Конструирование программного обеспечения по компьютерной оптике с использованием мультипарадигменного подхода

Д.Е. Яблоков[1], В.С. Павельев[2,3], А.Н. Агафонов[2,3], А.В. Ерёмин[3]

[1]ООО «ИнтеллектСофт», Мичурина 52, Самара, Россия, 443086
[2]Институт систем обработки изображений РАН - филиал ФНИЦ «Кристаллография и фотоника» РАН, Молодогвардейская 151, Самара, Россия, 443001
[3]Самарский национальный исследовательский университет им. академика С.П. Королева, Московское шоссе 34А, Самара, Россия, 443086

Аннотация. Общей целью для всех этапов конструирования программного обеспечения является многократное использование элементов решения, что весьма точно соответствует принципам, рассматриваемого в статье мультипарадигменного подхода. Акцент делается на формировании системы взглядов в процессе конструирования программ по компьютерной оптике, основанной на использовании наиболее значимых особенностей процедурной, объектно-ориентированной и обобщённой парадигм программирования. Для каждой из парадигм, на примере расчёта фазовой функции дифракционного оптического элемента – аксикона, на заданном множестве точек, рассматриваются их достоинства и недостатки. Фрагменты кода созданы с использованием языка C#, поддерживающего несколько парадигм программирования, что позволяет выражать алгоритмическую, поведенческую и синтаксическую общность. Стратегии конструирования, подобные представленной в статье, продолжают оставаться актуальными, несмотря на обилие гибких и разнообразных средств, присутствующих в современных языках программирования. Они позволяют в полной мере использовать все преимущества нескольких парадигм программирования и формируют концептуальный базис мультипарадигменного подхода при создании программного обеспечения по компьютерной оптике.

1. Введение

Применение современных информационных технологий в научной деятельности предполагает, что программные системы становятся всё более простыми для использования специалистами, но, в то же время, более сложными по внутренней архитектуре. Это оказало серьёзное влияние на подход к научному программированию [1], которое сейчас нельзя представить без качественно построенных абстракций, инструментальных средств и методологий, определяющих стратегию конструирования [2] прикладного программного обеспечения. При этом задачи, решаемые современной наукой и современными технологиями, также значительно усложнились. Исключением не является и область компьютерной оптики, где сложность проявляется в виде необходимости работы с различными типами оптических элементов, поддержки большого количества структур данных, использования множества форматов и преобразований. Сложность может быть присуща как самой научной проблеме, так и всей

предметной области, но она также может быть и следствием неудачно выбранного способа или стиля изложения заложенных в программе идей. Неправильно выбранный инструментарий или стратегия конструирования приводят к проблемам [2], которые, к сожалению, могут быть не выявлены на этапе проектирования, а проявят себя лишь после начала работы. Такие программы, как правило, имеют очень непродолжительный жизненный цикл и их проще заменить новыми, чем повторно использовать, пытаться адаптировать или совершенствовать. Одним из критериев качества программного продукта [3] является то, насколько внимательно разработчики относятся к процессу формализации отношений и взаимодействий между его компонентами. Таким механизмам следует уделять особое внимание, чтобы программная архитектура получилась более компактной, простой и понятной, а код более структурированным и адаптируемым к вновь специфицируемым требованиям.

В виду того, что процесс разработки программного обеспечения может быть организован по-разному, а способов трактовки и понимания каждого из вариантов также может быть несколько, особую актуальность приобретает формализация логики выбора того или иного направления.

Целью данной статьи является исследование возможности применения мультипарадигменного подхода [4] при проектировании и реализации приложений для компьютерной оптики. На примере расчёта фазовой функции дифракционного оптического элемента – аксикона [5] основной акцент будет сделан на симбиозе наиболее значимых характеристик процедурной, объектно-ориентированной и обобщённой парадигм. В рамках небольшого примера будут проанализированы достоинства и недостатки трёх основных парадигм, исследованы возможности их применения в процессе разработки программ для компьютерной оптики и произведена оценка работы примера для конкретного оптического элемента в соответствии с выбранной парадигмой программирования.

