Представление трансформаций MDA в виде логических объектов

Рассматривается задача моделирования программного обеспечения, которая представлена набором исходных моделей, а трансформация – на основе логического вывода. Исходные модели преобразуются в RDF-графы и, затем, обрабатываются системой знаний, структурированной в сеть объектов. При этом существует возможность использования различных нотаций, например таких, как UML, SysML, CMMN, BPMN2.0, графов RDF и результаты анализа исходного кода других систем, если реализовать соответствующий модкль преобразования. Объекты представлены в языка программирования LogTalk. Сценарий трансформации задается в виде объектов, делающих запросы друг к другу, и реализующих, в целом, парадигму модельно–управляемой архитектуры (Model Driven Architecture).

Использования такого подхода к трансформации позволяет разрабатывать исходный код каркасов программных систем еще на этапе общего дизайна, включать в трансформацию различные источники модельных данных, задавать и структурировать знания. Приводится пример использования разрабатываемых инструментов для представленя модулей прикладного пакета Mothur в виде блоков визуального потокового программирования системы Rapidminer

модельно-управляемаяархитектура, логический вывод, RDF, LogTalk

# Введение

Основная проблема при разработке сложных информационных систем (ИС), а также друхих систем автоматизации протизводства, – это сложность, коотрая выражается в сложности испольуемых структур данных, интегрированием с другими системани, необходимости этапа быстрого прототипирования на этапе дизайна системы. Если проводить этап порождения исходного кода на этапах дизайна системы, тогда можно порождать прототипы системы сазу же как сформирована формализованна модели разрабатываемой ИС. Существующие CASE-системы реализуют стандартные подходы к проектированию и поэтому не популярны в разработке WEB-приложений, в небольших организациях и стартапах, где процесс разработки характеризуется с высоким уровнем неопределенности требований и спецификаций.

Полноценная поддержка быстрого прототипирования требует разработки выразительных средств программирования трансформаций из комплекса разнородных моделей, представляющих различные аспекты ИС, например, UML, SysML, BPMN, которые задаются при помощи соответствующих визуальных инструментов. Программирование трансформаций необходимо стелать близким к обычному традиционной программированию, позволяющему, в частности, создавать из существующих специализированные версии, а также накапливать опыт в библиотеках объектов и компонент. Внедрение объектно-ориентированного программирования позволило бы создавать трансформации, структурируя базу знаний, позволяя проводить манипуляции с наборами знаний, инкапсулировать знания в трансформационные компоненты.

Модельно-управляемая архитектура (Model Driven Architecture, MDA) – это наиболее изученный и развитый трансформационный подход, выходящий за ограничения CASE–систем. MDA представляет процесс порождения кода подсистем ИС как многостадийный процесс преобразования исходной визуальной модели. Код генерируется из так называемой платформо-зависимой модели (Platform Specific Model, PSM), которая представляет реализацию ИС с включением специфики программно-аппаратной платформы... PSM получается из платформо-независимой модели (Platform Independent Model, PIM), которая описывает ИС на более абстрактном уровне. Часть PIM порождается из вычислительно-независимой модели (Computational Independent Model, CIM), описывающей ИС и ее окружение на концептуальном, организационном, системном уровнях, абстрагированных, в целом, от преобразования информации и вычислений.

CIM и PIM обычно задаются набором SysML, CMMN, UML и других визуальных моделей, которые сохраняются стандартным способом в виде XMI-файлов[[1]](#footnote-21). Кроме этих моделей дополнительные данные для трансформации могут быть получены из Семантического Веба (СВ), например, определения онтологии, названия сущностей в различных естественных языках, текущее состояние процессов, и так далее. Популярные способы трансформации модели, как ATL[[2]](#footnote-22), основаны на трансформации одной XMI-моделm в другую XMI, а затем уже в код. Привлечение данных СВ достаточно проблематично.

Мы предлагаем более общий подход, где для всех исходных типов моделей разрабатываются конвертера в формат RDF[[3]](#footnote-23), в стандартное представление данных и знаний в СВ, а трансформация программируется в виде объектно-ориентированной логической программы на языке LogTalk. Это позволяет нам представлять трансформации как сценарии из объектов, обменивающихся сообщениям и запрашивающих модельные данные из графа СВ. Объектное программирование позволяет нам структурировать, управлять базой знаний при помощи инкапсуляции, наследования, расширения и композиции.

