Методы автоматизированного построения трансформаций информационных моделей

С. А. Ступников, Л.А. Калиниченко

Принимая во внимание многообразие существующих информационных моделей и трудоемкость процесса построения трансформаций таких моделей в интероперабельных системах, представляется актуальной задача автоматизации этого процесса. В статье рассматриваются два метода автоматизированного построения трансформаций. Первый метод предназначен для построения прямых трансформаций на основе установленных соответствий между элементами моделей. Второй метод предназначен для построения обратных трансформаций моделей на основе прямых трансформаций. Методы автоматизированного построения трансформаций моделей являются частью более общей работы – создания Конструктора унифицирующих информационных моделей (Унификатора моделей). Главной прагматической задачей, решаемой при помощи Унификатора, является отображение моделей неоднородных информационных ресурсов в каноническую информационную модель в процессе интеграции ресурсов.

1. Введение

Текущий период развития информационных систем характеризуется существенным ростом количества применяемых информационных моделей. Появляются новые и развиваются существующие модели данных, такие как ODMG 2000, SQL 2003, UML, XML и RDF стеки; модели потоков работ, например, Staffware, Meteor, Mobile, MQSeries, SAP/R3; языки процессной композиции сервисов - XPDL, BPEL4WS, BPML, XLANG, WSFL, WSCI; семантические модели, включающие онтологические модели и модели метаданных. Процесс развития информационных моделей сопровождается накоплением большого количества информационных ресурсов - компонентов и сервисов, определенных средствами различных информационных моделей. Растет потребность в совместном использовании (интеграции) ресурсов в различных приложениях, в том числе, для реализации новых информационных систем [1].

С ростом разнообразия моделей, используемых для представления информационных ресурсов, растет сложность интеграции и композиции ресурсов, обеспечения их интероперабельности. Обычно ресурсы являются неоднородными, представленными в разных моделях. Проблемы интеграции ресурсов при этом требуют унификации используемых моделей в рамках некоторой информационной модели, называемой *канонической*. Для унификации неоднородных информационных моделей необходимы специальные методы и средства, опирающиеся на семантику моделей и позволяющие проводить доказательные рассуждения о свойствах моделей.

Методы, изложенные в данной статье[1], являются частью более общей работы – создания Конструктора унифицирующих информационных моделей (Унификатора моделей, для краткости) [2, 3]. Целью Унификатора является доказательное приведение множества разнотипных моделей информационных ресурсов к каноническому, унифицированному представлению в процессе интеграции информационных ресурсов в предметных посредниках [1]. Процесс *унификации* информационной модели включает построение отображения модели в каноническую и верификацию полученного отображения. Отображение считается корректным, если информационная модель *уточняет* каноническую модель [2].

В рамках работ по созданию прототипа Унификатора исследуются различные технологические средства построения отображений информационных моделей. В [3] для формального описания синтаксиса моделей и трансформаций моделей рассмотрено применение декларативных языков метакомпиляции SDF (Syntax Definition Formalism) и ASF (Algebraic Specification Formalism), обеспеченных средствами инструментальной поддержки [4]. В данной работе приведены результаты исследования возможности использования в Унификаторе другого вида языков трансформации моделей, сочетающих как декларативные, так и императивные средства описания отображений моделей. Такие языки развиваются в контексте *Движимой моделями архитектуры* (Model-Driven Architecture, MDA [5]) – подхода, поддерживаемого стандартом OMG MOF (Meta-Object Facility) [6]. Примером подобного языка является QVT (Query-View-Transformation) [7] – язык трансформации моделей, предлагаемый консорциумом OMG (Object Management Group).

Базовыми составляющими MDA являются модели. Они рассматриваются как первичные сущности, и наиболее важными операциями манипулирования моделями становятся преобразования моделей, отображения моделей из одной в другую.

Модель определяется в соответствии с семантикой некоторой *метамодели*, при этом говорят, что модель *конформна* (conforms to) метамодели. Стандартом MOF определена четырехуровневая архитектура моделей: модели уровня M0 описывают объекты реального мира, модели уровня M1 называются обычно *схемами*, модели уровня M2 представляют собой собственно *информационные модели* (например, в архитектурах систем управления данными представляющие собой совокупность языка определения данных и языка манипулирования данными), модели уровня M3 – метаметамодели, предназначенные для описания моделей уровня M2.

Данная работа имеет своей целью разработку методов автоматизированного построения трансформаций информационных моделей на основе средств трансформации моделей, созданных в рамках MDA. Результатом применения методов являются декларативно-императивные трансформации, реализующие

[1] Работа выполнена при финансовой поддержке РФФИ (проект 08-07-00157-а) и Программы фундаментальных исследований Президиума РАН № 3 *Фундаментальные проблемы системного программирования* (проект *Исследование методов и средств промежуточного слоя предметных посредников, обеспечивающего решение задач над множеством неоднородных распределенных информационных ресурсов*).

отображения произвольных исходных моделей уровня M1, конформных исходной метамодели уровня M2, в целевые модели уровня M1, конформные целевой метамодели уровня M2.

Задача автоматизации процесса построения трансформаций является весьма актуальной, принимая во внимание трудоемкость процесса и многообразие информационных моделей.

В данной работе рассматриваются два аспекта автоматизации построения отображений:

- построение *прямых* трансформаций на основе установленных соответствий между элементами моделей;
- построение *обратных* трансформаций моделей на основе прямых трансформаций.

Соответствия между элементами моделей могут быть установлены различными способами. Например, элементы моделей могут быть аннотированы описаниями на естественном языке. При этом соответствие между элементами устанавливается (автоматическим или полуавтоматическим способом) на основании сходства этих описаний. Соответствие элементов моделей может быть установлено на основании анализа структуры моделей, с применением алгоритмов сопоставления схем (schema matching [8]). Элементы моделей также могут быть аннотированы понятиями некоторой онтологии [9]. При этом соответствие между элементами устанавливается, если понятия, которыми аннотированы элементы, связаны отношением понятие-подпонятие.

В данной работе механизм установления соответствий между понятиями не конкретизируется. Вопросы установления понятийных соответствий в результате частично автоматизируемого процесса обсуждались в работе [2]. Заметим, что информации, заложенной в наборе соответствий моделей, построенном на основе их сходства, в общем случае недостаточно для построения корректной трансформации. Так, ранее было показано, что для построения семантических соответствий уточняющего отображения некоторой информационной модели в каноническую в общем случае требуется определение адекватного расширения ядра канонической модели [10, 11], и были предложены доказательные методы [10 , 12] конструирования таких расширений и семантических соответствий. Здесь рассматривается метод, нацеленный на технику декларативно-императивного программирования отображения моделей (конструирование составляющих его правил) на основании установленных соответствий между элементами моделей в предположении, что проблемы семантики отображений разрешены. Вопрос о том, как соединить предлагаемую здесь технику построения трансформаций с разработанными ранее методами конструирования уточняющих семантических соответствий, требует дополнительных исследований и выходит за рамки данной работы.

