Семантическая трансформация канонической информационной модели в формальный язык спецификаций для верификации уточнения♣

© С.А. Ступников

Институт проблем информатики РАН, г. Москва
ssa@ipi.ac.ru

Аннотация

Статья посвящена разработке методов и средств достижения семантической интероперабельности неоднородных информационных ресурсов при создании интегрированных информационных систем. В качестве основного факта, подлежащего формальной проверке, рассматривается *уточнение* спецификации системы спецификациями ресурсов. Предложена трансформация языка СИНТЕЗ, нацеленного на разработку предметных посредников для решения задач над неоднородными ресурсами в формальный язык спецификаций AMN, поддерживающий уточнение. Тем самым достигается возможность доказательства уточнения спецификаций языка СИНТЕЗ при помощи существующих автоматических и интерактивных систем доказательства.

1 Введение

Для обеспечения семантической интероперабельности существующих неоднородных информационных ресурсов, используемых при создании интегрированных информационных систем (в том числе, цифровых библиотек [10]), необходимо применение методов формальной верификации. Одним из способов проверки правомочности использования ресурса для реализации части интегрированной системы является использование методов и средств, формализующих понятие *уточнения* [2]. Неформально, спецификация *B уточняет* спецификацию *A*, если пользователь может использовать *B* вместо *A*, не замечая факта замены *A* на *B*. Перспективной технологией, использующей уточнение для верификации интеграции неоднородных информационных ресурсов, является технология *предметных посредников* [1].

Можно выделить две крупные задачи, связанные с интеграцией ресурсов в посредниках, которые существенным образом используют понятие уточнения:

- интеграция информационных моделей ресурсов;
- регистрация ресурсов в предметных посредниках.

Модели информационных ресурсов предметной области, для которой создается предметный посредник, могут быть совершенно различными. Целью интеграции моделей является создание *канонической информационной модели*, унифицирующей (т. е. представляющей однородным образом) модели ресурсов [3]. *Унификация* модели ресурсов *R* есть приведение ее к канонической информационной модели *C*, т. е. создание такого расширения *E* канонической модели и такого отображения *M* исходной модели в расширенную каноническую, что исходная модель уточняет расширенную каноническую модель. Уточнение моделей означает, что для любой допустимой спецификации *r* модели *R* ее образ *M(r)* при отображении *M* уточняется спецификацией *r*. Верификация уточнения моделей осуществляется на наборе образцов спецификаций исходной модели. Интеграция моделей ресурсов является предварительным этапом для регистрации конкретных ресурсов в предметном посреднике [1].

Одним из этапов регистрации ресурсов является согласование (интеграция) онтологии посредника и онтологий ресурсов. Верификацию процесса интеграции онтологий возможно проводить при помощи автоматических систем вывода в том случае, если онтологии не выходят за рамки дескриптивных логик. В противном случае задачи интеграции сводятся к доказательству уточнения в логике первого порядка [4].

Еще одним важным этапом регистрации являются поиск спецификаций типов ресурсов, релевантных типам концептуальной схемы посредника, и доказательство уточнения типами ресурсов типов посредника. Тем самым обеспечивается возможность реализации виртуальной схемы посредника посредством схем ресурсов.

Заметим, что онтологии и концептуальные схемы посредника и ресурсов для последующей интеграции выражаются в канонической информационной модели. Таким образом, задачи верификации интеграции ресурсов сводятся к доказательству уточнения спецификаций канонической модели.

Труды 12й Всероссийской научной конференции «Электронные библиотеки: перспективные методы и технологии, электронные коллекции» – RCDL'2010, Казань, Россия, 2010

В данной статье в качестве канонической модели рассматривается язык СИНТЕЗ [5], нацеленный на разработку предметных посредников для решения задач в средах неоднородных ресурсов. Целью статьи является построение трансформации языка СИНТЕЗ в формальный язык спецификаций, поддерживающий понятие уточнения. В качестве такого языка выбрана Нотация абстрактных машин (AMN) [2], для которой существуют специализированные инструментальные средства доказательства уточнения [6]. Трансформация реализует отображение языка СИНТЕЗ в AMN [11] и представляет собой программу, получающую на вход произвольную допустимую спецификацию на языке СИНТЕЗ и автоматически порождающую соответствующую спецификацию на языке AMN. При этом можно говорить, что языку СИНТЕЗ сообщается формальная семантика в языке AMN, а потому данная трансформация называется *семантической*. Статья сосредоточена на трансформации основных объектных черт языка СИНТЕЗ.

В качестве языка для определения трансформации выбран ATL (ATLAS Transformation Language) [10]. ATL является одним из языков трансформации моделей, разрабатываемых в контексте *Движимой моделями архитектуры* (MDA), поддерживаемой стандартом MOF [8] консорциума OMG.

Статья организована следующим образом. В разделах 2 и 3 описаны основные черты языков AMN и ATL соответственно. В разделе 4 рассмотрена и продемонстрирована на примере трансформация канонической информационной модели в язык AMN.