Особенности различных парадигм будут рассматриваться по ходу изложения, а основными этапами при реализации примера будут:

1. Определение некоторой области расчёта (Рисунок 1) с разбиением на заданное количество точек (Рисунок 2).

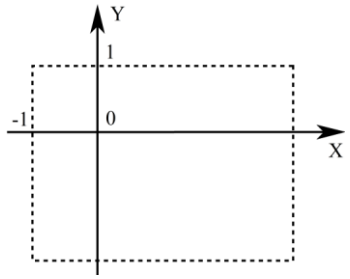

Рисунок 1. Определение области для расчёта.

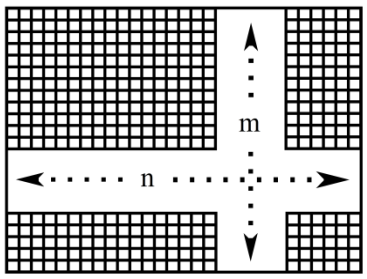

Рисунок 2. Разбиение области на точки.

2. Создание элемента программы для расчёта фазовой функции дифракционного аксикона:

$$\varphi(x,y) = -\frac{\alpha\sqrt{x^2\cos^2(\gamma)+y^2}}{2\cos(\gamma)}, \quad (1)$$

где α – параметр, определяющий угол, под которым лучи выходят из аксикона, а γ – угол падения пучка света на элемент.

3. Применение процедуры расчёта к множеству полученных на этапе 1 точек, с представлением результатов в различных форматах.

Вполне естественно, что простой пример не позволит в рамках данной работы охватить все особенности проектирования сложных программных систем. Но он обеспечит возможность сравнения различных парадигм программирования с рассмотрением типичных причин и приемов, мотивирующих для их совместного применения.

2. Процедурная парадигма

Современные языки высокого уровня, в большинстве своём, поддерживают парадигму процедурного программирования [6]. Являясь частным случаем императивной парадигмы [7], процедурный подход предполагает наличие набора процедур, каждая из которых есть последовательность элементарных действий или вызовов таких же процедур.

Метод `Phase` (рисунок 3) производит разбиение поступающей на вход области (аргумент `rt`) на задаваемое по вертикали и горизонтали количество шагов (строки 3, 4) в соответствие параметрам `m` и `n`. Затем для каждой из полученных точек с координатами `x` и `y` (строки 10, 11), вычисляется фазовая функция с использованием, также входящих в сигнатуру метода, параметров `alpha` и `gamma`, с последующей записью полученного значения в результирующий одномерный массив (строки 12 – 13).

```
1  double[] Phase(double alpha, double gamma, Rect rt, int m, int n)
2  {
3     var h = rt.Height / m;
4     var l = rt.Width / n;
5     var res = new double[m * n + m + n + 1];
6     for(int i = 0, f = 0; i <= m; ++i, f += n)
7     {
8        for(var j = 0; j <= n; ++j)
9        {
10          var x = rt.Left + j * l;
11          var y = rt.Top - i * h;
12          res[i + f + j] = Math.Sqrt(Math.Pow(x, 2) * Math.Pow(Math.Cos(gamma), 2) +
13          Math.Pow(y, 2)) * alpha / (2 * Math.Cos(gamma));
14       }
15    }
16 }
```

Рисунок 3. Процедурный подход к расчёту фазовой функции на множестве точек.

Функциональность метода полностью охватывает предварительные этапы (рисунки 1, 2), а также производит вычисления (1) и представляет результат. Такая стратегия реализации обладает следующими преимуществами:

1. Семантика метода проста и не содержит сложных абстракций. Она понятна и прозрачна для построения ассоциаций с контекстом решаемой задачи.
2. Непосредственная доступность используемых типов данных из любого места программы, а значит простота при сопровождении кода.
3. Базисы операций над типами, поступающих на вход, внутренних или возвращаемых объектов интерпретируются как вычислительные базисы встроенных типов данных.
4. Возможность организации подобных методов в других единицах трансляции на той же терминологической основе и с тем же уровнем сложности.

