Документация программного варианта реализации платформы интерпретации sc-моделей компьютерных систем

Программный вариант реализации платформы интерпретации sc-моделей компьютерных систем

⇒ *принципы реализации**:

[Поскольку sc-тексты представляют собой семантические сети, то есть, по сути, графовые конструкции определенного вида, то на нижнем уровне задача разработки программного варианта реализации платформы интерпретации sc-моделей сводится к разработке средств хранения и обработки таких графовых конструкций.

К настоящему времени разработано большое количество простейших моделей представления графовых конструкций в линейной памяти, таких как матрицы смежности, списки смежности и другие (**Diskrete_Math**). Однако, при разработке сложных систем как правило приходится использовать более эффективные модели, как с точки зрения объема информации, требуемого для представления, так и с точки зрения эффективности обработки графовых конструкций, хранимых в той или иной форме.

К наиболее распространенным программным средствам, ориентированным на хранение и обработку графовых конструкций относятся графовые СУБД (Neo4j **Neo4j**, ArangoDB **ArangoDB**, OrientDB **OrientDB** , Grakn **Grakn** и др.), а также так называемые rdf-хранилища (Virtuoso **Virtuoso**, Sesame **Sesame** и др.), предназначенные для хранения конструкций, представленных в модели RDF. Для доступа к информации, хранимой в рамках таких средств, могут использоваться как языки, реализуемые в рамках конкретного средства (например, язык Cypher в Neo4j), так и языки, являющиеся стандартами для большого числа систем такого класса (например, SPARQL для rdf-хранилищ).

Популярность и развитость такого рода средств приводит к тому, что на первый взгляд целесообразным и эффективным кажется вариант реализации *программного варианта реализации платформы интерпретации sc-моделей* на базе одного из таких средств. Однако, существует ряд причин, по которым было принято решение о реализации *программного варианта реализации платформы интерпретации sc-моделей* с нуля. К ним относятся следующие:

- □ для обеспечения эффективности хранения и обработки информационных конструкций определенного вида (в данном случае – конструкций SC-кода, sc-конструкций), должна учитываться специфика этих конструкций. В частности, описанные в работе **Koronchik2013** эксперименты показали значительный прирост эффективности собственного решения по сравнению с существующими на тот момент;
- □ в отличие от классических графовых конструкций, где дуга или ребро могут быть инцидентны только вершине графа (это справедливо и для rdf-графов) в SC-коде вполне типичной является ситуация, когда sc-коннектор инцидентен другому sc-коннектору или даже двум sc-коннекторам. В связи с этим существующие средства хранения графовых конструкций не позволяют в явном виде хранить sc-конструкции (sc-графы). Возможным решением данной проблемы является переход от sc-графа к орграфу инцидентности, пример которого описан в работе **Ivashenko2015** , однако такой вариант приводит к увеличению числа хранимых элементов в несколько раз и значительно снижает эффективность алгоритмов поиска из-за необходимости делать большое количество дополнительных итераций;
- □ в основе обработки информации в рамках Технологии OSTIS лежит многоагентный подход, в рамках которого агенты обработки информации, хранимой в sc-памяти (sc-агенты) реагируют на события, происходящие в sc-памяти и обмениваются информацией посредством спецификации выполняемых ими действий в sc-памяти **Shunkevich2018** . В связи с этим одной из важнейших задач является реализация в рамках *программного варианта реализации платформы интерпретации sc-моделей* возможности подписки на события, происходящие в программной модели sc-памяти, которая на данный момент практически не поддерживается в рамках современных средств хранения и обработки графовых конструкций;
- □ SC-код позволяет описывать также внешние информационные конструкции любого рода (изображения, текстовые файла, аудио- и видеофайлы и т.д.), которые формально трактуются как содержимое *sc-элементов*, являющихся знаками *внешних файлов ostis-системы*. Таким образом, компонентом *программного варианта реализации платформы интерпретации sc-моделей* должна быть реализация файловой памяти, которая позволяет хранить указанные конструкции в каких-либо общепринятых форматах. Реализация такого компонента в рамках современных средств хранения и обработки графовых конструкций также не всегда представляется возможной.

По совокупности перечисленных причин было принято решение о реализации *программного варианта реализации платформы интерпретации sc-моделей* "с нуля" с учетом особенностей хранения и обработки информации в рамках Технологии OSTIS.]

⇒ *декомпозиция программной системы**:

{
- *Программная модель sc-памяти*
- *Реализация интерпретатора sc-моделей пользовательских интерфейсов*

}

⇒ *пояснение**:

[Текущий *Программный вариант реализации платформы интерпретации sc-моделей компьютерных систем* является web-ориентированным, то есть с точки зрения современной архитектуры каждая ostis-система представляет собой web-сайт доступный онлайн посредством обычного браузера. Такой вариант реализации обладает очевидным преимуществом – доступ к системе возможен из любой точки мира, где есть Интернет, при этом для работы с системой не требуется никакого специализированного программного обеспечения. С другой стороны, такой вариант реализации обеспечивает возможность параллельной работы с системой нескольких пользователей.

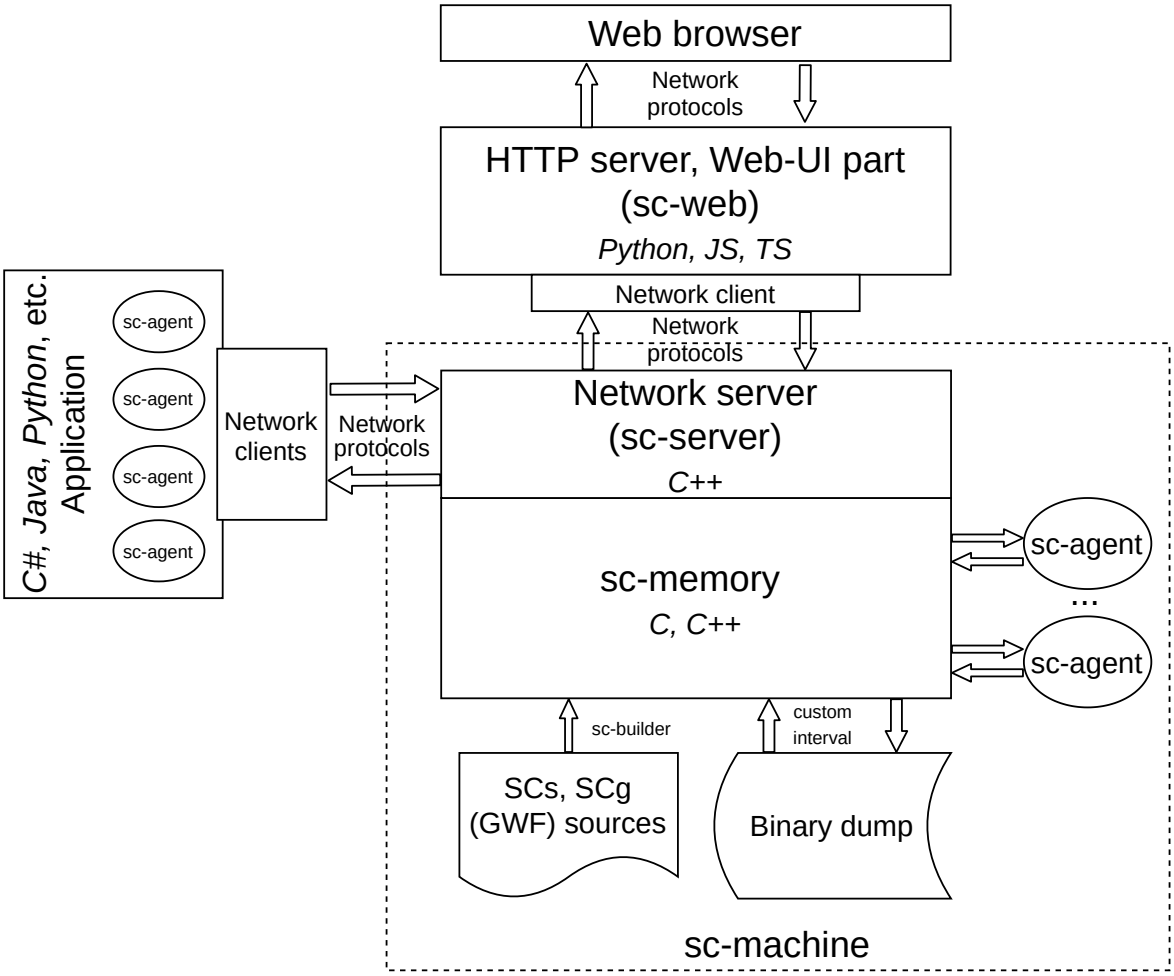
В то же время, взаимодействие клиентской и серверной части организовано таким образом, что web-интерфейс может быть легко заменен на настольный или мобильный интерфейс, как универсальный, так и специализированный.

