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. 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-эл* , ArangoDB *ArangoDB-эл* , OrientDB *OrientDB-эл* , Grakn *Grakn-эл* и др.), а также так называемые rdf-хранилища (Virtuoso *Virtuoso-эл* , 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 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 *Ивашенко.В.П..ПредставССАОСОВСМП-2015ст* , 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 *Shunkevich.D.V.AgentOMMTCPSDIS 2018ст* . 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.

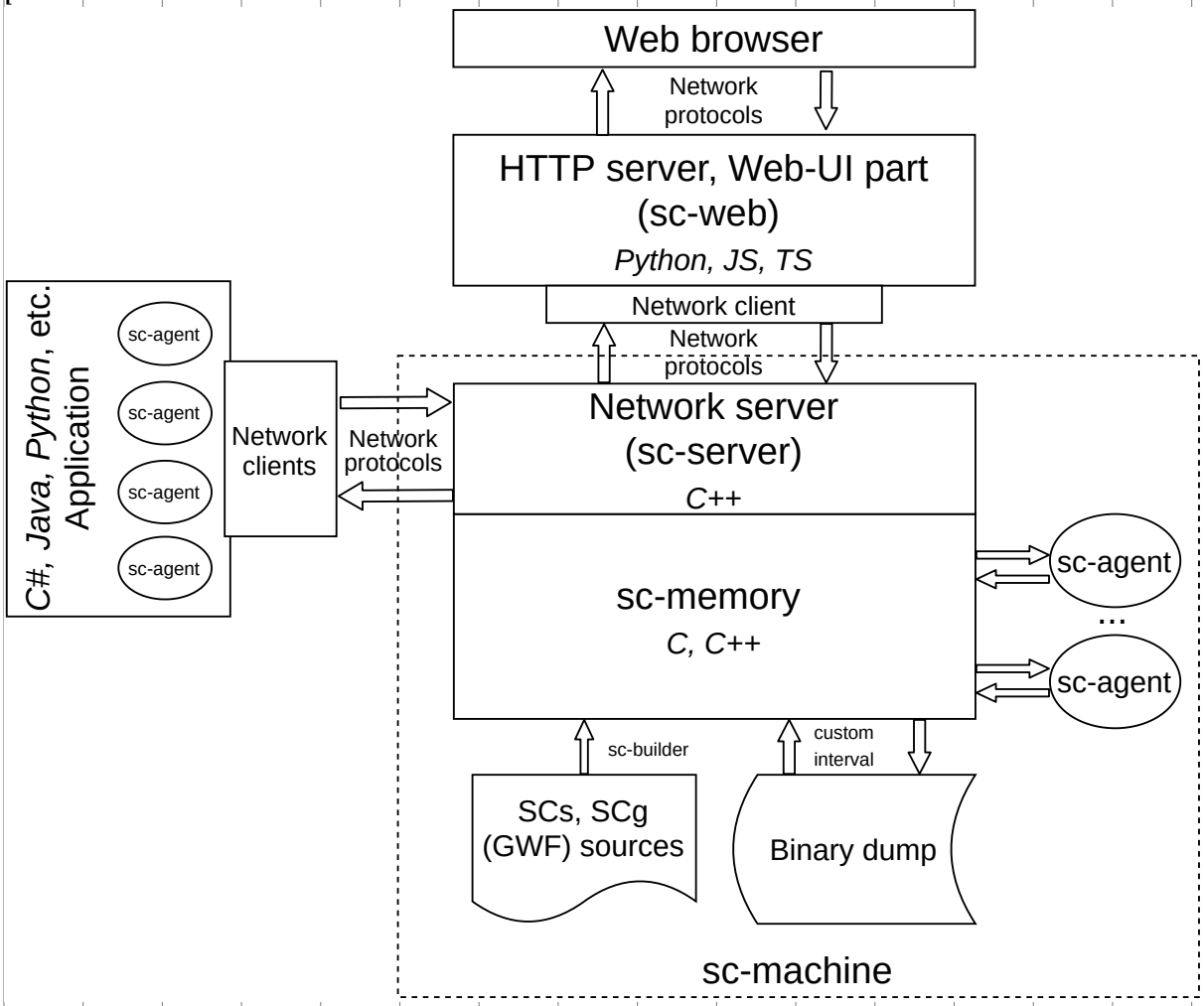
В то же время, взаимодействие клиентской и серверной части организовано таким образом, что web-интерфейс может быть легко заменен на настольный или мобильный интерфейс, как универсальный, так и специализированный.

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), 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.]

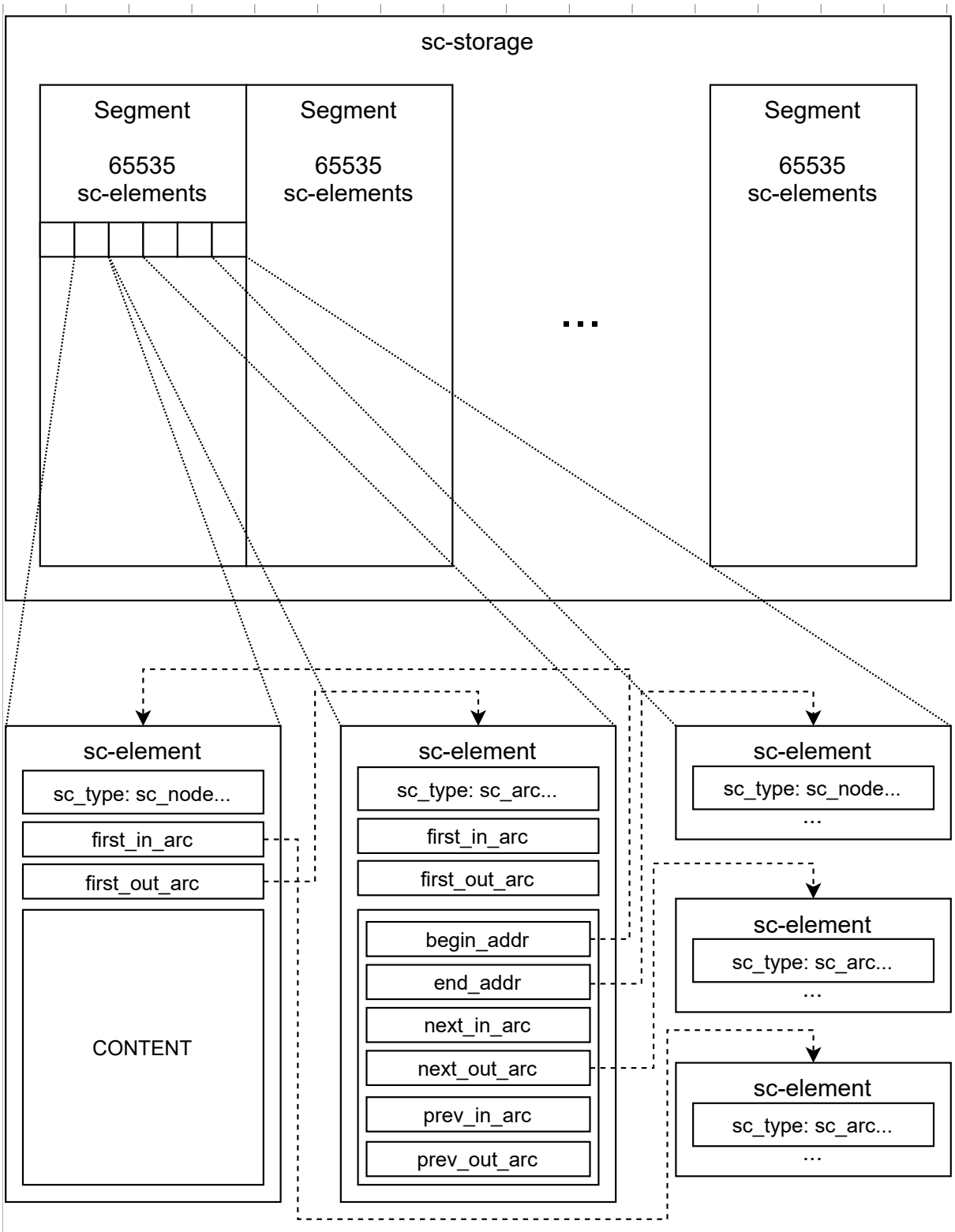
Программная модель sc-памяти
:= [sc-machine]
:= [Программная модель семантической памяти, реализованная на основе традиционной линейной памяти и включающая средства хранения sc-конструкций и базовые средства для обработки этих конструкций, в том числе удаленного доступа к ним посредством соответствующих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

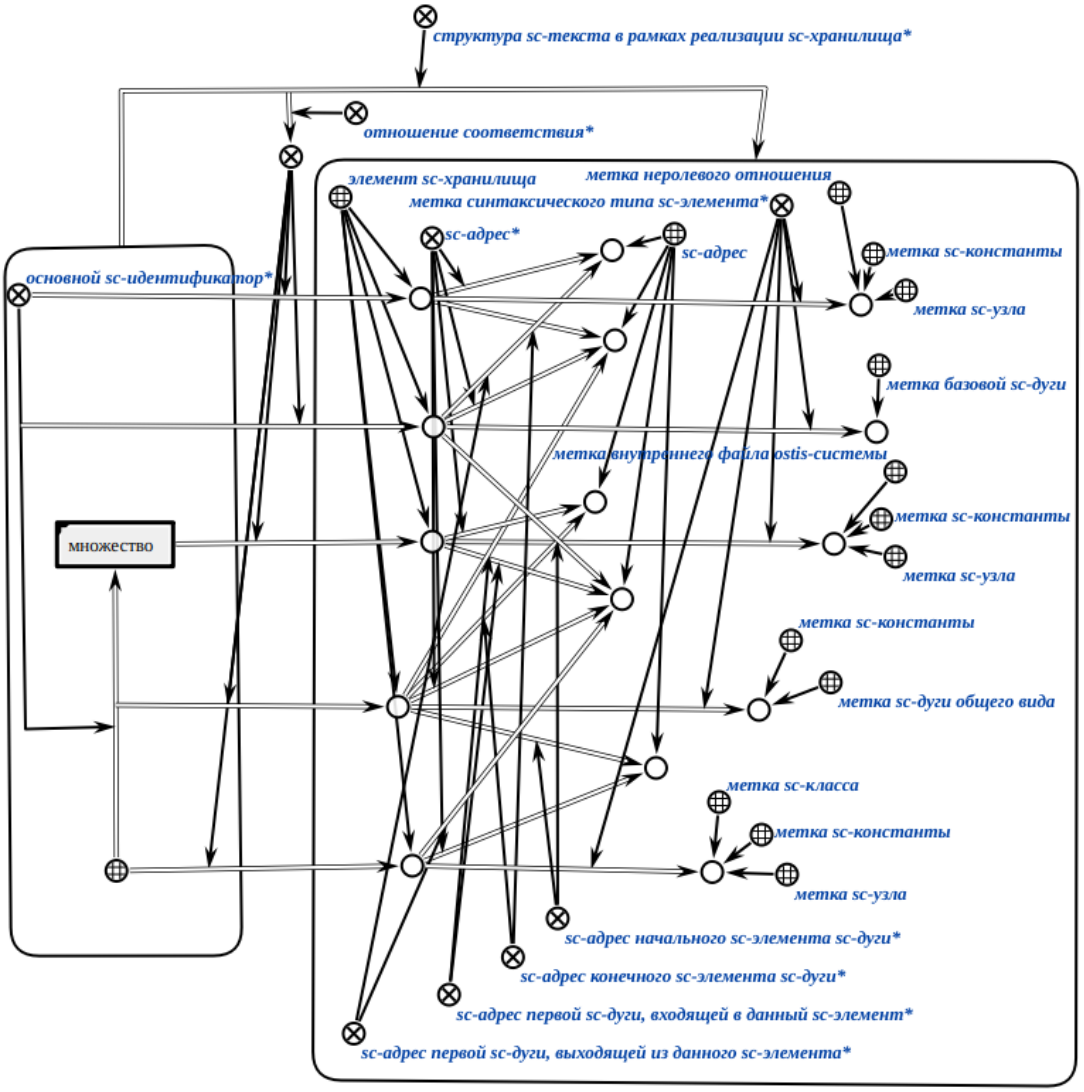
элемент sc-хранилища

⇒ *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**

⇒ *примечание**:

[Для каждого sc-адреса можно взаимно однозначно поставить в соответствие некоторый хэш, полученный в результате применения специальной хэш-функции над этим sc-адресом. Хэш является неотрицательным целым числом и является результатом преобразования номера сегмента sc-хранилища si, в котором располагается sc-элемент, и номера этого sc-элемента sc-хранилища ei в рамках этого сегмента si. В рамках sc-хранилища используется единственная хеш-функция для получения хеша sc-адреса sc-элемента и задаётся как $f(si, ei) = si << 16 \vee ei \wedge 0xffff$, где операция << - операция логического битового сдвига влево левого аргумента на количество единиц, заданное правым аргументом, относительно этой операции, операция $\vee$ - операция логического ИЛИ, операция $\wedge$ - операция логического И, число $0xffff$ - число 65535, представленное в шестнадцатеричном виде и обозначающее максимальное количество sc-элементов в одном сегменте sc-хранилища.]

