

КЛАССОВАЯ ТИПИЗАЦИЯ УЗЛОВ В МЕТА-АССОЦИАТИВНЫХ ГРАФАХ

А.Е. Мисник (*anton@misnik.by*)

Белорусско-Российский университет,
Республика Беларусь, Могилев

Статья посвящена теоретическому обоснованию классовой типизации узлов в мета-ассоциативных графах – гибридной модели представления знаний, объединяющей графовую структуру и объектно-ориентированные принципы. Описываются формальная структура, иерархические и ассоциативные связи, механизмы наследования и полиморфизма, а также способ хранения схемы знаний внутри самой модели. Показано, как типизация упрощает автоматизацию обработки данных, оптимизирует запросы и повышает адаптивность информационных систем. Отмечена важность унификации хранения и доступа к знаниям, обеспечивающей масштабируемое расширение онтологии без миграции данных. Полученные результаты расширяют теорию метаграфов и служат основой для разработки семантически богатых высокопроизводительных систем управления знаниями.

Ключевые слова: мета-ассоциативные графы, инженерия знаний, онтологии, сложные системы.

Введение

Одним из инструментов инженерии знаний выступают онтологии – формализованные описания понятий предметной области и отношений между ними. Онтологии позволяют создавать общие семантические основы для интеграции данных из разнородных источников, обеспечивая единое понимание информации различными компонентами сложных систем. Такая семантическая унификация необходима, например, для современных кибер-физических систем, где взаимодействуют программные компоненты и физические процессы: единая онтология служит концептуальным интерфейсом, через который различные подсистемы «разговаривают» на одном языке.

Традиционные графовые модели – такие как семантические сети, фреймовые системы, графы знаний на основе RDF/OWL – продемонстрировали возможности их применения, однако, по мере роста сложности моделируемых предметных областей и требований к гибкости знаний, выявляются ограничения этих подходов. RDF-графы оперируют простыми триплетами «субъект–свойство–объект» и не предполагают встроенного механизма для задания процедурных аспектов знаний (таких как поведение объектов или события), что затрудняет моделирование динамики систем [Argüez et al., 1998]. Объектно-ориентированные базы данных, хотя и близки по духу к онтологиям, не получили широкого распространения в силу недостаточной гибкости стандартных языков запросов и средств онтологического вывода.

Перспективным кажется подход, сочетающий объектно-ориентированное представление (классы, объекты, методы) с графовой природой знаний (узлы и связи между ними). Одной из таких гибридных моделей является мета-ассоциативный граф. Мета-ассоциативные графы были предложены как расширение концепции метаграфов для нужд онтологического инжиниринга. Метаграф в классическом понимании – это обобщение направленного графа и гиперграфа, позволяющее отображать отношения между множествами объектов (например, гиперребро может связывать одновременно несколько вершин) [Basu et al., 2007]. Мета-ассоциативный граф развивает эту идею, вводя единое понятие узла, объединяющее обычную вершину и мета-вершину (вершину, представляющую множество других вершин). Кроме того, каждый узел наделяется внутренней структурой – именем, атрибутами, событиями и методами – что привносит в граф элементы объектно-ориентированного описания.

Актуальность мета-ассоциативных графов обусловлена потребностью в моделях знаний, способных гибко эволюционировать, отражать многоуровневые ассоциации и поддерживать встроенную логику [Bobryakov et al., 2022]. В кибер-физических и других сложных системах нередко возникает ситуация, когда структура знаний должна оперативно перестраиваться при появлении новых типов сущностей или отношений, а инструменты поиска информации должны «понимать» новые взаимосвязи без полной переработки модели. Мета-ассоциативные графы, сочетая графовую гибкость с принципами инкапсуляции и наследования, предлагают решение этой проблемы. Они позволяют описывать сложные системы квази-иерархически: с одной стороны, присутствует иерархия классов и компонентов (дерево или сеть отношений «родитель–потомок»), с другой – допускаются рекурсивные связи и латеральные ассоциации между элементами, выходящие за рамки строгого дерева. Благодаря этому удаётся достичь высокого уровня точности и глубины при моделировании взаимосвязей реального мира.