Целью второго метода, рассматриваемого в данной работе, является построение обратных трансформаций моделей на основе прямых трансформаций. Рассмотрим две модели - *S* и *T*. Предположим, что создана трансформация *S2T* модели *S* в модель *T* (с применением упомянутого выше метода построения

трансформаций на основе соответствий между элементами моделей или полностью вручную). Такую трансформацию назовем *прямой*, модель *S* – исходной, модель *T* – целевой. Задача состоит в автоматизированном построении трансформации *T2S* модели *T* в модель *S*, называемой *обратной*, на основе спецификации трансформации *S2T*. Метод построения обратных трансформаций, как и метод построения трансформаций на основе соответствий элементов моделей, позволяет лишь частично автоматизировать процесс построения трансформаций. Конструкции языков трансформации моделей различаются по сложности и семантике: для одних возможно автоматически построить соответствующую конструкцию обратной трансформации, для других разработать такую конструкцию должен эксперт на основании семантики моделей.

Методы построения обратных трансформаций могут быть использованы при интеграции информационных ресурсов в предметных посредниках в различных целях, например:

- построение реверсивных отображений онтологических моделей для регистрации ресурсов в посредниках [13];
- доказательство эквивалентности информационных моделей (модель *S эквивалентна* модели *T,* если *S* уточняет *T* и *T* уточняет *S*; для доказательства двухстороннего уточнения моделей необходимы прямое и обратное отображения);
- построение отображения канонической модели посредника M_1 в каноническую модель посредника M_2 для регистрации M_2 как ресурса в M_1;
- построение отображения канонической модели посредника в некоторый прагматический язык, например UML.

В качестве языка трансформации моделей в данной работе рассматривается язык ATL (ATLAS Transformation Language) [14], разработанный научной группой ATLAS INRIA & LINA. ATL является ответом на RFP (Request for Proposal) языка QVT. Система типов и операций над типами языка ATL очень близка (но не тождественна) системе типов языка OCL [15]. Для языка ATL на базе платформы Eclipse реализована интегрированная среда разработки, называемая ATL Development Tools (ADT). В качестве модели уровня M3 рассматривается модель *Ecore* [16].

Представленные методы могут быть адаптированы для построения трансформаций, выраженных на сходных с ATL языках (например, QVT или его диалектах). Конкретная модель уровня M3 также не является существенной, могут быть использованы и другие метаметамодели, например MOF [6] или KM3 [17]. Методы также могут быть использованы независимо от Унификатора для неверифицируемого построения прямых и обратных трансформаций информационных моделей в контексте MDA.

Метод построения трансформаций на основе установленных соответствий между элементами моделей рассматривается в разделе 2. Метод построения обратных трансформаций моделей на основе прямых трансформаций рассматривается в разделе 3.

2. Построение отображений моделей на основе соответствий между элементами моделей

В качестве примера исходной модели в данной статье рассматривается OWL (Web Ontology Language [18]) – язык семантической разметки для публикации и совместного использования онтологий в Веб. В качестве целевой (канонической) информационной модели рассматривается язык СИНТЕЗ [19], ориентированный на семантическую интероперабельность и композиционное проектирование информационных систем в широком диапазоне существующих неоднородных информационных компонентов. OWL и СИНТЕЗ (а точнее, формальный синтаксис этих языков) были представлены в виде моделей уровня M2, конформных метамодели *Ecore*. В дальнейшем эти M2-модели обозначаются *OWL* и *Synthesis* соответственно. В данной статье рассматриваются только небольшие подмножества языков.

2.1 Соответствия элементов информационных моделей

Рассмотрим на примере, как могут выглядеть исходные данные для автоматизированного построения отображения, т.е. подмножество соответствий элементов моделей.

На рис. 1 изображены некоторые из элементов моделей (для лучшей читаемости они представлены в виде диаграммы классов UML).

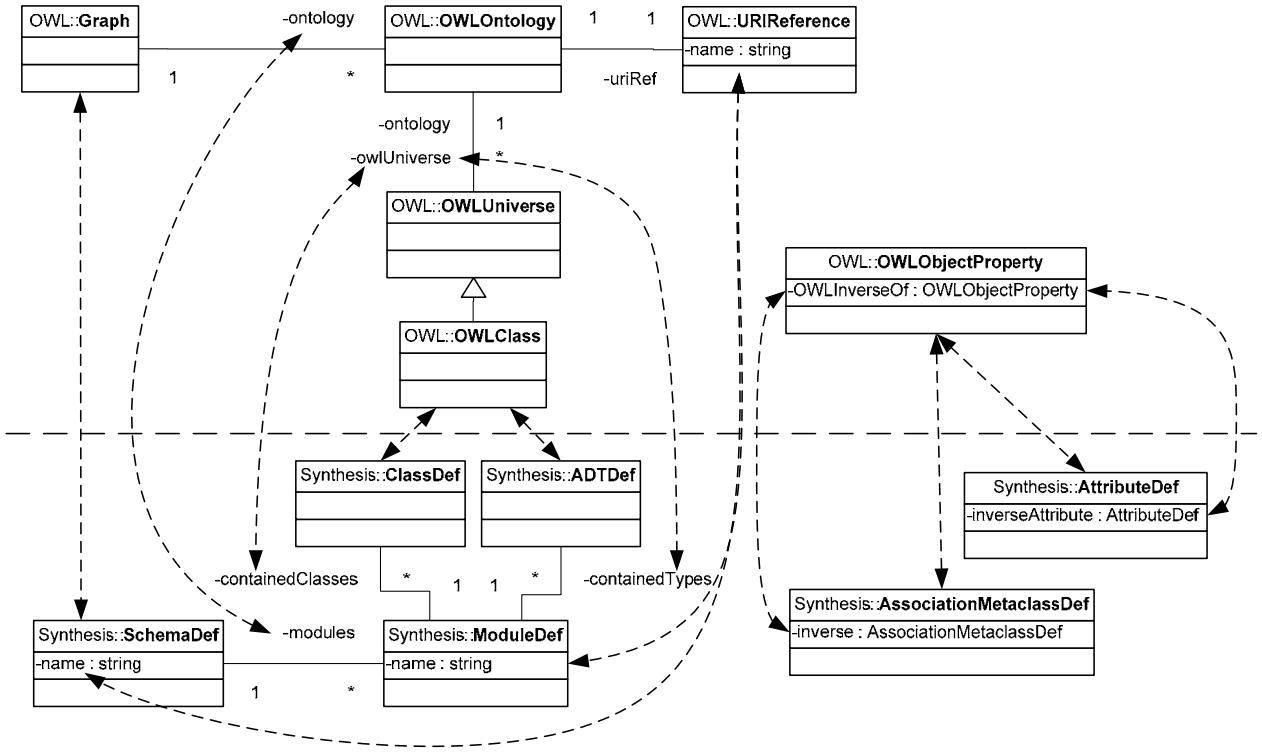

Рисунок 1. Соответствия между элементами моделей *OWL* и *Synthesis*

В верхней части диаграммы изображены элементы *OWL*, в нижней части – элементы *Synthesis*.

