|  |  |
| --- | --- |
| Зачем нужна эта система? | Система OntoNet представляет собой инструмент для моделирования РСП |
| В чем заключается мотивация? | Мы хотим разрабатывать модели сложных систем, упрощая т.о. процесс реализации |
| Почему используются РСП? | Потому что они позволяют моделировать асинхронные процессы при высоком уровне возможностей, присущим языкам программирования |
| Каковы особенности разрабатываемой системы? | Моделирование и интерпретация производятся в контексте сети (семантический веб) с использованием онтологий |
| В чем цель онтологии? | Вынести знания о статике и динамике РСП в декларативную область |
| Какие проблемы решает онтология? | Расширение и модификация семантики выполнения;  Переиспользование РСП в различных предметных областях путем деления на алгебру и сигнатуру;  Обмен промежуточными результатами исполнения в декларативном формате с возможностью применения агентов, написанных на других ЯП. |
| При чем тут семантический веб? | Для работы с метаданными используется язык запросов SPARQL, являющийся стандартом семантического веба.  SPARQL предоставляет возможности серверного API по типу GraphQL. |
| Каким образом достигается расширение и модификация системы? | **Правила исполнения** (рассмотрены в конце) должны быть включены в определение корневой (core) онтологии и обработаны движком/ризонером. Данная онтология в конечном итоге заменяет предыдущую, движок при этом не переписывается |
| Каким образом достигается переиспользование РСП? | Онтологии составляют иерархию с импортами. Создание онтологии экземпляров возлагается на отдельный инструмент. Использование этого инструмента для создания и изменения РСП – ключ к переиспользованию |
| Каким образом достигается обмен? | Статическая структура и динамика исполнения закладывается в качестве данных в онтологию. Это допускает ее использование клиентами, которые написаны на разных языках программирования. Для запросов применяется SPARQL, передача данных в формате JSON (или XML-эквивалент) |
| Как происходит процесс интерпретации РСП | Корневая онтология и онтология экземпляров располагаются на сервере. Доступ к ним осуществляется через SPARQL запросы. На сервере функционирует JavaScript Reasoner, который вычисляет выражения (term’ы) и формирует transition modes. Клиент получает с сервера состояние сети и transition modes и принимает решение о срабатывании перехода. |
| В какой роли выступает JavaScript? | Выражения на JS используются для определения наборов цветов, констант и функций, задания аннотаций дуг, сторожевых условий и кода переходов |
| Почему именно JavaScript? | JavaScript широко используется в вебе и хорошо знаком веб-разработчикам, что упрощает процесс ознакомления с системой (требуются знания о РСП и инструменте для формирования онтологии экземпляров) |
| Должен ли JavaScript предоставляться в формате «как есть»? | Скорее да, поскольку в данном случае JS является областью знаний, которая понятна компьютеру (ризонеру) |
| Из чего состоит статическая структура РСП в формате онтологии | Структура формируется в форме отдельного .owl файла, импортирующего core онтологию. В общем случае, это экземпляры узлов и связывающих их дуг. |
| Что включает в себя экземпляр позиции? | Позиция содержит маркировку, которая состоит из объектов-токенов, полученных в результате вычисления (evaluation) term-выражения на JS, являющегося мультимножеством |
| Что включает в себя экземпляр перехода? | Переход содержит сторожевое условие перехода в виде term, код, выполняемый при срабатывании в виде term, а также mode с возможными значениями переменных |
| Что включает в себя экземлпяр дуги? | Дуга содержит оконечные узлы и аннотацию в виде term |
| Из чего состоит динамика исполнения РСП в формате онтологии | Динамика включает в себя экземпляры срабатываний переходов, маркировку сети и позиций, mode переходов, сопоставления и вычисления значений. |
| Что представляет из себя срабатывание (firing) перехода? | Срабатывание перехода отражает в онтологии факт выполнения с транспортировкой меток, затрагивающих при этом мультимножество transition mode |
| Что такое маркировка сети? | Маркировка сети отражает маркировку позиций между срабатываниями |
| Как рассчитывается маркировка позиции? | Конкретные значения (Data, возможно в JSON) для токенов в позиции на этапе инициализации рассчитываются из term в ходе работы ризонера, а в ходе срабатываний могут быть удалены или пополнены в ходе вычислений |
