**Московский Государственный Университет им. М.В.Ломоносова**

**Факультет Вычислительной Математики и Кибернетики**

**Кафедра Автоматизация Систем Вычислительных Комплексов**



Дипломная работа

**Семантическое отображение канонической информационной модели СИНТЕЗ в формальный язык спецификаций AMN**

студента 522 группы

Сергеева Алексея Юрьевича

Научный руководитель:

с.н.с. ИПИ РАН, м.н.с. ЛОИТ ВМиК МГУ.,

к.т.н. Ступников Сергей Александрович

**Москва 2010**

# Аннотация

В настоящее время в различных областях информационных технологий наблюдается увеличения числа разнородных информационных ресурсов, представленных в различных моделях. Необходимость интеграции ресурсов и их повторного использования становится все более актуальна. Возможными методами к решению данной проблемы являются технологии композиционного проектирования ИС, регистрации ресурсов в предметных посредниках, интеграции моделей ресурсов. Важным этапом в упомянутых методах является этап доказательства уточнения.

Созданы инструментальные средства, которые позволяют почти автоматически доказывать факт уточнения одной спецификации другой. Для того чтобы воспользоваться этими средствами необходимо отобразить каноническую модель в модель, поддерживаемую данными инструментариями.

Целью данной работы является разработка трансформации канонической информационной модели СИНТЕЗ в формальный язык спецификаций AMN, на языке трансформации моделей ATL.

Оглавление

[Аннотация 2](#_Toc262474485)

[1. Введение 5](#_Toc262474486)

[2. Постановка задачи 9](#_Toc262474487)

[3. Язык СИНТЕЗ 10](#_Toc262474488)

[4. Построение модели языка AMN 12](#_Toc262474489)

[4.1. Компоненты языка AMN 13](#_Toc262474490)

[4.2. Подстановки 15](#_Toc262474491)

[4.2.1. Подстановка присваивания 15](#_Toc262474492)

[4.2.2. Подстановка с предусловием 16](#_Toc262474493)

[4.2.3. Подстановка неограниченного выбора 16](#_Toc262474494)

[4.2.4. Последовательная подстановка 16](#_Toc262474495)

[4.2.5. Подстановка вызова операции 17](#_Toc262474496)

[4.2.6. Параллельная подстановка 17](#_Toc262474497)

[4.3. Выражения 17](#_Toc262474498)

[4.3.1. Бинарное соответствие 18](#_Toc262474499)

[4.3.2. Пересечение множеств 18](#_Toc262474500)

[4.3.3. Переписывание отношений 18](#_Toc262474501)

[4.3.4. Частичная функция 18](#_Toc262474502)

[4.3.5. Полная функция 18](#_Toc262474503)

[5. Построение отображений моделей на основе соответствий между элементами моделей 20](#_Toc262474504)

[5.1. Экстенсиональная интерпретация АТД 20](#_Toc262474505)

[5.2. Моделирование структуры спецификации типов при помощи средств композиции абстрактных машин 22](#_Toc262474506)

[5.3. Контекстная машина 23](#_Toc262474507)

[5.4. Структура абстрактной машины, соответствующей типу: моделирование атрибутов, методов, инвариантов 25](#_Toc262474508)

[5.5. Моделирование формул канонической модели 29](#_Toc262474509)

[5.5.1. Атомарные предикаты 29](#_Toc262474510)

[5.5.2. Условия 30](#_Toc262474511)

[5.5.3. Составные формулы 31](#_Toc262474512)

[6. Реализация трансформации модели СИТНЕЗ в модель AMN на языке ATL 34](#_Toc262474513)

[6.1. Построение заголовка трансформации 37](#_Toc262474514)

[6.2. Построение контекстной машины модуля 37](#_Toc262474515)

[6.3. Построение абстрактных машин АТД модуля и дополнение контекстной машины 39](#_Toc262474516)

[6.3.1. Дополнение контекстной машины 39](#_Toc262474517)

[6.3.2. Генерация абстрактной машины для АТД 41](#_Toc262474518)

[6.3.3. Генерация секций машины АТД в соответствии с его нефункциональными атрибутами 42](#_Toc262474519)

[6.3.4. Генерация операций функциональных атрибутов 44](#_Toc262474520)

[6.4. Построение отображения формул и обработка функций доступа 47](#_Toc262474521)

[Заключение 50](#_Toc262474522)

[Список литературы 51](#_Toc262474523)

# Введение

В настоящее время в различных областях деятельности людей и науки в частности наблюдается экспоненциальный рост накапливаемых, экспериментальных данных, описанных с помощью всевозможных информационных моделей. Таким образом нынешний период развития информационных технологий сопровождается созданием множества различных информационных моделей. Наряду с созданием моделей для распределенных систем (таких как архитектуры OMG, архитектуры Семантического Веба, архитектуры цифровых библиотек, архитектуры информационных грид систем), можно выделить разработку в области конкретных информационных моделей - моделей данных (таких как SQL, UML, XML), моделей потоков работ (MQSeries, SAP/R3, Eastman), семантических моделей (включая онтологические модели и модели метаданных). В основе почти всех из них находятся разнообразные понятия и парадигмы, зачастую несовместимые между собой. Также можно выделить другую тенденцию, заключающуюся в накапливании основанных на данных моделях информационных ресурсах, количество которых непрерывно растет. На этом фоне возникает необходимость в использовании и интеграции компонентов и сервисов, представленных в различных моделях и приложениях, также как и их повторное использование для создания новых информационных систем. Можно заметить противоречивость такого рода тенденций, так как чем больше появляется всевозможных моделей наряду с уже использующимися, тем труднее становится задача их интеграции между собой. Данная проблема вынуждает разработчиков новых систем создавать новые ресурсы, вместо грамотного повторного использования уже существующих.

В течение последних лет в лаборатории композиционных методов проектирования ИС ИПИ РАН проводились интенсивные исследования по разработке методов решения задач над неоднородными информационными ресурсами, таких как композиционное проектирование ИС [2], регистрация ресурсов в предметных посредниках [3], интеграция моделей ресурсов [4].

Представленные выше методы нашли свое отражение в ряде проектов лаборатории. В частности, была разработана многоуровневая инфраструктура, включающая в себя уровень информационных ресурсов, промежуточный слой, обеспечивающий интероперабельность ресурсов благодаря технической унификации их интерфейсов и введению дистанционных механизмов обращения к ресурсам (адаптеры, осуществляющие преобразование запросов, выраженных в канонической информационной модели посредников, в их представление в информационной модели ресурса, включаются в состав промежуточного слоя), уровень *предметных посредников*, каждый из которых создает спецификацию предметной области для решения некоторого класса задач, используя каноническую информационную модель для представления семантики предметной области и унифицированного отображения разнообразных видов информационных моделей ресурсов (моделей данных, сервисных моделей, онтологических моделей, процессных моделей), уровень задач (приложений), формулируемых в терминах одного или нескольких посредников. Следует заметить, что посредники, в свою очередь, могут рассматриваться как информационные ресурсы для использования в посредниках более высокого уровня. В такой инфраструктуре предметные посредники играют ключевую роль для решения семантических проблем, перечисленных выше. В частности, при интеграции неоднородных ресурсов в посреднике нужно уметь семантически отождествлять объекты, представленные в различных информационных моделях, и семантически правильно отображать схемы интегрируемых ресурсов в схему посредника. Поскольку в общем случае ресурсы неоднородны (представлены в различных моделях), при интеграции неоднородных ресурсов для однородного представления их семантики требуется приведение различных информационных моделей к унифицированному виду в рамках некоторой унифицирующей информационной модели, которая называется *канонической*. Для унификации разнородных спецификаций прежде всего требуется умение сопоставлять спецификации различных ресурсов друг с другом так, чтобы можно было отвечать на вопрос, можно ли при реализации посредника использовать спецификацию существующего ресурса вместо фрагмента спецификации посредника. Для этого достаточно доказать, что рассматриваемые спецификации находятся в отношении уточнения. Говорят, что спецификация A уточняет спецификацию D, если A можно использовать вместо D так, что пользователь D не будет замечать этой замены. Средства доказательства факта уточнения спецификации некоторого компонента спецификацией другого компонента (реализуемые на основе теоретико-модельных нотаций и соответствующего инструментария [6,7]) составляют фундамент применяемых методов конструирования унифицирующих (канонических) моделей представления информации в посредниках. Каноническая информационная модель служит в качестве общего языка для адекватного выражения семантики разнородных моделей представления информации, используемых в разнообразных информационных ресурсах, релевантных посреднику. Методы отображения информационных моделей и синтеза расширяемых канонических информационных моделей для слоя предметных посредников подробно рассмотрены в [7]. Там же предложена архитектура Унификатора информационных моделей. Задачей Унификатора моделей является унификация множества информационных моделей(называемых *исходными*), использующихся в различных информационных ресурсах некоторой информационной системы. Унификация исходной модели R есть приведение ее к канонической информационной модели C, т. е. создание такого расширения E канонической модели (которое может быть и пустым) 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. Синтез канонической модели заключается в построении указанных расширений ядра канонической модели для всех исходных моделей и объединении таких расширений. В качестве ядра канонической информационной модели применяется язык СИНТЕЗ [1], который ориентирован на спецификацию предметных посредников и синтез канонических моделей. Спецификация предметного посредника для класса задач включает определения понятий предметной области, выражаемых соответствующими онтологическими спецификациями, спецификации классов объектов предметной области, спецификации типов экземпляров названных классов и их методов, определяющих их поведение, спецификации процессов решения задач данного класса как совмещенных во времени последовательностей действий, реализуемых методами классов, сервисами и другими процессами. Предполагается, что такие спецификации преобразуются в спецификации канонической модели, имеющей формальную семантику.