Данный вариант реализации распространяется под open-source лицензией, для хранения исходных текстов используется хостинг Github и коллективная учетная запись ostis-ai.

Реализация является кроссплатформенной и может быть собрана из исходных текстов в различных операционных системах.]

⇒ *иллюстрация**:

[



]

⇒ *пояснение**:

[На приведенной иллюстрации видно, что ядром платформы является *Программная модель sc-памяти* (sc-machine), которая одновременно может взаимодействовать как с *Реализацией интерпретатора sc-моделей пользовательских интерфейсов* (sc-web **sc_web**), так и с любыми сторонними приложениями по соответствующим сетевым протоколам. С точки зрения общей архитектуры *Реализация интерпретатора sc-моделей пользовательских интерфейсов* выступает как один из множества возможных внешних компонентов, взаимодействующих с *Программной моделью sc-памяти* по сети.]

Программная модель sc-памяти

:= [sc-machine]

:= [Программная модель семантической памяти, реализованная на основе традиционной линейной памяти и включающая средства хранения sc-конструкций и базовые средства для обработки этих конструкций, в том числе удаленного доступа к ним посредством соответствующих сетевых протоколов]

⇐ *программная модель**:
sc-память

∈ *программная модель sc-памяти на основе линейной памяти*

⇒ *основной репозиторий исходных текстов**:
[https://github.com/ostis-ai/sc-machine.git]

⇒ *компонент программной системы**:
- *Реализация sc-хранилища и средств доступа к нему*
  ⇒ *пояснение**:
  [В рамках текущей *Программной модели sc-памяти* под *sc-хранилищем* понимается компонент программной модели, осуществляющий хранение sc-конструкций и доступ к ним через программный интерфейс. В общем случае *sc-хранилище* может быть реализовано по-разному. Кроме собственно *sc-хранилища Программная модель sc-памяти* включает также *Реализацию файловой памяти ostis-системы*, предназначенную для хранения содержимого *внутренних файлов ostis-систем*. Стоит отметить, что при переходе с *Программной модели sc-памяти* на ее аппаратную реализацию файловую память ostis-системы целесообразно будет реализовывать на основе традиционной линейной памяти (во всяком случае, на первых этапах развития *семантического компьютера*).]
- *Реализация базового набора платформенно-зависимых sc-агентов и их общих компонентов*
- *Реализация подсистемы взаимодействия с внешней средой с использованием сетевых протоколов*
- *Реализация вспомогательных инструментальных средств для работы с sc-памятью*
- *Реализация scp-интерпретатора*

⇒ *программная документация**:
[http://ostis-ai.github.io/sc-machine/]

⇒ *используемый язык программирования**:
- *C*
- *C++*
- *Python*

⇒ *примечание**:
[Текущий вариант *Программной модели sc-памяти* предполагает возможность сохранения состояния (слепка) памяти на жесткий диск и последующей загрузки из ранее сохраненного состояния. Такая возможность необходима для перезапуска системы, в случае возможных сбоев, а также при работе с исходными текстами базы знаний, когда сборка из исходных текстов сводится к формированию слепка состояния памяти, который затем помещается в *Программную модель sc-памяти*.]

Реализация sc-хранилища и средств доступа к нему

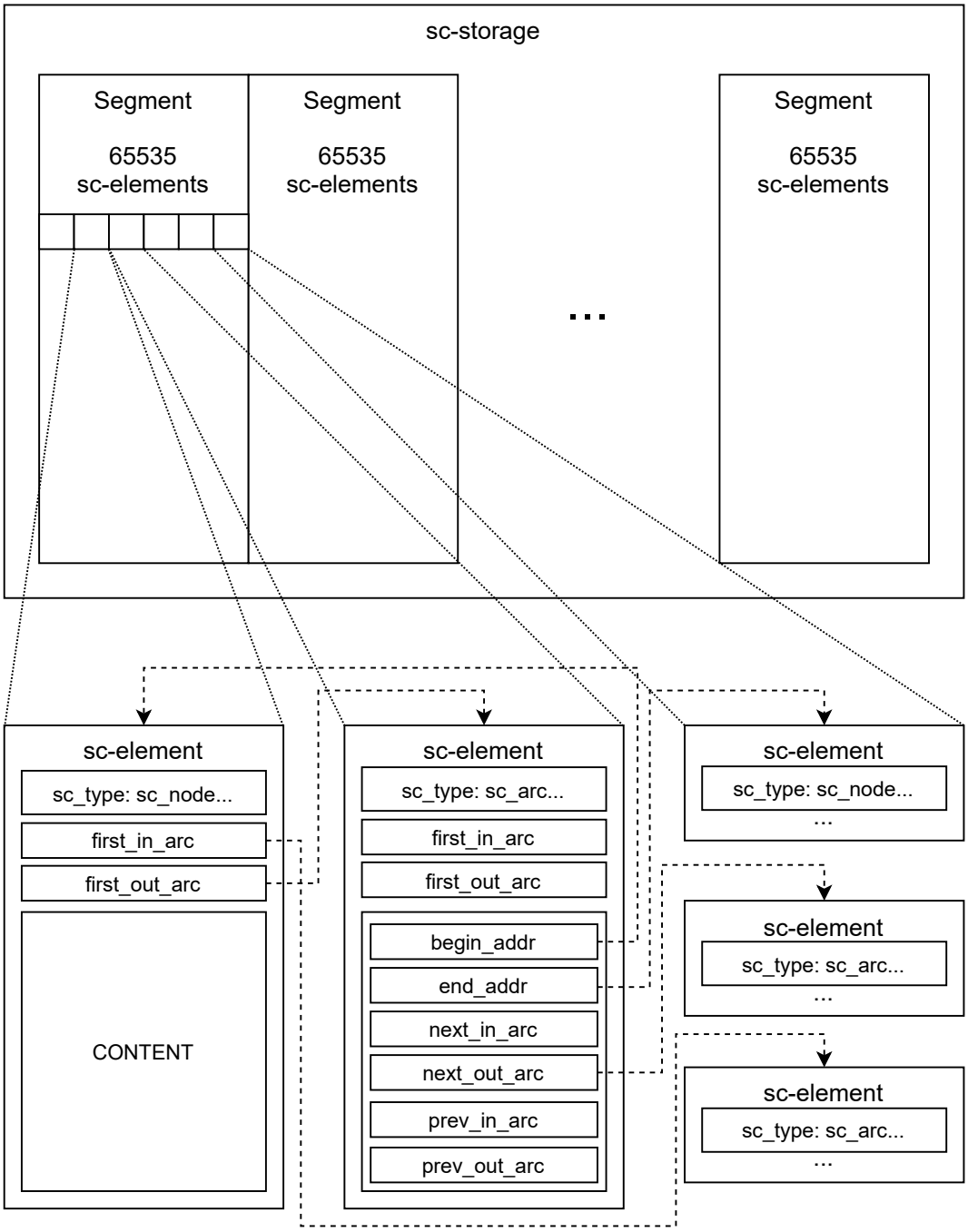
⇒ *компонент программной системы**:
- *Реализация sc-хранилища*
- *Реализация файловой памяти ostis-системы*

Реализация sc-хранилища

∈ *реализация sc-хранилища на основе линейной памяти*

⇒ *иллюстрация**:

[



]

⇒ *класс объектов программной системы**:
сегмент sc-хранилища

:= [страница sc-хранилища]

⇒ *пояснение**:

[В рамках данной реализации *sc-хранилища* *sc-память* моделируется в виде набора *сегментов*, каждый из которых представляет собой фиксированного размера упорядоченную последовательность *элементов sc-хранилища*, каждый из которых соответствует конкретному sc-элементу. В настоящее время каждый сегмент состоит из $2^{16} - 1 = 65535$ *элементов sc-хранилища*. Выделение *сегментов sc-хранилища* позволяет, с одной стороны, упростить адресный доступ к *элементам sc-хранилища*, с другой стороны – реализовать возможность выгрузки части sc-памяти из оперативной памяти на файловую систему при необходимости. Во втором случае сегмент sc-хранилища становится минимальной (атомарной) выгружаемой частью sc-памяти. Механизм выгрузки сегментов реализуется в соответствии с существующими принципами организации виртуальной памяти в современных операционных системах.]

⇒ *примечание**:

[Максимально возможное число сегментов ограничивается настройками программной реализации sc-хранилища (в настоящее время по умолчанию установлено количество $2^{16} - 1 = 65535$ сегментов, но в общем случае оно может быть другим). Таким образом, технически максимальное количество хранимых sc-элементов в текущей реализации составляет около 4.3×10^9 sc-элементов.]

⇒ *примечание**:

[По умолчанию все сегменты физически располагаются в оперативной памяти, если объема памяти не хватает, то предусмотрен механизм выгрузки части сегментов на жесткий диск (механизм виртуальной памяти).]

