Объектно-ориентированное программирование в ограничениях: новый подход на основе декларативных языков моделирования данных

Семенов В.А., Ильин Д.В., Морозов С.В., Сидяка О.В.

Аннотация. Объектно-ориентированное программирование в ограничениях (ООСР) сочетает две ортогональные, но комплементарные парадигмы программирования, а именно: объектно-ориентированное программирование (ООР) и логическое программирование в ограничениях (CLP). Несмотря на привлекательность идеи синтеза парадигм и известные попытки реализации, до сих пор не существует единого понимания, какие конструктивные очертания она может приобрести при дальнейшей проработке и развитии. Ключевыми вопросами при этом остаются выразительность описания прикладной задачи в ограничениях и ее алгоритмическая разрешимость. В настоящей работе предлагается и обсуждается новый системный подход к реализации ООСР на основе использования декларативных языков моделирования данных.

1. Введение

В последние годы наблюдается ренессанс идей логического программирования в ограничениях (CLP) в таких актуальных научных областях и дисциплинах, как интеллектуальные системы принятия решений (логические нейронные сети), компьютерная графика (декларативные сценарные модели и графические интерфейсы), экономические модели (недоопределенные вычисления), системы автоматизированного проектирования (параметрическое моделирование), информационные системы (активные базы данных и управление целостностью), семантический поиск данных (дескриптивные логики), верификация и тестирование программного обеспечения (временные логики), системы коллективной инженерии (полисиллогистический вывод) [1, 2]. Многие крупные индустриальные компании проявляют интерес к этой теме, а ACM признала ее одним из стратегических направлений исследований.

Вместе с тем, следует констатировать, что как технология общего назначения CLP не получило широкое распространение, а в перечисленных выше областях разрабатываются и используются специализированные языки и программные системы. Их главным недостатком и принципиальным

ограничением является невозможность специфицировать и решать задачи относительно переменных, являющихся сложно структурированными, семантически согласованными данными — в частности, объектно-ориентированными данными, определяемыми актуальными информационными моделями и международными стандартами ISO STEP [3], OMG MDA [4], W3C Semantic Web [5].

Эти факторы обуславливают выработку нового системного подхода, который бы, с одной стороны, обеспечил решение широкого класса задач в ограничениях, а с другой — приблизил способы их постановки к популярным универсальным технологиям объектно-ориентированного моделирования. Применение для этих целей универсальных языков моделирования данных EXPRESS [6], ODL/OQL [7], UML/OCL [8, 9], OWL [10], получивших признание и распространение в широких научных и индустриальных сообществах, приобретает особую привлекательность. В самом деле, прикладную задачу в ограничениях можно описать путем определения пользовательских типов данных и задания на них разнообразных семантических правил. Развитые средства алгебраической спецификации правил вместе с предопределенными конструкциями для задания областей значений переменных, типов и размеров коллекций, кардинальности объектных типов, обязательности атрибутов, свойств уникальности атрибутов, условий наличия или отсутствия ассоциативных циклов, позволяют сделать это относительно простым и наглядным образом.

В ряде случаев целесообразным представляется использование стандартных информационных моделей [11–14], разработанных индустриальными консорциумами для таких областей как программная инженерия, информационные технологии, машиностроение, атомная энергетика, авиационная и космическая промышленность, судостроение, архитектура и строительство, нефтегазовый комплекс, фармацевтика. Постановка и решение задач программирования в ограничениях с их применением может приводить к возникновению новых, индустриально значимых приложений. К числу таких приложений можно отнести, например, задачу тестирования интероперабельности приложений, осуществляющих обмен данными и функционирующих в составе интегрированных программных комплексов, задачу управления целостностью прикладных данных в развитых вычислительных и информационных системах, использующих оптимистические модели транзакций, или задачу верификации программной модели в соответствии с подготовленными контрактными спецификациями. Во всех упомянутых случаях речь идет о формировании (или согласованной модификации) коллекций структурированных данных по заданной спецификации объектно-ориентированной модели. Полученные данные должны при этом соответствовать заданной модели и удовлетворять ее семантическим правилам.

Естественно, что универсальность и декларативность перечисленных выше языков моделирования порождает проблему алгоритмической разрешимости систем ограничений, специфицированных на них. Идентификация математической задачи и выбор релевантного алгоритма решения при возможном переопределенном или недоопределенном характере системы ограничений являются общими проблемами для большинства подходов. Дополнительным фактором сложности, привносимым языками моделирования, является сам класс задач логического программирования в ограничениях на множествах объектно-ориентированных данных. Данный класс, обозначаемый в дальнейшем CLP(O), предполагает дополнительную структуризацию переменных по сравнению с традиционными постановками в булевых, рациональных, вещественных числах CLP(B), CLP(Q), CLP(R) и на конечных доменах CLP(FD) соответственно. Кроме того, возможность алгебраической спецификации произвольных семантических правил приводит к необходимости совместного анализа и разрешения неоднородных ограничений, что крайне затруднительно при использовании традиционных методов, ориентированных на частные математические постановки. Наконец, задание правил на типах данных, а не только на отдельных переменных, порождает проблему формирования множества неизвестных переменных, в данном случае — коллекций объектов и элементов данных, относительно которых задача в ограничениях должна решаться.

Отметим, что как самостоятельная научная дисциплина программирование в ограничениях охватывает три направления, а именно: разрешение ограничений CSP (Constraint Satisfaction Problem) [15], логическое программирование в ограничениях CLP (Constraint Logic Programming) [1] и конкурентное программирование в ограничениях CCP (Concurrent Constraint Programming) [16]. Несмотря на особенности в постановках и методах решения задач, все три направления тесно связаны между собой. В рамках обсуждаемого подхода к ООСР мы не видим причин отказываться ни от одного из них. Применяя обозначение CLP(O), мы лишь подчеркиваем роль методов логического программирования как конструктивной основы для выстраивания перспективных вычислительных стратегий разрешения систем неоднородных ограничений, формально специфицированных на декларативных языках объектно-ориентированного моделирования данных.