Регистрация релевантных посреднику ресурсов рассматривается как задача композиционного проектирования систем [2, 3]. Регистрация ресурсов есть процесс целенаправленной трансформации спецификаций, включающий декомпозицию спецификаций посредника на непротиворечивые фрагменты, поиск среди спецификаций релевантных ресурсов подходящих типов данных — кандидатов для уточнения ими спецификаций типов посредника, построение выражений, определяющих классы ресурсов в виде композиции классов посредника. Для подобного манипулирования спецификациями разработано специальное *исчисление спецификаций* [4].

Принципиальным моментом в этой схеме является реализация доказательства уточнения фрагментов спецификаций посредника спецификациями ресурсов в процессе отображения таких спецификаций.

# Постановка задачи

Таким образом, можно заметить, что необходимым требованием к разработанным методам и средствам является обеспечение возможности автоматизированного доказательства факта уточнения спецификации требований спецификацией существующих компонентов. Для таких манипуляций спецификациями требуется выражение их семантики в некотором формальном языке, позволяющем осуществлять доказательство непротиворечивости уточнения спецификаций. В качестве такого языка выбрана Нотация Абстрактных Машин (Abstract Machine Notation, AMN [6]), которая позволяет осуществлять необходимые рассуждения относительно спецификаций. Для AMN разработаны специальные инструментальные средства, составляющие так называемую В-технологию [7]. Для того, чтобы можно было манипулировать спецификациями информационных моделей в рамках В-технологии (в частности, доказывать корректность отношения уточнения между спецификациями, выраженными в различных моделях), необходимо корректно отобразить такие модели в язык AMN с сохранением семантики. Метод такого отображения создан [16] и существует возможность применять его на примере ядра канонической информационной модели (языка СИНТЕЗ).

В настоящее время в Лаборатории композиционных методов проектирования информационных систем разрабатываются прототипы инструментальные средства для регистрации ресурсов в предметных посредниках, а также средства унификации информационных моделей. В качестве объединяющей платформы для этих средств выбран Eclipse [17]. Задачей данной дипломной работы является программная реализация семантического отображения канонической информационной модели СИНТЕЗ в формальный язык AMN, с использованием языка трансформации моделей ATL (являющегося частью Eclipse Modeling Project). Компонент, реализующий отображение, рассматривается как неотъемлемая часть архитектуры упомянутых средств интеграции неоднородных ресурсов.

## Язык СИНТЕЗ

В данной работе в качестве ядра канонической модели был выбран язык СИНТЕЗ [1], разработанный в Лаборатории композиционных методов проектирования информационных систем Института проблем информатики РАН. Он ориентирован на семантическую интероперабельность и композиционное проектирование информационных систем в широком диапазоне существующих неоднородных информационных компонентов. Метаязыком, при помощи которого в ядре канонической информационной модели представляются средства определения информационных источников, является язык фреймов. Фрейм - это структурированная символическая модель некоторого понятия. Фрейм можно рассматривать как множество атрибутов, называемых слотами, каждый из которых используется для представления свойства понятия. Каждый слот может иметь множество значений. Единицей определения канонической модели является модуль. Каждый модуль задает обобщенное представление информационных источников, либо является модулем спецификации предметной области или концептуального проекта информационной системы. Каноническая модель содержит унифицированную систему типов, включающую универсальный конструктор типов (Абстрактный Тип Данных, АТД), а также представительный набор встроенных типов. АТД по умолчанию является объектным, хотя он также может быть объявлен и как необъектный. Описание абстрактного типа данных инкапсулирует спецификации атрибутов, методов и инвариантов типа. Методы и инварианты описываются встроенным типом функции. Типы языка организованы в иерархию обобщения, базирующуюся на отношении тип-подтип. Значение подтипа может использоваться всюду, где ожидается значение супертипа; подтип наследует элементы спецификации супертипа; допускается множественное наследование спецификаций подтипом. Отношение тип-подтип основывается на понятии уточнения. Спецификация типа функции включает описание параметров функции и предикативную спецификацию функции. Для задания предикативных спецификаций функций в канонической модели используется типизированный вариант логики предикатов первого порядка. Предикативная спецификация позволяет задавать смешанные и постусловия, определяющие действия, реализуемые функцией. Для обращения к информационным источникам в языке предусмотрены формулы специального вида - правила. В языке правил не разрешается использование конструкций, связанных с изменением состояния системы, однако допускается использование широкого спектра операций композиции источников, характерных для языков запросов: например, соединение, пересечение, объединение.

Однородные совокупности объектов предметной области (информационные источники) представляются в канонической модели коллекциями. Экземплярами коллекций являются значения АТД. В случае если экземпляр типа объектный, коллекция называется классом. Явно разграничивая объявление типа и объявление коллекции, язык СИНТЕЗ подчеркивает роль коллекции как представителя множества объектов и роль типа как спецификации структуры и поведения объектов, связанных с коллекцией.

# Построение модели языка AMN

Формальная семантика необходима для проведения доказательных рассуждений о моделях информационных ресурсов. Имея формальную семантику можно строго формулировать утверждения о непротиворечивости и уточнении спецификаций, выраженных в модели. Появляется возможность доказательства корректности отображений одних моделей информационных ресурсов в другие модели. Для таких манипуляций информационными моделями и выраженными в них спецификациями требуется выражение их семантики в некотором формальном языке, позволяющем осуществлять доказательство непротиворечивости и уточнения спецификаций. Таким языком может являться теоретико-модельный язык Нотация Абстрактных Машин (Abstract Machine Notation, AMN [6]), основанный на теории множеств и логике предикатов первого порядка, который позволяет осуществлять необходимые рассуждения относительно спецификаций. Язык AMN обеспечивает манипулирование теоретико-множественными спецификациями в логике первого порядка и дает возможность доказать правильность уточнения спецификаций. Для AMN разработаны специальные инструментальные средства, составляющие в совокупности так называемую В-технологию[7]. Язык AMN как теоретико-множественная нотация позволяет рассматривать состояния и операции системы интегрировано как спецификацию пространства состояний и поведения (определенного операциями на состояниях) абстрактных машин. Спецификация состояния абстрактной машины вводится переменными состояния вместе с инвариантами - ограничениями, которые должны всегда удовлетворяться. Операции определяются на основе расширения формализма охраняемых команд Дейкстры. Ключевым понятием AMN является уточнение, позволяющее соотносить спецификации систем различных уровней абстракции. Уточняющая спецификация может быть гораздо более детальной, чем уточняемая спецификация. Конструируется уточняющая спецификация на основе алгоритмического уточнения и уточнения данных. Уточнение формализуется в AMN путем формулировки ряда теорем специального вида, так называемых *proof obligations*. Такие теоремы формулируются автоматически при помощи инструментальных средств поддержки В-технологии на основании склеивающих инвариантов, соотносящих состояния уточняемой и уточняющей системы.

