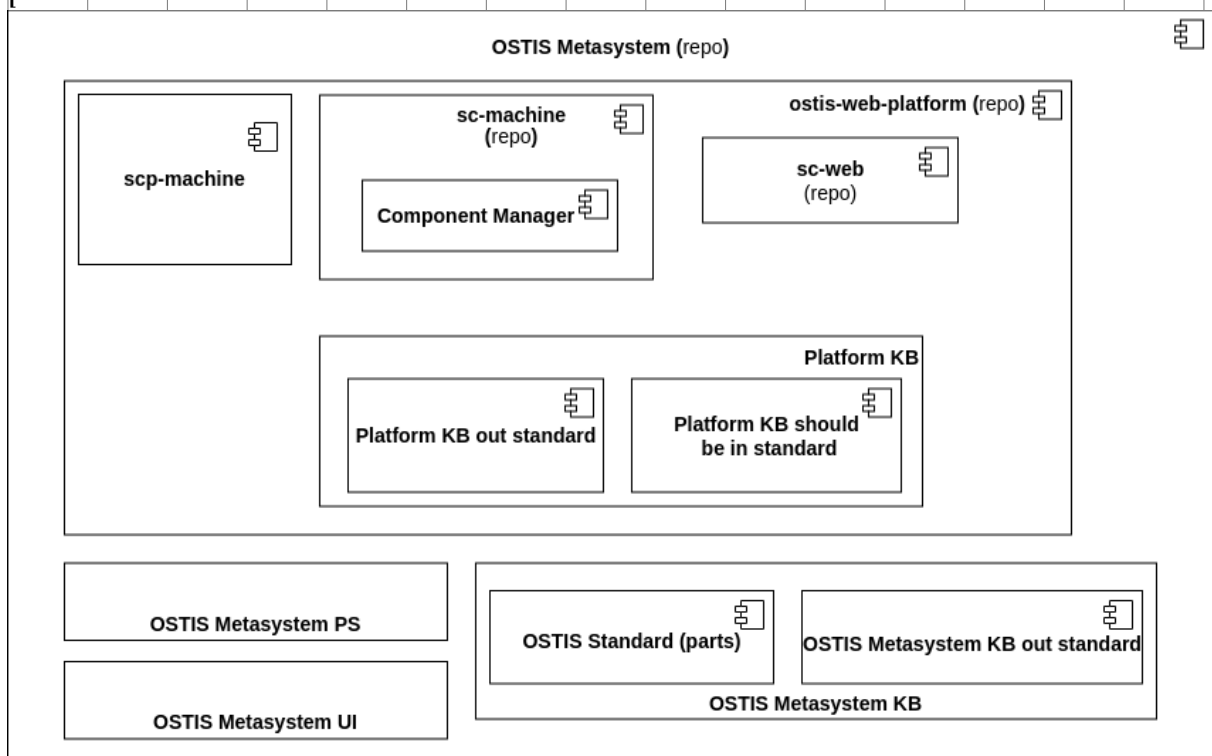


# Документация Метасистемы OSTIS

## Метасистема OSTIS

⇒ иллюстрация\*:



## Агент перевода основных и системных идентификаторов узлов из sc-памяти в текстовый файл

:= [sc-агент трансляции идентификаторов узлов из sc-памяти в текстовый файл]

⇒ задачи\*:

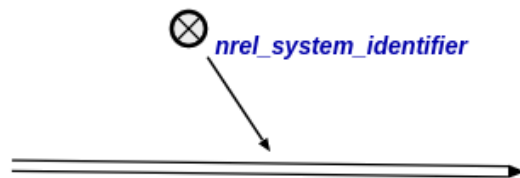
- поиск системных и основных идентификаторов узлов в sc-памяти
- проверка узлов на наличие только одного системного идентификатора и одного основного идентификатора на русском языке
- трансляция в текстовый файл является

⇒ аргументы агента\*:

- пустое множество

⇒ алгоритм\*:

- Поиск всех узлов с помощью итератора, который ищет все конструкции вида



- Проверка каждого узла на выполнение трех условий:
  - Наличие только одного системного идентификатора.
  - Наличие только одного основного русского идентификатора.
  - Принадлежность одному из sc-типов узлов.
- Если не выполняется одно из условий, то запись данных об узле в файл не выполняется.
- Если у узла более одного системного идентификатора, то вызывается исключение.
- Если все три условия выполняются, то данные об узле записываются в файл.

- Если произошла ошибка при работе с файлом, вызывается исключение.

]

⇒ ответ агента\*:

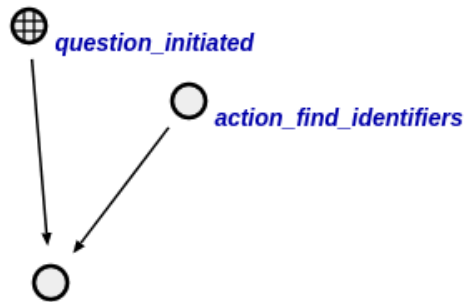
[В результате агент создает текстовый файл, в котором в виде словаря формируются структуры. Роль ключа играет основной русский идентификатор, роль значения – пара, в которой на первом месте стоит системный идентификатор, а на втором – sc-тип узла.]