В разделе 2 приводится несколько примеров постановки и решения классической математической задачи о ферзях с использованием парадигм логического, функционального и объектно-ориентированного программирования. В разделе 3 предлагаемый декларативный подход сравнивается с известными технологиями программирования в ограничениях и близкими им технологиями генерации тестовых данных с учетом контекстных ограничений. Общая вычислительная стратегия решения задач в ограничениях строится и обсуждается в разделе 4. В заключении указываются основные направления исследований для детальной алгоритмической проработки предлагаемого подхода и его практической реализации.

2. Некоторые примеры

Напомним, что задачи в ограничениях обычно описываются путем определения множества неизвестных переменных и зависимостей между ними. При этом процесс решения заключается в локализации областей значений переменных или в поиске значений, удовлетворяющих заданным зависимостям. Перечислим основные особенности их постановки.

Спецификации ограничений в рассматриваемых задачах носят декларативный характер, исключающий явное описание способов решения задачи (в качестве исключения здесь следует указать императивные языки ограничений, на которых пользователь описывает и некоторые методики решения). Спецификации обладают свойством нейтральности по отношению к неизвестным переменным и возможным альтернативным способам выражения и пересчета их друг через друга. Спецификации обладают также свойством аддитивности, предполагающим, что порядок задания ограничений не существенен для постановки исходной задачи, логические условия которой всегда могут быть представлены в эквивалентной конъюктивной форме. Исключение составляют так называемые иерархические системы, в которых разрешение ограничений осуществляется поэтапно в соответствии с предварительно назначенными приоритетами. Наконец, спецификации ограничений носят неопределенный характер с точки зрения их алгоритмической разрешимости. Поскольку система ограничений может быть недоопределена или переопределена, постановка задачи в ограничениях априори не предполагает ни существование, ни единственность решения, и его поиск должен вестись с учетом всех этих обстоятельств. Обсудим перечисленные свойства в контексте постановки и решения задач в классе CLP(O).

С этой целью рассмотрим классическую математическую задачу о ферзях и приведем возможные способы ее описания и решения на языках логического, функционального и объектно-ориентированного программирования. Задача формулируется следующим образом: необходимо расставить на шахматной доске восемь ферзей так, чтобы ни один из них не оказался под боем остальных фигур. Один из вариантов решения приведен на Рис. 1.

Положение каждой фигуры на шахматной доске характеризуется парой координат *x*/*y*, каждая из которых принимает целые значения от 1 до 8 (здесь мы отступим от традиционной буквенно-цифровой нотации ходов в шахматной партии, подобной “e2-e4”, и используем оператор «/» для объединения координат в один элемент списка). Тогда шахматная позиция представляется списком вида [*x*1/*y*1, *x*2/*y*2, *x*3/*y*3, *x*4/*y*4, *x*5/*y*5, *x*6/*y*6, *x*7/*y*7, *x*8/*y*8]. В этом представлении решение на Рис. 1 выглядит как [1/4, 2/2, 3/7, 4/3, 5/6, 6/8, 7/5, 8/1].

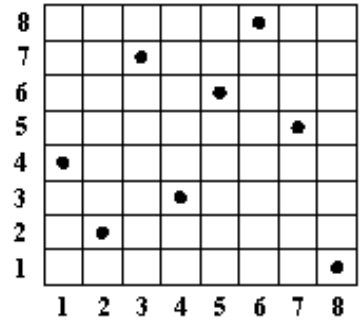


Рис. 1. Один из вариантов решения задачи о восьми ферзях

Принимая во внимание необходимость расположения ферзей на разных вертикалях (или на разных горизонталях), представление списка может быть сразу уточнено как [l/*y*l, 2/*y*2, 3/*y*3, 4/*y*4, 5/*y*5, 6/*y*6, 7/*y*7, 8/*y*8]. Таким образом, задача сводится к определению горизонтальных (или вертикальных) координат, исключающих расположение нескольких фигур на одних и тех же линиях шахматной доски.

Обсудим, каким образом задача может быть описана и решена на языке логического программирования Prolog. Уточненный список может использоваться в качестве шаблона решения, к которому последовательно применяются правила вывода в рамках общей схемы рекурсивного программирования. Будем считать список координат согласованным, если он определяет позицию, в которой ни один из ферзей не находится под боем. Возможны два случая:

Случай 1. Список пуст. Очевидно, пустой список является согласованным при отсутствии фигур и возможных атак.

Случай 2. Список не пуст. Тогда он представим в виде [*x*/*y* | *Остальные*], где *x*/*y* задает положение первого ферзя, а список "*Остальные*" — положение остальных. Определим необходимые условия, при которых непустой список является согласованным. Во-первых, список "*Остальные*" должен быть согласованным. Во-вторых, значения *x* и *y* должны принадлежать множеству целых чисел от 1 до 8 включительно. В-третьих, значения *x* и *y*, а также их разности (*x*-*y*) и суммы (*x*+*y*) не должны совпадать с соответствующими значениями из списка "*Остальные*".

Данные условия могут быть описаны на языке Prolog [17] в виде следующей программы (см. Рис. 2).


```
?-  шаблон( S), решение( S).
решение( [ ] ).
решение( [x/y | Остальные ] ) :-
       % Первый ферзь на поле x/y,
       % остальные ферзи на полях из списка
         Остальные
     решение( Остальные ),
     принадлежит( y, [1, 2, 3, 4, 5, 6, 7, 8] ),
     не_бьет( x/y | Остальные ).  % Первый ферзь
     не бьет остальных
не_бьет( x/y, [ ]).              % Некого бить
не_бьет( x/y, [x1/y1 | Остальные] ) :-
            y =\= y1,            % Разные
            y-координаты
            y1 - x1 =\= y - x,   % Разные
            диагонали
            y1 + x1 =\= y + x,
            не_бьет( x/y, Остальные).
принадлежит( x, [x | L] ).
принадлежит( x, [y | L] ) :-
            принадлежит( x, L).
% Шаблон решения
   шаблон([1/y1,2/y2,3/y3,4/y4,5/y5,6/y6,7/y7,8/y8]).
```

