Российская академия наук
Сибирское отделение
Институт систем информатики
им. А. П. Ершова

С. А. Бражник

ОБЗОР ФОРМАЛЬНЫХ ПОДХОДОВ К СПЕЦИФИКАЦИИ ЯЗЫКОВ UML И OCL

Препринт
121

Новосибирск 2004

Язык UML в последние годы стал фактически стандартом для объектно-ориентированного анализа и проектирования информационных систем. Предлагаемая работа содержит обзор основных подходов к формализации языка UML версий 1.x и языка OCL.

Siberian Division of the Russian Academy of Sciences
A. P. Ershov Institute of Informatics Systems

Sergey A. Brazhnik

REVIEW OF FORMAL APPROACHES FOR THE UML AND OCL LANGUAGES

Preprint
121

Novosibirsk 2004

UML is a de-facto standard language for the Object-Oriented analysis and design of information systems. This paper contains a review of basic formal approaches for the UML 1.x and OCL languages.

1. ВВЕДЕНИЕ

За последние несколько лет язык UML стал фактически стандартом для объектно-ориентированного анализа и проектирования информационных систем. Однако до сих пор в спецификации языка существуют неформальные описания и места с плохой внутренней структурой, что осложняет работу по формальной спецификации этого языка моделирования. Данной тематике, формализации языка UML, посвящено множество научных работ. Предлагаемая статья содержит обзор основных подходов к формализации языка UML версий 1.x и его неотъемлемой части — языка OCL.

Статья организована следующим образом: в первой части даны основные понятия языка моделирования и вводные понятия языков UML и OCL, вторая часть статьи содержит обзор существующих подходов к формализации языков UML и OCL и третья часть — заключительная.

Основные понятия

Краеугольным понятием в унифицированном языке моделирования является понятие модели.

Модель — это абстрактное представление физической системы с определенной точки зрения. При этом *физическая система* — это набор связанных между собой физических единиц, включающих в себя людей, программное и компьютерное обеспечение и функционирующих совместно для достижения определенной цели. Физическая система может быть описана одной или несколькими моделями, возможно с разных точек зрения [1, 2].

В UML существует также понятие метамодели и мета-метамодели.

Метамодель — это модель, определяющая язык описания модели.

Мета-метамодель — это модель, определяющая язык описания метамодели. Связь между мета-метамоделью и метамоделью аналогична связи между метамоделью и моделью.

Так, например, Meta Object Facility (MOF) является моделью для метамоделей языка UML и IDL (см. рис. 1, M3 layer и M2 layer соответственно), и потому называется мета-метамоделью. На уровне моделей (M1) находятся известные нам модели языка UML, а также интерфейсы языка IDL. Уровень M0 содержит непосредственную пользовательскую реализацию какой-

* Работа частично поддержана грантом РФФИ № 04-01-00272.

либо системы [3]. Это так называемая классическая четырехуровневая архитектура метаданных. Она имеет ряд следующих преимуществ:

- позволяет связывать разные типы метаданных;
- позволяет добавлять метамодели и новые типы метаданных;
- может поддерживать взаимодействие произвольных моделей и метамоделей с одной и той же мета-метамоделью.

Уровней абстракций (метауровней) может быть много. MOF позволяет на своей основе определять новые метамодели для новых языков моделирования.

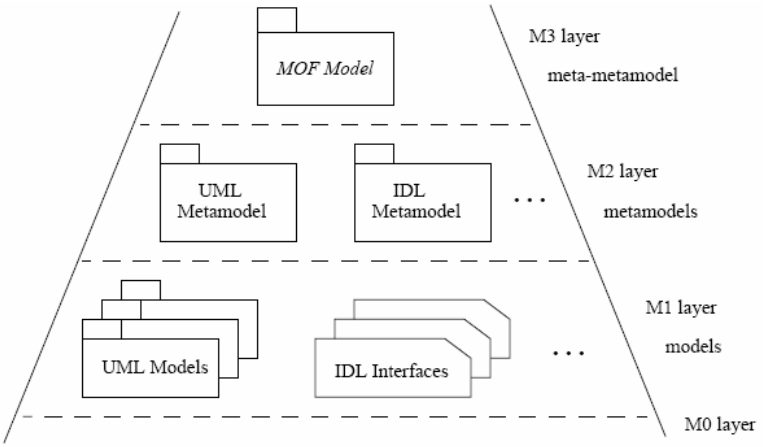


Рис. 1. Архитектура метаданных MOF

UML

Унифицированный язык моделирования (Unified Modeling Language) является фактически стандартом при проектировании программных систем. С помощью UML можно визуализировать, специфицировать, конструировать и документировать артефакты создаваемых систем.

Концептуальная модель UML состоит из трех основных блоков.

- Сущности.
- Отношения.
- Диаграммы.

Рассмотрим краткое описание каждой из них.

Сущность — это абстракция, являющаяся основным элементом модели. Отношения связывают различные сущности системы, а диаграммы группируют совокупности сущностей.

В UML существуют четыре вида сущностей:

- структурные сущности — статические части модели, соответствующие концептуальным или статическим элементам системы;
- поведенческие сущности — динамические составляющие языка;
- группирующие сущности — являются организующими частями модели UML;
- аннотационные сущности — пояснительные части модели.

Отношение — это семантическая связь между элементами модели языка.

Более детальное рассмотрение сущностей и отношений языка UML выходит за рамки данного обзора.

Диаграмма — это графическое представление набора элементов, изображаемое в виде связного графа с вершинами-сущностями и ребрами-отношениями. Диаграммы рисуют для представления системы с различных точек зрения.

В UML насчитывается девять видов диаграмм.

1. *Диаграмма вариантов использования* (use case diagram) — иллюстрирует статический вид системы с точки зрения вариантов использования. Вариант использования — это описание вариантов последовательностей действий, которые может выполнять система при взаимодействии с пользователем системы (актером).
2. *Диаграмма классов* (class diagram) — иллюстрирует набор статических элементов модели, таких как классы и типы, а также их отношения. Используется для представления статического вида системы с точки зрения проектирования.
3. *Диаграмма компонентов* (component diagram) — иллюстрирует множество компонентов и отношения между ними. Используется для представления статического вида системы с точки зрения реализации.
4. *Диаграмма развертывания* (deployment diagram) — иллюстрирует узлы и отношения между ними. Используется для представления статического вида системы с точки зрения развертывания.
5. *Диаграмма объектов* (object diagram) — иллюстрирует множество объектов и отношений между ними в заданный момент времени.

6. *Диаграмма состояний* (state diagram) — иллюстрирует автомат, содержащий состояния, переходы, события и действия, происходящие с объектом в течение его жизненного цикла.
7. *Диаграмма деятельности* (activity diagram) — иллюстрирует переходы от одного действия к другому, происходящие с объектом в течение его жизненного цикла.
8. *Диаграмма последовательности действий* (sequence diagram) — иллюстрирует взаимодействие объектов посредством посланных и принятых сообщений. При этом сообщения упорядочены по времени.
9. *Диаграмма взаимодействия* (collaboration diagram) — заостряет внимание на структурной организации объектов, принимающих или отправляющих сообщения. Данная диаграмма иллюстрирует ту же информацию, что и диаграмма последовательности действий, и потому может быть преобразована в последнюю без потери смысла.

В UML существует три механизма расширения языка.

- *Стереотип* (*stereotype*) — это расширение UML, позволяющее создавать новые элементы модели, производные от существующих, но специфичные для конкретной задачи. В UML существует ряд предопределенных стереотипов, другие же могут быть заданы пользователем.
- *Тегированное значение* (*tagged value*) — явное описание свойства как пары «имя-значение». Имя тегированного значения называют *тегом*. В UML существует ряд предопределенных тегированных значений, другие же могут быть заданы пользователем.
- *Ограничение* (*constraint*) — семантическое условие или ограничение. В UML существует ряд предопределенных ограничений, другие же могут быть заданы пользователем.

OCL

Язык ограничения объектов (Object Constraint Language) — это формальный язык, используемый для описания ограничений, накладываемых на объекты моделей языка UML [1]. При этом результат проверки инвариантов ограничений, специфицированных в системе, не производит никакого эффекта на саму систему, т.е. состояние системы до и после проверки неизменно.

Язык OCL использован в семантике UML при описании метаклассов метамодели языка. Применение OCL в диаграммах языка UML обусловлено необходимостью более детальной формальной спецификации элементов моделей. Здесь он может быть использован для следующих целей:

- определение инвариантов на классах и типах диаграммы классов;
- определение инвариантов типов для стереотипов;
- описание пред- и пост условий, накладываемых на операции и методы;
- описание охранных условий (guards);
- в качестве навигационного языка;
- определение ограничений, накладываемых на операции.

Детальное описание языка OCL выходит за рамки данного обзора.

2. ПОДХОДЫ К СПЕЦИФИКАЦИИ ЯЗЫКОВ UML И OCL

В научной литературе упоминается множество подходов к формализации языков UML и OCL. В приведенных ниже главах мы рассмотрим некоторые из подходов по формализации их моделей и метамоделей.

Стоит также отметить, что, в спецификации UML существует еще множество огрехов и противоречий. Так в работе G. Genova, J. Llorens и V. Quintana [4] рассматриваются отношения включения и расширения (include and extend relationships) для диаграммы вариантов использования.

- В спецификации присутствует противоречивое описание этих отношений. С одной стороны, оба отношения специфицируются как стереотипы над отношением зависимости (dependency). С другой стороны, в описании нотации отношений включения и расширения нет никакого упоминания об отношении зависимости [1].
  - *Отношение расширения между вариантами использования иллюстрируется как пунктирная линия с открытой стрелкой, направленной от расширяющего варианта использования к базовому. Стрелка помечается ключевым словом «extend».*
  - *Отношение включения между вариантами использования иллюстрируется как пунктирная линия с открытой стрелкой, направленной от базового варианта использования к включаемому. Стрелка помечается ключевым словом «include».*

  Наконец, метаклассы расширения и включения наследуются напрямую от метакласса отношения (рис. 2).

