Инкрементальная верификация объектно-ориентированных данных на основе спецификации ограничений[1]

В.А. Семенов, С.В. Морозов, О.А. Тарлапан

Аннотация. Рассматриваются задачи полной и инкрементальной верификации объектно-ориентированных данных. На основе теории графов строится формальный аппарат, а также описываются разработанные методы инкрементальной верификации, использующие статический анализ спецификации ограничений и позволяющие локализовать область потенциальных нарушений при изменении данных. Методы предоставляют весомую альтернативу полной верификации прикладных данных при локальных изменениях, преобладающих в многопользовательских графических приложениях. Постановка задачи и методы описываются применительно к языку EXPRESS, предоставляющему универсальную нотацию для спецификации объектно-ориентированных моделей данных и ограничений на них.

1. Введение

Обеспечение целостности является одним из фундаментальных ACID (atomicity, consistency, isolation, durability) принципов мультидоступа к данным на основе транзакций [1]. Для реализации данного принципа современные информационные системы должны предусматривать механизмы, которые бы гарантировали удовлетворение ограничений целостности сразу после завершения транзакции (integrity constraint enforcement). Наиболее распространенными являются подходы, связанные с проверкой ограничений (integrity constraint checking) и поддержкой целостности (integrity constraint maintenance). Первый консервативный подход предполагает проверку целостности всякий раз перед завершением транзакции и – в случае выявленных нарушений – отмену всех изменений, принятых в ее ходе. В рамках второго альтернативного подхода при изменениях, влекущих нарушение условий целостности, предпринимается попытка дополнительной модификации данных с тем, чтобы их окончательное состояние полностью им соответствовало. Оба подхода имеют известные преимущества и недостатки. Этим вопросам, применительно, прежде всего, к дедуктивной и реляционной метамоделям, посвящены обширные исследования. Достаточно упомянуть исчерпывающие обзоры работ [2–4].

[1] Работа поддержана грантами РФФИ (04-01-00527) и Фонда содействия отечественной науке (http://www.science-support.ru).

В существенно меньшей мере внимание уделяется проблемам обеспечения целостности прикладных объектно-ориентированных данных, модели которых специфицированы, в частности, на популярных универсальных языках объектно-ориентированного моделирования EXPRESS [5] и UML/OCL [6]. Подобные модели предоставляют дополнительные возможности для статического и динамического анализа зависимостей между данными и учета локальности инкрементальных изменений, преобладающих в многопользовательских графических приложениях [7].

В разделе 2 вводится формальная объектно-ориентированная метамодель. Раздел 3 иллюстрирует ее на примере языка моделирования EXPRESS, предоставляющего нейтральную к реализации универсальную нотацию для спецификации объектно-ориентированных данных и ограничений на них. Особое внимание уделяется возможным видам ограничений, допускаемых конструкциями языка. В разделе 4 рассматривается общая постановка задачи верификации прикладных данных на основе спецификации ограничений. Вспомогательные понятия с привлечением аппарата теории графов вводятся в разделе 5. Здесь же приводятся конструктивные утверждения относительно зависимостей между данными и возможностей локализации нарушений целостности. Данные утверждения легли в основу разработанных методов инкрементальной верификации, которые детально описываются в разделе 6. В заключении даются общие рекомендации по их практическому применению и обозначаются направления для дальнейших исследований.

2. Объектно-ориентированная метамодель

Рассмотрим сначала наиболее общие понятия, представляющие информационную объектно-ориентированную метамодель, подходящую для специфицирования произвольной предметной области. Используемая здесь формализация следует методу, применяемому в семантическом анализе языков [8, 9].

Определим объектно-ориентированную метамодель как информационную схему $S = \langle T, \pi, Rule \rangle$ со следующим значением:

- $T = T_D \cup T_{\overline{D}} \cup C$ — множество типов данных информационной схемы, состоящее из основных типов T_D, отображаемых в основную область семантики D, сложных типов $T_{\overline{D}}$, отображаемых в многозначную область семантики $\overline{D}$, а также объектных типов (классов) C;
- π — частичный порядок на T, отражающий отношения обобщения/специализации между типами данных и используемый в качестве основы для встраиваемого механизма полиморфизма;
- $Rule = Rule_T \cup Rule_S$ — множество правил, налагающих ограничения целостности данных. Правила подразделяются на локальные $Rule_T$,

определяемые на отдельных типах данных, и глобальные $Rule_S$, определяемые на некоторой совокупности типов информационной схемы.

Важные свойства или характеристики объектов представляются их атрибутами. Имя атрибута определяет роль, играемую связанным с ним значением, в контексте того объекта, в котором оно появляется. Тип данных атрибута определяет область его возможных значений. Обозначим $Attr_C$ множество атрибутов информационной схемы или, точнее, множество сигнатур операций $a_c : C \to T$, $a_c : C \times T \to C$ для функций доступа к значениям атрибутов. Объявление атрибута $a_c \in Attr_C$ в объекте типа C устанавливает отношение между типом данных объекта и типом данных атрибута T: $C.a \xrightarrow{\{m:n\} \quad \{p:q\}} T$, где $0 \le m \le n$ ограничивают количество элементов в прямом отношении от C к T, а $0 \le p \le q$ — в обратном отношении от T к C. Данную запись следует интерпретировать так: для каждого объекта типа C роль a играется не менее чем m и не более чем n экземплярами типа T, для каждого экземпляра типа T существует от p до q объектов типа C, в которых T играет роль a. Если в качестве типа атрибута выступает другой объектный тип, то данный атрибут представляет связь (ассоциацию) между двумя объектными типами.

Состояние объектов определяется значениями их атрибутов. Поведение отдельных объектов характеризуется множеством локальных правил $Rule_C$, которые могут определять количество, тип и организацию атрибутов, а также налагать ограничения на их значения. Формально, данные правила являются множеством сигнатур операций $rule_c : C \to logical$. Ограничения на множество объектов (возможно, разных типов $C_1, \ldots, C_n$) налагают глобальные правила $Rule_S$, представляющие собой сигнатуры операций $rule_S : C_1, \ldots C_n \to logical$.

Способом, аналогичным алгебраической спецификации, предоставим сигнатуру $\Sigma_S = (T_D \cup T_{\overline{D}} \cup C, \pi, \Omega_D \cup \Omega_{\overline{D}} \cup Attr_C \cup Rule_T \cup Rule_S)$, где Ω_D, $\Omega_{\overline{D}}$ являются множествами операций, определенных над основными и сложными типами. Сигнатура, сформированная подобным способом, описывает типы и символы операций, принадлежащие к объектно-ориентированной метамодели S, а также содержит исходное множество синтаксических элементов, над которыми могут быть определены выражения $Expr(\{var_T\}_{T \in T_D \cup T_{\overline{D}} \cup C})$ с переменными, индексируемыми данными типами.

3. Язык моделирования EXPRESS

EXPRESS — это формальный язык информационного моделирования [5]. Он основан на определении объектов, представляющих формализованные

описания моделируемых объектов реального мира. При создании языка EXPRESS ставилась задача избежать, насколько это возможно, влияния особенностей реализации. Таким образом, EXPRESS является декларативным объектно-ориентированным языком. В настоящее время он широко используется в качестве стандартного средства спецификации моделей данных для различных отраслей науки и промышленности.

Рассмотрим вкратце семантику языка EXPRESS, используя описанную в предыдущем разделе метамодель.

3.1. Типы данных и операции над ними

Язык EXPRESS определяет основные типы *{Real, Integer, Number, Boolean, Logical, String, Binary}* $\subseteq T_D$, соответствующие простым типам данных с общепринятой семантикой. Сложные типы *{Aggregate, Bag, Set, List, Array, Enumeration, Select, Defined}* $\subseteq T_{\overline{D}}$ соответствуют различным типам множеств, перечисляемым типам, выборкам и переопределенным типам, которые допускают определение производных и вложенных структур данных. Типы данных объектов C в языке EXPRESS определяются декларацией *Entity*.

На Рис. 1 представлена общая классификация имеющихся в языке EXPRESS типов. Типы, отмеченные жирным курсивом, соответствуют его исходным типам. Стрелки показывают отношения наследования между ними. Виртуальные типы, отмеченные обычным шрифтом, используются только для того, чтобы единообразно классифицировать конструкции языка и представить типы, определяемые пользователем.

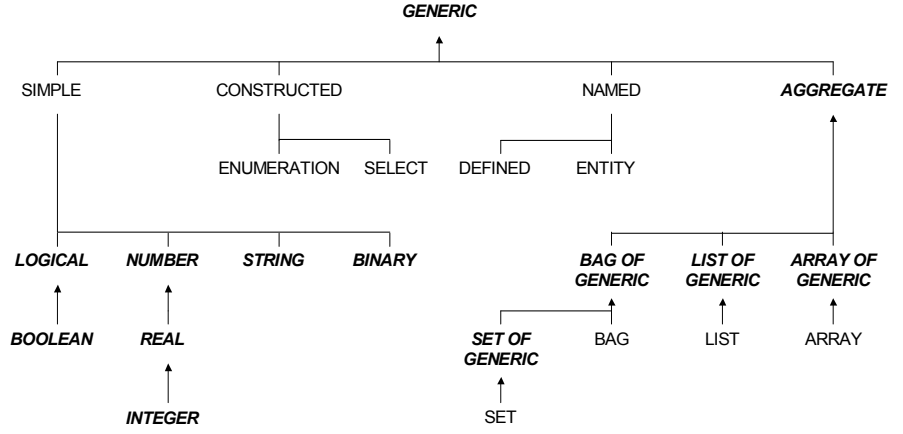

Рис. 1. Классификация исходных типов языка EXPRESS.