⇒ пример\*:

[{"main\_ru\_identifer", {"system\_identifier", "sc\_type"} }]

⇒ пример входной конструкции\*:

[



]

⇒ пример выходной конструкции\*:

[

```

2628 {"10'", {"rrel_10", "sc_node_role_relation"} },
2629 {"9'", {"rrel_9", "sc_node_role_relation"} },
2630 {"8'", {"rrel_8", "sc_node_role_relation"} },
2631 {"7'", {"rrel_7", "sc_node_role_relation"} },
2632 {"6'", {"rrel_6", "sc_node_role_relation"} },
2633 {"5'", {"rrel_5", "sc_node_role_relation"} },
2634 {"4'", {"rrel_4", "sc_node_role_relation"} },
2635 {"3'", {"rrel_3", "sc_node_role_relation"} },
2636 {"2'", {"rrel_2", "sc_node_role_relation"} },
2637 {"1'", {"rrel_1", "sc_node_role_relation"} },
2638 {"0'", {"rrel_0", "sc_node_role_relation"} }

```

]

### Агент поиска ответа на сообщение

:= [sc-агент поиска ответа на сообщение в sc-памяти]

⇒ задачи\*:

- поиск ответного действия для класса сообщения
- вызов найденного агента
- запись результата вызова найденного агента как ответа на сообщение

⇒ аргументы агента\*:

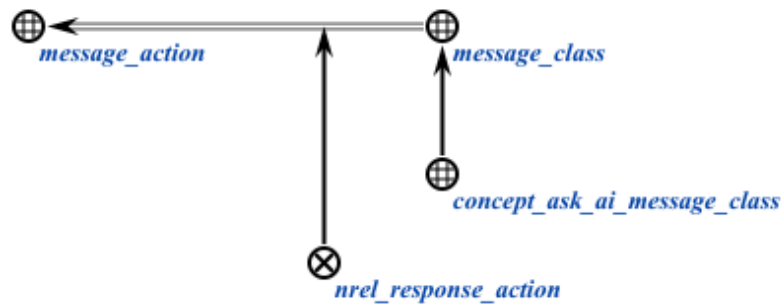
- сообщение

⇒ алгоритм\*:

- [Поиск класса сообщения]
- [Проверка принадлежности класса сообщения на принадлежность классу *concept\_ask\_ai\_message\_class*]
- [Если условие не выполняется, происходит завершение работы агента]
- [Поиск соответствующего классу сообщения класса действий с помощью следующей конструкции]

⇒ пример\*:

[



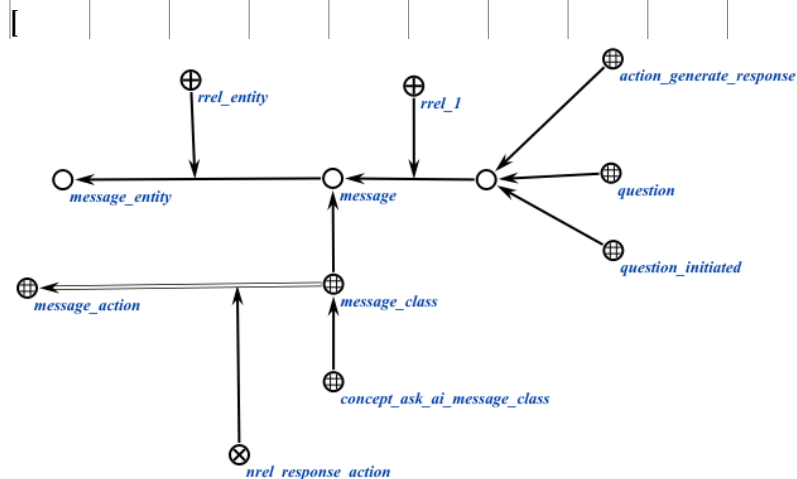
- [Если такой конструкции нет, то происходит завершение работы агента]
- [Вызов агента, соответствующего классу действий с параметрами, полученными из сообщения]
- [Ожидание завершения работы вызванного агента]
- [Если агент завершил работу успешно, то его ответ прикрепляется к сообщению отношением *nrel\_response*]
- [Если агент завершил работу неуспешно, то происходит завершение работы текущего агента]

}

⇒ ответ\*:

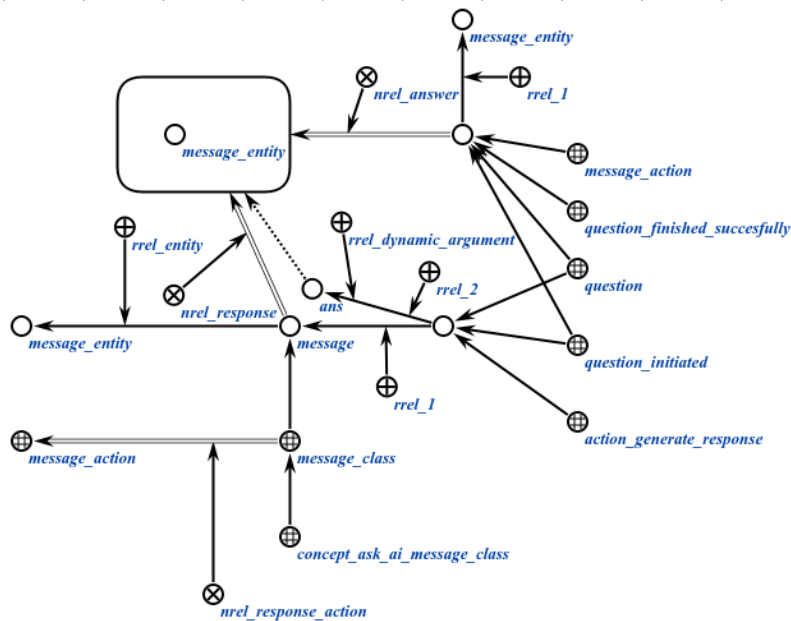
[В результате агент не выдаёт никакого ответа, однако прикрепляет полученный им ответ другого агента к сообщению]