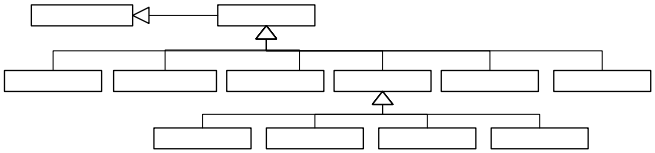


Рис. 2. Метакласс «отношение» и его подтипы

Все становиться на свои места, если отношения расширения и включения наследуются от отношения зависимости.

- Авторы критикуют нотацию языка, описывающую отношение расширения как пунктирную линию с открытой стрелкой от расширяющего варианта использования к базовому. По их мнению, стрелка должна быть направлена в противоположную сторону. Также утверждается, что отношения включения и расширения, по сути, являются одним и тем же отношением, только для расширения должно выполняться какое-либо условие, а для включения это условие всегда верно.

  Однако существует одно серьезное различие между включением и расширением. В отношении включения включаемый вариант использования вставляется в одно место базового варианта использования и представляет инкапсулированное поведение, т.е. происходит вызов подпрограммы.

  В то же время в отношении расширения разные фрагменты поведения вставляются одновременно в разные части базового варианта использования, т.е. происходит расслоение, и, следовательно, механизм инкапсулированного поведения не предоставляется.
- Варианты использования, которые включаются или расширяют какие-либо другие варианты использования, по мнению авторов, не являются ими (вариантами использования) как таковыми. Так, согласно спецификации: *вариант использования специфицирует поведение системы и представляет собой описание множества последовательностей действий, выполняемых системой для получения актером какого-либо результата*. Но включаемые и расширяющие варианты использования могут специфицировать только часть поведения системы и производить только часть результата, получаемого актером.

Другой пример незавершенности спецификации UML приводиться в работе M. Gogolla и B. Henderson-Sellera [5]. В ней анализируются стандартные стереотипы метамодели языка UML. По мнению авторов, зачастую стереотипы используются исключительно в информативных целях и, как следствие, механизм работы стереотипов, подразумевающий конкретизацию, расширение (метамодель языка) и ограничение (с помощью OCL) элементов моделей, совершенно не используется.

В качестве примера рассматривается метамодель, описывающая абстрактный синтаксис стереотипа (рис. 3) и его семантику. В тоже время в спецификации языка дано некорректное определение тегированного значения, являющегося частью спецификации стереотипа.

Любое тегированное значение должно иметь либо одно или несколько соединений с ссылочным значением, либо одно или несколько информационных значений, но не то и другое одновременно.

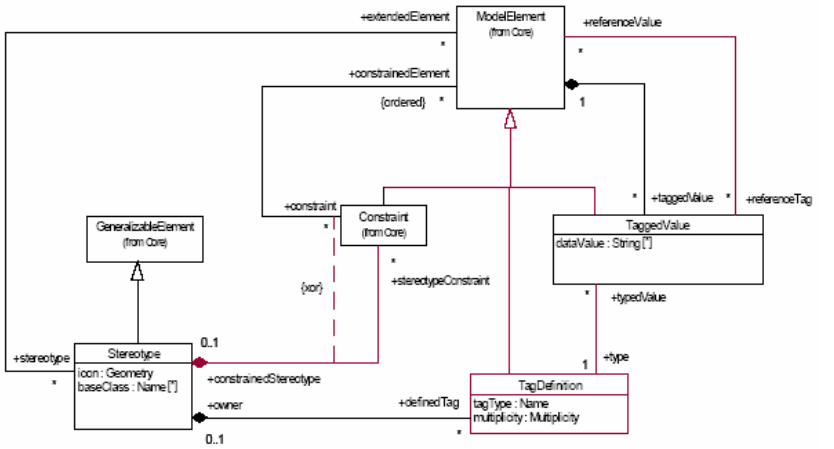


Рис. 3. Абстрактный синтаксис стереотипа

Это означает, что в диаграмме, изображенной на рис. 3, возможно пропущено xor-ограничение между ассоциациями с именами referenceValue-referenceTag и modelElement-taggedValue (modelElement — это неявное имя безымянного конца ассоциации). Кроме того, у ассоциации modelElement-taggedValue на стороне ModelElement неправильно указана множественность. Вместо 1 должно стоять значение 0..1. Для большей наглядности метамодели на стороне ModelElement желательно задать имя ассоциации dataValue.

Для более детального рассмотрения предопределенных стереотипов языка UML в работе рассматривается общая нотация стереотипа (рис. 4).

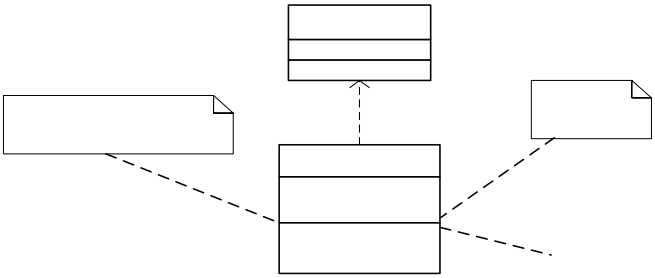


Рис. 4. Нотация стереотипа

Как видно из диаграммы, стереотип должен или может содержать следующие элементы: обязательное имя стереотипа (stereotypeName), обязательный базовый метакласс (baseClass), ноль или более факультативных определений тегов стереотипа (tagDefenition), ноль или более факультативных ограничений стереотипа (stereotypeConstraint) (эти ограничения автоматически влияют на элементы модели, к которым применяется данный стереотип), ноль или более факультативных ограничений применимости стереотипа (applicabilityConstraint), а также факультативные комментарий и графическое обозначение.

На рис. 5 изображена декларация стереотипа <<important>> согласно спецификации UML.

Рис. 5. Декларация стереотипа <<important>> согласно спецификации UML

{applicabilityConstra

Из нотации видно, что в UML существуют стереотипы без тегов и ограничений. Этот стереотип используется для пометки важных элементов модели, на которые следует обратить внимание при разработке. Синтаксически описание данного стереотипа верно, но базовым классом для стереотипа является ModelElement, самый верхний класс в метаиерархии UML, а это означает, что данный стереотип может быть применен к любому элементу модели, что в некоторых случаях неправильно. Самым простым решением может стать ограничение, гарантирующее, что стереотип может быть описан только для подклассов ModelElement.

Context Stereotype forbidModelElementAsBaseClass:
baseClass <> "Foundation::Core::ModelElement"

Стереотипы могут также использоваться во время разработки моделируемой системы для проверки модели на специфичные свойства. Конечно, для этого программное обеспечение, посредством которого ведется моделирование, должно поддерживать указанные выше функции.

На рис. 6 изображена декларация стереотипа <<statesHaveTransitions>> согласно спецификации UML.

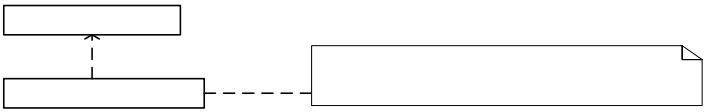


Рис. 6. Декларация стереотипа <<statesHaveTransitions>> согласно спецификации UML

Стереотип накладывает ограничение такое, что диаграмма состояний должна содержать состояния, в которых нет ни одного входящего и исходящего перехода. Можно заметить, что ограничение, накладываемое стереотипом <<statesHaveTransitions>>, задано неформально. Эквивалентное формальное ограничение на языке OCL выглядит следующим образом:

self.extendedElement->forAll(*sm*|
sm.oclKindOf(*StateMachine*) *and*
sm.oclAsType(*StateMachine*).*top.oclKindOf*(*CompositeState*) *and*
sm.oclAsType(*StateMachine*).*top.oclAsType*(*CompositeState*).*subvertex ->*
forAll(*sv*|*sv.incoming -> notEmpty or sv.outgoing -> notEmtpy*)

Программное обеспечение, посредством которого ведется моделирование, могло бы использовать для информирования пользователя следующую функцию, выдающую названия состояний, в которых нет ни одного входящего и исходящего перехода.

```
Context StateMachine namesOfUnconnectedStates():Bag(String)
    self.top.oclAsType(CompositeState).subvertex ->
        select(sv|sv.incoming -> isEmpty and sv.outgoing -> isEmpty) ->
            collect(sv|sv.name)
```

A. Naumenko и A. Wegmann в работе [6] предлагают воспользоваться RM-ODP (The Reference Model of Open Distributed Processing) [7] для решения проблем языка UML. В спецификации UML модель RM-ODP упоминается как структура, оказывающая непосредственное влияние на архитектуру метамодели языка (раздел Preface: Relationships to Other Models [1]). Кроме того, RM-ODP используется в MOF [3] для управления типами. В статье идентифицируются три проблемы метамодели UML и предлагаются их решения на базе RM-ODP.

Первая проблема — это структурный хаос семантики языка. По словам авторов, это «высокая техничность, немногословность и сложность для понимания новичком».

Решение данной проблемы видится авторам в использовании структуры RM-ODP и теории типов Рассела [8]. Приведем пару определений из структуры RM-ODP и теории Рассела.

Множество рассуждений (*universe of discourse*) — это предмет моделирования.

Частное (*individual*) — это нечто лишенное комплексности.

Предложение (*proposition*) — это нечто комплексное.

Предложение первого порядка (*first-order proposition*) — это элементарное предложение, содержащее в виде связанных переменных (apparent variable) некоторое множество частных.

Предложение высокого порядка (*higher-order proposition*), например, N — это предложение, содержащее в виде связанных переменных некоторое множество предложений низшего порядка (N-1).

На рис. 7 изображена диаграмма метамодели языка, полученная в результате применения данного подхода.