⇒ *класс объектов программной системы**:

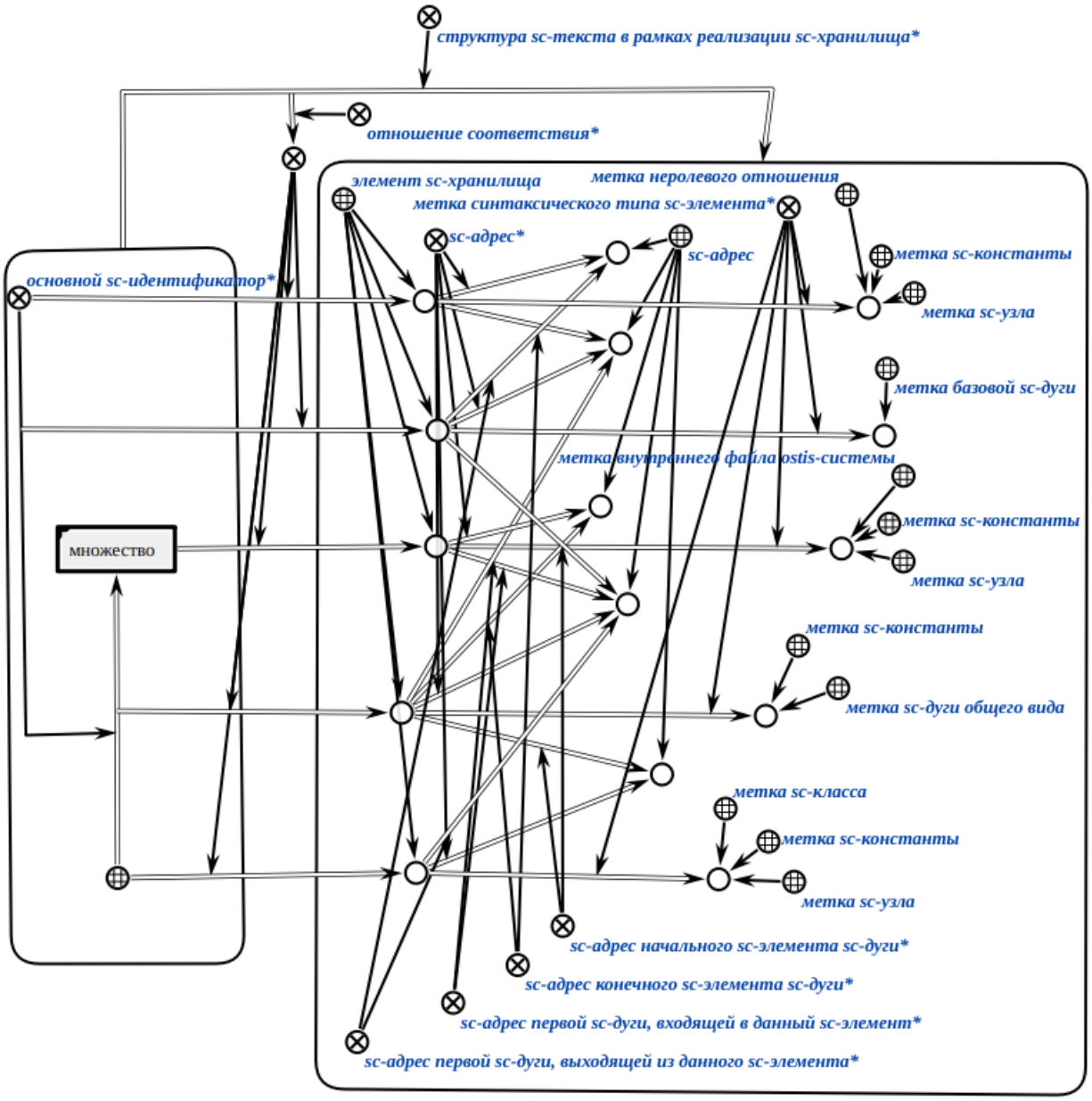
элемент sc-хранилища

⇒ *пояснение**:

[Каждый сегмент состоит из набора структур данных, описывающих конкретные *sc-элементы* (элементов sc-хранилища). Независимо от типа описываемого sc-элемента каждый *элемент sc-хранилища* имеет фиксированный размер (в текущий момент – 48 байт), что обеспечивает удобство их хранения. Таким образом, максимальный размер базы знаний в текущей программной модели sc-памяти может достигнуть 223 Гб (без учета содержимого *внутренних файлов ostis-системы*, хранимого на внешней файловой системе).]

⇒ *пример**:

[



]

⇒ *пояснение**:

[Для наглядности в данном примере опущены *метки уровня доступа*]

sc-адрес

:= [адрес элемента sc-хранилища, соответствующего заданному sc-элементу, в рамках текущего состояния реализации sc-хранилища в составе программной модели sc-памяти]

⇒ *пояснение**:

[Каждый элемент sc-хранилища в текущей реализации может быть однозначно задан его адресом (sc-адресом), состоящим из номера сегмента и номера *элемента sc-хранилища* в рамках сегмента. Таким образом, *sc-адрес* служит уникальными координатами *элемента sc-хранилища* в рамках *Реализации sc-хранилища*.]

⇒ *примечание**:

[Sc-адрес никак не учитывается при обработке базы знаний на семантическом уровне и необходим только для обеспечения доступа к соответствующей структуре данных, хранящейся в линейной памяти на уровне *Реализации sc-хранилища*.]

⇒ *примечание**:

[В общем случае sc-адрес элемента sc-хранилища, соответствующего заданному sc-элементу, может меняться, например, при пересборке базы знаний из исходных текстов и последующем перезапуске системы. При этом sc-адрес элемента sc-хранилища, соответствующего заданному sc-элементу, непосредственно в процессе работы системы в текущей реализации меняться не может.]

⇒ *примечание**:

[Для простоты будем говорить "sc-адрес sc-элемента", имея в виду *sc-адрес элемента sc-хранилища*, однозначно соответствующего данному *sc-элементу*.]

⇒ *семейство отношений, однозначно задающих структуру заданной сущности**:

- *номер сегмента sc-хранилища**
- *номер элемента sc-хранилища в рамках сегмента**

элемент sc-хранилища

:= [ячейка sc-хранилища]

:= [элемент sc-хранилища, соответствующий sc-элементу]

:= [образ sc-элемента в рамках sc-хранилища]

:= [структура данных, каждый экземпляр которой соответствует одному sc-элементу в рамках sc-хранилища]

⇒ *пояснение**:

[Каждый элемент sc-хранилища, соответствующий некоторому sc-элементу, описывается его синтаксическим типом (меткой), а также независимо от типа указывается sc-адрес первой входящей в данный sc-элемент sc-дуги и первой выходящей из данного sc-элемента sc-дуги (могут быть пустыми, если таких sc-дуг нет).

Оставшиеся байты в зависимости от типа соответствующего sc-элемента (sc-узел или sc-дуга) могут использоваться либо для хранения содержимого внутреннего файла ostis-системы (может быть пустым, если sc-узел не является знаком файла), либо для хранения спецификации sc-дуги.]

⇒ *разбиение**:

{• *элемент sc-хранилища, соответствующий sc-узлу*

⇒ *семейство отношений, однозначно задающих структуру заданной сущности**:

{• *метка синтаксического типа sc-элемента**
• *метка уровня доступа sc-элемента**
• *sc-адрес первой sc-дуги, выходящей из данного sc-элемента**
• *sc-адрес первой sc-дуги, входящей в данный sc-элемент**
• *содержимое элемента sc-хранилища**

⇒ *второй домен**:

содержимое элемента sc-хранилища

:= [содержимое элемента sc-хранилища, соответствующего внутреннему файлу ostis-системы]

⇒ *пояснение**:

[Каждый sc-узел в текущей реализации может иметь содержимое (может стать *внутренним файлом ostis-системы*). В случае, если размер содержимого внутреннего файла ostis-системы не превышает 48 байт (размер *спецификации sc-дуги в рамках sc-хранилища*, например небольшой *строковый sc-идентификатор*), то это содержимое явно хранится в рамках элемента sc-хранилища в виде последовательности байт. В противном случае оно помещается в специальным образом организованную файловую память (за ее организацию отвечает отдельный модуль платформы, который в общем случае может быть устроен по-разному), а в рамках элемента sc-хранилища хранится

уникальный адрес соответствующего файла, позволяющий быстро найти его на файловой системе.]

}

⇒ *примечание**:

[*sc-адрес первой sc-дуги, выходящей из данного sc-элемента**, *sc-адрес первой sc-дуги, входящей в данный sc-элемент** и *содержимое элемента sc-хранилища** в общем случае могут отсутствовать (быть нулевыми, "пустыми"), но размер элемента в байтах останется тем же.]