Охарактеризовать рассмотренный пример можно как плохо спроектированный и слабо структурированный. Для такого небольшого фрагмента кода сложность модификации, а также отсутствие достаточной гибкости и развитости семантики при работе с данными [8] не является совсем плохим и недопустимым решением. Но, если в дальнейшем планируется его использование как составной части какого-либо модуля или библиотеки, предназначенной для более сложных и масштабных вычислений, проблема внесения изменений в процессе сопровождения по своей значимости может выйти на лидирующие позиции. Изменениям могут быть подвержены и способ обхода области, передаваемой на вход алгоритма, и стратегия получения координат текущей точки, и индексация результирующего массива. Изменений могут потребовать, как тип и количество входных параметров, так и тип возвращаемого значения, а также сама последовательность операций для расчёта фазовой функции [9].

К сожалению, данный подход не предполагает каких-либо либо серьёзных изменений, из-за того, что проектное решение построено по принципу «как есть» [3].

3. Объектно-ориентированная парадигма

Парадигма объектно-ориентированного программирования, являясь развитием процедурного и модульного стилей, использует иерархии полиморфных типов данных [10], связанных отношением наследования. Это позволяет, с помощью каркаса понятий объектно-ориентированного подхода, выражать поведенческую общность для подобных объектов через наследование, а изменчивость – через полиморфизм [4].

Одним из наиболее распространенных типовых решений, применяемых для реализации настраиваемого поведения алгоритмов в объектно-ориентированном языке, является «Шаблонный метод» [11]. Оно задаёт строгую основу алгоритма, позволяя подклассам переопределять некоторые шаги, не затрагивая структуру в целом (рисунок 4).

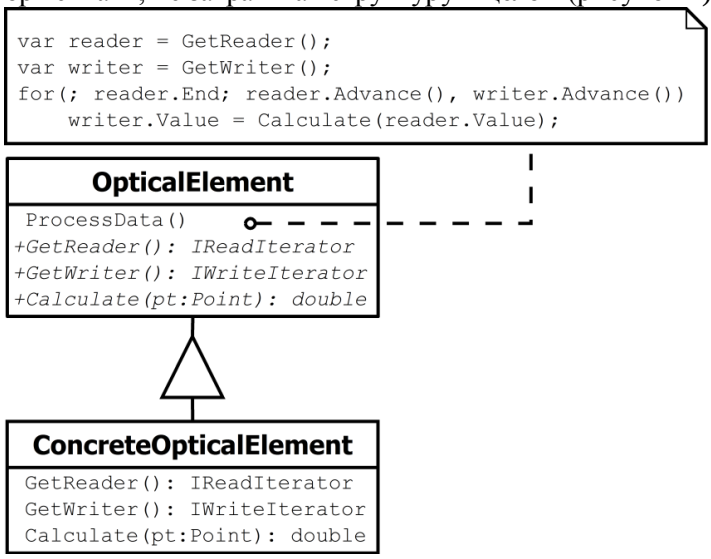

Рисунок 4. Диаграмма типового решения «Шаблонный метод».

Как видно из диаграммы, в основном, изменениям могут быть подвержены: средство чтения данных (класс-потомок `IReadIterator`), способ записи результата (реализация `IWriteIterator`) и сам расчёт фазовой функции для отдельной точки (метод `Calculate`). При этом целостность самого алгоритма, его итерационный механизм, остаётся неизменным. На каждом шаге цикла (метод `ProcessData`), перед перемещением на следующую позицию (рисунки 5, 6 – метод `Advance`), происходит трансформация значения (рисунок 4 – метод `Calculate`), полученного из объекта, поддерживающего свойство чтения [12] (рисунок 6), с последующей передачей преобразованного результата объекту, способному произвести запись [12] (рисунок 7).

```
1  interface IIterator
2  {
3     void Advance();
4  }
```

Рисунок 5. Интерфейс для реализации итератора.

```
1  interface IReadIterator : IIterator
2  {
3     Point Value { get; }
4     bool End { get; }
5  }
```

Рисунок 6. Интерфейс для реализации итератора чтения.

```
1  interface IWriteIterator : IIterator
2  {
3     double Value { set; }
4  }
```

Рисунок 7. Интерфейс для реализации итератора записи.