Одной из задач данной работы является построение модели языка AMN. Согласно условиям трансформации моделей данная модель должна быть конформна модели *Ecore* и быть уровня M2 согласно стандарту MOF.

Далее дается краткое описание языка в соответствии с документом

## Компоненты языка AMN

Основной синтаксической конструкцией языка являются компоненты. Существуют три вида компонентов: абстрактная машина, уточнение и реализация. Рассмотрим основной для данной работы компонент языка - абстрактную машину. Она включает в себя разделы определения данных, их свойств, а также операций. Порядок разделов может быть произвольным. Ниже дается краткое описание разделов абстрактной машины.

Таблица 1 – Разделы абстрактной машины

|  |  |
| --- | --- |
| CONSTRAINTS | Определение типа и свойств формальных скалярных параметров |
| SEES | Список экземпляров видимых машин |
| INCLUDES | Список экземпляров включенных машин |
| PROMOTES | Список используемых операций включенных абстрактных машин |
| EXTENDS | Список экземпляров расширяемых машин |
| USES | Список экземпляров используемых машин |
| SETS | Абстрактные множества, либо определение перечислимых множеств |
| CONCRETE\_CONSTANTS | Список конкретных констант |
| ABSTRACT\_CONSTANTS | Список абстрактных констант |
| PROPERTIES | Тип и свойства машинных констант |
| CONCRETE\_VARIABLES | Список конкретных переменных |
| ABSTRACT\_VARIABLES | Список абстрактных переменных |
| INVARIANT | Тип и свойства переменных |
| ASSERTIONS | Определение свойств, выводимых из инварианта |
| INITIALISATION | Инициализация переменных |
| OPERATIONS | Список и определение операций |

Разделы IMPORTS, SEES, INCLUDES, EXTENDS, или USES являются секциями видимости. С их помощью строятся связи между различными компонентами системы.

Составляющими элементами разделов PROPERTIES, INVARIANT, ASSERTIONS являются предикаты. В языке AMN предикат является формулой, которая может быть доказана либо опровергнута. Предикаты используются для выражения свойств данных или для задания условий применения подстановок.

Все многообразие предикатов языка может быть разбито на два класса:

* атомарный предикат
* составной предикат

Атомарный предикат состоит из выражений, объединенных каким-либо условием.

Составной предикат состоит из предикатов, объединенных отрицанием, конъюнкцией, дизъюнкцией, импликацией либо эквивалентностью.

Секция INITIALISATION представляет собой подстановку, инициализирующую переменные абстрактной машины.

## Подстановки

Подстановки представляют собой математические нотации, преобразующие предикаты. Пусть *S* – подстановка, а *P* – предикат. Тогда запись *S* [ *P* ] представляет предикат, полученный после трансформации *P* подстановкой *S*. Подстановки позволяют моделировать динамический аспект В-технологии: позволяют установить, как свойства данных преобразуются операциями компонента. Механизм трансформации предиката подстановкой позволяет генерировать теоремы специального вида - *proof obligations –* относящиеся к инициализации и операциям. Например, для того, чтобы абстрактная машина была семантически корректна, необходимо доказать, что каждая операция машины сохраняет инвариант.

Рассмотрим наиболее употребляемые в отображении подстановки.

### Подстановка присваивания

При помощи данной подстановки возможно присваивание переменной какого-либо выражения. Возможны разные варианты применения данной подстановки: применение к одной переменной, применение к списку переменных, применение к функции, применение к записи.

Рассмотрим примеры применения данной подстановки:

x2 := x1 + 1 ;

tab1 := {(0 m 3), (1 m 1), (2 m -7)} ;

tab1 (1) := 12 ;

tab2 := tab3 ;

y1, y2, y3 := 0, 0, 0 ;

z1, z2 := z2, z1 ;

### 4.2.2. Подстановка с предусловием

Подстановка с предусловием используется для задания предусловий перед вызовом операций. Операция вызывается только в том случае, если предусловие является истиной. Пример данной подстановки:

PRE

x1 ∈NAT1

THEN

x1 := x1 - 1

END

### 4.2.3. Подстановка неограниченного выбора

В общем виде данная подстановка выглядит следующим образом:

ANY X WHERE P THEN S END

где X является непустым списком переменных, чьей областью видимости является подстановка S и предикат P. Переменные доступны только на чтение в подстановке S. Переменные должны быть типизированы в предикате P с использованием конъюнкций.

Данная подстановка позволяет использовать в подстановке S абстрактные данные, объявленные в списке X и удостоверяющие предикат P.

Пример подстановки неограниченного выбора:

ANY r1, r2 WHERE

r1 ∈NAT ∧

r2 ∈NAT ∧

r12 + r22 = 25

THEN

SumR := r1 + r2

END

### 4.2.4. Последовательная подстановка

Последовательная подстановка отвечает за последовательное выполнение двух подстановок.

Пример последовательной подстановки, меняющей местами значения двух переменных:

z := x ; x := y ; y := z

### 4.2.5. Подстановка вызова операции

Подстановка вызова операции используется для применения операции путем замены формальных параметров на действительные параметры. Действительные входные параметры являются переменными, а действительные выходные параметры являются именами, ссылающимися на данные, которые можно изменять. Вызов операции определен в четырех разных формах, в зависимости от присутствия входных и выходных параметров.

op1 ;

op2 (x0 + 1, TRUE) ;

res1, res2 ←op3 ;

res, flag ←op4 (x0)

### 4.2.6. Параллельная подстановка

Параллельная подстановка отвечает за одновременное выполнение двух подстановок. Необходимо, чтобы подстановки могли выполняться независимо друг от друга. Данная подстановка обладает свойствами коммуникативности и ассоциативности. В следующем примере значения переменных меняются местами:

x := y ||

y := x

## Выражения

Выражение является основным конструкторским элементом языка. Оно является формулой, которая определяет элемент данных. В языке AMN выражения разбиты на семейства, объединенные по типу данных использующихся в выражении элементов. Например, простейшие выражения, арифметические выражения, выражения над множествами и т.д.

Приведем примеры выражений, наиболее часто использующихся при построении отображения.

### Бинарное соответствие

Бинарное соответствие определяет отношение двух множеств, которое в свою очередь является множеством. Так, отношение R множества E к множеству F это множество пар (x ↦ y), где x принадлежит E и y принадлежит F. Пример бинарного соответствия:

rel1 = {(0 ↦ FALSE), (1 ↦ TRUE), (2 ↦ FALSE), (3 ↦ TRUE), (4 ↦ FALSE), (5 ↦TRUE)}

Определяет соответствие между множеством целых чисел от одного до пяти и булевым выражением.

### Пересечение множеств

Пересечение множеств определяет множество элементов, принадлежащих обоим множествам. Так, для множеств E1 и E2:

E1 = {-1, 0, 3, 7, 8} и E2 = {-3, -1, 4, 7, 9},

E1∩F1 = {-1, 7}

### Переписывание отношений

Выражение вида R1 <+ R2 определяет переписывание отношения R1 отношением R2. Для отношений R1 и R2:

R1 = {(2 ↦ 1), (2 ↦ 8), (3 ↦ 9), (4 ↦ 7), (4 ↦ 9)} и R2 = {(0 ↦ -1), (1 ↦ 7), (2 ↦ 9)},

R1 <+ R2= {(0 ↦ -1), (1 ↦ 7), (2 ↦ 9), (3 ↦ 9), (4 ↦ 7), (4 ↦ 9)}

### Частичная функция

Частичная функция одного множества в другое – отношение, которое не содержит две различных пары с одинаковым первым элементом.

