**Знания-Онтологии-Теории (ЗОНТ-19)**

# Представление Трансформаций MDA в Виде Логических Объектов

Евгений Черкашин[1], Алексей Шигаров[1,2], Вячеслав Парамонов[1]

[1]*Институт динамики систем и теории управления СО РАН, Лермонтова 134, Иркутск, Россия,*
[2]*Иркутский научный центр СО РАН, Лермонтова 134, Иркутск, Россия*

{eugeneai,shig,slv}@icc.ru

**Аннотация.** *Рассматривается задача моделирования программного обеспечения, которая представлена набором моделей, а трансформация – в виде 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, BPMN, графов RDF и результатами анализа исходного кода других систем, если реализовать соответствующий модуль преобразования. Объекты представлены в языке 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.*

**Ключевые слова: модельно-управляемая архитектура, логический вывод, RDF, LogTalk**

## 1 Введение

Основная проблема при разработке сложных информационных систем (ИС), а также других систем автоматизации производства, – это сложность, которая выражается в используемых структурах данных, интегрированием с другими системами, 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 формализованная модель 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 формируются при помощи соответствующих визуальных инструментов. Программирование трансформаций необходимо сделать близким к обычному традиционному программированию, позволяющему, в частности, создавать из существующих инструментов 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 (структуры данных, автоматы). 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. 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, основаны на преобразовании одной XMI-модели в другую XMI, а затем уже в программный код. Привлечение данных СВ достаточно проблематично.

Мы предлагаем более общий подход, где для всех исходных типов моделей разрабатываются конвертеры в формат RDF, в стандартное представление данных и знаний в СВ, а трансформация программируется в виде объектно-ориентированной логической на логическом языке LogTalk [1]. 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 исследования в данной статье является описание предложенного выше представления данных и методов преобразования моделей.

## 2 Архитектура инструментов MDA

Архитектура средства трансформации основывается на обеспечения единообразия языка правил трансформации и представления входных данных в виде атомарных высказываний RDF. Поэтому мы требуем, 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 1.

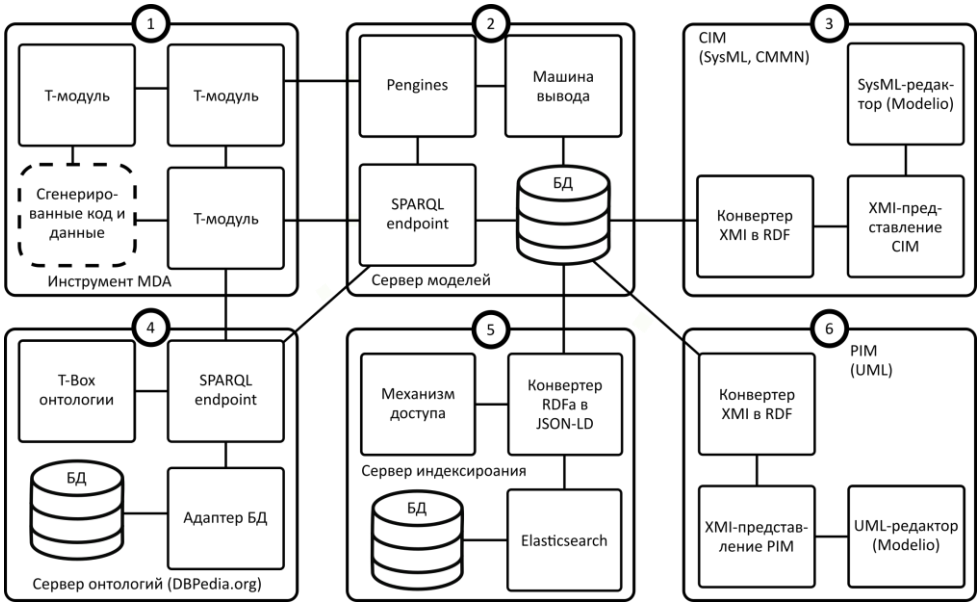


**Рис.1.** Архитектура средств трансформации MDA.

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.