Мета-ассоциативные графы

Мета-ассоциативный граф – это развитая модель графового представления знаний, в которой каждому узлу приписывается класс (тип), определяющий его структуру и поведение. В традиционных онтологиях (например, OWL) существует разграничение между классами и экземплярами: класс задаёт набор свойств, которыми могут обладать экземпляры. Мета-ассоциативный граф перенимает эту идею и интегрирует её непосредственно в узлы графа, делая классовую типизацию частью самой графовой модели.

Формально узел мета-ассоциативного графа можно представить как кортеж, включающий несколько составляющих: идентификатор, атрибуты, события и методы. Предложено следующее определение узла (N) мета-ассоциативного графа:

$$N = \{ I, \{AS\}, \{EV\}, \{M\} \},$$

где I – имя узла (уникальный идентификатор или метка, однозначно определяющая данный узел); AS – множество ассоциативных атрибутов узла; EV – набор событий, связанных с узлом; M – набор методов узла. Ассоциативные атрибуты (AS) при этом представляют собой совокупность значений и ссылок на другие узлы графа. Формально это можно записать как:

$$AS = \{ V, N \},$$

то есть атрибут узла может содержать либо скалярное значение (число, строку, дату и т.п.), либо ссылку (или несколько ссылок) на другие узлы мета-ассоциативного графа. Такое определение объединяет в одном понятии и традиционные «атрибуты-значения», и «атрибуты-связи» (ассоциации). То есть, граница между данными и связями размывается: атрибут может непосредственно содержать связь на другой объект, превращаясь тем самым в своеобразное ребро, встроенное в узел [Misnik, 2022].

События (EV) – это совокупность определённых ситуаций или изменений состояния, на которые узел способен реагировать. События могут быть разных типов: изменение значения атрибута, появление связи, удаление узла, достижение некоторого условия и т.д. Связав определённые обработчики с такими событиями, система приобретает свойство реактивности – способность запускать определённую логику при наступлении тех или иных условий.

Методы узла (M) – это функциональные возможности, заложенные в объект. Метод можно рассматривать как функцию или процедуру, оперирующую состоянием данного узла (его атрибутами, связями) и, возможно, влияющую на другие узлы. В отличие от чисто декларативных онтологий (OWL, RDF), где знание задаётся только в виде фактов и логических аксиом, наличие методов позволяет включить в модель элементы алгоритмической логики [Gomez-Perez et al., 2004]. Методы могут вызываться

извне (пользовательским кодом или интерфейсом системы) или внутри самой системы – например, как реакция на событие. Наличие методов и событий фактически превращает пассивную модель знаний в активную онтологию, где объекты могут обладать элементами поведения.

Мета-ассоциативный граф можно представить как ориентированный граф, узлы которого являются объектами (в смысле объектно-ориентированного подхода). Дуги (ребра) графа при этом служат для выражения базовых отношений иерархии между узлами – по определению, это ненагруженные (без собственных атрибутов) связи «родитель–потомок». Иерархическими ребрами могут оформляться, в частности, отношения между классами и подклассами, а также между классами и экземплярами, или отношения композиции (часть–целое) между объектами. Важное упущение мета-ассоциативной модели состоит в том, что все связи унифицируются под общим понятием “иерархия”. Фактически, любая направленная зависимость представляется дугой: более общий объект связывается с более частным или составным объектом как «родитель с потомком». Такой подход охватывает как классические отношения наследования, так и включение/агрегацию. Несмотря на использование единого типа связи, семантическая разница между «классическим» наследованием и композицией сохраняется за счёт смысла узлов: родитель, помеченный как класс, интерпретируется как обобщённое понятие, а родитель, помеченный как сущность-целое, трактуется как агрегирующий объект. Однако формально и то, и другое – просто отношения иерархии в графе.