Рис. 2. Программа решения задачи о ферзях на языке Prolog

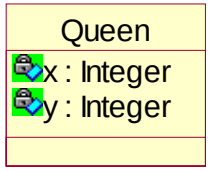
Обсудим возможность описания этой же задачи на языке ConstraintLisp [18], представляющим собой расширение стандарта ANSI Common Lisp [19] для программирования в ограничениях. Нововведением в ConstraintLisp является набор функций, который позволяет описывать арифметические ограничения в рамках функциональной и объектно-ориентированной парадигм, поддерживаемых языком ANSI Common Lisp. В программе на рисунке 3 определяется класс Queen с двумя атрибутами x и y для задания положения фигуры на шахматной доске. Для представления шахматной позиции используется массив Chessboard, в котором хранится восемь экземпляров класса Queen. Массив конструируется и инициализируется в результате вызова соответствующей функции make-queens. Отметим, что содержательные ограничения задачи определяются в виде вспомогательной функции constraints с формальными параметрами, соответствующими координатам анализируемых фигур. При этом фактическое назначение ограничений объектам массива Chessboard осуществляется в теле вложенного цикла, где описанный вид ограничений применяется ко всем парам итерируемых объектов. Решение генерируется и выводится на печать при вызове специальных функций generateObjs и printArrayObjs.

```
;;; определение класса Queen
(defclass Queen ( )
    (( x :initarg :x :accessor x ) ;;; координата по
     горизонтали
    ( y :initarg :y :accessor y))) ;;; координата по
     вертикали
;;; основная процедура поиска решения
(define Queens ( )
(let ((Chessboard ( make-queens)) ) ;;; создаем массив
из восьми ферзей

    (cldotimes ( I 7 ) ;;; начальное значение переменной
    I 0

        (cldotimes ( J (- 7 I )) ;;; начальное значение
        переменной J 0

            (constraints ( x (aref Chessboard I ))

                ( x (aref Chessboard (+ J I 1 ))) (1+ I)
                (+ J I 2 ))))

      ( generateObjs Chessboard ) ;;; генерация решения
      ( printArrayObjs Chessboard ))) ;;;
      вывод результатов
;;; ограничения отсутствия вертикальных и диагональных а
так
(define constraints (x1 x2 y1 y2)
    (constr (/= x1 x2)) ;;; по вертикали
    (constr (/= (+ x1 y1 ) (+ x2 y2 )))  ;;; по главной
    диагонали
    (constr (/= (- x1 y1 ) (- x2 y2 )))) ;;; по
     второй диагонали
;;; создаем массив ферзей
(defun make-queens ( )
    (let ( (queen-array (make-array 8 )))
        (dotimes (I 8 queen-array )
            (setf (aref queen-array I )
                (make-instance 'queen :x (make-cvar-in
                '((1 ,8 )))
                                      :y (1+ I) )))))
```

Рис. 3. Программа решения задачи о ферзях на языке ConstraintLisp

Наконец, опишем задачу о ферзях, используя декларативный язык объектно-ориентированного моделирования UML/OCL. Для этого определим

необходимые объектные типы данных и инвариантные условия на них. На диаграмме классов языка UML (см. рисунок 4) представлен объектный тип *Queen* с соответствующими целочисленными атрибутами *x* и *y*, задающими положение фигуры на шахматной доске. В виде инвариантов контекста *Queen* на языке OCL описаны исходные условия задачи о ферзях. Инвариант *A* задает необходимое число фигур на шахматной доске. Инвариант *B* определяет допустимую область значений для координат фигур, ограниченных полем 8x8 клеток. Условие расположения ферзей на разных вертикалях и горизонталях выражается инвариантом *C*, а условие расположения ферзей на разных диагоналях — инвариантом *D*.




```
context Queen
-- (A)
inv: self.AllInstances()->size() = 8
-- (B)
inv: self.x >= 1 and self.x <= 8 and self.y >= 1 and self.y <= 8
-- (C)
inv: self.AllInstances()->forAll(q1, q2 | q1 <> q2 implies
(q1.x <> q2.x and q1.y <> q2.y))
-- (D)
inv: self.allInstances()->forAll(q1, q2 | q1 <> q2 implies
((q1.x - q1.y) <> (q2.x - q2.y) and (q1.x + q1.y) <> (q2.x +
q2.y)))
```

Рис. 4. Описание задачи о ферзях в виде диаграммы классов и спецификации ограничений на языке UML/OCL

Несмотря на декларативность используемых в примерах языков, спецификации на языке UML/OCL обладают большей выразительностью для моделирования данных и постановки соответствующих задач в ограничениях. Для описания задачи о ферзях потребовалось лишь структурировать данные и определить ограничения, которым они должны удовлетворять. Подобный способ исключает задание каких-либо вспомогательных методик или схем решения, присущих языкам логического и функционального программирования. В самом деле, вычислительная модель Prolog

подразумевает редукцию исходной постановки к рекурсивной или итерационной схеме поиска решений на основе определяемых пользователем правил логического вывода и целей. Заметим, что порядок определения правил и целей может существенно влиять на ход решения. В рассмотренном примере на ConstraintLisp помимо определения объектной модели данных потребовалось явно сконструировать неизвестные переменные и для них задать схему применения ограничений. Отметим громоздкость подобного функционального способа постановки задач в ограничениях по сравнению с декларациями на UML/OCL.

3. Сравнительный анализ подходов к CLP

3.1. Краткий обзор технологий программирования в ограничениях

Программирование в ограничениях как самостоятельное научное направление сложилось в конце 60-ых – начале 70-ых годов. Примечательно, что первыми приложениями были задачи обработки изображений и параметрического моделирования пространственно-двумерных сцен. С тех пор направление существенно эволюционировало, охватывая новые классы задач и выдвигая новые подходы к их решению. Тем не менее, четыре фундаментальных принципа, а именно: декларативное моделирование ограничений, локальное распространение результатов, эффективный глобальный поиск решений и специализация языковых и математических средств, по-видимому, не претерпели серьезных изменений. Начиная со Sketchpad [20], ALICE [21], ThingLab [22], эти принципы успешно эксплуатируются и в современных реализациях.