- *элемент sc-хранилища, соответствующий sc-дуге*

⇒ *семейство отношений, однозначно задающих структуру заданной сущности**:

{
- *метка синтаксического типа sc-элемента**
- *метка уровня доступа sc-элемента**
- *sc-адрес первой sc-дуги, выходящей из данного sc-элемента**
- *sc-адрес первой sc-дуги, входящей в данный sc-элемент**
- *спецификация sc-дуги в рамках sc-хранилища**

⇒ *второй домен**:

спецификация sc-дуги в рамках sc-хранилища

⇒ *семейство отношений, однозначно задающих структуру заданной сущности**:

{
- *sc-адрес начального sc-элемента sc-дуги**
- *sc-адрес конечного sc-элемента sc-дуги**
- *sc-адрес следующей sc-дуги, выходящей из того же sc-элемента**
- *sc-адрес следующей sc-дуги, входящей в тот же sc-элемент**
- *sc-адрес предыдущей sc-дуги, выходящей из того же sc-элемента**
- *sc-адрес предыдущей sc-дуги, входящей в тот же sc-элемент**

}

}

⇒ *примечание**:

[sc-ребра в текущий момент хранятся так же, как sc-дуги, то есть имеют начальный и конечный sc-элементы, отличие заключается только в *метке синтаксического типа sc-элемента*. Это приводит к ряду неудобств при обработке, но sc-ребра используются в настоящее время достаточно редко.]

}

⇒ *примечание**:

[С точки зрения программной реализации структура данных для хранения sc-узла и sc-дуги остается остается та же, но в ней меняется список полей (компонентов).

Кроме того, как можно заметить каждый элемент sc-хранилища (в том числе, *элемент sc-хранилища, соответствующий sc-дуге*) не хранит список sc-адресов связанных с ним sc-элементов, а хранит sc-адреса одной выходящей и одной входящей дуги, каждая из которых в свою очередь хранит sc-адреса следующей и предыдущей дуг в списке исходящих и входящих sc-дуг для соответствующих элементов.

Все перечисленное позволяет:

- □ сделать размер такой структуры фиксированным (в настоящее время 48 байт) и не зависящим от синтаксического типа хранимого sc-элемента;
- □ обеспечить возможность работы с sc-элементами без учета их синтаксического типа в случаях, когда это необходимо (например, при реализации поисковых запросов вида “Какие sc-элементы являются элементами данного множества”, “Какие sc-элементы непосредственно связаны с данным sc-элементом” и т.д.);
- □ обеспечить возможность доступа к *элементу sc-хранилища* за константное время;
- □ обеспечить возможность помещения *элемента sc-хранилища* в процессорный кэш, что в свою очередь, позволяет ускорить обработку sc-конструкций;

]

⇒ *примечание**:

[Текущая *Программная модель sc-памяти* предполагает, что вся sc-память физически расположена на одном компьютере. Для реализации распределенного варианта *Программной модели sc-памяти* предполагается расширить *sc-адрес* указанием адреса того физического устройства, где хранится соответствующий *элемент sc-хранилища*.]

метка синтаксического типа sc-элемента

:= [уникальный числовой идентификатор, однозначно соответствующий заданному типу sc-элементов и приписываемый соответствующему элементу sc-хранилища на уровне реализации]

⇒ *примечание**:

[Очевидно, что тип (класс, вид) sc-элемента в sc-памяти может быть задан путем явного указания принадлежности данного sc-элемента соответствующему классу (sc-узел, sc-дуга и т.д.).

Однако, в рамках *платформы интерпретации sc-моделей компьютерных систем* должен существовать какой-либо набор *меток синтаксического типа sc-элемента*, которые задают тип элемента на уровне платформы и не имеют соответствующей sc-дуги принадлежности (а точнее – базовой sc-дуги), явно хранимой в рамках sc-памяти (ее наличие подразумевается, однако она не хранится явно, поскольку это приведет к бесконечному увеличению числа sc-элементов, которые необходимо хранить в sc-памяти). Как минимум, должна существовать метка, соответствующая классу *базовая sc-дуга*, поскольку явное указание принадлежности sc-дуги данному классу порождает еще одну *базовую sc-дугу*.

Таким образом, *базовые sc-дуги*, обозначающие принадлежность sc-элементов некоторому известному ограниченному набору классов представлены неявно. Этот факт необходимо учитывать в ряде случаев, например, при проверке принадлежности sc-элемента некоторому классу, при поиске всех выходящих sc-дуг из заданного sc-элемента и т.д.

При необходимости некоторые из таких неявно хранимых sc-дуг могут быть представлены явно, например, в случае, когда такую sc-дугу необходимо включить в какое-либо множество, то есть провести в нее другую sc-дугу. В этом случае возникает необходимость синхронизации изменений, связанных с данной sc-дугой (например, ее удалении), в явном и неявном ее представлении. В текущей *Реализации sc-хранилища* данный механизм не реализован.

Таким образом, полностью отказаться от *меток синтаксического типа sc-элементов* невозможно, однако увеличение их числа хоть и повышает производительность платформы за счет упрощений некоторых операций по проверке типов sc-элемента, но приводит к увеличению числа ситуаций, в которых необходимо учитывать явное и неявное представление sc-дуг, что, в свою очередь, усложняет развитие платформы и разработку программного кода для обработки хранимых sc-конструкций.]

⇐ *второй домен**:

*метка синтаксического типа sc-элемента**

⊃ *метка sc-узла*

⇒ *числовое выражение в шестнадцатеричной системе**:

[0x1]

⊃ *метка внутреннего файла ostis-системы*

⇒ *числовое выражение в шестнадцатеричной системе**:

[0x2]

⊃ *метка sc-ребра общего вида*

⇒ *числовое выражение в шестнадцатеричной системе**:

[0x4]

⊃ *метка sc-дуги общего вида*

⇒ *числовое выражение в шестнадцатеричной системе**:

[0x8]

⊃ *метка sc-дуги принадлежности*

⇒ *числовое выражение в шестнадцатеричной системе**:

[0x10]

⊃ *метка sc-константы*

⇒ *числовое выражение в шестнадцатеричной системе**:

[0x20]

⊃ *метка sc-переменной*

⇒ *числовое выражение в шестнадцатеричной системе**:

[0x40]

⊃ *метка позитивной sc-дуги принадлежности*

⇒ *числовое выражение в шестнадцатеричной системе**:

[0x80]

⊃ *метка негативной sc-дуги принадлежности*

⇒ *числовое выражение в шестнадцатеричной системе**:

[0x100]

⊃ *метка нечеткой sc-дуги принадлежности*

⇒ *числовое выражение в шестнадцатеричной системе*:
[0x200]

⊃ *метка постоянной sc-дуги*
⇒ *числовое выражение в шестнадцатеричной системе*:
[0x400]

⊃ *метка временной sc-дуги*
⇒ *числовое выражение в шестнадцатеричной системе*:
[0x800]

⊃ *метка небинарной sc-связки*
⇒ *числовое выражение в шестнадцатеричной системе*:
[0x80]

⊃ *метка sc-структуры*
⇒ *числовое выражение в шестнадцатеричной системе*:
[0x100]

⊃ *метка ролевого отношения*
⇒ *числовое выражение в шестнадцатеричной системе*:
[0x200]

⊃ *метка неролевого отношения*
⇒ *числовое выражение в шестнадцатеричной системе*:
[0x400]

⊃ *метка sc-класса*
⇒ *числовое выражение в шестнадцатеричной системе*:
[0x800]

⊃ *метка абстрактной сущности*
⇒ *числовое выражение в шестнадцатеричной системе*:
[0x1000]

⊃ *метка материальной сущности*
⇒ *числовое выражение в шестнадцатеричной системе*:
[0x2000]

⊃ *метка константной позитивной постоянной sc-дуги принадлежности*
:= [метка базовой sc-дуги]
:= [метка sc-дуги основного вида]
⇐ *пересечение*:
{
- *метка sc-дуги принадлежности*
- *метка sc-константы*
- *метка позитивной sc-дуги принадлежности*
- *метка постоянной sc-дуги*

}
⇒ *примечание*:
[*метки синтаксических типов sc-элементов* могут комбинироваться между собой для получения более частных классов меток. С точки зрения программной реализации такая комбинация выражается операцией побитового сложения значений соответствующих меток.]

⊃ *метка переменной позитивной постоянной sc-дуги принадлежности*
⇐ *пересечение*:
{
- *метка sc-дуги принадлежности*
- *метка sc-переменной*
- *метка позитивной sc-дуги принадлежности*
- *метка постоянной sc-дуги*

}

⇒ *примечание*:
[Числовые выражения некоторых классов меток могут совпадать. Это сделано для уменьшения размера элемента sc-хранилища за счет уменьшения максимального размера метки. Конфликт в данном случае не возникает, поскольку такие классы меток не могут комбинироваться, например *метка ролевого отношения* и *метка нечеткой sc-дуги принадлежности*.]