### Полная функция

Тотальная функция – это частичная функция, чьим доменом является именно первое множество.

Иллюстрация частичной и полной функции:

If r1 = {(0 ↦ 1), (1 ↦ 2), (2 ↦ 2)},

then r1 {0, 1, 2, 3} ⇸ {0, 1, 2}

and r1 {0, 1, 2} → {0, 1, 2}

При моделировании было решено абстрагироваться от стандартной классификации и объединить выражения по более общему признаку. Было выделено три больших класса выражений, объединенных общими свойствами:

* унарный оператор
* бинарный оператор
* функциональное выражение

Также был выделен класс простейших константных выражений: численных, булевых, строковых, множеств.

# Построение отображений моделей на основе соответствий между элементами моделей

В качестве исходной (канонической) информационной модели рассматривается язык СИНТЕЗ [1], ориентированный на семантическую интероперабельность и композиционное проектирование информационных систем в широком диапазоне существующих неоднородных информационных компонентов. В качестве целевой информационной модели рассматривается теоретико-модельный формальный язык спецификаций AMN, основанный на теории множеств и логике предикатов первого порядка.

Модуль, как основная единица определения канонической модели, является единицей отображения канонической модели в AMN. Модуль представляется в AMN набором абстрактных машин, состоящим из контекстной машины и машин, соответствующих АТД модуля. Контекстная машина содержит информацию, характеризующую модуль, как целое, машины типов содержат информацию, характерную для отдельных типов. Таким образом, для модуля канонической модели, , где контекстная машина, множество машин, соответствующих АТД.

## Экстенсиональная интерпретация АТД

Моделирование АТД в AMN, также, как и формальная семантика АТД, базируется на экстенсиональной принципе: тип моделируется константным множеством своих экземпляров. Такое множество экземпляров называется *экстенсионалом* типа. В отличие от понятия экстенсионала в формальной семантике канонической модели, экстенсионал типа в AMN представляет собой конечное множество *потенциальных* значений типа. Атрибутные функции не определены изначально на потенциальных значениях. При необходимости получить значение типа с определенными значениями атрибутов, из экстенсионала типа выбирается произвольный незанятый элемент и атрибутные функции доопределяются на нем требуемым образом. Такой подход, в отличие от прямого моделирования в AMN семантики АТД, позволяет избежать бесконечности экстенсионалов и значительно упрощает моделирование функций в AMN операциями абстрактных машин. При этом упрощается процесс автоматического и (или) интерактивного доказательства уточнения.

Потенциальные экземпляры всех абстрактных типов моделируются предопределенным множеством *AVAL* и для каждого типа *T* с экстенсионалом выполнено . Экстенсионалы типов соотносятся в соответствии с отношением тип-подтип; для каждой пары типов , , что является подтипом , выполнено отношение включения на экстенсионалах . Таким образом моделируется иерархия типов.

Заметим, что в общем случае отношение тип-подтип сводится к отношению уточнения типов, и факт уточнения для всех пар тип-подтип, входящих в модуль, является одним из свойств непротиворечивости спецификации модуля. Поэтому непротиворечивость отношения тип-подтип в спецификации канонической модели следует доказывать с использованием B-технологии, отображая его и его подтип в AMN раздельно. Для самого же отображения канонической модели в AMN достаточно экстенсиональной трактовки отношения тип-подтип: имея экстенсиональное представление отношения тип-подтип в AMN и доказательство уточнения соответствуюзих типов, проведенное с использованием B-технологии, можно утверждать, что отношение тип-подтип моделируется в AMN корректно.

Представление множества своих потенциальных экземпляров позволяет естественным образом моделировать спецификацию типа набором функций AMN: атрибут *a*  типа T моделируется функцией , где тип значений, которые может принимать атрибут *a*. Экстенсиональные константы позволяют также строго типизировать переменные в инвариантах и операциях абстрактных машин. Например, для того, чтобы передать значение некоторого типа в качестве параметра в операцию машины, достаточно типизировать переменную-параметр соответствующим экстенсионалом.

Потенциальные экземпляры всех объектных типов моделируются предопределенным множеством . Для обеспечения уникальной идентификации объектов заводится множество объектных идентификаторов *OID*. Биекция моделирует неявный атрибут **self** каждого объектного типа.

## Моделирование структуры спецификации типов при помощи средств композиции абстрактных машин

Спецификации типов канонической модели могут ссылаться друг на друга в спецификациях атрибутов, методов, инвариантов, образуя структуру спецификации типов. Основным принципом отображения структуры спецификации типов канонической модели в AMN является стремление отобразить каждый АТД отдельной абстрактной машиной. Структура спецификации типов находит свое отражение в спецификации абстрактных машин следующим образом:

* абстрактная машина включается в секцию SEES абстрактной машины если методы типов АТД *i* ссылаются на атрибуты, методы типов из АТД *j*
* абстрактная машина включается в секцию USES абстрактной машины если инварианты типов АТД *i* ссылаются на атрибуты, методы типов из АТД *j*
* абстрактная машина включается в секцию INCLUDES абстрактной машины если инварианты типов АТД *i* изменяют атрибуты типов из АТД *j*
* абстрактная машина включается в секцию EXTENDS абстрактной машины если АТД *j* является непосредственным супертипом АТД *i*

## Контекстная машина

Контекстная машина модуля *M* получает имя *ContextM,* и содержит разделы SETS, CONSTANTS, PROPERTIES. Основное содержание контекстной машины составляют экстенсионалы типов и отношения между экстенсионалами.

Для каждого типа *T ∊ Types(M)* вводится константное множество *ext\_T*, моделирующее полный экстенсионал и константное множество *extp\_T,* моделирующее собственный экстенсионал. Иерархия типов выстраивается при помощи следующего соотношения, записываемого в раздел PROPERTIES:

*;* длякаждой пары типов , , что является подтипом

В раздел PROPERTIES также помещается условие пустого пересечения собственных экстенсионалов абстрактных типов:

Здесь , … , обозначают все абстрактные типы модуля.

Множества и константы, определенные в *ContextM,* включают:

* множество *AVAL*, элементы которого моделируют потенциальные экземпляры всех абстрактных типов;
* константу *Obj,* типизированную в разделе PROPERTIES как *Obj∊ℙ(AVAL),* обозначающую множество, моделирующее совокупность экстенсионалов объектных типов;
* множество *OID* объектных идентификаторов;
* константу *self,* устанавливающую взаимно-однозначное соответствие между экземплярами объектных типов и объектными идентификаторами и типизированную в разделе PROPERTIES как
* константы *ext\_T, extp\_T* для каждого типа *T*, типизированные в разделе PROPERTIES как

в случае если тип объектный; и типизированная как

Контекстная машина содержит информацию, характеризующую модуль как целое, и поэтому каждая из машин, соответствующих АТД, включает контекстную машину в свой раздел SEES.

**Пример 1. Отображение спецификации модуля в контекстную машину**

Построим отображение модуля в контекстную машину на примере модуля FundingAgency.

{ FundingAgency\_module; in: module;  
type:  
{ Person; in: type;  
 name: string;  
},  
{ Researcher; in: type;  
 supertype: Person;  
 research\_area: string;  
}  
}

В примере представлен модуль FundingAgency, в котором определены два абстрактных типа данных – АТД Person и АТД Researcher. В АТД Person определен атрибут name типа строка, который определяет имя индивида. Из спецификации АТД Researcher видно, что данный АТД является подтипом АТД Person и имеет атрибут *research\_area*, устанавливающий сферу интересов данного исследователя. Стоит отметить, что АТД Researcher наследует атрибут *name* от своего супертипа.

