которых является появление в памяти системы некоторой определенной конструкции. При этом операции взаимодействуют между собой через память системы посредством генерации конструкций, являющихся условиями инициирования для другой операции. При таком подходе становится возможным обеспечить гибкость и расширяемость функционала системы путем добавления или удаления из ее состава некоторого набора операций.

Отличительной особенностью машины обработки знаний как многоагентной системы в рамках данного подхода является принцип взаимодействия операций-агентов. По сути, предлагаемый подход реализует принцип «доски объявления», рассматриваемый в теории многоагентных систем. Агенты обмениваются сообщениями исключительно через общую память путем использования соответствующего языка взаимодействия (языка вопросов-ответов, рассматриваемого далее), в отличие от большинства классических МАС, в которых агенты обмениваются сообщениями

непосредственно друг с другом. В рассматриваемом подходе каждый агент, формулируя вопросную конструкцию в памяти, априори не знает, какой из агентов будет обрабатывать указанную конструкцию, а лишь дожидается появления в памяти факта окончания обработки вопроса. При этом в решении поставленной таким образом задачи может принимать участие целый коллектив агентов. Аналогичным образом, реагируя на появление некоторой конструкции в памяти, агент в общем случае не знает, кто из его коллег поставил данный вопрос, а лишь может проверить соответствие сгенерированной конструкции своему условию инициирования. В случае наличия такого соответствия, агент начнет обработку указанного вопроса (решение поставленной задачи), и в результате работы сгенерирует некоторый ответ на поставленный вопрос.

Проверка соответствия сгенерированного вопроса условиям инициирования агентов происходит следующим образом: автору вопроса после его формулирования необходимо инициировать данный вопрос (включить его во множество инициированных вопросов). После инициирования вопроса каждый из агентов, работающих в памяти, переходит в активное состояние и начинает проверку условия инициирования. При этом проверка начинается с наиболее уникальных фрагментов условия (например, типа вопроса) с целью оптимизации данного процесса. В случае установления факта изоморфности вопросной конструкции и условия инициирования агент начинает решение поставленной задачи, в противном случае агент переходит в состояние пассивного ожидания.

С целью обеспечения модульности предлагаемой модели, все агенты изначально активизируются при возникновении в памяти системы одной и той же конструкции – выходящей дуги из узла инициированный вопрос, описанного в разделе о языке вопросов. Дальнейшее поведение агента определяется, как уже было сказано, соответствием вопросной конструкции условию инициирования агента. Несмотря на то, что все агенты активизируются при инициировании любого вопроса, логично считать активными только те из них, которые остались таковыми и после проверки соответствия вопросной конструкции условию инициирования. Все остальные акты активизации можно считать «ложными срабатываниями», поскольку реальных изменений в систему они не привносят.

Описанная модель взаимодействия агентов в общей памяти позволяет обеспечить максимальную расширяемость системы агентов и предельно упростить процесс добавления новых агентов в уже имеющийся коллектив.

Одним из основных преимуществ предлагаемой модели является ее ориентация на параллельную обработку знаний. Широкие возможности для реализации параллелизма обусловлены следующими моментами:

- Основными компонентами решателя являются sc-операции, по сути представляющие собой автономные самостоятельные агенты над общей памятью;
- Процедуры, реализующие операции решателя могут быть описаны как параллельные программы. Внутренний язык программирования SCP, являющийся основным языком реализации процедур решателя, изначально является языком параллельного программирования.
- Сама концепция использования графодинамической ассоциативной памяти как среды взаимодействия операций предоставляет широкие возможности для параллелизма. Единственным условием в данном случае является наличие в реализации памяти стандартных механизмов синхронизации, например, таких как блокировки.

Следует также отметить немаловажный момент: для описания процедур, реализующих принципы работы того или иного агента (т.е. программ агента) используется специализированный язык SCP, построенный на базе SC-кода, как и в случае с представлением знаний, предназначенных для обработки. Такой подход имеет ряд преимуществ:

И программа агента, и обрабатываемые знания, по сути, представлены на одном и том же языке. В связи с этим преобразование восприятий агента в его действия, описываемое функцией агента, значительно упрощается, т.к. отсутствует необходимость дополнительных преобразований во внутреннее представление агента;

Так как алгоритм работы агента описан на том же языке, что и другие знания в системе, то появляется возможность модифицировать сам алгоритм того или иного агента прямо в процессе его работы. Это предоставляет широкие возможности для построения принципиально нового класса программ и, соответственно, агентов, способных к самоконфигурированию в процессе работы.

6 Добавление фрагментов базы знаний

6.1 Базовый язык представления знаний (SC-код)

Достоинствами SC-кода являются следующие его свойства:

- Все основные семантические связи между текстами (семантическая эквивалентность, семантическое включение, семантическое пересечение) в SC-коде становятся теоретико-множественными (равенство, включение, пересечение множеств).
- Неограниченная возможность перехода от sc-текстов к sc-метатекстам, содержащим знаки описываемых sc-текстов.
- Тексты SC-кода (в том числе и те тексты, sc-знаки которых явно вводятся в рамках соответствующих им метатекстов) могут быть иерархическими структурами, имеющими любое число уровней иерархии, поскольку sc-элемент может обозначать множество, состоящее из любых sc-элементов (в т.ч. и из sc-элементов, обозначающих любые множества других sc-элементов). В отличие от этого, например, такие традиционные структуры, как алгебраические системы, являются трехуровневыми:
 - На первом уровне – элементы носителя (основного множества) алгебраической системы;
 - На втором уровне – кортежи, элементами которых являются элементы носителя;
 - На третьем уровне – отношения, элементами которых являются указанные кортежи.

Формализация знаний на основе SC-кода предполагает теоретико-множественную интерпретацию всех sc-элементов, не являющихся знаками внешних сущностей. Все такие sc-элементы являются знаками множеств sc-элементов и необходимо, прежде всего, уточнить то, какие sc-элементы являются знаками этих множеств. При этом совсем не обязательно, чтобы все эти sc-элементы были представлены в текущем состоянии sc-памяти.

Тексты SC-кода будем называть sc-текстами. Знаки, входящие в состав sc-текстов, будем называть sc-элементами. Переход от общего понятия семантической сети к унифицированным семантическим сетям (sc-текстам) рассмотрим как задание целого ряда ограничений на семантические сети общего вида, но таких ограничений, которые не снижают семантической мощности языка семантических сетей, претендующего на универсальность.

Ограничение 1. Семантическая нормализация описываемых множеств. Если элемент семантической сети (sc-элемент) является знаком некоторого множества, то каждый элемент этого множества, представляющий собой сущность, не являющуюся знаком (sc-элементом), заменяется на знак указанной сущности (на sc-элемент, обозначающий эту сущность). Таким образом, все описываемые sc-текстом множества становятся множеством знаков, а точнее, множеством sc-элементов (sc-множествами). Такие множества условно будем называть семантически нормализованными. Очевидно, любое множество может быть представлено в семантически нормализованном виде. Для описания связей между sc-множествами и элементами этих множеств вводятся связи принадлежности. Каждая из этих связей связывает знак некоторого sc-множества с одним из элементов этого множества, который всегда является знаком (sc-элементом) благодаря семантической нормализации sc-множества.

Ограничение 2. Каждая связь описываемых сущностей трактуется как множество, элементами которого являются описываемые сущности, связываемые этой связью. Семантическая нормализация каждого такого множества означает то, что все

связи, входящие в состав семантической сети, будут связывать не сами описываемые сущности, а знаки этих сущностей за исключением случаев, когда указанные сущности уже являются знаками. Напомним при этом, что знаки описываемых сущностей, связи между знаками описываемых сущностей и знаки этих связей включаются в число сущностей, описываемых семантической сетью. Подчеркнем также, что связь-*i* между *sc*-элементом, который является знаком некоторой связи-*j*, и *sc*-элементом, который является компонентом этой связи-*j* (т.е. является знаком, связываемым связью-*j*), в *sc*-тексте может быть представлена двумя способами:

- с помощью явно вводимого *sc*-элемента, обозначающего связь принадлежности;
- с помощью синтаксически задаваемой пары инцидентности, для которой соответствующий ей *sc*-элемент (знак) не вводится.

Ограничение 3. Если в состав семантической сети входит знак небинарной связи, то связи между этим знаком и компонентами обозначаемой им связи задаются не с помощью пар инцидентности (на синтаксическом уровне), а явно с помощью явно вводимых связей принадлежности. Это означает, что в текстах *SC*-кода все небинарные связи представляются с помощью бинарных (сводятся к бинарным).

Ограничение 4. Каждая семантическая сеть и, в частности, каждая унифицированная семантическая сеть (*sc*-текст), которая описывается и, соответственно, обозначается в другой семантической сети (метасети), трактуется как семантически нормализованное множество, элементами которого являются все те и только те знаки, которые входят в состав описываемой семантической сети.

Ограничение 5. В рамках *SC*-кода четко задается и минимизируется алфавит элементов семантических сетей (Алфавит знаков, входящих в состав унифицированных семантических сетей, Алфавит *sc*-элементов). Указанный алфавит представляет собой семейство тех классов элементов унифицированных семантических сетей (*sc*-элементов), принадлежность *sc*-элементов которым задается не с помощью явно вводимых связей принадлежности, а с помощью синтаксически задаваемых меток *sc*-элементов. При этом каждой такой метке взаимно однозначно соответствует свой синтаксически задаваемый класс *sc*-элементов. Семейство таких классов, т.е. Алфавит *sc*-элементов, включает в себя:

- Класс *sc*-узлов
- Класс *sc*-ссылок на внешние информационные ресурсы
- Класс *sc*-ребер
- Класс *sc*-дуг общего вида
- Класс *sc*-дуг принадлежности или не принадлежности
- Класс стационарных константных *sc*-дуг принадлежности

Ограничение 6. В рамках *SC*-кода четко задаются правила перехода *sc*-элементов из одного синтаксически задаваемого класса *sc*-элементов в другой при полном сохранении его семантики (т.е. при сохранении обозначаемого им денотата). Речь идет об изменении синтаксического типа (метки) *sc*-элемента при появлении определенного вида новой (дополнительной) информации об этом *sc*-элементе. Так, например, *sc*-элемент может перейти:

- из Класса *sc*-узлов в Класс *sc*-ссылок;
- из Класса *sc*-узлов в Класс *sc*-ребер;
- из Класса *sc*-ребер в Класс *sc*-дуг общего вида;
- из Класса *sc*-дуг общего вида в Класс *sc*-дуг принадлежности или не принадлежности;
- из Класса *sc*-дуг принадлежности или не принадлежности в Класс стационарных константных *sc*-дуг принадлежности.

Ограничение 7. Важнейшая особенность любого языка, текстами которого являются семантические сети, (в том числе и SC-кода) заключается в том, что все элементы (примитивы, атомарные фрагменты) семантических сетей (в т.ч. и sc-элементы) являются знаками. При этом в SC-коде сущности, обозначаемые sc-элементами, делятся на два вида:

- семантически нормализованные множества, элементами которых являются sc-элементы;
- внешние описываемые сущности, не являющиеся множествами.

Таким образом, каждый sc-элемент есть либо знак множества (но не просто множества, а множества, элементами которого являются sc-элементы), либо "внешней" описываемой сущности (сущности, которая множеством не является). Особо подчеркнем то, что sc-элементов, являющихся знаками множеств sc-элементов большинство:

- каждая связь трактуется как множество sc-элементов, обозначающих связываемые сущности;
- каждая структура трактуется как множество всех sc-элементов, входящих в эту структуру;
- каждое понятие трактуется как множество sc-элементов, обозначающих сущности, являющиеся экземплярами этого понятия.