⇒ пример входной конструкции\*:



⇒ пример выходной конструкции\*:

[



Агент ответа на сообщение

⇒ примечание\*:

[Данный агент генерирует ответное сообщение, связанное с исходным сообщением с помощью отношения `inrel_reply`]

$$\Rightarrow \text{класс действия}^*:$$

[action\_reply\_to\_message]

$$\Rightarrow \text{задачи}^*:$$

$\langle$	•	[сформировать ответ на сообщение]
$\rangle$		

$$\Rightarrow \text{аргументы агента}^*:$$

$\langle \bullet$	<i>linkAddr</i>
-------------------	-----------------

⇒ пояснение\*:

[linkAddr - sc-ссылка с текстом сообщения пользователя]

- *processingProgramAddr*

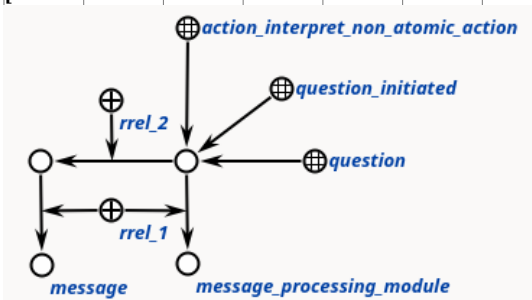
⇒ пояснение\*:

```
[processingProgramAddr - sc-узел обрабатывающей программы]
```

$$\Rightarrow \text{алгоритм}^*:$$

- [Генерация узла сообщения в базе знаний, идентификация полученного текстового файла как текст этого сообщения]

- [Генерация необходимой конструкции для вызова агента интерпретации неатомарного действия. Пример этой конструкции показан ниже]

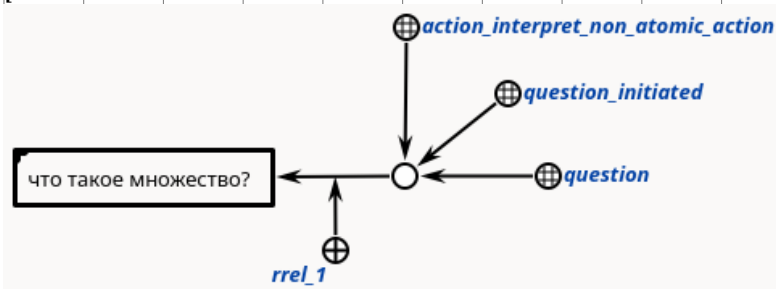


- [Ожидание завершения работы агента интерпретации и выполнение поиска ответного сообщения, сгенерированное во время работы агента интерпретации]

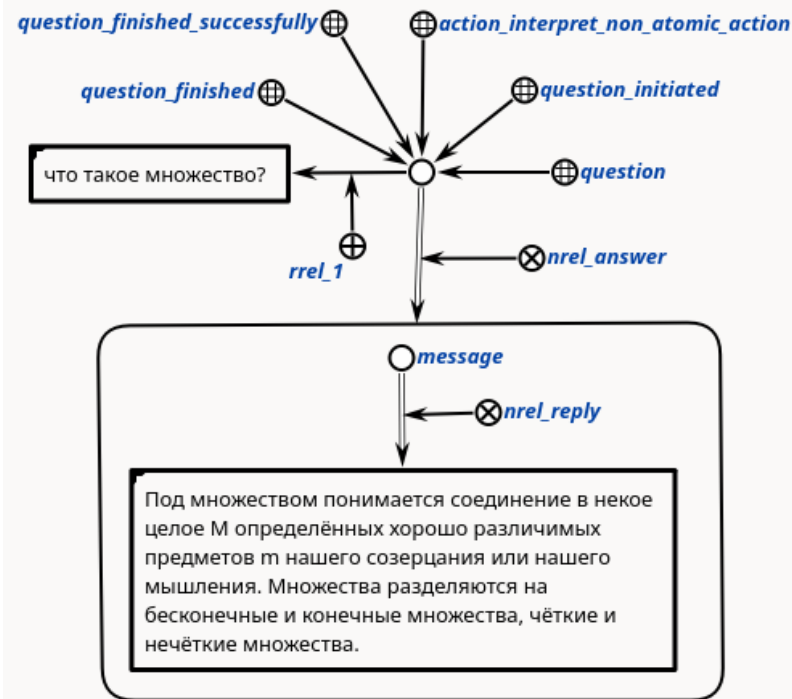
- [Добавление ответного сообщения к ответной структуре]

>

⇒ пример входной конструкции\*:



⇒ пример выходной конструкции\*:



⇒ язык реализации агента\*:

[c++]

⇒ возможные результаты\*:

- SC\_RESULT\_OK  
⇒ пояснение\*:  
[ответное сообщение сгенерировано]
- SC\_RESULT\_ERROR  
⇒ пояснение\*:  
[внутренняя ошибка]

#### Агент классификации сообщения

⇒ примечание\*:

[sc-агент классификации сообщения в sc-памяти]