Элементами моделей, участвующих в соответствиях, могут быть классы (например, *OWLOntology* – элемент типа *Eclass* метамодели *Ecore*), атрибуты (например, *URIReference.name* – элемент типа *EAttribute* в *Ecore*), связи между классами (например, *Graph.ontology* – элемент типа *EReference* в *Ecore*). Связи могут быть различной кардинальности: один-к-одному или один-ко-многим.

Основным элементом спецификации OWL является граф (*Graph*), состоящий из онтологий (*OWLOntology*). Основным элементом спецификации языка СИНТЕЗ является схема (*SchemaDef*), состоящая из модулей (*ModuleDef*). Модули содержат классы (*containedClasses*) и типы (*containedTypes*). Структуру типов составляют атрибуты (*AttributeDef*). Для описания специализированных свойств атрибутов, их категоризации, используются метаклассы ассоциаций (*AssociationMetaclassDef*). Атрибут может являться экземпляром некоторых метаклассов ассоциаций [19].

Соответствия между элементами моделей обозначены на рисунке пунктирными линиями, в тексте соответствие обозначается знаком ~. Соответствие элементов моделей может быть установлено между классами (*Graph ~ SchemaDef*; граф соответствует схеме), между связями (*OWLOntology.ontology ~ SchemaDef.modules*), между атрибутами (*OWLOntology.uriRef.name ~ ModuleDef.name*). Элементу одной модели может соответствовать несколько элементов другой модели. Если обозначить множество элементов исходной модели (*OWL*) через E_S, множество элементов целевой модели (СИНТЕЗ) – через E_T, то множество соответствий R между элементами моделей есть подмножество декартова произведения $E_S \times E_T$ (т.е. отношение на этих множествах).

Список соответствий элементов моделей, изображенных на рис. 1, приведен в таблице 1.

Элемент модели OWL	*Элемент модели Synthesis*
Graph	SchemaDef
OWLOntology	ModuleDef
OWLClass	ADTDef, ClassDef
OWLObjectProperty	AttributeDef, AssociationMetaclassDef
Graph.ontology->uriRef.name	SchemaDef.name
Graph.ontology	SchemaDef.modules
OWLOntology.uriRef.name	ModuleDef.name
OWLOntology.owlUniverse	ModuleDef.containedTypes, ModuleDef.containedClasses
OWLObjectProperty.OWLInverseOf	AttributeDef.inverseAttribute, AssociationMetaclass.inverse

Таблица 1. Список соответствий элементов моделей *OWL* и *Synthesis*

В выражениях пути, задающих элементы моделей, точка означает навигацию по атрибуту или связи один-к-одному, стрелка (->) –навигацию по связи один-ко-многим.

2.2 Метод построения трансформаций информационных моделей

Целью предлагаемого метода является автоматическое построение как можно более полной трансформации исходной модели в целевую на основе множества *R* соответствий между элементами моделей.

Технически трансформация представляет собой *модуль* языка ATL. Модуль состоит из заголовка (включающего имя модуля и имена переменных, соответствующих исходной и целевой моделям) и множества правил, определяющих способ построения элементов целевой модели на основе элементов исходной модели. Так называемые *сопоставляющие правила* (matched rules), составляющие ядро трансформации, позволяют описывать:

- какой элемент исходной модели должен быть взят;
- количество и тип порождаемых элементов целевой модели;
- способ, при помощи которого элементы целевой модели инициализируются на основании элементов исходной модели.

Так, в правиле *OWLOntology2ModuleDef*

```
rule OWLOntology2ModuleDef{
   from o: OWL!OWLOntology
   to m: Synthesis!ModuleDef(
         name <- o.uriRef.name
   )
}
```

входным элементом является элемент типа *OWLOntology,* порождается один элемент типа *ModuleDef*, и имя модуля формируется из имени онтологии, указанного в ее уникальном идентификаторе ресурса (*uriRef*).

Основным содержанием метода является набор *порождающих правил*, обеспечивающих автоматическое построение ATL-модуля трансформации заданной исходной модели *S* в заданную целевую модель *T,* включающих*:*

- порождающее правило для построения заголовка отображения;
- порождающие правила для построения сигнатуры сопоставляющих правил ATL, составляющих отображение (сигнатура включает имя, исходный и целевые элементы правила);
- порождающие правила для построения инициализации целевых элементов в правилах отображения.

Одному элементу исходной модели может соответствовать несколько элементов целевой модели и наоборот. В разных случаях требуются разные порождающие правила построения сигнатур и инициализаций.

Рассмотрим правила построения трансформаций информационных моделей на основании соответствий элементов моделей (на примере отображения OWL в язык СИНТЕЗ).

2.2.1 Построение заголовка трансформации

Трансформация OWL в язык СИНТЕЗ представляется модулем языка ATL и называется *OWL2Synthesis*. Заголовок трансформации выглядит следующим образом:

```
module OWL2Synthesis;
create OUT: Synthesis from IN: OWL;
```

Заголовок говорит о том, что трансформация получает на вход модель, конформную модели *OWL*, и порождает модель, конформную модели *Synthesis*.

2.2.2 Построение сигнатур правил трансформации

Для каждого элемента *A* типа *EClass* из области определения *domain(R)* $\subseteq E_S$ отношения соответствия элементов *R* создается правило установления соответствия трансформации. Если элементу *A* в E_F соответствует ровно один элемент *B*, то правило называется *A2B:*

```
rule A2B{
   from a: OWL!A
   to b: Synthesis!B
}
```

Если элементу *A* соответствуют несколько элементов $B_1, \ldots, B_n$, то правило называется *A:*

```
rule A{
   from a: OWL!A
   to b1: Synthesis!B1,
         b2: Fund!B2,
         …
         bn: Fund!Bn
}
```

Примером является правило *OWLClass*:

```
rule OWLClass{
   from c: OWL!OWLClass
   to
         type: Synthesis!ADTDef,
         class: Synthesis!ClassDef
}
```

Правило говорит о том, что на основании элемента типа *OWLClass* будут созданы два элемента - типов *ADTDef* и *ClassDef* соответственно.

2.2.3 Инициализация свойств целевого элемента в правилах

Для каждого построенного правила *A* (*A2B*), для каждого элемента B_i (соответствующего *A*), для каждого свойства (атрибута или связи) *f* элемента B_i

производится поиск соответствующего свойства элемента *A*. Затем, на основании найденного соответствия, в правило добавляется инициализация свойства *f*.

Порождающее правило для инициализации свойств примитивных типов. В случае, если свойство имеет примитивный тип (Boolean, Integer, String и т.д.), инициализация свойства производится в разделе *to* правила (называемом целевым образцом – *target pattern* – правила).