*Объектом* исследования является разработка технологии реализации трансформации как системы знаний такой, чтобы была возможность обработки модельных данных CIM и PIM из разнородных источников. *Предметом* статьи является описание предложенного выше представления данных и методов преобразования моделей.

# Архитектура инструментов MDA

Архитектура средства трансформации основывается на обеспечения единообразия языка правил трансформации язык и [[предикативного]] представления входных данных. Поэтому мы требуем, чтобы либо хранить все входные модели данных в виде графов СВ при помощи соответствующих онтологий, или создавать конвертер, представляющий исходные модели в виде таких графов. Графы хранятся в виде RDF-файлов или загружаются с сервера через точки доступа SPARQL. Общая архитектура средств разработки представлена на рисунке [[fig:archi]](#fig:archi).

Инструменты MDA состоят из следующих основных компонентов:

* сервер исходных моделей (2), который хранит модель данных в виде RDF-троек;
* механизм трансформации (1), который запрашивает данные с сервера моделей и других SPARQL-совместимых серверов (4);
* конвертеры форматов представления моделей CIM (3) и PIM (6);
* сервис полнотекстового индекса (5).

Все компоненты взаимодействуют при помощи сетевых протоколов и используют форматы HTML, TXT, XML, JSON в кодировке UTF-8. [[Inference engines also support SPARQL protocol for issuing subqueries]].

Подсистема MDA представляет собой набор модулей трансформации (Т–модули). Каждый Т–Модуль в общем случае является параметризованным объектом LogTalk. Объект запрашивает у сервера моделей предметной области. Сервер построен на базе системы ClioPatria . Объект может запрашивать также другие объекты и другие сервера онтологий. В полученные данные о структуре моделей анализируются и распознаются заданные структурные комбинации. Результаты кэшируются в состояниях объектов или индексирутся механизмом индексирования текстов. Совокупность состояний объектов представляет собой PSM, которая на последнем этапе трансформации преобразуется в исходный код и начальные данные.

Исходные данные моделей CIM и PIM преобразуются в графы RDF специальными адаптерами. Например, на рисунке [[fig:archi]](#fig:archi) CIM (3) сконструирована в редакторе Modelio и, затем, преобразована в тройки в модуле “Конвернтер XMI в RDF”. Аналогичный конвертер разработан и для PIM, представляемой диаграммами UML. Такой подход позволяет расширять инструмент новыми видами моделями благодаря такой модульности.

Одним из вспомогательных RDF–совместимым источником данных является сервер DBPedia.org, содержащий именованные константы в различных языках, а также полезные отношения сущностей, представленных в Википедии. Эта информация может быть использована для описания свойств элементов пользовательского интерфейса для редактирования объектов ИС, заданных в исходной модели. Например, мы можем использовать данные DBPedia для формирования атрибутов title и placeholder элемента input, а также соответствующий текст метки (label) с учетом локальных настроек пользователя и help-текста.

# Представление данных моделей в RDF

Последние 20 лет разработки технологий СВ привели к созданию стандартов описания большого количества предметных областей, причем большинство из них стандартизованы консорциумом W3C. Наличие этих стандартов позволяет разработчикам ИС использовать глобальные ссылки URI для идентификации конкретных объектов, а также стандарты представления онтологий RDF, RDFa, TTL. Для большинства сред программирования реализованы библиотеки поддержки технологий СВ. В результате, отрасль ИТ постепенно на глобальном уровне накапливает и стандартизирует знания и данные предметных областей. Именно поэтому, было решено использовать СВ как основной формат представления моделей, что обогащают имеющиеся знания предметных областях дополнительной спецификой. С другой стороны, преобразование сложных древовидных структур, используемых для представления модели, в набор троек, т.е. фактов для машинного вывода, позволяет использовать Prolog в качестве основного механизма трансформации.