2 Формальный язык спецификаций AMN

Язык AMN представляет собой теоретико-модельную нотацию, основанную на теории множеств Цермело – Френкеля с аксиомой выбора и типизированном языке первого порядка со встроенными типами и конструкторами типов (сортов).

Язык предикатов AMN включает обычный набор логических связок (конъюнкцию, дизъюнкцию, импликацию, отрицание), кванторы существования, предикат принадлежности и простую арифметическую теорию.

Спецификации AMN называются *абстрактными машинами*. AMN позволяет интегрированно рассматривать спецификацию пространства состояний и поведения машины (определенного операциями на состояниях). Спецификация состояния вводится переменными состояния вместе с инвариантами – ограничениями, которые должны всегда удовлетворяться. Операции абстрактных машин основаны на обобщенных подстановках (generalized substitutions), описывающих переходы между состояниями системы.

Уточнение формализуется в AMN путем формулировки ряда теорем специального вида, так называемых *proof obligations*. Такие теоремы формулируются автоматически при помощи инструментальных средств поддержки [6] на основании *склеивающих инвариантов* – инвариантов, соотносящих состояния уточняемой и уточняющей спецификаций. Теоремы могут быть доказаны при помощи инструментальных средств поддержки автоматического и (или) интерактивного доказательства.

3 Язык трансформации моделей ATL

Базовыми составляющими MDA являются модели. Они рассматриваются как первичные сущности, и наиболее важными операциями манипулирования моделями становятся трансформации 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*, модели уровня M3 – метаметамодели, предназначенные для описания моделей уровня M2.

ATL представляет собой декларативно-императивный язык, позволяющий описывать способ порождения моделей (называемых *целевыми*) на основании моделей, называемых *исходными*. Система типов и операций над типами языка ATL очень близка (но не тождественна) системе типов языка OCL. Для языка ATL на базе платформы Eclipse реализована интегрированная среда разработки, называемая ATL Development Tools (ADT). В качестве модели уровня M3 языком ATL поддерживается модель *Ecore* [9].

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. Правила 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.

Целью данной статьи является описание трансформации, преобразующей язык СИНТЕЗ в язык AMN. При этом абстрактный синтаксис языков СИНТЕЗ и AMN представляется в виде моделей уровня M2 (называемых *Synthesis* и *Amn* соответственно), конформных метамодели *Ecore.* Трансформация автоматически порождает схему (модель уровня M1), удовлетворяющую модели *Amn,* на основании схемы, удовлетворяющей модели *Synthesis*.

4 Трансформация канонической информационной модели в язык AMN

4.1 Спецификация на языке СИНТЕЗ

Рассмотрим пример части спецификации предметного посредника *University* в предметной области высшего образования. Синтаксически элементы языка СИНТЕЗ представляются *фреймами*, заключаемыми в фигурные скобки. Фреймы состоят из *слотов,* разделяемых точкой с запятой. Слот состоит из имени и списка значений. Имя слота отделяется от списка значений двоеточием, значения одного слота разделяются запятыми.

```
{ University; in: module;
 type:
 { Person; in: type; code: integer; },
 { Student; in: type; supertype: Person;
   grade: integer; },
 { ClassManager; in: type;
   enrolled: {set; type_of_element: Student;};
   tested: {set; type_of_element: Student;};
   classSize: integer;
   subset: { in: invariant;
     { all cm/ClassManager (cm.tested <= cm.enrolled &
       cardinal(cm.enrolled) <= cm.classSize) }
    };
   enroll: { in: function; params: {+st/Student};
    {{ cardinal(this.enrolled) < this.classSize &
       ^is_in(this.enrolled, st) &
       this.enrolled' = union(this.enrolled, {st})  }}
   };
   test: { in: function; params: {+st/Student};
    {{ is_in(this.enrolled, st) &
       ^is_in(this.tested, st) &
       this.tested' = union(this.tested, {st})  }}
   };
   leave: { in: function; params: {+st/Student};
    {{ is_in(this.enrolled, st) &
       this.enrolled' = differ(this.enrolled, {st}) &
       (is_in(this.tested, {st}) ->
       this.tested' = differ(this.tested, {st}) &
       (is_in(this.tested, {st}) ->
       this.tested' = this.tested)
    }}
   };
}; }
```

Основной единицей определения канонической модели является модуль [5] (*University*), включающий определения абстрактных типов данных (АТД). Пример включает три типа: *Person*, *Student* и *ClassManager.* Тип *Student* является подтипом типа *Person* (значение подтипа может использоваться всюду, где ожидается значение супертипа; подтип наследует элементы спецификации супертипа, например, атрибут *code*).