Принцип декларативности [23] нашел конструктивное воплощение во многих языках функционального и логического программирования, включая Lisp и Prolog, допускающих описание и решение некоторых задач в ограничениях. Со временем понимание, что функциональная рекурсия и логический вывод являются лишь отдельными элементами более общей стратегии разрешения ограничений для сложно структурированных неизвестных переменных, привело к появлению CostraintLisp, Prolog III [24] и CHIP [25]. Язык реляционных баз данных SQL [26], предусматривающий развитые декларативные конструкции для описания запросов и ограничений целостности, также несет в себе черты языка программирования в ограничениях. Связь с технологиями управления базами данных становится еще более очевидной при анализе относительно новой концепции Constraint Database (CDB) [27], которая, расширяя реляционную, дедуктивную и объектно-ориентированную модели, устанавливает непосредственную связь между типами данных и ограничениями, которые их предопределяют.

Принцип локального распространения (в англоязычной литературе соответствующий терминам constraint propagation или consistency inference)

[28] имеет долгую историю и многочисленные приложения, в частности, в области интеллектуализации графических интерфейсов пользователя. Спекулятивные попытки предвосхитить результаты поиска путем локальных и циклических уточнений частных решений оказались довольно плодотворными при решении больших систем уравнений и неравенств. Многочисленные “радужные” воплощения принципа привели к созданию целой палитры методов: Red (красный), Orange (оранжевый), Yellow (желтый), Green (зеленый), Blue (синий), DeltaBlue (инкрементально-синий), SkyBlue (небесно-синий), UltraViolet (ультрафиолет), Purple (фиолетовый), DeepPurple (пурпурный), Indigo (темно-синий) [29–31]. Некоторые из них обеспечивают как полный, так и инкрементальный поиск решений применительно к простым и иерархическим системам ограничений.

Принцип эффективного глобального поиска решений довольно естественен для прикладных задач в ограничениях, большая часть из которых является NP-полными. В отличие от методов распространения, направленных на пропозициональный вывод и локальное улучшение частных решений, методы глобального поиска обеспечивают систематический или стохастический поиск общих решений, удовлетворяющих всем заданным ограничениям. Заметим, что поиск с возвратом (backtracking) в сочетании с локальным распространением нашел применение практически во всех реализациях систем программирования в ограничениях, включая упомянутые выше диалекты языков Lisp и Prolog. В случаях редукции исходной задачи в ограничениях к типовой математической постановке появляется возможность применения известных и апробированных методов решения. Например, симплекс-метод используется в качестве стандартного математического средства в системе 2LP [32], а метод комбинаторного поиска на конечных доменах — в успешном коммерческом продукте ILOG Solver [33].

В некоторых случаях для усиления выразительности языковых средств, а также для упрощения идентификации математических постановок выделяются специальные типы ограничений и для них предусматриваются особые математические методики. В частности, в CHIP и ILOG Schedule поддерживаются некоторые типовые ограничения, возникающие в задачах планирования и составления расписаний. Специализация языковых и математических средств является общей чертой для всех известных реализаций систем программирования в ограничениях, каждая из которых охватывает лишь некоторые типы ограничений и порождаемые ими классы задач. Например, Prolog III обеспечивает решение систем линейных ограничений, условий на логических переменных и списках, CHIP — линейных ограничений, условий на логических переменных и конечных доменах, CHARME [34] — условий на конечных доменах, 2LP — только линейных ограничений, ILOG Solver — линейных ограничений и условий на конечных доменах и интервалах, HELIOS [35] — условий на интервалах, NEWTON [36] — нелинейных уравнений. Схожая ситуация сложилась и в области систем конкурентного программирования в ограничениях CCP, к

ярким представителям которых следует отнести AKL [37], Oz [38], CIAO [39], TAO [40]. Для краткости мы не рассматриваем CCP технологии в настоящей работе, адресуя заинтересованного читателя к обзорам [16, 41].

3.2. О задачах генерации тестовых данных

Обсуждаемый класс задач программирования в ограничениях довольно широк и охватывает самые разнообразные приложения. Безусловно, к ним могут быть отнесены и задачи генерации данных, возникающие при тестировании разного рода программного обеспечения, начиная с протоколов обмена данными и заканчивая трансляторами языков программирования. Несмотря на отмеченную общность, технологии генерации тестовых данных развиваются как самостоятельное направление программной инженерии с конца 70-ых – начала 80-ых годов. Как правило, в постановках задач генерации первостепенное внимание уделяется синтаксически адекватному представлению данных, порождаемому заданной формальной грамматикой некоторого целевого языка. Позитивные и негативные тесты строятся как наборы синтаксически корректных и некорректных программ, полнота которых обеспечивается покрытием продукционных правил грамматики. Сформированные таким образом наборы данных могут использоваться, например, для тестирования синтаксических анализаторов.

Понимание важности учета семантики языка при генерации тестов привело к целому ряду работ, основные идеи которых тесно перекликаются с рассмотренными выше принципами программирования в ограничениях. В частности, декларативное описание семантических правил, последовательная локализация решений, применение мутаций при формировании семантически согласованных и рассогласованных наборов данных, стохастический поиск решений могут быть отнесены к рассмотренным выше принципам и методам программирования в ограничениях. Эти идеи нашли применение в рамках концепции тестирования в ограничениях CBT (Constraint-Based Testing) [42].

Отметим роль аппарата атрибутных грамматик Кнута [43] при генерации тестов. Данный аппарат удобен как для описания синтаксиса целевого языка, так и для задания семантических правил. На его основе успешно решаются задачи комплексного тестирования программного обеспечения с учетом синтаксиса и статической семантики целевого языка. В классических работах программы, сгенерированные на основе заданной контекстно-свободной грамматики, подвергаются дополнительным испытаниям для отбора семантически корректных тестов. В ряде работ предпринимаются попытки внедрить элементы императивного подхода, в частности, явно задавать последовательность и процедуры пересчета атрибутов в дереве синтаксиса для удовлетворения семантических правил языка. Эти элементы применяются, в частности, в технологии тестирования программного обеспечения на основе формальных спецификаций и моделей, развиваемой в ИСП РАН [44–47].

3.3. Основные достоинства предлагаемого подхода

Главной особенностью предлагаемого подхода к объектно-ориентированному программированию в ограничениях является использование универсальных языков моделирования данных EXPRESS, ODL/OQL, UML/OCL, OWL. Язык UML/OCL рассматривается здесь лишь в контексте моделирования данных, хотя его функциональное назначение существенно шире. Язык описания онтологий OWL отнесен к группе объектно-ориентированных языков в силу известного соответствия между их конструкциями и возможностью непротиворечивой трансформации моделей, описанных на них, друг в друга.