Согласно рекомендациям СВ, каждое описание модели должно быть помечено глобальным URI и, для удобства использования, некоторым синонимом-префиксом. Все отношения и атрибуты должны быть формально описаны в соответствующей онтологии, предпочтительно являющейся стандартной и общедоступной. Имена сущностей и их взаимосвязей, представленных в исходном формате визуального редактора, должны быть конвертируемыми в RDF по запросу во время трансформации.

## Определение сущностей

Идентификация сущностей и их поименование в RDF [[многоаспектно]]. Сущность и отношения существуют, если они где-то определены или где-то есть на них URI-ссылка. Поименование сущностей осуществляется при помощи отношений, *например*, dc:title и rdfs:label. Субъектом отношения является URI сущности, а объектом – литерал (строка) и метка естественного языка, задаваемая дополнительным атрибутом. Каждый элемент и отношение в нотации модели (метамодели) должны иметь соответствующие обозначения в онтологическом представлении.

Так как сущности глобально идентифицированы, их наличие в различных моделях обозначают один и тот же объект, например, в BPMN-диаграмме объект – запись базы данных (или документ) может быть создан из типа, определенного в UML диаграмме классов. Межмодельные ссылки также делаются при помощи URI или префиксной формы RDF <prefix>:<identifier>, где <prefix> – это аббревиатура подграфа, представляющего модель, в которой задан <identifier>. Большинство средств визуального редактирования UML не ограничивают имена идентификаторов, так же как и семантику структур, предполагая, что она определяется трансформацией.

## Средства расширения семантики моделей

Гибкие инструменты моделирования также позволяют дизайнеру расширить семантику структур модели (метамодели). Например, все диаграммы UML поддерживают *стереотипы* (stereotypes), *теговые значения* (tag values) и *ограничения над объектами* (object constrains). Стереотипы задают расширенный набор атрибутов для объекта или связи, например, в диаграмме классов UML стереотипы используются для задания разновидности класса. Если классу назначен стандартный стереотип <<interface>>, тогда класс превращается в определение интерфейса. По умолчанию классам назначен стереотип <<class>>. Те же идею используются, чтобы описать связи между вариантами использования в UML-диаграмме вариантов использования. Один вариант использования может "включать" (<<include>>) или "уточнять" (<<extend>>) другой. Новые стереотипы могут задавать разработчики, например, при разработке ИС удобно использовать ORM[[4]](#footnote-28), при этом тот факт, что конкретный класс будет отображаться в запись реляционной базы данных можно при помощи назначенного ему стереотипа <<RDBMSRecord>>. В данном случае, класс добавляется в перечень классов, определяющих структуру базы данных, все отношения между такими классами трактуются как реляционные (один-к-одному, один-к-многим, многие-к-многим, обязательные и нет) отношения. Каждой структуре можно назначить множество стереотипов.

Теговые значения позволяют связать некоторое строковое значение с ключом (строкой), которые так или иначе интерпретируются с процедурой трансформации. Например, самый простой способ назначения полю класса age имени метки в интерфейсе пользователя – это в диаграмме классов назначить этому полю теговое значение interface-label-name равное строке “Возраст”. При использовании RDF имена тегов можно задавать в форме dc:title, а значение тега интерпретировать и как литерал и как URI, например, на объект DBPedia.

Опыт использования стереотипов и теговых значений показал, что назначение определенного стереотипа, как правило, предполагает определение набора одного и того же перечня теговых значений. Стандарт UML версии 2.4 этот момент учтен, и современные средства визуального моделирования позволяют формально определять данный перечень для собственных стереотипов. [[Разработаны специальные интерфейсы пользователя для управления перечнем стереотипов.]] В нашем примере назначение стереотипа <<RDBMSRecord>> классу не позволяет явно задавать соответствующую таблицу, для этого можно использовать теговое значение rdbms:table-name. Таким образом, явная спецификация стереотипов в UML-2.4 – это мощный инструмент расширения семантики элементов визуальных моделей.

## Другие источники модельных данных

При реализации сложных программных систем, их иногда составляют из действующих модулей, спецификации этих модулей могут выступать дополнительными источниками модельных данных. Например, в нашем проекте , основная модель PIM извлекается из исходного кода библиотеки Mothur и преобразуются в ТТЛ-формат. Этот подход требует один раз запрограммировать такой конвертер, и при выпуске новой версии библиотеки мы автоматом получаем новую версию PIM.