Простыми типами данных являются *Boolean*, *Logical*, *Number* (включая конкретные подтипы *Real* и *Integer*), *String* и *Binary*. Интерпретация простых типов является обычной, но каждое множество расширяется специальной вели-

чиной *?*, обозначающей неопределенное значение. Тип $T \in T_D$ отображается в область семантики D функцией $I : T \alpha D$ следующим образом: I (*Boolean*) = {*true, false*} Y{*?*} , I (*Logical*) = {*true, false, unknown*} Y{*?*} , I (*Real*) = R Y{*?*} , I (*Integer*) = Z Y{*?*} , I (*String*) = A^* Y {*?*} , где A — конечный алфавит, A^* — множество всех последовательностей над алфавитом A , и I (*Binary*) = $\{0,1\}^*$ Y {*?*} . Длины строк и двоичных последовательностей могут быть неограниченными, ограниченными сверху или фиксированными.

Над простыми типами определено множество операций. Существует несколько групп операторов:

{*+, -, *, /, **: Number* × *Number* → *Number; +, -, abs: Number* → *Number*} Y

{*div, mod: Integer* → *Integer; odd: Integer* → *Logical*} Y

{*sin, cos, tan, asin, acos, atan, exp, log, log2, log10, sqrt: Number* → *Real*} Y

{*format: Number* × *String* → *String; value: String* → *Number*} Y

{*and, or, xor: Logical* × *Logical* → *Logical; not: Logical* → *Logical*} Y

{*+: String* × *String* → *String, like: String* × *String* → *Logical; length: String* → *Integer; []: String* × *Integer* → *A, [:]: String* × *Integer* × *Integer* → *String*} Y

{*+: Binary* × *Binary* → *Binary; blength: Binary* → *Integer; []: Binary* × *Integer* → *0|1, [:]: Binary* × *Integer* × *Integer* → *Binary*} Y

{*=, <>, <=, >=, <, >: Number* × *Number* → *Logical, Logical* × *Logical* → *Logical, String* × *String* → *Logical, Binary* × *Binary* → *Logical*}.

Семантику данных операций легко истолковать. В частности, это арифметические операции над числовыми операндами типов *Number*, *Real*, *Integer* и стандартные математические функции; преобразование чисел в строковое представление и обратно; логические операции над логическими операндами типов *Logical* или *Boolean*; конкатенация, сопоставление с образцом, индексация, выделение подмножества над операндами типов *String* и *Binary*, а также операции сравнения, определенные для всех простых типов. Некоторые операции имеют тот же самый перегруженный символ в качестве имени и могут различаться только по типам своих аргументов.

Созданные типы, которыми могут являться перечисление либо выборка, расширяют множество основных типов. Область определения перечисляемого типа *Enumeration* задается упорядоченным множеством значений, представленных уникальными именами $I(Enumeration(a_i, i = 1,\text{K}\ ,n)) = \{a_i \in A^*, i = 1,\text{K}\ ,n\}$ Y {*?*} . На литералы перечисляемого типа ссылаются как на элементы перечисления. Порядок элементов перечисления предопределяется их относительными позициями. Данное упорядочение дает в результате операции сравнения, определенные между значениями, относящимися к одному и тому же перечисляемому типу:

{*=, <>, <=, >=, <, >: Enumeration* × *Enumeration* → *Logical*}.

Выбираемый тип данных *Select* определяет производный тип, представляемый списком других исходных типов. Экземпляр выбираемого типа данных является экземпляром одного из типов, заданных в списке выбора. Допускается, что экземпляр данных принадлежит одному из нескольких возможных типов. Другими словами, область определения значений для подобного типа является объединением областей определения типов в его списке $I(Select(T_i, i = 1,\text{K}\ ,n)) = I(T_1)$ Y K Y $I(T_n)$ Y {*?*} . Выбираемый тип является обобщением каждого из типов данных, перечисленных в его списке.

К поименованным относятся типы данных, определяемые пользователем в конкретных информационных схемах. Так, определенный (*Defined*) тип данных является пользовательским расширением стандартных типов (простых, созданных или агрегатных), при этом он дает возможность определить дополнительные семантические ограничения, налагаемые на специфицируемые данные $I(Defined(T)) = I(T)$ Y {*?*} . Определенный тип является конкретизацией исходного типа данных, который используется при его объявлении.

Объектные типы определяются пользователем с помощью конструкции *ENTITY*. При специфицировании объектного типа указываются его явные (*EXPLICIT*) атрибуты (в том числе ассоциативные связи с другими объектными типами), обратные ассоциации (*INVERSE*), а также производные вычисляемые свойства (*DERIVED*), определяемые типами и выражениями, которые могут включать в себя значения явных атрибутов, константы, исполняемые операторы, включая вызов функций и процедур, как стандартных, так и пользовательских. Язык EXPRESS предусматривает разнообразные модели простого и множественного наследования объектных типов. Объект производного типа наследует все атрибуты, определяемые в его родительских типах. При этом допускается переопределение отдельных атрибутов с целью специализации их типов, постановки более жестких ограничений на них или задания другого выражения для вычисляемых свойств. Кроме того, явные атрибуты базовых типов могут быть переопределены как вычисляемые в типах-наследниках.

Над объектными типами определены следующие операции:

{*=, <>, :=:, :<>: : C* × *C* → *Logical*} Y

{*rolesof: C* → *Set*(*String*), *usedin: C* × *String* → *Bag*(*C*)}.

Операторы сравнения = и <> реализуют глубокий алгоритм сравнения объектов. В отличие от них, операторы *:=:* и *:<>:* проверяют лишь эквивалентность экземпляров (или совпадение идентификаторов объектов). Функция *rolesof* возвращает набор строк, содержащих имена ролей (ассоциативных атрибутов), играемых заданным объектом. Функция *usedin* возвращает множество объектов, которые используют указанный объект в заданной роли (под *C* здесь понимается некий обобщенный объектный тип). Вследствие наличия данной функции все связи между объектами в языке EXPRESS являются двунаправленными. Инверсные атрибуты используются лишь для именования ряда обратных связей, а также постановки ограничений

использования объектов в связях с другими, о чем будет упомянуто в следующем разделе.

Многозначные выражения в языке EXPRESS описываются агрегатными типами. Агрегации используются для представления упорядоченных и неупорядоченных коллекций данных каких-либо основных, предопределенных сложных и объектных типов. Они конструируются путем указания типа для их элементов. Эти коллекции могут иметь фиксированные или переменные размеры, хранить уникальные или дублированные представления элементов, а также могут допускать неопределенные элементы, что дает в результате разреженные структуры данных. Как и любой другой тип данных языка EXPRESS, сами агрегации могут иметь неопределенное значение (это состояние следует отличать от пустой коллекции, не содержащей ни одного элемента).

Так, тип данных *Array* (массив) представляет собой структуру с фиксированным размером, где существенной является индексация элементов. Опционально массивы могут допускать, что не все их элементы имеют значение. Свойство уникальности массивов означает, что каждый элемент отличается от любого другого в одном и том же экземпляре массива. Тип данных *List* (список) представляет упорядоченную коллекцию идентичных элементов. Список может хранить любое количество элементов, допуская или, опционально, не допуская их дублирование. Тип данных *Bag* (мультимножество) является коллекцией элементов, в которой порядок не является важным и допускается дублирование. И, наконец, *Set* (набор) является коллекцией элементов, в которой порядок не является важным и дублирование элементов не допускается. Количество элементов в списках, мультимножествах, наборах может изменяться в зависимости от определения их границ. Важно, что агрегатные типы могут быть вложенными, таким образом давая в результате многомерные структуры данных.

Над агрегатными типами в языке EXPRESS определены следующие группы операций:

{+, -, *: *Aggregate*(T_1) × *Aggregate*(T_2) → *Aggregate*(*T*)} Y

{=, <>, <=, >=, :=:, :<>: : *Aggregate*(T_1) × *Aggregate*(T_2) → *Logical*} Y

{*in, value_in: T* × *Aggregate*(*T*) → *Logical*, *value_unique: Aggregate*(*T*) → *Logical*, *query: Aggregate*(*T*) × *Logical Expression* → *Aggregate*(*T*)} Y

{*[]: Aggregate*(*T*) × *Integer* → *T*} Y

{*sizeof, hiindex, loindex, hibound, lobound: Aggregate*(*T*) → *Integer*} Y

{*insert: List*(*T*) × *T* × *Integer* → *List*(*T*), *remove: List*(*T*) × *Integer* → *List*(*T*)}.

Операторы над множествами +, -, * определяются для объединения, вычитания и пересечения коллекций совместимых типов T_1, T_2 (т.е. одинаковых типов или типов, один из которых является подтипом или конкретизацией другого). Операторы сравнения, включая операторы подмножества и надмножества <=, >=, предназначены для установления отношений равенства и общности между

совместимыми по типу коллекциями. Операторы сравнения экземпляров *:=:*, *:<>:* отличаются от операторов сравнения значений =, <> лишь для коллекций объектов, используя соответствующие вышеупомянутые операции, определенные над объектными типами. Для коллекций остальных типов данных операторы сравнения экземпляров эквивалентны соответствующим операторам сравнения значений. Операторы принадлежности *in* и *value_in* проверяют наличие в коллекции некоторого элемента, используя, соответственно, операции эквивалентности экземпляров и равенства значений. Функция *value_unique* проверяет уникальность по значению всех элементов коллекции. Выражение запроса *query* вычисляет логическое условие отдельно для каждого элемента коллекции и возвращает коллекцию, содержащую те элементы, для которых логическое выражение истинно. Оператор индексирования *[]* извлекает из коллекции отдельный элемент. Операции *sizeof*, *hiindex*, *loindex, hibound, lobound* возвращают, соответственно, количество элементов в агрегации, верхний и нижний индексы элементов массива, верхнюю и нижнюю границы списков, мультимножеств и наборов. Операции *insert* и *remove* определяются дополнительно, чтобы упростить манипулирование специфическими коллекциями — списками.

Агрегации, выборки и определенные типы могут быть вложенными, поэтому в языке EXPRESS допустимо составление более сложных типов. В нетривиальных прикладных моделях данных невозможно избежать подобных сложных структур.