Обсуждаемый системный подход к ООСР обладает рядом достоинств по сравнению с упомянутыми выше технологиями, а именно:

— *декларативность* описания задачи в ограничениях, исключающая необходимость предварительной проработки редукции исходной постановки к схемам поиска решения на основе функциональной рекурсии и логического вывода. Отметим, что данный этап неизбежно присутствует при использовании языков функционального и логического программирования и, как правило, вызывает наибольшие трудности;

— *выразительность* описания задачи в ограничениях, обусловленная непосредственной интерпретацией исходной постановки в терминах информационного моделирования. Вместо определения отдельных переменных и зависимостей между ними пользователем описывается единая согласованная модель данных и ограничений, которая служит входной информацией для системы программирования в ограничениях. Результатом работы системы являются наборы данных, структурированные в соответствии с заданной моделью и удовлетворяющие ее семантическим правилам;

— *универсальность* описания задачи в ограничениях, определяемая возможностью его использования при решении разнообразных задач программной инженерии. Спецификации, подготовленные на актуальных языках моделирования, могут применяться для автоматизированной разработки программ с использованием CASE инструментов. К числу подобных приложений могут быть отнесены задачи генерации схем баз данных, программных интерфейсов доступа к данным, протоколов обмена данными, графических пользовательских интерфейсов на соответствующих реализационных языках. На основе подобных спецификаций может решаться и задача генерации тестовых данных в рамках концепции CBT.

Естественно, общность предлагаемого подхода при описании задач в ограничениях на декларативных языках объектно-ориентированного

моделирования порождает проблемы алгоритмического характера, связанные с необходимостью разрешения больших систем разнородных алгебраических ограничений. Выработка единой вычислительной стратегии для удовлетворения подобных ограничений и решения задач в обсуждаемом классе CLP(O) представляется наиболее критичной.

4. Вычислительная стратегия программирования в ограничениях

В основе предлагаемой вычислительной стратегии разрешения ограничений лежит несколько конструктивных принципов и идей:

— использование единого метамодельного представления для работы с данными и моделями, специфицированными на языках EXPRESS, ODL/OQL, UML/OCL, OWL. Для взаимного преобразования моделей и определяемых ими данных могут быть применены известные трансформации;

— статический анализ спецификаций модели данных и идентификация математических постановок, к которым может быть сведена исходная задача в ограничениях. Для простых ограничений формируются альтернативные правила пересчета неизвестных переменных друг через друга, которые могут использоваться в методах распространения. Для сложных систем ограничений (и их подсистем) осуществляется анализ зависимостей по данным и определяется тип математической задачи, а также возможный метод решения. Например, идентификация множества ограничений как системы линейных алгебраических уравнений позволяет использовать известные методы матричной факторизации;

— последовательная редукция исходной задачи в ограничениях к задачам формирования множества неизвестных переменных (расчет необходимого числа объектов каждого типа), установления взаимосвязей (отношений ассоциации, агрегации и композиции между объектами) и определения числовых параметров (значений атрибутов объектов);

— применение методов линейного и квадратичного целочисленного программирования [48–50] для определения кардинальности множеств типизированных объектов. Выбор вида функционала позволяет эмулировать содержательные ситуации, когда, например, требуется сформировать минимальные, максимальные или репрезентативные наборы данных;

— применение методов логического вывода [51, 52] (прежде всего, метода резольвент и метода полисиллогистического вывода) для задания взаимосвязей объектов и определения неизвестных логического типа;

— применение методов интервальной арифметики, локализации и распространения для нахождения неизвестных вещественного типа при наличии простых зависимостей по данным;

— применение известных методов символьных преобразований [53, 54] и вычислительной математики для совместного решения систем (линейных, полиномиальных, нелинейных) алгебраических уравнений и неравенств;

— применение метода семантической реконсиляции [55–58] для инкрементального поиска общих решений с учетом всей системы наложенных разнородных ограничений.

Таким образом, в качестве основной стратегии предлагается использовать вышеописанную редукционную схему, состоящую в последовательном конструировании необходимого числа типизированных объектов, задании для них взаимосвязей и определении значений их атрибутов. Заметим, что большая часть семантических правил в объектно-ориентированных моделях специфицируется независимым друг от друга образом и не приводит к сложным зависимостям по данным. Это дает основания полагать о конструктивности применения подобной стратегии.

В более сложных ситуациях, когда основная стратегия не обеспечивает нахождение общего решения, предлагается использовать оригинальный метод семантической реконсиляции для коррекции уже найденных частных решений. Ключевые элементы этого метода были успешно апробированы в приложениях коллективной инженерии, к которым предъявляются близкие требования целостности и корректности результирующего представления данных. Применительно к обсуждаемому классу задач CLP(O) метод семантической реконсиляции предполагает следующие этапы:

— валидация семантических правил модели для исходных объектно-ориентированных данных и идентификация ограничений, не разрешенных в результате применения редукционного подхода;

— определение альтернативных способов коррекции для каждого нарушенного правила (определение наборов элементарных операций над множествами объектов, исправляющих нарушенные правила);

— совместный анализ выявленных способов коррекции с учетом уже выполненных правил и установление логических отношений между элементарными операциями;

— формирование и решение разреженной системы логических уравнений и определение допустимых комбинаций элементарных операций, удовлетворяющих всем ограничениям модели;

— коррекция исходных данных путем применения вычисленных наборов операций.

Ожидается, что применение редукционного подхода в сочетании с методом семантической реконсиляции обеспечит надежное решение задач в классе CLP(O).

Проиллюстрируем применение описанной вычислительной стратегии на примере задачи о правильном многограннике. Задача формулируется следующим образом: в трехмерном пространстве требуется построить правильный многогранник с заданным числом граней. Отметим, что в подобной постановке решение может и не существовать (известным фактом, например, является невозможность построения правильного пятигранника), поэтому желателен анализ существования решения.