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 соответствуюещго 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

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 базового набора платформенно-зависимых sc-агентов и их общих компонентов

:= [sc-kpm]

⇒ *компонент программной системы**:

- *Реализация базового набора поисковых sc-агентов*
  - ⇒ *используемый язык программирования**:
    - *C*
  - ⇒ *компонент программной системы**:
    - *Реализация Абстрактного sc-агента поиска семантической окрестности заданной сущности*
    - *Реализация Абстрактного sc-агента поиска всех сущностей, частных по отношению к заданной*
    - *Реализация Абстрактного sc-агента поиска всех сущностей, общих по отношению к заданной*
    - *Реализация Абстрактного sc-агента поиска всех sc-идентификаторов, соответствующих заданной сущности*
    - *Реализация Абстрактного sc-агента поиска базовых sc-дуг, инцидентных заданному sc-элементу*
      - ⇒ *компонент программной системы**:
        - *Реализация Абстрактного sc-агента поиска базовых sc-дуг, входящих в заданный sc-элемент*
        - *Реализация Абстрактного sc-агента поиска базовых sc-дуг, выходящих из заданного sc-элемента*
        - *Реализация Абстрактного sc-агента поиска базовых sc-дуг, входящих в заданный sc-элемент, с указанием множеств, которым принадлежат эти sc-дуги*
        - *Реализация Абстрактного sc-агента поиска базовых sc-дуг, выходящих из заданного sc-элемента, с указанием множеств, которым принадлежат эти sc-дуги*
- *Реализация базового механизма сборки информационного мусора*
  - ⇒ *используемый язык программирования**:
    - *C*
  - ⇒ *примечание**:

    [Текущая реализация механизма сборки информационного мусора содержит один sc-агент, реагирующий на явное добавление какого-либо sc-элемента во множество “информационный мусор” и осуществляющий физическое удаление этого sc-элемента из sc-памяти]
- *Реализация базового набора интерфейсных sc-агентов*
  - ⇒ *используемый язык программирования**:

- *C++*

⇒ *компонент программной системы**:
- *Реализация Абстрактного sc-агента обработки команд пользовательского интерфейса*
- *Реализация Абстрактного sc-агента трансляции из внутреннего представления знаний во промежуточный транспортный формат*

⇒ *примечание**:

[В настоящее время используется подход, при котором независимо от формы внешнего представления информации, информация хранимая в sc-памяти вначале транслируется в промежуточный транспортный формат на базе JSON, который затем обрабатывается sc-агентами пользовательского интерфейса, входящими в состав *Реализации интерпретатора sc-моделей пользовательских интерфейсов*]

/* Сегмент **********/

Описание реализации подсистемы взаимодействия с внешней средой с использованием сетевых языков
⊃=
{

Реализация подсистемы взаимодействия с внешней средой с использованием сетевых языков
⇒ *компонент программной системы**:
- *Реализация подсистемы взаимодействия с внешней средой с использованием сетевых языков на основе языка 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++);
- □ 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 универсального языка, не зависящего от средств реализации того или иного компонента или системы. В качестве такого языка был разработан строковый язык на базе языка JSON.]

Описание подсистемы взаимодействия с sc-памятью на основе языка JSON
⊃=
{

Реализация подсистемы взаимодействия с sc-памятью на основе языка JSON
⇒ *пояснение**:

[Реализация подсистемы взаимодействия с sc-памятью на основе языка JSON позволяет ostis-системам взаимодействовать с системами из внешней среды на основе общепринятого транспортного формата передачи данных JSON и предоставляет API для доступа к sc-памяти платформы интерпретации sc-моделей.]

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

- *Java*

⇒ *используемый язык**:
- *SC-JSON-код*

⇒ *архитектура**:
Клиент-серверная архитектура

⇐ *реализация**:
Подсистема взаимодействия с sc-памятью на основе языка JSON

:= [Подсистема взаимодействия с sc-памятью на основе формата JSON]

:= [Подсистема взаимодействия с sc-памятью на основе транспортного формата передачи данных JSON]

∈ *многократно используемый компонент ostis-систем*

∈ *неатомарный многократно используемый компонент ostis-систем*

∈ *зависимый многократно используемый компонент ostis-систем*

⇒ *автор**:
- *Корончик Д. Н.*
- *Шункевич Д. В.*
- *Зотов Н. В.*
- *Загорский А. Г.*

⇒ *пояснение**:
[Взаимодействие с sc-памятью обеспечивается с помощью передачи информации на ***SC-JSON-коде*** и ведётся, с одной стороны, между сервером, являющегося частью ostis-системы, написанным на том же языке реализации этой ostis-системы и имеющим доступ к её sc-памяти, и с другой стороны множеством клиентом, которым известно о наличии сервера в пределах сети их использования.]

⇒ *примечание**:
[Осмысленные фрагменты текстов ***SC-JSON-кода*** представляют семантически корректную структуру сущностей и связей между ними.]

⇒ *примечание**:
[С помощью подсистемы взаимодействия с sc-памятью на основе языка JSON можно взаимодействовать с ostis-системой на таком же множестве возможных операций, как и в случае, если бы взаимодействие происходило (непосредственно) напрямую, на том же языке реализации платформы интерпретации sc-моделей компьютерных систем. При этом результат работы отличается только скоростью обработки информации.]

⇒ *декомпозиция программной системы**:
{
- *Серверная система на основе Websocket, обеспечивающая доступ к sc-памяти платформы интерпретации sc-моделей при помощи команд SC-JSON-кода*

Множество клиентских систем, подключаемых и взаимодействующих с *Серверной системой на основе Websocket, обеспечивающей доступ к sc-памяти платформы интерпретации sc-моделей при помощи команд SC-JSON-кода*

⇒ *декомпозиция программной системы**:
{
- *Клиентская система, подключаемая и взаимодействующая с SC-сервером, реализованная на языке программирования Python*
- *Клиентская система, подключаемая и взаимодействующая с SC-сервером, реализованная на языке программирования TypeScript*
- *Клиентская система, подключаемая и взаимодействующая с SC-сервером, реализованная на языке программирования C#*
- *Клиентская система, подключаемая и взаимодействующая с SC-сервером, реализованная на языке программирования Java*

}

}

SC-JSON-код

:= [Semantic JSON-code]

:= [Semantic JavaScript Object Notation code]

:= [Язык внешнего смыслового представления знаний для взаимодействия с ostis-системами на основе языка JSON]

:= [Метаязык, являющийся подмножеством языка JSON и обеспечивающий внешнее представление и структуризацию *sc-текстов*, используемых ostis-системой в процессе своего функционирования и взаимодействия со внешней средой.]

⇒ *часто используемый неосновной внешний идентификатор sc-элемента**:

[sc-json-текст]
∈ *имя нарицательное*
∈ *абстрактный язык*
∈ *линейный язык*
⊂ *JSON*
⇒ *автор**:
- *Зотов Н. В.*
- *Корончик Д. Н.*

⇒ *принципы, лежащие в основе**:
⟨
- *Тексты, описываемые на языке внешнего представления знаний* ***SC-JSON-код*** *представляют собой линейную структуру, представляемую в виде линейного строкового текста и состоящую из набора корректных осмысленных команд, записанных в виде sc-json-пар вида {отношение: объект}, где отношением выступает знак квазибинарного отношения, состоящего из пар вида {субъект: объект}, где объектом выступает знак, обозначаемый предложением, включающее такие пары, а субъектом - sc-json-объекты: sc-json-литерал, sc-json-списки sc-json-объектов, sc-json-предложения, состоящие из sc-json-списков sc-json-объектов.*
- *Тексты* ***SC-JSON-кода*** *представляют собой sc-json-команды. Каждая команда представляет собой json-объект, в котором указываются уникальный идентификатор команды, тип этой команды и ее аргументы. С каждой командой ассоциируется ответ на эту команду. Ответ на команду представляет собой команду, в котором указываются идентификатор команды, ее статус (выполнена успешно/безуспешно) и результаты. Структура аргументов и результатов команды определяется типом команды. Для каждого ответа существует запрос.*

⟩
⇒ *достоинство**:
- [Язык 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 ***SC-JSON-код***.]
- [Реализация подсистемы взаимодействия со внешней средой на базе ***SC-JSON-код*** не накладывает принципиальных ограничений на объем (длину) каждой команды, в отличие от других бинарных протоколов. 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 создания sc-конструкции, изоморфной заданному sc-шаблону*, и *команда поиска sc-конструкций, изоморфных заданному sc-шаблону*.]

⇒ *примечание**:
[Можно сказать, что язык 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.]

Синтаксис SC-JSON-кода
⊃=
{

Синтаксис SC-JSON-кода
⇒ *примечание**:
[*Синтаксис SC-JSON-кода* задается: (1) *Алфавитом SC-JSON-кода*, (2) Грамматикой SC-JSON-кода]
⇐ *синтаксис**:
SC-JSON-код

Синтаксическая классификация элементов SC-JSON-кода
⊃=
{

SC-JSON-код

⇐ *семейство подмножеств**:
sc-json-предложение
⊂ *json-список json-пар*
⇐ *семейство подмножеств**:
*sc-json-пара**
⇐ *декартово произведение**:
⟨• *sc-json-строка*
• *sc-json-объект*
⇒ *разбиение**:
{• *sc-json-список*
• *sc-json-пара*
• *sc-json-литерал*
⇒ *разбиение**:
{• *sc-json-строка*
• *sc-json-число*
}
}
⟩
⇒ *разбиение**:
{• *команда на SC-JSON-коде*
• *ответ на команду на SC-JSON-коде*
}
}
/* Завершили представление *Синтаксической классификации элементов SC-JSON-кода* */

Алфавит SC-JSON-кода^
:= [Множество всех возможных символов в SC-JSON-коде]
⇒ *пояснение**:
[Поскольку SC-JSON-код является линейным строковым языком представления знаний, то его алфавит включает объединение алфавитов всех языков, тексты на которых могут представлять внешние идентификаторы и/или содержимое файлов ostis-системы, множество всех цифр и множество всех других специальных символов.]
⇐ *алфавит**:
SC-JSON-код
⇒ *примечание**:
[Последовательности знаков алфавита могут образовывать sc-json ключевые слова, sc-json-пары, sc-json-предложения из sc-json-пар и sc-json-тексты из sc-json-предложений.]

SC-JSON-код
⇒ *синтаксические правила**:
•
[Каждое правило *Грамматики SC-JSON-кода* описывает корректный с точки зрения *Синтаксиса SC-JSON-кода* порядок sc-json-объектов в sc-json-предложении. Совокупность правил *Грамматики SC-JSON-кода* описывает корректный с точки зрения *Синтаксиса SC-JSON-кода* порядок sc-json-предложений в sc-json-тексте.]
•
[Каждое sc-json-предложение является sc-json-списком, состоящим из sc-json-пар и представляет собой команду или ответ на эту команду.]
•
[Каждое *команда (ответ на команду) на SC-JSON-коде* состоит из заголовка, включающего sc-json-пары описания самой команды (ответа на команду), и сообщения, различного для каждого класса команд (ответов на команды). Сообщение команды (ответа на команду) на SC-JSON-коде обычно представляет собой список sc-json-объектов и может не ограничиваться по мощности.]
•
[Каждая sc-json-пара состоит из двух элементов: ключевого слова и некоторого другого sc-json-объекта, ассоциируемого с этим ключевым словом. Набор ключевых слов в sc-json-парах определяется конкретным классом команд (ответов на команды) на SC-JSON-коде. Sc-json-пара начинается знаком открывающейся фигурной скобки "{" и заканчивается знаком закрывающейся фигурной скобки "}". Ключевое слово и sc-json-объект, ассоциируемый с ним, разделяются при помощи знака двоеточия ":".]

- [Sc-json-строки, записанные в sc-json-текстах, начинаются и заканчиваются знаком двух ковычек “.]
- [Sc-json-списки, состоящие не из sc-json-пар, начинаются знаком открывающейся квадратной скобки "[" и заканчиваются знаком закрывающейся квадратной скобки "]". Sc-json-объекты в sc-json-списках разделяются запятыми ",".]

Грамматика SC-JSON-кода

:= [Множество всех возможных правил, используемых при построении команд и ответов на них на SC-JSON-коде]

⇒ *пояснение**:

[Каждой команде *SC-JSON-кода* однозначно соответствует правило грамматики *SC-JSON-кода*.]

⇐ *грамматика**:

SC-JSON-код

⇒ *пояснение**:

[Правила *Грамматики SC-JSON-кода* позволяют правильно составить команду на SC-JSON-коде. Каждое правило грамматики *SC-JSON-кода* представляется в виде правила на *Языке описания грамматик ANTLR* и его интерпретации на естественном языке.]

∋ *ключевой sc-элемент′*:

Правило, задающее синтаксис команд на SC-JSON-коде

⇔ *семантическая эквивалентность**:

[

```
sc_json_command
 : '{'
     '"id"' ':' NUMBER ','
     sc_json_command_type_and_payload
   '}'
 ;

sc_json_command_type_and_payload
 : sc_json_command_create_elements
 | sc_json_command_check_elements
 | sc_json_command_delete_elements
 | sc_json_command_handle_keynodes
 | sc_json_command_handle_link_contents
 | sc_json_command_search_template
 | sc_json_command_generate_template
 | sc_json_command_handle_events
 ;
```

]

∈ *Язык описания грамматики языков ANTLR*

⇒ *интерпретация**:

[Класс *команд на SC-JSON-коде* включает *команду создания sc-элементов*, *команду получения соответствующих типов sc-элементов*, *команду удаления sc-элементов*, *команду обработки ключевых sc-элементов*, *команду обработки содержимого файлов ostis-системы*, *команду поиска sc-конструкций, изоморфных заданному sc-шаблону*, *команду генерации sc-конструкции, изоморфной заданному sc-шаблону*, и *команду обработки sc-событий*. В *команду на SC-JSON-коде* включаются идентификатор этой команды, тип и сообщение.]