Возможная дополнительная классификация типов не противоречит описанной в предыдущем разделе объектно-ориентированной метамодели. В частности, в языке EXPRESS в отдельную группу выделяются обобщенные типы данных *Generic, Aggregate, Bag Of Generic, Set Of Generic, List Of Generic, Array Of Generic*, которые используются для установления обобщения других типов данных и могут применяться в очень специфических контекстах (для представления формальных параметров, локальных переменных и результатов функций). Следующие операции, определенные над обобщенными типами, применимы к любому типу данных языка EXPRESS:

{*exists: Generic* → *Boolean; nvl: T* × *T* → *T; typeof: Generic* → *Set*(*String*)}.

Это функция проверки существования значения (возвращает *true* в случае фактического значения ее параметра или *false* в случае неопределенного значения), функция возврата альтернативного значения, если исходный параметр находится в неопределенном состоянии, а также функция, определяющая тип данных входного параметра (возвращает имена всех типов, членом которых является ее входной параметр).

3.2. Ограничения в языке EXPRESS

Помимо декларативной части, содержащей описания типов данных, язык EXPRESS имеет и исполняемые операторы, связанные с проверкой правил и вычислением значений производных атрибутов. Правила налагают

ограничения на область определения значений специфицируемых типов данных. Помимо правил в языке EXPRESS существуют и другие виды ограничений, связанные со способом организации и типом атрибутов отдельных объектов. Ограничения являются критериями целостности данных, специфицируемых на языке EXPRESS.

Можно выделить три группы ограничений в зависимости от места их определения (утверждения). К первой группе относятся ограничения, налагаемые на отдельные типы данных (и атрибуты соответствующих типов), а именно:

- ограничение длины в строковом типе данных: $length(String) \le n, n \in Z, n > 0$ для строк переменной длины или $length(String) = n, n \in Z, n > 0$ для строк фиксированной длины (если n не указано в спецификации типа, ограничение отсутствует);
- ограничение длины в двоичном типе данных: $blength(Binary) \le n, n \in Z, n > 0$ для двоичных последовательностей переменной длины или $blength(Binary) = n, n \in Z, n > 0$ для двоичных последовательностей фиксированной длины (если n не указано в спецификации типа, ограничение отсутствует);
- ограничение количества элементов в агрегатном типе данных: $m \le sizeof(Aggregate(T)) \le n, m \in Z, n \in Z, 0 \le m \le n$ для списков, мультимножеств и наборов (для данных разновидностей агрегатов ограничение отсутствует, если *m=0 and n=?*), $sizeof(Array(T)) = n - m + 1, m \in Z, n \in Z, m \le n$ для массивов (в данном случае *m, n* указывают диапазон изменения индекса элементов);
- запрет на наличие повторяющихся элементов в агрегатном типе данных: $elem_i :\neq: elem_j, i = 1, \text{K}, sizeof(Aggregate(T)) - 1, j = i + 1, \text{K}$, $sizeof(Aggregate(T))$ для списков и наборов (для наборов ограничение устанавливается по умолчанию, для списков — если в спецификации типа указано ключевое слово *UNIQUE*), $elem_i :\neq: elem_j, i = m, \text{K}, n - 1, j = i + 1, \text{K}, n$ для массивов (в данном случае ограничение установлено, если в спецификации типа указано ключевое слово *UNIQUE*); заметим, что здесь используется операция эквивалентности экземпляров, а не равенства значений;
- запрет на наличие необязательных элементов в массиве: $exists(elem_i) = True, i = m, \text{K}, n$ (ограничение может быть снято указанием ключевого слова *OPTIONAL* в спецификации типа массива);
- ограничения области определения значений в определенном типе данных: $f_i(self) : Defined(T) \alpha Logical$ ($f_i(self)$ — опционально указываемые в спецификации определенного типа логические выражения, в которых переменной *self* присваивается текущее значение атрибута определенного типа при проверке соответствующего правила; область значений определенного типа данных *Defined*(*T*) состоит из всех значений области определения значений исходного типа *T*, не нарушающих ни одно из данных правил);
- требование совместимости типов атрибута и присваиваемого ему значения: $T_{value} \pi T_{attr}$ — типы должны совпадать или тип присваиваемого значения должен быть специализацией типа, приписанного атрибуту (наиболее часто проверяют совместимость типа ассоциированного объекта с типом ассоциации).

Во вторую группу входят локальные ограничения, которые устанавливаются в объектных типах данных и действуют для каждого объекта соответствующего типа. Локальные ограничения, определенные в базовом типе объекта, наследуются всеми его производными типами, как и атрибуты. К данной группе относятся:

- запрет на неопределенные значения для явных атрибутов объекта: $exists(attr_i) = True, i = 1, \text{K}, k$, где k — количество явных обязательных атрибутов (ограничение может быть снято указанием ключевого слова *OPTIONAL* в спецификации атрибута);
- ограничения использования объекта в ассоциативных связях с другими объектами, устанавливаемые через его инверсные атрибуты: $C_1.a \xrightarrow{\{m:n\} \quad \{p:q\}} C_2.inv\_a, p \le sizeof(inv\_a) \le q, p \in Z, q \in Z, 0 \le p \le q$, если инверсный атрибут имеет тип мультимножества или набора объектов C_1, причем в последнем случае дополнительно устанавливается ограничение уникальности инверсного отношения: $inv\_a_i :\neq: inv\_a_j$, $i = 1, \text{K}, sizeof(inv\_a) - 1, j = i + 1, \text{K}, sizeof(inv\_a)$ (т.е. объект типа C_2 не может играть роль *a* более одного раза в одном и том же объекте типа C_1), $sizeof(inv\_a) = 1$, если инверсный атрибут имеет тип объекта C_1;
- ограничения уникальности экземпляров отдельного атрибута: $o_i.a :\neq: o_j.a, T(o_i) \pi C, T(o_j) \pi C$ — никакие два объекта совместимых типов не должны использовать тот же самый экземпляр для атрибута, имя которого указано в простом правиле уникальности;
- ограничения уникальности комбинации экземпляров атрибутов: $o_i.a_1 :\neq: o_j.a_1 \; and \; o_i.a_2 :\neq: o_j.a_2 \; \text{K} \; and \; o_i.a_k :\neq: o_j.a_k, T(o_i) \pi C, T(o_j) \pi C$ — никакие два объекта совместимых типов не должны использовать ту же самую комбинацию экземпляров атрибутов, список имен которых указан в совместном правиле уникальности;
- ограничения области значений в типе данных объекта: $f_i(Attr_C) : C \alpha Logical$ ($f_i(Attr_C)$ — опционально указываемые в спецификации объектного типа логические выражения над атрибутами самого объекта (собственными и унаследованными от его супертипов) и,

возможно, объектов, ассоциированных с ним; объект попадает в область значений, если он не нарушает ни одно из данных правил).

Наконец, к третьей группе относятся глобальные ограничения, налагаемые на все объекты одного или нескольких типов:

$$f(Set(C_1),\text{K},Set(C_k)) : C_1,\text{K},C_n \ \alpha \ \ Logical\ .$$

Другими словами, глобальное ограничение является логической функцией над наборами объектов тех типов, на которых оно установлено. Наборы объектов, не нарушающие глобальные ограничения, считаются верными. Заметим, что любое ограничение является логической функцией. Ограничение считается нарушенным, если только соответствующая функция возвращает *false*. Ограничение считается выполненным, если функция возвращает *true*. Если функция возвращает *unknown*, ограничение ни выполнено, ни нарушено (такое возможно, если в выражениях участвуют необязательные явные атрибуты объектов с вероятными неопределенными значениями).

Рассмотрим пример использования ограничений в языке EXPRESS на фрагменте спецификации геометрической модели (см. рис. 2). Во-первых, ограниченной является область определения значений ряда атрибутов в специфицируемых типах объектов. Так, в качестве значений атрибута *Identifier* в объектном типе *GeometryObject* должны использоваться строки фиксированной длины в 16 символов, диапазон значений размерности (*Dim*) точек (*Point*), кривых (*Curve*) и их сегментов (*CurveSegment*) ограничен целыми числами в интервале от 1 до 3, что определяется правилом *WR1* в типе *Dimension*, а также недопустимыми являются неопределенные значения для явных атрибутов всех типов объектов, за исключением атрибута *TextCommentary* в типе *GeometryObject*. Используемые агрегатные типы данных налагают ограничения на количество элементов в коллекциях *Coordinates*, определяемых в типе *Point*, и *Segments*, определяемых в типе *CompositeCurve*, причем в последней ограничено только минимальное количество элементов.

Дополнительные ограничения на отдельные объекты налагают правила и инверсные атрибуты, определенные в соответствующих объектных типах. Так, правило *UR1* устанавливает уникальность значений атрибута *Identifier* в объектах типа *GeometryObject*. Правило *DimCoincide* в объектном типе *CompositeCurve* определяет требование совпадения размерностей ассоциированных объектов типа *CurveSegment*. Правило *CheckTransition* в объектном типе *CompositeCurve* устанавливает, что сопряжение сегментов составной кривой должно быть непрерывным, за исключением последнего сегмента незамкнутой кривой. Заметим, что в данных правилах используются не только локальные атрибуты, но и атрибуты ассоциированного объектного типа *CurveSegment*. Также, на объекты типа *CurveSegment* посредством инверсного атрибута *UsingCurves* налагается следующее ограничение: сегмент должен быть ассоциирован по крайней мере с одной составной кривой, причем с одной и той же кривой он не может быть ассоциирован многократно.

