МЕТОД АБСТРАКЦИИ В СОВРЕМЕННЫХ ЯЗЫКАХ ПРОГРАММИРОВАНИЯ

Швец А. Г.
Кафедра ЭВМ ДонГТУ
shvets@cs.dgtu.donetsk.ua

Abstract

Shvets A.G. The method of abstraction in the modern programming languages. *The basic concepts, arising at the last time in the programming, are discussed. The main idea of the programming as scientific subject are delivered. The method of abstraction is considered as the main meaning for realization of the new concepts. The term evolution for the data and operations are offered. In the conclusion the main requirements to the modern programming language are formulated.*

Введение

В настоящее время наблюдается стремительное развитие научной дисциплины, называемой программированием. При этом появляются не просто новые языки, появляются новые идеи, увеличивающие мощность и эффективность языков. Можно, не вдаваясь в подробности любого из существующих или только разрабатываемых языков, отметить следующую тенденцию: развитие языков идет в сторону повышения *выразительности* исходного текста программы. Это способствует сокращению размера программы и повышению ее надежности [1].

Для повышения выразительности языка необходимо, чтобы язык содержал средства для выражения *абстрактных понятий*. Это помогает сделать большие программы более простыми для понимания. Поэтому поддержка абстракций является обязательным условием для любого современного языка программирования. При этом базис языка (множество предоставляемых языком возможностей, смысловых конструкций) должен иметь минимальную мощность.

К наиболее общим понятиям, которыми оперирует программист при использовании конкретного *языка программирования*, относятся понятия *программы* и *виртуальной машины*. Программа должна удовлетворять требованиям (спецификациям) конкретного языка программирования и служит *контейнером* для хранения последовательности действий и множества данных. Виртуальная машина выступает в роли интерпретатора основных понятий, используемых в языке программирования и является *средой существования* программы. Все остальные абстракции, рассматриваемые в статье, группируются вокруг этих базовых абстракций.

В ряде случаев можно рассматривать процесс программирования как процесс моделирования. При этом создается программа-модель, способная реализовывать *поведение* оригинала, описываемого в постановке задачи. Поэтому в дальнейшем заменителем для понятия программа будет выступать понятие *модель*, а для понятия виртуальная машина - понятие *моделирующая среда*.

Целью данной статьи является анализ использования абстрактных понятий в современных языках программирования и выявление основных тенденций развития современных языков. На базе выполненного анализа формулируются основные требования, предъявляемые к современным языкам программирования.

1. Последовательность нахождения решения задачи

Прежде всего подробно рассмотрим основные процессы, в которые вовлечен программист при написании программы. Пусть имеется задача, требующая своего разрешения. Основная цель работы программиста состоит в том, чтобы используя конкретный язык программирования, создать *модель* системы, соответствующую поставленной задаче. Модель системы представлена в виде программы и существует в моделирующей среде, реализованной в виде виртуальной машины той или иной степени сложности.

В простейшем случае задачей может быть некоторая формула. Такая модель строится тривиально и не требует отдельного рассмотрения. В реальной жизни задача может быть представлена лишь в виде некоторой сложной системы [1]. Такая система состоит из набора сущностей разного типа и связей между этими сущностями. Объектно-ориентированный подход успешно справляется с нахождением таких сущностей и реализацией их в виде смысловых структур языка программирования.

При переходе от поставленной задачи к реализующей ее программе программист должен выполнить определенные действия. Ниже приведен порядок действий программиста:

- Для поставленной задачи составляется ее словесное описание. Этот этап соответствует формулировке, постановке задачи.
- Словесное описание задачи подвергается анализу с целью нахождения обобщенных механизмов, способных решить поставленную задачу. Наиболее общие механизмы должны быть реализованы в языке (например, механизмы контейнеризации и наследования). Остальные механизмы реализуются самим программистом.
- Найденные механизмы реализуются в виде программы при помощи определенных методов программирования с использованием смысловых структур языка. Конкретный язык поддерживает определенное множество методов решения поставленных задач.