Данному модулю в соответствие ставится следующая абстрактная машина:

MACHINE FundingAgency\_Context

SETS

AVAL; OID

CONSTANTS

Obj,

self,

extp\_Person, ext\_Person,

extp\_Researcher, ext\_Researcher

PROPERTIES

Obj: POW(AVAL) &

self: Obj ⤖ OID &

extp\_Person ∧ extp\_Researcher = {} &

extp\_Person: POW(Obj) & ext\_Person: POW(Obj) &

extp\_Person ⊆ ext\_Person &

extp\_Researcher: POW(Obj) & ext\_Researcher: POW(Obj) & extp\_Researcher ⊆ ext\_Researcher & ext\_Researcher ⊂ ext\_Person

END

## Структура абстрактной машины, соответствующей типу: моделирование атрибутов, методов, инвариантов

Рассмотрим, каким образом моделируются атрибуты, методы, инварианты некоторого типа *T* в абстрактной машине *M­T*, инкапсулирующей отображение спецификации типа *T* в AMN.

Для каждого атрибута *a : Ta* типа *T* машина *M­T* содержит переменную *a* в разделе VARIABLES. Переменная *a* типизируется в разделе INVARIANT предикатом языка AMN: , где является типом атрибута *a*.

В случае, если атрибут имеет тип boolean, раздел SEES машины *M­T* включает предопределенную в AMN машину *Bool\_TYPE*, если атрибут имеет тип string машину *String\_TYPE.*

Методы модификации атрибутов моделируются операциями set\_имя\_атрибута(s) машины *M­T* в разделе OPERATIONS. Параметр s представляет собой множество пар , где *o* – значение типа *T, v* – значение, которое должно быть сообщено соответствующему атрибуту значения *o* операцией *set.* В теле операции используется обобщенная подстановка c предварительным условием, где предварительным условием является принадлежность параметра *s* частичной функции , где является типом атрибута *a*. Подстановкой является постановка присваивания определяемому атрибуту параметра *s.*

Инициализация атрибутов осуществляется при помощи подстановки ANY в разделе INITIALIZAITON. В раздел WHERE подставляется предикат , аналогичный предикатам типизации атрибутов.

Инициализирующие подстановки атрибутов соединяются между собой оператором параллельной подстановки ∥.

Поясним данное отображение на примере.

**Пример 2. Моделирование атрибутов АТД**

Рассмотрим спецификацию АТД Dispatcher:

{ Dispatcher; in: type;

experts: {set; type\_of\_element: Specialist;};

experts\_got: boolean;

experts\_speciality: string;

};

В данном АТД определены три атрибута различных типов. Далее идет представление этого АТД в виде абстрактной машины.

MACHINE Dispatcher

SEES

String\_TYPE,

Bool\_TYPE,

FundingAgency\_Context

VARIABLES

experts\_speciality

experts\_got,

experts

INVARIANT

experts\_speciality ∈ ext\_Dispatcher → STRING &

experts\_got ∈ ext\_Dispatcher → BOOL &

experts ∈ ext\_Dispatcher → POW(ext\_Specialist)

INITIALISATION

ANY arelevant\_experts WHERE

arelevant\_experts ∈ ext\_Dispatcher → STRING

THEN relevant\_experts:= arelevant\_experts

END ||

ANY aexperts\_got WHERE aexperts\_got ∈ ext\_Dispatcher → BOOL

THEN experts\_got:= aexperts\_got

END ||

ANY aexperts WHERE aexperts ∈ ext\_Dispatcher → POW(ext\_Specialist)

THEN experts:= aexperts

END

OPERATIONS

set\_experts\_speciality(sav) =

PRE sav ∈ ext\_Dispatcher ⇸ STRING

THEN relevant\_experts:= relevant\_experts <+ sav

END;

set\_experts\_got(sav) =

PRE sav ∈ ext\_Dispatcher ⇸ BOOL

THEN experts\_got:= experts\_got <+ sav

END;

set\_experts(sav) =

PRE sav ∈ ext\_Dispatcher ⇸ POW(ext\_Specialist)

THEN experts:= experts <+ sav

END

END

Метод *meth* типа *T* всегда имеет представление в AMN операцией *meth\_op* машины *M­T*. Набор входных параметров метода отображается в набор входных параметров операции, аналогично поступаем с выходными параметрами. Также во входные параметры добавляется объект, для которого вызывается метод *meth.* Тело операции формирует подстановка с предусловием. В качестве предусловия выступает конъюнкция ограничений на параметры, так для объекта ограничение заключается в принадлежности этого объекта экстенсионалу АТД. Для других параметров записывается предикат принадлежности экстенсионалу типа данных параметра, либо просто типу, если тип простой. В конструкцию THEN подстановки записывается подстановка ANY. В секцию ANY записываются новые переменные, имена которых не имеют вхождений в формулу спецификации метода. Новые переменные соответствуют выходным параметрам метода, либо атрибутам, значение которых изменяется в теле метода при помощи операции *set*. В секцию WHERE записывается конъюктивный предикат, состоящий из предикатов принадлежности новых переменных типам данных соответствующих им атрибутов и предикатом AMN, смоделированным по формуле спецификации метода. В секцию THEN записывается подстановка, присваивающая атрибуту значение соответствующей ему новой переменной. Моделирование формул канонической модели описано далее.

## Моделирование формул канонической модели

В данном разделе рассматривается моделирование формул предикатами AMN. Отображение строится индуктивно, начиная от атомарных предикатов и заканчивая составными формулами.

### Атомарные предикаты

Атомарные предикаты могут являться предикатами-классами, предикатами-методами и встроенными предикатами канонической модели.

Моделирование предиката класса строится следующим образом:

Атомарным предикатом может быть вызов метода

где путь в композиционной иерархии АТД, при условии, что *M* не изменяет состояния ИС. В случае, если *M* имеет единственный выходной параметр типа boolean (т.е. является логической функцией ), моделирование вызова осуществляется следующим образом:

Метод *M* может иметь несколько выходных параметров, для определенности будем считать, что все параметры с *n*-го по *m*-й, *n ≤ m*, являются выходными. В этом случае, для каждого *i*, что *i*-ый параметр *M* является выходным, терм *pi* должен представлять собой имя переменной, а моделирование вызова осуществляется следующим образом:

Если началом пути является указатель текущего экземпляра типа this, моделирование пути осуществляется следующим образом: . Переменная *o* обозначает объект, для которого вызывается метод. Если началом пути является имя переменной, то моделирование пути осуществляется следующим образом: .

Моделирование встроенных предикатов same, in осуществляется следующим образом:

### Условия

Условия-отношения формируются при помощи операций-отношений

из выражений. Операции-отношения имеют аналоги, определенные на типах AMN, соответствующих целому, перечислимому, булевскому, строковому типам канонической модели. В случае, если операндами операции-отношения являются множества, семантикой операции-отношения является соответствующая операция, определенная на множествах AMN.

Операндами отношений могут быть также объекты АТД.

Арифметические выражения формируются при помощи арифметических операций (+,-,\*,/) из атомарных термов. Арифметические операции имеют аналоги, определенные на типе *NAT*, и поэтому обладают естественной семантикой. Выражения-множества представляют собой атомарный терм.

Рассмотрим теперь значения, которые может принимать атомарный терм и их отображение в AMN. Атомарным термом условия может быть значение (идентификатор, целое число, строка, логическое значение), имя переменной, путь в композиционной иерархии АТД, вызов метода, формула-выделение множества, имя класса. Типы значений канонической модели имеют аналоги в AMN, и поэтому значения интерпретируются естественным образом. Имя переменной тождественно отображается по её id. Отображение пути показано в разделе моделирования вызова метода. Выделение множества моделируется следующим образом:

### Составные формулы

Составные формулы канонической модели могут быть образованы из атомарных предикатов и условий при помощи логических операций (&, |, ^, ->).

Логические операции в формулах канонической модели моделируются естественным для логики предикатов первого порядка образом:

Моделирование разрешенных в канонической модели формул с кванторами осуществляется следующим образом:

Таким образом построено отображение пространства формул языка СИНТЕЗ на пространство предикатов AMN.

Приведем пример моделирования спецификации функционального атрибута.