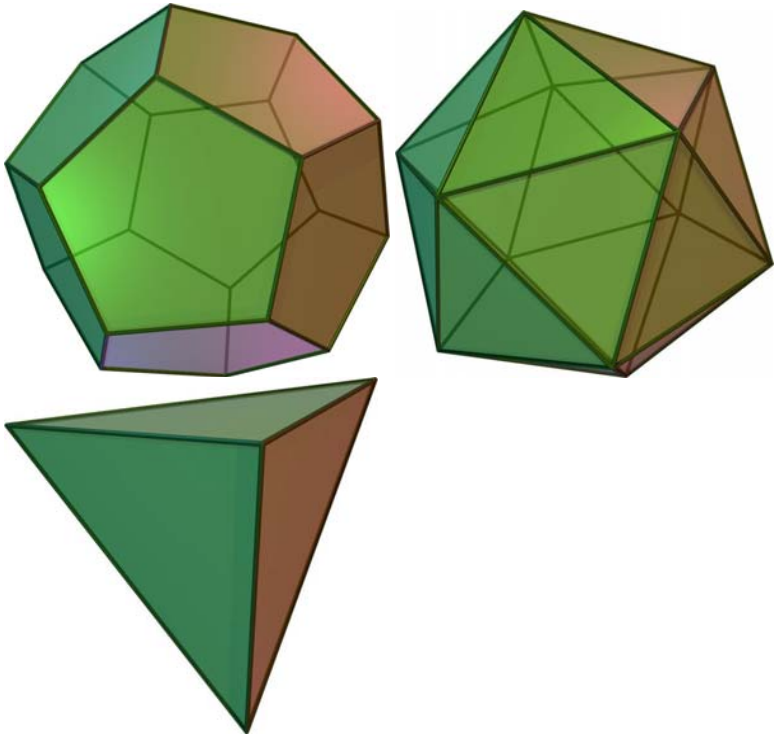


Рис. 5. Додекаэдр, икосаэдр, тетраэдр

Опишем задачу в ограничениях в виде спецификаций модели данных. Определим абстрактный класс *polyhedron* для представления произвольного правильного многогранника. Многогранник состоит из граней, ребер и вершин, моделируемых классами *face*, *edge* и *vertex* соответственно. Конкретизации класса *polyhedron* (см. Рис. 5), соответствующие геометрическим моделям *тетраэдра* (имеет 4 треугольных грани, 4 вершины, 6 рёбер, в каждой сходятся 3 ребра), *октаэдра* (имеет 8 треугольных граней, 8 ребер, 6 вершин, в каждой сходятся 4 ребра), *гексаэдра* (имеет 6 четырехугольных граней, 12 ребер, 8 вершин, в каждой сходятся 4 ребра), *додекаэдра* (имеет 12 пятиугольных граней, 30 рёбер и 20 вершин, в каждой сходятся 3 ребра) и *икосаэдра* (имеет 20 треугольных граней, 30 рёбер, 12 вершин, в каждой сходятся 5 ребер), моделируются классами *tetrahedron*, *octahedron*, *hexahedron*, *dodecahedron* и *icosahedron* соответственно. В рассматриваемой постановке абсолютный масштаб многогранника не существенен, поэтому для определенности положим длину ребер равной единице.

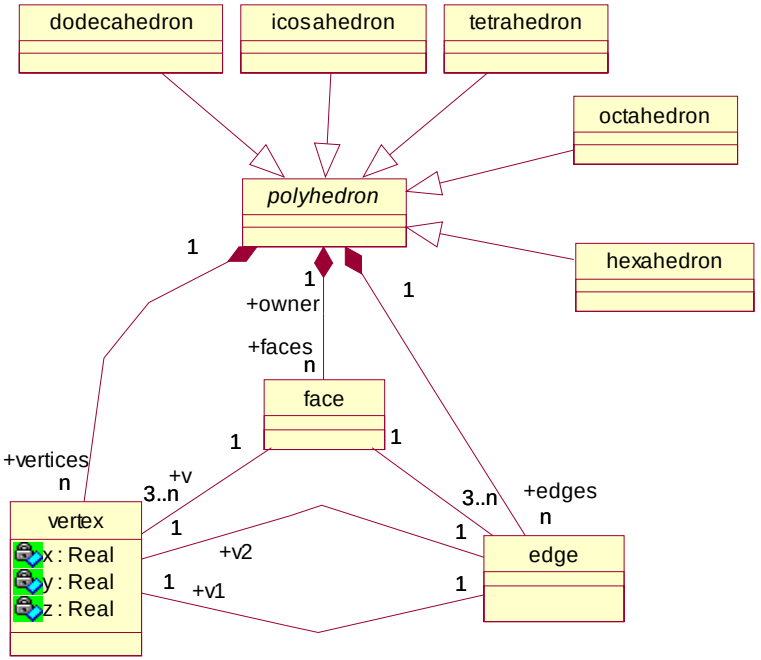


Рис. 6. UML модель для представления многогранников

На Рис. 6 приведена диаграмма классов UML, определяющая основные типы данных для представления многогранников. Условия исходной задачи

редуцированы в ограничения кратности для отношений ассоциаций и агрегаций между классами модели данных. Однако для исчерпывающей постановки исходной задачи требуется также задать:

— условие согласованности и связности топологических элементов многогранника, определяемое, в частности, формулой Эйлера для общего числа вершин, ребер и граней;

— условие выпуклости многогранника, например, в виде правил пространственной локализации его вершин в полупространствах, задаваемых плоскостями граней;

— условие правильности многогранника как равенства длин всех его ребер выбранному единичному значению.