Все этапы описанного выше мыслительного процесса тесно взаимосвязаны, а конкретная их реализация соответствует определенной *парадигме программирования*.

2. Типы моделей

Модель системы при программировании представлена в виде программы. Она может быть простой и состоять из одной сущности или быть составной, состоящей из множества сущностей. При этом моделирующая среда должна обеспечивать *одновременное* функционирование этих сущностей. Для решения такого рода задач требуются специальные конструкции языка и своя виртуальная машина, реализующая или симулирующая одновременное существование множества сущностей. Не вдаваясь в подробности реализации той или иной виртуальной машины, можно выделить следующие типы моделей:

- **Статическая модель.** Не имеет значения фактор времени; система представлена в виде графа состояний, переходы в котором зависят от входного потока данных и текущего состояния модели. В качестве примера можно привести модель калькулятора [2].
- **Динамическая модель**. Переходы между состояниями модели зависят как от входного потока данных и текущего состояния модели, так и от дополнительных парамстров, например времени и расстояния.

По мнению автора, при построении программных моделей возможны следующие типы динамических моделей:

1) **Модель типа "клеточный автомат".** Модель представлена в виде последовательности из n компонентов. Для каждого компонента, входящего в модель, выполняется отдельная операция за фиксированный интервал времени *delta*. Выполнение каждой операции не связано с выполнением других операций модели. Моделирование выполняется за

m циклов, причем длительность каждого цикла составляет *delta*n;* в цикле выполняется соответствующая операция для каждого компонента последовательности. Моделирование такой модели возможно на однопроцессорных машинах (псевдопараллелизм). В качестве примера можно привести игру "жизнь" [3].

2) Модель с сосредоточенными параметрами. В этой модели каждая операция имеет различную длительность, поэтому моделирование с использованием псевдопараллелизма недопустимо. Кроме того, могут иметь место зависимости между разными операциями.

В случае модели с сосредоточенными параметрами возможны следующие решения:

- представление операций в виде легковесных процессов или сопрограмм. Это требует использования языков, поддерживающих такие концепции или написание своего планировщика задач [4];
- использование параллельных структур типа SIMD, MIMD; при этом реализуется истинная многозадачность.

3) Модель с распределенными параметрами. В этой модели переходы между состояниями зависят как от времени, так и от расстояния. Такая модель строится на основе решения системы дифференциальных уравнений с распределенными параметрами.

3. Метод абстракции

Метод абстракции представляет собой процесс *игнорирования* несущественных и *выделения* важных деталей рассматриваемого явления (системы, сущности), для которой мы хотим создать модель. В основе метода абстракции лежит принцип наименьшей выразительности: *абстракция должна охватывать лишь самую суть рассматриваемого явления, не больше, но и не меньше* [1]. Результатом использования метода абстракции применительно к программированию должно быть построение некоторой *модели*.

Метод абстракции при построении модели использует следующие операции:

- *разбиение* - выделение наиболее существенных деталей рассматриваемого явления;
- *слияние* - объединение сущностей, присутствующих в рассматриваемом явлении (подразумевается, что явление может быть разложено, *разбито* на составляющие); при этом определенный набор понятий, свойств некоторой сущности рассматривается как одно целое;
- *обобщение* - нахождение наиболее общих черт у разных явлений и отбрасывание незначащих в данный момент подробностей;
- *специализация* - нахождение различий между родственными явлениями.

Операции разбиения и слияния служат для нахождения *состава* разрабатываемой модели (реализуют отношение включения), а операции обобщения и специализации - для отображения *отношений родства* между разными явлениями (реализуют отношение наследования). Применяя те или иные операции метода абстракции, мы можем выразить сложные взаимоотношения реального мира в виде ограниченного множества абстракций. В следующих частях статьи сделана попытка отображения основных приемов, применяемых в современном программировании, через набор операций метода абстракции.

4. Разбиение

Разбиение реализуется в двух полезных абстрактных приемах: декомпозиции и модульности. Рассмотрим эти приемы более подробно.