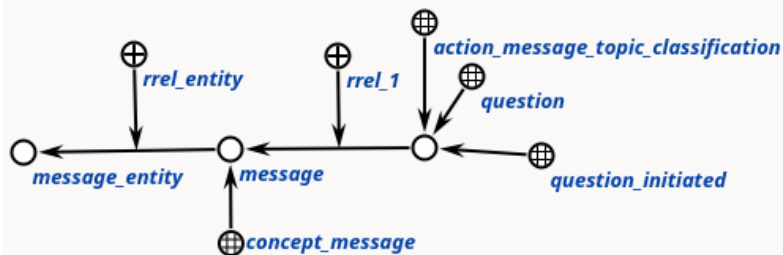
⇒ класс действия\*:

[action\_message\_topic\_classification]

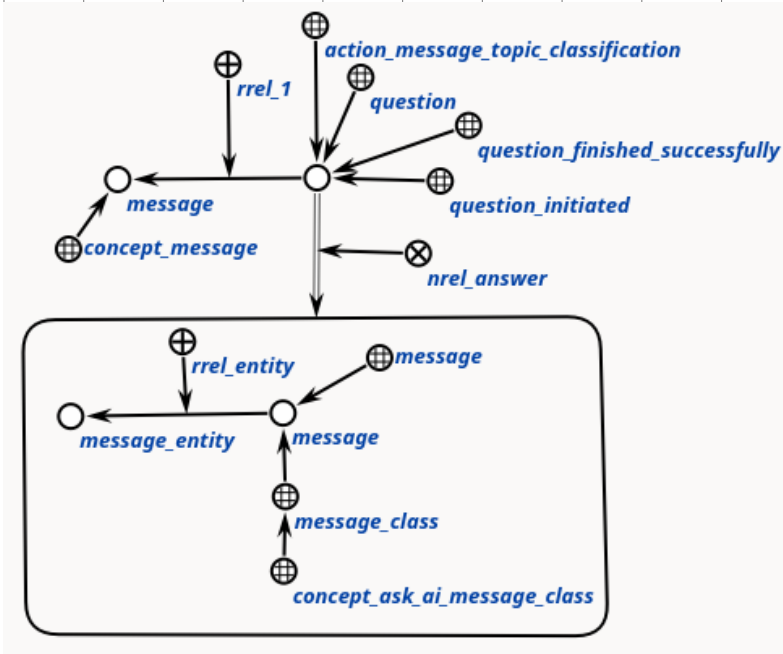
⇒ используемые библиотеки\*:

- Wit.ai  
⇒ пояснение\*:  
[Сервис для классификации сообщений и получения сущностей сообщений.]

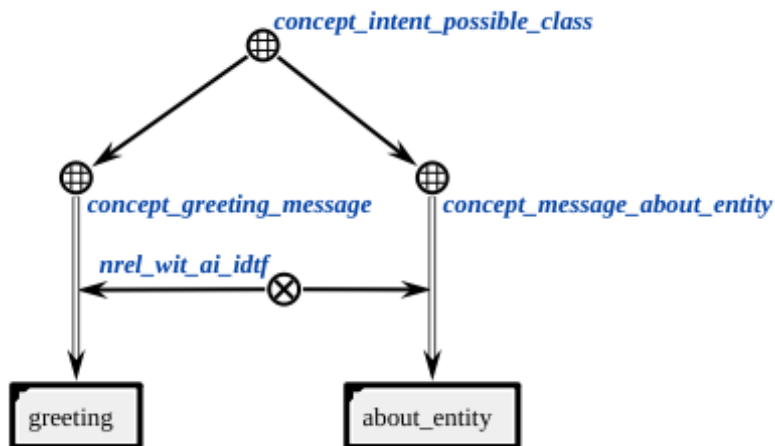
- ⇒ комментарий\*:
- входное сообщение должно содержать текстовый файл с текстом на русском языке
  - выделяемая сущность должна быть формализована в базе знаний
- ⇒ задачи\*:
- определить класс сообщения
- ⇒ аргументы агента\*:
- сообщение
    - ⇒ пояснение\*:  
[MessageAddr - элемент класса concept\_message]
- ⇒ алгоритм\*:
- [Получение текста сообщения]
  - [Получение темы сообщения]
  - [Получение признаков сообщения]
  - [Получение сущности сообщения]
  - [Если что-то не получилось получить, то происходит завершение работы агента]
  - [Если агент завершил работу успешно, то его ответ прикрепляется к сообщению отношением nrel\_entity]
  - [Если агент завершил работу неуспешно, то происходит завершение работы текущего агента]
- ⇒ пример входной конструкции\*:
- [