Наконец, глобальное правило *CurveDimensionsCoincide* устанавливает следующее ограничение: в правильном наборе данных должны совпадать размерности всех кривых.

```
SCHEMA Geometry;
TYPE Dimension = INTEGER;
WHERE
  WR1: { 0 < SELF <= 3 };
END_TYPE;
ENTITY GeometryObject ABSTRACT SUPERTYPE OF (ONEOF(Point, Curve, Surface));
  Identifier: STRING(16) FIXED;
  TextCommentary: OPTIONAL STRING;
UNIQUE
  UR1: Identifier;
END_ENTITY;
ENTITY Point SUBTYPE OF(GeometryObject);
  Coordinates: LIST [2:3] OF REAL;
DERIVE
  Dim: Dimension := HIINDEX(Coordinates);
END_ENTITY;
ENTITY Curve
  ABSTRACT SUPERTYPE OF (ONEOF(CompositeCurve, Conic, Line, OffsetCurve,
Polyline, TrimmedCurve))
  SUBTYPE OF(GeometryObject);
DERIVE
  Dim: Dimension := CalculateCurveDimension(SELF);
END_ENTITY;
ENTITY CurveSegment;
  ContinuousTransition: LOGICAL;
  ParentCurve: Curve;
DERIVE
  Dim: Dimension := ParentCurve.Dim;
INVERSE
  UsingCurves: SET [1:?] OF CompositeCurve FOR Segments;
END_ENTITY;
ENTITY CompositeCurve SUBTYPE OF(Curve);
  Segments: LIST [1:?] OF CurveSegment;
  SelfIntersect: LOGICAL;
DERIVE
  NSegments: INTEGER := SIZEOF(Segments);
  ClosedCurve: LOGICAL := Segments[NSegments].ContinuousTransition;
  SELF\Curve.Dim: Dimension := Segments[1].Dim;
WHERE
  CheckTransition: ((NOT ClosedCurve) AND (SIZEOF(QUERY(Temp <* Segments |
     NOT Temp.ContinuousTransition)) = 1)) OR ((ClosedCurve) AND
     (SIZEOF(QUERY(Temp <* Segments | NOT Temp.ContinuousTransition)) = 0));
  DimCoincide: SIZEOF(QUERY( Temp <* Segments | Temp.Dim <>
Segments[1].Dim)) = 0;
```

```
END_ENTITY;
RULE CurveDimensionsCoincide FOR (Curve);
  WHERE
     WR1: SIZEOF( QUERY( Temp <* Curve | Temp.Dim <> Curve[1].Dim)) = 0;
END_RULE;
FUNCTION CalculateCurveDimension (c : Curve) : Dimension;
LOCAL
  result : Dimension := 3;
END_LOCAL;
  IF ('GEOMETRY.LINE' IN TYPEOF(c))
     THEN result := …;
  ELSE
     IF ('GEOMETRY.CONIC' IN TYPEOF(c))
        THEN result := …;
     ELSE
        IF ('GEOMETRY.POLYLINE' IN TYPEOF(c))
           THEN result := …;
        ELSE
           IF ('GEOMETRY.COMPOSITECURVE' IN TYPEOF(c))
              THEN result := c.Dim;
           ELSE
              …
           END_IF;
        END_IF;
     END_IF;
  END_IF;
  RETURN (result);
END_FUNCTION;
…
END_SCHEMA;
```

Рис. 2. Фрагмент спецификации геометрической модели на языке EXPRESS.

4. Задача верификации модели

Определение 1 (Прикладная модель). Прикладной моделью M_S назовем множество ассоциативно связанных между собой объектов, типы которых определяются некоторой информационной схемой *S*.

По существу, прикладная модель задает множество объектов, семантически объединенных единой информационной схемой и связанных единым контекстом использования. Предполагается, что все ассоциативные связи модели являются внутренними, поскольку в противном случае возможная навигация по ним нарушала бы целостность контекста использования.

Определение 2 (Объектная популяция). Поименованное подмножество объектов *p* одного типа назовем объектной популяцией прикладной модели

M_S. Пусть $O(p)$ — множество объектов популяции, а $T(p)$ — ее тип, тогда верно следующее: $O(p) \subset M_S$, $T(p) \in S$.

Определением допускается, что разные популяции p_1 и p_2 одной прикладной модели могут иметь один и тот же тип $T(p_1) = T(p_2)$, $O(p_1) \subset M_S$, $O(p_2) \subset M_S$, $p_1 \neq p_2$, а один и тот же объект o может одновременно принадлежать нескольким популяциям модели $o \in O(p_1) \subset M_S$, $o \in O(p_2) \subset M_S$.

Определение 3 (Экстент типа). Подмножество всех объектов прикладной модели M_S, принадлежащих типу $T \in S$, назовем экстентом типа модели $Ext(M_S, T)$.

Очевидно, что если объектный тип $T_1 \in S$ является подтипом $T_2 \in S$, $T_1 \pi T_2$, то порождаемый им экстент является подмножеством экстента супертипа $Ext(M_S, T_1) \subset Ext(M_S, T_2)$. Кроме того, любая популяция модели *p* подтипа *T* принадлежит соответствующему экстенту модели, порождаемому этим типом, т.е. если $T(p) \pi T$, то $O(p) \subset Ext(M_S, T)$.

Приведем постановку задачи верификации прикладной модели. Пусть M_S — некоторая прикладная модель информационной схемы *S*, и схема *S* определяет множества типов данных *T* (в т.ч. объектных), а также правил *Rule*, ставящих логические условия целостности данных. Будем считать, что во множество *Rule* входят как локальные ограничения *local_rule*, определяемые на отдельных типах данных и действующие для каждого экземпляра соответствующего типа, так и глобальные правила *global_rule*, определяемые на некоторой совокупности типов и действующие для экстентов данных типов. Требуется найти подмножество объектов модели $O \subset M_S$, таких что:

$$\exists local\_rule \in Rule : local\_rule(o) = false, o \in O,$$

$$\exists global\_rule \in Rule : global\_rule(Ext(M_S, T_1, \text{K}, T_n)) = false, Ext(M_S, T_1, \text{K}, T_n) \subset O$$

Другими словами, целостность данных прикладной модели нарушена, если нарушено хотя бы одно локальное ограничение для любого из ее объектов или глобальное ограничение для соответствующих экстентов.

Алгоритм верификации модели приведен ниже. Верификатор записывает в журнал (log), какие именно ограничения и для каких объектов в ней были нарушены. В отдельных циклах проверяются локальные ограничения для каждого из объектов модели и глобальные правила, определяемые в информационной схеме. Отметим, что для сокращения записи алгоритма не приводятся варианты выбора ограничений на отдельном атрибуте в зависимости от его типа и контекста использования в классе объектов (явный, инверсный). Подразумевается, что выбор конкретных ограничений и их

последующая проверка осуществляется здесь: `Check AttributeConstraints(Object, Attribute)`.

```
for each Object in Model
  for each Attribute in Object
     check AttributeConstraints(Object, Attribute), log if violated
  for each DomainRule in T(Object)
     check DomainRule(Object), log if violated
  for each UniquenessRule in T(Object)
     for each Object2 in Model after Object
        if T(Object2) is subtype of T(Object)
           for each Attribute in UniquenessRule
              compare Attribute in Object2 with Attribute in Object
           log if the comparison is true for each Attribute
for each GlobalRule in Schema
  check GlobalRule, log if violated
```

5. Применение графов для анализа объектно-ориентированных моделей

Определение 4 (Ассоциативный граф — RG). Пусть M_S — прикладная модель, определяемая информационной схемой *S*, *P* — некоторое множество популяций модели, $R = R_S \cup R_a$ — множество переходов между ними, каждый из которых представляется либо отношением принадлежности одной популяции другой, либо ассоциативным отношением, определяемым схемой *S* для соответствующих объектных типов популяций. Тогда ассоциативным графом $RG\langle P, R\rangle$ для модели M_S назовем ориентированный граф, вершины которого *P* соответствуют всем популяциям модели, а направленные ребра *R*, инцидентные парам вершин, — простым и ассоциативным переходам между популяциями (R_S и R_a соответственно).

Сделаем важные замечания относительно используемого понятия перехода. Под простым переходом $r \in R_S, r: p_i \to p_j$ $(i \neq j)$ понимается отношение принадлежности популяции p_i родственной популяции p_j, т.е. $T(p_i) \pi T(p_j)$ и $O(p_i) \subset O(p_j)$.

Под ассоциативным переходом $r \in R_a, r: p_i \to p_j$ (возможно, $i = j$) понимается соответствующее ассоциативное отношение *role*, определенное информационной схемой для соответствующих объектных типов $T(p_i)$ и $T(p_j)$ явным или неявным образом. В частности, если схема определяет ассоциацию *role* между объектными типами T_1 и T_2, то на основе этой ассоциация могут строиться переходы между популяциями модели p_i и p_j соответствующих подтипов $T(p_i) \pi T_1$ и $T(p_j) \pi T_2$.

Способы задания ассоциативных переходов, как и способы определения самих ассоциативных отношений информационной схемы, могут быть довольно разнообразными. В частности, в языке EXPRESS простые ассоциации между типами T_1 и T_2 задаются путем определения в T_1 атрибутов объектного типа T_2, а множественные ассоциации — путем определения в T_1 атрибутов, являющихся простыми или вложенными агрегатами элементов объектного типа T_2. Кроме того, в EXPRESS ассоциативные отношения могут задаваться как некоторые альтернативные состояния атрибутов селективного типа, основанные на простых и множественных ассоциациях всех упомянутых видов. Поскольку селективные конструкции могут содержать в себе ассоциации разных, иногда не родственных объектных типов, то при задании ассоциативного графа необходимо указать, какое именно состояние селективного атрибута порождает соответствующий переход. Один и тот же селективный атрибут, определенный для некоторого объектного типа, может порождать в ассоциативном графе набор переходов из одной и той же вершины, приписанной популяции этого типа. В дальнейшем под ассоциативными будем понимать все объектные типы информационной схемы, а также основанные на них агрегатные и селективные типы.

В основе ассоциативного перехода могут лежать и обратные ассоциативные отношения, которые в EXPRESS реализуются либо явно посредством определения атрибутов INVERSE в ассоциируемом типе, либо неявно путем использования функции USEDIN для любых ассоциируемых объектов, которая по роли прямой ассоциации возвращает множество всех объектов, принимающих в ней участие. Аналогичные возможности предусматриваются и другими языками объектно-ориентированного моделирования. Тем самым, применение введенных понятий в рамках проводимого общего подхода к анализу прикладных объектно-ориентированных данных вполне мотивировано.