⇒ *примечание*:
[Важно отметить, что каждому из выделенных классов меток (кроме классов, получаемых путем комбинации других классов) однозначно соответствует порядковый номер бита в линейной памяти, что можно заметить, глядя на соответствующие числовые выражения классов меток. Это означает, что классы меток не включаются друг в друга, например, указание *метки позитивной sc-дуги принадлежности* не означает

автоматическое указание *метки sc-дуги принадлежности*. Это позволяет сделать операции комбинирования и сравнения меток более эффективными.]

⇐ *недостатки текущего состояния**:

{• [На данный момент число *меток синтаксического типа sc-элемента* достаточно велико, что приводит к возникновению достаточно большого числа ситуаций, в которых нужно учитывать явное и неявное хранение sc-дуг принадлежности соответствующим классам. С другой стороны, изменение набора меток с какой-либо целью в текущем варианте реализации представляет собой достаточно трудоемкую задачу (с точки зрения объема изменений в программном коде платформы и sc-агентов, реализованных на уровне платформы), а расширение набора меток без увеличения объема элемента sc-хранилища в байтах оказывается и вовсе невозможным.]

⇒ *вариант решения**:

[Решением данной проблемы является максимально возможная минимизация числа меток, например, до числа меток, соответствующих *Алфавиту SC-кода*. В таком случае принадлежность sc-элементов любым другим классам будет записываться явно, а число ситуаций, в которых необходимо будет учитывать неявное хранение sc-дуг, будет минимальным.]

• [Некоторые метки из текущего набора *меток синтаксического типа sc-элемента* используются достаточно редко (например, *метка sc-ребра общего вида* или *метка негативной sc-дуги принадлежности*), в свою очередь, в sc-памяти могут существовать классы, имеющие достаточно много элементов (например, *бинарное отношение* или *число*). Данный факт не позволяет в полной мере использовать эффективность наличия меток.]

⇒ *вариант решения**:

[Решением данной проблемы является отказ от заранее известного набора меток и переход к динамическому набору меток (при этом их число может оставаться фиксированным). В этом случае набор классов, выражаемых в виде меток будет формироваться на основании каких-либо критериев, например, числа элементов данного класса или частоты обращений к нему.]

}

метка уровня доступа sc-элемента

⇐ *второй домен**:

*метка уровня доступа sc-элемента**

⇒ *обобщенная структура**:

{• *метка уровня доступа sc-элемента на чтение*

• *метка уровня доступа sc-элемента на запись*

}

⇒ *пояснение**:

[В текущей *Реализации sc-хранилища* *метки уровня доступа* используются для того, чтобы обеспечить возможность ограничения доутспа некоторых процессов в sc-памяти к некоторым sc-элементам, хранимым в sc-памяти.

Каждому элементу sc-хранилища соответствует *метка уровня доступа sc-элемента на чтение* и *метка уровня доступа sc-элемента на запись*, каждая из которых выражается числом от 0 до 255.

В свою очередь, каждому процессу (чаще всего, соответствующему некоторому sc-агенту), который пытается получить доступ к данному элементу sc-хранилища (прочитать или изменить его) соответствует уровень доступа на чтение и запись, выраженный в том же числовом диапазоне. Указанный уровень доступа для процесса является частью *контекста процесса*. Доступ на чтение или запись к элементу sc-хранилища не разрешается, если уровень доступа соответственно на чтение или запись у процесса ниже, чем у элемента sc-хранилища, к которому осуществляется доступ.

Таким образом нулевое значение *метки уровня доступа sc-элемента на чтение* и *метки уровня доступа sc-элемента на запись* означает, что любой процесс может получить неограниченный доступ к данному элементу sc-хранилища.]

Реализация файловой памяти ostis-системы

⇒ *пояснение**:

[Для хранения содержимого внутренних файлов ostis-систем, размер которого превышает 48 байт, используются файлы, явно хранимые на файловой системе, доступ к которой осуществляется средствами операционной системы, на которой работает *Программный вариант реализации платформы интерпретации sc-моделей компьютерных систем*.

В общем случае множество различных внутренних файлов ostis-системы могут иметь одинаковое содержимое. Было бы разумно не хранить содержимое одинаковых файлов дважды. Для этого при создании

соответствующего sc-узла и указании файла на файловой системе, который является содержимым данного sc-узла, вычисляется hash-сумма содержимого с помощью алгоритма SHA256. В результате получается строка из 32 символов, которая и выступает в качестве *содержимого элемента sc-хранилища**. Само же содержимое копируется в файл на файловой системе, путь к которому строится на основании hash-суммы. Рядом с этим файлом создается файл, в котором хранятся sc-адреса всех sc-узлов, имеющих одно и то же ранее указанное содержимое. Таким образом, для того, чтобы найти все sc-узлы, имеющие указанное содержимое, необходимо вычислить hash-сумму искомого содержимого-образца и проверить наличие файла на файловой системе по пути, вычисляемому из hash-суммы и если он существует, то вернуть список хранящихся sc-адресов.

Кроме того, для реализации быстрого поиска sc-элементов по их строковым sc-идентификаторам или их фрагментам (подстрокам) используется дополнительное хранилище вида ключ-значение, которое ставит в соответствие *строковому sc-идентификатору* *sc-адрес* того *sc-элемента*, идентификатором которого является данная строка (в случае основного и системного sc-идентификатора) или *sc-элемента*, который является знаком *внутреннего файла ostis-системы* (в случае неосновного sc-идентификатора).]

контекст процесса в рамках программной модели sc-памяти

:= [ScContext]

:= [контекст процесса, выполняемого на уровне программной модели sc-памяти]

:= [метаописание процесса в sc-памяти, выполняемого на уровне программной модели sc-памяти]

:= [структура данных, содержащая метаинформацию о процессе, выполняемом в sc-памяти на уровне платформы]

⇐ *класс компонентов**:
Реализация sc-хранилища

⇒ *пояснение**:

[Каждому процессу, выполняемому в sc-памяти на уровне *платформы интерпретации sc-моделей компьютерных систем* (и чаще всего соответствующего некоторому *sc-агенту*, реализованному на уровне платформы) ставится в соответствие *контекст процесса*, который является структурой данных, описывающей метаинформацию о данном процессе. На текущий момент контекст процесса содержит сведения об уровне доступа на чтение и запись для данного процесса (См. *метка уровня доступа sc-элемента*).

При вызове в рамках процесса любых функций (методов), связанных с доступом к хранимым в sc-памяти конструкциям одним из параметров обязательно является *контекст процесса*.]

блокировка sc-элемента в рамках программной модели sc-памяти

:= [ScLock]

⇐ *класс компонентов**:
Реализация sc-хранилища

⇒ *смотрите**:
??

подписка на событие в sc-памяти в рамках программной модели sc-памяти

:= [ScEvent]

:= [структура данных, описывающая в рамках программной модели sc-памяти соответствие между классом событий в sc-памяти и действиями, которые должно быть совершены при возникновении в sc-памяти событий данного класса]

⇐ *класс компонентов**:
Реализация sc-хранилища

⇒ *пояснение**:

[Для того, чтобы обеспечить возможность создания sc-агентов в рамках *платформы интерпретации sc-моделей компьютерных систем* реализована возможность создать подписку на событие, принадлежащее одному из классов *элементарных событий в sc-памяти** (см. Раздел "*Предметная область и онтология темпоральных сущностей базы знаний ostis-системы*"), уточнив при этом sc-элемент, с которым должно быть связано событие данного класса (например, sc-элемент, для которого должна появиться входящая или исходящая sc-дуга). Подписка на событие представляет собой структуру данных, описывающую класс ожидаемых событий и функцию в программном коде, которая должна быть вызвана при возникновении данного события.

Все подписки на события регистрируются в рамках таблицы событий. При любом изменении в sc-памяти происходит просмотр данной таблицы и запуск функций, соответствующих произошедшему событию.

В текущей реализации обработка каждого события осуществляется в отдельном потоке операционной системы, при этом на уровне реализации задается параметр, описывающий число максимальных потоков, которые могут выполняться параллельно.

Таким образом оказывается возможным реализовать sc-агенты, реагирующие на события в sc-памяти, а также при выполнении некоторого процесса в sc-памяти приостановить его работу и дождаться возникновения некоторого события (например, создать подзадачу некоторому коллективу sc-агентов и дождаться ее решения).]

sc-итератор

:= [ScIterator]

⇐ *класс компонентов**:
Реализация sc-хранилища

⇒ *пояснение**:

[С функциональной точки зрения *sc-итераторы* как часть *Реализации sc-хранилища* представляют собой базовое средство доступа к конструкциям, хранимым в sc-памяти, которое позволяет осуществить чтение (просмотр) конструкций, изоморфных простейшим шаблонам – *трехэлементным sc-конструкциям* и *пятиэлементным sc-конструкциям*.

С точки зрения реализации *sc-итератор* представляет собой структуру данных, которая соответствует определенному дополнительно уточняемому классу sc-конструкций и позволяет при помощи соответствующего набора функций последовательно осуществлять просмотр всех sc-конструкций данного класса, представленных в текущем состоянии sc-памяти (итерацию по sc-конструкциям).