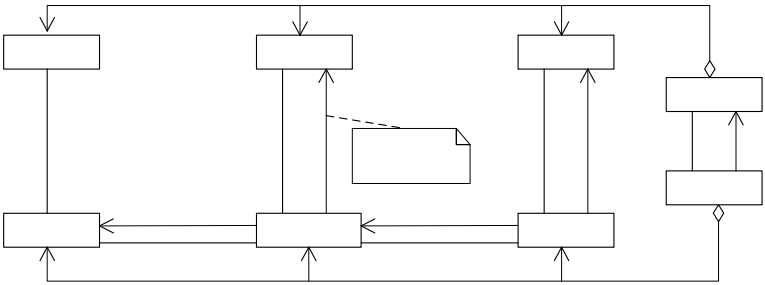


Рис. 7. Диаграмма метамодели языка после применения подхода

Главный элемент метамодели UML ModelElement сопоставлен здесь частному из теории Рассела. Базовые концепции моделирования (Basic Modeling Concept) сопоставлены предложению первого порядка с помощью формальной декларативной семантики Тарского [9], а концепции спецификации (Specification Concept) — предложению высокого порядка.

Вторая проблема спецификации UML, упоминающаяся в статье, это отсутствие декларативной семантики. Семантика языка противоречива в описании отношений между моделями, построенными с использованием языка и предметов моделирования. Решение этой проблемы основывается на приведенном выше подходе и заключается в реализации базовой концепции моделирования (Basic Modeling Concept). Рис. 8 иллюстрирует общую организацию базовой концепции моделирования.

Разработчик представляет в модели сущности из множества рассуждений и позже добавляет в нее сущности, не относящиеся к пространственно-временному измерению. Континуумы пространства и времени порождают еще один континуум информации о структуре модели внутри пространства и времени.

При определении границ по координатам «пространство-время» и «структура модели» мы получаем концепции пространственного интервала (space interval), временного интервала (time interval) для пространства и времени, а также концепцию объекта (object) и концепцию окружения (environment) объекта для координаты «структура модели». Концепции объекта и его окружения можно рассматривать следующим образом:

- в определенный момент времени и следовательно определить понятие статического информационного элемента (состояния) внутри информационного континуума (information continuum);
- в интервале между двумя моментами времени и следовательно определить понятие динамического информационного элемента (действия) внутри информационного континуума.

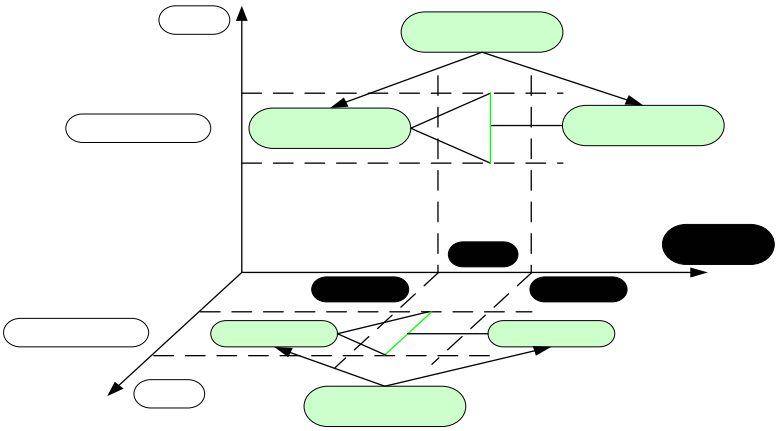


Рис. 8. Организация базовой концепции моделирования

Рис. 9 изображает диаграмму базовой концепции моделирования.

Третьей проблемой является недостаточное теоретическое обоснование используемой метамодели языка UML. Предлагаемая в статье концепция моделирования на основе RM-ODP, теории Рассела и теории Тарского полностью теоретически обоснована.

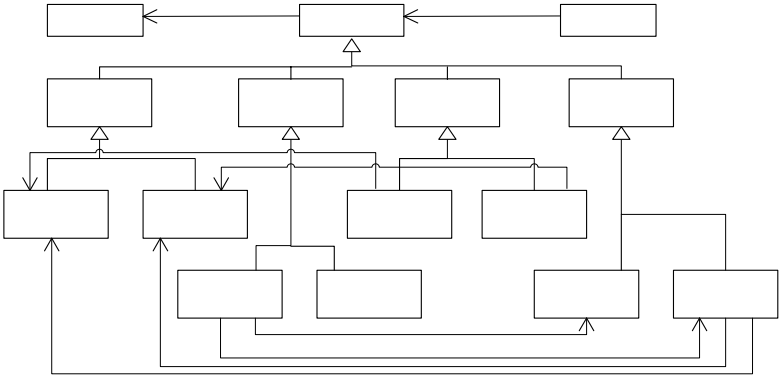


Рис. 9. Диаграмма базовой концепции моделирования

Формализация модели и метамодели языка UML

Многие научные исследования, посвященные формализации модели и метамодели языка UML, основываются не на самом UML, а на некотором его подмножестве: формальном и строго структурированном.

В работе R. Paige и J. Ostroff [10] идет речь о BON (Business Object Notation) [11] и PVS [12]. BON — это объектно-ориентированный язык моделирования, который по сути повторяет собой диаграмму классов UML. PVS — это язык спецификаций, разработанный для автоматического анализа метамоделей языков моделирования.

Рассмотрим основные компоненты BON. Фундаментальным понятием языка BON является понятие класса. Класс имеет имя, факультативный инвариант класса, ноль или более функций (feature). Функция может быть запросом (query) и при этом возвращать некоторое значение без воздействия на состояние системы или командой (command) и при этом менять состояние системы и ничего не возвращать. Функции могут факультативно иметь ограничения, написанные на языке утверждений BON в форме пре- и пост условий. Рис. 10(а) содержит иллюстрацию класса CITIZEN.

Specification
Concept

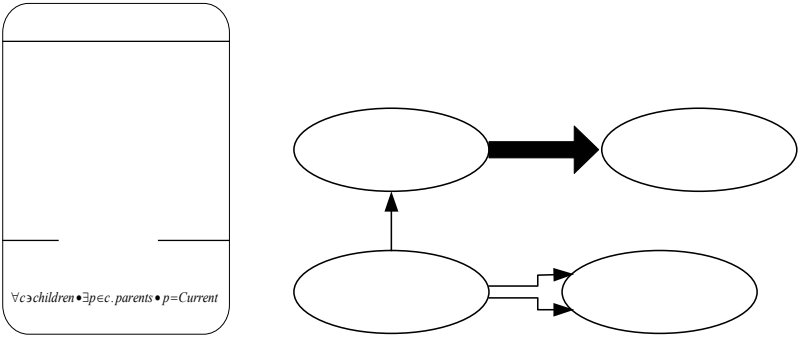
0..1 hig

Discontinuity
Concept

timeLimit

<<concept>>
Point in Time

<<con
Point in



(а) (б)

Рис. 10. Синтаксис BON для классов и отношений

В пост условии словосочетание **old** используется для ссылки на значение выражения до вызова функции, переменная **Result** – для спецификации возвращаемых запросом значений.

Инвариант класса — это утверждение, которое должно быть верно всякий раз, когда экземпляр класса используется другим объектом. В тексте инварианта под переменной **Current** подразумевается ссылка на текущий объект.

Классы и функции могут быть помечены ограниченным набором стереотипов.

- Символ «*», следующий за именем класса, указывает на то, что данный класс отложенный (deferred) и часть его функций не имеет реализации, потому класс не может быть конкретизирован.
- Символ «+», следующий за именем класса, указывает на то, что данный класс может быть конкретизирован.
- Символ «++», следующий за именем функции, указывает на то, что данная функция переопределена.

Модели BON обычно состоят из классов, собранных в кластеры, которые взаимодействуют посредством отношений двух типов. Кластер изображается прямоугольником со скругленными краями, нарисованным пунктирной линией.

- Наследование изображается стрелкой от потомка к родителю (на рис. 10(б) отношение между CHILD и ANCESTOR).
- Отношение «клиент-поставщик» состоит из двух подтипов, отношения агрегации и отношения ассоциации. Отношения агрегации и ассоциации изображаются стрелкой, направленной от клиента к

поставщику. При использовании отношения ассоциации клиентский класс имеет ссылку на объект класса поставщика, а при использовании отношения агрегации клиентский класс содержит объект класса поставщика. На рис. 10(b) отношение агрегации изображено между классами CHILD и SUPPLIER2, тогда как отношение ассоциации изображено между классами ANCESTOR и SUPPLIER1.

После рассмотрения основных компонент языка авторы предлагают к рассмотрению метамодель языка, определенную в терминах BON. Модель BON — это экземпляр класса MODEL (рис. 11). Каждая модель имеет множество абстракций. На рис. 12 изображен интерфейс класса MODEL.

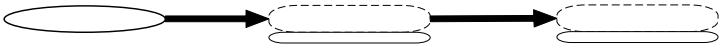


Рис. 11. Метамодель BON

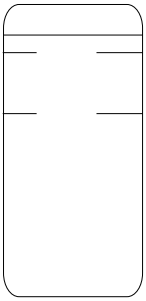


Рис. 12. Интерфейс класса MODEL

На рис. 13 изображена иерархия отношений языка, а на рис. 14 — кластер абстракции. Абстракции классифицируются на два типа: статические и динамические. Статические абстракции — это классы и кластеры, а динамические — это объекты и кластеры объектов. Кластеры могут содержать

только абстракции своего типа, т.е. статические кластеры могут содержать только статические абстракции.