⇐ *синтаксическое правило**:

команда на SC-JSON-коде

∋ *ключевой sc-элемент′*:

Правило, задающее синтаксис ответа на команду на SC-JSON-коде

⇔ *семантическая эквивалентность**:

[

```
sc_json_command_answer
 : '{'
     '"id"' ':' NUMBER ','
     '"status"' ':' BOOL ','
     sc_json_command_answer_payload
  '}'
 ;

sc_json_command_answer_payload
 : sc_json_command_answer_create_elements
 | sc_json_command_answer_check_elements
 | sc_json_command_answer_delete_elements
 | sc_json_command_answer_handle_keynodes
 | sc_json_command_answer_handle_link_contents
 | sc_json_command_answer_search_template
 | sc_json_command_answer_generate_template
 | sc_json_command_answer_handle_events
 ;
```

]

∈ *Язык описания грамматики языков ANTLR*

⇒ *интерпретация**:

[Класс *ответов на команды на SC-JSON-коде* включает *ответ на команду создания sc-элементов*, *ответ на команду получения соответствующих типов sc-элементов*, *ответ на команду удаления sc-элементов*, *ответ на команду обработки ключевых sc-элементов*, *ответ на команду обработки содержимого файлов ostis-системы*, *ответ на команду поиска sc-конструкций, изоморфных заданному sc-шаблону*, *ответ на команду генерации sc-конструкции, изоморфной заданному sc-шаблону*, и *ответ на команду обработки sc-событий*. В *ответ на команду на SC-JSON-коде* включаются идентификатор соответствующей команды, статус обработки ответа и ответное сообщение.]

⇐ *синтаксическое правило**:

ответ на команду на SC-JSON-коде

∋ *Правило, задающее синтаксис команды создания sc-элементов*

⇔ *семантическая эквивалентность**:

[

```
sc_json_command_create_elements
  : '"type"' ':' '"create_elements"' ','
    '"payload"' ':'
    '['(
        '{'
            '"el"' ':' '"node"' ','
            '"type"' ':' SC_NODE_TYPE ','
        '}' ','
        |
        '{'
            '"el"' ':' '"link"' ','
            '"type"' ':' SC_LINK_TYPE ','
            '"content"' ':' NUMBER_CONTENT | STRING_CONTENT
        '}' ','
        |
        '{'
            '"el"' ':' '"edge"' ','
            '"type"' ':' SC_EDGE_TYPE ','
            '"src"' ':' (
            '{'
                '"type"' ':' '"ref"' ','
                '"value"' ':' NUMBER
            '}' ','
            |
            '{'
                '"type"' ':' '"addr"' ','
                '"value"' ':' SC_ADDR_HASH
            '}' ','
            |
            '{'
                '"type"' ':' '"idtf"' ','
                '"value"' ':' SC_NODE_IDTF
            '}' ','
            )
            '"trg"' ':' (
            '{'
                '"type"' ':' '"ref"' ','
                '"value"' ':' NUMBER
            '}' ','
            |
            '{'
                '"type"' ':' '"addr"' ','
                '"value"' ':' SC_ADDR_HASH
            '}' ','
            |
            '{'
                '"type"' ':' '"idtf"' ','
                '"value"' ':' SC_NODE_IDTF
            '}' ','
            )
        '}' ','
        |
        scs_text ','
    )*']' ','
  ;
```

⇒ *интерпретация**:
[В сообщении *команды создания sc-элементов* представляется список описаний создаваемых sc-элементов. Такими sc-элементами могут быть sc-узел, sc-дуга, sc-ребро или файл ostis-системы. Тип sc-элемента указывается в паре с ключевым словом "el": для sc-узла sc-json-тип элемент представляется как "node", для sc-дуги и sc-ребра - "edge", для файла ostis-системы - "link". Метки типов sc-элементов уточняются в соответствующих им описаниях в сообщении команды в паре с ключевым словом "type". Если создаваемым sc-элементом является файл ostis-системы, то дополнительно указывается содержимое этого файла ostis-системы в паре с ключевым словом "content", если создаваемым sc-элементом является sc-дуга или sc-ребро, то указываются описания sc-элементов, из которых они выходят, и sc-элементов, в которые они входят. Описание таких sc-элементов состоят из двух пар: первая пара указывает на способ ассоциации с sc-элементом и представляется как "addr" или "idtf" или "ref" в паре с ключевым словом "type", вторая пара - то, по чему происходит ассоциация с этим sc-элементом: его хэшу, системному идентификатору или номеру в массиве создаваемых sc-элементов - в паре с ключевым словом "value".]

⇐ *синтаксическое правило**:
команда создания sc-элементов

∋ *Правило, задающее синтаксис ответа на команду создания sc-элементов*

⇔ *семантическая эквивалентность**:
[

```
sc_json_command_answer_create_elements
  : '"payload"' ':'
    '['
      (SC_ADDR_HASH ',')*
    ']' ','
  ;
```

]

∈ *Язык описания грамматики языков ANTLR*

⇒ *интерпретация**:
[Сообщением *ответа на команду создания sc-элементов* является список хэшей созданных sc-элементов, соответствующих описаниям *команды создания sc-элементов* со статусом 1, в случае успешной обработки команды.]

⇐ *синтаксическое правило**:
ответ на команду создания sc-элементов

∋ *Правило, задающее синтаксис команды создания sc-элементов по фрагменту SCs-текста*

⇔ *семантическая эквивалентность**:
[

```
sc_json_command_create_elements_by_scs
  : '"type"' ':' '"create_elements_by_scs"' ','
    '"payload"' ':'
    '['(
      scs_text ','
    )*']' ','
  ;
```

]

∈ *Язык описания грамматики языков ANTLR*

⇒ *интерпретация**:
[В списке описаний создаваемых sc-элементов сообщения этой команды вместо описания создаваемого отдельного sc-элемента указывается фрагмент SCs-текста.]

⇐ *синтаксическое правило**:
команда создания sc-элементов по фрагменту SCs-текста

∋ *Правило, задающее синтаксис ответа на команду создания sc-элементов по фрагменту SCs-текста*

⇔ *семантическая эквивалентность**:
[

```
sc_json_command_answer_create_elements_by_scs
  : '"type"' ':' '"check_elements"' ','
    '"payload"' ':'
    '['
      (BOOL ',')*
    ']' ','
  ;
```

]
∈ *Язык описания грамматики языков ANTLR*
⇒ *интерпретация**:
[Сообщением *ответа на команду создания sc-элементов* является список результатов обработки переданных SCs-текстов. Нулевой статус говорит о том, что обработка соотвествующего SCs-текста завершилась безуспешно.]
⇐ *синтаксическое правило**:
ответ на команду создания sc-элементов по фрагменту SCs-текста

∋ *Правило, задающее синтаксис команды получения соответствующих типов sc-элементов*
⇔ *семантическая эквивалентность**:
[

```
sc_json_command_check_elements
  : '"type"' ':' '"check_elements"' ','
    '"payload"' ':'
    '['
      (SC_ADDR_HASH ',')*
    ']' ','
  ;
```

]
∈ *Язык описания грамматики языков ANTLR*
⇒ *интерпретация**:
[Сообщением *команды получения соответствующих типов sc-элементов* является списком хэшей sc-элементов, у которых необходимо получить синтаксические типы.]
⇐ *синтаксическое правило**:
команда получения соответствующих типов sc-элементов

∋ *Правило, задающее синтаксис ответа на команду получения соответствующих типов sc-элементов*
⇔ *семантическая эквивалентность**:
[

```
sc_json_command_answer_check_elements
  : '"payload"' ':'
    '['
      (SC_ADDR_TYPE ',')*
    ']' ','
  ;
```

]
∈ *Язык описания грамматики языков ANTLR*
⇒ *интерпретация**:
[Сообщением *ответа на команду получения соответствующих типов sc-элементов* является список типов проверенных sc-элементов, соответствующих описаниям *команды получения соответствующих типов sc-элементов* со статусом 1, в случае успешной обработки команды.]
⇐ *синтаксическое правило**:
ответ на команду получения соответствующих типов sc-элементов

∋ *Правило, задающее синтаксис команды удаления sc-элементов*
⇔ *семантическая эквивалентность**:
[

```
sc_json_command_delete_elements
  : '"type"' ':' '"delete_elements"' ','
    '"payload"' ':'
    '['
      (SC_ADDR_HASH ',')*
    ']' ','
  ;
```

]
∈ *Язык описания грамматики языков ANTLR*
⇒ *интерпретация**:
[Сообщением *команды удаления sc-элементов* является список хэшей sc-элементов, которые необходимо удалить из sc-памяти.]
⇐ *синтаксическое правило**:
команда удаления sc-элементов
∋ *Правило, задающее синтаксис ответа на команду удаления sc-элементов*
⇔ *семантическая эквивалентность**:
[

```
sc_json_command_answer_delete_elements
  : '"payload"' ':'
    '["]' ','
  ;
```

]
∈ *Язык описания грамматики языков ANTLR*
⇒ *интерпретация**:
[Сообщение *ответа на команду удаления sc-элементов* является пустым со статусом 1, в случае успешной обработки команды.]
⇐ *синтаксическое правило**:
ответ на команду удаления sc-элементов
∋ *Правило, задающее синтаксис команды обработки ключевых sc-элементов*
⇔ *семантическая эквивалентность**:
[

```
sc_json_command_handle_keynodes
  : '"type"' ':' '"keynodes"' ','
    '"payload"' ':'
    '['(
      '{'
        '"command"' ':' '"find"' ','
        '"idtf"' ':' SC_NODE_IDTF ','
      '}' ','
      |
      '{'
        '"command"' ':' '"resolve"' ','
        '"idtf"' ':' SC_NODE_IDTF ','
        '"elType"' ':' SC_ADDR_TYPE
      '}' ','
    )']' ','
  ;
```

]
∈ *Язык описания грамматики языков ANTLR*
⇒ *интерпретация**:
[Сообщение *команды обработки ключевых sc-элементов* может включать описание ключевых sc-элементов, которые необходимо найти и/или разрешить по их идентификаторам.

Такое деление осуществляется с помощью подкоманд, содержащихся в сообщении команды. Идентификаторами подкоманд могут быть "find" и "resolve" соответственно, стоящие в паре с ключевым словом "command". Описание искомого sc-элемента команды "find" включает системный идентификатор sc-элемента, по которому необходимо найти этот sc-элемент, стоящий в паре с ключевым словом "idtf". Описание разрешаемого sc-элемента команды "resolve" включает системный идентификатор sc-элемента, по которому необходимо найти этот sc-элемент, либо в случае безуспешного поиска создать sc-элемент некоторого типа, указанного в его описании в паре с ключевым словом "elType", и установить для него системный идентификатор, по которому была произведена попытка найти другой sc-элемент.]

⇐ *синтаксическое правило**:
команда обработки ключевых sc-элементов

∋ *Правило, задающее синтаксис ответа на команду обработки ключевых sc-элементов*

⇔ *семантическая эквивалентность**:

[

```
sc_json_command_answer_handle_keynodes
  : '"payload"' ':'
    '['
      (SC_ADDR_HASH ',')*
    ']' ','
  ;
```

]

∈ *Язык описания грамматики языков ANTLR*

⇒ *интерпретация**:

[Сообщением *ответа на команду обработки ключевых sc-элементов* является список хэшей sc-элементов, соответствующих описаниям *команды обработки ключевых sc-элементов* со статусом 1, в случае успешной обработки команды.]

⇐ *синтаксическое правило**:
ответ на команду обработки ключевых sc-элементов

∋ *Правило, задающее синтаксис команды обработки содержимого файлов ostis-системы*

⇔ *семантическая эквивалентность**:

[

```
sc_json_command_handle_link_contents
  : '"type"' ':' '"content"' ','
    '"payload"' ':'
    '['(
        '{'
          '"command"' ':' '"set"' ','
          '"type"' ':' SC_LINK_CONTENT_TYPE ','
          '"data"' ':' NUMBER_CONTENT | STRING_CONTENT ','
          '"addr"' ':' SC_ADDR_HASH ','
        '}' ','
        |
        '{'
          '"command' ':' '"get"' ','
          '"addr"' ':' SC_ADDR_HASH ','
        '}' ','
        |
        '{'
          '"command' ':' '"find"' ','
          '"data"' ':' NUMBER_CONTENT | STRING_CONTENT ','
        '}' ','
        |
        '{'
           '"command' ':' '"find_by_substr"' ','
           '"data"' ':' NUMBER_CONTENT | STRING_CONTENT ','
        '}' ','
      )*']' ','
  ;
```

]

∈ *Язык описания грамматики языков ANTLR*

⇒ *интерпретация**:

[Сообщение *команды обработки содержимого файлов ostis-системы* может включать описание ключевых файлов ostis-системы, которые необходимо найти по их содержимому или части этого содержимого, для которых необходимо установить содержимое разрешить и/или у которых необходимо получить содержимое. Как и в *Правиле, задающее синтаксис команды обработки ключевых sc-элементов* деление осуществляется с помощью подкоманд, содержащихся в сообщении команды. Идентификаторами подкоманд могут быть "find", "find_by_substr", "set" и "get" соответственно, стоящие в паре с ключевым словом "command". В описаниях команд "set" и "get" указывается хэш файла ostis-системы в паре с ключевым словом "addr". В описаниях команд "set", "find" и "find_by_substr" указывается содержимое файла ostis-системы в паре с ключевым словом "data". Дополнительно в описании подкоманды "set" указывается тип устанавливаемого содержимого файла ostis-системы.]

