ВИЗУАЛЬНОЕ КОНСТРУИРОВАНИЕ И ОТЛАДКА ПРОЦЕДУРНОГО КОДА В СОСТАВЕ ОБЪЕКТОВ

В. И. Курганский
Иркутский государственный университет, Россия
e-mail: krg@irkl.ru

Technology and visual designing and debugging tools of the procedural programs in object structure are offered. The programming system architecture and tools for constructing program tests and program specification are discussed.

Введение

Несмотря на впечатляющие достижения современного визуального объектно-ориентированного программирования, разработка процедурных программ в составе объектов по-прежнему требует значительных усилий разработчиков. Высокий потенциал визуального представления самой программы и некоторых ее свойств при исполнении на примерах привлекает внимание разработчиков новых технологий программирования [1]. Ниже предлагается технология создания процедурных модулей в составе объектных программ, в рамках которых самый сложный текстовый программный код представляет собой оператор присваивания или условное выражение. Тенденция к такому представлению массовых программ ярко проявляется в практике применения табличных процессоров [2].

1. Системы примеров при визуальном конструировании процедур

Роль примеров в современных системах программирования недооценена. Примеры достаточно последовательно используют при тестировании и отладке программ [3, 4]. При конструировании алгоритмов и их программной реализации существующие системы программирования не обеспечивают программиста систематической поддержкой в отношении использования примеров.

Общеизвестно, что испытание программы на примерах необходимо для получения обоснованных выводов о ее качестве. При этом примеры являются связующим звеном между спецификацией программы и ее реализацией. При наличии подходящих инструментальных средств примеры помогают уяснить спецификации, найти в них недостатки и недоработки. Естественным критерием “полноты” системы примеров в этом случае является

покрытие фактор-графика отображения, реализуемого конструируемой программой в ее пространстве состояний и заданного ее спецификацией [5].

Под примером здесь понимаем последовательность состояний программы, генерируемых при ее исполнении. Каждое состояние представляет собой соответствующий ему комплект значений программных переменных. Для визуализации состояния к совокупности значений программных переменных добавим ссылку на соответствующую интерфейсную форму. Отметим, что интерфейсные формы в современных системах объектно-ориентированного программирования достаточно просто и естественно строятся средствами их визуального конструирования. Более того, интерфейсные формы, сконструированные для диалоговой обработки состояния программы в составе примера, зачастую оказываются полезными и при ее эксплуатации.

Таким образом, система примеров в рамках представляемой системы программирования существует в виде совокупности файлов, каждый из которых — это один из примеров. Отдельная запись такого файла соответствует одному из состояний в составе примера. Поля этой записи содержат значения программных переменных. Значение одного из служебных полей представляет ссылку на интерфейсную форму для диалоговой работы с данным состоянием. Разумеется, для работы с разными состояниями зачастую используются одни и те же интерфейсные формы. Все элементы одной системы примеров группируются в отдельной папке (каталоге). Таким образом, папка одновременно является и каталогом системы примеров.

Интерфейсные формы, предназначенные для диалогового доступа к отдельным состояниям конструируемой программы, могут быть дополнены индексами и фильтрами. Индексы и фильтры являются элементами спецификации конструируемой программы.

Под индексом понимается нумерация элементов управления, размещенных на форме, с помощью одной или нескольких координат. Для каждой интерфейсной формы автоматически создается одномерный базовый индекс. Он представляет собой нумерацию элементов управления в порядке их создания при визуальном конструировании интерфейсной формы. Над любым индексом, в том числе и базовым, может быть задано столько вторичных индексов, сколько необходимо для спецификации конструируемой программы. Определение вторичного индекса включает:

— выражение связи координаты базового индекса или координат уже сформированного вторичного индекса с координатами данного индекса;

— условные выражения, характеризующие принадлежность заданного значения каждой из координат данного индекса множеству ее допустимых значений;

— скалярные выражения для перехода от текущего значения координаты индекса к ее следующему или предыдущему значениям (в смысле перечислимого типа данных).