| Как рассчитывается mode перехода? | mode формируется в ходе анализа маркировок входных позиций ризонером и состоит из набора вычислений, именуемых сопоставлениями (bindings), связывающих data-значения с переменными |
| Каким образом обрабатывается код, выполняемый при срабатывании перехода? | term в данном случае представляет из себя тело функции, принимающей сопоставления (bindings из transition modes) и возвращающей новые сопоставления (new bindings).  При этом transition modes для этого срабатывания (firing) получают новые наборы сопоставлений (new bindings) вместо старых |
|  |  |
|  |  |
| Что такое term? | Это JS-выражение, содержащее переменные и вызовы функций и использующееся в качестве аннотаций сети. |
| Каким образом вычисляется term? | term вычисляется JS-ризонером, в ходе чего в онтологии формируются наборы вычислений (evaluations) в конкретный |
| Что из себя представляет мультимножество? | Мультимножество – это представление множества в виде базовых наборов, для которых определены свойства многочисленности (multiplicity) |
| В какой роли используются мультимножества? | Они используются для представления наборов токенов в позициях, term’ов дуг и mode у переходов. |
| Как представляется начальная маркировка позиции (мультимножество) в виде term’а | Вызывается реализованная функция-конструктор Multiset:  Multiset(…basisSets),  где basisSets состоят из вызовов аналогичной функции BasisSet:  BasisSet(tokenColorSetConstructorData, multiplicity),  где первый аргумент подается на вход реализации множества цветов позиции, маркировка которой рассматривается |
| Как представляется аннотация дуги (мультимножество) в виде term’a | Аналогично вышесказанному, но  Данные basisSet’ов являются не аргументами конструктора, а являются JSON-описанием токена, который может пройти по этой дуге через переход  В конечном счете, объект, получившийся в ходе вычисления ризонером значения токена, обходится по шаблону, представленному в аннотации. |
| Как выглядит пример JSON-шаблона для basisSet’а мультимножества аннотации дуги? | Для массива объектов со свойством prop:  {  “0”: {  “prop/const”: “constant”,  },  “1/var”: “obj”,  “2/rest”: “list”,  “3/expr”: “[constant\_value + 1, 2, 9 % 3]”,  “/var”: “arr”,  “/const”: “constant\_value”, // compare only  “/expr”: “func\_call(arr)” // keeps parsing  }  Данный формат немногословен. Если считать, что процесс составления шаблонов будет возложен на инструмент, а пользователю будет предоставлен более удобный интерфейс, то разделение свойств токена на фактические значения, константы, переменные и остаток можно сделать более явным, убрав при этом постфиксы. |
| Как представляется мультимножество transition mode? | В отличие от токенов и аннотаций, данное мультимножество не представляется через term. Вместо этого на этапе логического вывода ризонер формирует binding’и, как пары переменных и их значений для КАЖДОЙ входной дуги. |
| Многочисленность (multiplicity) basisSet’ов мультимножества transition mode всегда равна 1, почему? | Сопоставление (binding) значения переменной может быть только одно. |
| Как выглядит мультимножество токенов позиции, формируемое в ходе срабатываний переходов и при первичной инициализации. | В ходе выполнения переходов или вычисления первичной маркировки полученные примитивы или объекты составляют мультимножество базовый наборов с даными в формате JSON |
| Есть ли необходимость в глубоком сравнении объектов в процессе добавления объекта в мультимножество? | Необходимо уточнить этот момент.  а) JSON, сформированный в ходе вычислений, может совпадать со значением BasisSet’а маркировки позиции – необходимости нет, т.к. процесс детерминирован;  б) Можно сохранять метаданные о токенах в позиции и использовать их при добавлении новых токенов – необходимости нет;  в) Нежелательно: необходимость есть. |
| Как выглядит статическое (неизменяемое) мультимножество аннотации дуги после обработки ризонером? | Такое мультимножество включает составляющие части аннотации в формате JSON, позволяющие осуществлять подбор меток входной позиции или формировать метки в выходной позиции |
| Какова роль множества цветов (ColorSet)? | Множество цветов – это просто строка с именем множества. Реализация является частью конфигурации ризонера. |