С каждым переходом $r \in R, r: p_i \to p_j$ или, более точно с каждым отношением, порождающим переход от p_i к p_j, свяжем правило определения нового состояния популяции $O'(p_j)$ по заданным текущим состояниям популяций $O(p_i)$ и $O(p_j)$.

Правило простого перехода $r \in R_S, r: p_i \to p_j$ задается функцией $O'(p_j) = O(p_j) \cup o \in O(p_i) \mid T(o) \pi T(p_j)$. Функция определяет $O'(p_j)$ как результат объединения текущего состояния популяции p_j и всех объектов популяции p_i, принадлежащих типу популяции $T(p_j)$. Применение правила при $T(p_i) \pi T(p_j)$ обеспечивает выполнение условия принадлежности популяции p_i популяции p_j.

Правило ассоциативного перехода $r \in R_a, r: p_i \to p_j$ задается функцией $O'(p_j) = O(p_j) \cup o \in Adj(p_i, role) \mid T(o) \pi T(p_j)$, которая определяет новое состояние $O'(p_j)$ как результат объединения текущего состояния p_j и всех объектов, связанных с объектами популяции p_i по роли ассоциации перехода *role*. Правила переходов задают некоторые способы формирования и обновления популяций модели.

Хотя отношения принадлежности и ассоциативные отношения сами по себе являются однонаправленными, будем считать, что они могут определять соответствующие переходы в прямом и обратном направлении. Смена направления перехода происходит в результате применения соответствующей операции инвертирования, приписывающей переходу $r \in R, r: p_i \to p_j$ соответствующий статус $\bar{r} \in R, \bar{r}: p_j \to p_i$. Повторное инвертирование возвращает переход в его начальное состояние. Применение операции инвертирования к множеству переходов R означает ее применение ко всем переходам этого множества. Результат операции будем обозначать $\bar{R}$.

Для переходов в инвертированном состоянии также определены функциональные правила. Правило простого обратного перехода $\bar{r} \in R_S, \bar{r}: p_j \to p_i$ задается функцией:

$$O'(p_i) = O(p_i) \cup o \in O(p_j) \mid T(o) \pi T(p_i).$$

Правило ассоциативного обратного перехода $\bar{r} \in R_a, \bar{r}: p_j \to p_i$ задается функцией:

$$O'(p_i) = O(p_i) \cup o \in Adj(p_j, \overline{role}) \mid T(o) \pi T(p_i),$$

в которой под $\overline{role}$ понимается обратное ассоциативное отношение. Правило определяет новое состояние $O'(p_i)$ как результат объединения текущего состояния p_i и всех объектов модели типа $T(p_i)$, связанных с объектами популяции p_j обратным ассоциативным отношением $\overline{role}$. Как было замечено ранее, прямой переход может основываться и на обратном ассоциативном отношении, определяемом информационной схемой для соответствующих объектных типов. Для этих случаев $\overline{role}$ означает применение соответствующего прямого ассоциативного отношения.

В силу определения ассоциативного графа каждой его вершине могут быть инцидентны ребра переходов разных видов, причем как входящих в нее, так и выходящих из нее независимо от направленности порождающих их отношений принадлежности и ассоциативных отношений. Поэтому допускается, что ассоциативные графы могут содержать циклы, в том числе и ориентированные.

Для задач композиции ассоциативных графов, рассматриваемых ниже, важно обеспечить некоторый единый механизм связывания отдельных структур подграфов. Для удобства будем считать, что простыми переходами могут быть соединены популяции, принадлежащие разным графам AG_1 и AG_2 одной и той же прикладной модели M_S. С этой целью переходы содержат ссылки не только на инцидентные им популяции, но и на содержащие их графы.

Определение 5 (Атрибутивный граф — AG). Атрибутивным графом $AG\langle P, R, A\rangle$ для модели M_S будем называть соответствующий ассоциативный граф $RG\langle P, R\rangle$, всем вершинам которого $p \in P$ дополнительно приписаны некоторые подмножества атрибутов $A(p)$, определенных информационной схемой S для соответствующих объектных типов $T(p)$.

В дальнейшем атрибутные графы используются при анализе зависимостей правил целостности от свойств отдельных объектов. При упрощенном анализе на основе ассоциативных графов отдельные атрибуты игнорируются и определяются зависимости от состояния объектов. Упрощенный анализ приводит к результатам, эквивалентным результатам детального анализа для случаев, когда атрибутные множества включают в себя полные наборы атрибутов, определенные информационной схемой для соответствующих типов популяций. В этом смысле ассоциативные графы $RG\langle P, R\rangle$ могут рассматриваться как частные случаи атрибутивных графов $AG\langle P, R, A\rangle$, в которых вершинам приписаны полные наборы атрибутов.

Как и в случае ассоциативных графов, использование селективных атрибутов имеет свои особенности. Если состояния селективного атрибута составляют только ассоциативные типы, то атрибут может порождать ребра переходов в атрибутивном графе. Если всеми состояниями селективного атрибута являются неассоциативные типы данных, то атрибут может быть приписан соответствующим вершинам графа. Наконец, если состояния селективного атрибута составляют как ассоциативные, так и неассоциативные типы данных, то он может одновременно определять ребра переходов в графе и быть приписан некоторым его вершинам.

Определение 6 (Граф навигации — NG). Пусть M_S — прикладная модель информационной схемы S, P — множество популяций модели, $R = R_S \cup R_a$ — множество простых и ассоциативных переходов между ними, A — приписанные вершинам атрибутные множества, $U \subset P$ — популяции инициации, начальные состояния которых $O(u_i) \subset M_S, u_i \in U$ задаются, и $V \subset P$ — целевые популяции, состояния которых $O(v_i) \subset M_S, v_i \in V$ представляют интерес. Тогда графом навигации $NG\langle P, R, A, U, V\rangle$ для модели M назовем ориентированный граф, вершины которого соответствуют популяциям модели P и приписанным им атрибутным множествам A; ребра,

инцидентные парам вершин — переходам между популяциями *R*; ребра, входящие в вершины — начальным состояниям популяций инициации *U*; ребра, выходящие из вершин — конечным состояниям целевых популяций *V*.

В случаях, когда какая-либо составляющая графа навигации не существенна, будем использовать сокращенное обозначение. В частности, если при рассмотрении не важны атрибутивные множества, то будем использовать соответствующую запись $NG\langle P, R, A, U, V\rangle$.

Граф навигации задает способ обхода групп объектов прикладной модели по их отношениям и может интерпретироваться как некоторый параметризуемый объектный запрос. Параметрами запроса являются множества инициации, а результатом — объекты целевых популяций.

Утверждение 1 (Навигационный запрос). Граф навигации $NG\langle P, R, U, V\rangle$ для прикладной модели M_S определяет некоторый объектный запрос в ней. Результатом исполнения запроса является множество объектов целевых популяций модели $O(V) \subset M_S$, полученных в результате задания множеств инициации $O(U) \subset M_S$ и множественного применения правил переходов графа *R*. Параметризуемый запрос, определенный на основе графа навигации, будем в дальнейшем называть навигационным.

Множественность означает многократное (итерационное) сериализованное (последовательное по ребрам переходов) выполнение всех правил до тех пор, пока их применение приводит к изменению состояния хотя бы одной популяции.

Подобная интерпретация навигационного запроса, задаваемого декларативными правилами переходов, нуждается в дополнительных комментариях. Во-первых, введенное определение корректно с точки зрения *детерминизма* результата запроса, поскольку результат не зависит от конкретного способа маршрутизации графа и сериализации выполнения правил переходов. Во-вторых, определение корректно с позиций *конструктивности* процедуры исполнения запроса, поскольку множество объектов прикладной модели конечно и процедура всегда завершается за конечное число циклов. В самом деле, каждый новый цикл выполнения правил перехода неминуемо приводит к добавлению по крайней мере одного объекта в одну из популяций модели, в противном случае процедура исчерпывает себя. С другой стороны, процедура завершится, если каждая популяция будет включать в себя все объекты модели. Это означает, что процедура всегда завершится за конечное число итераций независимо от возможного наличия циклов в графе навигации.

Для графов навигации, являющихся деревьями, всегда (независимо от вершин инициации и целевых вершин) существует способ сериализации, соответствующий однократному обходу вершин графа и последовательному применению правил, ассоциируемых с ними. Для графов, содержащих циклы, может потребоваться многократное итерационное выполнение одних и тех же

правил. В любом случае, всегда существует способ сериализации правил, гарантированно приводящий к получению окончательного результата навигационного запроса. Тривиальный способ состоит в циклическом применении всего набора правил (независимо от их предварительного упорядочивания) вплоть до установления фиксированных множеств объектов во всех популяциях. Для дальнейшего обсуждения возможные способы исполнения навигационных запросов не столь существенны.

Для графов навигации можно конструктивно определить операции инверсии, редукции и слияния следующим образом.

Определение 7 (Инверсия графа навигации). Пусть $NG\langle P, R, A, U, V\rangle$ — граф навигации для модели M_S, тогда инвертированным к нему назовем навигационный граф $\overline{NG}\langle P, R, A, U, V\rangle = NG\langle P, \overline{R}, A, V, U\rangle$.

Согласно определению направления всех переходов в инвертированном графе меняются на противоположные, а популяции инициации и целевые популяции меняются местами. В графической нотации инвертирование графа соответствует замене ориентации всех ребер на противоположную (см. рис. 3). На рисунке проиллюстрирована также возможность композиции отдельных подграфов на основе простых переходов, соединяющих их вершины.