Общеизвестным примером вторичного индекса являются соотношения, которые позволяют рассматривать одномерный массив как прямоугольную матрицу. При решении задачи перебора диагоналей квадратной матрицы, параллельных ее главной диагонали (рис. 1 и 2), уместно над базовым одномерным индексом задать двумерный вторичный индекс, обеспечивающий доступ к элементам матрицы по номерам строки и столбца, и, далее, еще один двумерный вторичный индекс для доступа к элементам матрицы по номеру диагонали и номеру элемента в диагонали.

Под фильтром понимается условное выражение, которое является характеристической функцией некоторого множества значений, принимаемых координатами индекса. В определение фильтра включают также указание имени свойства элемента управления и некоторого значения этого свойства для визуального выделения элементов управления. Речь

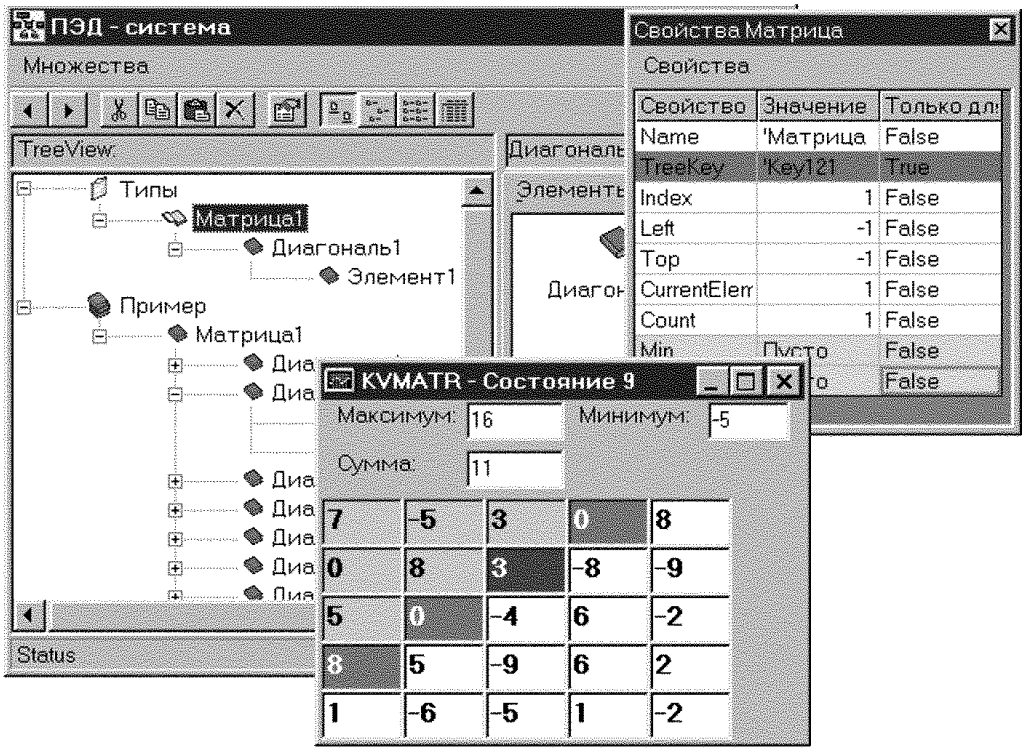

Рис. 1. Пример визуального выделения тестовых элементов управления с помощью фильтра.