Для описания этих ограничений воспользуемся языком OCL. Формула Эйлера для многогранника представляется в виде следующего инварианта, распространяемого на экземпляры класса *polyhedron*:

```
context polyhedron
inv: self.vertices->size() - self.edges->size() +
self.faces->size() = 2
```

Для задания ограничения выпуклости воспользуемся уравнением плоскости $Ax + By + Cz + D = 0$, проходящей через первые три вершины каждой грани и задающей полупространство для локализации остальных вершин многогранника. На языке OCL условие выпуклости многогранника может быть выражено соответствующим инвариантом класса face, в котором используются объявления вспомогательных локальных переменных:

```
context face
  let secVertices : Sequnce(Vertex) = collect(self.v) in
  let first : Vertex = secVertices(1) in
  let second : Vertex = secVertices(2) in
  let third : Vertex = secVertices(3) in

    let A : Real = first.y * (second.z - third.z ) +
                   second.y * (third.z - first.z ) +
                   third.y * (first.z - second.z ) in
    let B : Real = first.z * (second.x - third.x ) +
                   second.z * (third.x - first.x ) +
                   third.z * (first.x - second.x ) in

    let C : Real = first.x * (second.y - third.y ) +
                   second.x * (third.y - first.y ) +
```



```
                   third.x * (first.y - second.y ) in

    let D : Real = first.x * (third.y * second.z -
    second.y * third.z) +
                   second.x * (first.y * third.z -
                   third.y * first.z) +
                   third.x * (second.y * first.z -
                   first.y * second.z) in

self.owner.vertices->collect( v1 : vertex | not
self.vertices->includes( v1 ) )
    ->forAll( v2 : vertex | A * v2.x + B * v2.y + C *
    v2.z + D > 0 )
```

Наконец, условие равенства длин ребер прозрачным образом специфицируется средствами языка OCL как инвариант класса *edge*:

```
Context edge
inv: (self.v2.x – self.v1.x)*(self.v2.x – self.v1.x) +
     (self.v2.y – self.v1.y)*(self.v2.y – self.v1.y) +
     (self.v2.z – self.v1.z)*(self.v2.z – self.v1.z) = 1
```

В рамках описанной выше общей вычислительной стратегии программирования в ограничениях правильный многогранник может быть построен путем последовательной редукции исходной задачи к типовым математическим постановкам и их согласованному решению. Следуя данной стратегии, вначале проводится идентификации и разрешение ограничений, связанных с количеством топологических элементов многогранника. Как результат, определяется требуемое количество объектов соответствующих типов UML модели, а из их атрибутов формируется вектор неизвестных переменных. Затем на основе анализа спецификаций ограничений формируется система нелинейных алгебраических уравнений и неравенств относительно переменных, соответствующих координатам вершин многогранника. Решение системы может быть осуществлено численным образом, например, с использованием квази-ньютоновских методов, хорошо зарекомендовавших себя при решении широких классов нелинейных алгебраических задач. Рассмотренный пример иллюстрирует возможность применения единой стратегии для решения довольно сложных вычислительных задач, в постановке которых участвуют разнородные ограничения.

5. Заключение

Таким образом, предложен новый системный подход к реализации парадигмы объектно-ориентированного программирования в ограничениях на основе использования декларативных языков моделирования данных. Обсуждены и проиллюстрированы преимущества данного подхода по сравнению с известными технологиями функционального и логического программирования, а также отмечена его общность с методами генерации тестовых данных с учетом контекстных ограничений. Определена единая вычислительная стратегия для решения задач в выделенном классе CLP(O), сочетающая в себе ряд конструктивных идей и принципов. В дальнейшем предполагается детальная алгоритмическая проработка предложенного подхода и полнофункциональная реализация системы объектно-ориентированного программирования в ограничениях на основе декларативных языков моделирования данных.

Литература

[1] Dechter R. Constraint processing. Morgan Kaufmann, San Francisco, 2003.
[2] M. G. Buscemia, U. Montanarib. A survey of constraint-based programming paradigms. // Computer Science Review, Vol. 2, Issue 3, Dec. 2008, pp. 137–141.
[3] ISO 10303: 1994, Industrial automation systems and integration — Product data representation and exchange.
[4] OMG Model Driven Architecture: How systems will be built, http://www.omg.org/mda.
[5] W3C Semantic Web Activity, http://www.w3.org/2001/sw.
[6] ISO 10303-11: 2004, Industrial automation systems and integration — Product data representation and exchange — Part 11: Description methods: The EXPRESS language reference manual. Edition 2.
[7] Cattel R.G., Barry D.K., Berler M., et.al. The Object Data Management Standard: ODMG 3.0. Morgan Kaufmann, San Francisco, 2000.
[8] Unified Modeling Language, OMG Available Specification, Version 2.2, http://www.omg.org/spec/UML/2.2.
[9] Object Constraint Language Specification, Version 2.0, http://www.omg.org/technology/documents/formal/ocl.htm.
[10] OWL Web Ontology Language Guide, http://www.w3.org/TR/2004/REC-owl-guide-20040210.
[11] List of STEP Application Protocols, http://www.steptools.com/library/standard/step_2.html.
[12] Energetics, The Energy Standards Resource Centre, http://www.energistics.org/posc/Default.asp.
[13] BuildingSMART International Alliance for Interoperability, http://www.buildingsmart.com.
[14] Catalog of UML Profile Specifications, http://www.omg.org/technology/documents/profile_catalog.htm.
[15] E. Tsang. Foundations of constraint satisfaction. Academic Press Limited, London & San-Diego, 1993.