АТД описывают состояние и поведение своих экземпляров в терминах *атрибутов* и *методов* типа. Так, тип *ClassManager* (описывающий преподавателя, управляющего тестированием студентов), включает атрибуты *classSize* (максимальный размер класса тестируемых студентов), *enrolled* (множество студентов, находящихся в классе в данный момент) и *tested* (множество тех студентов, находящихся в классе, которые прошли тестирование). Метод *enroll* вызывается при входе нового студента в класс, метод *test* – при окончании студентом тестирования, метод *leave* – при выходе студента из класса. Спецификации методов представляют собой формулы многосортного объектного исчисления – типизированного варианта логики предикатов первого порядка. Формулы задают смешанные пред- и постусловия методов.

В формулах ключевое слово *this* указывает на текущий экземпляр типа; функция *cardinal* обозначает мощность множества; предикат *is_in* – принадлежность элемента множеству; функции *union* и *differ* – объединение и разность множеств; предикат <= – отношение подмножества на множествах и отношение «меньше либо равно» на числах; логические связки &, ^, -> – конъюнкцию, отрицание и импликацию; *all* – квантор существования; / – типизацию переменной некоторым типом; термы, отмеченные апострофом, ссылаются на состояние экземпляра типа после выполнения метода, т. е. обозначают *побочные эффекты* метода.

Так, предусловиями метода *enroll* является проверка того, что текущее количество студентов в классе меньше максимально допустимого (*cardinal(this.enrolled) < this.classSize*) и что студент, желающий войти в класс, еще не входил в класс (*^is_in(this.enrolled, st)*). Если предусловие выполняется, студент считается вошедшим в класс, т. е. помещается в множество *enrolled* (*this.enrolled'= union(this.enrolled, {st})*).

В определения типов могут быть включены *инварианты* типов – ограничения, выраженные в виде замкнутой логической формулы. Например, инвариант *subset* утверждает, что каждый тестируемый студент является вошедшим в класс (*tested <= enrolled*) и что количество вошедших не может быть больше максимального (*cardinal(enrolled) <= classSize*).

4.2 Формализация абстрактного синтаксиса языка СИНТЕЗ

Рассмотрим теперь, каким образом абстрактный синтаксис языка СИНТЕЗ представляется в виде модели *Synthesis* уровня М2, конформной метамодели *Ecore* [9]. Элементами модели, конформной *Ecore*, могут быть классы (*EClass*), атрибуты (*EAttribute*), ссылки (*EReference*). Например, синтаксис АТД формализуется следующим классом *ADTDef* метамодели *Ecore*:

```
<eClassifiers xsi:type="ecore:EClass" name="ADTDef"
  eSuperTypes="#//TypeDef">
  <eStructuralFeatures xsi:type="ecore:EReference"
    name="attributes" upperBound="-1"
    eType="#//AttributeDef"
    eOpposite="#//AttributeDef/attributeOf"/>
```

```
  <eStructuralFeatures xsi:type="ecore:EReference"
    name="invariants" upperBound="-1"
    eType="#//InvariantDef"/>
  <eStructuralFeatures xsi:type="ecore:EReference"
   name="subtypes" lowerBound="1" upperBound="-1"
   eType="#//ADTDef" eOpposite="#//ADTDef/supertypes"/>
  <eStructuralFeatures xsi:type="ecore:EReference"
    name="supertypes" upperBound="-1" eType="#//ADTDef"
    eOpposite="#//ADTDef/subtypes"/>
</eClassifiers>
```

Из описания можно видеть, что класс *ADTDef* наследуется от класса *TypeDef* и обладает атрибутами (*attributes*), инвариантами (*invariants*), подтипами (*subtypes*) и супертипами (*supertypes*). Без «синтаксического сахара» XML это описание может быть представлено в следующем виде:

```
ADTDef(attributes: AttributeDef*, invariants: InvariantDef*,
  subtypes: ADTDef*, supertypes: ADTDef*): TypeDef
```

Знак * здесь означает кардинальность свойства: например, АТД может обладать несколькими атрибутами или инвариантами. Целиком фрагмент модели *Synthesis*, необходимый для представления спецификации *University* (раздел 4.1), выглядит следующим образом:

```
ValueDef(typeOfValue: TypeDef)
ElementDef(name: String): ValueDef
FrameDef(parameters: ParameterDef*): ElementDef
ParameterDef(parameterKind: String,
  parameterOf: FrameDef, type: TypeDef): ElementDef
TypeDef(typeInModule: ModuleDef): FrameDef
ModuleDef(containedTypes: TypeDef*): FrameDef
ADTDef(attributes: AttributeDef*, invariants: InvariantDef*,
  subtypes: ADTDef*, supertypes: ADTDef*): TypeDef
AttributeDef(attributeOf: ADTDef, type: TypeDef):
  ElementDef
InvariantDef(predicativeSpec: Formula): TypeDef
FunctionDef(functionInModule: ModuleDef ,
  predicativeSpec: Formula): TypeDef
IntegerDef(unsigned: Boolean): TypeDef
SetDef(ofType: TypeDef): TypeDef
SetValueDef(values: ValueDef):ValueDef

Formula()
Variable(): ValueDef
QuantifiedFormula(variables: Variable*,
  formula: Formula): Formula
ExsistentiallyQuantifiedFormula(): QuantifiedFormula
UniversallyQuantifiedFormula(): QuantifiedFormula
Conjunction(formula: Formula*):Formula
Negation(formula: Formula): Formula
Implication(antecedent: Formula, consequent: Formula):
  Formula
CollectionComprehension(vars: Variable*,
  formula: Formula): ValueDef
Atom(symbol: String, terms: ValueDef): Formula
BuiltInPredicate(): Atom
ArithmeticPredicate(): Atom
SetPredicate(): Atom
FunctionCall(terms: ValueDef): ElementDef
BuiltInFunction(): FunctionCall
AccessFunction(): FunctionCall
SetFunction(): AccessFunction
GetFunction(): AccessFunction
```