Подсистема MDA представляет собой набор модулей трансформации (T–модули). Каждый T–модуль в общем случае является параметризованным объектом LogTalk. Объект запрашивает у сервера модели предметной области. Сервер построен на базе системы ClioPatria [2]. Объект может запрашивать также другие объекты и другие сервера онтологий. В полученные данные о структуре моделей анализируются и распознаются заданные структурные композиции. Результаты кэшируются в объектах или индексируются 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 1 CIM (3) сконструирована в редакторе Modelio [3], и, затем, преобразована в тройки в модуле “Конвертер XMI в RDF”. Аналогичный конвертер разработан и для PIM, представляемой диаграммами UML. Такой подход позволяет расширять инструмент новыми видами моделями.

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.

# 3 Представление данных моделей в RDF

Последние 20 лет разработки технологий СВ привели к созданию формальных описаний 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 форматы текстового 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 обогащает 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.

## 3.1 Определение сущностей

Идентификация сущностей и их поименование в RDF осуществляется при помощи отношений. 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.

## 3.2 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 проектироании ИС удобно использовать ORM[1], 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

[1] Объектно-реляционное отображение

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 дополнительное 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. В визуальных редакторах UML 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.

### 3.3 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 TTЛ-формат. Этот подход требует один раз запрограммировать такой конвертер, и при выпуске новой версии библиотеки автоматически получается новая версия PIM.

## 4 Методика реализации трансформации

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

- поддержка нескольких источников данных модели,
- трансформация представляется в виде правил первого порядка,
- наборы правил должны инкапсулироваться в объекты,
- объекты должны запрашивать другие объекты и получать ответ,
- поддерживать средства конфигурирования сценария трансформации,
- преобразование структуры знаний (наборов правил), в том числе их 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 реализовано 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 библиотеки RDF, которая привязана к синтаксису языка и совместима с LogTalk. Библиотека представляет сущности RDF в виде, рекомендованном W3C.

## 4.1 Библиотека SWI-Prolog для реализации технологий Семантического Веба

SWI-Prolog в стандартном дистрибутиве включает библиотеку для обработки RDF. Библиотека представляет URI при помощи функтора с аргументом–атомом. URI также может задается при помощи специальной формы <namespace>:<identifier>, что очень удобно при программировании трансформации. 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 Prolog-запросов или на языке SPARQL. Реализована базовая оптимизация исполнения запроссов. Для отношений subclass и subpropery библиотека реализует транзитивное замыкание на системном уровне. Библиотеки импортируется в LogTalk как объекты, то есть как инкапсуляции всех библиотечных предикатов.

## 4.2 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.

## 4.3 Представление PSM

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

```
:- protocol(code_block_proto).
   :- public([append/1, prepend/1, clear/0, remove/1, item/1, items/1]).
   :- protected([ renderitem/2, render_to/2]).
:- end_protocol.
```

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 в списке элементов держать вспомогательные блоки кода, добавляя в них новое содержимое.

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.

## 4.4 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.

```
:- object(direct(_Package,_LocalProf,_CodeProf)).
   :- public([tr/4,tr/3]).
   :- protected([package/1, profiles/2, profile/1]).
   package(Package):- parameter(1, Package).
   profile(Profile):- parameter(2, Profile).
   profile(Profile):- parameter(3, Profile).
   profiles(L):- findall(Profile, ::profile(Profile), L).
   tr(class, Class, ClassID):- ::package(Package),
       query(Package)::class(Name, ClassID),
       create_object(Class, [instantiates(class)],[],[]),
       create_object(Attributes, [instantiates(params)],[],[]),
       create_object(Methods, [instantiates(methodlist)],[],[]),
       Class::append(name(Name)),
       forall( ::tr(attribute,Attribute,ClassID,_AttrID),
           Attributes::append(Attribute) ),
       forall( ::tr(method, Method, ClassID, _MethodID),
           Methods::append(Method) ),
       Class::append(attributes(Attributes)),
       Class::append(methods(Methods)).
   tr(attribute, Attribute, ClassID, AttributeID):- ::package(Package),
   query(Package)::attribute(Name,ClassID,AttrID),
       create_object(Attribute, [instantiates(param)],[],[]),
       Attribute::append(name(Name)).
   tr(method, Method, ClassID, MethodID):- ::package(Package),
       query(Package)::method(Name,ClassID,MethodID),
       create_object(Method, [instantiates(method)],[],[]),
       Method::append(name(Name)).
   :- end_object.
```