Следует отметить, что мета-ассоциативные графы предложены именно как развитие теории метаграфов, поэтому важно понимать их связь и отличия. Метаграф – это структура, обобщающая понятие графа, где вершины могут группироваться в метавершины, а рёбра отображают один набор вершин в другой набор. Такой аппарат удобен для моделирования, например, зависимостей «многие-ко-многим» или задач моделирования потоков данных, где каждое действие рассматривается как отображение множества входных документов в множество выходных [Garanuyk, 2019], [Chernenkiy et al., 2018]. Однако классический метаграф, будучи мощнее обычного графа, всё же остаётся статической конструкцией: в нём нет понятия событий, отсутствует механизм для описания поведения при изменениях структуры. Кроме того, разделение на вершины, мета-вершины и атрибуты в ряде случаев оказывается слишком ригидным, особенно в динамично изменяющихся системах знаний – порой грань между атрибутом и отдельным узлом становится условной. Например, некоторая характеристика объекта (атрибут) может со временем превратиться в самостоятельный объект системы (узел) по мере роста её значимости. В классической модели требовалась бы перестройка: удаление атрибута, введение новой вершины и переопределение связей.

Мета-ассоциативный граф устраняет эти недостатки за счёт унификации понятий вершины и мета-вершины до единого понятия узла и включения в описание узла дополнительных компонент (событий и методов). По сути, любой узел мета-ассоциативного графа может выступать и как отдельный объект с собственными данными, и как контейнер для других узлов. А событие и методы, «встроенные» в узел, предоставляют естественный механизм реагирования на изменения и инкапсуляции поведения. Именно благодаря этому мета-ассоциативные графы ориентированы на использование в онтологическом инжиниринге – они позволяют заложить в модель знаний не только статические связи, но и динамическую логику реакций внешние и внутрисистемные изменения.

Классовая типизация узлов в мета-ассоциативных графах

Классовая типизация узлов напрямую заимствует базовые принципы объектно-ориентированного программирования: инкапсуляцию, наследование и полиморфизм. Узел мета-ассоциативного графа инкапсулирует данные (атрибуты) и связанные с ними методы, образуя автономный объект, самостоятельно управляющий своим состоянием. Наследование проявляется в том, что класс узла может быть организован в иерархию подчинённых классов, перенимающих свойства и поведение базовых классов. Полиморфизм в контексте знаний означает, что разные по типу узлы могут обрабатываться единообразно через интерфейс родительского класса. В онтологии это выражается в возможности задавать запросы или действия на уровне обобщённого класса и применять их к любому подтипу.

Заметим, что класс в мета-ассоциативном графе сам может быть представлен узлом графа (т.е. классы – такие же объекты верхнего уровня, обладающие атрибутами, связями с родительскими классами и т.д.). Это значит, что онтология (схема) системы может быть выражена теми же средствами, что и сами данные. Данный подход соответствует философии метаданных, когда описание данных хранится вместе с данными. Таким образом, мета-ассоциативный граф способен хранить не только факты предметной области, но и свою собственную схему (иерархию классов, описание атрибутов и методов), фактически выступая в роли самодокументируемой модели.

Одно из ключевых достоинств введения классов в графовую модель знаний – это унификация обработки узлов. Имея классовую типизацию, система приобретает способность обрабатывать все узлы одного типа по единым правилам, что упрощает код, повышает надёжность и облегчает сопровождение знаний.

Рассмотрим каким образом классы узлов позволяют унифицировать и автоматизировать работу с данными.

Единый интерфейс и полиморфизм. Когда узлы структурированы по классам, возможно определение общих операций, применимых ко всем узлам данного класса или иерархии классов. Полиморфная унификация повышает гибкость системы: логика работы описывается один раз, а применяется к многим разнотипным объектам.