Каждому классу *sc-итераторов* соответствует некоторый известный класс (шаблон, образец) sc-конструкций. При создании sc-итератора данный шаблон уточняется, то есть некоторым (как минимум одному) элементам шаблона ставится в соответствие конкретный заранее известный *sc-элемент* (отправная точка при поиске), а другим элементам шаблона (тем, которые нужно найти) ставится в соответствие некоторый тип sc-элемента из числа типов, соответствующих *меткам синтаксического типа sc-элемента*.

Далее путем вызова соответствующей функции (или метода класса в ООП) осуществляется последовательный просмотр всех sc-конструкций, соответствующих полученному шаблону (с учетом указанных типов sc-элементов и заранее заданных известных sc-элементов), то есть *sc-итератор* последовательно "переключается" с одной конструкции на другую до тех пор, пока такие конструкции существуют. Проверка существования следующей конструкции проверяется непосредственно перед переключением. В общем случае конструкций, соответствующих указанному шаблону, может не существовать, в этом случае итерирование происходить не будет (будет 0 итераций).

На каждой итерации в sc-итератор записываются sc-адреса sc-элементов, входящих в соответствующую sc-конструкцию, таким образом найденные элементы могут быть обработаны нужным образом в зависимости от задачи.]

⊃ *трехэлементный sc-итератор*
⇒ *класс sc-конструкций**:
трехэлементная sc-конструкция

⊃ *пятиэлементный sc-итератор*
⇒ *класс sc-конструкций**:
пятиэлементная sc-конструкция
⇒ *примечание**:
[В настоящее время *пятиэлементный sc-итератор* реализуется на основе *трехэлементных sc-итераторов* и в этом смысле не является атомарным. Однако, введение *пятиэлементных sc-итераторов* целесообразно с точки зрения удобства разработчика программ обработки sc-конструкций.]

sc-шаблон

:= [ScTemplate]

:= [структура данных в линейной памяти, описывающая обобщенную sc-структуру, которая в свою очередь может быть либо явно представлена sc-памяти, либо не представлена в ее текущем состоянии, но может быть представлена при необходимости]

⇐ *класс компонентов**:
Реализация sc-хранилища

⇒ *пояснение**:

[*Sc-итераторы* позволяют осуществлять поиск только sc-конструкций простейшей конфигурации. Для реализации поиска sc-конструкций более сложной конфигурации, а также генерации сложных sc-конструкций используются *sc-шаблоны*, на основе которых затем осуществляется поиск или генерация конструкций.

Sc-шаблон представляет собой структуру данных, соответствующую некоторой *обобщенной структуре*, т.е. *структуре*, содержащей *sc-переменные*. При помощи соответствующего набора функций можно осуществлять

- поиск в текущем состоянии sc-памяти всех sc-конструкций, изоморфных заданному шаблону. В качестве параметров поиска можно указать значения для каких-либо из sc-переменных в составе шаблона. После осуществления поиска будет сформировано множество результатов поиска, каждый из которых представляет собой множество пар вида “sc-переменная из шаблона – соответствующая ей sc-константа”. Данное множество может быть пустым (в текущем состоянии sc-памяти нет конструкций, изоморфных заданному образцу) или содержать один или более элементов. Подстановка значений sc-переменных может осуществляться как по sc-адресу, так и по системному sc-идентификатору;
- генерацию sc-конструкции, изоморфной заданному шаблону. Параметры и результаты генерации формируются так же, как в случае поиска, за исключением того, что в случае генерации результат всегда один и множество результатов не формируется;

Таким образом, каждый *sc-шаблон* фактически задает множество шаблонов, формируемых путем указания значений для sc-переменных, входящих в исходный шаблон.

Важно отметить, что *sc-шаблон* представляет собой структуру данных в линейной памяти, соответствующую некоторой *обобщенной структуре* в sc-памяти, но не саму эту *обобщенную структуру*. Это означает, что sc-шаблон может быть автоматически сформирован на основе *обобщенной структуры*, явно представленной в sc-памяти, а также сформирован на уровне программного кода путем вызова соответствующих функций (методов). Во втором случае *sc-шаблон* будет существовать только в линейной памяти и соответствующая *обобщенная структура* не будет явно представлена в sc-памяти. В этом случае подстановка значений sc-переменных будет возможна только по системному sc-идентификатору, поскольку sc-адресов у соответствующих элементов шаблона существовать не будет.]

⇒ *примечание**:

[При поиске sc-конструкций, изоморфных заданному шаблону, крайне важно с точки зрения производительности с какого sc-элемента начинать поиск. Как известно, в общем случае задача поиска в графе представляет собой NP-полную задачу, однако поиск в sc-графе позволяет учитывать семантику обрабатываемой информации, что, в свою очередь, позволяет существенно снизить время поиска.

Одним из возможных вариантов оптимизации алгоритма поиска, реализованным на данный момент, является упорядочение трехэлементных sc-конструкций, входящих в состав sc-шаблона, по очередности поиска по этим sc-конструкциям по критерию снижения числа возможных вариантов поиска, которые порождает та или иная трехэлементная sc-конструкция, содержащая sc-переменные. Так, в первую очередь при поиске выбираются те трехэлементные sc-конструкции, которые изначально содержат две sc-константы, затем те, которые изначально содержат одну sc-константу. После выполнения шага поиска приоритет sc-конструкций изменяется с учетом результатов, полученных на предыдущем шаге.

Другой вариант оптимизации основывается на той особенности формализации в SC-коде, что в общем случае число sc-дуг, входящих в некоторый sc-элемент, как правило значительно меньше числа выходящих из него sc-дуг. Таким образом, целесообразным оказывается осуществлять поиск вначале по входящим sc-дугам.]

⇒ *примечание**:

[Можно предположить, что возможности, предоставляемые *sc-шаблонами* позволяют полностью исключить использование *sc-итераторов*. Однако это не совсем так по следующим причинам:

- функции поиска и генерации по шаблону реализуются на основе sc-итераторов, как базового средства поиска sc-конструкций в рамках *Реализации sc-хранилища*.
- *sc-итераторы* дают возможность более гибко организовать процесс поиска с учетом семантики конкретных sc-элементов, участвующих в поиске. Так например, можно учесть тот факт, что для некоторых sc-элементов число входящих sc-дуг значительно меньше, чем выходящих (или наоборот) таким образом, при поиске конструкций, содержащих такие sc-элементы более эффективно начать перебор с тех участков, где дуг потенциально меньше.

]

Реализация подсистемы взаимодействия с внешней средой с использованием сетевых протоколов

⇒ *компонент программной системы**:

- *PРеализация подсистемы взаимодействия с внешней средой с использованием протоколов на основе формата JSON*

⇒ *пояснение**:

[Взаимодействие программной модели sc-памяти с внешними ресурсами может осуществляться посредством специализированного программного интерфейса (API), однако этот вариант неудобен в большинстве случае, поскольку:

- □ поддерживается только для очень ограниченного набора языков программирования (C, C++, Python);
- □ требует того, чтобы клиентское приложение, обращающееся к программной модели sc-памяти, фактически составляло с ней единое целое, таким образом исключается возможность построения распределенного коллектива ostis-систем;
- □ как следствие предыдущего пункта, исключается возможность параллельной работы с sc-памятью нескольких клиентских приложений.

Для того, чтобы обеспечить возможность удаленного доступа к sc-памяти не учитывая при этом языки программирования, с помощью которых реализовано конкретное клиентское приложение, было принято решение о реализации возможности доступа к sc-памяти с использованием универсальных протоколов, не зависящих от средств реализации того или иного компонента или системы. В качестве такого протокола был разработан текстовый протокол на базе JSON.]

Реализация подсистемы взаимодействия с внешней средой с использованием протоколов на основе формата JSON

⇒ *пояснение**:

[В связи с большим числом недостатков протокола SCTP было принято решение о разработке другого протокола на основе какого-либо общепринятого текстового транспортного формата. В качестве такого формата был выбран формат JSON.]

⇐ *реализация**:

Протокол взаимодействия с sc-памятью на основе JSON

⇒ *примечание**:

[Данный протокол пока не имеет собственного названия]

⇒ *программная документация**:

[http://ostis-ai.github.io/sc-machine/http/websocket/]

⇒ *пояснение**:

[В рамках *Протокола взаимодействия с sc-памятью на основе JSON* каждая команда представляет собой json-объект, в котором указываются идентификатор команда, тип команды и ее аргументы. В свою очередь ответ на команду также представляет собой json-объект, в котором указываются идентификатор команды, ее статус (выполнена успешно/безуспешно) и результаты. Структура аргументов и результатов команды определяется типом команды.]