**Пример 3. Моделирование функционального атрибута**

Возьмем уже использовавшийся нами АТД Dispatcher и расширим его двумя функциональными атрибутами get\_experts и count\_experts.

{ Dispatcher;

in: type;

experts: {set; type\_of\_element: Specialist;};

experts\_got: boolean;

relevant\_experts: {set; type\_of\_element: Specialist;};

get\_experts:

{ in: function;

{ predicative:

{" this.experts~ = expert & this.experts\_got~ = true

"}

};

};

count\_experts:

{in: function;

params: {-number\_of\_experts/integer};

{ predicative:

{" isempty(this.experts) & this.experts\_got~ = true &

number\_of\_experts = card(this.relevant\_experts)

"}

};

};

};

Легко заметить, что оба метода являются методами с явными эффектами, т.е. изменяют состояние ИС. Так метод get\_experts изменяет атрибуты experts и experts\_got. А метод count\_experts изменяет значение атрибута experts\_got. Более того метод count\_experts также имеет один выходной параметр - number\_of\_experts типа INTEGER.

Посмотрим, как будет выглядеть отображение в AMN данной спецификации. Опустим разделы абстрактной машины, описанные ранее, оставив раздел OPERATIONS, в котором и определяется функциональные атрибуты.

OPERATIONS

get\_experts(av)=

PRE av: ext\_Dispatcher

THEN

ANY v2, v1 WHERE

v2: BOOL & v1: POW(ext\_Specialist) &

(v1 = expert &

v2 = TRUE)

THEN

experts\_got(av):= v2;

experts(av):= v1

END

END;

number\_of\_experts <-- count\_experts(av)=

PRE av: ext\_Dispatcher

THEN

ANY v1, number\_of\_experts1 WHERE

v1: BOOL &

number\_of\_experts1: NAT &

(experts(av) = {} &

v1 = TRUE &

number\_of\_experts1 = card(relevant\_experts(av)))

THEN

experts\_got(av):= v1;

number\_of\_experts:= number\_of\_experts1

END

END;

# Реализация трансформации модели СИТНЕЗ в модель AMN на языке ATL

Для трансформации моделей созданы специальные языки, которые развиваются в контексте *Движимой моделями архитектуры* (Model-Driven Architecture, MDA [8]) - подхода, поддерживаемого стандартом OMG MOF (Meta-Object Facility) [12]. 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 метамодели. Стандартом MOF определена четырехуровневая архитектура моделей: модели уровня М0 описывают объекты реального мира, модели уровня М1 называются обычно *схемами*, модели уровня М2 представляют собой собственно *информационные модели* (например, в архитектурах систем управления данными представляющие собой совокупность языка определения данных и языка манипулирования данными), модели уровня М3 – метаметамодели, предназначенные для описания моделей уровня М2.

Примером языка трансформации моделей может являться QVT (Query-View-Transformation) [13].

В качестве языка трансформации моделей в данной работе рассматривается язык ATL (ATLAS Transformation Language) [9], 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 [10]. Для языка ATL на базе платформы Eclipse реализована интегрированная среда разработки, называемая ATL Development Tools (ADT). В качестве модели уровня М3 рассматривается модель *Ecore* [11].

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.

Для построения трансформации языка СИНТЕЗ в язык AMN, абстрактные синтаксисы обоих языков представляется в виде моделей уровня М2, конформных модели *Ecore*. Трансформация позволит осуществлять преобразование произвольных моделей уровня М1, конформных модели СИНТЕЗ уровня М2, (т.е. спецификаций языка СИНТЕЗ) в соответствующие модели уровня М1, конформные модели AMN уровня М2, (т.е. спецификации языка AMN).

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 *SynthesisModule2AMNContextAbstractMachine*

**rule** SynthesisModule2AMNContextAbstractMachine {

**from** m: Synthesis!Module

**to** am: AMN!AbstractMachine(

name <- ‘Context’ + m.name

)

}

входным элементом является элемент типа *Module,* порождается один элемент типа *AbstractMachine*, и имя абстрактной машины формируется из имени с прибавлением к нему приставки ‘Context’. Таким образом, мы породили контекстную абстрактную машину для модуля.

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.

Наряду с сопоставляющими правилами, в языке ATL существуют вызываемые правила (called rules) – специальные правила ATL для явной генерации элементов целевой модели, которые можно вызывать из императивных блоков сопоставляющих правил. К отдельному классу относятся вспомогательные правила (helpers), позволяющие задавать факторизованный ATL код, носящий вспомогательный характер, который можно вызывать из разных частей модуля. Вспомогательные правила не могут генерировать элементы целевой модели. У вспомогательного правила может быть задан контекст, определяющий для какого элемента исходной модели правило может быть вызвано. Если контекст не задан, то обращение к правилу происходит в контексте всего модуля с использованием ключевого слова thisModule. Из сопоставляющих правил можно выделить отдельный подкласс так называемых *ленивых правил* (*lazy rules*). Для того, чтобы ленивое правило отработало, его необходимо явно вызвать из другого правила.

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

## Построение заголовка трансформации

Трансформация языка СИНТЕЗ в язык AMN представляется модулем языка ATL и называется *Synthesis2AMN*. Заголовок трансформации выглядит следующим образом:

**module** Synthesis2AMN;

**create** OUT:AMN **from** IN:Synthesis;

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

## Построение контекстной машины модуля

Генерация контекстной машины модуля производится при помощи сопоставляющих правил, с использованием вызовов ленивых правил. Проиллюстрируем данную трансформацию на примере. Ниже дана неполная спецификация контекстной машины.

**rule** SynthesisModule2AMNAbstractMachine {

**from**

m : Synthesis!ModuleDef(m.oclIsTypeOf(Synthesis!ModuleDef))

**to**

am : AMN!AbstractMachine (

name <- 'Context' + m.name,

sets <- Sequence {avalSet},

abstractConstants <- Sequence {objConst},

properties <- propertiesPredicate),

avalSet : AMN!Deferred (name <- 'AVAL'),

objConst : AMN!NamedConstant (name <- 'Obj' ),

propertiesPredicate : AMN!Conjunction (

predicate <- Sequence {objProp}

),

objProp : AMN!AtomicPredicate (

sign <- 'Belongs',

expression <- objConst,

expression <- rightAval

),

rightAval : AMN!FunctionalExpression (

sign <- 'POW',

expression <- avalSet

)

}

На основании данного правила генерируется абстрактная машина, которой в раздел множеств записывается множество *AVAL,* в раздел констант записывается константа *Obj*. Следует отметить, что напрямую эти данные записывать нельзя, сначала происходит их инициализация, при которой создается объект определенного типа, а затем он записывается в соответствующий раздел. В разделах множеств и констант могут присутствовать более одного элемента, в таком случае можно воспользоваться конструкцией *Sequence*, для которой через запятую записываются необходимые элементы. Аналогично для конструкции *Sequence,* в правиле для элемента *propertiesPredicate.* Необходимо заметить, что некоторые элементы отображения опущены, поэтому *objProp* не единственный конъюнкт. Элемент *objProp* представляет собой атомарный предикат принадлежности элемента множеству, в данном случае множеством является множество подмножеств *AVAL.*

## Построение абстрактных машин АТД модуля и дополнение контекстной машины

Правило отображения АТД преследует две цели: во-первых, создать абстрактную машину для АТД, во-вторых, дополнить разделы PROPERTIES и CONSTANTS контекстной машины информацией по экстенсионалам АТД.

Проиллюстрируем данное высказывание примерами. Для простоты, разделим сопоставляющее правило на два, в соответствии с нашими целями.

### Дополнение контекстной машины

Рассмотрим пример отображения АТД модуля в секции контекстной машины. Данный пример формирует две константы, обозначающие полный и собственный экстенсионалы типа. Также формируется предикат, типизирующий эти константы формулой вида

Интересна секция *do*. Она позволяет использовать императивную семантику при описании трансформаций. В данном случае, она также несет в себе и операцию *resolveTemp,* являющуюся неотъемлемым инструментом сложных трансформаций. Данная конструкция позволяет обращаться из правил языка ATL к любому элементу целевой модели, который будет сгенерирован из исходной модели при помощи сопоставляющего правила.