Следовательно, имеет место четкая теоретико-множественная трактовка таких сущностей, как связи, структуры и понятия. Это означает то, что SC-код имеет четкую базовую теоретико-множественную семантическую интерпретацию всех его sc-элементов – одни sc-элементы могут быть только элементами каких-либо множеств, а другие могут быть как элементами одних множеств, так и знаками других множеств. Все sc-элементы, кроме знаков связей принадлежности или не принадлежности, можно разбить на следующие уровни иерархии:

- первичные sc-элементы – знаки "внешних" сущностей;
- sc-элементы второго уровня – знаки множеств, элементами которых являются только первичные sc-элементы;
- sc-элементы третьего уровня – знаки множеств, среди элементов которых есть по крайней мере, один sc-элемент второго уровня, но нет ни одного sc-элемента более высокого уровня;
- и так – до бесконечности.

Ограничение 8. Все sc-элементы имеют базовую семантическую типологию, описанную в SC-тексте базовой типологии sc-элементов.

6.2 SCg-код

6.2.1 SCg-код и его sc-модель

SCg-код - один из возможных способов визуального представления sc-текстов (далее sc-текст отображенный с помощью SCg-кода будем называть sc.g-текстом). Основным принципом заложенным в основу SCg-кода является то, что каждому sc-элементу в соответствие ставится sc.g-элемент (графическое отображение). Другими словами любому изображаемому на экране sc.g-элементу соответствует некоторый sc-элемент в базе знаний. В рамках SCg-кода выделено несколько его уровней.




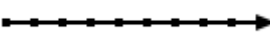

6.2.2 SCg-код 1-го уровня (SCg-ядро)

SCg-ядро - это наиболее близкий способ представления sc-текстов в графическом виде. Каждому элементу алфавита SCg-кода соответствует элемент алфавита SC-кода. Основными свойствами SCg-ядра являются:

- каждый sc.g-текст записанный с помощью SCg-ядра (SCg-кода 1-го уровня), изоморфен тому sc-тексту, который он графически изображает. В этом смысле средства SCg-ядра максимально близки к SC-коду.
- в sc.g-текстах, использующих SCg-ядро, как и в sc-текстах, запрещена синонимия элементов этих текстов.

Тексты записанные с помощью SCg-ядра удобны для иллюстрации синтаксических и семантических свойств SC-кода, но не удобны для широкого практического использования в качестве языка визуализации sc-текстов, т.к. не являются наглядными. Разработка SCg-кода как раз и направлена на разрешение этого противоречия. В таблице №1 представлен алфавит SCg-ядра.

Таблица №1. Алфавит SCg-ядра.

Алфавит элементов SC-кода	Алфавит элементов SCg-ядра	Синтаксический тип sc.g-элемента
<i>sc-узел</i>	•	<i>sc.g-узел общего вида</i>
<i>sc-ссылка</i>		<i>sc.g-ссылка общего вида</i>
<i>sc-ребро общего вида</i>		<i>sc.g-ребро общего вида</i>
<i>sc-дуга общего вида</i>		<i>sc.g-дуга общего вида</i>
<i>sc-дуга принадлежности</i>		<i>sc.g-дуга принадлежности</i>
<i>константная позитивная стационарная sc-дуга принадлежности</i>		<i>константная позитивная стационарная sc.g-дуга принадлежности</i>

Некоторым из sc.g-элементов могут быть приписаны строковые идентификаторы, которые являются именами изображаемых sc-элементов (то как эти идентификаторы представлены в базе знаний, будет описано далее). Это нужно для того, чтобы пользователь мог читать sc.g-тексты. Указанные идентификаторы могут быть:

- глобальными (системными), которые являются абсолютно уникальными в рамках системы. В основном эти идентификаторы используются самой системой и её разработчиками;
- основными, которые являются уникальными в рамках режима диалога, ориентированного на носителя соответствующего естественного языка. Другими словами не может быть два объекта с одинаковым основным русскоязычным идентификатором* или основным англоязычным идентификатором* и т. д.;
- дополнительными, которые являются различными синонимами.

Идентификатор изображается рядом с идентифицируемым sc.g-элементом (на небольшом расстоянии). К примеру, узел с идентификатором *бинарное отношение* изображается следующим образом:

•
бинарное отношение

Рис. 6.1 - Пример изображения идентификатора

При большом количестве идентифицируемых sc.g-узлов на экране становится сложно определить к какому из них относится тот или иной идентификатор. Чтобы немного упростить эту задачу, идентификаторы sc.g-узлов могут располагаться лишь в 4 позициях относительно идентифицируемого объекта. Чтобы описать возможные позиции идентификатора, введем два прямоугольника: прямоугольник *A* - это прямоугольник, который описан вокруг идентифицируемого узла, а прямоугольник *B* - это прямоугольник, который описан вокруг идентификатора узла. Тогда расположение идентификатора относительно идентифицируемого узла может быть следующим:

- справа внизу (рекомендуемая позиция для расположения идентификаторов). При таком расположении, левый верхний угол прямоугольника Б, должен совпадать с нижним правым углом прямоугольника А;
- справа сверху (левый нижний угол прямоугольника Б совпадает с правым верхним углом прямоугольника А);
- слева сверху (правый нижний угол прямоугольника Б совпадает с левым верхним углом прямоугольника А);
- слева внизу (правый верхний угол прямоугольника Б совпадает с левым нижним углом прямоугольника А).

Более подробно возможные варианты расположения идентификаторов относительно sc.g-узлов можно посмотреть на рисунке 5.2.

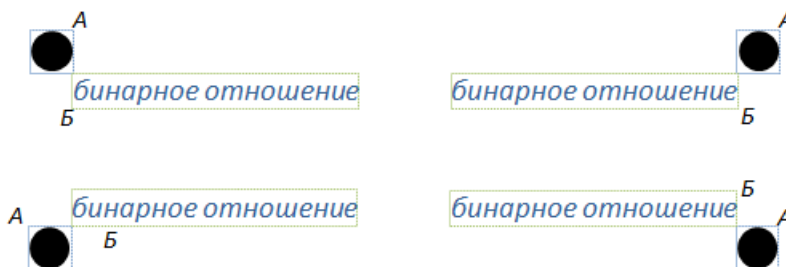


Рис 6.2 - Возможные варианты расположения идентификаторов относительно sc.g-узлов

Разрешено использовать дополнительные средства для обозначения принадлежности идентификатора некоторому sc.g-узлу. К примеру, при наведении указателя мыши на идентификатор или sc.g-узел, можно менять их цвет. Важно, чтобы эти дополнительные средства не загромождали и не усложняли, отображаемый на экране, sc.g-текст.

Рассмотрим примеры некоторых sc.g-текстов, с помощью SCg-ядра (SCg-кода 1-го уровня):

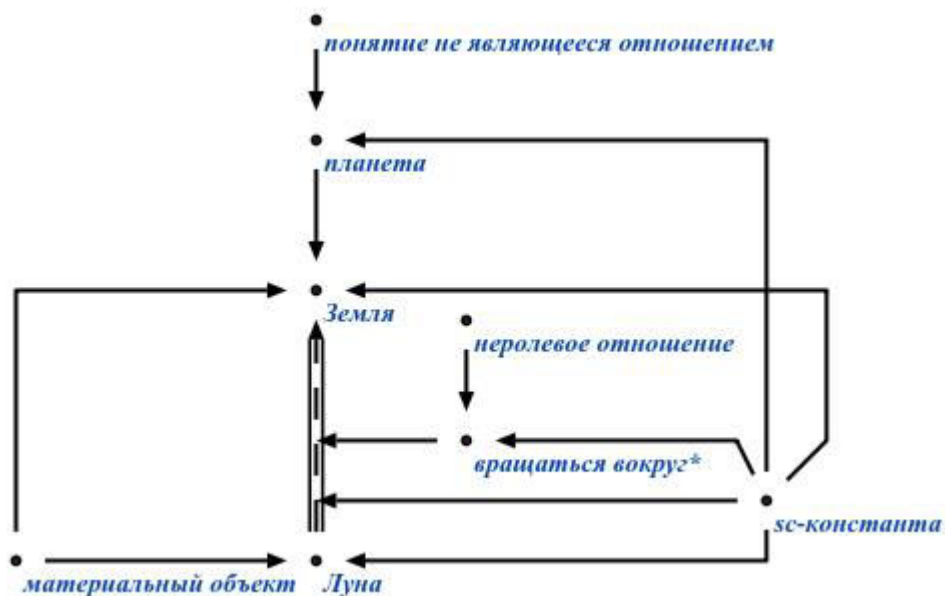


Рис 6.3 - Запись факта: “Луна вращается вокруг планеты Земля” с помощью SCg-кода 1-го уровня

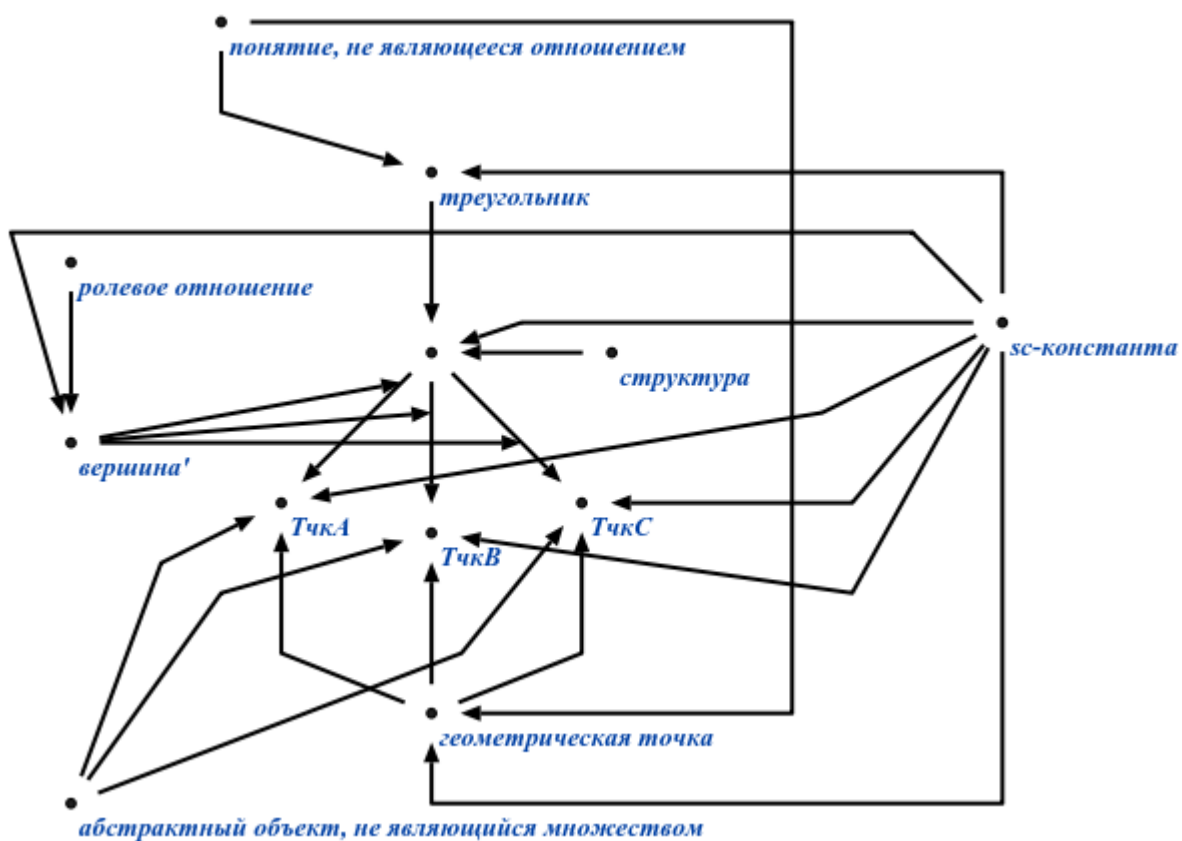


Рис 6.4 - Запись факта: “Имеется треугольник с вершинами в точках А, В и С” с помощью SCg-кода 1-го уровня

Из примеров видно, что читабельность sc.g-текстов записанных с помощью SCg-ядра не высока, отчасти это обусловлено большим количеством изображаемых sc.g-элементов. Поэтому одним из способов повышения читабельности таких текстов

является уменьшение количества изображаемых объектов. Важно, чтобы при уменьшении количества объектов тексты не теряли записанную в них информацию.