# Исследования–аналоги

Наиболее широко используемая технология трансформации моделей – это ATL (ATLAS Transformation Language) и его предшественник QVT, эти технологии – стандарт OMG . Язык и его интерпретатор поддерживают преобразование из одной модели, сохраненной в формате XMI, в другую в том же формате XMI. Структуры языка описывают распознавание свойств композиций элементов исходной модели и построение соответствующих структур в целевой модели.

Аналогично ATL, Язык представления модели трансформации (Transformation Model Representation Language, TMRL) описанный в , но ориентированный на визуальное представление базы знаний с последующим преобразованием в продукционные правила CLIPS и OWL. Визуальное представление правил трансформации используется также в . В , ATL используется для преобразования CIM, педставленной в виде диаграммы BPMN на набор UML-диаграмм, т.е. PIM. В статье предложена модель трансформации CIM в PIM для веб-приложений, CIM представлен диаграммами UML, диаграммой состояний и диаграммой вариантов использования. Логические связи между диаграммами реализуются в ATL. В предложен способ оценки аспектов безопасности распределенных приложений при помощи моделей MDA. Оценка реализуется при помощи логического вывода над данными, полученными в результате трансформации MDA. В статье рассматривается сильно несвязных распределенных вычислительных сред, управляемых событиями. Состав среды и ее функционирование динамически управляется трехуровневым описанием в DSL[[5]](#footnote-31) и соответствующей трансформацией MDA, поддерживающей распространение изменений (change propagation).

Таким образом, большинство представленных методов является замкнутым относительно файла формат XMI: исходные данные и результат трансформации представлены, в основном, в этом формате, выход за границу формата происходит только на этапе генерации кода. Наш подход позволяет использовать другие (не только в XMI) модели, а также библиотеками общего назначения в реализации модуля преобразования. Использование объектно-ориентированного логического языка LogTalk позволяет нам активно использовать структурирование и управление базой знаний.

# Методика реализации трансформации

Основная идея реализации механизма преобразования является использование объектного логического языка программирования, который соответствует следующим ограничениям:

* поддержка нескольких источников данных модели,
* язык первого порядка для определения трансформации,
* трансформация представляется в виде правил,
* правила должны инкапсулироваться в объекты,
* объекты должны запрашивать другие объекты и получать результат,
* поддерживать средства конфигурирования сценария трансформации,
* преобразование структуры знаний, в том числе композиция,
* оперировать тройками RDF.

SWI-Prolog с макропакетом LogTalk соответствует этим ограничениям. Объекты LogTalk инкапсулируют правила языка в виде методов. Каждый объект представляет собой фасад к инкапсулированным знаниям. Объекты наследуются (как прототипы или классы), параметризуются и комплексируются с *категориями*. При наследовании методы можно перепрограммировать наследуемый набор знаний на определенную специфику. Конфигурирование реализована в виде иерархии прототипирования: при наследовании, конфигурация уточняются при помощи замены существующих и добавления новых значений. Сценарии реализуются в виде упорядоченного набора методов. Наконец, для SWI-Prolog существует реализация LogTalk, совместимая с библиотекой RDF, которая привязана к синтаксису языка. Библиотека представляет сущности RDF в виде, рекомендованном W3C.

## Библиотека СВ SWI-Prolog-а

SWI-Prolog в стандартном дистрибутиве включает библиотеку для обработки RDF. Библиотека представляет URI при помощи функтора с аргументом – атомом. URI также задается при помощи специальной формы <namespace>:<identifier>. Библиотека содержит предикаты для загрузки графов из файлов и интернет-сайтов, хранить их в различных текстовых и двоичных форматах для последующего использования.

Библиотека поддерживает запросы к графам в виде комбинации предикатов или на языке SPARQL. Реализована базовая оптимизация запросов. Для отношений subclass и subpropery библиотека реализует транзитивное замыкание, которое использовано в предикативных запросах. Библиотеки импортируется в LogTalk как объекты, то есть как инкапсуляции всех библиотечных предикатов.