Для того чтобы продемонстрировать, каким образом конкретные спецификации на языке СИНТЕЗ представляются в модели *Synthesis*, рассмотрим фрагмент спецификации *University*:

```
1   <xmi:XMI xmi:version="2.0">
    <syn:ModuleDef name="University"
2     containedTypes="/2 /3 /4"/>
    <syn:ADTDef name="Person" typeInModule="/0"
3     attributes="/6 "/>
    <syn:ADTDef name="Student"
      typeInModule="/0" attributes="/7"
4     supertypes="/1"/>
    <syn:ADTDef name="ClassManager"
5     typeInModule="/0" attributes="/8" />
6   <syn:IntegerDef/>
    <syn:AttributeDef name="code" attributeOf="/1"
7   type="/5"/>
    <syn:AttributeDef name="grade" attributeOf="/2"
8   type="/5"/>
    <syn:FunctionDef name="enroll" parameters="/9"
9   predicativeSpec="/10">
    <syn:ParameterDef name="st"
      parameterKind="input" parameterOf="/8"
10    type="3"/>
11  <syn-form:Conjunction formula="/11 /12 /13" />
    <syn-form:ArithmeticPredicate symbol="le"
14    terms="/14 /15" />
    <syn-form:BuiltInFunction name="cardinal"
16    terms="/16" />
    <syn-form:GetFunction name="enrolled"
17    terms="/17" />
    <syn-form:Variable name="this"/>
    </xmi:XMI>
```

Элементы спецификации по умолчанию нумеруются последовательными числами для удобства организации ссылок на них. Например, модулю *University* присвоен номер 1, и ссылка на него в других элементах спецификации осуществляется следующим образом: /1. Для удобства номера элементов спецификаций приведены слева (но они не являются частью спецификации).

4.3 Отображение спецификаций в язык AMN

Рассмотрим теперь, каким образом модуль *University* представляется в AMN.

```
MACHINE University_Context
SETS Obj; OID
ABSTRACT_CONSTANTS self,
ext_Person, ext_Student, ext_ ClassManager
PROPERTIES self: Obj >->> OID &
ext_Person: POW(Obj) & ext_Student: POW(Obj) &
ext_ClassManager: POW(Obj) &
ext_Student <: ext_Person
END

REFINEMENT Person
SEES University_Context
ABSTRACT_VARIABLES code
INVARIANT code: ext_Person --> INTEGER
END
```

```
REFINEMENT Student
SEES University_Context
EXTENDS Person
ABSTRACT_VARIABLES grade
INVARIANT grade: ext_Student --> INTEGER
END

REFINEMENT ClassManager
SEES University_Context
ABSTRACT_VARIABLES classSize, enrolled, tested
INVARIANT classSize: INTEGER &
enrolled: ext_ClassManager --> POW(ext_Student) &
tested: ext_ClassManager --> POW(ext_Student) &
!cm.( cm: ext_ClassManager =>
  tested(cm) <: enrolled(cm) &
  card(enrolled(cm) <= classSize(cm)) )
OPERATIONS
enroll(av, st) =
PRE av: ext_ClassManager & st: ext_Student
THEN
  ANY v WHERE card(enrolled(av)) < classSize(av) &
    not(st: enrolled(av)) & v = enrolled(av) \/ {st}
  THEN
    enrolled(av) := v
  END
END
END
```

Модуль представляется в AMN набором абстрактных машин, состоящим из контекстной машины (*University_Context*) и машин, соответствующих АТД модуля. Контекстная машина содержит информацию, характеризующую модуль как целое, машины типов содержат информацию, характерную для отдельных типов.