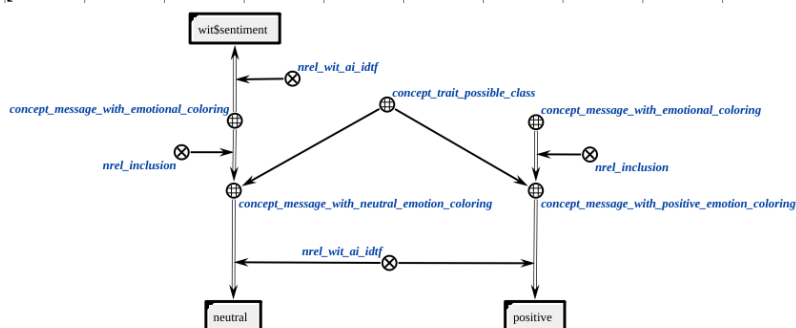
- ⇒ пример выходной конструкции\*:
- [



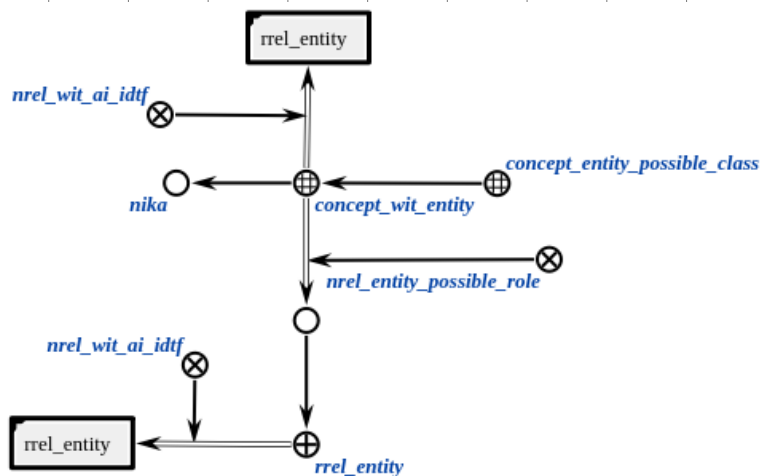
- ⇒ Пример структуры, необходимой для классификации сообщения по теме\*:
- [



⇒ Пример структуры, необходимой для классификации сообщения по признакам\*:



⇒ Пример структуры, необходимой для получения сущностей сообщения\*:



⇒ язык реализации агента\*:

**[c++]**

⇒ возможные результаты\*:

- |                   |           |               |           |
|-------------------|-----------|---------------|-----------|
| $\langle \bullet$ | <i>SC</i> | <i>RESULT</i> | <i>OK</i> |
|-------------------|-----------|---------------|-----------|

$\Rightarrow$  — пояснение\*:

[сообщение успешно классифицировано (или произошла пустая классификация), или действие не принадлежит action message topic classificatin]

- *SC RESULT ERROR*

⇒ пояснение\*:

[внутренняя ошибка]



### Агент интерпретации неатомарных действий

:= [Non atomic action interpretation agent]

⇒ пояснение\*:

[Агент создаёт описание неатомарного действия в базе знаний на основе полученного шаблона (программы). Если передано множество аргументов, то перед генерацией происходит подстановка аргументов действия.]

⇒ класс действия\*:

*action\_interpret\_non\_atomic\_action*

⇒ аргументы агента\*:

- программа
- множество аргументов

⇒ понятие, специфицирующее действие\*:

*nrel\_subaction*

⇒ примечание\*:

[Экземпляр действия, которое инициирует действие, создаваемое. Необходимо, чтобы прервать интерпретацию неатомарного действия при отмене действия более высокого уровня.]

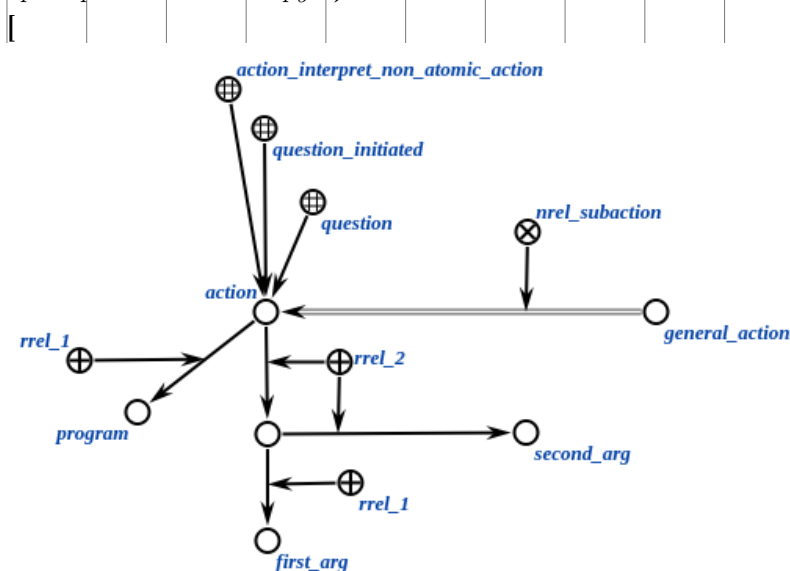
⇒ алгоритм\*:

- [Первый аргумент неатомарного действия заменяется узлом, принадлежащим множеству аргументов как первый, второй аргумент неатомарного действия заменяется узлом, принадлежащим множеству как второй. Обратите внимание, что порядок аргументов задается отношениями *rrel\_1*, *rrel\_2* и т. д., а не отношением последовательности.]
- [В ходе работы агента программа (шаблон) генерирует описание неатомарного действия. Составляющие его атомарные действия также добавляются в класс выполняемых (успешно или неуспешно) выполнившими их агентами.]