[16] V. Saraswat, P. Lincoln. Higher-Order Linear Concurrent Constraint Programming. Technical Report, Xerox PARC, Department of Computer Science, Stanford University, 1992.
[17] ISO/IEC 13211-1:1995. Information Technology — Programming Languages — Prolog — Part 1: General Core.
[18] Bing Liu, Yuen-Wah Ku. ConstraintLisp: An Object-Oriented Constraint Programming Language. // ACM SIGPLAN Notices, Vol. 27, No. 11, 1992, pp. 17–26.
[19] ANSI INCITS 226-1994 (R2004). Information Technology — Programming Language — Common Lisp.
[20] I.E. Sutherland. Sketchpad: A man-machine graphical communication system. Technical Report No. 296, Lincoln Laboratory, Massachusetts Institute of Technology, 1963.
[21] Alice, http://www.ps.uni-saarland.de/alice/index.html.
[22] A. Borning. The Programming Language Aspects of ThingLab, a Constraint-Oriented Simulation Laboratory. // ACM Transactions on Programming Languages and Systems, Vol. 3, No. 4, 1981, pp. 353–387.
[23] T. Mancini. Declarative constraint modelling and specification-level reasoning. PhD Thesis, Università degli Studi di Roma “La Sapienza”, Roma, Italy, March 2005.
[24] A. Colmerauer. An introduction to Prolog III. // Communications of the ACM, Vol. 33, No. 7, 1990, pp. 69–90.
[25] CHIP V5, http://www.cosytec.com.
[26] A. Beaulieu. Learning SQL, 2nd Edition. O’Reilly Media Inc., 2009.
[27] G. Kuper, L. Libkin. Constraint Databases. Springer-Verlag, Berlin, 2000.
[28] G. Tack. Constraint Propagation — Models, Techniques, Implementation. Doctoral Thesis, Saarland University, Germany, 2009.
[29] Freeman-Benson B.N., Maloney J., Borning A. An Incremental Constraint Solver. // Communication of the ACM, Vol. 33, No. 1, 1990, pp. 54–63.
[30] M. Sannella. SkyBlue: A Multi-Way Local Propagation Constraint Solver for User Interface Construction. // Proceedings of the 1994 ACM Symposium on User Interface Software and Technology, 1994, pp. 137–146.
[31] A. Borning, K. Marriott, P. Stuckey, Y. Xiao. Solving Linear Arithmetic Constraints for User Interface Applications: Algorithm Details. Technical Report 97-06-01, Department of Computer Science and Engineering University of Washington, 1997.
[32] K. McAloon, C. Tretkoff. 2LP: Linear Programming and Logic Programming. // Proceedings of PPCP’93, Newport, Rhode Island, 1993, pp. 178–189.
[33] IBM ILOG CP — Features and Benefits, http://www-01.ibm.com/software/integration/optimization/cp1/about/?S_CMP=rnav.
[34] Charme Reference Manual. AI Development Centre, Bull, France, 1990.
[35] L. Michel, P. Van Hentenryck. Helios: A modeling language for global optimization and its implementation in Newton. // Theoretical Computer Science, Vol. 173, No. 1, 1997, pp. 3–48.
[36] P. Van Hentenryck, L. Michel, F. Benhamou. Newton — constraint programming over nonlinear constraints. // Science of Computer Programming, Vol. 30, No. 1–2, 1998, pp. 83–118.
[37] P. Brand. Enhancing the AKL compiler using global analysis. Technical Report, Deliverable D.WP2.1.3.M2 in the ESPRIT project ParForce, 6707, 1994.
[38] The Mozart Programming System, http://www.mozart-oz.org.
[39] M. Hermenegildo. Some Challenges for Constraint Programming // Constraints, Vol. 2, No. 1, 1997, pp. 63–69.

[40] I. Shvetsov, T. Nesterenko, S. Starovit. Technology of Active Objects. AAAI Technical Report WS-97-05, Russian Research Institute of Artificial Intelligence & Institute of Informatics Systems, 1997.
[41] V. A. Saraswat. Concurrent Constraint Programming. The MIT Press, 1993.
[42] A. Gotlieb, T. Denmat, B. Botella. Constraint-based test data generation in the presence of stack-directed pointers. // Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering, Long Beach, CA, USA, 2005, pp. 313–316.
[43] М.В. Архипова. Автоматическая генерация тестов для семантических анализаторов трансляторов. Автореферат диссертации на соискание ученой степени кандидата физико-математических наук. Москва, ИСП РАН, 2006.
[44] А. Петренко, Е. Бритвина, С. Грошев, А. Монахов, О. Петренко. Тестирование на основе моделей. // Открытые системы, № 9, 2003, http://www.osp.ru/os/2003/09/183388.
[45] А.В. Демаков, С.В. Зеленов, С.А. Зеленова. Генерация тестовых данных сложной структуры с учетом контекстных ограничений. // Труды Института системного программирования: т. 9. / Под ред. В.П. Иванникова — М.: ИСП РАН, 2006, с. 83–96.
[46] A.K. Petrenko. Specification Based Testing: Towards Practice. // LNCS-2244, 2001, p.p. 287–300.
[47] С.В. Зеленов, С.А. Зеленова. Автоматическая генерация позитивных и негативных тестов для тестирования фазы синтаксического анализа. // Труды Института системного программирования: т. 8, ч. 1. / Под ред. В.П. Иванникова — М.: ИСП РАН, 2004, с. 41–58.
[48] Васильев Ф.П., Иваницкий А.Ю. Линейное программирование. — М.: Факториал Пресс, 2003.
[49] О.В. Хамисов. Численное решение специальных задач невыпуклого квадратичного программирования. // Дискретный анализ и исследование операций, сер. 1, т. 12, № 4, 2005, с. 81–91.
[50] У. И. Зангвилл. Нелинейное программирование. Единый Подход. — М.: Советское радио, 1973.
[51] И. Братко. Алгоритмы искусственного интеллекта на языке PROLOG. — М.: Вильямс, 2004.
[52] Л. Стерлинг, Э. Шапиро. Искусство программирования на языке Пролог. — М.: Мир, 1990.
[53] M. Chetty, K.P. Dabke. Symbolic computations: an overview and application to controllerdesign. // Proceedings of IEEE sponsored international conference on Information, Decision and Control, Adelaide, 8th-10th February, 1999, pp.451–456.
[54] Марчук Г.И. Методы вычислительной математики. — М.: Наука, 1977.
[55] Semenov V.A., Karaulov A.A. Semantic-Based Decomposition of Long-Lived Transactions in Advanced Collaborative Environments. // Proceedings of 6 European Conference on Product and Process Modeling, ECPPM 2006, Spain, Valencia, September 11–15, 2006, pp. 223–232.
[56] Семенов В.А., Ерошкин С.Г., Караулов А.А., Энкович И.В. Семантическая реконсиляция прикладных данных на основе моделей. // Труды Института системного программирования: т. 13, ч. 2. / Под ред. В.П. Иванникова — М.: ИСП РАН, 2007, с. 141–164.
[57] Semenov V.A. Collaborative Software Engineering Using Metamodel-Driven Approach. // Proceedings 16th IEEE International Workshops on Enabling Technologies:

Infrastructure for Collaborative Enterprises, WET ICE 2007, IEEE Computer Society Conference Publishing Services, 2007, pp. 178–179.
[58] Semenov V.A. Semantics-Based Reconciliation of Divergent Replicas in Advanced Concurrent Engineering Environments. // Complex Systems Concurrent Engineering: Collaboration, Technology Innovation and Sustainability, Springer-Verlag, London, 2007, pp. 557–564.