⇐ *синтаксическое правило**:

команда обработки содержимого файлов ostis-системы

∋ *Правило, задающее синтаксис ответа на команду обработки содержимого файлов ostis-системы*

⇔ *семантическая эквивалентность**:

[

```
sc_json_command_answer_handle_link_contents
  : '"payload"' ':'
    '['(
      BOOL
      |
      '{'
        '"value"' ':' NUMBER_CONTENT | STRING_CONTENT ','
        '"type"' ':' SC_LINK_CONTENT_TYPE ','
      '}' ','
      |
      '['
        (SC_ADDR_HASH ',')*
      ']' ','
    )*']' ','
  ;
```

]

∈ *Язык описания грамматики языков ANTLR*

⇒ *интерпретация**:

[Сообщением *ответа на команду обработки содержимого файлов ostis-системы* является список, состоящий из булевого результата установки содержимого в файл ostis-системы и/или найденных файлов ostis-системы по их содержимому и/или описания полученного содержимого файлов ostis-системы, соответствующих описаниям *команды обработки содержимого файлов ostis-системы* со статусом 1, в случае успешной обработки команды.]

⇐ *синтаксическое правило**:

ответ на команду обработки содержимого файлов ostis-системы

∋ *Правило, задающее синтаксис команды поиска sc-конструкций, изоморфных заданному sc-шаблону*

⇔ *семантическая эквивалентность**:

[

```
sc_json_command_search_template
  : '"type"' ':' '"search_template"' ','
    sc_json_command_template_payload
  ;
```

]

∈ *Язык описания грамматики языков ANTLR*

⊂ *Правило, задающее синтаксис сообщения команды поиска sc-конструкций, изоморфных заданному sc-шаблону, и команды генерации sc-конструкции, изоморфной заданному sc-шаблону*

⇒ *интерпретация**:

[*Правило, задающее синтаксис команды поиска sc-конструкций, изоморфных заданному sc-шаблону* включает *Правило, задающее синтаксис сообщения команды поиска sc-конструкций, изоморфных заданному sc-шаблону, и команды генерации sc-конструкции, изоморфной заданному sc-шаблону* и описывает команду поиска sc-конструкций по сформированному этой командой sc-шаблону (см. *Правило, задающее синтаксис сообщения команды поиска sc-конструкций, изоморфных заданному sc-шаблону, и команды генерации sc-конструкции, изоморфной заданному sc-шаблону*).]

⇐ *синтаксическое правило**:

команда поиска sc-конструкций, изоморфных заданному sc-шаблону

∋ *Правило, задающее синтаксис ответа на команду поиска sc-конструкций, изоморфных заданному sc-шаблону*

⇔ *семантическая эквивалентность**:

[

```
sc_json_command_answer_search_template
 : '"payload"' ':'
  '{'
    '"addrs"' ':'
    '['
      ('['
        (SC_ADDR_HASH ',')*
      ']' ',')*
    ']' ','
    '"aliases"' ':'
    '{'
      (SC_ALIAS ':' NUMBER ',')*
    '}' ','
  '}' ';'
 ;
```

]
∈ *Язык описания грамматики языков ANTLR*
⇒ *интерпретация**:
[Сообщение *ответа на команду поиска sc-конструкций, изоморфных заданному sc-шаблону* состоит из списка найденных sc-конструкций и отображения псевдонимов sc-элементов на их позиции в тройках sc-шаблона. Ответ имеет статус 1, в случае успешной обработки команды.]

⇐ *синтаксическое правило**:
ответ на команду поиска sc-конструкций, изоморфных заданному sc-шаблону

∋ *Правило, задающее синтаксис команды создания sc-конструкции, изоморфной заданному sc-шаблону*
⇔ *семантическая эквивалентность**:
[

```
sc_json_command_generate_template
 : '"type"' ':' '"generate_template"' ','
  sc_json_command_template_payload
 ;
```

]
∈ *Язык описания грамматики языков ANTLR*
⊂ *Правило, задающее синтаксис сообщения команды поиска sc-конструкций, изоморфных заданному sc-шаблону, и команды генерации sc-конструкции, изоморфной заданному sc-шаблону*
⇒ *интерпретация**:
[*Правило, задающее синтаксис команды создания sc-конструкции, изоморфной заданному sc-шаблону* включает *Правило, задающее синтаксис сообщения команды поиска sc-конструкций, изоморфных заданному sc-шаблону, и команды генерации sc-конструкции, изоморфной заданному sc-шаблону* и описывает команду создания sc-конструкции по сформированному этой командой sc-шаблону (см. *Правило, задающее синтаксис сообщения команды поиска sc-конструкций, изоморфных заданному sc-шаблону, и команды генерации sc-конструкции, изоморфной заданному sc-шаблону*).]

⇐ *синтаксическое правило**:
команда создания sc-конструкции, изоморфной заданному sc-шаблону

∋ *Правило, задающее синтаксис ответа на команду создания sc-конструкции, изоморфной заданному sc-шаблону*
⇔ *семантическая эквивалентность**:
[

```
sc_json_command_answer_generate_template
  : '"payload"' ':'
    '{'
      '"addrs"' ':'
      '['
        (SC_ADDR_HASH ',')*
      ']' ','
      '"aliases"' ':'
      '{'
        (SC_ALIAS ':' NUMBER ',')*
      '}' ','
    '}' ','
  ;
```

]

∈ *Язык описания грамматики языков ANTLR*

⇒ *интерпретация**:

[Сообщение *ответа на команду создания sc-конструкции, изоморфной заданному sc-шаблону* состоит из списка найденных sc-конструкций и отображения псевдонимов sc-элементов на их позиции в тройках sc-шаблона. Ответ имеет статус 1, в случае успешной обработки команды.]

⇐ *синтаксическое правило**:

ответ на команду создания sc-конструкции, изоморфной заданному sc-шаблону

∋ *Правило, задающее синтаксис сообщения команды поиска sc-конструкций, изоморфных заданному sc-шаблону, и команды создания sc-конструкции, изоморфной заданному sc-шаблону*

⇔ *семантическая эквивалентность**:

[

```
sc_json_command_template_payload
  : '"payload"' ':'
    '{'
      '"templ"' ':' ('['
        ('['
          (
          '{'
            '"type"' ':' '"addr"' ','
            '"value"' ':' SC_ADDR_HASH ','
            ('"alias"' ':' SC_ALIAS ',')?
          '}' ','
          |
          '{'
            '"type"' ':' '"type"' ','
            '"value"' ':' SC_ADDR_TYPE ','
            ('"alias"' ':' SC_ALIAS ',')?
          '}' ','
          )
          '{'
            '"type"' ':' '"type"' ','
            '"value"' ':' SC_EDGE_TYPE ','
            ('"alias"' ':' SC_ALIAS ',')?
          '}' ','
          (
          '{'
            '"type"' ':' '"addr"' ','
            '"value"' ':' SC_ADDR_HASH ','
            ('"alias"' ':' SC_ALIAS ',')?
          '}' ','
          |
          '{'
            '"type"' ':' '"type"' ','
            '"value"' ':' SC_ADDR_TYPE ','
            ('"alias"' ':' SC_ALIAS ',')?
          '}' ','
          )
        ']' ',')*
      ']' ','
      |
      scs_text | '{' '"type"' ':' '"addr"' ',' '"value"' ':' SC_ADDR_HASH ',' '}' ','
      | '{' '"type"' ':' '"idtf"' ',' '"value"' ':' SC_NODE_IDTF ',"}' ',')
      '"params"' ':'
      '{'
        (SC_ALIAS ':' (SC_ADDR_HASH | SC_ALIAS) ',')*
      '}' ','
    '}' ","
  ;
```

]

∈ *Язык описания грамматики языков ANTLR*

⇒ *интерпретация**:

[Сообщения *команды поиска sc-конструкций, изоморфных заданному sc-шаблону,* и *команды создания sc-конструкции, изоморфной заданному sc-шаблону* включают описание троек, необходимых для создания sc-шаблона поиска или генерации изоморфных sc-конструкций. Описание каждой тройки sc-шаблона включает описание sc-элементов этой тройки. Описания первого и третьего sc-элементов тройки должны всегда содержать хэш или тип в паре с ключевым словом "value". Если выбран тип, то в паре с ключевым словом "type" указывается "type", если - хэш, то - "addr". Вторым sc-элементом тройки должна быть дуга, для которой всегда указывается тип в паре с ключевым словом "value". Для каждого sc-элемента тройки может указываться псевдоним в паре с "alias". Псевдоним представляет собой строку и может быть использован для ассоциации с sc-элементами в других тройках sc-шаблона, либо ассоциации со значениями переменных sc-шаблона, которые указываются в списке под ключевым словом "params" и могут представлять собой либо хэш sc-элемента, либо его системный идентификатор. Таким образом, в некоторых случаях может отсутствовать необходимость указания хэша или типа sc-элемента. Также вместо списка описаний троек sc-шаблона, может указываться хэш или системный идентификатор sc-структуры, хранящейся в sc-памяти. хэш и системный идентификатор указываются в паре с ключевым словом "value".]

∋ *Правило, задающее синтаксис команды обработки sc-событий*
⇔ *семантическая эквивалентность**:
[
```
sc_json_command_handle_events
  : '"type"' ':' '"events"' ','
    '"payload"' ':'
    '{'
       ('"create"' ':'
       '['
          ('{'
             '"type"' ':' SC_EVENT_TYPE ','
             '"addr"' ':' SC_ADDR_HASH ','
          '}' ',')*
       ']' ',')?
       ('"delete"' ':'
       '['
          (NUMBER ',')*
       ']' ',')?
    '}' ','
  ;
```
]
∈ *Язык описания грамматики языков ANTLR*
⇒ *интерпретация**:
[Сообщение *команды обработки sc-событий* может включать описание sc-элементов, по котором необходимо зарегистрировать или разрегистрировать sc-события. Идентификаторами подкоманд в описании команды могут быть "create" и "delete" соответственно, стоящие в паре с ключевым словом "command". Описание команды регистрации sc-событий "create" представляет собой список описаний типов sc-событий и sc-элементов, по которым необходимо зарегистрировать sc-события. Описания sc-элементов включают хэши этих sc-элементов в парах с ключевым словом "addr". Описание команды разрегистрации sc-событий представляет собой список позиций sc-событий в очереди sc-событий, которые необходимо удалить из этой очереди sc-событий.]

⊃ *Правило, задающее синтаксис типов sc-событий*
⇔ *семантическая эквивалентность**:
[
```
SC_EVENT_TYPE
  : '"add_outgoing_edge"'
  | '"add_ingoing_edge"'
  | '"remove_outgoing_edge"'
  | '"remove_ingoing_edge"'
  | '"content_change"'
  | '"delete_element"'
  ;
```
]
∈ *Язык описания грамматики языков ANTLR*
⇒ *интерпретация**:
[Sc-событиями могут быть *sc-события добавления выходящей дуги из sc-элемента (add_ outgoing_ edge)*, *sc-события добавления входящей дуги в sc-элемент (add_ ingoing_ edge)*, *sc-события удаления выходящей дуги из sc-элемента (remove_ outgoing_ edge)*, *sc-события удаления входящей дуги в sc-элемент (remove_ ingoing_ edge)*, *sc-события изменения содержимого файла ostis-системы (content_ change)* и *sc-события удаления sc-элемента (delete_ element)*.]

⇐ *синтаксическое правило**:
команда обработки sc-событий

∋ *Правило, задающее синтаксис ответа на команду обработки sc-событий*
⇔ *семантическая эквивалентность**:
[

```
sc_json_command_answer_handle_events
  : '"payload"' ':'
    '['
      (SC_ADDR_HASH ',')*
    ']' ','
  ;
```

]

∈ *Язык описания грамматики языков ANTLR*

⇒ *интерпретация**:

[Сообщение *ответа на команду обработки sc-событий* состоит из позиций зарегистрированных sc-событий в очереди. Успешным результатом *команды обработки sc-событий* является статус 1.]