Фактически, с переходом на объектно-ориентированную парадигму, выражение «вычислить фазовую функцию на множестве точек» приобрело некоторую зафиксированную стадийность. Её суть – последовательное применение элементарных операций, поведение которых, при

неизменности тела алгоритма, зависит от типов объектов, наследующих определённым интерфейсам.

К преимуществам данного подхода можно отнести:

1. Эффективное использование выразительных возможностей языка программирования, поддерживающего объектно-ориентированную парадигму.
2. Локализация данных и операций в виде отдельных абстракций, что повышает наглядность кода и удобство сопровождения.
3. Стимуляция к повторному использованию не только отдельного компонента, но и всего проектного решения. Объектно-ориентированная архитектура обладает большей компактностью, чем её процедурный эквивалент (рисунки 3, 4).
4. Создание систем с устойчивыми промежуточными формами, что упрощает их изменение и позволяет вносить улучшения со временем, не прибегая к полной переделке при первой серьезной модификации.
5. Уменьшение степени риска, связанного с созданием сложных приложений. Межкомпонентные взаимодействия, подразумевающие разумное разделение ответственности и обязанностей, подталкивают к правильной стратегии принятия проектных решений.
6. Расширение каркаса понятий, являющихся развивающими для базовых абстракций. Реализация подклассов, специфичных для конкретного поведения и обращающихся к различным структурам данных, может происходить на протяжении всего жизненного цикла программы.
7. Параметризация поведения алгоритма на этапе выполнения, когда один компонент может быть заменён другим. Это даёт возможность обобщать алгоритмы до такой степени, что они смогут работать более чем с одним типом объектов.

Но, к сожалению, все эти преимущества сильно нивелируются двумя серьёзными недостатками, касающимися:

1. Жесткой привязки структуры алгоритма (рисунок 4 – метод `ProcessData`) к классу `OpticalElement`. Выполнение данной операции может иметь более широкие смысловые границы и потребоваться не только для реализации абстракции оптического элемента.
2. Способность работы только с ограниченным диапазоном типов (`double`, `Point`, `Rect`). Это не даёт простора для применения алгоритма к типам данных, имеющим различия по семантике, но схожим с точки зрения синтаксиса.

Без более развитых средств обобщения, в рамках только объектно-ориентированного подхода, применение такой реализации алгоритма для множества других объектов, удовлетворяющих тем же синтаксическим и семантическим требованиям, но оперирующих другими типами данных становится невозможным.

4. Обобщённая парадигма

Обобщённым программированием [13] называется технология, где основной акцент делается на проектировании таких алгоритмов и структур данных, которые бы работали в наиболее общей ситуации с наибольшим количеством встроенных или пользовательских типов. С учётом того, что в настоящий момент всё большую популярность для научных исследований и вычислительных экспериментов продолжает набирать такой язык как C# [14, 15], нельзя не упомянуть используемые в нём механизмы обобщения (generics). Данная технология внесла значительный вклад в выразительность, безопасность типов и производительность языка, но она также не лишена и некоторых отрицательных особенностей [16]. Среди других недостатков самым весомым продолжает оставаться недоступность информации о свойствах параметра типа во время компиляции.