Повторное использование и модульность. Классовая организация знаний способствует повторному использованию фрагментов онтологии и связанных процедур. Общие свойства (атрибуты) и методы можно вынести в базовые классы. Если несколько категорий объектов имеют нечто общее, нет необходимости дублировать информацию – достаточно сформировать класс-родитель с общими элементами, что соответствует принципу «не повторяться» (DRY), и считается хорошей практикой проектирования. В инженерии знаний повторное использование проявляется и в том, что целые фрагменты онтологии – классы с поддеревьями – могут служить шаблонами для различных приложений. Такие классы-шаблоны могут быть оформлены как модули или библиотеки знаний и при необходимости подключаться к основному графу.

Автоматизация задач на основе схемы. Когда для узлов определены схемы (наборы атрибутов определённого типа), это открывает возможности для автоматической генерации кода, пользовательских интерфейсов и других вспомогательных средств. Такой подход активно применяется в информационных системах (технологии low-code, no-code), где метаданные о типах сущностей используются для максимального сокращения ручного кодирования. Мета-ассоциативный граф сам хранит метаданные (классы и атрибуты), что даёт возможность встроенным инструментам генерировать интерфейсы и шаблоны обработки.

Унификация хранения и доступа. Классовая типизация подразумевает, что объекты одного класса хранятся и организованы схожим образом. Данный аспект можно использовать для оптимизации хранения данных: например, для каждого класса можно создать специализированную структуру, хранящую все экземпляры этого класса и быстро предоставляющую к ним доступ по основному ключу (имени или ID). В реляционных СУБД подобное разделение по схемам – норма (таблица на класс), но в графовых хранилищах классическая модель RDF не предусматривает физического разбиения данных по типу (все триплеты лежат в общей массе). В мета-ассоциативном графе же мы явно оперируем объектами классов, что позволяет реализовать гибридное хранение: часть данных (например, скалярные атрибуты) хранить в структурированной форме, а ссылки – отдельно как указатели на другие объекты. Такой подход улучшает когерентность данных в памяти: однородные объекты размещаются близко друг к другу, что положительно сказывается на возможностях кэширования и скорости обхода. Кроме того, зная класс объекта, можно

напрямую переходить к операциям, специфичным для этого класса, без многочисленных проверок во время выполнения. По сути, система получает «знание», как обрабатывать объект, глядя на его принадлежность к конкретному классу – далее задействуется соответствующий метод или алгоритм, что аналогично виртуальным методам в объектно-ориентированном программировании, когда вызов процедуры объекта выполняется по-разному в зависимости от реального класса объекта.

Управление жизненным циклом через классы. Наличие классов позволяет централизованно контролировать создание, удаление и преобразование объектов. Можно определить методы-конструкторы на уровне класса – процедуры, которые вызываются при создании нового экземпляра. Они могут выполнять дополнительную инициализацию, проверять ограничения или автоматически связывать новый объект с другими компонентами. Аналогично, при удалении объекта класс может предписать каскадное удаление связанных объектов (для отношения часть-целое) или другие изменения в данных. Классы выступают как «супервизоры» экземпляров, обеспечивая стандартные действия по управлению их жизненным циклом, что обеспечивает ещё один уровень унификации: вместо того чтобы каждый раз при добавлении узла выполнять набор однотипных шагов, класс гарантирует выполнение всех необходимых процедур (через конструкторы, события или системные методы).

Перечисленные особенности показывают, что классовая типизация узлов приводит к стандартизации способов обработки объектов. Система знаний с самого начала проектируется структурированной и модульной. В результате, инженеры по знаниям получают более упорядоченную, предсказуемую среду. Добавляя новый класс в систему, они сразу определяют, какие атрибуты он имеет, как взаимодействует с другими классами, какие методы доступны – и все экземпляры этого класса будут вести себя согласно заданному шаблону, что резко контрастирует с неструктурированными подходами (произвольные графы без схемы), где каждый узел может иметь произвольные свойства и заранее неизвестно, как с ним работать. Мета-ассоциативный граф с типизированными узлами вводит «сильную» семантическую типизацию в графы знаний, аналогично тому, как статическая типизация в языках программирования помогает идентифицировать ошибки и упрощать разработку. Классы узлов выполняют роль «каркаса», на котором строится обработка знаний.