Представление АТД в AMN основано на экстенсиональном принципе: тип представляется константным множеством потенциальных экземпляров этого типа. Такое множество называется *экстенсионалом* типа (например, *ext_ClassManager*). Множество потенциальных экземпляров всех абстрактных типов моделируются константным множеством *AVAL*, так что для каждого типа *T* с экстенсионалом *ext_T* выполнено *ext_T: POW(AVAL)*. Здесь знак «:» означает принадлежность множеству, *POW* – множество подмножеств. Для каждой пары тип – подтип (например, *Person* и *Student*) выполнено отношение включения на экстенсионалах (*ext_Student <<: ext_Person*). Таким образом моделируется иерархия типов. Для обеспечения уникальной идентификации объектов вводится множество объектных идентификаторов *OID*. Биекция *self: Obj >->> OID* моделирует неявный атрибут *self* каждого типа. Упомянутые константы и их свойства определяются в контекстной машине. Структура спецификаций типов канонической модели также моделируется при помощи средств композиции абстрактных машин. Так, машина, соответствующая типу *Student,* расширяет (*EXTENDS*) машину, соответствующую супертипу *Person*.

Представление типа множеством своих потенциальных экземпляров позволяет естественным образом моделировать атрибуты типа функциями AMN: атрибут *enrolled* моделируется функцией *enrolled: ext_ClassManager --> POW(ext_Student)* в разделе переменных машины *ClassManager*.

Инварианты типа (например, *subset*) представляются конъюнктивными частями раздела *INVARIANT* соответствующей машины (знак ! в AMN обозначает квантор всеобщности, <: – отношение подмножества).

Методы типа (например, *enroll*) представляются операциями абстрактной машины (операции, соответствующие остальным методам, опущены). Первым входным параметром *av* операции является объект, для которого вызывается метод. В предусловиях *PRE* операции осуществляется типизация входных параметров.

Одной из основных особенностей представления предикативных спецификаций методов языка СИНТЕЗ в AMN является выделение помеченных апострофом термов, которые ссылаются на состояние экземпляра типа после выполнения метода (например, *this.enrolled'*). Для каждого из таких термов заводится новая переменная (*v*), вхождение терма в спецификацию метода заменяется на вхождение переменной. Сама спецификация метода представляется в AMN подстановкой неограниченного выбора *ANY*. Подстановка *ANY v WHERE P(v) THEN S END* устанавливает переменной *v* произвольное значение, удовлетворяющее предикату *P(v),* после чего выполняется вложенная подстановка *S.* Вложенной подстановкой представляются побочные эффекты метода.

4.4 Формализация абстрактного синтаксиса языка AMN

Рассмотрим, каким образом абстрактный синтаксис языка AMN представляется в виде модели *AMN* уровня M2, конформной метамодели *Ecore*. Как и модель *Synthesis* (раздел 4.2), модель *AMN* рассматривается в данном разделе в виде, лишенном «синтаксического сахара» XML. Целиком фрагмент модели *AMN*, необходимый для представления спецификации *University* (раздел 4.3), выглядит следующим образом:

```
Machine(sees: AbstractMachine*,
  extends: AbstractMachine*, sets: Set*,
  properties: Predicate, invariant: Predicate,
  operations: Operations*): Element
Refinement(abstractConstants: Element*,
  abstractVariables: Element*):Machine
Operation(inputParams: Variable*, outputParams: Variable*,
  Substitution: Substitution): Element
Element(name: String)
Set(): Element
Deferred(): Set

Predicate(stringRepr: String)
QuantifiedPredicate(variables: Variable*,
  predicate: Predicate): Predicate
```

UniversallyQuantified(): QuantifiedPredicate
ExistentiallyQuantified(): QuantifiedPredicate
AtomicPredicate(sign: String, expression: Expression*): Predicate
Conjunction(predicate: Predicate*): Predicate
Negation(predicate: Predicate): Predicate
Disjunction(predicate: Predicate*): Predicate
Implication(antecedent: Predicate, consequent: Predicate): Predicate
Equivalence(predicate: Predicate*): Predicate

Expression()
SetComprehension(variable: Variable*, predicate: Predicate): Expression
OperationalExpression(sign: String): Expression
UnaryOperator(expression: Expression): OperationalExpression
BiaryOperator(expression: Expression*): OperationalExpression
FunctionalExpression(expression: Expression*): OperationalExpression
Variable(name: String): Expression
NamedConstant (name: String): Expression
IntegerValue(value: Integer): Expression
IntegerValue(values: Expression): Expression

Substitution()
BecomesEqual(leftExpression: Expression*, rightExpression: Expression*): Substitution
Precondition(pre: Predicate, thenPart: Substitution): Substitution
Any(any: Variable*, where: Predicate, thenPart: Substitution):Substitution

4.5 Правила трансформации языка СИНТЕЗ в AMN

Трансформация, преобразующая спецификации языка СИНТЕЗ в спецификации языка AMN, представляет собой модуль языка ATL, получающий на вход модель уровня M1, конформную модели *Synthesis* и порождающую модель уровня M1, конформную модели *AMN*. Заголовок модуля выглядит следующим образом:

```
module Synthesis2AMN;
create OUT:AMN from IN:Synthesis;
```