Примером такого свойства является *SchemaDef.name,* участвующее в соответствии *Graph.ontology->uriRef.name ~ SchemaDef.name.* Инициализация этого свойства в правиле *Graph2SchemaDef* выглядит следующим образом:

```
rule Graph2SchemaDef{
   from g: OWL!Graph
   to s: Synthesis!SchemaDef(
         name <- g.ontology->any().uriRef.name
   )
}
```

Свойство *name* элемента *SchemaDef* инициализируется выражением, построенным на основании пути в структуре элемента *Graph*, указанном в соответствии. Точечная нотация в структуре пути остается без изменений, для навигации по связи один-ко-многим необходимо применить операцию *any* выбора произвольного экземпляра из коллекции. В данном случае имя схемы инициализируется одним из имен онтологий графа.

Часто подобная инициализация не является окончательной, и эксперт способен найти семантически более корректный способ инициализации. Например, в данном случае, имя схемы можно образовать, соединив все имена онтологий в одну строку и добавив к ней слово *Schema:*

```
name <- g.ontology->
   collect(e| e.uriRef->any().name->sum()+'Schema'
```

Имена собираются из онтологий при помощи операции *collect*, и затем соединяются при помощи операции *sum* (семантика этих операций в ATL совпадает с семантикой одноименных операций OCL). Тем не менее, автоматически построенная инициализация является основой для дальнейшей работы эксперта.

Порождающее правило для инициализации свойства, тип которого является единственным образом своего прообраза в отношении соответствия R. Инициализация свойства в целевом образце возможна также, если тип свойства является единственным образом своего прообраза в отношении соответствия *R*. Примером такого свойства является *SchemaDef.modules,* участвующее в соответствии *Graph.ontology ~ SchemaDef.modules.* Свойство *modules* является связью один-ко-многим между элементами *SchemaDef* и *ModuleDef*, т.е. тип свойства *modules – ModuleDef.* Прообразом элемента *ModuleDef* в отношении *R*

является элемент *OWLOntology*. Единственным образом в отношении *R* у этого элемента является *ModuleDef*. Инициализация этого свойства в правиле *Graph2Schema* выглядит следующим образом:

```
rule Graph2Schema{
  from g: OWL!OWLGraph
  to s: Synthesis!SchemaDef(
       modules <- g.ontology
  )
}
```

Такая инициализация предполагает, что для экземпляров коллекции *g.ontology* (которые являются экземплярами класса *OWLOntology*) существует сопоставляющее правило, которое преобразует их в экземпляры коллекции *s.modules* (которые являются экземплярами класса *ModuleDef*). Такое правило будет неявно вызываться для каждого экземпляра коллекции *g.ontology* (в соответствии с семантикой языка ATL). Действительно, согласно приведенным выше правилам, транслятор включает необходимое правило *OWLOntology2ModuleDef*.

Порождающие правила для инициализации свойства, тип которого является не единственным образом своего прообраза в отношении соответствия R. Рассмотрим теперь свойства, типы которых являются не единственными образами своих прообразов в *R*. Правила преобразования типов этих свойств порождают несколько целевых элементов. Поэтому, в общем случае, необходимо явно указать, какой же именно целевой элемент будет использован для инициализации свойства.

Для такого рода инициализаций используется встроенная функция *thisModule.resolveTemp(var, target-pattern-name)*. Идентификатор *thisModule* говорит о том, что *resolveTemp* является функцией модуля трансформации. Первым параметром функции является переменная, содержащая элемент исходной модели, из которого должен быть получен искомый целевой элемент. Вторым параметром является строка с именем искомого целевого элемента. В языке ATL использование функции *thisModule.resolveTemp()* возможно только в императивном разделе *do* правила. Этот раздел располагается в правилах после раздела *to*; в нем при помощи императивных операторов инициализируются те свойства, которые не инициализированы декларативно в разделе *to*.

Примером свойства, тип которого есть не единственный образ своего прообраза в *R,* является *ModuleDef.containedTypes*. Свойство является связью один-ко-многим. Его тип *ADTDef* участвует в сопоставлении *ADTDef ~ OWLClass*. Однако, *OWLClass* участвует также в сопоставлении *OWLClass ~ ClassDef*. Поэтому инициализация свойства *containedTypes* в правиле *OWLOntology2ModuleDef* выглядит следующим образом:

```
rule OWLOntology2ModuleDef{
  from o: OWL!OWLOntology
```

```
    to m: Synthesis!ModuleDef
    do{
          m.containedTypes <- o.owlUniverse->
           select(e|e.oclIsTypeOf(OWL!OWLClass))->
           collect(e|thisModule.resolveTemp(e, 'type'));
    }
}
```

Преобразование экземпляров коллекции *o.owlUniverse* в экземпляры коллекции *m.containedTypes* производится при помощи операции *collect,* вызывающей для каждого экземпляра функцию *resolveTemp*. Второй аргумент функции указывает, что для преобразования следует использовать целевой элемент *type* правила *OWLClass* (раздел 2.2.2).

В инициализации используется также операция *select* (семантика которой совпадает с семантикой одноименной операции OCL), выбирающая из коллекции для последующего преобразования только элементы класса *OWLClass*. Это делается потому, что тип коллекции *o.owlUniverse* является суперклассом *OWLClass*, и в коллекции могут содержаться элементы, имеющие отличный от *OWLClass* тип. Такое появление операции *select* при инициализации является примером применения вспомогательного *Порождающего правила выборки подходящих элементов коллекции при инициализации свойства*.

Для свойств, являющихся связью один-к-одному, порождающее правило инициализации при помощи *resolveTemp* выглядит более просто. Так, для свойства *AssociationMetaclass.inverse* инициализация выглядит следующим образом:

```
rule OWLObjectProperty{
    from p: OWL!OWLObjectProperty
    to
          assoc: Synthesis!AssociationMetaclassDef
    do{
          assoc.inverse <-
           thisModule.resolveTemp(
            p.OWLInverseOf, 'assoc');
    }
}
```

Применения операций *select* и *collect* в данном случае не требуется, достаточно вызова функции *resolveTemp*, обращающейся к правилу преобразования элемента *OWLObjectProperty*:

```
rule OWLObjectProperty{
    from p: OWL!OWLObjectProperty
    to a: Synthesis!AttributeDef(…),
    assoc: Synthesis!AssociationMetaclassDef(…)
    do{…}
}
```

На основании полученной таким образом спецификации модуля *OWL2Synthesis* на языке ATL генерируется транслятор, обеспечивающий трансформацию произвольных моделей уровня M1 на языке OWL в соответствующие модели уровня M1 на языке СИНТЕЗ.

Заметим, что соответствия между элементами моделей всегда содержат лишь часть семантики отображения моделей. Например, элементы могут находиться в соответствии, а отображение при этом зависит от некоторого сложного условия. Недостающая семантика должна быть добавлена в трансформацию вручную экспертом. Таким образом, сценарий построения трансформации состоит из двух шагов:

- автоматического построения трансформации на основе соответствий между элементами моделей (построенное отображение называется *образцом*);
- превращения экспертом образца в полноценную трансформацию на основании семантики, не содержащейся во множестве соответствий.