⇐ *синтаксическое правило**:

ответ на команду обработки sc-событий

∋ *Правило, задающее синтаксис ответа инициализации sc-события*

⇔ *семантическая эквивалентность**:

[

```
sc_json_command_answer_init_event
  : '"event"' ':' '1' ','
    '"payload"' ':'
    '['
          (SC_ADDR_HASH ',')*
    ']' ','
  ;
```

]

∈ *Язык описания грамматики языков ANTLR*

⇒ *интерпретация**:

[*Ответ инициализации sc-события* возникает тогда и только тогда, когда в sc-памяти инициализируется соответствующее sc-событие. *Ответ инициализации sc-события* всегда отсылается той клиентской системе, которая зарегистрировала это sc-событие. В сообщение *ответа инициализации sc-события* включаются хэши тех sc-элементов, которые связаны с зарегистрированным sc-событием. Таким образом, если было зарегистрировано sc-событие выходящей sc-дуги, то в списке сообщения *ответа инициализации sc-события* будут находится хэши трёх sc-элементов: хэш sc-элемента, который был подписан на sc-событие, хэш добавленной выходящей из него sc-дуги и хэш sc-элемента, являющегося концом этой sc-дуги.]

⇐ *синтаксическое правило**:

ответ инициализации sc-события

∋ *Правило, задающее синтаксис синтаксических типов sc-элементов*

⇔ *семантическая эквивалентность**:

[

```
SC_ADDR_TYPE
  : SC_NODE_TYPE
  | SC_EDGE_TYPE
  | SC_LINK_TYPE
  ;
```

]

∈ *Язык описания грамматики языков ANTLR*

⇒ *интерпретация**:

[*Правило, задающее синтаксис синтаксических типов sc-элементов* включает *Правило, задающее синтаксис синтаксических типов sc-узлов*, *Правило, задающее синтаксис синтаксических типов sc-дуг*, *Правило, задающее синтаксис синтаксических типов файлов ostis-системы*. Синтаксические типы sc-элементов представляются в виде целого числа и соответствуют программным синтаксическим типам, представляемым в sc-памяти.]

⇒ *примечание**:

[На данный момент форма представления синтаксического типа sc-элемента зависит от того, как располагаются биты в маске sc-элемента. Следующим шагом в развитии *SC-JSON-кода* и

его грамматики могли быть стать устранение такой зависимости и переход к представлению синтаксических типов в виде строковых литералов, интерпретируемых *Серверной системы на основе Websocket, обеспечивающей доступ к sc-памяти платформы интерпретации sc-моделей при помощи команд SC-JSON-кода.*]

∋ *Правило, задающее синтаксис синтаксических типов sc-узлов*

⇔ *семантическая эквивалентность**:

[

```
SC_NODE_TYPE
  : '1'    // sc_type_node
  | '128'  // sc_type_node_tuple
  | '256'  // sc_type_node_struct
  | '512'  // sc_type_node_role
  | '1024' // sc_type_node_norole
  | '2048' // sc_type_node_class
  | '4096' // sc_type_node_abstract
  | '8192' // sc_type_node_material

  | '33'   // sc_type_node_const
  | '168'  // sc_type_node_const_tuple
  | '288'  // sc_type_node_const_struct
  | '544'  // sc_type_node_const_role
  | '1056' // sc_type_node_const_norole
  | '2080' // sc_type_node_const_class
  | '4128' // sc_type_node_const_abstract
  | '8224' // sc_type_node_const_material

  | '65'   // sc_type_node_var
  | '192'  // sc_type_node_var_tuple
  | '320'  // sc_type_node_var_struct
  | '576'  // sc_type_node_var_role
  | '1088' // sc_type_node_var_norole
  | '2112' // sc_type_node_var_class
  | '4160' // sc_type_node_var_abstract
  | '8256' // sc_type_node_var_material
  ;
```

]

∈ *Язык описания грамматики языков ANTLR*

⇒ *интерпретация**:

[*Правило, задающее синтаксис синтаксических типов sc-узлов* описывает возможные синтаксические типы sc-узлов, интерпретируемые на стороне *Серверной системы на основе Websocket, обеспечивающей доступ к sc-памяти платформы интерпретации sc-моделей при помощи команд SC-JSON-кода.*]

∋ *Правило, задающее синтаксис синтаксических типов sc-дуг*

⇔ *семантическая эквивалентность**:

[

```
SC_EDGE_TYPE
  : '4'    // sc_type_edge_common
  | '8'    // sc_type_arc_common
  | '16'   // sc_type_arc_access

  | '2212' // sc_type_edge_common_const_pos_perm
  | '2216' // sc_type_arc_common_const_pos_perm
  | '2224' // sc_type_arc_access_const_pos_perm
  | '2340' // sc_type_edge_common_const_neg_perm
  | '2344' // sc_type_arc_common_const_neg_perm
  | '2352' // sc_type_arc_access_const_neg_perm
  | '2596' // sc_type_edge_common_const_fuz_perm
  | '2600' // sc_type_arc_common_const_fuz_perm
  | '2608' // sc_type_arc_access_const_fuz_perm

  | '1188' // sc_type_edge_common_const_pos_temp
  | '1192' // sc_type_arc_common_const_pos_temp
  | '1200' // sc_type_arc_access_const_pos_temp
  | '1316' // sc_type_edge_common_const_neg_temp
  | '1320' // sc_type_arc_common_const_neg_temp
  | '1328' // sc_type_arc_access_const_neg_temp
  | '1572' // sc_type_edge_common_const_fuz_temp
  | '1576' // sc_type_arc_common_const_fuz_temp
  | '1584' // sc_type_arc_access_const_fuz_temp

  | '2244' // sc_type_edge_common_var_pos_perm
  | '2248' // sc_type_arc_common_var_pos_perm
  | '2256' // sc_type_arc_access_var_pos_perm
  | '2372' // sc_type_edge_common_var_neg_perm
  | '2376' // sc_type_arc_common_var_neg_perm
  | '2384' // sc_type_arc_access_var_neg_perm
  | '2628' // sc_type_edge_common_var_fuz_perm
  | '2632' // sc_type_arc_common_var_fuz_perm
  | '2640' // sc_type_arc_access_var_fuz_perm

  | '1220' // sc_type_edge_common_var_pos_temp
  | '1224' // sc_type_arc_common_var_pos_temp
  | '1232' // sc_type_arc_access_var_pos_temp
  | '1348' // sc_type_edge_common_var_neg_temp
  | '1352' // sc_type_arc_common_var_neg_temp
  | '1360' // sc_type_arc_access_var_neg_temp
  | '1604' // sc_type_edge_common_var_fuz_temp
  | '1608' // sc_type_arc_common_var_fuz_temp
  | '1616' // sc_type_arc_access_var_fuz_temp
  ;
```

]

∈ *Язык описания грамматики языков ANTLR*

⇒ *интерпретация**:

[*Правило, задающее синтаксис синтаксических типов sc-дуг* описывает возможные синтаксические типы sc-дуг, в том числе и sc-рёбер, интерпретируемые на стороне *Серверной системы на основе Websocket, обеспечивающей доступ к sc-памяти платформы интерпретации sc-моделей при помощи команд SC-JSON-кода.*]

∋ *Правило, задающее синтаксис синтаксических типов файлов ostis-системы*

⇔ *семантическая эквивалентность**:

[

```
SC_LINK_TYPE
  : '2'    // sc_type_link
  | '34'   // sc_type_link_const
  | '66'   // sc_type_link_var
  ;
```

]
∈ *Язык описания грамматики языков ANTLR*
⇒ *интерпретация**:
[*Правило, задающее синтаксис синтаксических типов файлов ostis-системы* описывает возможные синтаксические типы файлов ostis-системы, интерпретируемые на стороне *Серверной системы на основе Websocket, обеспечивающей доступ к sc-памяти платформы интерпретации sc-моделей при помощи команд SC-JSON-кода.*]

команда на SC-JSON-коде
:= [sc-json-code command]
⊂ *SC-JSON-код*
⇒ *примечание**:
[Множество *команд на SC-JSON-коде* легко расширяемо засчёт гибкости и функциональности языка JSON.]

ответ на команду на SC-JSON-коде
:= [sc-json-code command answer]
⊂ *SC-JSON-код*
⇒ *примечание**:
[Множество *ответов на команды на SC-JSON-коде* легко расширяемо вместе с расширением *команд на SC-JSON-коде.*]

команда создания sc-элементов
:= [create elements command]
⊂ *команда на SC-JSON-коде*
⇒ *пример**:
Пример команды создания sc-элементов
=
[


```
{
 "id": 3,
 "type": "create_elements",
 "payload": [
  {
   "el": "node",
   "type": 1,
  },
  {
   "el": "link",
   "type": 2,
   "content": 45.4,
  },
  {
   "el": "edge",
   "src": {
    "type": "ref",
    "value": 0,
   },
   "trg": {
    "type": "ref",
    "value": 1,
   },
   "type": 32,
  },
 ],
}
```


]
∈ *команда создания sc-элементов*
⇒ *ответ**:
Пример ответа на команду создания sc-элементов

⇒ *интерпретация**:
[Обработать команду создания sc-элементов: sc-узла с типом 1 (неуточняемого типа), файла ostis-системы с типом 2 (неуточняемого типа) и содержимым в виде числа с плавающей точкой 45.4 и sc-дуги типа 32 (константного типа) между sc-элементом, находящимся на нулевой позиции в массиве создаваемых sc-элементов, и sc-элементом, находящимся на первой позиции в том же самом массиве.]

⇒ *класс команд**:
ответ на команду создания sc-элементов

⇒ *примечание**:
[Стоит отметить, что на уровне интерфейса sc-памяти команда интерпретируется быстро за счёт того, что не используются шаблоны создания изоморфных им конструкций. Также содержимое сообщения *команды создания sc-элементов* может быть пустым.]

ответ на команду создания sc-элементов

:= [create elements command answer]

⊂ *ответ на команду на SC-JSON-коде*

⇒ *пример**:
Пример ответа на команду создания sc-элементов
=
[

```
{
 "id": 3,
 "status": 1,
 "payload": [
  323,
  534,
  342,
 ],
}
```

]

∈ *ответ на команду создания sc-элементов*

⇒ *интерпретация**:
[Созданы sc-элементы с хэшами 323, 534 и 342 соответственно. Команда обработана успешно.]

команда получения соответствующих типов sc-элементов

:= [check elements command]

⊂ *команда на SC-JSON-коде*

⇒ *пример**:
Пример команды получения соответствующих типов sc-элементов
=
[

```
{
 "id": 1,
 "type": "check_elements",
 "payload": [
  885,
  1025,
 ],
}
```

]

∈ *команда получения соответствующих типов sc-элементов*

⇒ *ответ**:
Пример ответа на команду получения соответствующих типов sc-элементов

⇒ *интерпретация**:
[Получить синтаксические типы sc-элементов с хэшами 885 и 1025.]

⇒ *класс команд**:
ответ на команду получения соответствующих типов sc-элементов

⇒ *примечание**:

[Содержимое сообщения *команды получения соответствующих типов sc-элементов* может быть пустым.]

ответ на команду получения соответствующих типов sc-элементов
:= [check elements command answer]
⊂ *ответ на команду на SC-JSON-коде*
⇒ *пример**:
Пример ответа на команду получения соответствующих типов sc-элементов
=
[

```
{
 "id": 1,
 "status": 1,
 "payload": [
  32,
  0,
 ],
}
```

]
∈ *ответ на команду получения соответствующих типов sc-элементов*
⇒ *интерпретация**:
[Типы sc-элементов 32 и 0 соответственно. Команда обработана успешно.]
⇒ *примечание**:
[Если sc-элемент с указанным хэшем не существует, то его тип будет равен 0.]

команда удаления sc-элементов
:= [delete elements command]
⊂ *команда на SC-JSON-коде*
⇒ *пример**:
Пример команды удаления sc-элементов
=
[

```
{
 "id": 1,
 "type": "delete_elements",
 "payload": [
  885,
  1025,
 ],
}
```

]
∈ *команда удаления sc-элементов*
⇒ *ответ**:
Пример ответа на команду удаления sc-элементов
⇒ *интерпретация**:
[Удалить sc-элементы с хэшами 885 и 1025.]
⇒ *класс команд**:
ответ на команду удаления sc-элементов
⇒ *примечание**:
[Содержимое сообщения *команды удаления sc-элементов* может быть пустым.]

ответ на команду удаления sc-элементов
:= [delete elements command answer]
⊂ *ответ на команду на SC-JSON-коде*
⇒ *пример**:
Пример ответа на команду удаления sc-элементов

=
[

```
{
 "id": 1,
 "status": 1,
 "payload": [
 ],
}
```

]
∈ *ответ на команду удаления sc-элементов*
⇒ *интерпретация**:
[Sc-элементы были удалены из sc-памяти успешно.]
⇒ *примечание**:
[Если sc-элемент с указанным хэшем не существует, ответ на команду будет безуспешным.]

команда обработки ключевых sc-элементов
:= [handle keynodes command]
⊂ *команда на SC-JSON-коде*
⇒ *пример**:
Пример команды обработки ключевых sc-элементов
=
[

```
{
 "id": 1,
 "type": "keynodes",
 "payload": [
  {
   "command": "find",
   "idtf": "any_system_identifier",
  },
  {
   "command": "resolve",
   "idtf": "any_system_identifier",
   "elType": 1,
  },
 ],
}
```

]
∈ *команда обработки ключевых sc-элементов*
⇒ *ответ**:
Пример ответа на команду обработки ключевых sc-элементов
⇒ *интерпретация**:
[(1) Найти sc-элемент по системному идентификатору "any_system_identifier"; (2) Разрешить sc-элемент с типом 1 (неуточняемого типа) по системному идентификатору "any_system_identifier".]
⇒ *класс команд**:
ответ на команду обработки ключевых sc-элементов
⇒ *примечание**:
[Данный класс команд позволяет быстро обращаться к sc-элементам по их системным идентификаторам, поскольку ключевые sc-элементы кэшируются на уровне интерфейса.]