Следуя обобщённой парадигме, основная идея которой состоит в использовании наиболее общих формулировок, возможно получение минимального набора элементарных операций, которых будет вполне достаточно для реализации требуемых в (1) математических действий (рисунок 8).

```
interface INumber<T>
{
   T Add(T arg1, T arg2);
   T Mul(T arg1, T arg2);
   T Div(T arg1, T arg2);
   T Mod(T arg1, T arg2);
   T Ups(T arg, int n);
   T Dns(T arg, int n);
   bool Et(T arg1, T arg2);
   bool Lt(T arg1, T arg2);
   T Twice(T arg);
   T Half(T arg);
   T Neg(T arg);
   T Abs(T arg);
   T Zero { get; }
   T One { get; }
   T Eps { get; }
};
```

Рисунок 8. Обобщённая концепция универсального числового типа.

Смысловая составляющая такого подхода может быть выражена через понятие концепции [17, 18], как средства описания семейств взаимосвязанных типов, объединённых синтаксической и семантической общностью. С помощью технологии методов расширения [19] возможна реализация в терминах обобщённой концепции `INumber` дополнительных, более сложных, математических действий, таких как: нахождения квадратного корня (рисунок 9), возведения в степень, вычисления тригонометрических функций.

```
T Sqrt(this INumber<T> concept, T arg)
{
   T temp;
   T result = concept.Half(arg);
   do
   {
      temp = result;
      result = concept.Half(concept.Add(temp, concept.Div(arg, temp)));
   } while (!concept.Et(concept.Add(temp, concept.Neg(result)), concept.Eps));

   return result;
}
```

Рисунок 9. Вычисление квадратного корня для обобщённой концепции числового типа.

Адаптированная к интерфейсу функционального объекта (рисунок 10) обобщённая концепция (рисунок 8) может использоваться для вычисления (1).

```
class Axicon<T> : IUnaryOperation<Point<T>, T>
{
   private INumber<T> _cpt;
   private T _a, _g;
   public T Execute(Point<T> pt)
   {
      /* Вычисление фазовой функции */
   }
}
```

Рисунок 10. Функциональный объект для расчёта фазовой функции.

Передавая в обобщённый алгоритм функциональный объект (рисунок 10) в качестве аргумента, возможно, его применение для различных типов данных, при сохранении сигнатуры представляющих их концепций.

Параметризация поведения алгоритма может производиться на различных уровнях. Это и рассмотренный выше обобщённый функциональный объект, агрегирующий экземпляр обобщённой концепции, экстернализирующей основные операции передаваемого параметра

типа. Также это может быть и полностью специализированная операция (рисунок 11), выполняющая набор действий, подобный её обобщённому аналогу, но уже с конкретным типом данных, устанавливающая набор абстрактных требований к типам передаваемых в алгоритм аргументов.

5. Мультипарадигменный подход

Большинство программных проектов, включая и проекты для компьютерной оптики, требуют поддержки нескольких парадигм программирования. Даже в самых «чистых» с точки зрения стилистики и следования основополагающим принципам объектно-ориентированного подхода проектах, приходится возвращаться к алгоритмической или синтаксической общности, которые выявляются на разных этапах проектирования и реализации. Пристальное внимание к общим свойствам и отличиям компонентов, способствует разбиению программной системы на независимые составляющие (рисунок 11).

```
1  public static OE Product2<OE, T1, T2, R>(this IEnumerable<T1> source1,
2     IEnumerable<T2> source2, IOutputEnumerable<OE, R> target, Func<T1, T2, R> op)
3  where OE : IOutputEnumerable<OE, R>
4  {
5     using (var input2 = source2.GetEnumerator())
6     using (var output = target.GetEnumerator())
7     {
8        foreach(var value1 in source1)
9        {
10           while (input2.MoveNext() && output.MoveNext())
11              output.Current = op(value1, input2.Current);
12
13           input2.Reset();
14        }
15        return output.GetEnumerable();
16    }
17 }
```

Рисунок 11. Возможная реализация алгоритма основного цикла вычисления фазовой функции для множества точек.

В приведённом фрагменте кода (рисунок 11) показано, что обобщённый алгоритм `Product2` может быть применён к множеству типов данных. Он не привязан к какой-либо абстракции, отвечающей за узкоспециализированный, ограниченный определенным фрагментом предметной области, функционал. Такой алгоритм может использоваться во многих ситуациях (рисунок 12), когда возникает необходимость:

1. Поэлементно считать данные из входных полуоткрытых интервалов.
2. Применить к значениям каждого элемента некоторый функциональный объект, передаваемый в алгоритм в качестве параметра.
3. Произвести запись полученного результата в выходной диапазон.