6.2.3 SCg-код 2-го уровня

Второй уровень SCg-кода направлен на уменьшение количества отображаемых на экране sc.g-элементов для повышения наглядности (читабельности) sc.g-текстов. Основная идея, которая лежит в основе SCg-кода 2-го уровня заключается в том, что изображение наиболее часто используемых дуг принадлежности sc-элементов к типовым (базовым) sc-множествам, можно заменить на некоторый, характерный для них, графический признак. Стоит отметить что графические признаки отличаются для sc.g-узлов и sc.g-дуг, поэтому их будем рассматривать по отдельности.

Множество всех sc-узлов разбивается по двум признакам. Первым признаком является константность (узел может быть элементом множества *sc-констант*, либо множества *первичных sc-переменных*). На рисунке 5.5 представлен пример записи sc.g-текста, с помощью SCg-ядра, в котором присутствуют sc-узлы являющиеся элементами множества *sc-констант* и sc-узлы, которые являются элементами множества *первичных sc-переменных*.

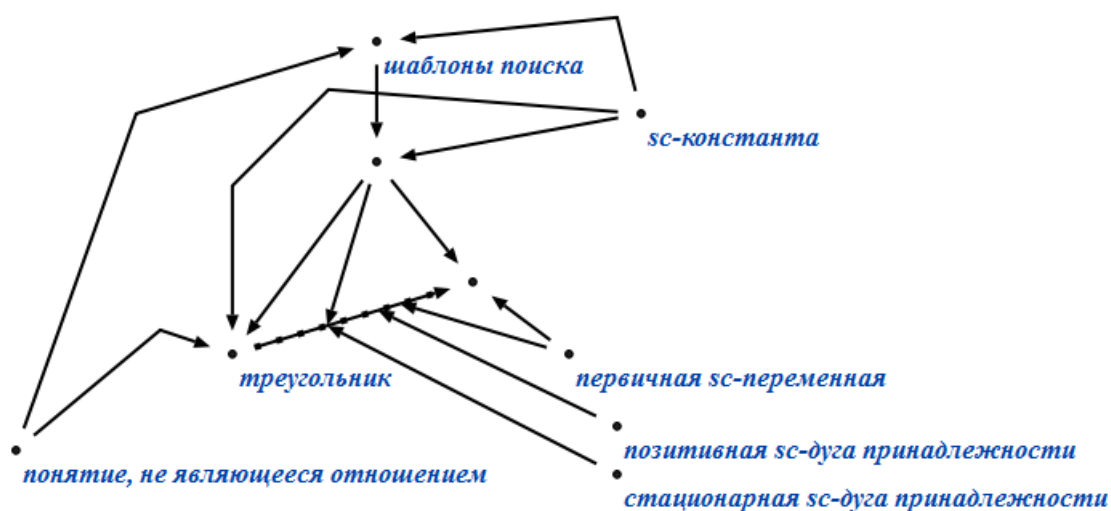


Рис 6.5 - Пример записи поискового шаблона, для поиска всех элементов множества треугольников, с помощью SCg-ядра

Вместо явного изображения, с помощью SCg-кода, sc-дуг принадлежности и sc-узлов обозначающих эти два множества, используются следующие признаки:

- sc-узлы, которые являются элементами множества *sc-констант*, изображаются в виде окружности (рекомендуемая толщина линий и габариты всех sc.g-элементов можно посмотреть в приложении №);
- sc-узлы, которые являются элементами множества *первичных sc-переменных*, изображаются в виде квадратов.

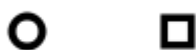


Рис 6.6. Пример изображения константного sc-узла (слева) и переменного sc-узла (справа), с помощью SCg-кода 2-го уровня

Очевидно, что даже это небольшое упрощение значительно повысило наглядность изображаемого sc.g-текста, за счет уменьшения количества изображаемых элементов.

Множество sc-узлов, кроме разбиения по признаку константности, разбивается на следующие подмножества, по структурному признаку:

- множество sc-узлов, обозначающих небинарные связи;
- множество sc-узлов, обозначающих структуры;
- множество sc-узлов, обозначающих ролевые отношения;
- множество sc-узлов, обозначающих неролевые отношения;
- множество sc-узлов, обозначающих понятия, не являющиеся отношениями;
- множество sc-узлов, обозначающих абстрактные объекты, не являющиеся множествами;
- множество sc-узлов, обозначающих материальный объект.

Принадлежность sc-узла к одному из выше перечисленных множеств изображается с помощью специальных графических примитивов, которые расположены внутри фигур изображающих константные или переменные sc-узлы. Пример изображения структурного признака представлен на рисунке 5.7.



Рис 6.7. Пример изображения константного и переменного sc-узлов, обозначающих понятия, не являющиеся отношениями, с помощью SCg-кода 2-го уровня

Используя приведенный выше подход, можно упростить изображение, представленное на рисунке 5.7, до конструкции на рисунке 5.8.

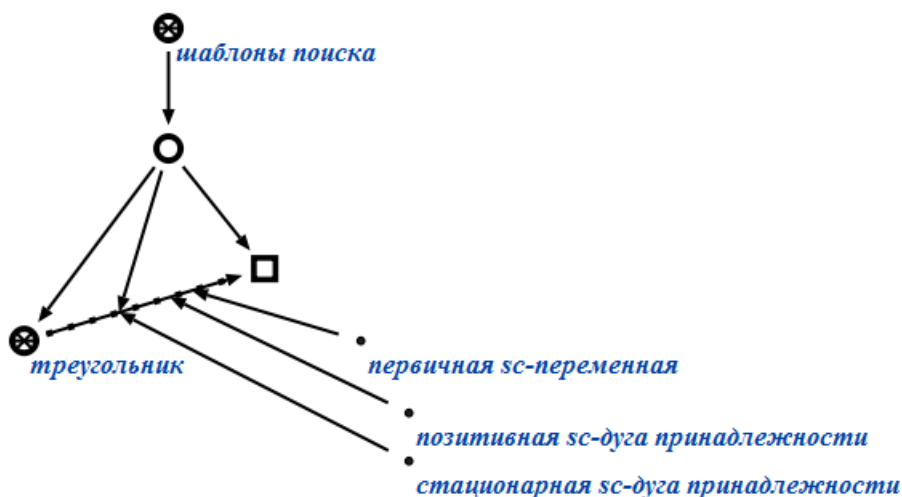


Рисунок 6.8. Пример записи поискового шаблона с упрощенным отображением sc.g-узлов

Аналогичный подход используется для отображения sc-коннекторов. SC-дуги (ребра) общего вида изображаются в виде двух параллельных линий, с достаточно большим промежутком между ними. Признак константности или переменности таких

дуг указывается с помощью заполнения этого пустого пространства между линиями. Для переменных дуг оно заполняется штриховой линией. В случае константных дуг, это пространство остается пустым (рисунок 5.9).

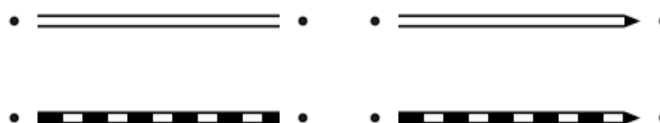



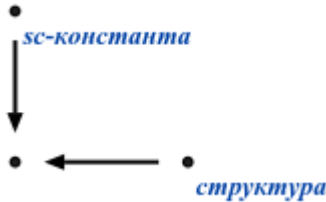

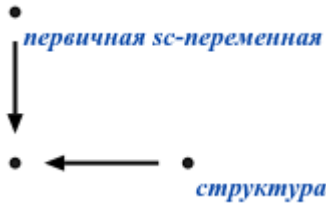

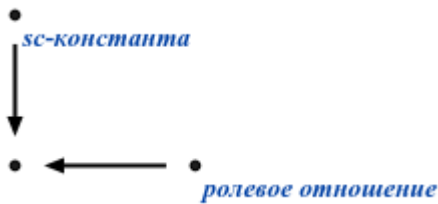

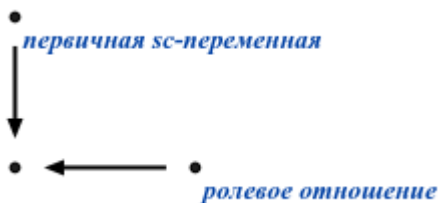






Рисунок 6.9. Пример изображения константных и переменных sc-ребер (дуг)




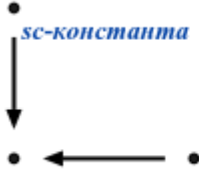



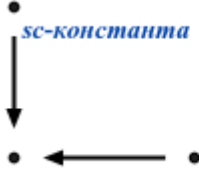



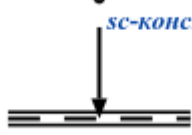

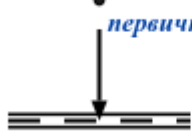

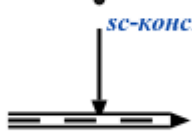
В случае с дугами принадлежности, используется тот же признак - переменные дуги принадлежности изображаются штрихованной линией. Помимо признака константности множество sc-дуг принадлежности разбивается на подмножества по следующим признакам:


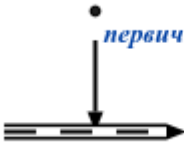

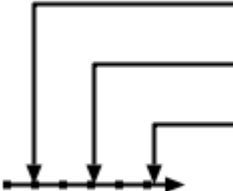

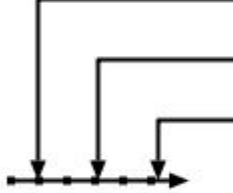

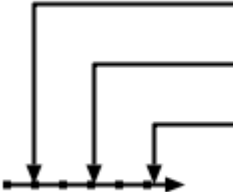

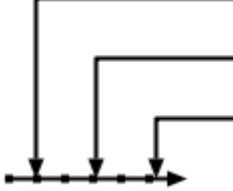
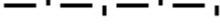
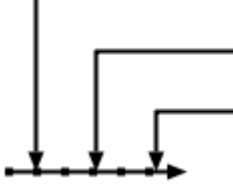

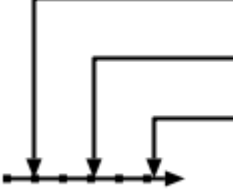
- признак позитивности. По этому признаку они разбиваются на множество позитивных, негативных и нечетких sc-дуг принадлежности;
- признак стационарности. По этому признаку они разбиваются на множество стационарных и нестационарных sc-дуг принадлежности.