ответ на команду обработки ключевых sc-элементов
:= [handle keynodes command answer]
⊂ *ответ на команду на SC-JSON-коде*
⇒ *пример**:

Пример ответа на команду обработки ключевых sc-элементов
=
[


```
{
 "id": 1,
 "status": 1,
 "payload": [
  0,
  128,
 ],
}
```


]
∈ *ответ на команду обработки ключевых sc-элементов*
⇒ *интерпретация**:
[Ключевый sc-элемент с системным идентификатором "any_system_identifier" не был найден, поэтому был создан. хэш нового ключевого sc-элемента - 128. Команда выполнена успешно.]

команда обработки содержимого файлов ostis-системы
:= [handle link contents command]
⊂ *команда на SC-JSON-коде*
⇒ *пример**:
Пример команды обработки содержимого файлов ostis-системы
=
[


```
{
 "id": 1,
 "type": "content",
 "payload": [
  {
   "command": "set",
   "type": "int",
   "data": 67,
   "addr": 3123,
  },
  {
   "command": "get",
   "addr": 232,
  },
  {
   "command": "find",
   "data": "exist",
  },
 ],
}
```


]
∈ *команда обработки содержимого файлов ostis-системы*
⇒ *ответ**:
Пример ответа на команду обработки содержимого файлов ostis-системы
⇒ *интерпретация**:
[(1) Установить содержимое 67 типа "int" в файл ostis-системы с хэшем 3123; (2) Получить

содержимое файла ostis-системы с хэшем 232; (3) Найти файлы ostis-системы с содержимым "exist".]

⇒ *класс команд**:
ответ на команду обработки содержимого файлов ostis-системы

⇒ *примечание**:
[Стоит отметить, что в случае, если файл ostis-системы уже имеет содержимое, то при установке нового содержимого старое содержимое будет удалено из памяти. Содержимое файла ostis-системы может быть установлено как пустое.]

ответ на команду обработки содержимого файлов ostis-системы

:= [handle link contents command answer]
⊂ *ответ на команду на SC-JSON-коде*
⇒ *пример**:
Пример ответа на команду обработки содержимого файлов ostis-системы
=
[

```
{
 "id": 1,
 "status": 1,
 "payload": [
  1,
  {
   "value": 67,
   "type": "int",
  },
  [
   324,
   423,
  ],
 ],
}
```

]
∈ *ответ на команду обработки содержимого файлов ostis-системы*
⇒ *интерпретация**:
[(1) Содержимое 67 типа "int" было установлено успешно в файл ostis-системы с хэшем 3123; (2) Содержимое файла ostis-системы с хэшем 232 - число 67 целочисленного типа; (3) Файлы ostis-системы с содержимым "exist": 324 и 423.]

команда поиска sc-конструкций, изоморфных заданному sc-шаблону

:= [search template command]
⊂ *команда на SC-JSON-коде*
⇒ *пример**:
Пример команды поиска sc-конструкций, изоморфных заданному sc-шаблону
=
[

```
{
 "id": 1,
 "type": "search_template",
 "payload":  {
   “templ” : [
        [
         {
           "type": "addr",
           "value": 23123,
         },
         {
           "type": "type",
           "value": 32,
           "alias": “_edge1",
         },
         {
           "type": "type",
           "value": 2,
           "alias": "_trg" ,
         }
        ],
        [
         {
           "type": "addr",
           "value": 231342 ,
         },
         {
           "type": "type",
           "value": 2000,
           "alias": "_edge2" ,
         },
         {
           "type": "alias",
           "value": "_edge1",
         },
        ],
    ],
    “params”: {
       “_trg”: 564,
    },
 },
}
```

]

∈ *команда поиска sc-конструкций, изоморфных заданному sc-шаблону*

⇒ *ответ**:

Пример ответа на команду поиска sc-конструкций, изоморфных заданному sc-шаблону

⇒ *интерпретация**:

[Найти все такие тройки, в которых первым элементом является sc-элемент с хэшем 23123, третьим sc-элементом является файл ostis-системы неуточняемого константного типа с псевдонимом "_trg", а вторым элементом - sc-дуга типа sc_edge_d_common с псевдонимом "_edge1", исходящая от sc-элемента с хэшем 23123 и входящая в файл ostis-системы с псевдонимом "_trg", и найти все такие тройки, в которых первым элементом является sc-элемент с хэшем 231342, третьим элементов является sc-дуга под псевдонимом "_edge1", а вторым элементом - sc-дуга типа sc_edge_access_const_pos_perm, исходящая от sc-элемента с хэшем 231342 и входящая в sc-дугу "_edge1". На место переменной с псевдонимом "_trg" подставить sc-элемент с хэшем 564.]

⇒ *класс команд**:
ответ на команду поиска sc-конструкций, изоморфных заданному sc-шаблону
⇒ *примечание**:
[Поиск sc-конструкций по сформированному sc-шаблону осуществляется специализированным модулем, являющимся частью sc-памяти.]

ответ на команду поиска sc-конструкций, изоморфных заданному sc-шаблону
:= [search template command answer]
⊂ *ответ на команду на SC-JSON-коде*
⇒ *пример**:
Пример ответа на команду поиска sc-конструкций, изоморфных заданному sc-шаблону
=
[


```
{
 "id": 1,
 "status": 1,
 "payload": {
    "aliases": {
    "_trg": 2,
    "_edge1": 1,
    "_edge2": 4,
   },
   "addrs": [[23123, 4953, 564, 231342, 533, 4953]],
 },
}
```


]
∈ *ответ на команду поиска sc-конструкций, изоморфных заданному sc-шаблону*
⇒ *интерпретация**:
[Найдена одна sc-конструкция, состоящая из двух троек. хэши sc-элементов в тройках: 23123, 4953, 564 и 231342, 533, 4953 соответственно их расположению в тройках. Команда выполнена успешно.]
⇒ *примечание**:
[Важно отметить, что sc-шаблон поиска sc-конструкций не может быть пустым.]

команда создания sc-конструкции, изоморфной заданному sc-шаблону
:= [generate template command]
⊂ *команда на SC-JSON-коде*
⇒ *пример**:
Пример команды создания sc-конструкции, изоморфной заданному sc-шаблону
=
[

```
{
 "id": 1,
 "type": "generate_template",
 "payload": {
   “templ” : [
       [
        {
          "type": "addr",
          "value": 589,
        },
        {
          "type": "type",
          "value": 32,
          "alias": “_edge1",
        },
        {
          "type": "type",
          "value": 1,
          "alias": "_trg" ,
        }
       ],
    ],
    “params”: {
      “_trg”: 332,
    },
  },
}
```

]
∈ *команда создания sc-конструкции, изоморфной заданному sc-шаблону*
⇒ *ответ**:
Пример ответа на команду создания sc-конструкции, изоморфной заданному sc-шаблону
⇒ *интерпретация**:
[Создать такую тройку, в которой первым элементом является sc-элемент с хэшем 589, третьим sc-элементом является sc-узел неуточняемого типа с псевдонимом "_trg", а вторым элементом - sc-дуга типа sc_edge_d_common с псевдонимом "_edge1", исходящая от sc-элемента с хэшем 589 и входящая в sc-узел с псевдонимом "_trg". На место переменной с псевдонимом "_trg" подставить sc-элемент с хэшем 332.]

⇒ *класс команд**:
ответ на команду создания sc-конструкции, изоморфной заданному sc-шаблону
⇒ *примечание**:
[Создание sc-конструкции по сформированному sc-шаблону осуществляется специализированным модулем, являющимся частью sc-памяти.]

ответ на команду создания sc-конструкции, изоморфной заданному sc-шаблону
:= [search template command answer]
⊂ *ответ на команду на SC-JSON-коде*
⇒ *пример**:
Пример ответа на команду создания sc-конструкции, изоморфной заданному sc-шаблону
=
[

```
{
 "id": 1,
 "status": 1,
 "payload": {
   "aliases": {
   "_trg": 2,
   "_edge1": 1,
  },
  "addrs": [128, 589, 332],
 },
}
```


]
∈ *ответ на команду создания sc-конструкции, изоморфной заданному sc-шаблону*
⇒ *интерпретация**:
[Создана одна sc-конструкция, состоящая из одной тройки. хэши sc-элементов в тройке: 128, 589, 332 соответственно их расположению в тройках. Команда выполнена успешно.]
⇒ *примечание**:
[Важно отметить, что sc-шаблон создания sc-конструкции не может быть пустым.]

команда обработки sc-событий
:= [handle events command]
⊂ *команда на SC-JSON-коде*
⇒ *пример**:
Пример команды обработки sc-событий
=
[


```
{
 "id": 1,
 "type": "events",
 "payload": {
  "create": [
   {
    "type": "add_outgoing_edge",
    "addr": 324,
   },
  ],
  "delete": [
   2, 4, 5,
  ],
 },
}
```


]
∈ *команда обработки sc-событий*
⇒ *ответ**:
Пример ответа на команду обработки sc-событий
⇒ *интерпретация**:
[(1) Зарегистрировать sc-событие типа "add_outgoing_edge" по sc-элементу с хэшем 324; (2) Разрегистрировать sc-события с позициями sc-элементов 2, 4 и 5 соответственно.]
⇒ *класс команд**:
ответ на команду обработки sc-событий
⇒ *класс команд**:
ответ инициализации sc-события
⇒ *примечание**:
[*Ответ инициализации sc-события* может производиться несколько раз за разные промежутки

времени.]

ответ на команду обработки sc-событий
:= [handle events command answer]
⊂ *ответ на команду на SC-JSON-коде*
⊃ *SC-JSON-код*
⇒ *пример**:
Пример ответа на команду обработки sc-событий
=
[

```
{
 "id": 1,
 "event": 1,
 "payload": [
  7,
 ],
}
```

]
∈ *ответ на команду обработки sc-событий*
⇒ *интерпретация**:
[(1) Sc-событие типа "add_outgoing_edge" по sc-элементу с хэшем 324 было зарегистрировано успешно на 7-ой позиции очереди sc-событий; (2) Sc-события под позициями 2, 4, 5 удалены успешно.]

ответ инициализации sc-события
:= [init event command answer]
⊂ *ответ на команду на SC-JSON-коде*
⇒ *пример**:
Пример ответа инициализации sc-события
=
[

```
{
 "id": 2,
 "event": 1,
 "payload": [
  324,
  328,
  35,
 ]
}
```

]
∈ *ответ инициализации sc-события*
⇒ *интерпретация**:
[Sc-событие было инициализировано успешно: добавлена выходящая sc-дуга с хэшем 328 из зарегистрированного sc-элемента с хэшем 324 в sc-элемент с хэшем 35. Статус sc-события - 1.]
}
/* Завершили представление *Синтаксиса SC-JSON-кода* */

Серверная система на основе Websocket, обеспечивающая доступ к sc-памяти платформы интерпретации sc-моделей при помощи команд SC-JSON-кода
:=

[Система, работающая по принципам Websocket и предоставляющая параллельно-асинхронный многоклиентский доступ к sc-памяти платформы интерпретации sc-моделей при помощи SC-JSON-кода]
:= [SC-JSON-сервер]
⇒ *часто используемый неосновной внешний идентификатор sc-элемента**:
[SC-сервер]
∈ *многократно используемый компонент ostis-систем*
∈ *атомарный многократно используемый компонент ostis-систем*
∈ *зависимый многократно используемый компонент ostis-систем*
⇐ *компонент системы**:
Программный вариант реализации платформы интерпретации sc-моделей компьютерных систем
⇒ *автор**:
• *Зотов Н. В.*
⇒ *используемый язык программирования**:
• *C*
• *C++*
⇒ *используемый язык**:
• *SC-JSON-код*
⇒ *зависимости компонента**:
• *Библиотека программных компонентов для обработки и, задающее синтаксис json-текстов JSON for Modern C++ версии 3.10.5*
:= [nlohmann-json 3.10.5]
⇐ *версия компонента**:
Библиотека программных компонентов для обработки и, задающее синтаксис json-текстов JSON for Modern C++
:= [nlohmann-json]
∈ *многократно используемый компонент ostis-систем*
∈ *неатомарный многократно используемый компонент ostis-систем*
∈ *зависимый многократно используемый компонент ostis-систем*
⇒ *адрес хранилища**:
[https://github.com/nlohmann/json]
∈ *адрес хранилища на GitHub*
⇒ *скрипт установки**:
[sudo add-apt-repository universe
sudo apt-get update
sudo apt-get install -y nlohmann-json3-dev]
∈ *скрипт на языке Bash*
∈ *скрипт на языке Bash, поддерживаемый семейством операционных систем UNIX*
⇒ *скрипт установки**:
[brew install nlohmann-json]
∈ *скрипт на языке Bash*
∈ *скрипт на языке Bash, поддерживаемый семейством операционных систем MacOS*
• *Библиотека кросс-платформенных программных компонентов для реализации серверных приложений на основе Websocket WebSocket++ версии 0.8.2*
:= [websocketcpp 0.8.2]
⇐ *версия компонента**:
Библиотека кросс-платформенных программных компонентов для реализации серверных приложений на основе Websocket WebSocket++
:= [websocketcpp]
∈ *многократно используемый компонент ostis-систем*
∈ *неатомарный многократно используемый компонент ostis-систем*
∈ *зависимый многократно используемый компонент ostis-систем*
⇒ *адрес хранилища**:
[https://github.com/zaphoyd/websocketpp]
∈ *адрес хранилища на GitHub*
⇒ *скрипт установки**:
[sudo apt-get update
sudo apt-get install -y libwebsocketpp-dev]