Правило, порождающее контекстную машину *am* на основании модуля *m* языка СИНТЕЗ, выглядит следующим образом:

```
rule SynthesisModule2AMNAbstractMachine {
 from m: Synthesis!ModuleDef
 to
  am: AMN!AbstractMachine (
   name <- m.name + '_Context',
   sets <- Sequence{objSet, oidSet},
   abstractConstants <-
    Sequence{objConst, selfConst},
   properties <- propertiesPredicate
  ),
  objSet: AMN!Deferred(name <- 'Obj'),
  oidSet: AMN!Deferred(name <- 'OID'),
  selfConst: AMN!NamedConstant(name<-'self'),
  propertiesPredicate : AMN!Conjunction (
   predicate <- Sequence{selfProp}
  ),
  selfProp: AMN!AtomicPredicate(
   sign <- ':', expression <- selfConst,
   expression <- selfType),
  selfType: AMN!BinaryOperator (
   sign <- '>->>', expression <- objConst,
   expression <- oidSet
  )
}
```

Контекстной машине присваивается имя, добавляются константные множества *Obj* и *OID* и предикат *propertiesPredicate*, описывающий их свойства (см. раздел 4.3). В секции *from* правила описывается имя (*m*) и тип (*Synthesis!ModuleDef*) входного элемента, в секции *to* – имена (*am, objSet, oidSet, selfConst, selfProp, selfType*) и типы выходных элементов. Для каждого выходного элемента устанавливаются его свойства. Так, например, контекстной машине присваивается имя: *name <- m.name + '_Context'*.

При помощи отдельного правила в контекстную машину добавляются константы и предикаты, отражающие экстенсиональный принцип представления АТД в AMN:

```
rule ADT2ContextMachineProperties{
 from adt : Synthesis!ADTDef
 to
  ext: AMN!NamedConstant(
   name <- 'ext_' + adt.name
  ),
  pred: AMN!AtomicPredicate (
   sign <- ':', expression <- ext,
   expression <- extType
  ),
  extType: AMN!FunctionalExpression(
   sign <- 'POW',
   expression <- thisModule.resolveTemp(
    adt.typeInModule,'objConst')
  ),
 do {
  thisModule.resolveTemp(adt.typeInModule,
   'am').abstractConstants <- ext;
  thisModule.resolveTemp(adt.typeInModule,
   'propertiesPredicate').predicate <- pred;
}}
```

При помощи этого правила, например, в машину *University_Context* добавляются константа *ext_Person* и предикат ее типизации *ext_Person: POW(Obj)*. Можно заметить, что в правиле используется секция императивного кода *do* (выполняющегося после создания элементов, указанных в секции *to*). Специальная функция *resolveTemp* применяется для того, чтобы получить доступ к элементам, созданным другими правилами. Так, выражение *thisModule.resolveTemp(adt.typeInModule, 'am')* возвращает контекстную машину, созданную на основе модуля *adt.typeInModule* языка СИНТЕЗ (которому принадлежит АТД *adt*) правилом *SynthesisModule2AMNAbstractMachine*.

Правило, порождающее отдельную машину для каждого АТД, выглядит следующим образом:

```
rule ADT2Machine {
 from adt : Synthesis!ADTDef
 to
  am: AMN!AbstractMachine (
   name <- adt.name,
   invariant <- predicate,
   sees <- thisModule.resolveTemp(
    adt.typeInModule, 'am')
   ),
  predicate: AMN!Conjunction(),
 do {
  if(adt.subtypes->notEmpty()){
   for(e in adt.subtypes)
    thisModule.genStrictInclusion(adt,e);
  }
  if(adt.supertypes->notEmpty()){
   for(e in adt.supertypes)
    am.extendsClause <-
     thisModule.resolveTemp(e, 'am');
  }
}
```

Правило добавляет контекстную машину в секцию *SEES* машины типа; добавляет машины, соответствующие супертипам в секцию *EXTENDS,* а также для каждого подтипа вызывает правило *GenStrictInclusion,* добавляющее в контекстную машину предикаты, моделирующие иерархию типов (например, *ext_Student <: ext_Person*):

```
rule GenStrictInclusion(
  adt : Synthesis!ADTDef,
  subType: Synthesis!ADTDef){
 to
  inclusion: AMN!AtomicPredicate(
   sign <- '<:',
   expression <- Sequence{
    thisModule.resolveTemp(subType,'ext'),
    thisModule.resolveTemp(adt, 'ext')
   }
  )
 do{
  thisModule.resolveTemp(
   adt.typeInModule,'propertiesPredicate').
    predicate <- inclusion;
}}
```

Правило, порождающее на основании спецификации атрибута АТД соответствующие конструкции в машине типа, выглядит следующим образом:

```
rule Attribute2MachineSections {
from at: Synthesis!AttributeDef(
          at.isNotFunctional())
to
 attrPred : AMN!AtomicPredicate (
  sign <- ':', expression <- attrVar,
  expression <- totalFunction
 ),
 totalFunction : AMN!BinaryOperator (
  sign <- '-->',
  expression <- thisModule.resolveTemp(
                 at.attributeOf, 'ext')
 ),
 attrVar: AMN!Variable(name <- at.name),
 do{
  if(at.type.oclIsTypeOf(
     Synthesis!IntegerDef))
   thisModule.AttributeTypeStringExp(at,
   'NAT');
  if(at.type.oclIsTypeOf(Synthesis!SetDef))
   thisModule.AttributeTypeSetDef(at);
  thisModule.resolveTemp(at.attributeOf,
   'am').abstractVariables <- attrVar;
  thisModule.resolveTemp(at.attributeOf,
   'predicate').predicate <- attrPred;
}}
```