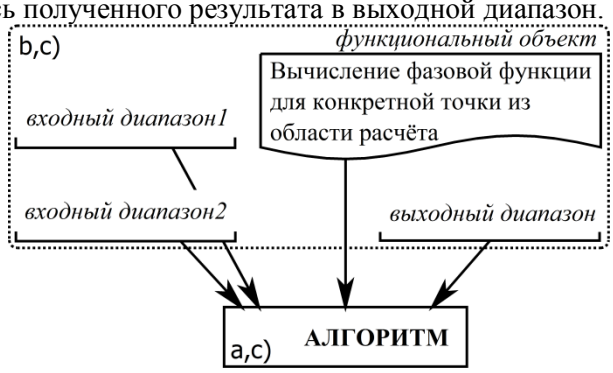

Рисунок 12. Диаграмма взаимодействий компонентов при мультипарадигменном подходе: a) – процедурная парадигма; b) – объектно-ориентированная парадигма; c) – обобщённая парадигма.

Ограничения, накладываемые алгоритмом `Product2` на свои аргументы, формируют небольшой, но в тоже время достаточный для развитой семантики обработки данных набор требований. Объекты, представляющие входные диапазоны должны соответствовать семантике итератора чтения [18, 20] или энумератора [16, 21] (рисунок 11) с возможностью чтения данных из текущей позиции (строка 11). Для объектов, моделирующих выходной диапазон, должна обеспечиваться поддержка семантики итератора записи [18] или модифицирующего энумератора (рисунок 11 – строка 11) с возможностью доступа к значению элемента в режиме записи. Являясь именованным набором ограничений для одного или нескольких параметров типов, обобщённые концепции, объединяющие средства обхода и доступа к элементам, совместно с бинарной операцией, позволяют задавать необходимую стратегию поведения.

6. Результаты

Результаты тестов (Табл. 1-2, Рис. 13-14) для каждого из подходов содержат идентичные, с точки зрения вычисленных значений, данные, представленные в различных форматах. В исследовании показана прямая зависимость жизненного цикла программного продукта от главного критерия оценки эффективности разработки – качества проектного решения [2, 3].

Для всех тестовых случаев предполагалось:

1. Область расчёта имеет координаты для верхнего левого угла (-1; 1) и размеры 2×2.
2. Число шагов, на которые будет разбиваться область по вертикали и горизонтали – 10000.

Участвующие в расчёте параметр α и угол γ равны произвольно выбранным значениям 57.562348 и 25.347845 соответственно.

Таблица 1. Результаты вычислений для процедурного подхода с представлением результатов в виде одномерного массива размерностью [100020001].

Элемент	Значение
[0]	41.185600842600067
[1]	41.181578497059711
[2]	41.1775565632722
[3]	41.173535041358164
[4]	41.169513931438331
[5]	41.165493233633427
. . .	

Таблица 2. Результаты вычислений для объектно-ориентированного подхода с представлением результатов в виде двумерного массива размерностью [10001, 10001].

Элемент	Значение
[0, 0]	41.185600842600067
[0, 1]	41.181578497059711
[0, 2]	41.1775565632722
[0, 3]	41.173535041358164
[0, 4]	41.169513931438331
[0, 5]	41.165493233633427
. . .	

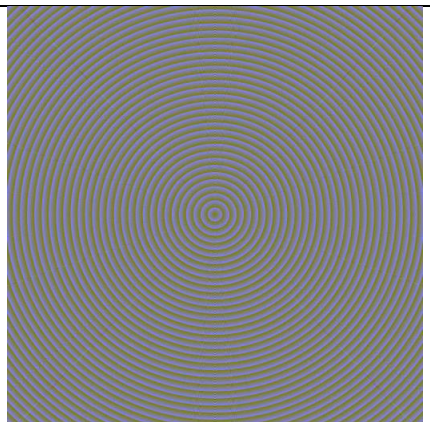

Рисунок 13. Результаты вычислений для обобщённого подхода в виде цветного изображения. Фаза аксикона (1) приведена к диапазону $[0; 2\pi]$.