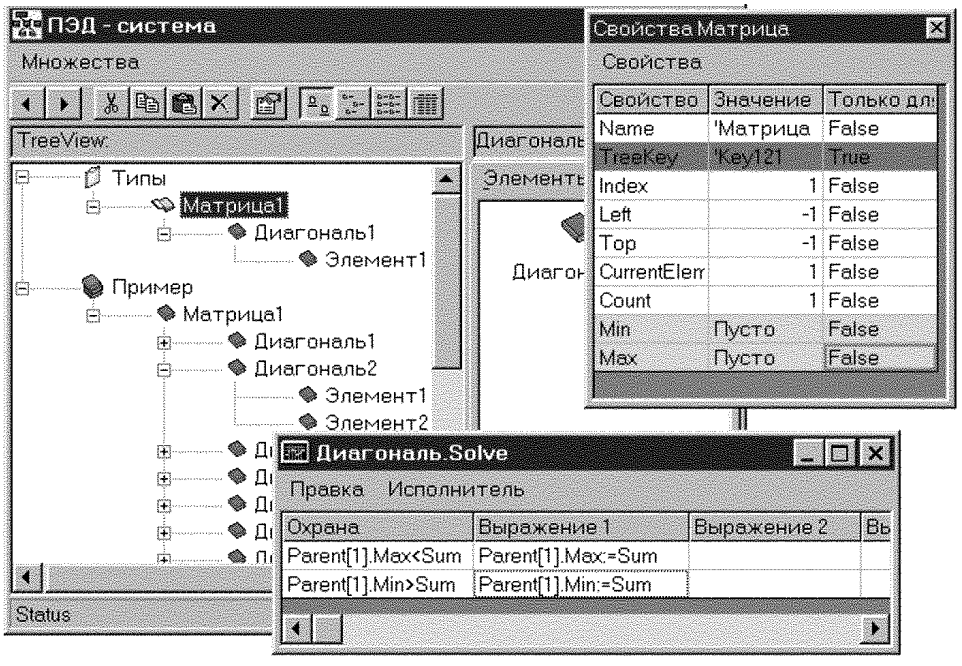

Рис. 2. Примеры форм для обзора и редактирования дерева ПЭД и инкапсулируемых в ПЭД скалярных выражений и данных.

идет об элементах управления, размещенных на интерфейсной форме и адресуемых теми значениями координат индекса, которые удовлетворяют фильтру. На рис. 1 продемонстрировано выделение одной из диагоналей обрабатываемой матрицы и всех диагоналей матрицы, предшествующих данной диагонали, а также текущего элемента на диагонали. Этот фильтр задан для вторичного индекса, предназначенного для доступа к элементам матрицы по номеру диагонали и номеру элемента на данной диагонали. На основе одного индекса может быть задано необходимое количество фильтров.

Для разработки требований к системам примеров, на которых испытываются конструируемые программы, создан язык темпоральных спецификаций [6]. Этот язык является сужением темпоральной логики [7] на случай непараллельных процедурных программ. В лингвистическом плане этот язык является развитием метода функциональных диаграмм, предназначенного для формирования требований к системам примеров, каждый из которых рассматривается как пара состояний — входное и выходное [8, 9]. Темпоральные спецификации позволяют рассматривать примеры как более длинные последовательности состояний. Темпоральная спецификация является характеристической функцией принадлежности заданной последовательности состояний программы множеству порождаемых при ее исполнении примеров.

Для языка темпоральных спецификаций разработаны две нотации — формульная и диаграммная. Диаграммная нотация подобно функциональным диаграммам удобна при конструировании спецификации и ее визуальном анализе. Формульная нотация используется для эквивалентных преобразований спецификации к одной из двух канонических форм. В первом случае примеры рассматриваются как пары состояний, а во втором — каждый пример представляет собой наиболее длинную последовательность состояний. Требования к системам примеров задаются с помощью конструирования диаграммной нотации темпоральной спецификации, дальнейшего ее преобразования к формульной нотации и приведения формульной нотации к одному из ее канонических видов [6].

2. Визуальная объектная модель процедурной программы

Индексы и фильтры представляют собой частичные спецификации конструируемой программы. Окончательные требования к программе формируются в виде ее визуальной объектной модели.

В качестве промежуточного языка между визуальной объектной моделью процедурной программы и входным языком базовой системы визуального объектного программирования выбран язык структурированных программ. Он был введен Э. Дейкстрой и развит Д. Грисом для решения задач верификации программ [10]. Причиной такого выбора послужил тезис о возможности сужения выразительных средств языка процедурного программирования без утери его выразительной мощности с последующим построением визуальной объектной модели конструируемой программы на базе суженного набора выразительных средств.