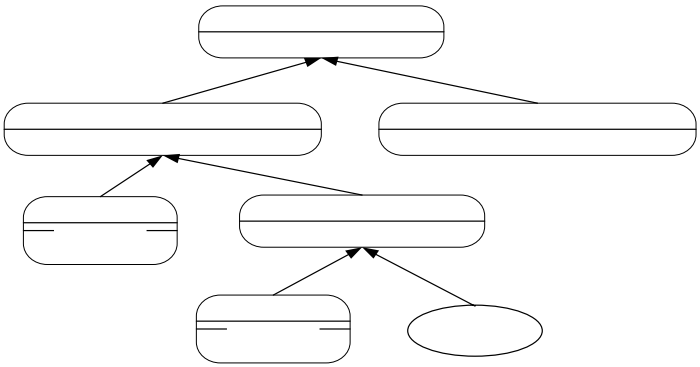
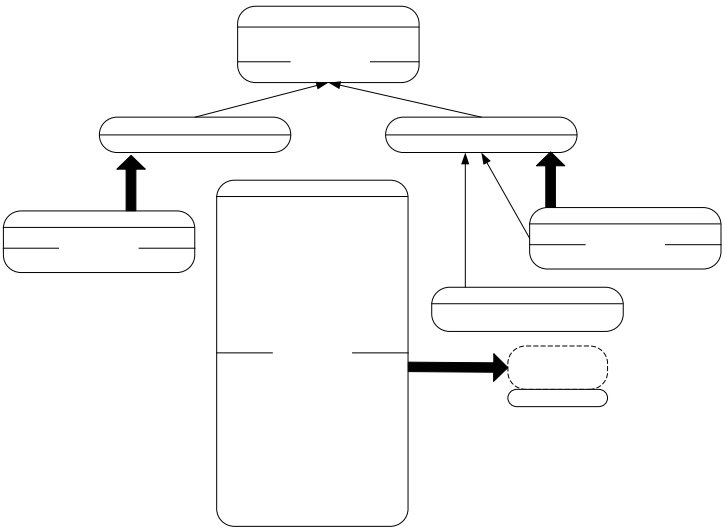


Рис. 13. Кластер отношений



STATIC_F

source++, target++;

Рис. 14. Кластер абстракции

На рис. 15 изображен кластер функций.

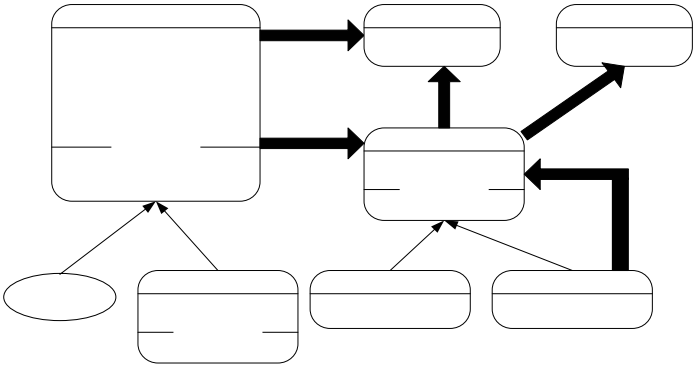


Рис. 15. Кластер функций

С помощью языка спецификаций PVS можно формально описать представленную выше метамодель языка BON. Так, например, формальное описание абстракций классов, кластеров, объектов и кластеров объектов на языке PVS выглядит следующим образом:

```
abs_names: THEORY
BEGIN
    ABS: TYPE+

    %Статические и динамические абстракции
    STATIC_ABS, DYN_ABS: TYPE+ FROM ABS

    %Конкретизируемые абстракции
    CLASS, CLUSTER: TYPE+ FROM STATIC_ABS
    OBJECT, OBJECT_CLUSTER: TYPE+ FROM DYN_ABS
END abs_names
```

Формальное описание отношений состоит в следующем:

FEATURE*
pre: SINGLE_STATE_AS
post: DOUBLE_STATE_
deferred, effective,
redefined: BOOLEAN
name: STRING
accessors: SET[CLASS]
rename(s:STRING)
frame:SET[QUERY]
Invariant

```
rel_names: THEORY
BEGIN
    %Абстрактное понятие отношения
    REL: TYPE+

    % Конкретизируемые отношения
    INH, C_S, MESSAGE: TYPE+ FROM REL
    AGG, ACCOS: TYPE + FROM C_S
END rel_names
```

Надо заметить, что, кроме формального описания метамодели языка BON, с помощью PVS можно задать ограничения, накладываемые на метамодель. Проиллюстрируем это с помощью ограничения, накладываемого на отношение наследования. То, что абстрактный элемент не может быть наследован от самого себя, указывается следующим образом:

```
Inh_ax: AXIOM (FORALL (i:INH): NOT (inh_source(i)-inh_target(i)))
```

Формально метамодель языка BON (рис. 11) представляется следующим образом:

```
metamodel: THEORY
BEGIN
    IMPORTING abstractions, relationships

    %Модель BON состоит из множества абстракций
     abs: SET[ABS]
     rels: SET[REL]
     …
```

Таким образом, результатом работы является полная формальная спецификация метамодели объектно-ориентированного языка моделирования в форме, употребимой для автоматического анализа. Однако BON в сравнении с UML более формализован и «подогнан» под условия решаемой задачи.

Схожий подход использован в работе G. Overgaard [13]. В качестве концепции для формализации языков моделирования выбран Boom [14]. Boom состоит из метамодели и языка формальных спецификаций Odal. Odal — это простой, строго типизированный объектно-ориентированный

язык с семантикой, определенной с помощью пи-исчисления [15], который используется для определения классов в метамодели Boom.

В Boom применяется принцип локализации (localization principle), который заключается в разделении семантики отношений и других конструкций языка моделирования.

Еще одной отличительной чертой Boom является то, что он сводит все метауровни в одну модель, вследствие чего разные метауровни не разделены на разные модели.

Ядро Boom содержит около 30 классов для определения конструкций класса, бинарных отношений, экземпляров и связей. При этом каждый класс определяет абстрактный синтаксис, а также статическую и динамическую семантику конструкции. Кроме того, Boom содержит классы для задания правил о том, как разные конструкции могут взаимодействовать друг с другом. Структура Boom позволяет добавлять новые классы, которые обычно базируются на уже существующих, переопределяя или расширяя их.

В работе рассматривается конструкция класса Boom. Она описывается с помощью класса Declaration, содержащего набор свойств, а также имеющего возможность создавать экземпляры класса. Свойства могут быть структурными, например атрибутами и ассоциациями, или поведенческими, например операциями и методами. Для описания структурных свойств в Boom используется класс Connection, а для поведенческих — класс Operation. Объекты описываются с помощью класса Instance.

Так как Boom описывает все метауровни в одной модели, то класс Declaration может быть экземпляром самого себя, следовательно, Declaration — это подкласс класса Instance. Все вышесказанное записывается с помощью Odal в следующем виде:

```
Class Instance
    Variables
        Origin : Declaration,
        Links : Link*
    Invariant
        Origin <> NULL
    Methods
        initialize(o : Declaration, c : Connection) : instance
            origin : = 0;
            FOREACH con IN c DO
                Links ADD con createLink()
```

```
            entityType() : EntityType
                origin instanceKind()
        . . .

    Class Declaration superclass Instance
        Variables
            operation : Operation*,
            connection : Connection*,
            instanceType : EntityType,
            name : Name
        Invariant
            name <> NULL AND instanceType <> NULL
        Methods
            addConnection(c : Connection) : Boolean
        . . .
```

Класс Connection определяет направленное отношение между классами Declaration, а класс Link определяет направленное отношение между классами Instance. Экземпляр класса Link между двумя классами Instance существует тогда и только тогда, когда существует экземпляр класса Connection между двумя классами Declaration. Все вышесказанное записывается с помощью Odal в следующем виде:

```
Class Connection
    Variables
        name : Name,
        target : Declaration,
        instanceType : ConnectionType
    Invariant
        target <> NULL AND name <> NULL AND instanceType <> NULL
    Methods
        target(c : Declaration)
            target
        connectionType() : ConnectionType
            instanceType
        createLink() : Link
            Link NEW (SELF)
    . . .
```

Class Link
Variables
origin : Connection,
value : Instance
Invariant
origin <> *NULL*
Methods
connectionType() *:* ConnectionType
origin *connectionType*()
. . .

Структура Boom позволяет производить сопоставление классов Boom сущностям языка моделирования. Для этого используются три класса EntityType, ConnectionType и LinkType. Класс EntityType образует пару с классом Instance, а классы ConnectionType и LinkType образуют пары с классами Connection и Link соответственно.

Например, предположим, что конструкция Class имеет семантику, определенную классом Declaration, а конструкция Object определена классом Instance. Кроме того, предположим, что каждый объект может иметь множество указателей на другие объекты. Для этого язык моделирования содержит конструкции Pointer и PointerDeclaration. В структуре Boom этим двум классам соответствуют классы Link и Connection. Сопоставление этих конструкций с помощью языка Odal приведено ниже.

mapping ADD (*EntityType NEW* (*CLASS, Declration*))*;*
mapping ADD (*EntityType NEW* (*OBJECT, Instance*))*;*
mapping ADD (*ConnectionType NEW* (*POINTERDECLARATION, Connection*))*;*
mapping ADD (*LinkType NEW* (*POINTER, Link*))*;*

В структуре Boom используется явное задание множества правил ассоциаций, чтобы всегда знать, какие типы связей поддерживаются между конструкциями. Каждое правило описывается кортежем имен сущностей и связей: <тип начала, тип конца, тип связи>.

rulset ADD (*AssociationRule new* (*CLASS, CLASS, POINTERDECLARATION*))*;*

Продемонстрируем формальную спецификацию некоторых конструкций языка UML с помощью структуры Boom. Начнем с конструкции клас-

са. На рис. 16 изображена метамодель конструкций класса и атрибута UML, а ниже приведена ее спецификация на языке Odal.