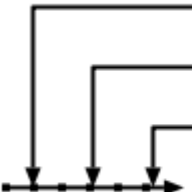
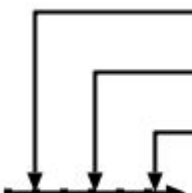
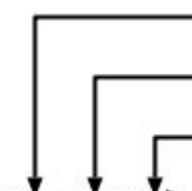
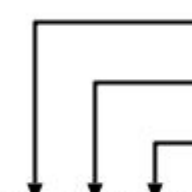
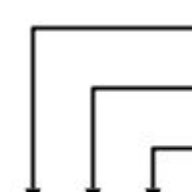
Полную таблицу правил SCg-кода второго уровня можно посмотреть ниже:

Расширение алфавита sc.g-элементов	Синтаксический тип sc.g-элемента	Представление изображаемых sc-элементов в SC-коде (точнее, в SCg-коде 1-го уровня)
○	константный sc.g-узел	
□	sc.g-узел, являющийся первичной переменной	
⊖	константный sc.g-узел обозначающий небинарную связку	
⊞	первичная sc.g-переменная, обозначающая небинарную связку	

	константный sc.g-узел, обозначающий структуру	
	первичная sc.g- переменная, обозначающая структуру	
	константный sc.g-узел, обозначающий ролевое отношение	
	первичная sc.g- переменная, обозначающая ролевое отношение	
	константный sc.g-узел, обозначающий неролевое отношение	
	первичная sc.g- переменная, обозначающая неролевое отношение	
	константный sc.g-узел, обозначающий понятие, не являющееся отношением	

	первичная sc.g-переменная, обозначающая понятие, не являющееся отношением	 <p>первичная sc-переменная</p> <p>понятие, не являющееся отношением</p>
	константный sc.g-узел, обозначающий абстрактный объект, не являющийся множеством	 <p>sc-константа</p> <p>абстрактный объект, не являющийся мно...</p>
	первичная sc.g-переменная, обозначающая абстрактный объект, не являющийся множеством	 <p>первичная sc-переменная</p> <p>абстрактный объект, не являющийся мно...</p>
	константный sc.g-узел, обозначающий материальный объект	 <p>sc-константа</p> <p>материальный объект</p>
	первичная sc.g-переменная, обозначающая материальный объект	 <p>первичная sc-переменная</p> <p>материальный объект</p>
	константное sc.g-ребро	 <p>sc-константа</p>
	первичная sc.g-переменная, обозначающая пару sc-элементов	 <p>первичная sc-переменная</p>
	константная sc.g-дуга	 <p>sc-константа</p>

	первичная sc.g- переменная, обозначающая ориентированную пару sc-элементов		первичная sc-переменная
	первичная переменная позитивная стационарная sc.g-дуга принадлежности		<ul style="list-style-type: none"> первичная sc-переменная позитивная sc-дуга принадлежности стационарная sc-дуга принадлежности
	константная негативная стационарная sc.g-дуга принадлежности		<ul style="list-style-type: none"> sc-константа негативная sc-дуга принадлежности стационарная sc-дуга принадлежности
	первичная переменная негативная стационарная sc.g-дуга принадлежности		<ul style="list-style-type: none"> первичная sc-переменная негативная sc-дуга принадлежности стационарная sc-дуга принадлежности
	константная нечеткая стационарная sc.g-дуга принадлежности		<ul style="list-style-type: none"> sc-константа нечеткая sc-дуга принадлежности стационарная sc-дуга принадлежности
	первичная переменная нечеткая стационарная sc.g-дуга принадлежности		<ul style="list-style-type: none"> первичная sc-переменная нечеткая sc-дуга принадлежности стационарная sc-дуга принадлежности
	константная позитивная нестационарная sc.g- дуга принадлежности		<ul style="list-style-type: none"> sc-константа позитивная sc-дуга принадлежности нестационарная sc-дуга принадлежности

....	первичная переменная позитивная нестационарная sc.g- дуга принадлежности	 <ul style="list-style-type: none"> первичная sc-переменная позитивная sc-дуга принадлежности нестационарная sc-дуга принадлежности
.....+.....+.....+.....+	константная негативная нестационарная sc.g- дуга принадлежности	 <ul style="list-style-type: none"> sc-константа негативная sc-дуга принадлежности нестационарная sc-дуга принадлежности
.... 	первичная переменная негативная нестационарная sc.g- дуга принадлежности	 <ul style="list-style-type: none"> первичная sc-переменная негативная sc-дуга принадлежности нестационарная sc-дуга принадлежности
.....+.....+.....+.....+	константная нечеткая нестационарная sc.g- дуга принадлежности	 <ul style="list-style-type: none"> sc-константа нечеткая sc-дуга принадлежности нестационарная sc-дуга принадлежности
.... 	первичная переменная нечеткая нестационарная sc.g- дуга принадлежности	 <ul style="list-style-type: none"> первичная sc-переменная нечеткая sc-дуга принадлежности нестационарная sc-дуга принадлежности

Рассмотрим несколько примеров представления знаний с помощью SCg-кода второго уровня. Для наглядности перед каждой записью с помощью SCg-кода 2-го уровня повторим семантически эквивалентную конструкцию записанную с помощью SCg-кода 1-го уровня (примеры приведенные выше):

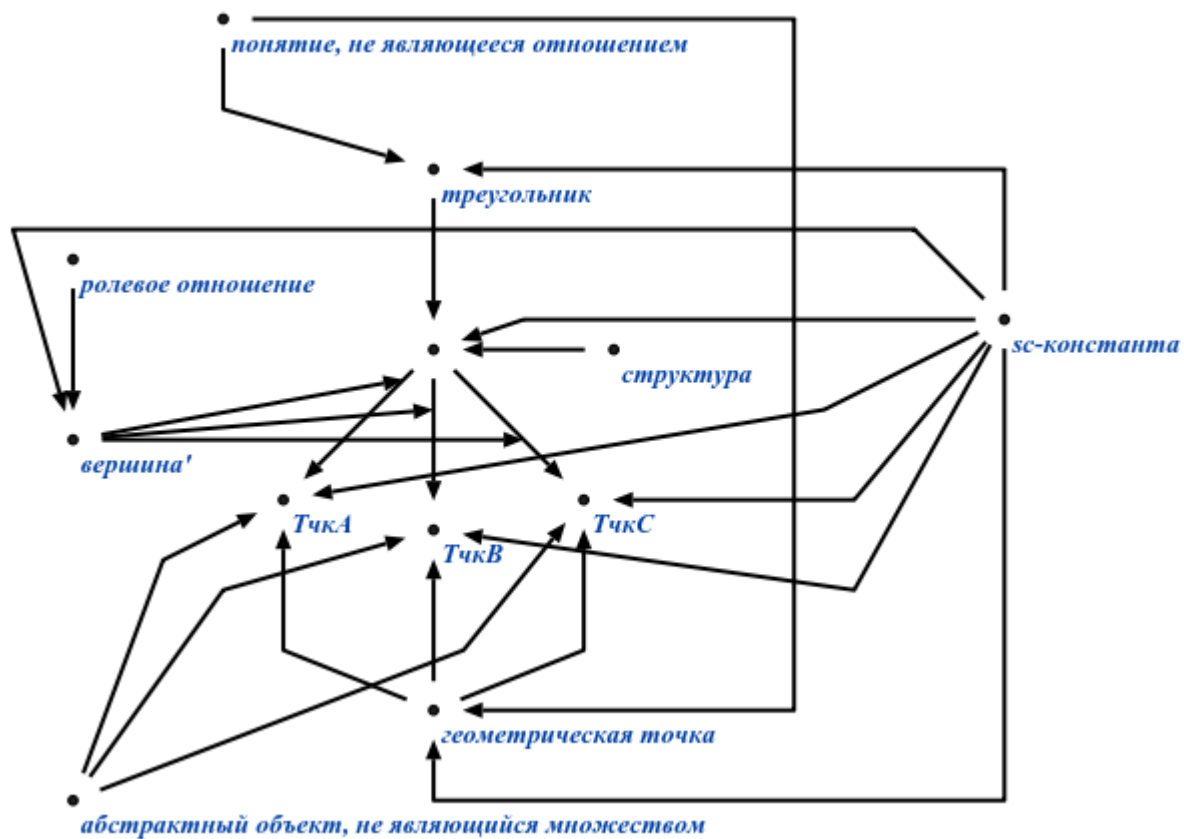


Рис.6.10 Имеется треугольник с вершинами в точках А, В и С (запись с помощью SCg-кода 1-го уровня).

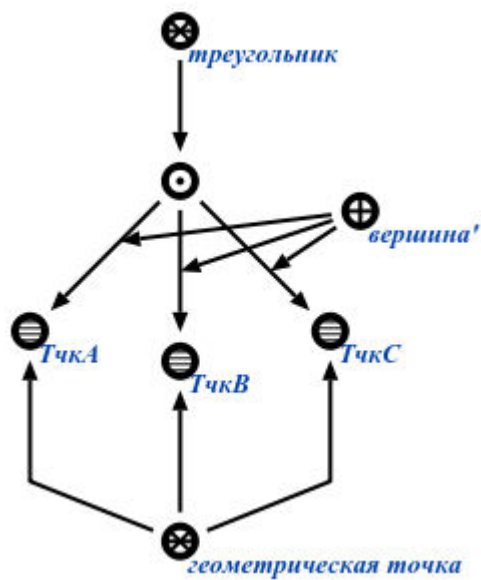


Рис. 6.11 Имеется треугольник с вершинами в точках А, В и С (запись с помощью SCg-кода 2-го уровня).

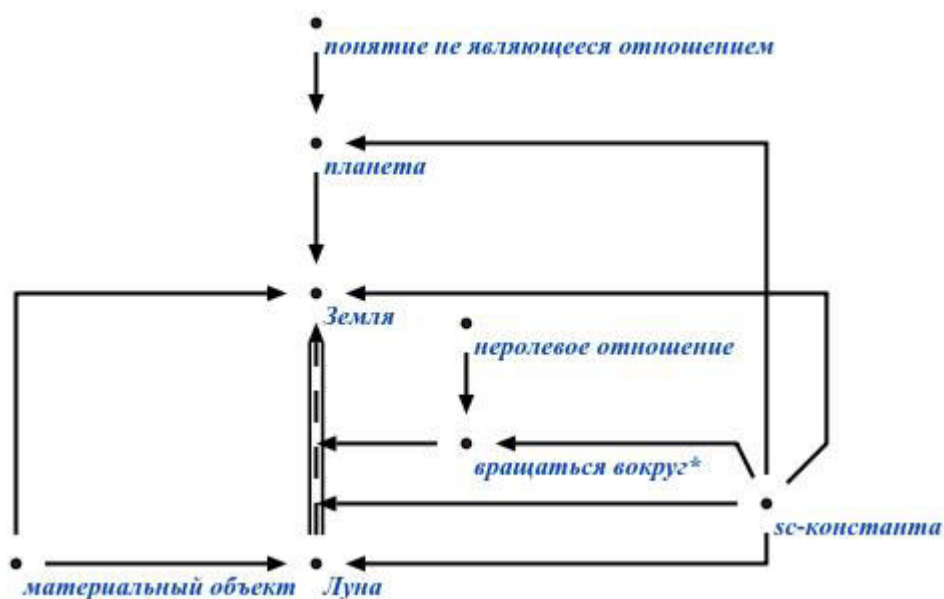


Рис. 6.12 Луна вращается вокруг планеты "Земля" (запись с помощью SCg-кода 1-го уровня).



Рис. 6.13 Луна вращается вокруг планеты "Земля" (запись с помощью SCg-кода 2-го уровня).