Основным элементом визуальной объектной модели структурированной программы является визуальный объект, названный программируемым элементом данных (ПЭД). В этот объект инкапсулируются описания программных переменных, названных прикладными свойствами, и простейшие текстовые коды в виде условных выражений и операторов присваивания. Программируемый элемент данных моделирует одну из следующих синтак-

сических конструкций языка структурированных программ:

$$S_1;\ S_2 \tag{1}$$

или

$$X_{0i} := E_{0i};$$

$$Do\blacklozenge B_{1i} \Rightarrow X_{1i} := E_{1i};\ [G_{1i}] \ldots \blacklozenge B_{ki} \Rightarrow X_{ki} := E_{ki};\ [G_{ki}]Od, \tag{2}$$

где $X_{0i} := E_{0i},\ \ldots,\ X_{ki} := E_{ki}$ — операторы кратного присваивания, $B_{1i}, \ldots, B_{ki}$ — условные выражения, а $S_1,\ S_2$ и G_{ki} — программы, которые в свою очередь также моделируются ПЭД. Напомним, что конструкция (1) представляет собой обычную последовательную композицию операторов языка структурированных программ, конструкция (2) — оператор цикла с начальной установкой, заданной оператором кратного присваивания. Выбор выразительных средств обусловлен характером их естественной теоретико-множественной интерпретации. Каждый из ПЭД задает переход от одного подмножества элементов пространства состояний программы к другому. Внутри же отдельного ПЭД, моделирующего конструкцию вида (2), переход от одного подмножества элементов пространства состояний к другому происходит при выполнении начального оператора присваивания или отдельной ветви оператора цикла. Теоретико-множественная интерпретация ПЭД вместе с механизмом визуального выделения элементов управления интерфейсных форм, предназначенных для доступа к состояниям программы в составе примеров, обеспечивают новую метафору визуализации потока данных, реализуемого программой. Эта метафора является развитием метафоры кинематографа (например, [1, 11]) с двумя отличиями. Первое отличие заключается в том, что отдельные кадры фильма — не мультимедийные продукты, а интерактивные механизмы доступа к текущему состоянию программы. Второе отличие позволяет иметь дело с виртуальной моделью последовательности кадров, когда основу каждого кадра составляет одна и та же интерфейсная форма, а смена состояний при выполнении программы демонстрируется визуальным выделением начальных, уже обработанных и текущих элементов управления (см. рис. 1).

В [12] доказано, что для любой структурированной программы существует функционально эквивалентная форма вида (2). Понятие функциональной эквивалентности программ уточнено на языке множеств примеров, порождаемых структурированными программами, и операций реляционной алгебры над множествами порождаемых программами примеров. Таким образом, созданная на основе этого результата и теоремы о структурировании [13] визуальная модель структурированной программы в виде дерева программируемых элементов данных представляет для разработчика достаточно мощный набор выразительных средств.

3. Технология визуального конструирования процедурного кода в составе объекта

Реализация технологии визуального конструирования процедурной части объектной программы на основе систем примеров и визуальной объектной модели конструируемого кода выполнена в виде набора ActiveX-компонентов, эксплуатируемых под управлением MS Windows (Windows 95 и выше, Windows NT), и настроек системы визуального

объектно-ориентированного программирования Visual Basic. ActiveX-компоненты доступны как Add-Ins-расширения системы программирования. Технологические средства визуального конструирования процедурных программ в составе объектов включают:

— конструктор темпоральных спецификаций;

— конструктор интерфейсных форм для диалогового доступа к отдельным состояниям примера, который включает средства формирования и редактирования индексов и фильтров;

— конструктор примеров;

— конструктор дерева программируемых элементов данных и его отдельных элементов;

— отладчик;

— генератор текста конструируемой программы на входном языке базовой системы программирования.

Рассмотрим их подробнее. Конструктор темпоральных спецификаций позволяет формировать в диалоговом режиме диаграммную нотацию спецификации и преобразовывать ее в каталог системы примеров.