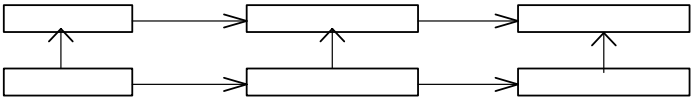


Рис. 16. Метамодель конструкций класса и атрибута UML

mapping ADD (*EntityType NEW* (*CLASS, Declaration*))*;*
mapping ADD (*EntityType NEW* (*DATATYPE, Declaration*))*;*
mapping ADD (*EntityType NEW* (*OBJECT, Instance*))*;*
mapping ADD (*EntityType NEW* (*DATAVALUE, TokenInstance*))*;*
mapping ADD (*ConnectionType NEW* (*ATTRIBUTE, MultiplicityConnection*))*;*
mapping ADD (*LinkType NEW* (*ATTRIBUTELINK, NumberedLink*))*;*
rulset ADD (*AssociationRule new* (*CLASS, DATATYPE, ATTRIBUTE*))*;*

Надо заметить, что в конструкции класса формализуется только часть спецификации, так например, автомат состояний (state machine) в описании отсутствует.

Ассоциация UML также может быть специфицирована с помощью структуры Boom. На рис. 17 изображена метамодель конструкции ассоциации UML, а ниже приведена ее спецификация на языке Odal.

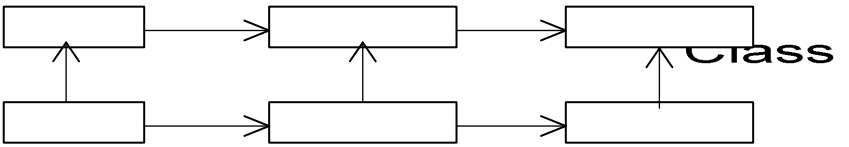


Рис. 17. Метамодель конструкции ассоциации UML

mapping ADD (*EntityType NEW* (*ASSOCIATION, AssociationDeclaration*));
mapping ADD (*EntityType NEW* (*LINK, Instance*));
mapping ADD (*ConnectionType NEW* (*ASSOCIATIONEND, MultiplicityConnection*));
mapping ADD (*LinkType NEW* (*LINKED, SingleLink*));
rulset ADD (*AssociationRule new* (*ASSOCIATION, CLASS, ASSOCIATIONEND*));

Итогом работы является демонстрация применимости подхода к формальной спецификации языка UML. Подход ограничен представлением всех метауровней моделируемой системы в одной модели, что при большом объеме моделируемой системы может стать проблемой.

В работе T. Clark, A. Evans и S. Kent [16] предложена формализация MML (The Metamodelling Language), являющегося подмножеством UML и предложенного авторами в качестве базы для UML версии 2.0. Формализация MML ведется с помощью MML-исчисления. Основой MML-исчисления является пси-исчисление Cardelli и Abadi [17]. Семантика, описанная в статье, является повелительной в отличие от декларативной неформальной семантики языка UML.

Авторы статьи входят в precise UML group [10] — международную группу исследователей и практиков, занимающихся доработкой языка UML как точного (формального) языка моделирования. Отличительной особенностью подхода группы является следование спецификации языка UML. Введение новых механизмов и методов возможно, только, если в спецификации обнаруживается концептуальная проблема. Группа концентрирует свое внимание на ядре языка, как на фундаменте формализации.

На рис. 18 изображена модель MML.

Ниже приводится описание с помощью MML-исчисления следующих конструкций и элементов языка: объектов, методов, вызовов методов, типов, классов, OCL-выражений.

Объекты и атомарные выражения описываются следующим образом:

m,l ::=	*MML выражение*
v, w	*переменные*
n	*integer*
b	*boolean*
s	*string*
m.v : = l	*модификация поля*
Seq{}	*пустая последовательность*
Seq{l|m}	*парное выражение*
let v=a in b end	*локальное определение*

if e then a else b endif	*условное выражение*
$\oplus m_i^{i\in[1,n]}$	*оператор*
$Set\{m_i\}^{i\in[o,n]}$	*множество*

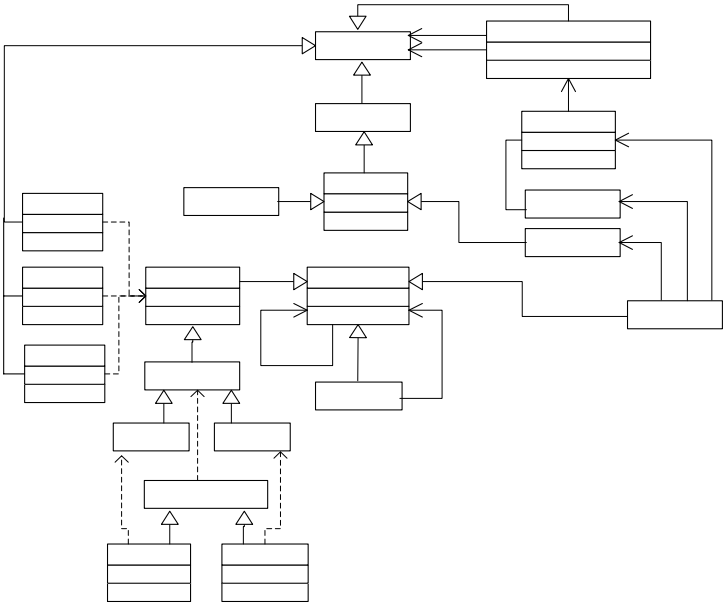


Рис. 18. Модель MML

Все значения данных MML имеют поле ‘of’, содержащее классификатор (classifier) данного значения.

Методы в MML — это параметризованные OCL-выражения, чьи значения вычисляются в момент посылки сообщения объекту. Ниже приведено описание метода этого языка.

m,l ::= ... как и ранее	*MML выражение*
meth (*v**) *m end*	*метод*

meth (*vz ... vn*) *m end* = *fun* (*self, v1, ..., vn*) *m end*

Вызов метода в MML происходит при выполнении выражения посылки (send expression), которое имеет следующий синтаксис:

l,m	$::=$ *... как и ранее*	*MML expression*
	$mv(l_i^{i\in[0,n]})$	*send expression*

где m — цель, v — имя метода, l_i — аргумент.

В языке MML у класса Classifier есть два подкласса: Class, DataType. Экземпляры класса DataType описывают необъектные значения данных. Синтаксис класса описывается следующим образом:

l,m	$::=$ *... как и ранее* $\mid c \mid k$	*MML выражение*
c	$::=$ *class* $v\mu^?\pi^?(\alpha \mid \nu \mid \sigma)^* i^?$	*определение класса*
μ	$::=$ *metaclass m*	*метакласс*
π	$::=$ *extends* $m(,m)^*$	*родители*
α	$::= v : \tau$	*атрибут*
ν	$::= v(\delta^?(,\delta)^*)m$	*метод*
σ	$::= v = m$	*слот*
δ	$::= v : \tau$	*описание*
τ	$::=$	*тип*
	v	*имя типа*
	$Set(\tau)$	*набор типа*
	$Seq(\tau)$	*последовательность типа*
i	$::=$ *inv*$(sm)^*$	*инвариант*

Класс — это объект с собственным классификатором. Классификатор указывается в выражении класса в факультативном пункте metaclass, а если такого пункта нет в описании, то по умолчанию классификатором считается Class. Класс может иметь несколько родителей. Родители описываются в факультативном пункте parents, а если такого пункта нет в описании, то по умолчанию класс имеет единственного родителя Object. В пункте definitions описываются методы, атрибуты и слоты (slot) класса. Конструкция «слот» используется в языке при работе с нестандартными метаклассами. Детальное рассмотрение особенностей трансляции MML-класса в MML-объект может быть найдено в [16].

Корневой элемент иерархии OCL-выражений — это класс Exp (рис. 18). Каждая новая синтаксическая категория вызывает создание нового ме-

тода для соответствующего классификатора и трансляцию OCL-выражения в вызов соответствующего метода. Приведем синтаксис для операций над коллекцией.

l,m ::= ... как и ранее	*MML выражение*
m -> v(*m*(,*m*)*)?	*операция коллекции*
m -> v(*w* | *l*)	*кванторная операция*
m -> iterate(*vm = m* | *m*)	*перебор*

Операции коллекции — это size, includes (элемент), count (вхождение элемента), includesAll, isEmpty, notEmpty, sum (элементов). Кванторные (quantified) операции — это exists и forAll. CollectionOfInstance — это корневой класс иерархии классов коллекций. Указанные операции можно разбить на две категории: те, которые вводят новые переменные (exists, forAll, iterate), и те, которые нет. Выражения, не вводящие новых переменных, транслируются в выражения посылки MML. Например, m->includes(l) становиться m.includes(l) со следующей реализацией:

CollectionOfInstances:includes(*o : Instance*) *: Boolean*
self -> count(*o*) *> 0*
CollectionOfInstances:count(*o : Instance*) *: Integer*
self -> iterate (*v1 v2 = 0* | *if v1.equals*(*o*) *then v2 + 1 else v2 endif*)

Выражения, вводящие новые переменные, транслируются в выражения посылки MML, которые связывают переменные с одной или несколькими функциями. Функции — это объекты в MML и потому могут быть аргументами методов.

Данная работа отличается от двух рассмотренных выше непосредственной близостью к языку UML.

K. Lellahi в работе [18] демонстрирует применимость алгебраического метода для формального описания ER-диаграмм, являющихся аналогом диаграмм классов UML.

В ER-диаграммах существуют понятия концептуального и физических уровней (conceptual, physical level). Концептуальный уровень — это уровень метаданных диаграммы, а физический уровень — это уровень непосредственных «физических» данных в какой-то определенный момент времени. Граница между двумя этими понятиями очень размыта. Автор полагает, что алгебраический метод, описанный в его работе, служит как раз для четкого определения этой границы.

Моделирование данных посредством ER-диаграмм подчиняется некоторому множеству правил, называемых *метаправилами* (meta-rules). Метаправила описывают общие концепции и не зависят от области применения. Например, общими концепциями в ER-модели являются сущность (entity), атрибут (attribute), отношение (relationship), мощность (cardinality) и ярлык (label), называемый ролью. Метаправила для такой модели формулируются следующим образом.