**rule** abstractDataType2ContextMachineSections {

**from**

adt : Synthesis!ADTDef(adt.oclIsTypeOf(Synthesis!ADTDef))

**to**

extp : AMN!NamedConstant (

name <- 'extp\_' + adt.name

),

ext : AMN!NamedConstant (

name <- 'ext\_' + adt.name

),

conjunction : AMN!Conjunction (

predicate <- leftPred,

predicate <- middlePred,

predicate <- rightPred

),

leftPred : …,

middlePred : …,

rightPred : …

**do** { thisModule.resolveTemp(adt.typeInModule,'am').abstractConstants<- Sequence {extp, ext};

thisModule.resolveTemp(adt.typeInModule,'intersection').expression<- extp;

thisModule.resolveTemp(adt.typeInModule,'propertiesPredicate').predicate<-conjunction;

}

}

В данном случае при помощи первой операции *resolveTemp* мы обращаемся к контекстной абстрактной машине, полученной на основании спецификации модуля, и записываем константные множества, моделирующие полный и собственный экстенсионал в раздел абстрактных констант. При помощи второй операции мы обращаемся к элементу *intersection*, являющемуся бинарным оператором, и записываем в выражение константное множество, моделирующее полный экстенсионал. Таким образом, мы получили условие пустого пересечения собственных экстенсионалов. При помощи третьей операции мы записываем формулу, типизирующую эти константы в элемент *propertiesPredicate,* являющийся конъюнкцией, записываемой затем в раздел PROPERTIES контекстной машины.

### Генерация абстрактной машины для АТД

Трансформация АТД в абстрактную машину выглядит следующим образом:

**rule** abstractDataType2ADTAbstractMachine {

**from**

adt : Synthesis!ADTDef(adt.oclIsTypeOf(Synthesis!ADTDef))

**to**

am : AMN!AbstractMachine (

name <- adt.name,

initialization <- initSubt,

invariant <- predicate,

sees <- thisModule.resolveTemp(adt.typeInModule, 'am')

),

predicate : AMN!Conjunction (),

initSubt : AMN!Simultaneous ()

**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');

}

}

}

**rule** genStrictInclusion(adt : Synthesis!ADTDef, subType : Synthesis!ADTDef) {

**to**

strictInclusionSubtypeRelation : AMN!AtomicPredicate (

sign <- 'Inclusion SUBTYPE',

expression <- Sequence{

thisModule.resolveTemp(subType,'ext'), thisModule.resolveTemp(adt, 'ext')

}

)

**do** {

thisModule.resolveTemp(adt.typeInModule,'propertiesPredicate').predicate <-strictInclusionSubtypeRelation;

}

}

На основе спецификации АТД строится абстрактная машина. Имя машины задает имя абстрактного типа данных, для разделов INITIALIZATION, INVARIANT создаются заглушки, далее в них будут подставляться элементы трансформации при помощи операции *resolveTemp*. Так как инвариант представляется конъюнкцией, для него создается конъюктивный предикат, инициализирующие подстановки соединяются между собой при помощи оператора параллельной подстановки, для чего был создан соответствующий элемент. В императивной секции при помощи конструкции *for* для АТД, имеющих подтипы, производится генерация атомарного предиката включения константного множества экстенсионалов подтипа в множество экстенсионалов супертипа. Генерация производится при помощи вызываемого правила, на вход которому подают АТД типа и его супертипа. В теле правила при помощи конструкций *resolveTemp* берутся экстенсионалы, и записываются в атомарный предикат. В императивной секции данная подстановка записывается в предикат раздела PROPERTIES модуля, содержащего эти типы. При помощи второй конструкции *for* опять же происходит итерация по всем супертипам данного АТД и машина, сгенерированная для супертипа записывается в раздел EXTENDS.

### Генерация секций машины АТД в соответствии с его нефункциональными атрибутами

Будем считать, что атрибуты абстрактного типа данных не являются функциональными. Правила для функциональных атрибутов рассматриваются в следующем разделе. Рассмотрим правило, при помощи которого происходит наполнение разделов абстрактной машины, созданной в предыдущем пункте. Для примера было взято правило формирования операции. Операция представляет метод модификации переменной и строится следующим образом:

**rule** attribute2ATDMachineOperation {

**from**

at : Synthesis!AttributeDef(at.isNotFunctional())

**to**

inpVar : AMN!Variable (

name <- 'sav'

),

attrVar : AMN!Variable (

name <- at.name

),

operation : AMN!Operation (

name <- 'set\_' + at.name,

inputParams <- inpVar,

substitution <- subst

),

subst : AMN!Precondition (

pre <- predicate,

thenPart <- substitution

),

predicate : AMN!AtomicPredicate (

sign <- 'Belongs',

expression <- inpVar,

expression <- partialFunction

),

partialFunction : AMN!BinaryOperator (

sign <- 'PartialFunction',

expression <- thisModule.resolveTemp(at.attributeOf, 'ext')

),

substitution : AMN!BecomesEqual (

leftExpression <- attrVar,

rightExpression <- overWrite

),

overWrite : AMN!BinaryOperator (

sign <- 'Overwrite',

expression <- attrVar,

expression <- inpVar

)

**do** {

thisModule.resolveTemp(at.attributeOf, 'am').operations <- operation;

}

}

Заголовок операции состоит из имени операции, формируемом при помощи конкатенации строки ‘set’ и имени атрибута и входного параметра, значение которого должно быть сообщено соответствующему атрибуту. Тело операции составляет подстановка с предусловием. Сформированная операция заносится в раздел операций машины АТД.

Следует заметить, что данное правило применяется только к нефункциональным атрибутам. Это обеспечивается специфицированием условия *isNotFunctional()* в секции **from**, являющегося вызовом вспомогательного правила вида:

**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**;

Генерация инициализации атрибутов и предикатов, специфицирующих атрибуты, производится аналогичным образом. Она реализована совместно с генерацией операции, так как использует одни и те же элементы.

### Генерация операций функциональных атрибутов

Для функционального атрибута формируется операция, конструкция которой аналогична конструкциям для нефункциональных атрибутов. Различие состоит в том, что в теле подстановки находится подстановка ANY, с помощью которой моделируется предикативная спецификация атрибута, вводятся новые переменные для атрибутов, меняющих свое значение в ходе работы функции, и ставятся условия на эти переменные. Далее приведен пример отображения:

**rule** functionalattributesToOperations {

**from**

at : Synthesis!AttributeDef(at.isFunctional())

**to**

operation : AMN!Operation (

…

substitution <- subst

…

),

…

subst : AMN!Precondition (

pre <- precondPredicate,

thenPart <- thenSubstAny

),

thenSubstAny : AMN!Any (

where <- whereConj,

thenPart <- sequenceSubstitution

)

**do** {

**for** ( e **in** at.type.parameters ) {

**if** ( e.type.oclIsTypeOf(Synthesis!ADTDef) ) {

thisModule.generateParameters(e, at);

}

}

thisModule.resolveTemp(at.attributeOf,'am').operations<-operation;

}

}

Из тела правила были убраны конструкции, аналогичные правилу для нефункционального атрибута. Секция ANY и WHEREподстановки ANY формируется на основании переменных, изменяющих свое значение, и наполняется в ходе правил обработки выходных параметров и функций, устанавливающих значение атрибута. Также в конструкцию WHERE заносится смоделированный предикат. Операцией *resolveTemp* данная операция заносится в раздел операций абстрактной машины АТД.