| Как осуществляется проверка типа? | typeof !== “object” | “function”  ++  instanceOf === ColorConstructor  ++  проверка цепочки делегатов |
| Зачем в онтологии определения функций и констант? | Чтобы указать типы аргументов и возвращаемого значения, а также обозначить их вызов внутри term’ов |
| Необходима ли концепция Step в качестве промежуточной для Firing? | На данный момент ответственность этих концепций совпадает. Т.е. срабатывание firing производится при срабатывании перехода. Вероятно, концепция Step подразумевает, что для одного умышленного срабатывания могут выполняться несколько срабатываний одного перехода. В таком случае, лучше вернуться к этому вопросу при проработке дескрипторов ризонера. |
| Нужна ли концепция Operator в качестве абстрактного определения Function? | Концепция Function и так достаточно абстрактна, поскольку реализация закладывается в дескриптор ризонера. |
| Нужна ли концепция OperatorApplication как вызова функции с аргументами-term’ами? | В данный момент реализация не включает в себя процесс разбиения term’ов на составляющие, поэтому определения самого факта вызова функции в term’е должно быть достаточно |
| Нужна ли концепция OperationEvaluation как вычисление значения некоторой операции? | Поскольку данная концепция отражает композицию вызовов функций/операций, а процесс разбиения (как выяснилось ранее) не предусмотрен, то в ней также нет необходимости |
|  |  |
|  |  |
| В чем результат применения правил исполнения РСП в системе? | Данные правила определяют то, каким образом создаются сущности концепций, а также знание, необходимые для исполнения РСП, в частности, получения новых состояний |
| Как выглядят правила исполнения РСП? | Возможно, дескрипторы в формате JSON?  Можно динамически подгружать и парсить их в клиенте, после чего через интерфейсы настраивать работу JS-ризонера. |
| Доступны ли аксиомы онтологии для обеспечения полноценности описания модели (статического и динамического)? |  |
| Какие доступны улучшения онтологии и системы в целом? | - ингибиторные дуги  - сбрасывающие дуги  - прогон сразу нескольких наборов токенов с упором на transition modes (+ концепция step)  - временные метки, глобальное время |
| Какие части системы затрагиваются при внедрении улучшений? | - концепции и отношения онтологии  - аксиомы онтологии  - логика разрешенности перехода  - логика формирования токенов |
| Как производится процесс усовершенствования? | - изменяется корневая (core) онтология: классы, отношения и аксиомы  - изменяются правила исполнения РСП, формируется новый JSON дескриптор  - ризонер конфигурируется путем подгрузки JSON-файла дескриптора  - обновляется core-онтология на SPARQL сервере (для онтологии экземпляров обеспечивается обратная совместимость) |
| В дескриптор включаются шаблоны SPARQL-запросов к серверу? | Необходимо рассмотреть возможность написания абстрактных запросов, не требующих изменения в их структуре  (однако, при этом, мы теряем в детерминированности получаемых данных).  Это, прежде всего, касается клиента, для которого возможности конфигурации может не быть  Дескриптор не для клиента, а для ризонера на сервере!!! Вероятный ответ – да, включаются. |
| Какая **последовательность действий** при моделировании динамики функционирования РСП? | - на момент первичной инициализации или непосредственно после выполнения перехода ризонер выполняет действия по получению информации о разрешенных переходах  - создается экземпляр ризонера, обслуживающего онтологию по указанному URL  - на этапе первичной инициализации по команде ризонер получает и сохраняет список переходов и их параметров: id, условие guard, выполняемый код, id входных и выходных дуг и их аннотации (шаблоны), id входных и выходных позиции (+term нач. маркировки).  - также им вычисляются доступные term’ы ( начальные маркировки позиций) и формируются соответствующие данные  -V между выполнениями переходов ризонер запрашивает содержимое входных позиций (только данные мультимножеств) и формирует transition mode под каждую входную дугу  - ризонер выполняет SPARQL-запрос разрешенных переходов и помечает возможные комбинации и переходы  - на этом работа ризонера приостановлена, идет ожидание действий пользователя/внешнего возбудителя, где выбирается определенная комбинация для срабатывания перехода  - создается срабатывание firing для мультимножества transition modes  - ризонер приступает к обработке и запрашивает данные по последнему firing  - ризонер запрашивает содержимое выходных позиций и формирует (?) новые токены их маркировок  - переход к шагу V |