∈ *скрипт на языке Bash*
∈ *скрипт на языке Bash, поддерживаемый семейством операционных систем UNIX*
⇒ *скрипт установки**:
[brew install libwebsocketpp]
∈ *скрипт на языке Bash*
∈ *скрипт на языке Bash, поддерживаемый семейством операционных систем MacOS*

- *Программный компонент настройки программных компонентов ostis-систем версии 0.1.0*

:= [sc-config-utils 0.1.0]
⇐ *версия компонента**:
Программный компонент настройки программных компонентов ostis-систем
:= [sc-config-utils]
∈ *многократно используемый компонент ostis-систем*
∈ *неатомарный многократно используемый компонент ostis-систем*
∈ *зависимый многократно используемый компонент ostis-систем*
⇒ *автор**:
- *Зотов Н. В.*
- *Насевич П. Е.*
- *Хорошавин В. Д.*

⇒ *адрес хранилища**:
[https://github.com/ostis-ai/sc-machine/tree/main/sc-tools/sc-config-utils]
∈ *адрес хранилища на GitHub*

- *Программная модель sc-памяти версии 0.6.1*

:= [sc-machine 0.6.1]
⇐ *версия компонента**:
Программная модель sc-памяти

⇒ *адрес хранилища**:
[https://github.com/ostis-ai/sc-machine/tree/main/sc-tools/sc-server]
∈ *адрес хранилища на GitHub*
⇒ *пояснение**:
[*Серверная система на основе Websocket, обеспечивающая доступ к sc-памяти платформы интерпретации sc-моделей при помощи команд SC-JSON-кода*, представляет собой интерпретатор команд и ответов на них *SC-JSON-кода* на программное представление sc-конструкций в sc-памяти при помощи Библиотеки программных компонентов для обработки и, задающее синтаксис json-текстов JSON for Modern C++ и Библиотека кросс-платформенных программных компонентов для реализации серверных приложений на основе Websocket WebSocket++, а также обеспечивается комплексным тестовым покрытием посредством программных фреймворков Google Tests и Google Benchmark Tests. Библиотека программных компонентов для обработки и, задающее синтаксис json-текстов JSON for Modern C++ имеет богатый, удобный и быстродействующий функционал, необходимый для реализации подобных компонентов ostis-систем, а Библиотеки кросс-платформенных программных компонентов для реализации серверных приложений на основе Websocket WebSocket++ позволяет элегантно проектировать серверные системы без использовании избыточных зависимостей и решение. Настройка программного компонента осуществляется с помощью *Программного компонента настройки программных компонентов ostis-систем* и скриптов языков CMake и Bash.]
⇒ *пояснение**:
[Стоит отметить, что текущая реализация *Серверной системы на основе Websocket, обеспечивающая доступ к sc-памяти платформы интерпретации sc-моделей при помощи команд SC-JSON-кода* не является первой в своём роде и заменяет предыдущую её реализацию, написанную на языке Python. Причиной такой замены состоит в следующем:

- предыдущая реализация *Серверной системы на основе Websocket, обеспечивающая доступ к sc-памяти платформы интерпретации sc-моделей при помощи команд SC-JSON-кода*, реализованная на языке программирования Python, зависит от библиотеки Boost Python, предоставляемой сообществом по развитию и коллаборации языков C++ и Python. Дело в том, что такое решение требует поддержки механизма интерпретации программного кода на языке Python на язык C++, что является избыточным и необоснованным, поскольку большая часть программного кода ***Программного варианта реализации платформы интерпретации sc-моделей компьютерных систем*** реализована на языках C и C++. Новая реализация описываемой программной системы позволяет избавиться от использования ёмких и ресурсозатратных библиотек (например, boost-python-lib, llvm) и

языка Python;

- при реализации распределённых подсистем важную роль играет скорость обработки знаний, то есть возможность быстро и срочно отвечать на запросы пользователя. Качество доступа к sc-памяти посредством реализованной *Подсистемы взаимодействия с sc-памятью на основе языка JSON* должно быть соизмеримо с качеством доступа к sc-памяти при помощи специализированного программного интерфейса API, реализованного на том же языке программирования, что и сама система. Новая реализация позволяет повысить скорость обработки запросов *Подсистемой взаимодействия с sc-памятью на основе языка JSON*, в том числе и обработка знаний, не менее чем в 1,5 раза по сравнению с предыдущим вариантом реализации этой подсистемы.

]

⇒ *формальная модель**:

Модель Серверной системы на основе Websocket, обеспечивающая доступ к sc-памяти платформы интерпретации sc-моделей при помощи команд SC-JSON-кода

⇒ *характеристика**:

•

[*Серверная система на основе Websocket, обеспечивающая доступ к sc-памяти платформы интерпретации sc-моделей при помощи команд SC-JSON-кода* обеспечивает многофункциональный доступ к sc-памяти ostis-системы и удовлетворяет требованиям своей модели. С точки зрения прагматики, программный компонент имеет такой же специализированный программный интерфейс, как и *Программная модель sc-памяти*, однако взаимодействие с ним осуществляется посредством сети.]

•

[Реализованный программный компонент позволяет многопользовательский асинхронный доступ к sc-памяти. В ходе тестирования sc-сервера выяснилось, что его реализация позволяет обрабатывать запросы 1000 клиентских систем. В связи с необходимостью обеспечения параллельного доступа к sc-памяти на уровне реализации программного компонента были добавлены блоки синхронизации. Среди таких можно заметить очередь команд на обработку системой - вне зависимости от того сколько клиентских систем и в каком количестве они отправляют команды на обработку, все команды добавляются в очередь - такое решение позволит обойти проблемы взаимодействия блоков синхронизации на уровне sc-памяти. При этом серверную систему невозможно отключить до тех пор, пока очередь команд имеет какие-нибудь элементы. Также серверная система продолжает работать, если в списке идентификаторов клиентских систем остались некоторые идентификаторы этих клиентских систем. Эти функции обуславливаются необходимостью поддержки атомарности запросов, обрабатываемых системой.]

•

[В процессе тестирования реализации *Серверной системы на основе Websocket, обеспечивающая доступ к sc-памяти платформы интерпретации sc-моделей при помощи команд SC-JSON-кода*, были получены оценки скорости обработки запросов этим компонентом. При нагрузочном тестировании использовалась тестовая клиентская система, написанная на C++ и не имеющая функционала обработки текстов SC-JSON-кода. В результате тестирования было выяснено, что при отправке серверной системы 1000 различных команд: команд создания sc-элементов, команд обработки содержимого файлов ostis-системы и команд удаления sc-элементов, время потраченное на их обработку не превышало 0,2 секунды. При этом в отдельных случаях на обработку 1000 команд создания sc-элементов уходило не более 0,14 секунды, команд удаления sc-элементов - не более 0,07 секунды, команд обработки содержимого файлов ostis-системы - не более 0,27 секунды, команд поиска sc-конструкций, изоморфных заданному sc-шаблону - не более 0,45 секунды.]

Модель Серверной системы на основе Websocket, обеспечивающая доступ к sc-памяти платформы интерпретации sc-моделей при помощи команд SC-JSON-кода

⇐ *декартово произведение**:

⟨• {}

⇐ *декартово произведение**:

⟨• {}

⇐ *декартово произведение**:

⟨• {}

⇐ *декартово произведение**:

⟨• *функция создания sc-элементов по их описаниям: типам и инцидентным sc-элементам**

• *функция получения соответствующих синтаксических типов sc-элементов**

⟩
• {}
⇐ *декартово произведение**:
⟨• *функция проверки существования sc-элементов в sc-памяти**
• *функция получения ключевого sc-элемента по системному идентификатору**
⟩
⟩
• {}
⇐ *декартово произведение**:
⟨• {}
⇐ *декартово произведение**:
⟨• *функция разрешения ключевого sc-элемента по системному идентификатору**
• *функция установки содержимого в файл ostis-системы**
⟩
• {}
⇐ *декартово произведение**:
⟨• *функция получения содержимого из файлов ostis-системы**
• *функция получения файлов ostis-системы по содержимому**
⟩
⟩
⟩
• {}
⇐ *декартово произведение**:
⟨• {}
⇐ *декартово произведение**:
⟨• *функция поиска sc-конструкций, изоморфных заданному sc-шаблону, состоящего из троек, специфицированных инцидентными sc-элементами и/или синтаксическими типами и/или идентификаторами и/или псевдонимами их sc-элементов**
• *функция создания sc-конструкции, изоморфной заданному sc-шаблону, состоящего из троек, специфицированных инцидентными sc-элементами и/или синтаксическими типами и/или идентификаторами и/или псевдонимами их sc-элементов**
⟩
• {}
⇐ *декартово произведение**:
⟨• *функция подписки sc-события на sc-элемент**
• *функция отмены подписки sc-события на sc-элемент**
⟩
⟩
⟩
⇒ *пояснение**:
[*Модель Серверной системы на основе Websocket, обеспечивающая доступ к sc-памяти платформы интерпретации sc-моделей при помощи команд SC-JSON-кода* описывает необходимы и достаточный программный интерфейс для взаимодействия с sc-памятью. В общем случае описывает функциональные возможности не только *Серверной системы на основе Websocket, обеспечивающая доступ к sc-памяти платформы интерпретации sc-моделей при помощи команд SC-JSON-кода*, но клиентских систем взаимодействующих с ней, поскольку зачастую эти клиентские системы включают специализированный программный интерфейс, схожий с интерфейсом серверной системы, но на другом языке программирования.]
= {• *функция создания sc-элементов по их описаниям: типам и инцидентным sc-элементам**
⇐ *декартово произведение**:
⟨• {}
⇐ *декартово произведение**:
⟨• *Множество всех синтаксических типов sc-элементов*
• *Множество всех sc-элементов, имеющих инцидентные*

sc-элементы в sc-памяти

⟩

- *Множество всех sc-элементов, хранящихся в sc-памяти*

⟩

⇒ *примечание**:

[Создается sc-элемент заданного синтаксического типа из Множества всех синтаксических типов sc-элементов и имеющий заданные связи с заданными элементами из Множества всех sc-элементов, имеющих инцидентные sc-элементы в sc-памяти, дополняя Множество всех sc-элементов, хранящихся в sc-памяти.]

- *функция получения соответствующих синтаксических типов sc-элементов**

⇐ *декартово произведение**:

⟨• *Множество всех sc-элементов, хранящихся в sc-памяти*

• *Множество всех синтаксических типов sc-элементов*

⟩

⇒ *примечание**:

[У каждого sc-элемента из Множества всех sc-элементов, хранящихся в sc-памяти, можно получить соответствующий синтаксический тип из Множества всех синтаксических типов sc-элементов.]

- *функция проверки существования sc-элементов в sc-памяти**

⇐ *декартово произведение**:

⟨• *Множество всех sc-элементов, хранящихся в sc-памяти*

• *Множество, состоящее из знаков истины и лжи*

⟩

⇒ *примечание**:

[Каждый sc-элемент из Множества всех sc-элементов, хранящихся в sc-памяти, должен находиться в sc-памяти.]

- *функция получения ключевого sc-элемента по системному идентификатору**

⇐ *разность множеств**:

⟨• {}

⇐ *объединение**:

{• {}

⇐ *декартово произведение**:

⟨• *Множество всех sc-элементов, хранящихся в sc-памяти*

• *Множество всех ключевых sc-элементов, хранящихся в sc-памяти*

⟩

• {}

⇐ *декартово произведение**:

⟨• *Множество всех системных идентификаторов sc-элементов, хранящихся в файловом хранилище*

• *Множество всех ключевых sc-элементов, хранящихся в sc-памяти*

⟩

}

• *функция разрешения ключевого sc-элемента по системному идентификатору**

⟩

⇒ *примечание**:

[Каждый ключевой sc-элемент из Множества всех ключевых sc-элементов, хранящихся в sc-памяти, является sc-элементом Множества всех sc-элементов, хранящихся в sc-памяти, и имеет, по крайней мере, системный идентификатор из Множества всех системных идентификаторов sc-элементов, хранящихся в файловом хранилищеs.]

- *функция разрешения ключевого sc-элемента по системному идентификатору**

⇐ *разность множеств**:

⟨• {}

⇐ *объединение**:

{• {}

⇐ *декартово произведение**:

⟨• *Множество всех sc-элементов, хранящихся в sc-памяти*

- *Множество всех ключевых sc-элементов, хранящихся в sc-памяти*

⟩

- {}

⇐ *декартово произведение**:

⟨• *Множество всех системных идентификаторов sc-элементов, хранящихся в файловом хранилище*

• *Множество всех ключевых sc-элементов, хранящихся в sc-памяти*

⟩

}

• *функция получения ключевого sc-элемента по системному идентификатору**

⟩

⇒ *примечание**:

[Из каждого sc-элемента Множества всех sc-элементов, хранящихся в sc-памяти, можно получить ключевой sc-элемент Множества всех ключевых sc-элементов, хранящихся в sc-памяти, зная, по крайней мере, его системный идентификатор из Множества всех системных идентификаторов sc-элементов, хранящихся в файловом хранилище.]