Заметим также, что в процессе унификации моделей [2] может потребоваться расширение [10] целевой (канонической) модели. Список соответствий между элементами моделей при этом пополняется соответствиями между элементами исходной модели и элементами расширения. Предлагаемый метод построения трансформаций при этом должен быть применен к полному списку соответствий.

Пример применения полученного транслятора *OWL2Synthesis* для преобразования конкретных моделей уровня M1 из области культурного наследия приведен в приложении.

3 Построение обратных трансформаций моделей на основе прямых трансформаций моделей

3.1 Обратные трансформации моделей и обратимые конструкции языков трансформации

Рассмотрим трансформацию t модели U в модель V и трансформацию r модели V в модель U. Обозначим через $M(U)$ множество моделей, конформных модели U.

Определение. Назовем *r обратной* трансформации *t* на множестве $N \subseteq M(U)$, если для любой модели *s* из *N* выполнено $r(t(s)) = s$. Назовем *r всюду обратной* для *t*, если *r* является обратной для *t* на $M(U)$.

Определение. Назовем *r слабо обратной* трансформации *t* на множестве $N \subseteq M(U)$, если для любой модели *s* из *N* выполнено $t(r(t(s))) = t(s)$. Назовем *r всюду слабо обратной* для *t*, если *r* является слабо обратной для *t* на $M(U)$.

Обратную трансформацию *r* возможно построить только для такой прямой трансформации *t,* которая передает всю информацию, содержащуюся в исходной модели. Слабо обратная трансформация восстанавливает всю информа-

цию, передаваемую прямой трансформацией, при этом прямая трансформация может передавать только часть информации, содержащейся в исходной модели. Очевидно, что обратная трансформация на множестве является слабо обратной на том же множестве. Обратную трансформацию возможно построить в том случае, когда *в каждой инициализации каждого правила атрибут целевой модели связывается ровно с одним атрибутом исходной модели.* В некоторых случаях возможно ослабление данного условия.

Обратная трансформация является полным и правильным, не требующей ручной модификации. Слабо обратная трансформация является образцом, который предлагается эксперту для проверки и модификации в случае необходимости.

Определение. Назовем подмножество L трансформаций, выраженных на некотором языке трансформации моделей (например, ATL), *(слабо) обратимым*, если существует алгоритм, конструирующий для любой трансформации t из L всюду (слабо) обратную ей трансформацию r.

Определение. Назовем подмножество K конструкций (операторов, функций, операций) языка трансформации моделей (например, ATL) *обратимым (слабо обратимым)*, если множество трансформаций, построенных на основе этих конструкций, является обратимым (слабо обратимым).

Возможно, что слабо обратимым является каждый язык трансформации моделей (и ATL, в частности) целиком. Однако, слабое обращение имеет смысл не для всех конструкций языка.

Задача автоматизации построения обратных трансформаций сводится к следующим подзадачам:

- нахождение максимального (слабо) обратимого подмножества K_R конструкций языка трансформации моделей;
- разработка алгоритма построения обратного отображения для прямого отображения, построенного на основе K_R;
- доказательство корректности разработанного алгоритма. Доказательство проводится индукцией по набору конструкций языка.

В данной статье рассматриваются правила построения обратных трансформаций для подмножества конструкций языка ATL. Полное формальное доказательство корректности является задачей дальнейшей работы.

3.2 Правила построения обратных трансформаций информационных моделей

Обозначим семантическую функцию, преобразующую конструкции прямой трансформации в конструкции обратной трансформации, через *r[]*. Будем также называть преобразование конструкций прямой трансформации в конструкции обратной трансформации *обращением*.

Метод построения обратных трансформаций в настоящее время включает *правила обращения* следующих конструкций языка ATL:

- заголовок отображения;

- вспомогательные правила (helpers), являющиеся ATL-эквивалентом методов языка Java;
- сопоставляющие правила;
- локальные переменные правил;
- инициализация целевых элементов в декларативной и императивной частях правил;
- условный оператор *if-then* и оператор присваивания в инициализациях;
- выражения путей в элементах исходной модели;
- OCL-выражения *if-then* и *let*;
- операции примитивных типов OCL: логическое отрицание; сложение, вычитание, операция *toString* для числовых типов; умножение, деление, операции *sqrt, exp, log, toDegrees, toRadians* типа *Real*; операции *toInteger, toReal, toSequence, split, concat* (в частном случае) типа *String*;
- операция *toSequence* OCL-типов *Set, OrderedSet, Bag*; операция *toSet* OCL-типов *OrderedSet, Bag*;
- операции *append, prepend, insertAt* OCL-типа *Sequence*;
- операции над коллекциями OCL *any* и *select*.

Для некоторых конструкций возможно лишь слабое обращение (условный оператор, операции *any* и *select*).

Целью дальнейшей работы является изучение возможности обращения таких конструкций ATL как вызываемые (императивные) правила (called rules), итеративный целевой образец правила, оператор *for* и некоторых других.

В целом, построение обратной трансформации состоит из двух последовательных шагов:

- автоматическое построение обратной трансформации на основе прямой трансформации. Каждая слабо обратимая конструкция прямой трансформации обращается, остальные конструкции помечаются;
- ручная модификация автоматически построенной трансформации экспертом. Обратимые конструкции не требуют модификации. Обращение слабо обратимых конструкций проверяется и модифицируется, если это необходимо. Помеченные конструкции обращаются вручную.

Ниже в данном разделе рассмотрены и проиллюстрированы некоторые из упомянутых правил обращения.

3.2.1 Обращение заголовка трансформации

Заголовок прямой трансформации исходной модели *S* в целевую модель *T* преобразуется в заголовок обратной трансформации следующим образом:

```
r[      module S2T
        create OUT: T from IN: S
] =
module T2S;
create OUT: S from IN: T;
```

3.2.2 Обращение правил трансформации

Рассмотрим правило установления соответствия *U2V* общего вида, преобразующее элемент *U* модели *S* в элемент *V* модели *T*:

```
rule U2V{
   from u: S!U
   to v: T!V
   do{ statements }
}
```

Заметим, что в таком правиле инициализация свойств целевого элемента производится не в разделе *to* образца целевого элемента, а в императивном разделе *do*. Этот раздел состоит из последовательности *statements* императивных операторов, соединенных секвенциальным оператором (;). Без ограничения общности можно рассуждать об обращении правил именно такого вида. Действительно, правило, содержащее инициализацию свойства в разделе *to* можно преобразовать в эквивалентное правило, содержащее инициализацию только в разделе *do*. Так, правило

```
rule OWLOntology2ModuleDef{
   from o: OWL!OWLOntology
   to s: Synthesis!ModuleDef(
         name <- o.uriRef.name
   )
}
```

эквивалентным образом преобразуется в правило

```
rule OWLOntology2ModuleDef{
   from o: OWL!OWLOntology
   to m: Synthesis!ModuleDef
   do{ m.name <- o. uriRef.name; }
}
```

Итак, сопоставляющее правило обращается следующим образом:

```
r[       rule U2V{
          from u: S!U
          to v: T!V
          do{ statements }
}
] =
rule V2U{
   from v: T!V
   to u: S!U
   do{ r[statements] }
}
```

Каждый оператор из секции *do* прямого правила преобразуется в один или несколько операторов секции *do* обратного правила.