Семантические связи и оптимизация запросов

Связи в мета-ассоциативном графе несут определённый смысл (семантику) в контексте предметной области. Благодаря введению классов и строгой организации связей, появляется возможность оптимизировать запросы к графу, опираясь на знание о типах узлов и видах связей между

ними. Мета-ассоциативный граф по умолчанию трактует все рёбра как иерархические (родитель–потомок). Однако семантически эти связи могут представлять различные отношения.

Связь класс–подкласс отражает отношение наследования между классами в онтологии. Такая связь обозначает, что все свойства и методы родительского класса применимы к потомку (стандартное наследование). Семантически это «является разновидностью» (is-a). В графе она помогает быстро идентифицировать, что узел принадлежит более общему классу, и применить к нему соответствующие обобщённые правила. Система может кэшировать для каждого класса список всех подклассов и потомков, чтобы такие запросы выполнялись мгновенно, а не через обход всего графа.

Связь класс–экземпляр – фактически, экземпляр считается потомком своего класса. Это отношение аналогично `rdf:type` в RDF, но в мета-ассоциативном графе оно выражено просто как ребро от класса к объекту. Семантически – «является экземпляром». Для каждого класса целесообразно автоматически поддерживать индекс или список всех его экземпляров (потомков). Тогда запрос «выбрать все экземпляры класса» может быть выполнен очень быстро, обращением к индексу, вместо обхода всего графа в поисках узлов с определённым атрибутом типа. При добавлении или удалении связей класс–экземпляр индекс обновляется. Это эквивалентно тому, как реляционная база использует индекс по столбцу класса для выборки строк – но тут мы опираемся на семантику графа.

Связь часть–целое (композиция) описывает, что один узел является компонентом другого. В графе это тоже выражается как связь родитель–потомок (целое – родитель, часть – потомок). Семантически – «является частью» (part-of). Зная, что данное отношение иерархическое (а оно является по-умолчанию таковым для мета-ассоциативного графа), можно применять рекурсивные запросы ограниченно. Например, можно заранее вычислять транзитивное замыкание для отношений часть–целое (подобно тому, как в реляционной базе могут храниться материализованные пути в дереве).

Ассоциативные связи (неиерархические) используются когда необходимо отразить отношение, которое не обладает строгой древовидной структурой. В случае мета-ассоциативного графа такую связь можно моделировать через ассоциативный атрибут. Если такие связи часты и критичны, можно также хранить их в индексированной форме: каждому типу ассоциации соответствует таблица индекса. Система, понимая смысл отношения, может выбирать оптимальный способ хранения (в зависимости от преобладающих запросов).

Иерархия задач/процессов – если узлы могут представлять события или процессы, между ними тоже возможны отношения (например, последовательность операций). Их можно отразить либо иерархически (как подзадачи в задаче), либо тоже через ассоциативные поля вида «следую-

щая стадия». Семантика таких связей помогает оптимизировать планирование: но это уже более специфично и выходит за рамки общей модели, поэтому здесь упомянем лишь, что любые предопределённые типы связей можно учитывать при оптимизации.

Когда запрос к базе знаний сформулирован, он обычно оперирует определёнными условиями на свойства узлов и их связи. В мета-ассоциативном графе наличие классов и иерархий позволяет сузить область поиска задействованных данных и тем самым ускорить выполнение запроса. Приведём типичные ситуации.

Поиск по классу. Запрос «найти все объекты класса X с условием ...» – один из самых распространённых в системах знаний. В классической графовой БД без индексации по типу пришлось бы перебрать все узлы или все узлы, помеченные определённым свойством. В случае мета-ассоциативного графа можно сразу взять список его потомков (что можно сделать очень быстро при наличии индекса) и далее отфильтровать по необходимому атрибуту. Если искомый класс имеет подклассы, то по семантике наследования мы знаем, что все они тоже относятся к этому классу, и потому их потомки тоже должны учитываться. Система может автоматически пройти по дереву подклассов (или иметь подготовленный список всех подклассов) и объединить списки их экземпляров.