## Методы LogTalk

Объекты LogTalk состоят только из инкапсулированных предикатов (методов), некоторые из них могут быть определены как динамические, что позволяет задавать состояния объектов. LogTalk как макропакет позволяет создавать два вида объектов – статические и динамические. Статические объекты создаются при помощи отношения создания экземпляра во время компиляции, динамические – создаются предикатом create\_object/4 во время исполнения программы. Статические параметризованные объекты можно рассматривать как третий способ создания экземпляров. Роль объекта полностью определяется системой отношений с другими объектами, например, экземпляры классов – это объекты, состоящие в отношении instantiates/1 со своим классом.

Статические экземпляры используются для определения конфигураций, сценариев, интерфейсов к входной и выходной информации, базам данных онтологий, и другим глобальным сущностям, известным во время компиляции. Их основная задача – структурировать множества правил Prolog. Динамические объекты используются в формировании объектов PSM, и на данном этапе каждый экземпляр является элементом синтезируемого программного кода. Параметризованные объекты выполняют роль интерфейсов–фасадов к другим объектам и данным графов. Такие объекты дают инструмент создания правил распознавания структур в некотором окружении (контексте), задаваемыми параметрами объекта. Параметризованные объекты должны быть только статическими объектами, в этом случае, LogTalk компилирует их в статически код.

## Представление PSM

PSM и порождение исходного кода реализуется объектами, поддерживающими (оснащающими) интерфейс code\_block. Идея взята из реализации блока кода в библиотеке llvmlite. Итерфейс блока кода имеет следующий вид:

Объекты code\_block внутри содержат строки исходного кода, структуры, из которых порождаются куски исходного кода, объекты LogTalk и другие блоки кода. Каждый элемент содержимого доступен при помощи метода item/1. Тип элемента определяется его внешним функтором, например, сообщение append(attributes(L)) добавляет список L атрибутов в класс. Генерация исходного кода реализуется при наследовании, при этом надо добавить правила, соответствующие новым структурам: render/1 и renderitem/2. Первый аргумент renderitem/2 – это структура из item/1, а второй – список строк, представляющий сгенерированный исходный код. Элементы блока кода можно добавлять в конец (append) и в начало (prepend) списка, а также удалять (remove). Вставка специально запрещена, так как это усложняет реализацию. Вместо этого можно вставлять вспомогательные блоки кода как item-элементы основного блока, добавляя новое содержимое во вспомогательные блоки.

Структура элемента блока не регламентируется, поэтому к нему можно добавлять дополнительные поля, информации которых может быть использована для хранения промежуточных данных или данных для других этапов трансформации.

## Определение правил преобразования

Процесс трансформации организован в виде сценария, составленного из объектов, посылающих сообщения друг другу. Также сценарии можно реализовывать как упорядоченные наборы правил (методов) вида tr/N. Правила распознают композиции элементов входного графа и конструируют блоки кода.

Предыдущий листинг использует статический параметризованный объект query/1, с контекст-аргументом – графом, представляющим трансформируемую модельно. Это мощный инструмент абстрагирования LogTalk, который отменяет необходимость создавать объекты-адаптеры протокола в динамической памяти. Запросы SPARQL инкапсулируются в методах.

Следующий пример показывает как происходит порождение класса языка Python. Объект содержит публичные (public) методы для определения элементов класса.

Сделаем несколько пояснений. Метод класса root::iswritef/3 используется для порождения исходного кода с необходимыми отступами, регулируемыми методами root::indent и root::unindent. Список списков Result, аргумент метода render/1, специально не разравнивается, что позволяет не использовать операцию append. Объект импортирует категорию named, которая определяет поведение поименованных языковых структур (переменных, типов, классов и т.д.).

*Категории* языка LogTalk являются удобным инструментом реализации однотипного поведения среди несвязанных одной иерархией классов. Например, объекты языка программирования поыменовываются идентификаторами и, часто, принадлежат к некоторому типу данных. Эти свойства реализуются в категориях named, и ее производной namedtyped.

Представленная категория требует, чтобы контекстный объект реализовывал интерфейс code\_object. Текст другой категории и тексты остальных объектов трансформации хранятся на сервере github.com .