⇒ *достоинство**:

- [JSON является общепринятым открытым форматом, для работы с которым существует большое количество библиотек для популярных языков программирования. Это, в свою очередь, упрощает реализацию клиента и сервера для протокола, построенного на базе JSON.]
- [Реализация протокола на базе JSON не накладывает принципиальных ограничений на объем (длину) каждой команды, в отличие от бинарного протокола. Таким образом, появляется возможность использования неатомарных команд, позволяющих, например, за один акт пересылки такой команды по сети создать сразу несколько sc-элементов. Важными примерами таких команд являются *Команда генерации по произвольному образцу* и *Команда поиска по произвольному образцу*.]

⇒ *примечание**:

[Можно сказать, что протокол на базе JSON является следующим шагом на пути к созданию мощного и универсального языка запросов, аналогичного языку SQL для реляционных баз данных и предназначенному для работы с sc-памятью. Следующий шагом станет реализация такого протокола на основе одного из стандартов внешнего отображения sc-конструкций, например, *SCs-кода*, что, в свою очередь, позволит передавать в качестве команд целые программы обработки sc-конструкций, например на языке SCP.]

Реализация вспомогательных инструментальных средств для работы с sc-памятью

⇒ *компонент программной системы**:

Реализация сборщика базы знаний из исходных текстов, записанных в SCs-коде

:= [sc-builder]

⇒ *используемый язык**:

SCs-код

⇒ *пояснение**:

[Сборщик базы знаний из исходных текстов позволяет осуществить сборку базы знаний из набора исходных текстов, записанных в SCs-коде с ограничениями (см. *Раздел **про исходные тексты***) в бинарный формат, воспринимаемый *Программной моделью sc-памяти*. При этом возможна как сборка "с нуля" (с уничтожением ранее созданного слепка памяти), так и аддитивная сборка, когда информация, содержащаяся в заданном множестве файлов, добавляется к уже имеющемуся слепку состояния памяти.

В текущей реализации сборщик осуществляет "склеивание" ("слияние") sc-элементов, имеющих на уровне исходных текстов одинаковые *системные sc-идентификаторы*.]

Реализация интерпретатора sc-моделей пользовательских интерфейсов

:= [sc-web]

⇒ *пояснение**:

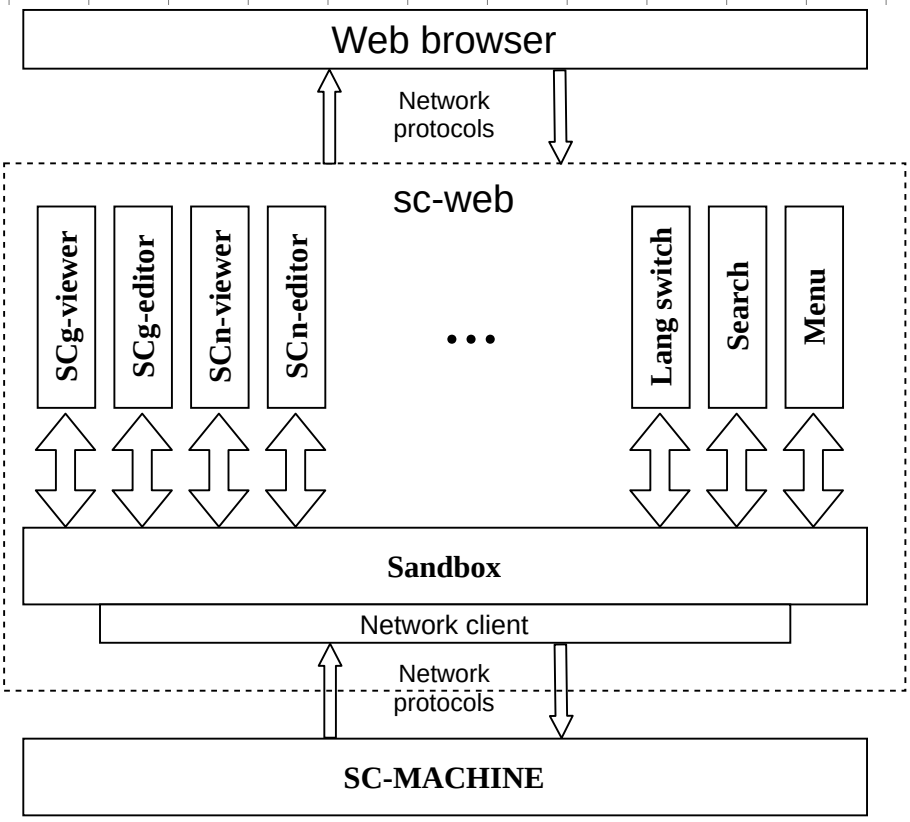
[Наряду с реализацией *Программной модели sc-памяти* важной частью *Программного варианта реализации платформы интерпретации sc-моделей компьютерных систем* является *Реализация интерпретатора sc-моделей пользовательских интерфейсов*, которая предоставляет базовые средства просмотра и редактирования базы знаний пользователем, средства для навигации по базе знаний (задания вопросов к базе знаний) и может дополняться новыми компонентами в зависимости от задач, решаемых каждой конкретной ostis-системой.]

⇒ *используемый язык программирования**:

- *JavaScript*
- *TypeScript*
- *Python*

⇒ *иллюстрация**:

[



]

⇒ *пояснение**:

[На данной иллюстрации показан планируемый вариант архитектуры *Реализация интерпретатора sc-моделей пользовательских интерфейсов*, важным принципом которой является простота и однотипность подключения любых компонентов пользовательского интерфейса (редакторов, визуализаторов, переключателей, команд меню и т.д.). Для этого реализуется программная прослойка Sandbox, в рамках которой реализуются низкоуровневые операции взаимодействия с серверной частью и которая обеспечивает более удобный программный интерфейс для разработчиков компонентов.]

⇒ *недостатки текущей реализации**:

{ •

[Отсутствие единого унифицированного механизма клиент-серверного взаимодействия. Часть компонентов (визуализатор sc-текстов в SCn-коде, команды меню и др.) работают по протоколу HTTP, часть по протоколу SCTP с использованием технологии WebSocket, это приводит к значительным трудностям при развитии платформы.]

- [Протокол HTTP предполагает четкое разделение активного клиента и пассивного сервера, который отвечает на запросы клиентов. Таким образом, сервер (в данном случае – sc-память) практически не имеет возможности по своей инициативе отправить сообщение клиенту, что повышает безопасность системы, но значительно снижает ее интерактивность. Кроме того, такой вариант реализации затрудняет реализацию принятого в Технологии OSTIS многоагентного подхода, в частности, затрудняет реализацию sc-агентов на стороне клиента. Указанные проблемы могут быть решены путем постоянного мониторинга определенных событий со стороны клиента, однако такой вариант неэффективен. Кроме того, часть интерфейса фактически работает напрямую с sc-памятью с использованием технологии WebSocket, а часть – через прослойку на базе библиотеки tornado для языка программирования Python, что приводит к дополнительным зависимостям от сторонних библиотек.]
- [Часть компонентов (например, поле поиска по идентификатору) реализована сторонними средствами и практически никак не связана с sc-памятью. Это затрудняет развитие платформы.]
- [Текущая *Реализация интерпретатора sc-моделей пользовательских интерфейсов* ориентирована только на ведение диалога с пользователем (в стиле вопрос пользователя – ответ системы). Не поддерживаются такие очевидно необходимые ситуации, как выполнение команды, не предполагающей ответа; возникновение ошибки или отсутствие ответа; необходимость задания вопроса системой пользователю и т.д.]
- [Ограничена возможность взаимодействия пользователя с системой без использования специальных элементов управления. Например, можно задать вопрос системе, нарисовав его в SCg-коде, но ответ пользователь не увидит, хотя в памяти он будет сформирован соответствующим агентом.; Большая часть технологий, использованных при реализации платформы, к настоящему моменту устарела, что затрудняет развитие платформы.]
- [Идея платформенной независимости пользовательского интерфейса (построения sc-модели пользовательского интерфейса) реализована не в полной мере. Полностью описать sc-модель пользовательского интерфейса (включая точное размещение, размеры, дизайн компонентов, их поведение и др.) в настоящее время скорее всего окажется затруднительно из-за ограничений производительности, однако вполне возможно реализовать возможность задания вопросов ко всем компонентам интерфейса, изменить их расположение и т.д., однако эти возможности нельзя реализовать в текущей версии реализации платформы.]
- [Интерфейсная часть работает медленно из-за недостатков протокола SCTP и некоторых недостатков реализации серверной части на языке Python.]
- [Не реализован механизм наследования при добавлении новых внешних языков. Например, добавление нового языка даже очень близкого к SCg-коду требует физического копирования кода компонента и внесение соответствующих изменений, при этом получаются два никак не связанных между собой компонента, которые начинают развиваться независимо друг от друга.]
- [Слабый уровень задокументированности текущей *Реализации интерпретатора sc-моделей пользовательских интерфейсов*.]