Текущая реализация языка ATL поддерживает три вида операторов: оператор присваивания, итеративный оператор *for* и условный оператор *if-then*. В данной статье рассматриваются правила обращения лишь для *оператора присваивания*. При эквивалентном преобразовании инициализация свойства в разделе *to* преобразуется именно в оператор присваивания в разделе *do*. Это иллюстрируется приведенным выше примером *OWLOntology2ModuleDef*.

3.2.3 Обращение оператора присваивания

Оператор присваивания в правилах имеет следующий вид:

```
rule U2V{
   from u: S!U
   to v: T!V
   do{ t(v) <- s(u) }
}
```

Здесь *t(v)* – некоторое свойство элемента целевой модели, *s(u)* – выражение языка ATL (являющееся декларативным выражением OCL). В данной работе рассматриваются такие присваивания, в которых *s* зависит от исходной переменной *u* (именно в них происходит связывание элементов исходной и целевой моделей).

Обращение присваивания (действие функции *r* на присваивании) задается следующим образом. К присваиванию применяются *правила обращения присваиваний*. Каждое правило обращения преобразует присваивание таким образом, что правая часть присваивания упрощается, а левая – усложняется. Правило имеет вид $b_1 \Rightarrow b_2$, где b_1 и b_2 – присваивания. Правила применяются до тех пор, пока правая часть не превратится выражение вида *u.a*, где *u* – имя целевой переменной, *a* – атрибут целевого элемента. Если в процессе обращения обнаружилось, что к присваиванию не применимо ни одно правило, а правая часть представляет собой выражение, отличное от *u.a*, то присваивание считается *необратимым* (автоматически необратимым). Обращение такого присваивания производится экспертом вручную, если обращение вообще возможно. Правило обращения может разбить присваивание на несколько присваиваний. При этом в дальнейшем правила обращения применяются к каждому из полученных присваиваний до максимального упрощения их правых частей.

Пусть присваивание *t(v) <- s(u)* превратилось в процессе применения правил обращения в набор присваиваний

t_1 <- u.a_1; … , t_n <- u.a_n;

где t_i – выражения, a_i – имена свойств. Тогда действие функции *r* на присваивании *t(v) <- s(u)* определяется следующим образом:

r[t(v) <- s(u)] = u.a_1 <- t_1, … , u.a_n <- t_n

Рассмотрим различные правила обращения присваиваний в контексте правила *U2V*.

В случае, если присваивание представляет собой простое соответствие свойств, не требуется применение никаких правил обращения присваиваний, преобразование следует из определения функции *r*:

```
r[v.b <- u.a] = u.a <- v.b
```

3.2.5 Обращение путей в структуре исходного элемента

Рассмотрим правило обращения *путей* в структуре исходного элемента. Рассмотрим присваивание

```
t <- s.d.b
```

где *s* – выражение типа T_s; *d* – атрибут типа T_s , имеющий тип T_d; *b* – атрибут типа T_d, имеющий тип T_b. Заметим, что при использовании таких выражений пути происходит уплощение структуры исходного типа. Отсюда следует, что в обратном преобразовании следует восстановить структуру. В данном случае это означает создание экземпляра типа T_d:

```
t <- s.d.b ⇒ t <- w.b; w <- s.d
```

Здесь *w* – новая переменная типа T_d, которая становится дополнительным элементом целевого образца (target pattern) обратного правила. Тем самым правило, кроме экземпляра типа *U*, будет порождать экземпляр типа T_d:

```
rule V2U{
   from v: T!V
   to u: S!U, w: T!Td
   do{ … }
}
```

Первое из полученных присваиваний (*t <- w.b*) относится к элементу *w* целевого образца, второе – к элементу *u* целевого образца.

Рассмотрим пример использования обращения путей. Прямое правило *OWLOntology2ModuleDef* преобразуется в обратное правило *ModuleDef2OWLOntology*:

```
rule OWLOntology2ModuleDef{
   from o: OWL!OWLOntology
   to m: Synthesis!ModuleDef(
         name <- o.uriRef.name
   )
}
rule ModuleDef2OWLOntology{
   from m: Synthesis!ModuleDef
   to o: OWL!OWLOntology,
```

```
        u: OWL!URIReference
    do{ o.uriRef <- u; u.name <- m.name; }
}
```

3.2.6 Условие обратимости присваиваний

Вообще, обращение (автоматическое) присваивания возможно, если в *правой части присваивания используется ровно одно свойство (атрибут или связь) исходной модели*. Если свойству целевой модели присваивается выражение, которое включает несколько свойств исходной модели, то обращение такого связывания невозможно (из-за неоднозначности). При таких отображениях в общем случае теряется информация о том, какие значения имели исходные свойства. В целевом свойстве сохраняется лишь некоторая агрегатная информация об этих значениях.

3.2.7 Обращение операций примитивных типов

В ряде случаев возможно автоматическое обращение присваиваний, в которых OCL-выражения содержат операции над примитивными типами. В данном разделе мы рассмотрим лишь некоторые из обратимых операций. Пусть *c* обозначает константу некоторого примитивного типа (*Boolean, Integer, Real*), *t* и *s* – выражения, включающие атрибуты целевого и исходного элемента соответственно.

Возможно, например, обращение логического отрицания. Правило обращения присваивания для операции отрицания выглядит следующим образом:

```
t <- not s ⇒ not t <- s
```

При использовании остальных логических операций (*and, or, implies*) в присваиваниях возможна потеря информации, а потому обращение невозможно. Слабое обращение логических операций возможно, но семантически некорректно.

Для числовых типов (*Integer, Real*) возможно обращение операций сложения и вычитания:

```
t <- s - c ⇒ t + c <- s
t <- c - s ⇒ c - t <- s
t <- s + c ⇒ t - c <- s
```

Для типа *Real* возможно обращение операций деления и умножения:

```
t <- s/c ⇒ t*c <- s
t <- c/s ⇒ c/t <- s
t <- s*c ⇒ t/c <- s
```

Для типа *String* возможно обращение операции преобразования к типу коллекции:

```
t <- s->toSequence() ⇒ t->sum() <- s
```

Строка преобразуется в последовательность строк, состоящих из одного символа, в обратном преобразовании все строки из последовательности объединяются в одну.

3.2.8 Обращение операций над коллекциями

В языке OCL определены различные типы коллекций: *Set, OrderedSet, Bag, Sequence*. В данном разделе рассматриваются правила обращения присваиваний, содержащих некоторые из операций, определенных над типами коллекций. Семантика всех операций над коллекциями, используемых в данном разделе, и оператора *let* совпадает с соответствующей семантикой в языке OCL.