Использование отношений для ограничения поиска. Предположим, запрос: «найти все объекты класса A, которые связаны с объектом класса B». Даже без явного индекса, сам факт того, что рассматриваются только узлы класса A, уже отсекает все прочие узлы. Очевидно, что в больших графах такой подход экономит огромное количество операций.

Предикаты на иерархические связи. Запросы могут требовать учёта иерархии. Запрос сводится к обходу поддерева определённого класса на определённую глубину. Можно оптимизировать такой обход за счёт хранения указателей на родителя у каждого узла (для быстрого подъёма) и/или списка потомков (для быстрого спуска).

Семантические сокращения путей. Будучи осведомлена о типах связей, система может упрощать логический путь запроса. Если бы граф не имел схемы, система могла бы выполнять произвольный поиск путём соединения всех триплетов. Но имея схему запрос может быть трансформирован в две стадии: 1) получить все объекты нужного (по индексу или списку условий); 2) отфильтровать эти объекты по нужному условию. Каждая стадия использует конкретную связь, а не обходит весь граф.

Благодаря описанным возможностям, мета-ассоциативный граф обеспечивает высокую производительность при выполнении типичных запросов инженерии знаний, несмотря на общую сложность модели. Если сравнить с RDF-хранилищем (триплетным), то там любой сложный запрос распадается на множество сочетаний триплетов и требует соединения по

общим узлам [Rasheed et al., 2019]. При большом числе триплетов (миллионы и более) такие соединения могут быть ресурсозатратны, поэтому RDF-движки применяют тяжёлые оптимизации (многоиндексные схемы, сжатие). В случае мета-ассоциативного графа некоторая информация уже находится в самой структуре: класс задаёт область поиска, иерархия задаёт ограничение путей. То есть запросы в мета-ассоциативном графе могут более точно адресовать нужные узлы. Даже смешанные запросы (с условиями на значения и на связи) выигрывают: условие на значение может проверяться только на узлах нужного класса, не затрагивая остальные.

Оценка производительности

Приведём пример количественного бенчмарка. Пусть в графе 100 000 узлов, из них 10 000 узлов класса «Студент». Запрос: «студенты со средним баллом > 80». В триплетной модели нужно просмотреть все узлы с свойством «средний балл», а ещё сначала найти кто является студентом (или хранить предикат `rdf:type`). Даже при индексе по `rdf:type`, мы получаем 10 000 кандидатов, потом среди них отфильтровываем по «средний балл» — итого 10 000 проверок. В мета-ассоциативном графе мы обращаемся к узлу класса «Студент», сразу берём 10 000 потомков (или имеем их список) — 10 000 проверок, в целом, кажется, что порядок сложности тот же. Но если запрос сложнее, скажем: «студенты факультета А с «средний балл» > 80». Без семантики пришлось бы связывать студента с факультетом: либо через два-три триплета (студент -> кафедра -> факультет), соединяя результаты. С семантикой: мы берём узел факультета А, собираем всех студентов (потомков по иерархии организационной структуры) — предположим их 500. Затем отфильтровываем по «средний балл» — всего 500 проверок. Это в 20 раз меньше, чем проверять 10 000 узлов, как в ситуации, когда мы бы сначала отобрали всех студентов, а потом проверяли их принадлежность факультету. Экономия растёт с увеличением общего размера графа, если семантические ограничения сильно сужают выборку.