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

```
:- object(query(_XMI)).
   :- protected(xmi/1).
```

```
  :- public([class/2, attribute/3, method/3]).
  xmi(XMI) :- parameter(1, XMI).
  class(Name, ID):- ::xmi(XMI),
      XMI::rdf(ID,rdf:type,uml,'Class'),
      XMI::rdf(ID,rdfs:label, literal(Name)).
  attribute(Name, ClassID, ID):- ::xmi(XMI),
      XMI::graph(G), XMI::rdf(ClassID, G:ownedAttribute, ID),
      XMI::rdf(ID, rdfs:label, literal(Name)).
  method(Name, ClassID, ID):- ::xmi(XMI), XMI::graph(G),
      XMI::rdf(ClassID, G:ownedOperation, ID),
      XMI::rdf(ID, rdfs:label, literal(Name)).
:- end_object.
```

Следующий пример показывает, как происходит порождение класса языка Python.

```
:- object(class, specializes(code_block), imports([named])).
  :- public([classlist/1, methods/1, attributes/1]).
  renderitem(Item, Result):- ^^renderitem(Item, Result).
  render(Result):- ^^render(Name),
      ( ::item(classlist(List)) -> List::render(ClassList),
        root::iswritef(Signature,'class %w(%w):', [Name, ClassList]);
        root::iswritef(Signature,'class %w:', [Name]) ),
      root::indent, % Add an indent
      ( ::item(attributes(Attributes))-> Attributes::render(DefAttrList),
        root::iswritef(ConstructorDef, 'def __init__(self, %w):',
        [DefAttrList]),
        root::indent, % more indent
        Attributes::items(InstanceAttrs), findall(S, ( % initialize attributes
           lists::member(Attr, InstanceAttrs),
        Attr::item(name(AttrName)), root::iswritef(S, "self.%w=%w",
        [AttrName, AttrName]) ), AttrAssigns),
        root::unindent,
        AttrList=[ConstructorDef|AttrAssigns];
        root::iswritef(ConstructorDef, 'def __init__(self): ', []),
        root::indent,
        root::iswritef(Pass,'pass', []),
        root::unindent, AttrList=[ConstructorDef, Pass] ),
      ( ::item(methods(Methods))-> Methods::render(MethodList); MethodList=[] ),
      lists::append(AttrList,MethodList,StringList),
      root::unindent, Result=[Signature|StringList].
  :- end_object.
```

Сделаем несколько пояснений. Метод класса root::iswritef/3 используется для порождения строки 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 лишний раз 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 классов, не связанных одной иерархией. Например, объекты языка программирования поименовываются идентификаторами и, часто, принадлежат к некоторому типу данных. Эти свойства реализуются в категориях named, и ее производной namedtyped.

```
:- category(named).
   :- public([name/1, render/1]).
   :- protected([renderitem/2]).
   name(Name):- ::prepend(name(Name)).
   renderitem(name(Name), String):-!,
       atom_string(Name, String).
   render(String):-
       ::item(name(Name)), ::renderitem(name(Name), String).
:-end_category.
```

Представленная категория требует, чтобы контекстный объект реализовывал интерфейс code_object. Текст другой категории и тексты остальных объектов трансформации хранятся на сервере github.com [4].

# 5 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 [1] 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 [5]. 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 147 модулей Mothur представлены в виде блоков. Одно из направлений развития данного проекта – порождение отчетов о результатах исследований в виде размеченных LOD документов [6].

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, широко используемые при проектировании небольших информационных систем для развивающихся компаний.

# 6 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.

## Литература


[1] P. Moura. Programming Patterns for Logtalk Parametric Objects. Applications of Declarative Programming and Knowledge Management, *Lecture Notes in Artificial Intelligence*, Vol. 6547, April 2011.

[2] 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.

[3] Modelio open source – UML and BPMN free modeling tool. URL:https://www.modelio.org/.

[4] E. Cherkashin. GitHub project page of ICC.XMITransform project. URL:https://github.com/isu-enterprise/icc.xmitransform (access date: 01.10.2019)

[5] 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.

[6] 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.