Правило создает переменную машины, соответствующую атрибуту (например, *enrolled*) и предикат типизации этой переменной (например, *enrolled: ext_ClassManager --> POW(ext_Student)*). При этом для отображения типа атрибута используются специальные правила *AttributeTypeStringExp* (для простых типов, например, *integer*) и *AttributeTypeSetDef* для типов множества:

```
rule AttributeTypeStringExp(
  attr: Synthesis!AttributeDef,
  stringVal: String){
 to
  funcExp: AMN!StringValue(
   value <- stringVal
  )
 do {
  thisModule.resolveTemp(attr,
  'totalFunction').expression <- funcExp;
}}
rule attributeTypeSetDef(
  attr: Synthesis!AttributeDef){
 to
  funcExp : AMN!FunctionalExpression(
   sign <- 'POW',
   expression <- thisModule.resolveTemp(
    attr.type.ofType, 'ext')
  )
 do{
  thisModule.resolveTemp(attr,     'totalFunc-
tion').expression <- funcExp;
}}
```

Заметим, что правило *Attribute2MachineSections* применяется только для атрибутов, тип которых не есть тип функции. Данное предусловие проверяется при помощи вспомогательного правила *IsNotFunctional*:

```
helper context Synthesis!AttributeDef def:
  IsNotFunctional(): Boolean =
 if self.oclIsTypeOf(Synthesis!AttributeDef)
 then
  if not self.type.oclIsTypeOf(
         Synthesis!FunctionDef)
  then true
  else false
  endif
 else false
 endif;
```

Функциональные атрибуты отображаются в операции машин при помощи следующего правила:

```
rule Function2Operation{
 from at: Synthesis!AttributeDef(
          at.isFunctional())
```

```
 to
  operation: AMN!Operation(
   name <- at.name,
   inputParams <- objParam,
   substitution <- subst
  ),
  objParam: AMN!Variable(name <- 'av'),
  subst: AMN!Precondition(
   pre <- precondPredicate,
   thenPart <- thenSubstAny
  ),
  precondPredicate: AMN!Conjunction(
   predicate <- objPred
  ),
  objPred: AMN!AtomicPredicate (
   sign <- ':',
   expression <- thisModule.resolveTemp(
                 at.attributeOf, 'ext'),
  expression <- objParam
  ),
  thenSubstAny: AMN!Any(
   where <- whereConj,
   thenPart <- sequenceSubstitution
  ),
  whereConj : AMN!Conjunction (
   predicate <- at.type.predicativeSpec
  ),
  sequenceSubstitution : AMN!SequenceSubst()
 do{
  for(e in at.type.parameters)
   if(e.type.oclIsTypeOf(Synthesis!ADTDef))
    thisModule.generateParameters(e, at);
  thisModule.resolveTemp(at.attributeOf,
   'am').operations <- operation;
}}
```

Правило создает операцию машины, соответствующую функциональному атрибуту (например, *enroll*), присваивает ей имя, создает первый параметр *av* операции его предикат типизации (например, *av: ext_ClassManager*), создает подстановки, составляющие тело операции. Дополнительное правило *GenerateParameters* создает остальные параметры операции и их предикаты типизации (например, *st: ext_Student*):

```
rule GenerateParameters(
  param: Synthesis!ParameterDef,
  attr: Synthesis!AttributeDef){
 to
  var: AMN!Variable(name <- param.name),
  varPred: AMN!AtomicPredicate(
   sign <- ':', expression <- var,
   expression <- thisModule.resolveTemp(
                 param.type, 'ext')
  )
  do{
   if(param.parameterKind = 'input')
    thisModule.resolveTemp(attr,
     'operation').inputParams <- var;
   thisModule.resolveTemp(attr,
    'precondPredicate').predicate <- varPred;
}}
```

Каждая из конструкций, составляющих формулы языка СИНТЕЗ, преобразуется в формулы AMN при помощи отдельного правила. Так, существуют правила для преобразования встроенных предикатов (например, *is_in*):

```
rule BuiltInPredicate2AtomicPredicate{
 from builtIn : Synthesis!BuiltInPredicate
 to
  atomPred: AMN!AtomicPredicate(
   expression <- builtIn.terms
  )
 do{
  if(builtIn.symbol = 'is_in')
   atomPred.sign <- ':';
}
```

переменных:

```
rule Variable{
 from synthVar: Synthesis!Variable
 to
  amnVar: AMN!Variable(name<-synthVar.name)
}
```