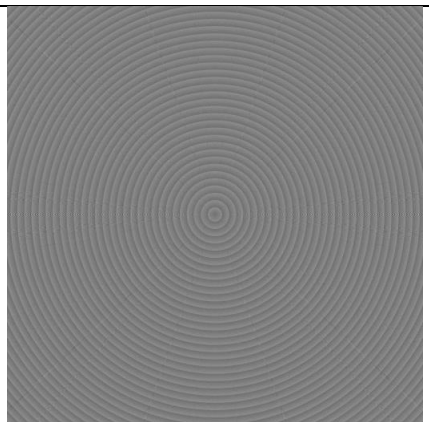

Рисунок 14. Результаты вычислений для мультипарадигменного подхода в виде полутонового изображения. Фаза аксикона (1) приведена к диапазону $[0; 2\pi]$.

В дальнейшем планируется использование рассматриваемого подхода, создающего концептуально целостную основу применения нескольких парадигм программирования, для расчёта дифракционных оптических элементов других типов, а также для расчёта двумерных фотонно-кристаллических и фотонно-квазикристаллических волноводных структур [22].

7. Заключение

В работе проанализированы достоинства и недостатки трёх основных парадигм программирования применительно к разработке программного обеспечения по компьютерной оптике. Исследована возможность применения мультипарадигменного подхода для реализации вычислительных приложений на языке C# для компьютерной оптики. Использование нескольких парадигм в научно-исследовательской работе подталкивает исследователя обращать внимание на анализ общности и изменчивости – основу предлагаемого подхода. Может показаться, что для программирования, тем более научного, подходят только конкретные модели. Может показаться также, что предлагаемые в статье обобщённые компоненты не полностью определены. Но на самом деле эта недоопредлённость подталкивает к дополнительному анализу и более общим решениям, в которых, при использовании средств и механизмов нескольких парадигм программирования, появляется возможность создания более широкого спектра реализаций и применения к более широкому классу задач [5,22,23].

8. Благодарности

Работа выполнена при поддержке РФФИ (грант № 18-29-03303).

9. Литература

[1] Ортега, Дж. Введение в численные методы решения дифференциальных уравнений / Дж. Ортега, У. Пул – М.: Наука, 1986. – 288 с.
[2] Макконнелл, С. Совершенный код – М.: Питер, 2005. – 867 с.
[3] Буч, Г. Объектно-ориентированный анализ и проектирование с примерами приложений – М.: Вильямс, 2008. – 718 с.
[4] Coplien, J.O. Multi-Paradigm Design for C++ – Canada: Addison-Wesley, 1998. – 304 p.
[5] Сойфер, В.А. Дифракционная компьютерная оптика – М.: «Физматлит», 2007. – 736 с.
[6] Floyd, R.W. Paradigms of Programming // Communications of the ACM. – 1979. – Vol. 22(8). – P. 455-460.
[7] Яблоков, Д.Е. Парадигмы программирования // Prospero. – 2015. – Т. 2, № 14. – С. 94-98.
[8] Калинин, А.Г. Универсальные языки программирования: Семантический подход / А. Г. Калинин, И. В. Мацкевич – М.: Радио и связь, 1991. – 398 с.
[9] Волотовский, С.Г. Программное обеспечение по компьютерной оптике / С.Г. Волотовский, Л.Л. Досколович, М.А. Голуб, Н.Л. Казанский, В.С. Павельев, П.Г. Серафимович, В.А. Сойфер, А.Г. Храмов// Компьютерная оптика. – 1995. – Т. 14-15, № 2. – С. 94-106.
[10] Агафонов, В.Н. Данные в языках программирования: Абстракция и типология. Сб. статей – М.: Мир, 1982. – 328 с.
[11] Гамма, Э. Приёмы объектно-ориентированного проектирования. Паттерны проектирования – М.: Питер, 2014. – 366 с.
[12] Степанов, А. Начала программирования / А. Степанов, П. Мак-Джоунс – М.: Вильямс, 2011. – 272 с.
[13] Степанов, А. От математики к обобщённому программированию / А. Степанов, Д. Роуз – М.: Пресс, 2015. – 264 с.
[14] Passos, W. Numerical methods, algorithms, and tools in C# – CRC Press, 2009. – 600 p.
[15] Xu, J. Practical Numerical Methods with C# – Unicomp, 2008. – 408 p.
[16] Скит, Дж. C# для профессионалов, тонкости программирования – М.: Вильямс, 2014. – 602 с.