}

⇒ *требования к будущей реализации**:

{
- [Унифицировать принципы взаимодействия всех компонентов интерфейса с *Программной моделью sc-памяти*, независимо от того, к какому типу относится компонент. Например, список команд меню должен формироваться через тот же механизм, что и ответ на запрос пользователя, и команда редактирования, сформированная пользователем, и команда добавления нового фрагмента в базу знаний и т.д.]
- [Унифицировать принципы взаимодействия пользователей с системой независимо от способа взаимодействия и внешнего языка. Например, должна быть возможность задания вопросов и выполнения других команд прямо через SCg/SCn интерфейс. При этом необходимо учитывать принципы редактирования базы знаний, чтобы пользователя не мог под видом задания вопроса внести новую информацию в согласованную часть базы знаний.]
- [Унифицировать принципы обработки событий, происходящих при взаимодействии пользователя с компонентами интерфейса – поведение кнопок и других интерактивных компонентов должно задаваться не статически сторонними средствами, а реализовываться в виде агента, который, тем не менее, может быть реализован произвольным образом (не обязательно на платформенно-независимом уровне). Любое действие, совершаемое пользователем, на логическом уровне должно трактоваться и обрабатываться как инициирование агента.]

- [Обеспечить возможность выполнять команды (в частности, задавать вопросы) с произвольным количеством аргументов, в том числе – без аргументов.]
- [Обеспечить возможность отображения ответа на вопрос по частям, если ответ очень большой и для отображения требуется много времени.]
- [Каждый отображаемый компонент интерфейса должен трактоваться как изображение некоторого sc-узла, описанного в базе знаний. Таким образом, пользователь должен иметь возможность задания произвольных вопросов к любым компонентам интерфейса.]
- [Максимально упростить и задокументировать механизм добавления новых компонентов.]
- [Обеспечить возможность добавления новых компонентов на основе имеющихся без создания независимых копий. Например, должна быть возможность создать компонент для языка, расширяющего язык SCg новыми примитивами, переопределять принципы размещения sc-текстов и т.д.]
- [Свести к минимуму зависимость от сторонних библиотек.]
- [Свести к минимуму использование протокола HTTP (начальная загрузка общей структуры интерфейса), обеспечить возможность равноправного двустороннего взаимодействия серверной и клиентской части.]
- [Полностью отказаться от протокола SCTP, перейти на протокол на базе JSON, задокументировать его.]

}

⇒ *примечание**:

[Очевидно, что реализация большинства из приведенных требований связана не только с собственно вариантом реализации платформы, но и требует развития теории логико-семантических моделей пользовательских интерфейсов и уточнения в рамках нее общих принципов организации пользовательских интерфейсов ostis-систем. Однако, принципиальная возможность реализации таких моделей должна быть учтена в рамках реализации платформы.]

⇒ *компонент программной системы**:

- *Панель меню команд пользовательского интерфейса*
  ⇒ *пояснение**:
  [*Панель меню команд пользовательского интерфейса* содержит изображения классов команд (как атомарных, так и неатомарных), имеющихся на данный момент в базе знаний и входящих в декомпозицию *Главного меню пользовательского интерфейса* (имеется в виду полная декомпозиция, которая в общем случае может включать несколько уровней неатомарных классов команд).
  Взаимодействие с изображением неатомарного класса команд инициирует команду изображения классов команд, входящих в декомпозицию данного неатомарного класса команд.
  Взаимодействие с изображением атомарного класса команд инициирует генерацию команды данного класса с ранее выбранными аргументами на основе соответствующей *обобщенной формулировки класса команд* (шаблона класса команд).]
- *Компонент переключения языка идентификации отображаемых sc-элементов*
  ⇒ *пояснение**:
  [*Компонент переключения языка идентификации отображаемых sc-элементов* является изображением множества имеющихся в системе естественных языков. Взаимодействие пользователя с данным компонентом переключает пользовательский интерфейс в режим общения с конкретным пользователем с использованием *основных sc-идентификаторов*, принадлежащих данному *естественному языку*. Это значит, что при изображении sc-идентификаторов sc-элементов на каком-либо языке, например, SCg-коде или SCn-коде будут использоваться *основные sc-идентификаторы*, принадлежащие данному *естественному языку*. Это касается как sc-элементов, отображаемых в рамках *Панели визуализации и редактирования знаний*, так и любых других sc-элементов, например, классов команд и даже самих *естественных языков*, изображаемых в рамках самого *Компонента переключения языка идентификации отображаемых sc-элементов*.]
- *Компонент переключения внешнего языка визуализации знаний*
  ⇒ *пояснение**:
  [*Компонент переключения внешнего языка визуализации знаний* служит для переключения языка визуализации знаний в текущем окне, отображаемом на *Панели визуализации и редактирования знаний*. В текущей реализации в качестве таких языков по умолчанию поддерживаются SCg-код и SCn-код, а также любые другие языки, входящие во множество *внешних языков визуализации SC-кода*.]
- *Поле поиска sc-элементов по идентификатору*
  ⇒ *пояснение**:

[*Поле поиска sc-элементов по идентификатору* позволяет осуществлять поиск sc-иден|-ти|-фи|-ка|-то|-ров, содержащих подстроку, введенную в данное поле (с учетом регистра). В результате поиска отображается список sc-идентификаторов, содержащих указанную подстроку, при взаимодействии с которыми осуществляется автоматическое задание вопроса “Что это такое?”, аргументом которого является либо для сам sc-элемент, имеющий данный sc-идентификатор (в случае, если указанный sc-идентификатор является основным или системным, и, таким образом, указанный sc-элемент может быть определен однозначно), либо для самого внутреннего файла ostis-системы, являющегося sc-идентификатором (в случае, если данный sc-идентификатор является неосновным).]

- *Панель отображения диалога пользователя с ostis-системой*
  - ⇒ *пояснение**:
    [*Панель отображения диалога пользователя с ostis-системой* отображает упорядоченный по времени список sc-элементов, являющихся знаками действий, которые инициировал пользователь в рамках диалога с ostis-системой путем взаимодействия с изображениями соответствующих классов команд (то есть, если действие было инициировано другим способом, например, путем его явного инициирования через создание дуги принадлежности множеству *инициированных действий* в sc.g-редакторе, то на данной панели оно отображено не будет). При взаимодействии пользователя с любым из изображенных знаков действий на *Панели визуализации и редактирования знаний* отображается окно, содержащее результат выполнения данного *действия* на том языке визуализации, на котором он был отображен, когда пользователь просматривал его в последний (предыдущий) раз. Таким образом, в текущей реализации данная панель может работать только в том случае, если инициированное пользователем действие предполагает явно представленный в памяти результат данного действия. В свою очередь, из этого следует, что в настоящее время данная панель, как и в целом *Реализация интерпретатора sc-моделей пользовательских интерфейсов*, позволяет работать с системой только в режиме диалога ”вопрос-ответ”.]
- *Панель визуализации и редактирования знаний*
  - ⇒ *пояснение**:
    [*Панель визуализации и редактирования знаний* отображает окна, содержащие sc-текст, представленный на некотором языке из множества *внешних языков визуализации SC-кода* и, как правило, являющийся результатом некоторого действия, инициированного пользователем. Если соответствующий визуализатор поддерживает возможность редактирования текстов соответствующего естественного языка, то он одновременно является также и редактором.]
  - ⇒ *компонент программной системы**:
    - *Визуализатор sc.n-текстов*
    - *Визуализатор и редактор sc.g-текстов*
  - ⇒ *примечание**:
    [При необходимости пользовательский интерфейс каждой конкретной ostis-системы может быть дополнен визуализаторами и редакторами различных внешних языков, которые в текущей версии *Реализации интерпретатора sc-моделей пользовательских интерфейсов* будут также располагаться на *Панели визуализации и редактирования знаний*.]

/* Раздел **/
Библиографический раздел программного варианта реализации платформы интерпретации sc-моделей компьютерных систем
⊃=
{

ArangoDB
⇒ *библиографическая ссылка**:
[ArangoDB]

Diskrete_Math
⇒ *библиографическая ссылка**:
[Diskrete_Math]

Grakn
⇒ *библиографическая ссылка**:
[Grakn]

Ivashenko2015
⇒ *библиографическая ссылка**:
[Ivashenko2015]

Koronchik2013
⇒ *библиографическая ссылка**:
[Koronchik2013]

Neo4j
⇒ *библиографическая ссылка**:
[Neo4j]

OrientDB
⇒ *библиографическая ссылка**:
[OrientDB]

Sesame
⇒ *библиографическая ссылка**:
[Sesame]

Shunkevich2018
⇒ *библиографическая ссылка**:
[Shunkevich2018]

Virtuoso
⇒ *библиографическая ссылка**:
[Virtuoso]

sc_web
⇒ *библиографическая ссылка**:
[sc_web]

} /* Завершили Раздел "*Библиографический раздел программного варианта реализации платформы интерпретации sc-моделей компьютерных систем*" */