Из примеров видно, что SCg-код 2-го уровня значительно сокращает количество элементов присутствующих на экране. Это в свою очередь позволяет повысить читабельность sc.g-текстов.

6.2.4 SCg-код 3-го уровня

При больших объемах sc.g-текстов и второго уровня может оказаться недостаточно. Как и при переходе с 1-го уровня на 2-й уровень, повышение читабельности sc.g-текстов можно достичь еще более существенным сокращением, изображаемых на экране, элементов. Для этого введен 3-й уровень SCg-кода.

Часто возникает ситуация, когда из sc.g-узла выходит много sc.g-дуг(пар) (или же большое количество sc.g-дуг (пар) в него входит). В таком случае sc.g-текст становится сложно читать, так как вокруг этого sc.g-узла рисуется много sc.g-дуг (пар), которые могут накладываться на текстовый идентификатор. Кроме того в таком тексте средняя

длина sc.g-дуг (пар) увеличивается, что приводит к трудностям в поиске начальных и конечных элементов этих sc.g-дуг (см. рисунок ниже)

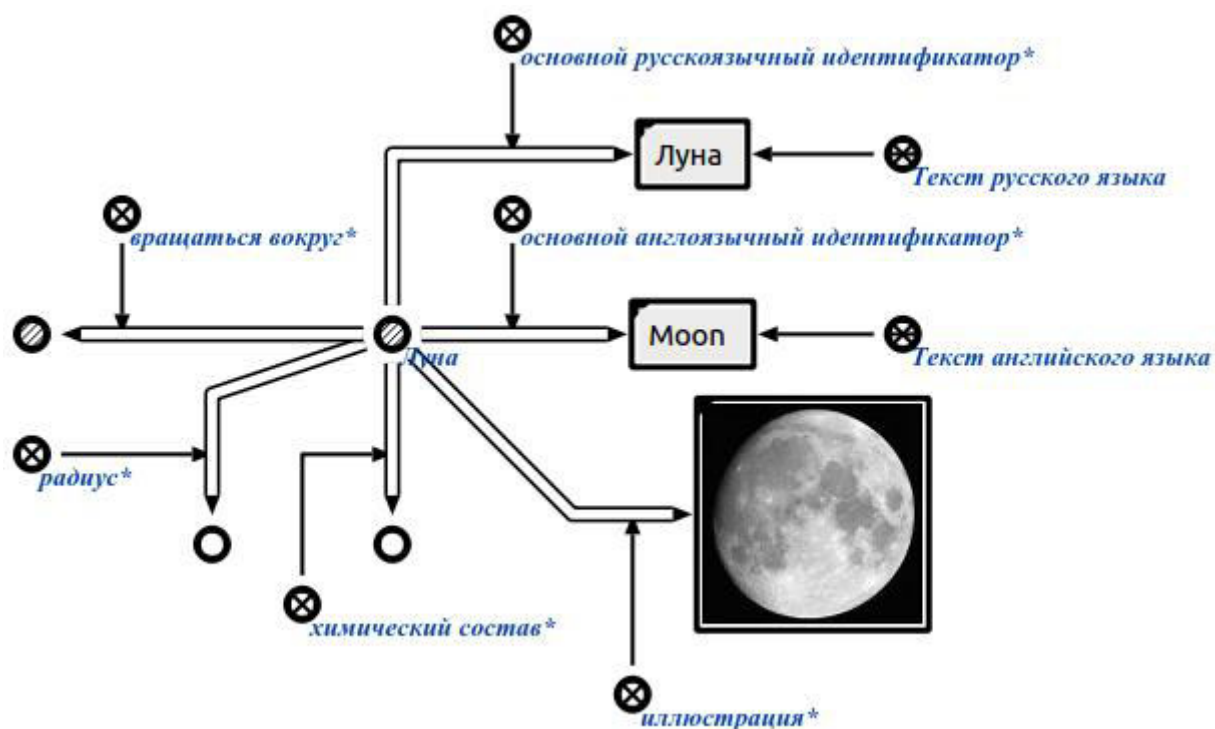


Рис.6.14 Запись некоторой информации о луне с помощью SCg-кода 2-го уровня

Очевидно, что если потребуется добавить еще информации об объекте, то единственным выходом будет его дублирование. Другими словами придется создать еще один узел с идентификатором *Луна* и связывать новые свойства sc.g-парами с ним. Решить данную проблему можно увеличив контактную площадь узла. Для этого вводится новый элемент - sc.g-шина. Пример того же sc.g-текста записанного с помощью SCg-кода 3-го уровня можно посмотреть на следующем изображении:

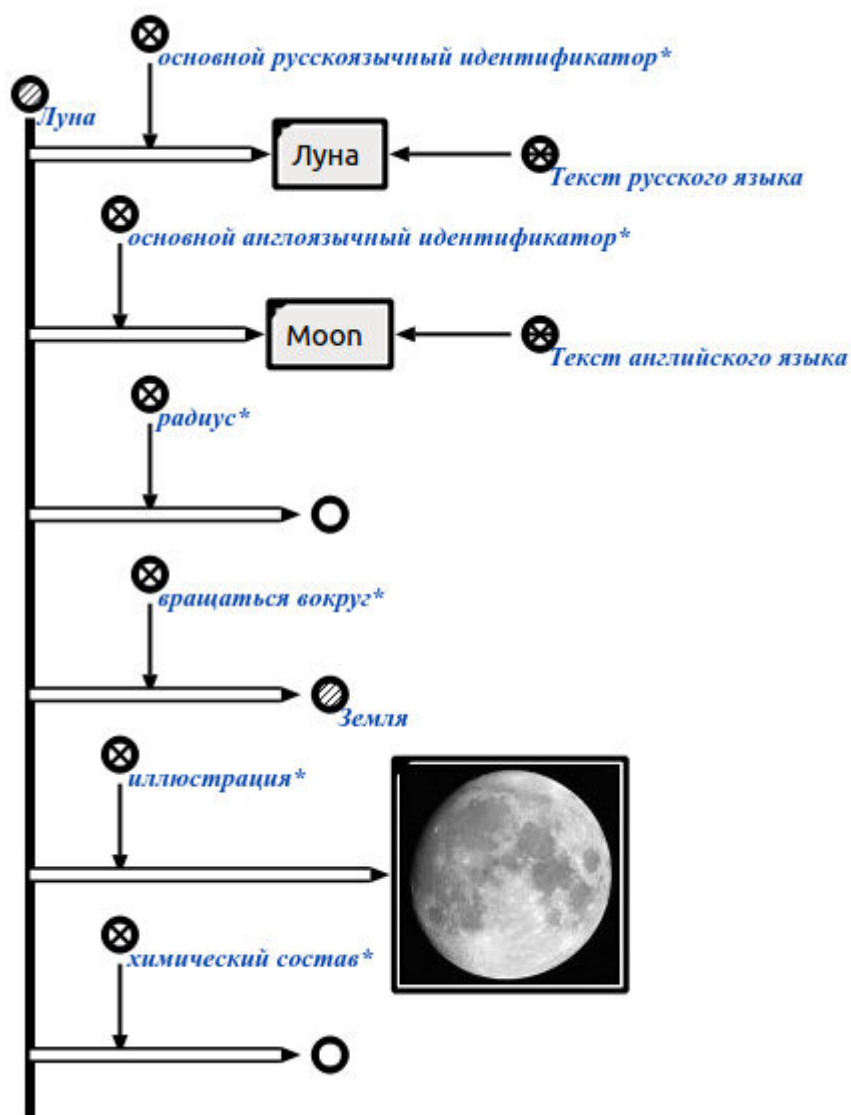

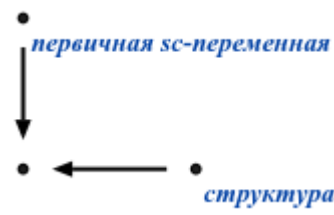






Рис. 6.15 Запись некоторой информации о луне с помощью SCg-кода 3-го уровня

Переход от элементов SCg-кода 1-го уровня к SCg-коду 3-го уровня осуществляется по правилам описанным в таблице:

Расширение алфавита sc.g-элементов	Синтаксический тип sc.g-элемента	Представление изображаемых sc- элементов в SC-коде
	sc.g-шина	
	константный sc.g-контур	

	<i>первопеременный sc.g-контур</i>	
	<i>sc.g-рамка</i>	
	<i>sc.g-узел</i>	 <i>множество</i>

6.2.5 SCg-код 4-го уровня

SCg-код является не только способом визуального представления sc-текстов, а также используется в графическом пользовательском интерфейсе для организации диалога пользователя с системой. Конечно, пользователь может производить общение с помощью трех первых уровней (пользователь рисует сообщения вручную). Но, чтобы повысить эффективность диалога, на 4-ом уровне SCg-кода вводятся sc.g-элементы управления. Нажатие на такой объект инициирует некоторое действие, которое он обозначает.

6.3 SCs-код

SCs-код - строковый (линейный) вариант представления SC-кода. Предназначен для представления sc-графов (текстов SC-кода) в виде последовательностей символов, которые могут быть отредактированы как при помощи стандартных текстовых редакторов, так и при помощи специализированного sc.s-редактора. Следовательно, одним из требований, предъявляемых к SCs-коду, помимо полноты и непротиворечивости, является возможность набора исходных текстов без помощи специализированного редактора, используя только символы стандартной клавиатуры. В отличие от SCn-кода, также являющегося текстовым вариантом отображения sc-графов, в SCs-коде форматирование не имеет значения, все предложения могут быть записаны в одну строку.

В отличие от SCg-кода, в SCs-коде нет специальных обозначений для указания структурных типов sc-узлов, однако есть изображения sc-коннекторов, соответствующие ядру и расширению SCg-кода, а также изображения sc-коннекторов специального типа, для которых нет соответствия в SCg-коде.

С формальной точки зрения SCs-код - множество sc.s-текстов. Каждый sc.s-текст представляет собой последовательность sc.s-предложений, каждое из которых оканчивается разделителем ;; (двойная точка с запятой). Множество sc.s-предложений разбивается на множество простых и множество сложных sc.s-предложений. Каждое сложное sc.s-предложение содержит в своем составе встроенные предложения, ограниченные ограничителем (*...*).

В рамках sc.s-текста любого уровня допустимо использование комментариев следующего вида:

```
// однострочный комментарий
/* многострочный комментарий */
```

В начале файла, содержащего sc.s-текст настоятельно рекомендуется указывать уровень и версию используемого SCs-кода. Для этого используются комментарии специального вида:

```
/* SCs_code<" версия ">.Level <" номер уровня "> */
```

Например:

```
/* SCs_code0.1.0.Level 6 */
```

Следует отметить, что комментарий вида `/*...*/`, использованный внутри sc.s-рамки (см. соотв. раздел) не опускается при разборе sc.s-текста, и становится частью содержимого соответствующей sc-ссылки. Например при разборе предложения `X => r1: [TEXT1/*COMMENT*/TEXT2];;`

Будет создана sc-ссылка с содержимым `TEXT1/*COMMENT*/TEXT2`, а не `TEXT1TEXT2`.

SCs-код условно разделяется на 7 уровней сложности. Все уровни равнозначны по возможностям представления знаний, однако тексты более высоких уровней описывают sc-графы более лаконично и удобно. Введение уровней призвано облегчить работу разработчиков баз знаний при наборе sc.s-текстов. При разработке баз знаний рекомендуется использовать уровни до шестого включительно.

Более подробно все уровни SCs-кода описаны на сайте <http://ims.ostis.net>

6.3 Примеры из заданной предметной области.