# Применение в NGS

Предложенная методика применяется при разработке программного обеспечения для проектирования визуальных сценариев вычислительных процедур в NGS[[6]](#footnote-38), таких как загрузки данных секвенирования, фильтрации, выравнивания гена, оценки качества результата . Процедура представлена в виде диаграммы потоков данных (dataflow diagram), они визуально конструируются и исполняются в системе Rapidminer. Каждый блок диаграммы определяет операцию, которая преобразует набор входных файлов в набор выходных. В Mothur находится 147 модулей. Файлы передаются между модулями. В диаграмме это отражается в виде связей между блоками. Визуальный инструмент предназначен для биологов, позволяя им самостоятельно проводить биоинформатические исследования.

Структура каждого блока, а именно, входные и выходные соединения, а также внутренние параметры, отражают интерфейсы и структуры модулей Mothur. Модули Mothur реализованы в виде классов на C++. При помощи дополнительных структур каждый модуль дополнительно описывается, и эти описания доступны во время выполнения скрипта. Анализ этой дополнительной информации дает данные о структуре межмодульного взаимодействия, которые активно используются при трансформации.

Анализ исходного кода выполняется при помощи регулярных выражений и простого синтаксического анализа. Мы пытались использовать специальную версию компилятора GCC<Р0>, который представляет абстрактное синтаксическое дерево в виде XML-файла. Но, к сожалению, структура этого файла является слишком сложной для анализа и содержит лишь декларативные части источников (типы, классы с полями и объявлениями методов, внешние переменные).

Трансформация порождает модули Java, реализующие структуру и функции блоков. По логике некоторых модулей Mothur заложена гибкая функциональность: набор выходных файлов зависит от набора входных и настроек модуля. В идеальном случае в dataflow-блоках необходимо учитывать такие нюансы и предоставлять пользователю комбинации входов и выходов отражающие эти особенности. Сейчас мы работаем над расширением базы знаний для генерации алгоритмов, реализующих вычисление наборов допустимых комбинаций соединений в зависимости от комбинации входных и конфигурации модуля.

Общее время, затраченное на доработку предложенной методики трансформации на основные (структурные) аспекты Mothur и реализацию модуля трансформации – два человека-месяца. Реализованная внутренняя логика позволила идентифицировать ошибки в исходном коде описаний модулей Mothur: процедура трансформации PIM в PSM для "бракованных" описаний попросту не выполнялась. Поэтому модули преобразования могут быть использованы в качестве верификации исходного кода Mothur.

Основным режимом функционирования полученной среды визуального программирования будет взаимодействие с сервером Больших данных, где хранятся результаты секвенирования собираемых проб. После реализации данной части проекта необходимо будет разработать интерфейсы и конвертеры данных к серверу, а также системы порождения отчетов исследований, которые мы предполагаем размечать согласно методике LOD[[7]](#footnote-39). Разметка LOD основана на RDF, поэтому данные о структуре моделей можно использовать в генераторах отчетов для разметки выводимых значений.

# Заключение

В докладе представлен подход к реализации процедуры трансформации, поддерживающей подход Model Driven Architecture (MDA), который опосредует большинство удобных свойств логического и объектно-ориентированного программирования, представление данных исходной модели данных в виде графов RDF. Использование языка LogTalk позволило структурировать трансформационные знания, создать инструментальный базис для внесения модификаций в наборы знаний при помощи наследования и композиции. Инкапсуляция дает возможность скрывать специальные знаний за интерфейсом объекта.

Инструменты разработки использованы в создании инструмента представления биоинформатических модулей Mothur в виде dataflow-диаграмм системы Rapidminer <п0>. Исходные данные для преобразования – это спецификации модуля и дополнительные диаграмма классов UML, которая организует модули в виртуальную иерархию блоков. В результате был синтезирован модуль Rapidminer, в котором все модули Mothur представлены в виде блоков.

Разработанная методика проектирования инструментов MDA будет развиваться в направлении формирования приемов программирования, направленных на повторное использование знаний и их специализации, таким образом, поддерживая традиции программирования, используемые при проектировании небольших информационных систем для развивающихся компаний. Другой аспект исследования – это интеграция с существующими приложениями открытых связанных данных, таких как .