Конструирование интерфейсных форм обеспечивается стандартными средствами визуального конструирования SDI-форм в рамках базовой системы программирования на основе шаблона интерфейсной формы и специального набора тестовых элементов управления. Тестовые элементы управления получены из стандартных элементов управления базовой системы визуального объектно-ориентированного программирования, предназначенных для диалогового доступа к скалярным значениям программных переменных (текстовым, числовым, булевым и т.п.). По сравнению со стандартными элементами управления тестовые элементы управления дополнены специальными свойствами, которые обеспечивают преобразование значений программных переменных в составе ПЭД в значения основных свойств тестовых элементов управления, а также визуальное выделение этих элементов управления при интерпретации фильтров.

Конструктор примеров обеспечивает редактирование каталога системы примеров, каталога отдельного примера, а также загрузку интерфейсной формы для обзора и редактирования отдельного состояния в составе примера.

Конструктор ПЭД обеспечивает навигацию по дереву ПЭД, редактирование дерева ПЭД, редактирование списка прикладных свойств (программных переменных) для текущего ПЭД и инкапсулируемого в ПЭД простейшего программного кода в текстовом виде (см. рис. 2).

Отладчик реализован в виде модуля интерпретации дерева ПЭД. С помощью отладчика обеспечивается установка прерываний выделением узлов дерева ПЭД, отдельных состояний в составе примера, а также выбором условных выражений в составе определений индексов, фильтров и условных выражений, инкапсулированных в ПЭД.

Процедурная программа, заданная деревом ПЭД, может быть преобразована в код на входном языке базовой системы визуального объектно-ориентированного программирования. Атрибуты генерируемого процедурного кода задаются с помощью специальной диалоговой формы.

Заключение

Прикладным результатом работы является реализация системы программируемых элементов данных в виде Add-Ins-расширения Visual Basic.

В дальнейшем предполагается развитие системы за счет реализации синтаксически управляемого преобразования визуального представления конструируемых программ во входной язык базовой системы визуального объектно-ориентированного программирования.

Автор выражает признательность семинару Института динамики систем и теории управления СО РАН за неоднократные обсуждения работы и полезные замечания, а также сотруднику кафедры теории алгоритмов и программирования Иркутского государственного университета П. В. Кобелеву за значительный вклад в данную работу при ее программной реализации.

Список литературы

[1] Авербух В. Л. Метафоры визуализации // Программирование. 2001. №5. С. 3–17.

[2] Федоров А., Елманова Н. Базы данных для всех. М.: Компьютер Пресс, 2001. 256 с.

[3] Кауфман А. В., Черноножкин С. К. Критерии тестирования и система оценки полноты набора тестов // Программирование. 1998. №6. С.44–59.

[4] Черноножкин С. К. Задача автоматического построения тестов и статистический анализ // Программирование. 2001. №2. С. 47–59.

[5] Корольков Ю. Д. Математические модели качества программных средств. Иркутск: Изд-во Иркут. ун-та, 1996. 160 с.

[6] Курганский В. И. Конструирование информационно-расчетных систем на основе сложных объектов с контекстными отношениями. Иркутск: Изд-во Иркут. ун-та, 2001. 206 с.

[7] Смелянский Р. Л. Применение темпоральной логики для спецификации поведения программных систем // Программирование. 1993. №1. С. 3–28.

[8] Elmendorf W. R. Functional analysis using cause-effects grafs // Proc. of SHARE XLII. N. Y.: SHARE. 1974. P. 567–577.

[9] Майерс Г. Искусство тестирования программ. М.: Финансы и статистика, 1982. 178 с.

[10] Грис Д. Наука программирования: Пер. с англ. / Под ред. А.П. Ершова. М.: Мир, 1984. 416 с.

[11] Важенин А. П., Миренков Н. Н. Элементы системы визуального программирования VIM // Программирование. 2001. №4. С. 68–80.

[12] Курганский В. И. Метаинтерактивное программирование: концепции, основания и реализация // Оптимизация, управление, интеллект. 1995. №1. С. 198–209.

[13] Bohm C., Jacopini G. Flow diagrams, turing machines and languages with only two formation rules // Communications of the Association for Comp. Mach. 1965. Vol. 9. P. 366–371.

Поступила в редакцию 16 апреля 2002 г.,
в переработанном виде — 1 августа 2002 г.