Пример записи в SCg-коде:

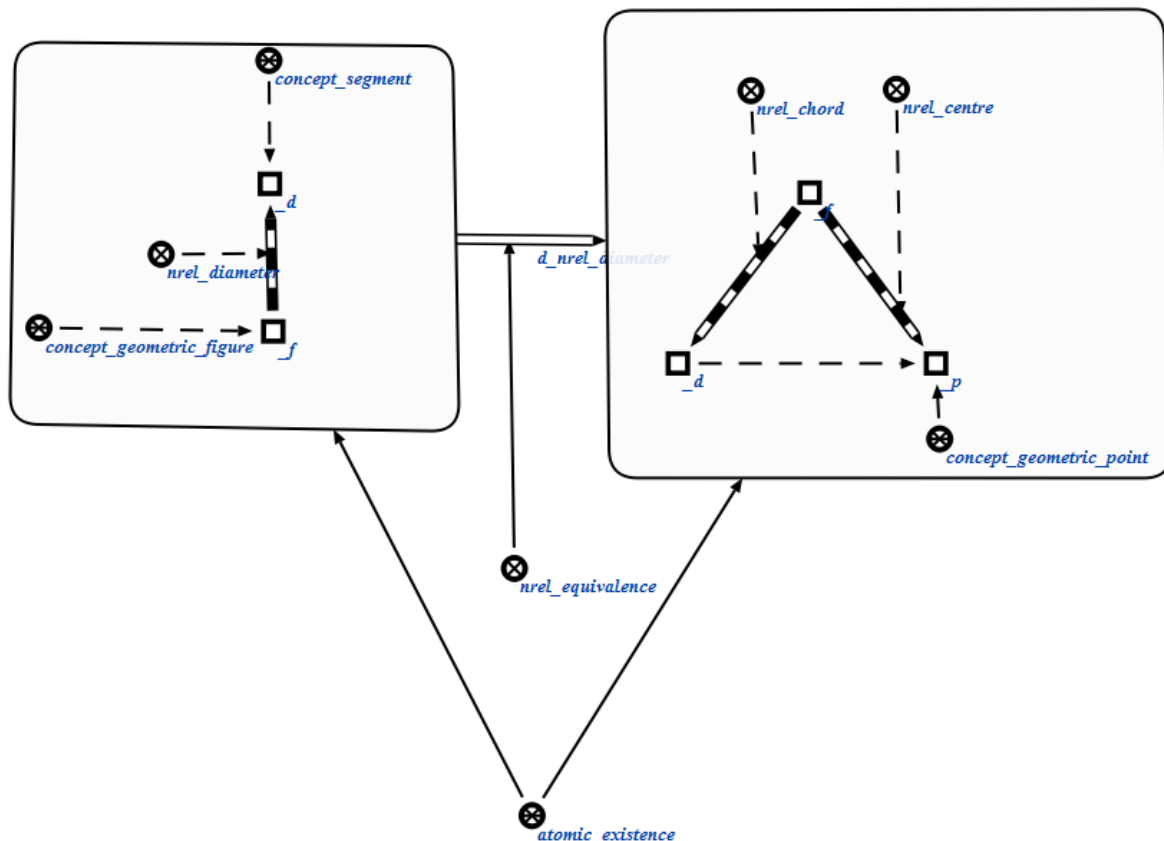


Рис. 6.16 Формальная запись определения понятия «диаметр»

Пример записи в SCs-коде:

```
concept_hexagon => nrel_main_idtf:
  [шестиугольник]
  (* <- lang_ru;; *);
  [hexagon]
  (* <- lang_en;; *);;
concept_hexagon => nrel_idtf: [Класс шестиугольников] (* <- lang_ru;; *);;
concept_hexagon => nrel_idtf: [Плоский шестиугольник] (* <- lang_ru;; *);;
concept_hexagon => nrel_idtf: [Понятие шестиугольника] (* <- lang_ru;; *);;
concept_hexagon => nrel_idtf: [Множество всевозможных шестиугольников] (* <-
lang_ru;; *);;
concept_hexagon => nrel_idtf: [Множество знаков всевозможных шестиугольников] (* <-
lang_ru;; *);;
concept_hexagon => nrel_idtf: [Многоугольник с шестью сторонами] (* <- lang_ru;; *);;
concept_hexagon <= nrel_strict_inclusion: concept_polygon;;
concept_hexagon => nrel_strict_inclusion: concept_right_hexagon;;
concept_hexagon <- rrel_key_sc_element:
  (*
    => nrel_main_idtf: [Опр.(шестиугольник)] (* <- lang_ru;; *);;
    <- sc_definition;;
    <= nrel_sc_text_translation:
      ...
      (*
        -> rrel_example:
```

[Шестиугольник- это многоугольник с шестью сторонами.]
(* <- lang_ru;; *);;

*);;

7 Добавление компонентов машины обработки знаний

7.1 Добавление команды меню

Для начала необходимо создать подпункт меню в системе, который будет соответствовать нашей операции. Для этого необходимо выполнить последовательность действий:

а) Перейти в каталог `geometry.ostis/ims.ostis.kb/ui/menu/`

б) Создать файл с именем `ui_menu_search_operation.scs` (имя для операции может быть выбрано любое).

В этом файле указывается, где будет видна операция в web-интерфейсе. Также указывается имя вопроса, который будет инициировать sc-агент. Имя вопроса мы придумываем сами. Он будет уникален для нашего агента.

```
ui_menu_search_operation<-                                     ui_user_command_class_atom;
ui_user_command_class_view_kb;;
// Указываем русский идентификатор нашей операции в меню
ui_menu_search_operation =>nrel_main_idtf: [Запроспоиска] (* <- lang_ru;; *);;
// Указываем английский идентификатор нашей операции в меню
ui_menu_search_operation =>nrel_main_idtf: [Request search] (* <- lang_en;; *);;
// Указываем шаблон нашей команды
ui_menu_search_operation =>ui_nrel_command_template:
  [*
    question_search_operation _-> ._question_search_operation_instance
    (*
      _-> ui_arg_1;;
    *);;
    ._question_search_operation_instance _<- question;;
  *];;
// Указываем текстовый шаблон команды на русском языке
ui_menu_search_operation      =>ui_nrel_command_lang_template:      [Запроспоиска:
$ui_arg_1] (* <- lang_ru;; *);;
// Указываем текстовый шаблон команды на английском языке
ui_menu_search_operation      =>ui_nrel_command_lang_template: [Request of search:
$ui_arg_1] (* <- lang_en;; *);;
```

с) В этой же папке открыть файл `ui_menu_na_view_kb_base.scs`

Внутри него необходимо добавить имя своего подпункта меню (добавленный текст выделен зелёным).

```
ui_menu_na_view_kb_base<= nrel_ui_commands_decomposition:
{
  ui_menu_view_all_output_const_pos_arc;
  ui_menu_view_all_input_const_pos_arc;
  ui_menu_view_all_output_const_pos_arc_with_rel;
  ui_menu_view_all_input_const_pos_arc_with_rel;
  ui_menu_search_operation
};;
```

Обратите внимание, что точка с запятой после последнего имени не нужна.

д) Необходимо пересобрать базу и перезапустить сервер (`./build_kb.sh`,

./run_sctp.sh, ./run_scweb).

В результате создания подпункта меню мы должны увидеть следующую картину:

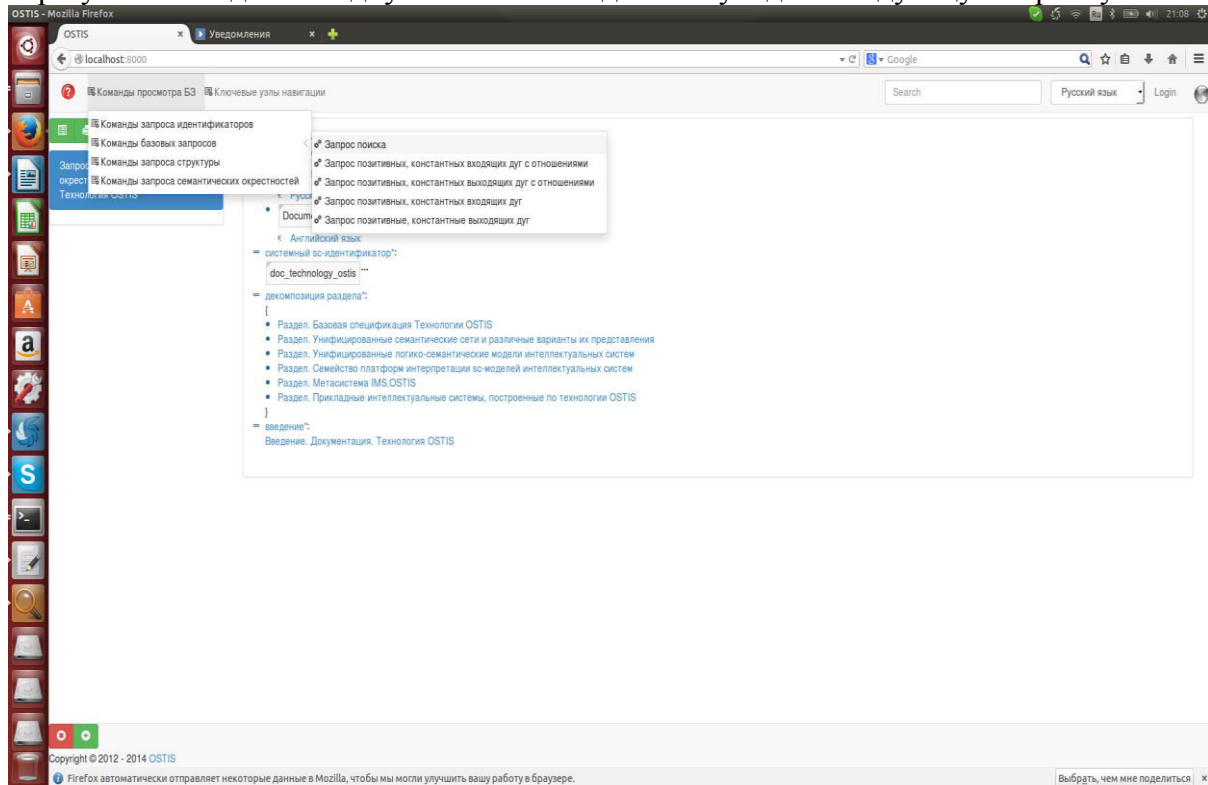


Рис. 7.1 Добавление пункта меню

7.2 Спецификация sc-агента

Для создания агента нужна агентная программа, поисковая программа и спецификация.

Под абстрактным sc-агентом понимается некоторый класс функционально эквивалентных sc-агентов, разные экземпляры которых могут быть реализованы по-разному.

Программой sc-агента является множество scr-программ, описывающих деятельность данного абстрактного sc-агента. Данное множество содержит одну агентную scr-программу и произвольное количество scr-программ, которые необходимы для выполнения указанной агентной scr-программы.

Каждый абстрактный sc-агент имеет соответствующую ему спецификацию. Спецификация агента – это его формальное описание, которое включает указание ключевых sc-элементов этого sc-агента, описание условий инициирования данного sc-агента, однозначно понимаемое описание деятельности данного sc-агента.

В спецификацию каждого абстрактного sc-агента входит:

- указание ключевых sc-элементов этого sc-агента, т.е. тех sc-элементов, хранимых в sc-памяти, которые для данного sc-агента являются «точками опоры»
- формальное описание условий инициирования данного sc-агента, т.е. тех ситуаций в sc-памяти, которые инициируют деятельность данного sc-агента.
- формальное описание первичного условия инициирования данного sc-агента, т.е. такой ситуации в sc-памяти, которая побуждает sc-агента перейти в активное состояние и начать проверку наличия своего полного условия инициирования.