# Благодарности

Представленные результаты исследований распределена среди трех грантов следующим образом: LOD– и RDF–представления данных и их интеграция поддержана Российским научным фондом, Грант 18–07–0075; логический вывод в процедуре трансформации над комплексами моделей и объектно-ориентированное представление знаний – поддержка Российского фонда фундаментальных исследований, Грант 18–71–10001; представленный пример Rapidminer разрабатывается при поддержке гранта Иркутского научного центра СО РАН, проект № 4.2.

99

J. Wielemaker, W. Beek, M. Hildebrand, J. Ossenbruggen, “ClioPatria: a SWI-Prolog infrastructure for the Semantic Web,” Semantic Web. vol. 7, no. 5, 2016, pp. 529-541. “Modelio open source – UML and BPMN free modeling tool.” URL:<https://www.modelio.org/>. A. Belghiat, M. Bourahla, “UML Class Diagrams to OWL ontologies: a graph transformation based approach,” International Journal of Computer Applications. no. 41. pp. 41–46. Y. Rhazali, Y. Hadi, A. Mouloudi. “Model transformation with ATL into MDA from CIM to PIM structured through MVC,”Procedia Computer Science 83 (2016) 1096-–1101. URL:<https://doi.org/10.1016/j.procs.2016.04.229> Y. Rhazali, Y. Hadi, I. Chana, M. Lahmer, A. Rhattoy, “A model transformation in model driven architecture from business model to web model,” IAENG International Journal of Computer Science, vol. 45(1), 2018, 104-117, URL:<http://www.iaeng.org/IJCS/issues_v45/issue_1/IJCS_45_1_16.pdf>. B. Hamid, D. Weber, “Engineering secure systems: models, patterns and empirical validation,” Computers & Security, vol. 77, 2018, p. 315-348. S. Tragatschnig, S. Stevanetic, U. Zdun, “[Supporting the evolution of event-driven service-oriented architectures using change patterns.](https://www.sciencedirect.com/science/article/abs/pii/S0950584916303251)” Information and Software Technology. vol. 100, 2018, p. 133-146.

E. Cherkashin, A. Shigarov, F. Malkov, A. Morozov, “An instrumental environment for metagenomic analysis,” In: Bychkov I., Voronin V. (eds) Information Technologies in the Research of Biodiversity. Springer Proceedings in Earth and Environmental Sciences. Springer, Cham, 2019, pp. 151–158.

F. Jouault, F. Allilaire, J. Bezivin, I. Kurtev, “ATL: a model transformation tool,” Sci. Comput. Program. vol. 72(1–2), pp. 31–39 (2008) “The MOF query/view/transformation specification version 1.1.” URL:<http://www.omg.org/spec/QVT/1.1> A. Berman, M. Grishchenko, N. Dorodnykh, O. Nikolaychuk, A. Yurin. “A model-driven approach and a tool to support creation of rule-based expert systems for industrial safety expertise,” Proc. of the 12-th International Forum on Knowledge Asset Dynamics (IFKAD-2017) – Russia, St. Petersburg : Graduate School of 16 Management of St. Petersburg University. 2017. P. 2034–2050. E. Cherkashin, “GitHub project page of ICC.XMITransform project.” URL:<https://github.com/isu-enterprise/icc.xmitransform> (access date: 01-oct-2019) E. Cherkashin, A. Shigarov, V. Paramonov, A. Mikhailov, “Digital archives supporting document content inference,” Procs of 42-nd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), 20–24 May, 2019, Opatija, Croatia. pp. 1037–1042.

1. XML Metadata Interchange [↑](#footnote-ref-21)
2. ATLAS Transformation Language [↑](#footnote-ref-22)
3. Resource Description Framework [↑](#footnote-ref-23)
4. Объектно-реляционное отображение [↑](#footnote-ref-28)
5. Язык описания предметной области [↑](#footnote-ref-31)
6. New Generation Sequencing [↑](#footnote-ref-38)
7. Linked Open Data [↑](#footnote-ref-39)