| Зачем на этапе инициализации получать какие-то данные? | Потому что предполагается, что об обновлении онтологии будет сообщено извне. При этом в остальных случаях (большую часть времени) процесс анализа будет выполняться быстрее (кол-во выполняемых переходов >> кол-ва обновлений онтологии) |
| Если для определения разрешенных переходов выполняется SPARQL-запрос, то как вынести логику определения их разрешенности в дескриптор? | Вероятно, дескриптор должен включать в себя фрагмент этого запроса, касающийся разрешенности. Т.е. по соглашению переход будет обозначен и предоставлен как ?t, а факт разрешенности будет формироваться, например, в ?is\_enabled, вся логика – в дескрипторе в формате SPARQL-запроса |
| Как можно было бы через дескриптор изменить логику, при которой в выходной позиции формируются токены, на обратную, при которой выходная позиция очищается? | За формирование в конечном счете отвечает SPARQL-запрос. Если сделать его фрагменты частью дескриптора, то и логика будет поддаваться изменению |
| Каков общий принцип использования дескрипторов? | Выносить максимум логики, касающейся динамики исполнения, в SPARQL-запросы, формирующиеся на основе данных из дескриптора. Для клиента – обобщенные SPARQL-запросы, допускающие изменения в концепциях онтологии |
| Что насчет реализации времени и множественного прогона modes? | Насчет множественного прогона: в этом случае с одним firing связаны сразу несколько мультимножеств transition mode, так же запрашиваются данные и выполняется формирование новых токенов. Это достижимо и без изменений в онтологии. Если подразумевается введение новой концепции Step, тогда изменяется процесс создания firing, относящегося к клиенту. В конечном счете все сводится к изменениям в SPARQL-запросах  Реализация времени вносит свои коррективы в момент между срабатываниями, где определяется глобальное время с разрешенным переходом. Вероятно, стоит ввести в логику процесс **повторного логического анализа** в случае отсутствия доступных разрешенных переходов. Для формирования временных меток все так же в дескрипторе изменяется фрагмент соответствующего SPARQL-запроса |
| Нужен ли дескриптор обработчика активных (изменяющих онтологию) запросов агента на сервере? |  |
|  |  |
|  |  |
| Что подразумевает распределенная структура системы моделирования? | Есть 3 варианта:  1) распределенное хранение данных (ресурсов);  2) распределенная обработка федеративных SPARQL-запросов (SERVICE);  3) **распределение узлов иерархических РСП** |
| Какой вариант распределенной структуры выбрать и почему? | Наиболее интересен 3 вариант, реализующий иерархические РСП в сети. Потенциальная польза от него больше:  + позволяет реализовывать микросервисы из иерархических моделей  + дает возможность вести разработку узлов параллельно |
| Что насчет федеративных запросов? | В приоритете такая структура, узлы которой имеют URI лишь связанных с ними других узлов. В таком случае, сбор информации возможен с помощью федеративных запросов или иных протоколов взаимодействия между узлами |
| Что из себя представляет узел иерархической РСП? | Узел является автономной ABox онтологией, имеющей входные и выходные узлы (позиции).  Узел используется **вместо перехода**.  Данные позиции инициализируются при загрузке онтологии на сервер адресами URL.  Передача данных между узлами осуществляется по протоколу UDP, т.к. подтверждение получения в нем не требуется |
| Что насчет TBox онтологии (core)? | Онтологию можно вынести на отдельный сервер по типу <https://www.w3.org/2002/07/owl>  Если допустить возможность хранения собственных экземпляров TBox, то функциональные возможности благодаря обратной совместимости разных версий онтологии будут ограничены лишь используемой версией |
| Как представляются оконечные узлы (позиции) в узле иерархии? | В узле объявляются экземпляры Port, связанные с определенными позициями. Для них определяется тип (input/output) и URI(URL) связанного узла |
| Где хранится информация о срабатываниях firings? | Экземпляры firing хранятся на каждом узле. Они создаются при срабатывании переходов в узле или при внешних входных воздействиях |
| Где хранится информация об ощих term’ах, вроде констант? | Эта информация могла бы храниться в корневой (core) онтологии на отдельном сервере |