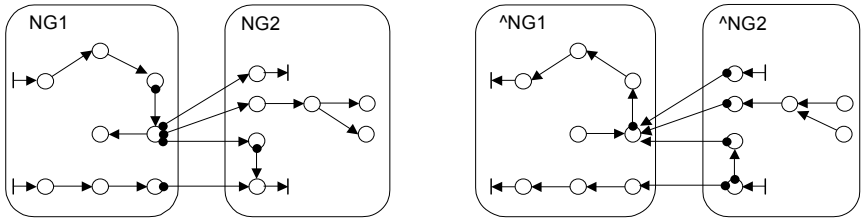

Рис. 3. Пример композиции и инверсии графов навигации.

Определение 8 (Маршрут и достижимость вершин в графе навигации). Маршрутом в графе навигации $NG\langle P, R, U, V\rangle$ назовем упорядоченное подмножество ребер переходов $r_i \in R,\ r_i : p_s(r_i) \to p_t(r_i),\ i = 1,\text{K}\ n$ и инцидентных им вершин популяций $p_s(r_i) \in P,\ p_t(r_i) \in P$ таких, что $p_s(r_{i+1}) = p_t(r_i)$ для всех i = 1,…,*n*-1. Началом маршрута будем называть $p_s(r_1)$, а концом маршрута — $p_t(r_n)$. Вершину $p_{target} \in P$ в графе $NG\langle P, R, U, V\rangle$ будем называть достижимой из $p_{source} \in P$, если существует маршрут в нем с началом в p_{source} и концом в p_{target}.

Определение 9 (Редукция графа навигации). Пусть $NG\langle P, R, U, V\rangle$ — граф навигации для модели M_S. Тогда под редукцией будем понимать граф

навигации $NG' = NG\langle P', R', U, V\rangle$, в котором каждая вершина популяции $p' \in P' \subset P$ и каждое ребро перехода $r' \in R' \subset R$ принадлежат хотя бы одному маршруту в исходном графе $NG\langle P, R, U, V\rangle$, ведущему из вершины инициации $u \in U$ в целевую вершину $v \in V$. Редуцированным по отношению к типу $T \in S$ будем называть граф, к которому применена операция редукции после удаления всех вершин инициации *U*, не принадлежащих типу *T*.

В графической нотации редукция графа навигации выглядит как удаление из него всех вершин и ребер, через которые не проходит ни один маршрут, ведущий из какой-либо вершины инициации в какую-либо целевую вершину. Реализация операции может быть основана на предварительном анализе одновременной достижимости вершины графа из множества инициации и целевого множества из вершины. Ниже на Рис. 4 приведен пример графа навигации и его редуцированного графа.

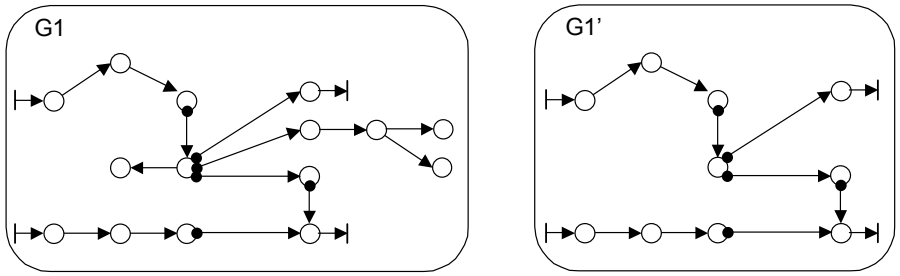

Рис. 4. Пример графа навигации и его редукции.

Определение 10 (Слияние графов навигации). Пусть $NG\langle P, R, A, U, V\rangle$, $NG'\langle P', R', A', U', V'\rangle$ — графы навигации для модели M_S. Две вершины $p \in P$ и $p' \in P'$ подобны друг другу, если они имеют тождественный тип $T(p) = T(p')$; два ребра назовем подобными друг другу, если они инцидентны парам соответствующих подобных вершинам и имеют идентичные имена и типы переходов. Два маршрута эквивалентны, если они составлены из эквивалентных последовательностей подобных вершин и ребер. Две вершины $p \in P$ и $p' \in P'$ назовем эквивалентными относительно пары подобных вершин $u \in U$, $u' \in U'$, если они достижимы из них по эквивалентным наборам маршрутов. Ребра эквивалентны, если они инцидентны парам эквивалентных вершин.

Тогда под операцией слияния графов относительно пары подобных вершин $u \in U$, $u' \in U'$ будем понимать преобразование графов к представлению $NG''\langle P'', R'', A'', U'', V''\rangle$ такому, что каждой паре эквивалентных элементов, а также каждому неэквивалентному элементу в графах *NG*, *NG'* соответствует

единственный подобный элемент графа *NG''* $p'' \in P''$, $r'' \in R''$, $u'' \in U''$, $v'' \in V''$, а приписанные вершинам атрибутные множества A'' формируются путем объединения множеств *A*, A' соответствующих вершин графов *NG*, *NG'*.

Определение операции слияния для графов навигации конструктивно и допускает ряд алгоритмических реализаций. Одна из них состоит в предварительном поиске пар потенциально эквивалентных вершин и ребер путем одновременного согласованного обхода графов по подобным направленным ребрам переходов от вершин инициации. Затем составленный список пар вершин последовательно просматривается с целью обнаружения каких-либо ребер, входящих в вершины списка и одновременно инцидентных вершинам, не включенным в список. Такие пары вершин из списка удаляются, а процедура поиска повторяется вплоть до исчерпывания удаляемых пар. Оставшиеся в списке пары вершин, а также инцидентные им пары ребер являются эквивалентными и поэтому при слиянии графов отображаются в соответствующие подобные элементы результирующего графа. Все вершины, не вошедшие в список, и инцидентные им ребра добавляются в результирующий граф.

Ниже на Рис. 5 приведены графы навигации *NG*, *NG'* и граф *NG''*, полученный в результате их слияния относительно подобных вершин *u* и u'.

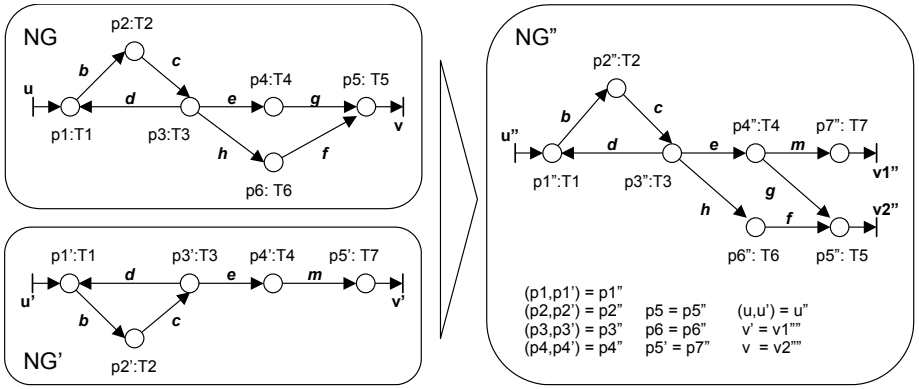

Рис. 5. Пример преобразования графов навигации на основе операции слияния.

Операция слияния графов навигации имеет прозрачную интерпретацию в терминах навигационных запросов. Результат исполнения запроса на основе результирующего графа слияния включает в себя результаты отдельных запросов, основанных на графах-операндах. В самом деле, введенное понятие эквивалентности в графах навигации предполагает, что множество объектных популяций, полученное в результате исполнения запроса, эквивалентно множествам, полученным в результате навигации по эквивалентным маршрутам из подобных вершин инициации с эквивалентными начальными состояниями.

Граф навигации и перечисленные операции достаточно естественно применяются к задачам инкрементальной верификации формально специфицированных объектно-ориентированных данных и ограничений на них. Рассмотрим некоторые приложения.

Определение 11 (Граф зависимости — DG). Пусть $\text{Procedure}(x_1, x_2, \text{K}, x_n)$ — некоторая процедура, специфицированная на языке EXPRESS. Ее графом зависимости назовем граф навигации $DG\langle P, R, A, U, V\rangle$, построенный на основе следующего набора правил:

- Все формальные параметры и локальные переменные ассоциативного типа (объектные типы, а также основанные на них агрегатные и селективные конструкции), представляются в графе вершинами P. Каждый формальный параметр и локальная переменная ассоциативного селективного типа порождает множество вершин P, соответствующих всем его ассоциативным состояниям. Вершины, соответствующие формальным параметрам процедуры, включаются также во множество вершин инициации U, если процедура реализует некоторое правило целостности;
- Все регулярные выражения путей с использованием атрибутов ассоциативного типа представляются в графе направленными ребрами ассоциативных переходов R_a и инцидентными им вершинами популяций P соответствующего объектного типа. Каждому сегменту пути соответствует одно ребро и одна вершина. Если выражение пути реализуется на основе ассоциативного селективного атрибута, то каждому его ассоциативному состоянию соответствует одно ребро и одна вершина. Ребра направлены от вершин популяций объектов, содержащих ассоциативную связь, к вершинам популяций ассоциируемых объектов. При использовании в выражениях обратных ассоциативных отношений переходу приписывается соответствующий статус инвертирования;
- Все операции явного и неявного приведения типа для объектов, являющихся формальными параметрами процедуры или локальными переменными, представляются в графе направленными ребрами простых переходов R_S и инцидентными им вершинами популяций P соответствующего приведенного объектного типа. Под неявным приведением типа понимается соответствующее использование объектных ссылок в телах управляющих операторов с условием на принадлежность подтипу;
- Теоретико-множественные операции над операндами агрегатного ассоциативного типа представляются ребрами простых переходов R_S от вершин популяций, соответствующих операндам операции, к новой построенной вершине, являющейся результатом операции. Объектный тип, приписываемый вершине, является супертипом всех объектных типов, связанных с вершинами операндов. При объединении множеств ребра конструируются для каждого операнда, при вычитании — только для первого операнда, при пересечении — для любого из них;
- Все операции использования значений атрибутов неассоциативного типа в правой части оператора присваивания порождают атрибутивные множества A, приписываемые соответствующим вершинам популяций объектов P, для которых эти операции были использованы;
- Все операции использования значений атрибутов ассоциативных типов в правой части оператора присваивания представляются в графе направленными ребрами простых переходов R_S от вершин популяций объектов, атрибуты которых присваиваются, к вершинам популяций объектов, которым эти значения присваиваются. При присваивании значений ассоциативного селективного атрибута строится множество ребер простых переходов R_S, каждому из которых соответствует свое ассоциативное состояние атрибута;
- Все вызовы процедур с параметрами ассоциативных типов представляются в графе ребрами простых переходов R_S. Ребра соединяют вершины популяций, являющиеся фактическими параметрами процедуры с вершинами популяций, соответствующих формальным параметрам вызываемой процедуры в ее подграфе зависимости. Таким образом, построение графа зависимости для процедуры включает в себя построение и композицию подграфов для всех используемых в ней внешних процедур;
- Все построенные вершины P также включаются во множество целевых вершин V;
- Все остальные операторы и операции языка EXPRESS игнорируются.