1. Каждый тип сущности, участвующий в отношении, должен сопровождаться min/max мощностью и ролью (факультативно).
2. Каждый атрибут имеет описание, специфицирующее его тип.
3. Некоторые атрибуты сущности могут быть описаны как ключевые (key).
4. Каждое отношение, как минимум, бинарно и если сущность участвует в отношении дважды, то соответствующие роли должны различаться.
5. Каждая сущность имеет хотя бы один атрибут.
6. Каждое имя атрибута — это имя атрибута сущности или имя атрибута отношения.
7. Одно и тоже имя атрибута не может быть использовано и в сущности, и в отношении (факультативное правило).
8. Одно и тоже имя атрибута не может быть использовано в двух разных сущностях (факультативное правило).

Пусть ATT, ENT, REL, LAB и CARD будут перечислимыми попарно не пересекаемыми множествами. Мы будем называть элементы множеств ATT, ENT, REL, LAB и CARD именем атрибута, именем типа сущности, именем типа отношения, меткой (или ролью) и мощностью соответственно. Также введем понятие множества DESC = {mono, multi, composite}, которое не пересекается с введенными выше множествами.

Приведенные выше метаправила формализуются с помощью простых математических концепций, как показано ниже.

Определение 1 (Entity-Relationship схема). Концептуальная ER-схема (или ER-схема) — это кортеж S = (d_att, desc_att, e_att, k_att, r_ent, r_att), где:

d_att : ATT -> DESC,

desc_att : ATT -> $\overline{ATT}$, где $\overline{ATT}$ = ATT $\cup$ set ATT,

e_att : ATT -> ENT,

k_att : ATT -> ENT,

r_att : ATT -> REL и

*r_ent : REL * ENT * LAB -> CARD*

частично вычислимые функции, удовлетворяющие следующим условиям (нумерация условий соответствует нумерации неформальных определений метаправил, приведенных выше):

2. $d\_att(A) \in \{\text{mono, multi}\} \Rightarrow desc\_att(A) = A,$
   $d\_att(A) = composite \Rightarrow desc\_att(A) \in \overline{ATT}$ и $A \notin desc\_att(A),$
3. $k\_att(A) \subseteq e\_att,$
4. $(R, E_1, l_1) \in def(r\_ent) \Rightarrow (\exists E_2, \exists l_2\ (R, E_2, l_2) \in def(r\_ent) \wedge ((E_1 \neq E_2) \vee (l_1 \neq l_2))),$
5. $def(e\_att) \neq 0,$
6. $e\_att \cup r\_att = def(d\_att),$
7. $e\_att \cap r\_att = 0.$

Алгебраическая спецификация ER-схемы представляется следующим образом:

Spec ER_Sch

Sorts: ENTITY, RELSHIP, ATTR, $\overline{ATTR}$, LABEL, CARDINALITY, DESCATT

Ops:

D_Att : ATTR -> DESCATT (*частичная*),

Desc_Att : ATTR -> $\overline{ATTR}$ (*частичная*),

E_Att : ATTR -> ENTITY (*частичная*),

K_Att : ATTR -> ENTITY (*частичная*),

R_Att : ATTR -> RELSHIP (*частичная*),

R_Ent : RELSHIP ENTITY LABEL -> CARDINALITY (*частичная*)

Axs:

E, E' : ENT, R : RELSHIP, A : ATTR, l, l' : LABEL,
c, c' : CARDINALITY, D : DESCATT.

$\forall$ A K_Att(A) = E $\Rightarrow$ E_Att(A) = E,

$\forall$ R $\exists$ E, $\exists$ E', $\exists$ c, $\exists$ c', $\exists$ l, $\exists$ l' (R_Ent(R, E, l) = c) $\wedge$ (R_Ent(R, E', l') = c') $\wedge$ $((E_1 \neq E_2) \vee (l_1 \neq l_2))$,

$\forall$ E $\exists$ A E_Att(A) = E,

$\neg$ ($\exists$ A $\exists$ E $\exists$ E' ((E_Att(A) = E) $\wedge$ (R_Att(A) = E'))),
$\forall$ A $\exists$ E ((E_Att(A) = E) $\vee$ (R_Att(A) = E')).

Факт 1 (схема как алгебра). Концептуальная схема, представленная Определением 1, является моделью спецификации *ER_Sch.*

Базовая интерпретация ER-схемы S заключается в сопоставлении множества |A| каждому имени атрибута A. Далее определяется понятие экземпляра схемы и мягкой семантики (loose semantics).

Определение 2 (экземпляра схемы). Экземпляр I схемы S состоит из:

- базовой интерпретации;
- множества $[\mathrm{E}]_I$ для каждого имени сущности E;
- множества $[\mathrm{R}]_I$ для каждого имени отношения R.

Определение 3 (Выполнение ограничений и свободная семантика). Пусть I будет экземпляром ER-схемы и r_ent(R, E ,l) = c. Говорят, что I удовлетворяет мощности c для R и E, если $|e_R| \in [c]$, где $|e_R|$ — это число элементов $\mathrm{r} \in [\mathrm{R}]_I$, в которых участвует e.

Говорят также, что экземпляр I для ER-схемы удовлетворяет ограничениям сущности E, если для K = k_att(E) выполняется следующее:

$$\forall e \in [E]_I, \forall e' \in [E]_I, (\forall A \in K, e.A = e'.A) \Rightarrow e = e'.$$

Корректный экземпляр схемы S — это экземпляр, удовлетворяющий всем ограничениям S. Множество всех корректных экземпляров S называется свободной семантикой.

Теперь определяется ER-метасхема. Каждый корректный экземпляр метасхемы есть схема (и наоборот). Метасхема получается формулировкой метаконцепций подобно тому, как это сделано для схемы в Определении 1.

Факт 2. META_ER = (meta_d_att, meta_desc_att, meta_k_att, meta_r_ent, meta_r_att) — это ER-схема, называемая *метасхемой* ER-модели.

Факт 3. Метасхема является корректным экземпляром самого себя.

Факт 4. Каждый корректный экземпляр S метасхемы META_ER определяет ER-схему S_I.

Важно заметить, что подход формализует не только модель системы, но и ее метамодель. Данная работа может быть применена к диаграммам языка UML, описывающим статическую структуру моделируемой систем

как, например, диаграммы классов, объектов и т.д. Более того, представляется возможным применение подхода к описанию динамических аспектов моделируемой системы.

Формальные подходы к языку OCL

Теме формализации языка OCL в научной литературе уделено не меньшее внимание, чем формальным подходам к языку UML. Рассматриваемые ниже работы посвящены созданию метамодели OCL, формальной семантике языка, а также использованию OCL в качестве языка запросов.

M. Richters и M. Gogolla в работе [19] предлагают метамодель OCL. Важной ее особенностью является интеграция с метамоделью UML. Уровень метамодели OCL совпадает с уровнем метамодели языка UML (так называемый уровень M2) и в качестве мета-метамодели используется модель MOF (уровень M3). На рис. 19 изображены отношения зависимости между компонентами языков UML и OCL.

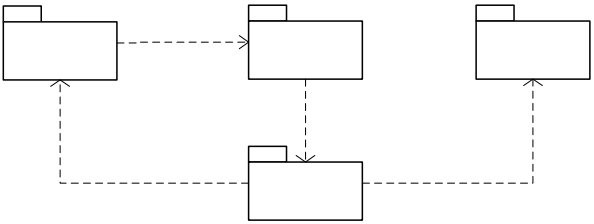


Рис. 19. Отношения зависимости между компонентами языков UML и OCL

Компонент «Ядро» (Core) зависит от компонента «Типы данных» (Data Types), где вводиться понятие выражения (expression), используемого в компоненте «Ядро» для моделирования концепции ограничения (constraint). Компонент «Типы данных» зависит от компонента OCL, так как последний используется для описания выражений. С другой стороны, компонент OCL зависит от элементов модели (классов, атрибутов и т.д.) компонента «Ядро». Наконец, результаты вычисления OCL- выражения являются значе-

ниями или сущностями, и, таким образом, мы имеем отношения зависимости между компонентом OCL и компонентом «Общее поведение» (Common Behavior).

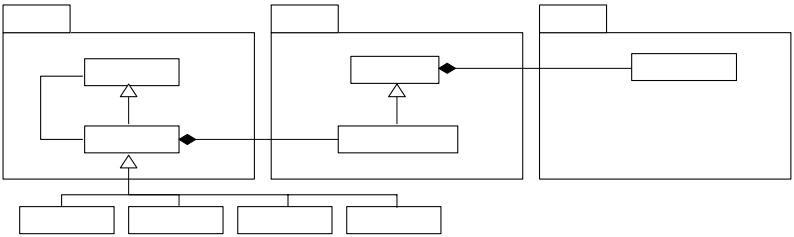


Рис. 20. Интеграция OCL-выражения
с компонентами «Ядро» и «Типы данных» UML

На рис. 20 изображена интеграция OCL-выражения с компонентами «Ядро» и «Типы данных» UML. Ограничения (Constraints) — это элементы модели (ModelElements), определяющие некоторые условия на других элементах модели. Булево выражение (BooleanExpression) формирует тело ограничения и является специальным типом выражения (Expression) с булевым результатом. Выражение в компоненте «Типы данных» UML имеет атрибуты «язык» (language) и «тело» (body), где атрибут «тело» содержит текстовое представление ограничения. В данной работе этот атрибут был заменен на необязательное отношение между выражением (Expression) и OCL-выражением (OclExpression). Таким образом, стало возможным использовать OCL для описания ограничений в UML. Более того, вместо OCL может быть использован другой язык. Абстрактный класс OclExpression определяет множество всех возможных OCL-выражений.

Так как ограничение может применяться в разных контекстах, например, для описания охранных, пред- и пост условий, а также инвариантов, были введены соответствующие подклассы.

Рис. 21 иллюстрирует компоненты метамодели OCL. Как можно заметить, метамодель языка разделена на три составляющих: выражения, типы и значения.