Заключение

Можно констатировать, что мета-ассоциативные графы с классовой типизацией узлов представляют значимый шаг вперёд в эволюции средств инженерии знаний. Они устраняют разрыв между статичными моделями данных и динамическими требованиями приложений, позволяя описывать и немедленно использовать знания в их естественной комплексности. Как и всякая новая парадигма, этот подход требует освоения и доводки, однако первые результаты — как теоретические, так и практические — демонстрируют его жизнеспособность и эффективность. Для достижения описан-

ных преимуществ необходима поддержка индексов и кэшей по ключевым семантическим признакам. Сама по себе графовая модель – это описание, а чтобы ускорить запросы, система хранения должна включать структуры данных: для каждого узла-класса – быстрый доступ к его экземплярам; для каждого узла-объекта – быстрый доступ к компонентам; для каждого именованного ассоциативного атрибута – индекс объектов по значению/ссылке этого атрибута. Создание и поддержание таких индексов требует дополнительных ресурсов, но без этого семантическая информация не будет эффективно использована. В реляционных СУБД администратор сам решает, на какие поля ставить индексы. В мета-ассоциативном графе можно автоматически ставить индексы на все связи, определённые в схеме, либо на те, которые часто используются в запросах (что можно определить динамически, профилировав систему). Такие избыточные хранения (денормализация) оправданы для ускорения, и они контролируемо реализуются на уровне системы.

Список литературы

- [Arpirez et al, 1998] Arpirez J., Gomez- Perez A., Lozano A., Pinto S. (ONTO) 2Agent: An ontology – based WWW broker to select Ontologies // Workshop on Applications of ontologies and Problem Solving Methods. ECAI'98.
- [Basu et al, 2007] Basu A., Blanning R. Metagraphs and their applications. – Springer, 2007. – 174 p.
- [Bobryakov et al., 2022] Bobryakov A.V., Borisov V.V., Misnik A.E. and Prokopenko S.A. Design and Implementation of Information-Analytical and Industrial and Technological Processes in Production Based on Neuro-Fuzzy Petri Nets // 2022 VI International Conference on Information Technologies in Engineering Education (Inforino). – 2022. – P. 1-6. – doi: 10.1109/Inforino53888.2022.9782997.
- [Bobryakov et al., 2019] Bobryakov A.V., Borisov V.V., Gavrilov A.I., Tikhonova E.A. Compositional Fuzzy Modeling of Energy- and Resource Saving in Socio-Technical Systems // EAI Endorsed Transactions on Energy Web and Information Technologies. – DOI: 10.4108/eai.12-9-2018.155863.
- [Borisov et al., 2021] Borisov V.V., Zakharchenkov K.V., Kutuzov V.V., Misnik A.E. and Prokopenko S.A. Modeling educational processes based on neuro-fuzzy temporal Petri nets // Applied Informatics. – 2021. – Vol. 16, No. 4. – P. 35-47. – DOI: 10.37791 / 2687-0649-2021-16-4-35-47.
- [Chernenkiy et al., 2018] Chernenkiy V.M., Gapanyuk Yu.E., Revunkov G.I., Andreev, A.M., Kaganov Yu.T., Dunin I.V. The Principles and the Conceptual Architecture of the Metagraph Storage System / In: Manolopoulos Ya., Stupnikov S. (eds.) // 20th International Conference, DAMDID/RCDL 2018, Revised Selected Papers, CCIS. – Vol. 1003. – P. 73-87.
- [Gapanyuk, 2019] Gapanyuk Yu.E. Metagraph Approach to the Information-Analytical Systems Development // In: Proceedings of the 6th International Conference Actual Problems of System and Software Engineering, Moscow, Russia, 2019. – P. 428-439.

- [Gomez-Perez et al., 2004]** Gomez-Perez A., Fernández-López M., Corcho O. Ontological Engineering: With Examples from the Areas of Knowledge Management, E-Commerce and the Semantic Web. – 2004.
- [Misnik, 2022]** Misnik A.E. Ontological Engineering on Metagraphs Basis // 2022 VI International Conference on Information Technologies in Engineering Education (Inforino). – 2022. – P. 1-6. – doi: 10.1109/Inforino53888.2022.9782909.
- [Rasheed et al., 2019]** Rasheed B., Popov A.Yu. Network graph datastore using DiSc processor // In: Proceedings of the 2019 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering, ElConRus. – 2019. – P. 1582-1587.