На основе перечисленных правил граф зависимости может быть непосредственно построен не только для процедур, но и для функций, производных атрибутов, локальных и глобальных правил, специфицированных на языке EXPRESS. Сигнатуры глобальных правил точно соответствуют обобщенной структуре задания параметров в процедурах EXPRESS. Фактическими параметрами процедур глобальных правил являются экстенты типов прикладной модели, подлежащей верификации. Производные атрибуты и локальные правила могут интерпретироваться как соответствующие процедуры, единственным входным параметром которых является сам типизированный объект, для которого эти атрибуты и правила определены. Фактическим параметром локальных правил является объект, подлежащий верификации. Любая используемая при верификации функция также может быть представлена в виде соответствующей процедуры, одним из параметров которой является возвращаемое значение функции, а все остальные — ее входные параметры. Таким образом, определение графа зависимости имеет общий характер для анализа спецификаций ограничений.

Утверждение 2 (Локализация объектов, используемых правилом). Пусть M_S — прикладная модель и $rule \in S$ — некоторое правило целостности

данных, специфицированное для информационной схемы S, $DG\langle P,R,A,U,V\rangle$ — его граф зависимости. Тогда при задании фактических параметров правила в качестве множества инициации $O(U) \subset M_S$ результат исполнения навигационного запроса, основанного на графе зависимости, включает в себя все объекты прикладной модели M_S, используемые при верификации данного правила.

В самом деле, опуская условные операторы и операции индексирования агрегатов, участвующие в определении правила целостности, граф зависимости учитывает лишь статические отношения между группами объектов (ассоциативные отношения и отношения принадлежности множеств) и по построению приводит к расширительному запросу объектов, участвующих в верификации правила. При этом в атрибутные множества попадают также все атрибуты объектов, потенциально используемые при верификации правила независимо от фактических, определяемых в процессе исполнения значений предикатов для управляющих операторов языка EXPRESS. Таким образом, утверждение 2 может конструктивно применяться для локализации групп объектов, участвующих в верификации правил целостности.

Определение 12 (Граф влияния — IG). Пусть M_S — прикладная модель и $rule \in S$ — некоторое правило целостности данных, специфицированное для информационной схемы S, DG — его граф зависимости. Графом влияния популяций объектного типа $T \in S$ для правила *rule* назовем соответствующий ему граф навигации $IG\langle P,R,A,U,V\rangle$, полученный в результате инвертирования графа зависимости и его редукции относительно заданного типа T.

Утверждение 3 (Локализация правил с участием объекта). Пусть M_S — прикладная модель и $rule \in S$ — некоторое правило целостности данных, специфицированное для информационной схемы S, $o \in M_S$ — некоторый объект прикладной модели типа $T(o) = T$ и $IG\langle P,R,A,U,V\rangle$ — граф влияния популяций типа T для правила *rule*. Тогда при задании объекта o в качестве начального состояния популяций инициаций U ($u_i \in U$, $O(u_i) = o$) результат исполнения навигационного запроса, основанного на графе влияния, включает в себя все объекты модели M_S, которые в качестве фактических параметров правила участвуют в его верификации вместе с заданным объектом o.

Поскольку при построении графа влияния каждый переход в графе зависимости инвертируется, то соответствующий объектный запрос реализует обратную навигацию по прикладной модели от заданного объекта ко всем объектам, участвующим в верификации правила в качестве фактических параметров. Поскольку при построении графа зависимости были игнорированы условные операторы и операции индексирования агрегатов, прохождение каждого перехода в обратном направлении лишь расширяет множество объектов, из которых заданный объект мог быть достижим по маршрутам

графа зависимости в результате прямой навигации. Это означает, что результаты запроса будут включать все объекты, для которых выполнение правила целостности может быть связано с состоянием анализируемого объекта. При этом на выполнение правила могут влиять лишь те свойства объекта, которые представлены в графе влияния соответствующими атрибутными множествами вершин инициации и инцидентными им ребрами ассоциативных переходов.

Проиллюстрируем построение графов зависимости и влияния для глобального правила информационной схемы, приведенной на Рис. 2.

На Рис. 6 приведен построенный для глобального правила *CurveDimensionsCoincide* граф зависимости. Участвующие в определении правила производные атрибуты *Curve::Dim*, *CurveSegment::Dim*, *CompositeCurve::Dim* и функция *CalculateCurveDimension* представлены собственными подграфами, связь с которыми в композиционном представлении графа реализуется посредством простых переходов, соответствующих передаче фактических параметров ассоциативного типа в функции. Простые переходы в подграфе функции *CalculateCurveDimension* соответствуют используемым операциям неявного приведения типа. В данном примере абстрактный тип геометрической кривой *Curve* приводится к соответствующим подтипам линий *Line*, конических кривых *Conic*, ломанных *Polyline*, композитных кривых *CompositeCurve*. Построенный граф правила содержит также два ассоциативных перехода, связанных с использованием соответствующих ассоциативных отношений *CompositeCurve::Segments* и *CurveSegment::ParentCurve* между объектами внутри функций. Цикличность построенного графа зависимости является следствием рекурсивного использования функций в определении глобального правила.

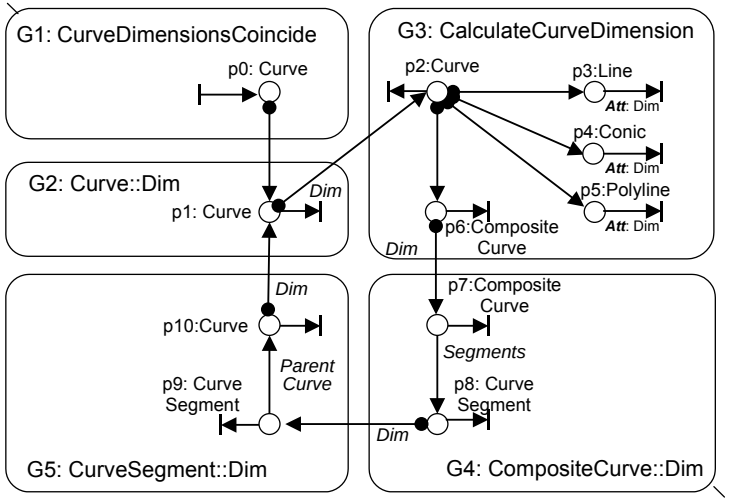

Рис. 6. Пример графа зависимости для глобального правила.

На рисунке 7 представлен соответствующий граф влияния популяций типа *Conic* для глобального правила *CurveDimensionsCoincide*. Граф построен в результате применения описанных выше операций инвертирования и редукции относительно заданного типа. В результате инвертирования все ребра в нем поменяли направления, а в результате редукции были удалены популяции *p3: Line*, *p5:Polyline* и инцидентные им переходы. Популяция *p6: CompositeCurve* и связанный с нею циклический маршрут *Curve->CompositeCurve->CurveSegment->Curve* в графе влияния сохранился. Связанно это с тем, что объект типа *Conic* может являться сегментом кривой типа *CompositeCurve* и влиять на вычисление правила через объекты, входящие в указанный циклический маршрут.

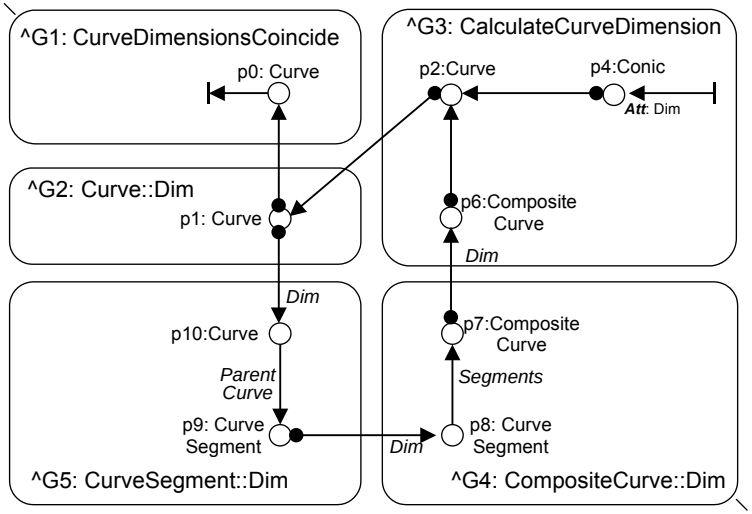

Рис. 7. Пример графа влияния популяций для глобального правила.

6. Инкрементальная верификация модели

Прикладная модель не находится постоянно в одном и том же состоянии и изменяется на протяжении своего жизненного цикла, т.е. переходит из одного состояния в другое. Подобные смены состояний модели осуществляются при выполнении трех основных операций жизненного цикла: вставке в нее новых объектов (INSERT), модификации (UPDATE) или удалении (DELETE) существующих объектов. Естественно, что подобные изменения могут привести к нарушению целостности данных модели. Поскольку реальные модели промышленных данных состоят из нескольких сотен тысяч и даже миллионов объектов, полная верификация модели по приведенному выше алгоритму требует больших затрат вычислительных ресурсов. Поэтому возникает естественное желание попытаться сузить (локализовать) область проверки условий целостности данных.