Рассмотрим правило generateParameters, строящее входные и выходные параметры функции на основе параметров функции и правило generateAdditionalParam строящее дополнительные переменные:

**rule** generateParameters ( param : Synthesis!ParameterDef, attr : Synthesis!AttributeDef ) {

**to**

var : AMN!Variable (

name <- param.name

),

varPred : AMN!AtomicPredicate (

sign <- 'Belongs',

expression <- var,

expression <- thisModule.resolveTemp(param.type, 'ext')

)

**do** {

**if** ( param.parameterKind = 'input' ) {

thisModule.resolveTemp(attr,'operation').inputParams<-var;

}

**else** {

**if** ( param.parameterKind = 'output' ) {

thisModule.resolveTemp(attr,'operation').outputParams<-var;

thisModule.generateAdditionalParam(param,attr);

}

}

thisModule.resolveTemp(attr,'precondPredicate').predicate<-varPred;

}

}

**rule** generateAdditionalParam ( param : Synthesis!ParameterDef, attr : Synthesis!AttributeDef ) {

**to**

varAny : AMN!Variable (

name <- 'v' + thisModule.counter

),

varPred : AMN!AtomicPredicate (

sign <- 'Belongs',

expression <- varAny,

expression <- thisModule.resolveTemp(param.type, 'ext')

)

**do** {

thisModule.resolveTemp(attr,'thenSubstAny').any<-varAny;

thisModule.resolveTemp(attr,'whereConj').predicate<-varPred;

thisModule.counter <- thisModule.counter + 1;

}

}

Вызываемое правило generateParameters создает переменную и предикат принадлежности переменной ее типу. В секции *do* на основании типа вида переменной она записывается во входные либо выходные параметры операции, а в раздел PRE подстановки с предусловием записывается предикат. Аналогично для правила generateAdditionalParam, только результаты записываются в подстановку ANY.

## Построение отображения формул и обработка функций доступа

Составные предикаты отображаются без изменений. Рассмотрим отображение на примере конъюнкции:

**rule** conjunctionToAMNConjunciton {

**from**

conjSynth : Synthesis!Conjunction(conjSynth.oclIsTypeOf(Synthesis!Conjunction))

**to**

conjAMN : AMN!Conjunction (

predicate <- conjSynth.formula

)

}

Формулы конъюнкции напрямую отображаются в предикаты AMN. Для того чтобы формулы отобразились корректно необходимо отображение всех возможных элементов формулы. Так, атомарная формула языка СИНТЕЗ отображается в атомарный предикат языка AMN при помощи следующего правила.

**rule** atomToAMNAtomicPredicate {

**from**

atomSynth : Synthesis!Atom(atomSynth.oclIsTypeOf(Synthesis!Atom))

**to**

atomicAMN : AMN!AtomicPredicate (

sign <- atomSynth.symbol,

expression <- atomSynth.terms

)

}

Термы, представляющие собой значения простых типов даных, отображаются в соответствующие элементы AMN.

Отображение функций доступа рассмотрим на примере функции set.

**rule** setFunction2AMNExpression {

**from**

synthSetFunction : Synthesis!SetFunction(synthSetFunction.oclIsTypeOf(Synthesis!SetFunction))

**to**

var : AMN!Variable (

name <- 'v' + thisModule.counter

),

varType : AMN!AtomicPredicate (

sign <- 'Belongs',

expression <- var,

expression <- thisModule.resolveTemp(synthSetFunction.attribute.attributeOf, 'ext')

),

varParam : AMN!Variable (

name <- synthSetFunction.name.split('\\.')->first()

),

varSubstBecomesEqual : AMN!BecomesEqual (

leftExpression <- leftExpBecomes,

rightExpression <- var

),

leftExpBecomes : AMN!BinaryOperator (

sign <- '()SetFunc',

expression <- Sequence {varAttr,varObjType}

),

varAttr : AMN!Variable (

name <- synthSetFunction.attribute.name

),

varObjType : AMN!Variable (

name <- 'o'

),

setExternalVar : AMN!Call (

operationName <- 'set\_' + synthSetFunction.attribute.name,

actualParam <- actualParamExp

),

actualParamExp : AMN!BinaryOperator (

sign <- 'binaryCorrespondence',

expression <- Sequence {varParam,var}

)

**do** {

…императивные конструкции…

}

}

В данном правиле создается вспомогательная переменная, с именем, формируемым с помощью счетчика, во избежание дублирования переменных. Далее новая переменная типизируется в соответствии с типом соответствующего ей атрибута. Создаются две подстановки, в зависимости от того, определен ли данный атрибут в том же типе данных, либо он является атрибутом другого. Для атрибута того же типа создается присваивающая подстановка, для атрибута другого типа создается подстановка вызова операции. В секции *do* происходит запись вновь созданных элементов в соответствующие им секции. Если меняется атрибут другого типа данных, то соответствующая данному типу абстрактная машина записывается в раздел INCLUDES.

# Заключение

Целью данной̆ дипломной̆ работы является разработка и реализация трансформации модели языка СИНТЕЗ в язык AMN с использованием декларативно-императивного языка трансформации моделей̆ ATL. В рамках работы была построена модель языка AMN, конформная модели Ecore. На основе существующего отображения языка СИНТЕЗ в язык AMN была разработана и реализована трансформация на языке ATL. Впоследствии компонент, реализующий трансформацию, будет внедрен в архитектуру унификатора моделей.

## Список литературы

1. *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. Moscow: IPI RAN, 2007. - 171 p.

2. *Briukhov D.O., Kalinichenko L.A.* Component-based information systems development tool supporting the SYNTHESIS design method // Advances in Databases and Information Systems: Proc. of the Second East European Conference. – Berlin-Heidelberg: Springer-Verlag, 1998. – P. 305-327.

3. *Briukhov D.O., Kalinichenko L.A., Skvortsov N.A.* Information sources registration at a subject mediator as compositional development // Advances in Databases and Information Systems: Proc. of the 5th East European Conference. – Berlin-Heidelberg: Springer-Verlag, 2001. – P. 70-83.

4. *Kalinichenko L.A.* Method for Data Models Integration in the Common Paradigm // Advances in Databases and Information Systems: Proc. of the First East-European Conference. – St. Petersburg: Nevsky Dialekt, 1997. – P. 275-284.

5. *Kalinichenko L.A., Martynov D.O., Stupnikov S.A.* Query rewriting using views in a typed mediator environment // Advances in Databases and Information Systems: Proceedings of the East European Conference. – Berlin-Heidelberg: Springer-Verlag, 2004. – P. 37-53.

6. *Abrial J.-R.* The B-Book: Assigning Programs to Meanings // Cambridge: Cambridge University Press, 1996.

7. *Abrial J.-R.* B-Technology// Technical overview. – B-Core (UK) Ltd., 1993.

8. OMG Model Driven Architecture // http://www.omg.org/mda/

9. ATL Project // <http://www.eclipse.org/m2m/atl/>

10. OMG/OCL Object Constraint Language (OCL) 2.0. OMG Final Adopted Specification. ptc/03-10-14, 2003.

11. *Budinsky, F., Steinberg, D., Ellersick, R., Grose, T.* Eclipse Modeling Framework, Chapter 5 ”Ecore Modeling Concepts”. - Addison Wesley Professional, 2004.

12. Meta Object Facility (MOF) 2.0 Core Specification // <http://www.omg.org/cgi-bin/apps/doc?ptc/03-10-04.pdf>, 2003.

13. Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification // http://www.omg.org/cgi-bin/doc?ptc/2007-07-07, 2007.

14. *Kalinichenko L.A., Briukhov D.O., Martynov D.O., Skvortsov N.A., Stupnikov S.A.* Mediation Framework for Enterprise Information System Infrastructures. Proc. of the 9th International Conference on Enterprise Information Systems ICEIS 2007. -- Funchal, 2007. -- Volume Databases and Information Systems Integration. -- P. 246--251.

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

16. *Stupnikov S.A.* Mapping of canonical model core specifications in Abstract Machine Notation // Formal Methods and Models for Compositional Infrastuctures of Distributed Information Systems: The Systems and Means of Informatics, Special Issue. -- Moscow: IPI RAN, 2005. -- P. 69--95

17. Eclipse // <http://www.eclipse.org/>

18. B language reference manual. Version 1.8.6. – ClearSy, 2009.