- строгое, полное, однозначно понимаемое описание деятельности данного sc-агента, оформленное при помощи каких-либо понятных, общепринятых средств, не требующих специального изучения, например на естественном языке.

- описание результатов выполнения данного sc-агента.

Для спецификации агентанеобходимо выполнить последовательность действий:

а) Перейти в

geometry.ostis/ims.ostis.kb/documentation/section_metasystem_ostis/section_ims_ostis_sub_systems_for_components_development/section_ims_ostis_subsystem_for_kpm_development_support/section_reusable_kpm_component_library/

б) Открыть section_kpm_agent_library_of_information_search.scsi

с) Дописать в самый конец

// Объявляем sc-агент поисковой операции

sc_agent_of_search_operation

// Указываем множество идентификаторов агента

=>nrel_main_idtf:

// lang_ru – атрибут идентификатора на русском языке

[sc-агентпоиска] (* <- lang_ru;; *);

// lang_en – атрибут идентификатора на английском языке

[sc-agent of search] (* <- lang_en;; *);

// Указываем, что наш агент принадлежит классу абстрактныхsc-агентов

<- abstract_sc_agent;

// Указываем начальное условие для нашей операции

=>nrel_primary_initiation_condition: (sc_event_add_output_arc =>question_initiated);

// Указываем начальное условие и результат для нашей операции

=>nrel_initiation_condition_and_result:

(..sc_agent_of_search_operation_initiation_condition

=>

..sc_agent_of_search_operation_result);

<= nrel_sc_agent_key_sc_elements:

{

question_initiated;

question;

question_search_operation

};// Описание деятельности sc-агента

<- key_sc_element:

...

(*

<- sc_description_of_sc_agent_behavior;;

<= nrel_sc_text_translation:

...

(*

->nrel_example:

[Задачей sc-агента поиска является поиск.];;

*);;

*/);// описание первичного условия инициирования sc-агента

=>nrel_inclusion: ...

(*

<- platform_independent_abstract_sc_agent;;


```

        <= nrel_sc_agent_program:
        {
            agent_proc_search_operation;
            proc_search_operation
        };
    ->sc_agent_of_search_operation_scp (* <- active_sc_agent;; *);;
*);;

// условия иницирования агента
..sc_agent_of_search_operation_initiation_condition
= [*
    question_search_operation _-> .._question;;
    question_initiated _-> .._question;;
    question _-> .._question;;
    .._question _-> .._parameter;;
*];;

// описание результатов выполнения sc-агента
..sc_agent_of_search_operation_result
= [*
    question_search_operation _-> .._question;;
    question_finished _-> .._question;;
    question _-> .._question;;
    .._question _=>nrel_answer:: .._answer;;
    .._question _-> .._parameter;;
*];;

```

7.3 Реализация sc-агента на платформенно-независимом уровне (язык SCP)

Вначале создадим процедуру, которая будет выполнять поиск выходящих дуг. Для этого необходимо выполнить следующую последовательность действий:

а) Перейти в `geometry.ostis/ims.ostis.kb/`

б) Создать папку в которой будут храниться наши агенты и процедуры. Например, `examples`. Тогда путь будем следующим “`~/ostis/kb.sources/examples/`”. В этой папке необходимо создать файл `proc_search_operation.scs`

В качестве его содержимого необходимо установить следующий код:

```

// Объявляем нашу процедуру как scp-программу
scp_program ->proc_search_operation (*
    // Устанавливаем связь между процедурой и множеством её параметров
    ->rrel_params: .proc_search_operation_params (*
        -> rrel_1: rrel_in: _parameter;;
        ->rrel_2: rrel_in: _answer;;
    *);;
    // Устанавливаем связь между процедурой и множеством её операторов
    ->rrel_operators: .proc_search_operation_operator_set (*
        // Данный оператор принадлежит множеству операторов
        // с атрибутом rrel_init, это означает, что он будет выполняться первым
        ->rrel_init: .proc_search_operation_operator1 (*
            <- genEl;;

```

```
-> rrel_1: rrel_assign: rrel_const: rrel_node: rrel_scp_var: _set_elem;;
```

```
=>nrel_goto: .proc_search_operation_operator2;;  
*);;
```

```
// Добавим параметр операции во множество, содержащее конструкцию
```

ответа

```
-> .proc_search_operation_operator2 (*  
<- genElStr3;;  
-> rrel_1: rrel_fixed: rrel_scp_var: _answer;;  
-> rrel_2: rrel_assign: rrel_pos_const_perm: rrel_scp_var: _arc1;;  
-> rrel_3: rrel_fixed: rrel_scp_var: _parameter;;
```

```
=>nrel_goto: .proc_search_operation_operator3;;  
*);;
```

```
// Добавим все дуги, выходящие из узла параметра, во множество
```

_set_elem

```
-> .proc_search_operation_operator3 (*  
<- searchSetStr3;;  
-> rrel_1: rrel_fixed: rrel_scp_var: _parameter;;  
-> rrel_2: rrel_assign: rrel_pos_const_perm: rrel_scp_var: _arc1;;  
-> rrel_3: rrel_assign: rrel_scp_var: _elem;;
```

```
-> rrel_set_2: rrel_fixed: rrel_scp_var: _set_elem;;  
-> rrel_set_3: rrel_fixed: rrel_scp_var: _set_elem;;
```

```
=>nrel_then: .proc_search_operation_operator4;;  
=>nrel_else: .proc_search_operation_operator_return;;  
*);;
```

```
// Операторы с четвертого по седьмой представляют собой цикл
```

```
-> .proc_search_operation_operator4 (*  
<- searchElStr3;;  
-> rrel_1: rrel_fixed: rrel_scp_var: _set_elem;;  
-> rrel_2: rrel_assign: rrel_pos_const_perm: rrel_scp_var: _arc1;;  
-> rrel_3: rrel_assign: rrel_scp_var: _curr_elem;;
```

```
=>nrel_then: .proc_search_operation_operator5;;
```

```
// Если заданная конструкция не найдена, то выйти из цикла
```

```
=>nrel_else: .proc_search_operation_operator8;;  
*);;
```

```
-> .proc_search_operation_operator5 (*  
<- eraseEl;;  
-> rrel_1: rrel_fixed: rrel_erase: rrel_scp_var: _arc1;;
```

```
=>nrel_goto: .proc_search_operation_operator6;;  
*);;
```

к

началу

```
// Проверяем, не входит ли найденный элемент во множество системных
// элементов
-> .proc_search_operation_operator6 (*
<- searchElStr3;;
  -> rrel_1: rrel_fixed: rrel_scp_const: system_element;;
  -> rrel_2: rrel_assign: rrel_pos_const_perm: rrel_scp_var: _arc1;;
  -> rrel_3: rrel_fixed: rrel_scp_var: _curr_elem;;

// Если элемент входит во множество системных элементов, то вернуться

// началу цикла
=>nrel_then: .proc_search_operation_operator4;;

=>nrel_else: .proc_search_operation_operator6_1;;
*);;

// Проверяем, есть ли в узле с конструкцией ответа найденный элемент
-> .proc_search_operation_operator6_1 (*
<- searchElStr3;;
  -> rrel_1: rrel_fixed: rrel_scp_var: _answer;;
  -> rrel_2: rrel_assign: rrel_pos_const_perm: rrel_scp_var: _arc1;;
  -> rrel_3: rrel_fixed: rrel_scp_var: _curr_elem;;

// Если элемент входит в узел с конструкцией ответа, то вернуться к
// началу цикла
=>nrel_then: .proc_search_operation_operator4;;

=>nrel_else: .proc_search_operation_operator7;;
*);;

// Добавляет элемент в ответ и возвращается к началу цикла
-> .proc_search_operation_operator7 (*
<- genElStr3;;
-> rrel_1: rrel_fixed: rrel_scp_var: _answer;;
  -> rrel_2: rrel_assign: rrel_pos_const_perm: rrel_scp_var: _arc1;;
  -> rrel_3: rrel_fixed: rrel_scp_var: _curr_elem;;
  =>nrel_goto: .proc_search_operation_operator4;;
*);;

// Переходим к оператору очистки памяти
=>nrel_goto: .proc_search_operation_operator8;;
*);;

// Очистка памяти, не забываем про атрибут rrel_erase при удалении элемента
-> .proc_search_operation_operator8 (*
<- eraseEl;;
  -> rrel_1: rrel_fixed: rrel_erase: rrel_scp_var: _set_elem;;
// Переходим к оператору завершения операции
=>nrel_goto: .proc_search_operation_operator_return;;
*);;
```

```

        // Оператор завершения операции
        -> .proc_search_operation_operator_return (*
        <- return;;
        *);;
    *);;
*);;

```

Теперь необходимо создать агент, который будет вызывать созданную нами процедуру.

Агентные scp-программы представляют собой частный случай scp-программ, однако заслуживают отдельного рассмотрения, поскольку используются наиболее часто. Scp-программы данного класса представляют собой реализации программ агентов обработки знаний и имеют жестко фиксированную структуру множества параметров. Множество параметров в данном случае состоит из двух in-параметров.

Для создания агента необходимо выполнить следующие шаги:

- a) Перейти в следующий каталог: ~/ostis/kb.sources/examples/
- b) Создать файл agent_proc_search_operation.scs

В качестве его содержимого необходимо установить следующий код:

```

agent_proc_search_operation
// Множество идентификаторов
=>nrel_main_idtf:
    [агентная scp-программа поиска] (* <- lang_ru;; *);
    [agentscp-program of search] (* <- lang_en;; *);
    <- agent_scp_program;;
// Указываем, что операция поиска агента является scp-программой
scp_program -> agent_proc_search_operation (*
    // Множество параметров агентной операции
    ->rrel_params: .agent_proc_search_operation_params (*
        -> rrel_1: rrel_in: _event;;
        -> rrel_2: rrel_in: _input_arc;;
    *);;
    // Множество операторов агентной операции
    ->rrel_operators: .agent_proc_search_operation_operator_set (*
        // Первый исполняемый оператор операции
        ->rrel_init: .agent_proc_search_operation_operator1 (*
            <- searchElStr3;;
            -> rrel_1: rrel_assign: rrel_scp_var: _temp;;
            -> rrel_2: rrel_fixed: rrel_scp_var: _input_arc;;
            -> rrel_3: rrel_assign: rrel_scp_var: _quest;;
            =>nrel_goto: .agent_proc_search_operation_operator2;;
        *);;
        // В rrel_1 устанавливаем имя придуманного нами вопроса (см. пункт 2.1).
        // Агент будет просыпаться, когда обнаружит наш вопрос.
        -> .agent_proc_search_operation_operator2 (*
            <- searchElStr3;;
            ->rrel_1: rrel_fixed: rrel_scp_const: question_search_operation;;
            -> rrel_2: rrel_assign: rrel_pos_const_perm: rrel_scp_var: _arc;;
            -> rrel_3: rrel_fixed: rrel_scp_var: _quest;;