Отсюда вытекает задача инкрементальной верификации, заключающаяся в выделении подмножества объектов прикладной модели, которые потенциально могут нарушить поставленные ограничения в результате ее перехода в новое состояние при условии, что в предыдущем состоянии модель была валидна. Решение задачи основано на анализе спецификаций ограничений, ставящих условия целостности данных. В рамках настоящего исследования ограничимся только статическим анализом спецификаций, поскольку более тонкий динамический анализ приводит к существенным увеличениям вычислительных затрат, иногда сопоставимых с затратами на полную верификацию. Это замечание в равной степени распространяется и на статический анализ, однако в силу экономичности его применение может оказаться оправданным в задачах верификации масштабных объектно-ориентированных моделей.

Утверждения 1–3 предоставляют конструктивную основу для построения семейства алгоритмов инкрементальной верификации объектно-ориентированных данных. В их основе лежит идея представления множественных изменений в прикладной модели последовательностью элементарных изменений и объединением локализованных на каждом шаге групп объектов модели и соответствующих правил целостности, которые могли бы быть нарушены вследствие принятых изменений. Под элементарным понимается изменение, затрагивающее только один объект модели. Элементарными являются модификация, удаление одного из существующих объектов модели или создание одного нового объекта модели. На каждом шаге алгоритмов множества объектов и правил, подлежащих дальнейшей верификации, расширяются с учетом последовательного принятия всех изменений модели.

В зависимости от степени локализации области проверки ограничений можно выделить два варианта постановки задачи инкрементальной верификации:

1) определить подмножество объектов модели, которые потенциально могут нарушить конкретные ограничения в ней в результате проведенного элементарного изменения, при этом проверке подлежат только те правила, которые являются зависимыми от состояния измененного объекта;
2) определить подмножество объектов модели, которые потенциально могут нарушить произвольные ограничения в ней в результате проведенного элементарного изменения, при этом проверке подлежат все без исключения правила для выделенного подмножества объектов.

Решение задачи в первой постановке требует больших затрат вычислительных ресурсов, но позволяет лучше локализовать область проверки и минимизировать количество проверяемых правил.

Рассмотрим алгоритмы решения задачи инкрементальной верификации в каждом из вариантов постановок. Все алгоритмы используют результаты предварительного статического анализа спецификаций информационной схемы, в ходе которого для всех локальных и глобальных правил целостности строятся соответствующие графы зависимости и влияния. Последние строятся относительно всех объектных типов информационной схемы и организуются в

виде специального репозитория. По заданному типу объекта средствами репозитория определяется набор правил, в верификации которых конкретный объект мог принимать участие, и конструируются соответствующие им графы влияния, на основе которых могут быть локализованы объекты, являющиеся фактическими параметрами анализируемых правил целостности. Для локальных правил репозиторий хранит также слитое представление графов влияния по отношению к вершинам инициации соответствующего объектного типа. Подобное представление используется в ходе решения задачи инкрементальной верификации во второй постановке.

Итак, рассмотрим алгоритм решения задачи в первой постановке. В примененной алгоритмической нотации *Obj* означает изменившийся, удалившийся или добавленный объект, *Obj′* — старое состояние объекта (до выполнения операции UPDATE), *operation* — операция, примененная к объекту *Obj*: INSERT, DELETE или UPDATE. Результат работы алгоритма формируется как множество правил, подлежащих проверке. Для локальных правил дополнительно указывается объект, служащий его фактическим параметром. Ограничения уникальности атрибутов, устанавливаемые в отдельных объектных типах, будем относить к глобальным правилам, поскольку они также проверяются на экстентах соответствующих типов.

Особенностью данного метода является локализация правил с учетом изменения состояний отдельных атрибутов объекта. Чтобы исключить лишний анализ глобального правила в тех случаях, когда оно уже попало в результат при анализе предыдущих элементарных изменений, выполняется дополнительная проверка. Выполнять аналогичную проверку для локальных правил нет необходимости, поскольку в результате анализа различных элементарных изменений будут получаться разные (возможно, пересекающиеся) множества пар «объект – локальное правило».

```
if operation is DELETE
  for each object inversly adjacent to Obj
     for each attribute in object
        if attribute is explicit and association type of attribute is
subtype of T(Obj)
           add check AttributeConstraints(object, attribute) to result
  for each object adjacent to Obj
     for each attribute in object
        if attribute is inverse and association type of attribute is
subtype of T(Obj)
           add check AttributeConstraints(object, attribute) to result
else
  for each changed attribute
     add check AttributeConstraints(Obj, attribute) to result
     if there is inverse attribute for attribute
        for each object adjacent to Obj by attribute
           add check AttributeConstraints(object, inverse attribute) to
result
```



```
        if operation is UPDATE
           for each object adjacent to Obj′ by attribute
              add check AttributeConstraints(object, inverse attribute) to
result
for each LocalRule represented by IG-graph with respect to T(Obj)
  if at least one changed attribute of Obj is in attribute set of initiation
node
     execute navigation query based on IG-graph and initial state Obj
     for each found object
        add check LocalRule(object) to result
     if operation is UPDATE
        execute navigation query based on IG-graph and initial state Obj′
        for each found object
           add check LocalRule(object) to result
for each GlobalRule represented by IG-graph with respect to T(Obj)
  if not (GlobalRule in result or (operation is DELETE and GlobalRule is
UniquenessRule))
     if at least one changed attribute of Obj is in attribute set of
initiation node
        execute navigation query based on IG-graph and initial state Obj
        if there is at least one found object
           add check GlobalRule to result
        else if operation is UPDATE
           execute navigation query based on IG-graph and initial state Obj′
           if there is at least one found object
              add check GlocalRule to result
```

Рассмотрим алгоритм решения задачи во второй постановке. В отличие от предыдущего алгоритма результат работы данного формируется как множество объектов, все локальные правила которых подлежат проверке, а также множество глобальных правил. Особенностью данного метода является использование для анализа локальных правил слитого представления графов влияния по отношению к вершинам инициации типа объекта *Obj*.

```
if operation is DELETE
  for each object inversly adjacent to Obj
     add object to result
else
  add Obj to result
for each object adjacent to Obj
  add object to result
if operation is UPDATE
  for each object adjacent to Obj′
     add object to result
get merged IG-graph with respect to T(obj)
execute navigation query based on IG-graph and initial state Obj
for each found object
  add object to result
if operation is UPDATE
  execute navigation query based on IG-graph and initial state Obj′
  for each found object
```

```
    add object to result
for each GlobalRule represented by IG-graph with respect to T(Obj)
  if not (GlobalRule in result or (operation is DELETE and GlobalRule is
UniquenessRule))
    execute navigation query based on IG-graph and initial state Obj
    if there is at least one found object
       add check GlobalRule to result
    else if operation is UPDATE
       execute navigation query based on IG-graph and initial state Obj'
       if there is at least one found object
          add check GlocalRule to result
```

7. Заключение

Таким образом, рассмотрены задачи верификации объектно-ориентированных данных. Для задачи инкрементальной верификации на основе теории графов предложен формальный аппарат, а также разработаны методы, позволяющие локализовать область потенциальных нарушений на основе статического анализа спецификации ограничений. Методы предоставляют весомую альтернативу полной верификации прикладных данных в тех случаях, когда их изменения не затрагивают глобальные ограничения, заданные на обширных популяциях тесно связанных между собой объектов. По-видимому, количественные критерии оптимальности применения методов инкрементальной верификации должны учитывать особенности информационной схемы, а их выработка представляет собой интересную научную задачу. Важным также представляется обобщение разработанных методов верификации на случай одновременных множественных изменений.

Литература

1. K. Ramamritham, P. Chrysanthis. Executive Briefing: Advances in Concurrency Control and Transaction Processing. IEEE Computer Society Press, 1997.
2. E. Mayol, E. Teniente. A Survey of Current Methods for Integrity Constraint Maintenance and View Updating. // Advances in Conceptual Modeling: ER '99 Workshops on Evolution and Change in Data Management, Reverse Engineering in Information Systems, and the World Wide Web and Conceptual Modeling, Paris, France, November 15–18, 1999, pp. 62–73.
3. M. A. Pacheco. Dynamic integrity constraints definition and enforcement in databases: a classification framework. // Proceedings of the IFIP TC-11 Working Group 11.5 First Working Conference on Integrity and Internal Control in Information Systems, Zürich, Switzerland, December 1997, pp. 65–87.
4. P. Fraternali, S. Paraboschi. A Review of Repairing Techniques for Integrity Maintenance. // Proceedings of the First International Workshop on Rules in Database Systems (RIDS'93), Edinburg, 1993, pp. 333–346.
5. ISO 10303-11: 1994, Industrial automation systems and integration — Product data representation and exchange — Part 11: Description methods: The EXPRESS language reference manual.

6. Unified Modeling Language (UML), Version 1.5, 2003, http://www.omg.org/cgi-bin/doc?formal/03-03-01.
7. J. Faria. Data-driven Rules for the Maintenance of Derived Data and Integrity Constraints in User Interfaces to Databases. INESC Technical Report RI 1/99. Porto, Portugal, May 1999.
8. R. Breu, U. Hinkel, C. Hofman, C. Klein, B. Paech, B. Rumpe, V. Thurner. Towards a Formalization of the Unified Modeling Language. // Proceedings of ECOOP'97 — 11th European Conference on Object-Oriented Programming, ed. M. Aksit and S. Matsuoka, LNCS 1241, Springer-Verlag, 1997, pp. 344–366.
9. M. Richters, M. Gogolla. On Formalizing the UML Object Constraint Language OCL. // Proceedings of ER'98 — 17th International Conference on Conceptual Modeling, ed. Tok-Wang Ling, LNCS 1507, Springer-Verlag, 1998, pp. 449–464.