OCL

{ordered} 1..*

ModelElemen

constrainedElement

Constraint

1..*

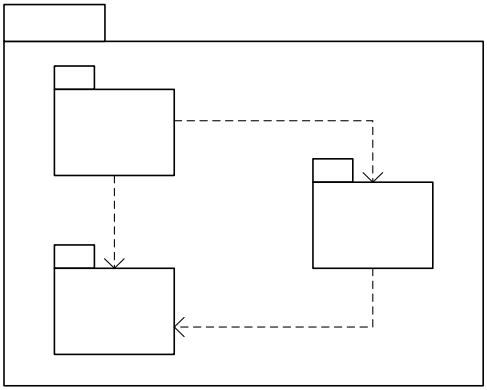


Рис. 21. Компоненты метамодели OCL

На рис. 22 изображена метамодель типов OCL

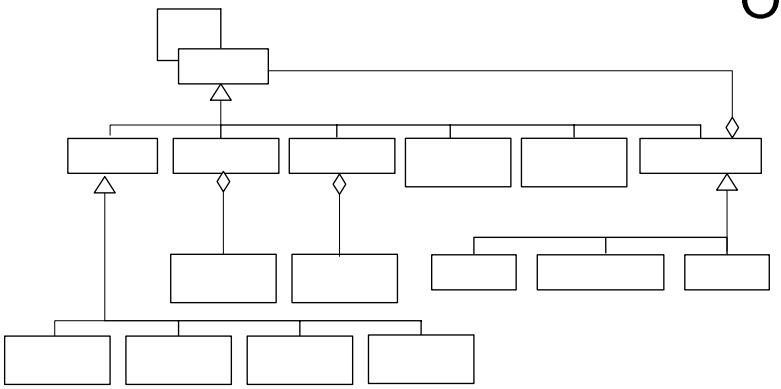


Рис. 22. Метамодель типов OCL

Все типы OCL являются специализаций абстрактного класса Тип (Type) и классифицируются следующим образом.

- Базовые типы: IntegerType, RealType, StringType, BooleanType.
- Тип экземпляра: InstanceType.
- Тип перечисления: EnumType.
- Специальные типы: OclAnyType, OclTypeType.
- Типы коллекций: SetType, SequenceType, BagType.

Классы, помеченные в модели стереотипом <<singleton>>, имеют только по одному экземпляру.

InstanceType используется для обращения к классификаторам (Classifier), определенным пользователем в модели классов UML, например, Person, Company.

Авторы работы не нашли в спецификации четкого определения выражения и составили свое определение, собранное из разных частей документации. OCL-выражение это представленные ниже конструкции:

1. Словосочетание self.
2. Словосочетание result в пост условии.
3. Переменная.
4. Применение операции:
   a. предопределенная операция: +, -, *, <, >, size, max, …,
   b. доступ к атрибуту: self.age,
   c. операция, определенная классификатором: p.income(),
   d. навигация по имени роли: self.employees,
   e. константа: 25.
5. Конструкция iterate и получаемые с ее помощью select, reject, collect, exists, forAll. Далее все эти функции будут называться *выражениями запросов* (query expression).

На рис. 23 изображена метамодель выражения OCL.

Все OCL-выражения (OclExpression) имеют определенный тип. Для вычисления OCL-выражения требуется некоторый контекст, задаваемый с помощью имени типа экземпляра, например, Person.

Результатом каждого OCL-выражения является значение (value). На рис. 24 изображена метамодель значения.

Структуры метамоделей значения и типа схожи (рис. 22). Каждое значение имеет тип, определяющий его свойства.

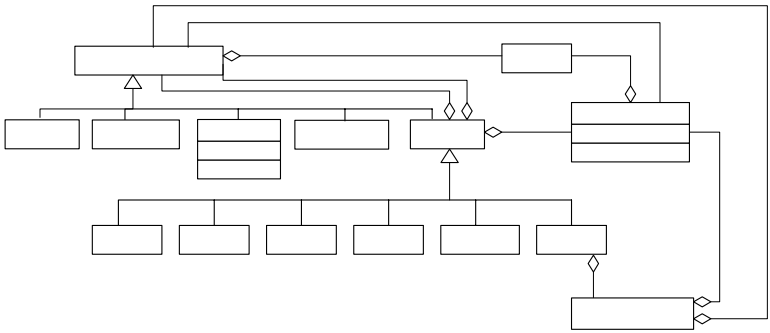


Рис. 23. Метамодель выражения OCL

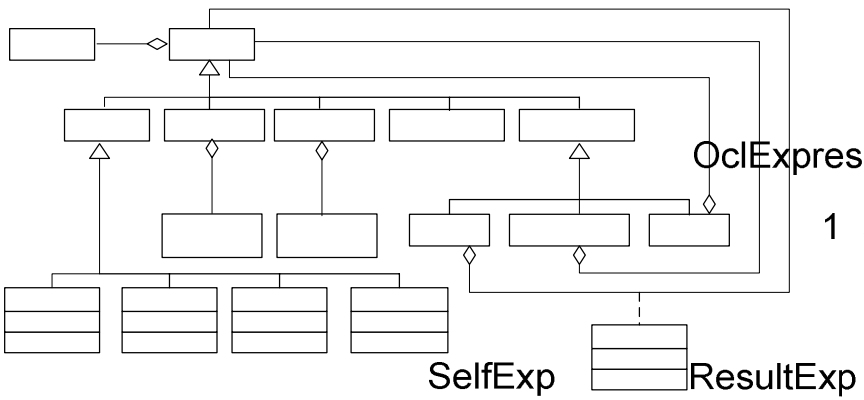


Рис. 24. Метамодель значения

Данная работа имеет большое значение, так как метамодель OCL, тесно интегрированная с метамоделью UML, является надежной базой для моделирования программных систем в рамках четырехуровневой архитектуры MOF.

A. Brucker и B. Wolff в работе [20] обсуждают вопросы построения формальной семантики языка OCL, встроенного в логику высокого порядка (higher-order logic или HOL). В качестве каркаса для HOL-OCL был выбран Isabelle — механизм общего доказательства теорем.

Логика высокого порядка — это классическая логика, расширенная параметрическим полиморфизмом функций высокого порядка. Она более выразительна, чем логика первого порядка, так как схема индукции может быть описана внутри логики. HOL может быть представлена типизированным функциональным языком программирования, таким как Standard ML или Haskell, расширенным логическими квантификаторами (logical quantifiers).

В таблице 1 представлены сравнительные характеристики OCL 1.4, OCL 2.0 [21] и HOL-OCL («+» — функциональность поддерживается, «-» — функциональность не поддерживается).

Таблица 1

Таблица свойств OCL 1.4, OCL 2.0 и HOL-OCL

	OCL 1.4	OCL 2.0 RfP	HOL-OCL
Extendible universes	-	-	+
General recursion	-	-	+
Smashing	?	-	+
Automated flattening	+	-	-
Tuple	-	+	+
Finite state	+	+	-
General Quantifiers	-	-	+
allInstances finite	+	+	-
Kleene logic	+	+	+
Strong and weak equality	-	+	+

В работе сделаны следующие выводы:

- логика Клини, используемая в OCL как базовая, в некоторых случаях не интуитивна;
- семантика описания метода нуждается в расширении;
- в OCL версии 2.0 можно поддерживать рекурсию, базирующуюся на семантике наименьшей неподвижности точки;
- семантику OCL можно упростить, введя бесконечные множества (infinite set) и общие квантификаторы (general quantifiers). Таким образом, OCL сможет быть более абстрактным и самодостаточным.

Данный подход делает OCL более понятным, прагматичным и ориентированным на инструментарий, кроме того, HOL-OCL разработан в духе спецификации OCL 2.0.

Использованию языка OCL в качестве языка запросов посвящена работа D. Akerhurst и B. Bordbar [22]. Для этого язык должен поддерживать множество из восьми операций реляционной алгебры: выборка (select), проекция (project), пересечение (intersect), разность (difference), соединение (join), деление (divide), объединение (union) и декартово произведение (product). Три операции — пересечение, соединение и деление — могут быть выражены через остальные пять операций, которые называются примитивными.

Версия OCL 1.3 не поддерживает в явном виде операции декартова произведения и проекции.

На рис. 25 изображена модель отношений между тремя классами: студент, курс, преподаватель.

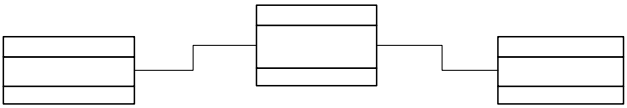


Рис. 25. Модель отношений между классами: студент, курс, преподаватель

На рис. 26 дано представление введенной модели классов в виде отношений реляционной модели данных.

Student

Stu_id	Name	Address

membership

Stu_id	Crs_id

Course

Crs_id	title	Code	Details

teaches

Crs_id	Tch_id

Teacher

Tch_id	Name	Salary

Рис 26. Модель данных

Следующий запрос на языке SQL позволяет выбрать имена всех преподавателей и студентов.

```
SELECT DISTINCT s.name, t.name
    FROM Stident s, Teacher t, membership m, teaches ts
    WHERE t.tch_id = ts.tch_id
    AND     ts.crs_id = m.crs_id
    AND     m.stu_id = s.stu_id;
```

Если нам дополнительно к именам потребуется получить еще какие-то поля, нам придется написать новый запрос. В идеальном случае хочется результатом выборки иметь множество объектов, удовлетворяющих заданным условиям.

В работе показано решение возникшей проблемы с помощью введения в модель классов дополнительного элемента — класса-ассоциации, служащего для моделирования кортежа. На рис. 27 представлена модель объектов и классов, описывающих кортеж с элементами A, B, C.