⇒ примечание\*:

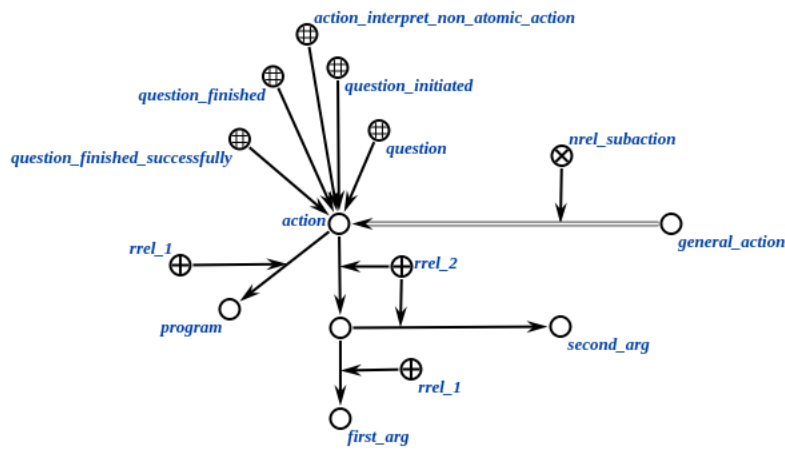
[Перед инициированием каждого атомарного действия происходит проверка, не прервано ли общее действие (*nrel\_subaction*). Если общее действие было прервано, то интерпретация неатомного действия прекращается.]

⇒ пример входной конструкции\*:



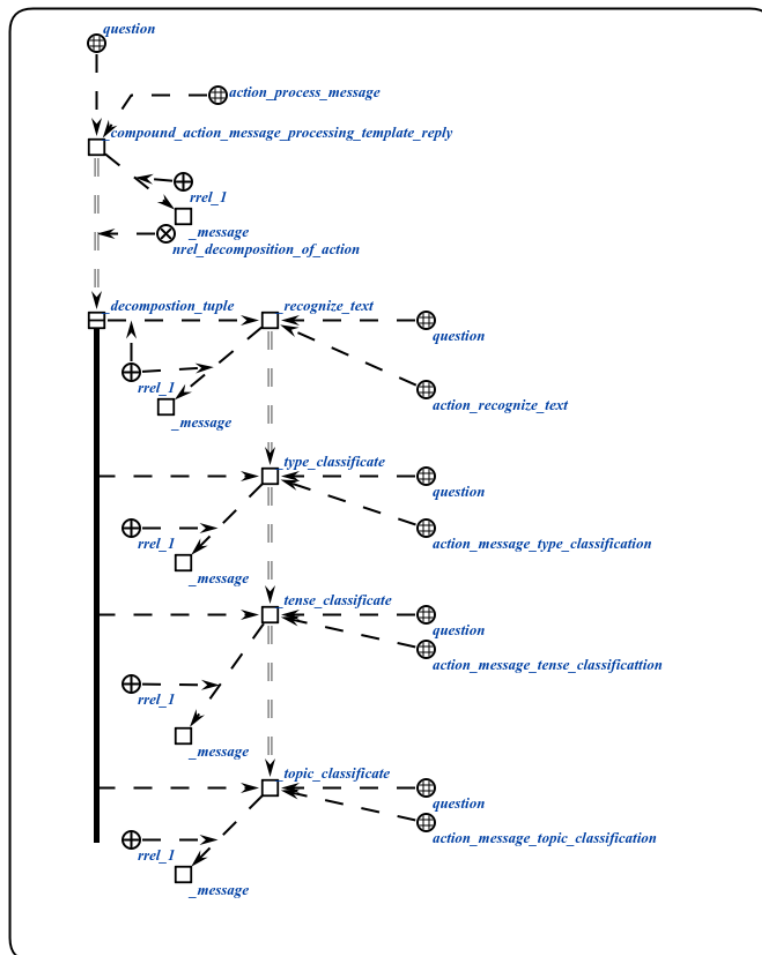
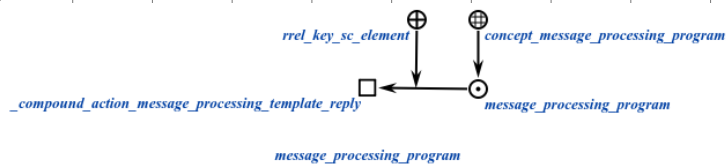
⇒ пример выходной конструкции\*:

[



⇒

пример программы\*:



⇒

возможные коды ответа\*:

- SC\_RESULT\_OK  
⇒ примечание\*:  
[Интерпретация неатомарного действия завершена.]
- SC\_RESULT\_ERROR  
⇒ примечание\*:

} [Произошла ошибка.]

программа

⇒ варианты обхода\*:

{ • [переход к следующему действию при успешном завершении действия]

⇒ примечание\*:

[Устанавливается отношением `nrel_then`. Переход по этому отношению осуществляется при успешном завершении действия, из которого осуществляется переход (его добавлении в класс `questions_finished_successfully`).]

• [переход к следующему действию при безуспешном завершении действия]

⇒ примечание\*:

[Устанавливается отношением `nrel_else`. Переход по этому отношению осуществляется при безуспешном завершении действия, из которого осуществляется переход (его добавлении в класс `questions_finished_unsuccessfully`).]

• [безусловный переход к следующему действию]

⇒ примечание\*:

[Устанавливается отношением `nrel_goto`.]

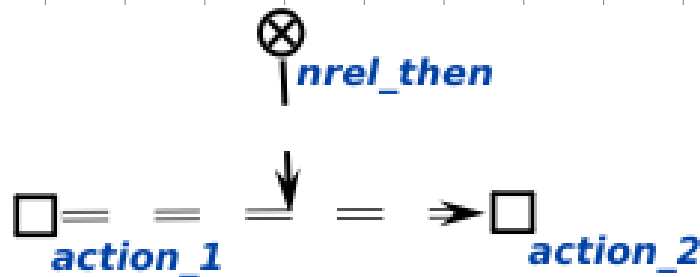
}

⇒ примечание\*:

[Переход к следующему действию зависит от результата предыдущего. После завершения действия (добавленного в класс `questions_finished`) проверяется его успешность для определения необходимого перехода.]