[17] Jarvi, J. Associated Types and Constraint Propagation for Mainstream Object-Oriented Generics / J. Jarvi, J. Willcock, A. Lumsdaine // ACM SIGPLAN Notices. – 2005. – Vol. 40(10). – P. 1-19.
[18] Austern, M.H. Generic Programming and the STL: Using and Extending the C++ Standard Template Library / M.H. Austern – Boston: Addison–Wesley, 1999. – 304 p.
[19] [Electronic resource]. – Access mode: https://docs.microsoft.com/ru-ru/dotnet/csharp/programming-guide/classes-and-structs/extension-methods.
[20] Яблоков, Д.Е. Исследование возможности увеличения производительности обобщённых алгоритмов за счёт использования развитой семантики обработки данных на основе параметрического полиморфизма подтипов // Аналитические и численные методы моделирования естественно-научных и социальных проблем Материалы XIII Международной научно-технической конференции, 2018. – С. 169-174.
[21] Albahari, J. C# 6.0 in a Nutshell: The Definitive Reference / J. Albahari, B. Albahari – Farnham: O'reilly, 2015. – 1115 p.
[22] Gavrilov, A.V. Diffractive Nanophotonics / A.V. Gavrilov, D.L. Golovashkin, L.L. Doskolovich, P.N. Dyachenko, S.N. Khonina, V.V. Kotlyar, A.A. Kovalev, A.G. Nalimov, D.V. Nesterenko, V.S. Pavelyev, Y.O. Shuyupova, R.V. Skidanov, V.A. Soifer – Boca Raton: CRC Press, Taylor & Francis Group, CISP, 2014.
[23] Харитонов, С.И. Решение обратной задачи фокусировки лазерного излучения в плоские области в рамках геометрической оптики / С.И. Харитонов, Л.Л. Досколович, Н.Л. Казанский // Компьютерная оптика. – 2016. – Т. 40, № 4. – С. 439-450. DOI: 10.18287/2412-6179-2016-40-4-439-450.

Computer optics software development using multi-paradigm design

D.E. Yablokov[1], V.S. Pavelyev[2,3], A.N. Agafonov[2,3], A.V. Eremin[3]

[1]«IntellectSoft » RAS, Michurin street 52, Samara, Russia, 443086
[2]Image Processing Systems Institute of RAS - Branch of the FSRC "Crystallography and Photonics" RAS, Molodogvardejskaya street 151, Samara, Russia, 443001
[3]Samara National Research University, Moskovskoe Shosse 34A, Samara, Russia, 443086

Abstract. The common goal for all stages of software design is the components reusing that corresponds to the basic principles of the multi-paradigm approach considered in the article. The main aim of the article is a forming of new conceptual bounds in the software design based on the most significant features of procedural, object-oriented and generic programming paradigms. For each of paradigms using an example of calculation of an axicon phase function on a given set of points, their advantages and disadvantages are considered. Code snippets were created using the C# language supporting several programming paradigms that allows defining algorithmic, behavioral and syntactic commonality. The multi-paradigms approach creates the ability for the implementation of different programming techniques during research in the area of computer optics.