Пусть *e, elm, n, cond* – выражения, не зависящие от атрибутов исходной модели, *t* и *s* – выражения, включающие атрибуты целевого и исходного элемента соответственно.

Для типа *Sequence* возможно обращение операции *append* присоединения элемента в конец последовательности:

```
t <- s->append(elm) ⇒
  let seq: Sequence = t in seq->subSequence(1, seq->size()-1) <- s

t <- e->append(s) ⇒ t->last() <- s
```

операции *prepend* присоединения элемента в начало последовательности:

```
t <- s->prepend(elm) ⇒
   let seq: Sequence = t in seq->subSequence(2, seq->size()) <- s

t <- e->prepend(s) ⇒ t->first() <- s
```

операции *insertAt* вставки элемента в коллекцию на место с некоторым номером *n*:

```
t <- e->insertAt(n, s) ⇒ t->at(n) <- s

t <- s->insertAt(n, e) ⇒
   let seq: Sequence = t in
   let head: Sequence = seq->subSequence(1, n-1) in
   let tail: Sequence = seq->subsequence(n+1, seq->size()) in
   tail.iterate(elm, acc: Sequence = head | acc.append(elm))
   <- s
```

Операции *any* выбора произвольного элемента коллекции и *select* выбора подмножества элементов коллекции по условию не являются обратимыми, однако допустимым является вариант слабого их обращения:

```
r[t <- u.c->any(e | cond)] = u.c <- u.c->including(t)

r[t <- u.c->select(e | cond)] = u.c <- u.c->union(t)
```

Здесь *u* – исходная переменная, *c* – атрибут исходного элемента. Тип атрибута *c* является типом коллекции.

4 Родственные работы

Существующие подходы к автоматизированному построению отображений моделей обычно включают некоторые методы сопоставления (matching) моделей. В работе [9] предлагается онтологически-базированный подход к отображению моделей (ontMT). Технологические пространства MDA и онтологий объединяются для автоматизации построения и эволюции отображений моделей. Целевая и исходная модели связываются с некоторой эталонной онтологией. Вычисляются отношения между элементами моделей и строится двунаправленное отображение моделей, выраженное в языке отношений QVT Relations.

В работе [20] предлагается полуавтоматический подход к построению отображений при помощи сплетающих моделей (weaving models). Рассматривается итеративная процедура построения сплетающих связей (weaving links), вычисления степени соответствия элементов моделей и выбора оптимальных связей. Сплетающая модель трансформируется в отображение на языке ATL при помощи высокоуровневых преобразований.

В работе [21] рассматривается подход, основанный на примерах (by-example). Отображение определяется сначала на конкретных моделях уровня M1, иллюстрирующих нотацию языков, которые необходимо отобразить друг в друга. Информация, содержащаяся в отображениях конкретных моделей, объединяется и на ее основании строится двунаправленное ATL-отображение моделей уровня M2.

Разрабатываются алгебраическая теория двунаправленных отображений моделей [22], а также двунаправленные языки запросов для обновляемых взглядов [23].

Подход, предлагаемый в данной статье, является более технически ориентированным. Он посвящен автоматическому построению трансформаций моделей и абстрагируется от методов сопоставления моделей. Предложены порождающие правила автоматического построения трансформаций моделей на основе класса соответствий между элементами моделей. Для построения трансформаций используются некоторые специальные конструкции языка ATL.

Предлагаемый метод построения обратных трансформаций особенно полезен, когда прямая трансформация не является двунаправленной изначально.

5 Заключение

Текущий период развития информационных систем характеризуется большим разнообразием разнородных информационных моделей. Принимая во внимание трудоемкость процесса построения трансформаций моделей, задача автоматизации построения трансформаций становится особенно актуальной.

В данной статье рассмотрены два метода автоматизированного построения трансформаций. Первый метод предназначен для построения трансформаций на основе установленных соответствий между элементами моделей. Второй метод предназначен для построения обратных трансформаций моделей на основе прямых трансформаций.

В качестве языка трансформации моделей в работе рассматривается ATL – аналог языка QVT, определенного стандартом OMG.

Методы автоматизации построения трансформаций моделей являются частью более общей работы – создания Конструктора унифицирующих информационных моделей. Целью Унификатора является доказательное приведение множества разнотипных моделей информационных ресурсов к каноническому, унифицированному представлению в процессе интеграции информационных ресурсов в предметных посредниках

Представленные методы могут быть адаптированы для построения трансформаций, выраженных на сходных с ATL языках (например, QVT или его диалектах). Конкретная модель уровня M3 также не является существенной, могут быть использованы и другие метаметамодели, например MOF или KM3. Предлагаемые методы также могут быть использованы независимо от Унификатора, для неверифицируемого построения прямых и обратных трансформаций информационных моделей на основе технологии MDA.

Список литературы

1. Kalinichenko L. A., Briukhov D. O., Martynov D. O., Skvortsov N.A., Stupnikov S.A. Mediation Framework for Enterprise Information System Infrastructures // The 9th International Conference on Enterprise Information Systems (ICEIS). – Funchal, 2007. – P. 246-251.
2. Захаров В. Н., Калиниченко Л. А., Соколов И. А., Ступников С. А. Конструирование канонических информационных моделей для интегрированных информационных систем // Информатика и ее применения. – М., 2007. – Т. 1, Вып. 2. – С. 15-38.
3. Kalinichenko L. A., Stupnikov S. A. Constructing of Mappings of Heterogeneous Information Models into the Canonical Models of Integrated Information Systems // Advances in Databases and Information Systems: Proc. of the 12th East-European Conference. - Pori: Tampere University of Technology, 2008. - P. 106-122.
4. Van den Brand M. G. J. et al. The ASF+SDF meta-environment: a component-based language development environment // Compiler Construction 2001 / Ed. by R. Wilhelm.- Berlin-Heidelberg: Springer, 2001. P. 365-370.
5. OMG Model Driven Architecture. - http://www.omg.org/mda/
6. Meta Object Facility (MOF) 2.0 Core Specification // http://www.omg.org/cgi-bin/apps/doc?ptc/03-10-04.pdf, 2003.
7. Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification // http://www.omg.org/cgi-bin/doc?ptc/2007-07-07, 2007.