⇒ пример условия обхода\*:

[

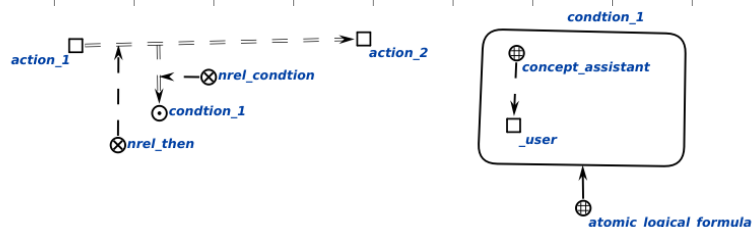


⇒ примечание\*:

[Помимо переходов, зависящих от результата предыдущего действия, вводятся условные переходы через отношение `nrel_condition`. Первым элементом пар этого отношения являются пары (дуги) переходов по успешности действия, вторым — логическая формула. При этом на пару переходов накладывается дополнительное условие (помимо успешности/неуспеха завершения действия) — истинность логической формулы. В этом случае истинность формулы рассчитывается для тех же подстановок, которые использовались при генерации процесса по программе.]

⇒ пример условия\*:

[

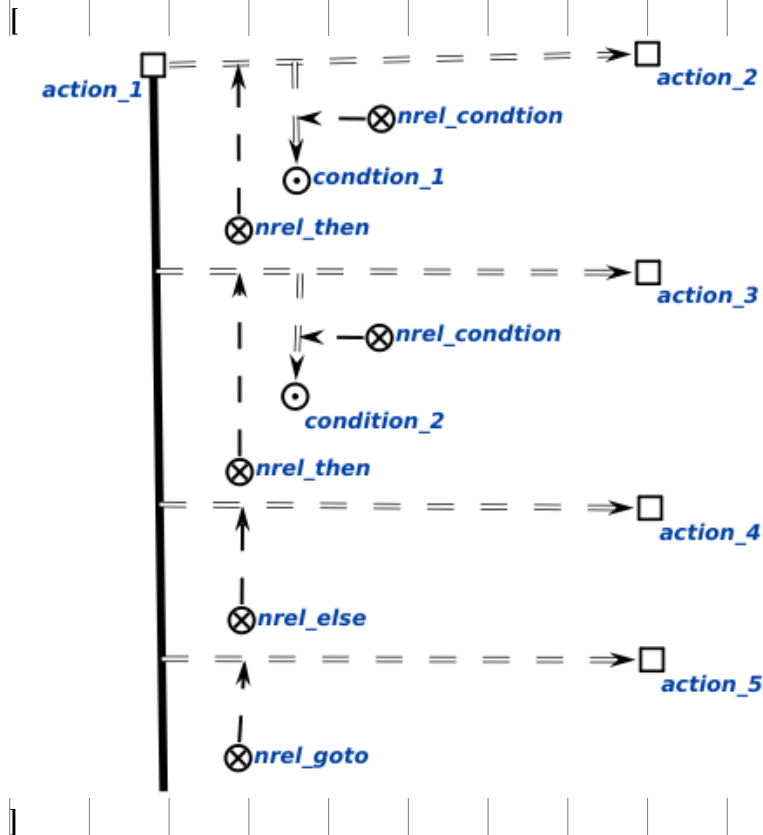


⇒ примечание\*:

[В программе также можно задать приоритет выполнения действий. В первую очередь проверяются переходы по парам отношения `nrel_then`. Если переход не выполнялся (из-за неудачного завершения действия, наличия (и ложности) дополнительного условия или отсутствия таких переходов), то аналогичная проверка производится для переходов отношения `nrel_else`, то — `nrel_goto`. При наличии нескольких переходов, принадлежащих одному и тому же отношению, последовательность

проверки таких переходов является случайной (зависит от состояния системной памяти). Если необходимо задать другой порядок проверки условий перехода, можно использовать отношение `nrel_basic_sequence`. В этом случае пара первого перехода в последовательности (наивысшего приоритета) дополнительно принадлежит отношению `nrel_priority_path`. В этом случае в первую очередь будут проверяться переходы в этой последовательности, а уже потом остальные.]

⇒ пример без приоритетов\*:



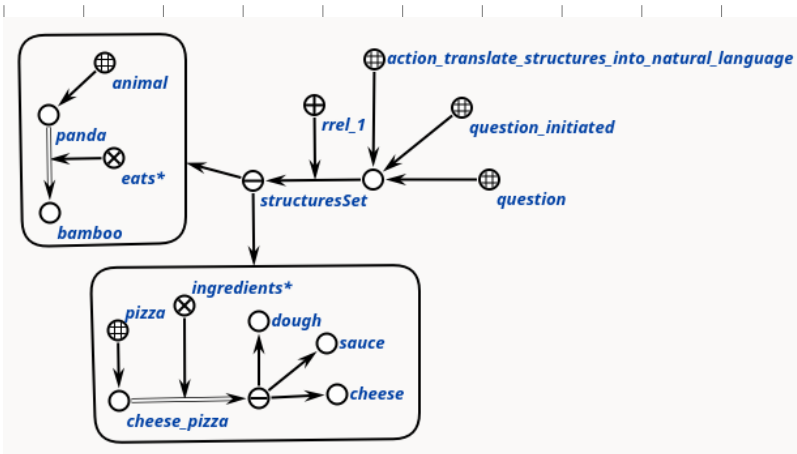
⇒ примечание\*:

[В данном примере возможны два варианта последовательности выполнения действий: **action\_1 -> action\_2** (через `nrel_then`), **action\_1 -> action\_3** (через `nrel_then`), **action\_1 -> action\_4** (через `nrel_else`), **action\_1 -> action\_5** (через `nrel_goto`). Вторым вариантом следующий: **action\_1 -> action\_3** (через `nrel_then`), **action\_1 -> action\_2** (через `nrel_then`), **action\_1 -> action\_4** (через `nrel_else`), **action\_1 -> action\_5** (через `nrel_goto`)]

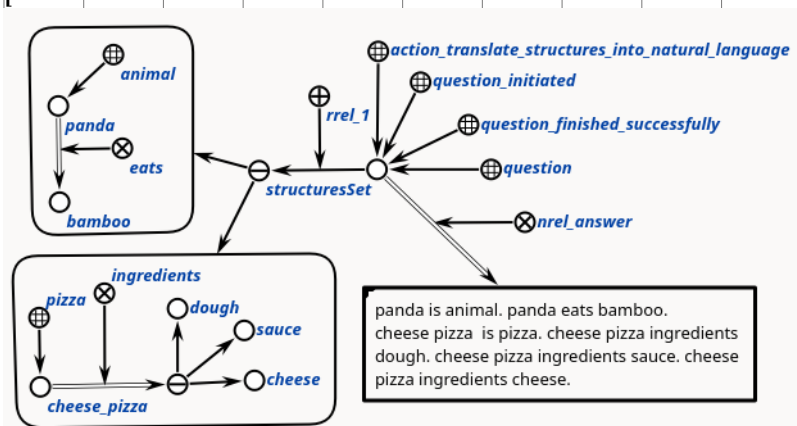
⇒ пример с приоритетами\*:

[



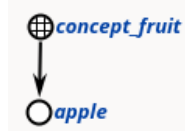


⇒ пример выходной конструкции\*:



⇒ Примеры связей, переводимые трансляторами\*:

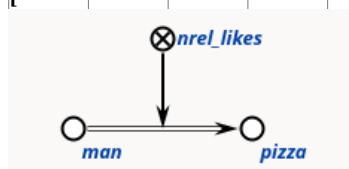
{• [FromConceptTranslator]



⇒ перевод\*:

[apple is fruit. ]

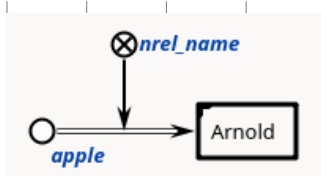
• [NrelFromNodeTranslator]



⇒ перевод\*:

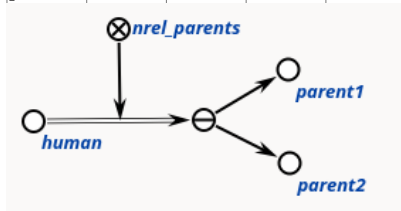
[man likes pizza. ]

} NrelInLinkTranslator



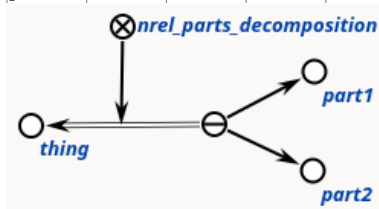
⇒ перевод\*:  
[apple name Arnold. ]

NrelInQuasybinaryNodeTranslator



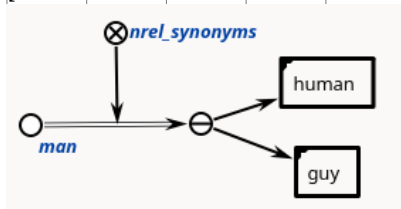
⇒ перевод\*:  
[human parents first parent. human parents second parent. ]

NrelFromQuasybinaryNodeTranslator



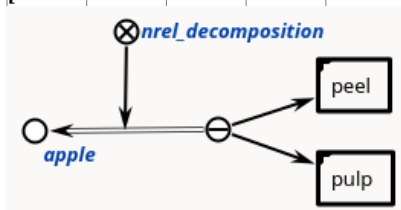
⇒ перевод\*:  
[thing parts decomposition big part. thing parts decomposition little part. ]

NrelInQuasybinaryLinkTranslator



⇒ перевод\*:  
[man synonyms human. man synonyms guy. ]

NrelFromQuasybinaryLinkTranslator



⇒ перевод\*:  
[apple decomposition pulp. apple decomposition peel. ]

⇒ примечание\*:  
[Для перевода используются главные идентификаторы узлов. Если у какого-либо узла в связи нет идентификатора, перевод не будет выполнен. Идентификаторы могут быть за пределами структуры.]

```

⇒ язык реализации агента*:
[с++]
⇒ возможные результаты*:
{ • SC_RESULT_OK
    ⇒ пояснение*:
    [структуры переведены]
  • SC_RESULT_ERROR
    ⇒ пояснение*:
    [внутренняя ошибка]
}

```



```
/* Section *****  
Библиографический раздел Метасистемы OSTIS  
⊃=  
{  
}
```