```

```

=>nrel_then: .agent_proc_search_operation_operator3;;
=>nrel_else: .agent_proc_search_operation_operator_return;;
*);;
// Найдём параметры операции – их подал на вход сам пользователь
-> .agent_proc_search_operation_operator3 (*
<- searchElStr3;;
-> rrel_1: rrel_fixed: rrel_scp_var: _quest;;
-> rrel_2: rrel_assign: rrel_pos_const_perm: rrel_scp_var: _arc;;
-> rrel_3: rrel_assign: rrel_scp_var: _param;;

=>nrel_then: .agent_proc_search_operation_operator4;;
=>nrel_else: .agent_proc_search_operation_operator_return;;
*);;

// Сгенерировать узел, в который мы поместим конструкцию ответа
-> .agent_proc_search_operation_operator4 (*
<- genEl;;
-> rrel_1: rrel_assign: rrel_const: rrel_node: rrel_scp_var: _answer;;

=>nrel_goto: .agent_proc_search_operation_operator5;;
*);;

-> .agent_proc_search_operation_operator5 (*
<- call;; // В первом параметре содержится название вызванной процедуры
-> rrel_1: rrel_fixed: rrel_scp_const: proc_search_operation;;
-> rrel_2: rrel_fixed: rrel_scp_const: // здесь описаны параметры, подаваемые
на
// вход процедуры
.agent_proc_search_operation_operator5_params (*
-> rrel_1: rrel_fixed: rrel_scp_var: _param;;
-> rrel_2: rrel_fixed: rrel_scp_var: _answer;;
*);;

// Знак scp-процесса, который может быть использован для того, чтобы
// дождаться завершения созданного scp-процесса
-> rrel_3: rrel_assign: rrel_scp_var: _descr;;

=>nrel_goto: .agent_proc_search_operation_operator6;;
*);;

// scp-оператор ожидания завершения выполнения программы
-> .agent_proc_search_operation_operator6 (*
<- waitReturn;;
-> rrel_1: rrel_fixed: rrel_scp_var: _descr;;

=>nrel_goto: .agent_proc_search_operation_operator_gen_answer;;
*);;

// Добавляем узел конструкции ответа
-> .agent_proc_search_operation_operator_gen_answer (*
<- genElStr5;;

```

```

-> rrel_1: rrel_fixed: rrel_scp_var: _quest;;
-> rrel_2: rrel_assign: rrel_const: rrel_common: rrel_scp_var: _arc;;
-> rrel_3: rrel_fixed: rrel_scp_var: _answer;;
-> rrel_4: rrel_assign: rrel_pos_const_perm: rrel_scp_var: _arc2;;
-> rrel_5: rrel_fixed: rrel_scp_const: nrel_answer;;

=>nrel_goto: .agent_proc_search_operation_operator_return;;
*);;

// Оператор завершения агентной операции
-> .agent_proc_search_operation_operator_return (*
<- return;;
*);;
*);;
*);;

```

7.4 Язык SCP

Язык SCP – это графовый язык процедурного программирования, предназначенный для эффективной обработки однородных семантических сетей с теоретико-множественной интерпретацией, закодированных с помощью SC-кода. Язык SCP является языком параллельного асинхронного программирования.

Языком представления данных для текстов языка SCP (scp-программ) является SC-код и, соответственно, любые варианты его представления. Язык SCP сам построен на основе SC-кода, вследствие чего scp-программы сами по себе могут входить в состав данных scp-программ, в т.ч. по отношению к самим себе. Таким образом, язык SCP предоставляет возможность построения реконфигурируемых программ. Однако для обеспечения возможности реконфигурирования программы непосредственно в процессе ее интерпретации необходимо на уровне интерпретатора языка SCP (scp-интерпретатора) обеспечить уникальность каждой исполняемой копии исходной программы.

Язык SCP рассматривается как ассемблер для графодинамического компьютера, ориентированного на хранение и обработку семантических сетей.

Более подробную информацию о языке SCP можно найти на сайте <http://ims.ostis.net> в разделе «Раздел. Синтаксис, денотационная семантика и операционная семантика базового языка программирования, ориентированного на обработку sc-моделей баз знаний»

8 Добавление компонентов пользовательского интерфейса

Клиентская часть ИСС состоит из ядра (набора базовых компонентов) и набора сторонних компонентов. Базовые компоненты обеспечивают интерфейс пользователя с системой — это компоненты главного меню, истории диалога и панели аргументов, а также включают в себя компоненты отображения `sc.g` и `sc.n`-текстов, так как `sc.g` и `sc.n`-представление являются наиболее полными и универсальными. Данный набор компонентов может быть дополнен компонентами отображения, которые обеспечивают вывод содержимого `sc`-ссылок. В общем случае дополнительный компонент отображения расширяет возможности интерфейса и зависит от предметной области, а потому не является элементом ядра. Процесс расширения пользовательского интерфейса состоит из разработки компонента отображения и интеграции его в систему.

8.1 Разработка компонента

Основная задача компонента - отображение содержимого `sc`-ссылок в требуемом виде. Поэтапно процесс разработки можно разделить на несколько частей. Рассмотрим их на примере редактора геометрических чертежей:

- определение набора сущностей, которые необходимо отобразить пользователю. Для предметной области «геометрия» это могут быть такие объекты, как точка, прямая, отрезок, длина, окружность;
- определение набора отношений, заданных на объектах предметной области, и правил их отображения в разрабатываемом компоненте, например, отношение длины отрезка;
- построение однозначного соответствия между `sc`-представлением и изображением, которое должно быть построено разрабатываемым компонентом.

С точки зрения технической реализации компонент просмотра является обособленным объектом, например объектом `javascript`, который взаимодействует с ядром через объект-посредник, любые данные, которые компонент может отобразить, он может получить либо от ядра, либо непосредственно от пользователя (в случае, если компонент поддерживает редактирование). В компоненте можно выделить модель представляемых данных, которая позволит абстрагироваться от данных, представленных в базе знаний, например, для отрисовки на экране точки нам необходимо знать ее координаты в пикселях, тогда как в базе знаний эта информация не нужна и не хранится. Также в компоненте должны быть определены функции трансляции данных из `sc`-представления в модель и обратно.

Для того, чтобы иметь возможность задать вопрос к какому-либо объекту, представленному в компоненте, необходимо, чтобы компоненту был известен `sc`-адрес отображения этого объекта в памяти. В случае, если компонент является только просмотрщиком, любая представленная информация получена из базы знаний и все `sc`-адреса известны. Если компонент поддерживает редактирование, то перед тем, как задать вопрос к новому элементу, его необходимо протранслировать в память. По завершении трансляции компонент может получить `sc`-адрес транслированного

объекта. Этот адрес может быть добавлен в качестве атрибута объекта в просмотрщике, после чего из ядра интерфейса можно будет указывать объект в качестве параметра запроса.

Процесс работы компонента выглядит следующим образом: при инициализации компонент ожидает данные от ядра интерфейса или от пользователя (если компонент является редактором), на основании набора правил он транслирует приходящие ему данные и отображает их. Для каждого элемента, который представлен в памяти и может быть указан в качестве аргумента поисковой операции, компонент выставляет атрибут с sc-адресом или обеспечивает правильное получение этого адреса по событию «получение адреса аргумента», инициированному из ядра. В случае, если приходит уведомление о том, что отображаемые данные изменились, компонент выполняет трансляцию новых данных или удаление старых, обновляет модель и перерисовывает содержимое.

8.2 Интеграция компонента в систему

Все компоненты просмотра содержимого sc-ссылок взаимодействуют с ядром пользовательского интерфейса через посредника - sandbox. Это позволяет при сильном изменении ядра системы и его внутреннего устройства обеспечить минимальное изменение компонентов, которые его расширяют.

Примером таких компонентов служат: просмотрщик sc.g-текстов, просмотрщик sc.n-текстов и т. д. Взаимодействие компонентов с ядром системы показано на рисунке 1.

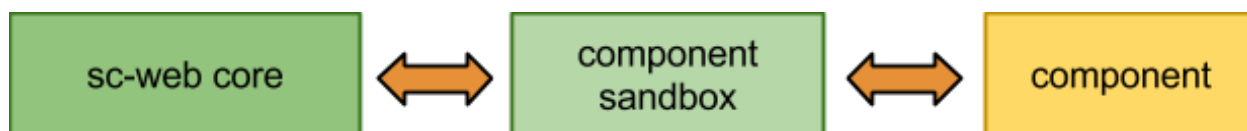


Рис 8.1. Взаимодействие компонента с ядром системы.

Рассмотрим подключение компонента просмотра на наиболее простом примере - примере просмотрщика sc-ссылок с html содержимым. В нашем случае компонент просмотра - это некоторый объект JavaScript, который реализует логику отображения содержимого:

```
var HtmlViewer = function(sandbox) {  
}
```

Единственный параметр у данной функции - это объект **sandbox**, через который наш компонент будет взаимодействовать с ядром ПИ. На старте ядра все компоненты должны быть зарегистрированы и для этого необходимо добавить описание компонента и его регистрацию в ядре ПИ:

```
HtmlComponent = {  
  formats: ['format_html'],
```



```

factory: function(sandbox) {
    return new HtmlViewer(sandbox);
}
};
SCWeb.core.ComponentManager.appendComponentInitialize(HtmlComponent);

```

где *HtmlComponent* - это объект описывающий регистрируемый компонент. Его поле *formats* - это список системных идентификаторов форматов sc-ссылок, поддерживаемых для отображения компонентом; *factory* - это функция, которая получает в качестве параметра **sandbox** от ядра ПИ и возвращает созданный экземпляр компонента. Последняя строка, регистрирует компонент в ядре системы.

Основная задача компонента - это отображение содержимого sc-ссылок, чтобы реализовать это, необходимо в **sandbox** присвоить функцию полю *eventDataAppend*. Эта функция, должна принимать лишь один параметр *data* - данные для отображения.

```

var HtmlViewer = function(sandbox) {
    this.receiveData = function(data) {
    };

    this.sandbox.eventDataAppend = $.proxy(this.receiveData, this);

    this.sandbox.updateContent();
};

```

В поле **sandbox.container** хранится id элемента страницы, в котором должно осуществляться отображение для указанного содержимого. В простейшем случае отображение html содержимого может выглядеть так:

```

var HtmlViewer = function(sandbox) {
    this.container = '#' + sandbox.container;
    this.receiveData = function(data) {
        $(this.container).html(data);
    };

    this.sandbox.eventDataAppend = $.proxy(this.receiveData, this);

    this.sandbox.updateContent();
};

```

Чтобы компонент загрузился, его необходимо добавить в [шаблон](#). Далее он будет зарегистрирован и доступен ядру ПИ. Когда его применять, будет решаться ядром, в зависимости от типа содержимого sc-ссылки.

В базе знаний системы необходимо описать наш новый формат, чтобы система понимала каким образом с ним необходимо работать:

```

format_html
<- sc_node_not_relation;

```

```

<- format;
=> nrel_main_idtf:
    [формат html]
    (*
    <- lang_ru;;
    *);
=> nrel_main_idtf:
    [format html]
    (*
    <- lang_en;;
    *);
=> nrel_mimetype:
    [text/html];;

```

где *nrel_mimetype* - отношение связывающее описываемый формат с его [mime типом](#).

Содержимое и формат связываются с помощью отношения *nrel_format*:

```
format_html <= nrel_format: [<h1>Hello World </h1>];;
```

При сборке БЗ из исходных текстов действует следующее правило: если содержимое указано как ссылка на файл, то форматом является *format_[ext]*, где *ext* - это расширение файла. К примеру, для:

```
x = "index.html";;
```

автоматически будет сгенерирована следующая конструкция:

```
x = "index.html";;
```

```
x => nrel_format: format_html;
```

Для конструкции:

```
y = "test.png";;
```

будет:

```
y = "test.png";;
```

```
y => nrel_format: format_png;
```