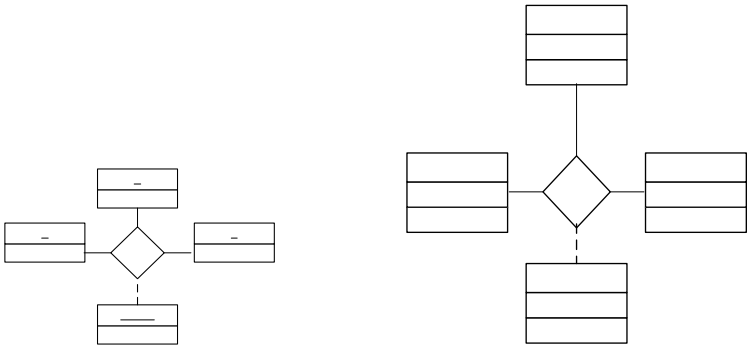


Рис. 27. Модель объектов и классов, описывающих кортеж

В таком случае модель отношений (рис. 25) может быть приведена к следующему виду:

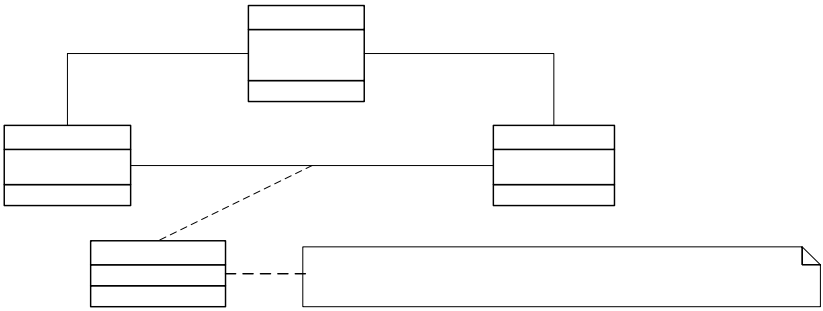


Рис. 28. Расширение модели отношений

Запрос, представленный выше, можно теперь заменить OCL-выражением.

```
Teacher.allInstances -> collect (t |
    t.cource.student -> collect (s |
        TeacherxStudent.allInstances -> select (ts |
            ts.teacher = t and
            ts.student = s ) ) ) -> asset
```

Однако данное решение не является оптимальным по причине загромождения модели классов дополнительными элементами, а также из-за низкой производительности реализации операции *allInstances.*

Авторы предлагают второй подход — реализацию расширения языка OCL путем введения в него операции декартова произведения и проекции. Введение данных операций в OCL возможно при определении в нем класса кортежей (Tuple) как параметризованного класса. Рис. 29 иллюстрирует данный подход.

В результате введения в OCL класса кортежей OCL-запрос будет, как и в первом случае, заменен на OCL-выражение.

```
Teacher.allInstances -> collect ( t |
    ( Set(t) ).product( t.course.student ) )
```

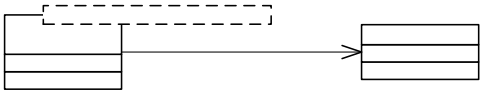


Рис. 29. Определение класса Tuple

Таким образом, решаются все поставленные в работе задачи.

3. ЗАКЛЮЧЕНИЕ

В данном обзоре рассмотрены формальные подходы к спецификации языков UML и OCL, которые можно резюмировать следующим образом.

В спецификации UML существует еще множество огрехов и противоречий, например:

- отношения включения и расширения (include and extend relationships) диаграммы вариантов использования описаны противоречиво;
- варианты использования, которые включаются в другие варианты использования или расширяют их, не являются вариантами использования как таковыми;
- в абстрактном синтаксисе стереотипа тегированное значение имеет неполное описание;
- семантика языка имеет «высокую техничность, немногословность и сложность для понимания новичком»;
- в языке отсутствует декларативная семантика, потому семантика языка противоречива в описании отношений между моделями, построенными с использованием языка и предметов моделирования, т.е. тем, что они моделируют;
- используемая метамодель языка недостаточно теоретически обоснована.

Многие научные исследования, посвященные формализации модели и метамодели языка UML, основываются не на самом UML, а на некотором его подмножестве — формальном и строго структурированном.

Результатом работы [10] является полная формальная спецификация метамодели объектно-ориентированного языка моделирования BON. Одна-

ко BON в сравнении с UML более формализован и «подогнан» под условия решаемой задачи.

Схожий подход использован в работе [13]. В ней продемонстрирована применимость этого подхода к формальной спецификации языка UML. При этом формализуется только часть спецификации. Стоит также отметить, что данное решение ограничено представлением всех метауровней моделируемой системы в одной модели, что при большом объеме моделируемой системы может стать проблемой.

Непосредственной близостью к языку UML отличается работа [16]. Формализация MML, являющегося подмножеством UML, с помощью MML-исчисления, а также использование повелительной семантики вместо декларативной является хорошим решением.

В работе [18] демонстрирует применимость алгебраического метода для формального описания ER-диаграмм, являющихся аналогом диаграмм классов UML.

Важно отметить, что этот подход формализует не только модель системы, но и ее метамодель. Данная работа может быть применена к диаграммам языка UML, описывающим статическую структуру моделируемой системы, как, например, диаграммы классов, объектов и т.д. Более того, представляется возможным применение предложенного подхода к описанию динамических аспектов моделируемой системы. Эти направления требуют дальнейшего научного исследования.

Теме формализации языка OCL в научной литературе уделено не меньшее внимание, чем формальным подходам к языку UML.

Важной особенностью метамодели языка OCL в работе [19] является ее интеграция с метамоделью UML. Уровень метамодели OCL совпадает с уровнем метамодели языка UML (так называемый уровень M2) и в качестве мета-метамодели используется модель MOF (уровень M3).

Использование HOL-OCL [20] делает OCL более понятным, прагматичным и ориентированным на инструментарий, кроме того, HOL-OCL разработан в духе спецификации OCL 2.0.

В результате применения подхода, описанного в работе [22], язык OCL можно использовать в качестве языка запросов.

СПИСОК ЛИТЕРАТУРЫ

1. **UML 1.4** Final Adopted Specification. — OMG, January 2002. — `http://www.omg.org/uml.`

2. **Буч Г., Рамбо Д., Джекобсон А.** Язык UML. Руководство пользователя. — М.: ДМК, 2000.
3. **MOF 1.4** Specification — OMG, April 2002. — `http://www.omg.org/mof`.
4. **Genova G., Llorens J., Quintana V.** Digging into Use Case Relationships // Lect. Notes Comput. Sci. — 2002. — Vol. 2460. — P. 115–127.
5. **Gogolla M., Henderson-Sellera B.** Analysis of UML Stereotypes within the UML Metamodel // Lect. Notes Comput. Sci. — 2002. — Vol. 2460. — P. 84–99.
6. **Naumenko A., Wegmann A.** A Metamodel for the Unified Modeling Language // Lect. Notes Comput. Sci. — 2002. — Vol. 2460. — P. 2–17.
7. **RM-ODP** Open Distributed Processing — Reference Model - ISO, ITU.: ISO/IEC 10746-1, 2, 3, 4 | ITU-T Recommendation X.901, X.902, X.903, X.904, 1995–1998;
8. **Russell, B.** Mathematical logic as based on the theory of types // Amer. Math. — 1908. — Vol. 30. — P. 222–262.
9. **Tarski A.** Logic, Semantics, Meta-mathematics. — Oxford University Press, 1956.
10. **Paige R., Ostroff J.** Metamodelling and Conformance Checking with PVS // Lect. Notes Comput. Sci. — 2001. — Vol. 2029. — P. 2–16.
11. **Walden K., Nerson J.-M.** Seamless Object-Oriented Software Development — Prentice-Hall, 1995.
12. **Owre S., Shankar N., Rushby J., Stringer-Calvert D.** The PVS Language Reverence Version 2.3 — September, 1999. — (Tech. Rep. / SRI International Technical Report).
13. **Overgaard G.** Formal Specification of OO Modeling // Lect. Notes Comput. Sci. — 2000. — Vol. 1783. — P. 193–207.
14. **Overgaard G.** A Formal Approach to Relationships in the Unified Modeling Language. — April, 1998 — (Tech Rep. / TUM-19803).
15. **Milner R., Parrow J., Walker D.** A Calculus of Mobile Processes // Inform. Comp. — 1992 — Vol. 100. — P. 1–40.
16. **Clark T., Evans A., Kent S.** The Metamodelling Language Calculus: Foundation Semantics for UML // Lect. Notes Comput. Sci. — 2001. — Vol. 2029. — P. 17–31.
17. **Cardeli L, Abadi M.** A theory of Objects — Springer-Verlag, 1996.
18. **Lellahi K.** Conceptual Data Modeling: An Algebraic Viewpoint // Lect. Notes Comput. Sci. — 2001. — Vol. 2244. — P. 336–348.
19. **Richters M., Gogolla M.** A Metamodel for OCL // Lect. Notes Comput. Sci. — 1999. — Vol. 1723. — P. 156–171.
20. **Brucker A., Wolff B.** HOL-OCL Experiences, Consequences and Design Choices // Lect. Notes Comput. Sci. — 2002. — Vol. 2460. — P. 196–211.
21. **Warmer J., Kleppe A., Clark T., et al.** Response to the UML 2.0 OCL RfP. — March, 2002 — (Tech. Rep.).

22. **Akerhurst D., Bordbar B.** On Querying UML data models with OCL // Lect. Notes Comput. Sci. — 2001. — Vol. 2185. — P. 91–103.

С. А. Бражник

ОБЗОР ФОРМАЛЬНЫХ ПОДХОДОВ К СПЕЦИФИКАЦИИ ЯЗЫКОВ UML И OCL

Препринт
121

Рукопись поступила в редакцию 19.11.04
Редактор З. В. Скок
Рецензент Н. В. Шилов

Подписано в печать 15.03.04
Формат бумаги 60 × 84 1/16 Объем 1.9 уч.-изд.л., 2.1 п.л.
Тираж 60 экз.

ЗАО РИЦ «Прайс-курьер»
630090, г. Новосибирск, пр. Акад. Лаврентьева, 6, тел. (383-2) 30-72-02