| Как можно реализовать временные РСП в рамках распределенной структуры? | Требуется синхронизация глобального времени для узлов распределенной РСП.  Возможно, это реализуется федеративными запросами по узлам с целью определения ближайшего времени перехода |
| Как синхронизировать глобальное время между узлами иерархической РСП? | Есть вариант с рассылкой запросов от узла к соседним узлам. В запрос включается хэш, а также с ростом глубины запроса пополняется список URL узлов, пробрасывающих эти запросы по сети. Окончанием синхронизации считается получение ответов от всех соседних узлов, которым этот запрос был адресован.  Также, возможен федеративный запрос, извлекающий в ходе выполнения новые узлы сети. |
| Какие есть виды взаимодействий узлов иерархической сети? | Каждый узел отвечает за срабатывание только своих переходов.  При формировании меток в позиции, являющейся еще и выходным портом, узел пересылает связанному соседнему узлу SPARQL-запрос на формирование, который аналогичен выполненному внутри самого узла.  Все функционирование сводится к передаче меток через порты. Решение о выполнении перехода осуществляется агентами |
|  |  |
| Как исполняется сеть Петри? | По внешнему воздействию срабатывает случайный/заранее определенный разрешенный переход со случайной/заранее определенной комбинацией токенов |
| Как исполняется временная сеть Петри? | Во временных СП большую роль играет глобальное время, определяющее доступность токенов для перехода  Создается список будущих событий, находится ближайшее событие  Запускаются возбужденные переходы, при этом, параллельные события, доступные в один момент времени, выполняются прежде любых других будущих событий  Т.е. разрешенные переходы в момент времени срабатывают до тех пор, пока их список не станет пустым |
| Как исполняется иерархическая распределенная СП? | 2 варианта:  - комплексное выполнение переходов через глобальные запросы  - **сконцентрированное выполнение переходов в рамках узлов**  Второй вариант является приоритетным |
| Какое применение у реализуемой иерархической РСП? | Выполнять моделирование с использованием РСП в рамках сети |
| Как выполняется моделирование в рамках узлов? | Каждый узел содержит фрагмент сети и отвечает за логический вывод (ризонинг).  Сервер предоставляет агентам доступ (GET) к состоянию фрагмента сети, а также интерфейс (REST? **SPARQL**) для выполнения разрешенных переходов в рамках фрагмента сети.  Также он взаимодействует со связанными узлами для передачи токенов |
| Какие существуют подходы к организации подключения модулей/узлов? | - попарное подключение 1 к 1  - разделение общего ресурса |
| В чем плюсы и минусы попарного подключения? | + можно реализовать подписку на события  **+ в связке всех со всеми реализуется общий ресурс**  + упрощенная синхронизация и процесс подключения  + разделение портов на In, Out и I/O |
| В чем плюсы и минусы организации контактной позиции как общего ресурса? | - все порты являются двухсторонними I/O  - подключение всех со всеми |
| В чем заключается отличие реализуемой системы от классических иерархических СП? | Каждый узел сети является модулем. Допускается связь **1 – много** для модулей. |
|  |  |
|  |  |
| Как решается проблема синхронизации при выполнении переходов в узлах, имеющих общие порты? | На узлах генерируется уникальная последовательность (**хеш**), с помощью которой **резервируются** локальные порты, используемые при переходе.  Выполняется попытка **занятия** используемых портов на удаленных узлах. Хеш при этом используется для определения приоритета узла.  Если заняты удаленные порты, занимаются и локальные порты, после чего выполняется переход и порты освобождаются. |
| В чем плюсы и минусы такой синхронизации? | + хорошо работает **для иерархических** сетей (1 к 1: дочерний модуль к родительскому), позволяя определить и выполнить приоритетный переход  - не решает **проблему коллизий** при множественных связях между модулями (1 порт используется 2+ другими модулями), что приводит к отменам выполнения переходов на всех взаимодействующих модулях |
|  |  |
| Почему переменные не привязаны к экземпляру сети РСП (CPN instance)? | Возможно динамическое создание анонимных переменных. |
|  |  |
|  |  |
|  |  |
| Какие есть примеры подходящих для моделирования систем? |  |
|  |  |
|  |  |
|  |  |
|  |  |