8. Falleri J.-R., Huchard M., Lafourcade M., Nebut C. Metamodel Matching for Automatic Model Transformation Generation // Proc. of MoDELS. – Berlin-Heidelberg: Springer, 2008. – P. 326-340.
9. Roser S., Bauer B. Automatic Generation and Evolution of Model Transformations Using Ontology Engineering Space // J. Data Semantics (JODS) – 2008. - V. 11. - P. 32-64.
10. Kalinichenko L.A. Methods and tools for equivalent data model mapping construction. Proc. of the International Conference on Extending Database Technology EDBT'90. LNCS 416. - Berlin-Heidelberg: Springer-Verlag, 1990. - P. 92-119.
11. Калиниченко Л.А. Синтез канонических моделей, предназначенных для достижения семантической интероперабельности неоднородных источников информации // Системы и средства информатики: Спец. вып. Формальные методы и модели в композиционных инфраструктурах распределенных информационных систем / Под ред. И. А. Соколова. — М.: ИПИ РАН, 2005. – С.11-39.
12. Ступников С.А. Автоматизация верификации уточнения при композиционном пректировании информационных систем и посредников // Системы и средства информатики: Спец. вып. Формальные методы и модели в композиционных инфраструктурах распределенных информационных систем / Под ред. И. А. Соколова. — М.: ИПИ РАН, 2005. – С. 96-119.
13. Л. А. Калиниченко, Н. А. Скворцов. Реверсивное онтологическое моделирование при унифицированном представлении различных онтологических моделей источников информации в предметном посреднике. // Системы и средства информатики: Спец. вып. Формальные методы и модели в композиционных инфраструктурах распределенных информационных систем / Под ред. И. А. Соколова. — М.: ИПИ РАН, 2005. – С. 184 - 212.
14. ATL Project // http://www.eclipse.org/m2m/atl/
15. OMG/OCL Object Constraint Language (OCL) 2.0. OMG Final Adopted Specification. ptc/03-10-14, 2003.
16. Budinsky, F., Steinberg, D., Ellersick, R., Grose, T. Eclipse Modeling Framework, Chapter 5 ”Ecore Modeling Concepts”. - Addison Wesley Professional, 2004.
17. Jouault, F., and Bezivin, J. KM3: a DSL for Metamodel Specification // Proc. of 8th IFIP International Conference on Formal Methods for Open Object-Based Distributed Systems, LNCS 4037. - Berlin-Heidelberg: Springer-Verlag, 2006. -P. 171-185.
18. OWL Web Ontology Language Reference. W3C Recommendation. - http://www.w3.org/TR/owl-ref/, 2004.
19. Kalinichenko L. A., Stupnikov S. A., Martynov D. O. SYNTHESIS: a Language for Canonical Information Modeling and Mediator Definition for Problem Solving in Heterogeneous Information Resource Environments. - M.: IPI RAS, 2007 - 171 p. - http://synthesis.ipi.ac.ru/synthesis/publications/07synthesis
20. Fabro, M.D.D., Valduriez, P. Semi-automatic model integration using matching transformations and weaving models // Proc. of the 22nd ACM Symposium on Applied Computing, Model Transformation Track. – 2007.
21. Wimmer M., Strommer M., Kargl H., Kramler G. Towards Model Transformation Generation By-Example // Proc. of HICSS. - Los Alamitos: IEEE Computer Society. – 2007.
22. Diskin Z. Algebraic Models for Bidirectional Model Synchronization // Proc. of MoDELS. – Berlin-Heidelberg: Springer, 2008. – P. 21-36.
23. Bohannon A., Pierce B. C., Vaughan J. A.: Relational lenses: a language for updatable views // Proc. of PODS. – 2006. – P. 338-347.
24. Jouault, F., Bézivin, J., and Kurtev, I. TCS: a DSL for the Specification of Textual Concrete Syntaxes in Model Engineering // Proc. of the Fifth International Conference on Generative programming and Component Engineering GPCE'06. - 2006.

Приложение

Здесь приводится пример применения прямой трансформации *OWL2Synthesis*, определенной в разделе 2, для преобразования конкретных моделей уровня M1 из области культурного наследия. Для удобства модели приводятся в виде, удовлетворяющем конкретному синтаксису языков OWL и СИНТЕЗ, но трансформация *OWL2Synthesis* преобразует модели в рамках метамодели *Ecore*. Для преобразования моделей из конкретного синтаксиса в *Ecore* и обратно используется технология TCS (Textual Concrete Syntax [24]), разрабатываемая в рамках проекта Eclipse GMT (Generative Modeling Technologies).

В качестве исходной модели рассматривается подмножество модели *Museum*:

```
<?xml version = '1.0' encoding = 'ISO-8859-1' ?>
<rdf:RDF xmlns:owl = 'http://www.w3.org/2002/07/owl#'
         xmlns = 'http://example.org/Museum#'>
  <owl:Ontology rdf:about = 'Museum'/>
  <owl:Class rdf:ID = 'Artefact'>
    <rdfs:label>Artefact</rdfs:label>
  </owl:Class>
  <owl:Class rdf:ID = 'Artist'>
    <rdfs:label>Artist</rdfs:label>
  </owl:Class>
  <owl:ObjectProperty rdf:ID = 'Artefact.hasArtist'>
    <rdfs:domain rdf:resource = '#Artefact'/>
    <rdfs:range rdf:resource = '#Artist'/>
    <owl:inverseOf rdf:resource = '#Artist.creates'/>
  </owl:ObjectProperty>
  <owl:DatatypeProperty rdf:ID = 'Artist.lastName'>
    <rdfs:domain rdf:resource = '#Artist'/>
    <rdfs:range rdf:resource =
        'http://www.w3.org/2001/XMLSchema#string'/>
  </owl:DatatypeProperty>
  <owl:ObjectProperty rdf:ID = 'Artist.creates'>
    <rdfs:domain rdf:resource = '#Artist'/>
    <rdfs:range rdf:resource = '#Artefact'/>
    <owl:inverseOf rdf:resource = '#Artefact.hasArtist'/>
  </owl:ObjectProperty>
</rdf:RDF>
```

Спецификация на OWL содержит два класса – *Artist* и *Artefact*. Художник является создателем некоторого артефакта (свойство *creates*), и наоборот, артефакт создается некоторым художником (свойство *hasArtist*). Данная спецификация, конформная модели *OWL*, автоматически трансформируется в спецификацию, конфирмную модели *Synthesis*:

```
{ MuseumSchema; in: schema;
}
```

```
{ Museum; in: module, ontology;
  schema: MuseumSchema;

  type:
  { Artefact; in: type, owl;
    hasArtist: Artist;
    metaslot
      in: hasArtistMetaclass;
      inverse: Artist.creates;
    end
  },
  { Artist; in: type, owl;
    lastName: string;
    creates: Artefact;
    metaslot
      in: createsMetaclass;
      inverse: Artefact.hasArtist;
    end
  };

  class_specification:
  { artefact; in: class, owl;
    instance_section: Artefact;
  },
  { artist; in: class, owl;
    instance_section: Artist;
  },
  { hasArtistMetaclass; in: association, metaclass, owl;
    instance_section: {
      association_type: {{0, inf},{0, inf}};
      domain: artifact;
      range: artist;
    };
  },
  { createsMetaclass; in: association, metaclass, owl;
    instance_section: {
      association_type: {{0, inf},{0, inf}};
      domain: artist;
      range: artifact;
    };
  };
}
```

Онтология преобразуется в модуль языка СИНТЕЗ, классы OWL – в одноименные типы и классы языка СИНТЕЗ, свойства OWL – в атрибуты и метаклассы ассоциаций.