• *функция установки содержимого в файл ostis-системы**

⇐ *декартово произведение**:

⟨• {}

⇐ *декартово произведение**:

⟨• *Множество всех файлов ostis-системы, хранящихся в sc-памяти*

• *Множество внешних знаков, являющихся содержимым файлов ostis-системы, хранящиеся в файловом хранилище*

⟩

• *Множество, состоящее из знаков истины и лжи*

⟩

⇒ *примечание**:

[В каждый sc-элемент из Множества всех файлов ostis-системы, хранящихся в sc-памяти может быть установлено содержимое из Множества всего содержимого файлов ostis-системы, хранящегося в файловом хранилище.]

;

• *функция получения содержимого из файлов ostis-системы**

⇐ *декартово произведение**:

⟨• *Множество всех файлов ostis-системы, хранящихся в sc-памяти*

• *Множество внешних знаков, являющихся содержимым файлов ostis-системы, хранящиеся в файловом хранилище*

⟩

⇒ *примечание**:

[Из каждого файла ostis-системы Множества всех файлов ostis-системы, хранящихся в sc-памяти, можно получить содержимое, принадлежащее Множеству внешних знаков, являющихся содержимым файлов ostis-системы, хранящиеся в файловом хранилище.]

• *функция получения файлов ostis-системы по содержимому**

⇐ *декартово произведение**:

⟨• *Множество внешних знаков, являющихся содержимым файлов ostis-системы, хранящиеся в файловом хранилище*

• *Множество всех файлов ostis-системы, хранящихся в sc-памяти*

⟩

⇒ *примечание**:

[По содержимому из Множества внешних знаков, являющихся содержимым файлов ostis-системы, хранящиеся в файловом хранилище, можно получить подмножество файлов ostis-системы из Множества всех файлов ostis-системы, хранящихся в sc-памяти, в которые установлено это содержимое.]

• *функция поиска sc-конструкций, изоморфных заданному sc-шаблону, состоящего из троек, специфицированных инцидентными sc-элементами и/или синтаксическими типами и/или идентификаторами и/или псевдонимами их sc-элементов**

⇐ *разность множеств**:

⟨• {}

⇐ *декартово произведение**:

⟨• {}
⇐ *декартово произведение**:
⟨• {}
⇐ *декартово произведение**:
⟨• *Множество всех sc-элементов, хранящихся в sc-памяти*
• *Множество всех синтаксических типов sc-элементов*
⟩
• {}
⇐ *декартово произведение**:
⟨• *Множество всех sc-элементов, имеющих инцидентные sc-элементы в sc-памяти*
• *Множество всех идентификаторов sc-элементов, хранящихся в файловом хранилище*
⟩
⟩
• *Множество всех sc-структур*
⟩
• *функция создания sc-конструкции, изоморфной заданному sc-шаблону, состоящего из троек, специфицированных инцидентными sc-элементами и/или синтаксическими типами и/или идентификаторами и/или псевдонимами их sc-элементов**
⟩
⇒ *примечание**:
[По синтаксическим типам sc-элементов из Множества всех синтаксических типов sc-элементов и/или самим sc-элементам из Множества всех sc-элементов, хранящихся в sc-памяти, связанными с sc-элементами, у которых известны синтаксические типы из Множества всех синтаксических типов sc-элементов, в том числе инцидентным sc-элементами из Множества всех sc-элементов, имеющих инцидентные sc-элементы в sc-памяти, и/или идентификаторам (в том числе псевдонимам) sc-элементов Множества всех идентификаторов sc-элементов, хранящихся в файловом хранилище можно найти sc-структуры, принадлежащие Множеству всех sc-структур и состоящие из всех этих sc-элементов.]

• *функция создания sc-конструкции, изоморфной заданному sc-шаблону, состоящего из троек, специфицированных инцидентными sc-элементами и/или синтаксическими типами и/или идентификаторами и/или псевдонимами их sc-элементов**
⇐ *разность множеств**:
⟨• {}
⇐ *декартово произведение**:
⟨• {}
⇐ *декартово произведение**:
⟨• {}
⇐ *декартово произведение**:
⟨• *Множество всех sc-элементов, хранящихся в sc-памяти*
• *Множество всех синтаксических типов sc-элементов*
⟩
• {}
⇐ *декартово произведение**:
⟨• *Множество всех sc-элементов, имеющих инцидентные sc-элементы в sc-памяти*
• *Множество всех идентификаторов sc-элементов, хранящихся в файловом хранилище*
⟩
⟩

- *Множество всех sc-структур*

⟩

- *функция поиска sc-конструкций, изоморфных заданному sc-шаблону, состоящего из троек, специфицированных инцидентными sc-элементами и/или синтаксическими типами и/или идентификаторами и/или псевдонимами их sc-элементов**

⟩

⇒ *примечание**:

[По синтаксическим типам sc-элементов из Множества всех синтаксических типов sc-элементов и/или самим sc-элементам из Множества всех sc-элементов, хранящихся в sc-памяти, связанными с sc-элементами, у которых известны синтаксические типы из Множества всех синтаксических типов sc-элементов, в том числе инцидентным sc-элементами из Множества всех sc-элементов, имеющих инцидентные sc-элементы в sc-памяти, и/или идентификаторам (в том числе псевдонимам) sc-элементов Множества всех идентификаторов sc-элементов, хранящихся в файловом хранилище всегда можно создать sc-структуру, дополняющую Множество всех sc-структур и состоящую из всех этих sc-элементов.]

- *функция подписки sc-события на sc-элемент*

⇐ *декартово произведение**:

⟨• {}

⇐ *декартово произведение**:

⟨• *Множество всех sc-элементов, храняшихся в sc-памяти*

- *Множество всех возможных типов sc-событий в sc-памяти*

⟩

- *Множество всех sc-событий, зарегистрированных в sc-памяти*

⟩

⇒ *примечание**:

[На каждый sc-элемент Множества всех sc-элементов, храняшихся в sc-памяти можно подписать sc-событие из Множества всех sc-событий, зарегистрированных в sc-памяти типа из Множества всех sc-событий, зарегистрированных в sc-памяти.]

- *функция отмены подписки sc-события на sc-элемент*

⇐ *декартово произведение**:

⟨• {}

⇐ *декартово произведение**:

⟨• *Множество всех sc-событий, зарегистрированных в sc-памяти*

- *Множество всех sc-элементов, храняшихся в sc-памяти*

⟩

- *Множество, состоящее из знаков истины и лжи*

⟩

⇒ *примечание**:

[На каждый sc-элемент Множества всех sc-элементов, храняшихся в sc-памяти можно отписать от sc-события из Множества всех sc-событий, зарегистрированных в sc-памяти.]

}

}

/* Завершили описание *Подсистемы взаимодействия с sc-памятью на основе языка JSON* */

} /* Завершили Сегмент "*Описание реализации подсистемы взаимодействия с внешней средой с использованием сетевых языков*" */

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**:

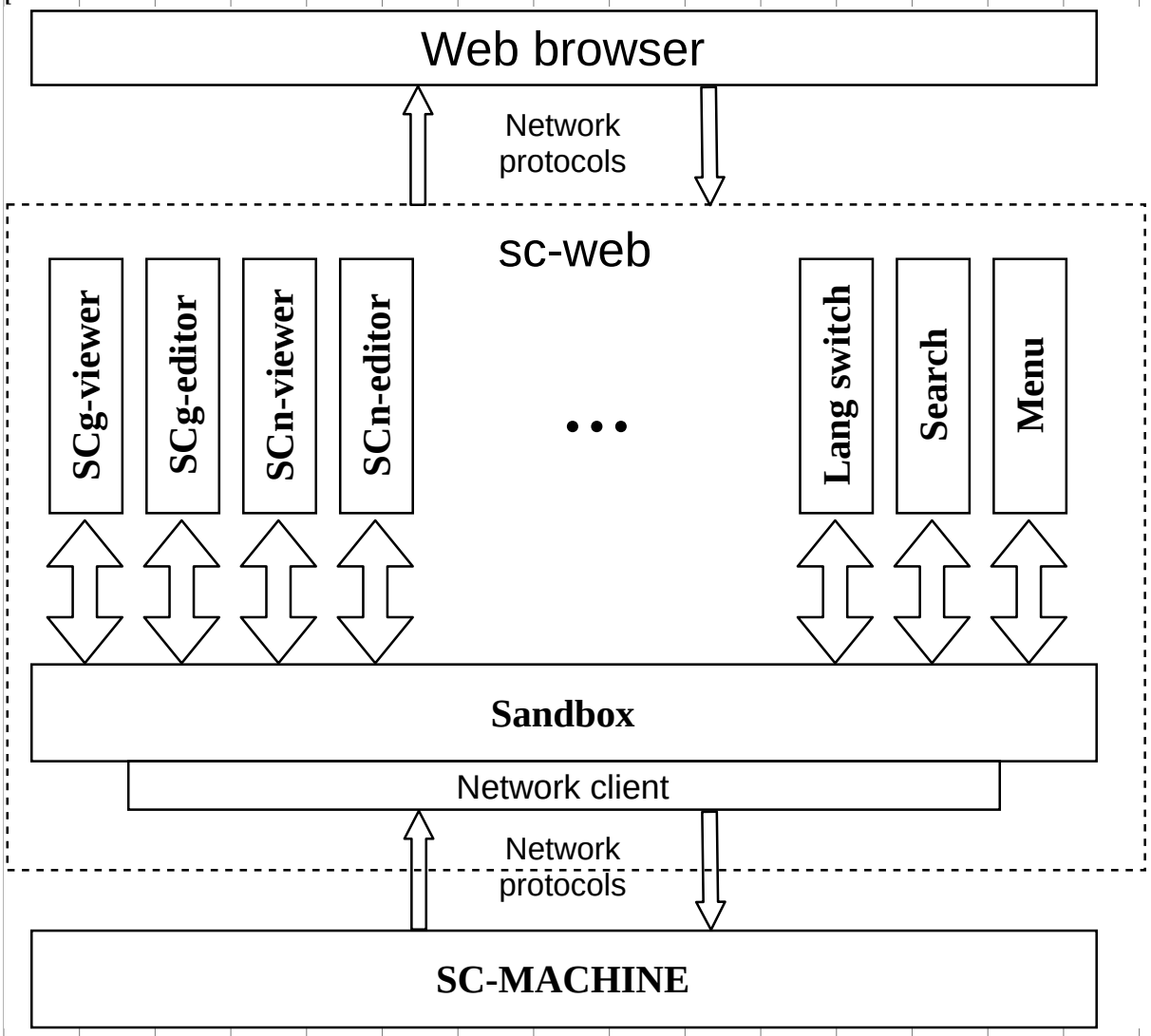
[Наряду с реализацией *Программной модели sc-памяти* важной частью *Программного варианта реализации платформы интерпретации sc-моделей компьютерных систем* является *Реализация интерпретатора sc-моделей пользовательских интерфейсов*, которая предоставляет базовые средства просмотра и редактирования 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 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.]

⇒ *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.]

⇒ *компоненты 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|-т 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-эл

⇒ *библиографическая ссылка**:

[«ArangoDB, the multi-model database for graph and beyond». Англ. В: сент. 2021. URL: https://www.arangodb.com]

Grakn-эл

⇒ *библиографическая ссылка**:

[«Vaticle | Home». Англ. В: сент. 2021. URL: https://vaticle.com/]

Ивашенко.В.П..ПредставССАОСОВСМП-2015ст

⇒ *библиографическая ссылка**:

[В. П. Ивашенко и др. «Представление семантических сетей и алгоритмы их организации и семантической обработки на вычислительных системах с массовым параллелизмом». В: *Открытые семантические технологии проектирования интеллектуальных систем (OSTIS-2015) : материалы V междунар. науч.-техн. конф., Минск, 19–21 февр. 2015 г.* Белорус. гос. ун-т информатики и радиоэлектроники ; редкол.: В. В. Голенков (отв. ред.) [и др.] Минск, 2015, с. 133—140]

Neo4j-эл

⇒ *библиографическая ссылка**:

[«Graph Database Platform | Graph Database Management System | Neo4j». Англ. В: сент. 2021. URL: https://neo4j.com/]

OrientDB-эл

⇒ *библиографическая ссылка**:

[«Home | OrientDB Community Edition». Англ. В: сент. 2021. URL: https://orientdb.org/]

Sesame

⇒ *библиографическая ссылка**:

[**Sesame**]

Shunkevich.D.V.AgentOMMTCPSDIS-2018ст

⇒ *библиографическая ссылка**:

[D. Shunkevich. «Agent-oriented models, method and tools of compatible problem solvers development for intelligent systems». Англ. В: *Open semantic technologies for intelligent systems*. Под ред. V. Golenkov. 2. BSUIR, Minsk, 2018, с. 119—132]

Virtuoso-эл

⇒ *библиографическая ссылка**:

[«OpenLink Software: Virtuoso Homepage». Англ. В: сент. 2021. URL: https://virtuoso.openlinksw.com/]

}