логических связок (например, конъюнкции):

```
rule Conjunction{
 from conjSynth: Synthesis!Conjunction
 to
  conjAMN: AMN!Conjunction(
   predicate <- conjSynth.formula
  )
}
```

арифметических предикатов:

```
rule ArithmeticPredicate2BinaryOperator{
 from
  arithmPred: Synthesis!ArithmeticPredicate
 to
  atomPred: AMN!BinaryOperator(
   sign <- arithmPred.symbol,
   expression <- arithmPred.terms
  )
}
```

встроенных функций (например, *union*):

```
rule BuiltInFunction2FunctionalExpression{
 from builtIn: Synthesis!BuiltInFunction
 do{
  if(builtIn.name = 'union')
   thisModule.genUnion(builtIn);
}
rule genUnion(
   builtIn: Synthesis!BuiltInFunction){
 to
  funcExp: AMN!BinaryOperator(
   sign <- '\\/', expression <- builtIn.terms
  )
}
```

значений (например, целочисленных):

```
rule Term2AMNIntegerValue{
 from term: Synthesis!IntegerValueDef
 to
  integerVal: AMN!IntegerValue (
   value <- term.value )
}
```

и других видов выражений.

Ввиду ограничения размера статьи, набор правил, приведенный в данном разделе, не образует полную трансформацию (в большинстве случаев даже описание самих правил не является полным). Однако приведенные правила позволяет продемонстрировать основные принципы построения трансформации.

5 Заключение

Одним из способов обеспечения семантической интероперабельности неоднородных информационных ресурсов при создании интегрированных информационных систем является использование методов и средств формализации уточнения. Использование ресурса для реализации части интегрированной системы является правомочным, если спецификация ресурса уточняет часть спецификации системы.

Перспективной технологией, использующей уточнение для верификации интеграции неоднородных информационных ресурсов, является технология предметных посредников. В статье рассматривается трансформация канонической информационной модели посредников (языка СИНТЕЗ) в формальный язык спецификаций AMN, поддерживающий уточнение. Таким образом, канонической модели сообщается формальная семантика, и достигается возможность доказательства уточнения спецификаций канонической модели при помощи существующих автоматических и интерактивных систем доказательства.

Литература

[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 Int. Conf. on Enterprise Information Systems (ICEIS). – 2007. – P. 246-251.

[2] Abrial J.-R. The B-book: assigning programs to meanings. – Cambridge: Cambridge University Press, 1996.

[3] Захаров В.Н., Калиниченко Л.А. Ступников С.А. Конструирование канонических информационных моделей для интегрированных информационных систем // Информатика и её применения. – 2007. – Т. 1, Вып. 2. – С. 15-38.

[4] Скворцов Н.А. Применение уточнения понятий в решении задач манипулирования онтологиями // Труды Девятой Всерос. науч. конф. «Электронные библиотеки: перспективные методы и технологии, электронные коллекции», RCDL' 2007. – Университет города Переславля: Переславль-Залесский, 2007. – С. 225-229.

[5] 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.

[6] Atelier B. The industrial tool to efficiently deploy the B method. – http://www.atelierb.eu/index-en.php.

[7] ATL Project. – http://www.eclipse.org/m2m/atl/.

[8] Meta Object Facility (MOF) 2.0 Core Specification. – http://www.omg.org/cgi-bin/doc?formal/06-01-01.pdf, 2006.

[9] Budinsky F., Steinberg D., Ellersick R., Grose T. Eclipse modeling framework, chapter 5 ”Ecore modeling concepts”. – Addison Wesley Professional, 2004.

[10] Candela L., Castelli D., Ferro N., Ioannidis Y. et al. The DELOS digital library reference model – foundations for digital libraries. Version 0.98. – http://www.delos.info/files/pdf/ReferenceModel/DELOS_DLReferenceModel_0.98.pdf, 2008.

[11] Ступников С.А. Отображение спецификаций, выраженных средствами ядра канонической модели, в язык AMN // 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. 69-95.

A semantic transformation of the canonical information model into a formal specification langage for the refinement verification

S.A. Stupnikov

The paper is devoted to the development of methods and tools achieving the semantic interoperability of heterogeneous information resources during the development of integrated information systems. A *refinement* of a system specification by the resources specifications is considered as a basic fact requiring a formal proof. A transformation of the SYNTHESIS language aimed at development of subject mediators for tasks solving over heterogeneous resources into AMN formal specification language supporting the refinement. Thus a possibility of refinement proving in the SYNTHESIS language with the help of the existing automatic/interactive proves is achieved.

♣ Работа выполнена при финансовой поддержке РФФИ (проекты 08-07-00157, 10-07-00342, 10-07-00640) и Программы фундаментальных исследований Президиума РАН № 3 – Фундаментальные проблемы системного программирования, раздел 4 – Системы управления базами данных, проект – Исследование методов и средств промежуточного слоя предметных посредников, обеспечивающего решение задач над множеством неоднородных распределенных информационных ресурсов