При реализации сложных систем размер программ увеличивается лавинообразно, так как нужно реализовывать сложные модели. Однако, в то время как сложность реализуемых моделей растет, возможности человеческого мозга ограничены. Многие психологи счи-

тают, что число единиц информации, которое может усвоить человек одновременно, не превышает семи [1]. Причиной этому является ограниченный объем краткосрочной памяти человека. Для разрешения возникающего противоречия необходимо научиться выполнять *декомпозицию* сложных систем. При этом происходит разбиение сложной системы на слабо связанные компоненты. Данный прием широко используется в объектно-ориентированном программировании на начальных этапах при поиске подходящей иерархии классов, эффективно описывающей исходную задачу.

Модульность подразумевает разбиение программы на осмысленные части (набор программных модулей). Разбиение на модули осуществляется таким образом, чтобы объединить логически связанные абстракции и минимизировать логические связи между абстракциями; это влечет за собой минимизацию использования глобальных переменных. Можно рассматривать *модуль* как отдельно компилируемый фрагмент программы (файл).

Модульность отражает представление сложной системы в виде слабо связанных подсистем. Она поддерживает защиту данных - скрывает внутри модулей реализации объектов, несущественные для рассмотрения на следующем уровне абстракции, и инкапсуляцию - позволяет рассматривать группу объектов как одно целое (модуль). Идея модульности имеет свое развитие в форме *пакетов* [6]. Они позволяют рассматривать несколько модулей как одно целое и позволяют создание произвольного числа уровней иерархии.

5. Слияние

Операция слияния находит свое отражение в приеме, называемом контейнеризацией или объединением. При этом происходит объединение нескольких сущностей и рассмотрение их как одно целое. Операция слияния является очень мощным и одним из наиболее продуктивных способов порождения нового в программировании. Покажем, как используя объединение, можно построить основные смысловые структуры, используемые в современном программировании. Прежде всего введем основные понятия.

- **Виртуальная машина Ассемблера** - это исполняющая среда для интерпретации смысловых структур языка ассемблера. Она состоит из двух главных компонент: памяти и процессора.
- **Память** - это устройство, предназначенное для хранения информации. Она состоит из набора *элементарных ячеек*, способных хранить *элементарные данные*.
- **Процессор** - это устройство, предназначенное для исполнения заданной последовательности команд (операций, инструкций). Процессор способен реализовывать некоторые *преобразования* над данными, находящимися в памяти.

Простое объединение элементарных данных и ячеек порождает новое понятие.

- **Программа** - это простое объединение *множества* элементарных данных и *последовательности* операций, используемая для нахождения решения некоторой задачи.

В результате применения операции слияния отдельно к элементарным данным и элементарным операциям образуются понятия структуры данных и функции.

- **Структура данных** - это объединение множества элементарных данных.
- **Функция** - это объединение последовательности элементарных операций.

Структура данных позволяет размышлять о некотором наборе данных как об одном целом. Однако при манипуляции с таким набором приходится обращаться к отдельным элементам набора, что в общем случае не способствует идее абстракции. Недостаток структуры данных состоит в том, что при этом возможно создание только одного уровня абстракции.

Введение понятия структуры данных в язык влечет за собой *типизацию* языка. Это означает, что язык программирования имеет дело с сущностями, соответствующими тому или иному *типу*. Программа при этом содержит дополнительный раздел, содержащий описание типа. Используя это описание, программист может создавать *экземпляры*, представители этого типа.

При использовании функции сокращается размер исходного текста программы и появляется возможность вводить элементарные абстракции, соответствующие некоторым действиям в программе. Это облегчает понимание программы и позволяет мыслить в терминах логических операций. Составляя функцию, мы создаем определенный интерфейс между пользователем и кодом, соответствующим этой функции. Это указывает на тесную связь слияния с понятием защиты данных.

Язык, использующий новые абстракции, должен иметь свою виртуальную машину.

- **Структурированная виртуальная машина** - это исполняющая среда для интерпретации языков, поддерживающих абстракции структуры данных и функции.

Используя операцию слияния к понятиям структура данных и функции и рассматривая полученную сущность как новый тип, получим новое понятие.

- **Абстрактный тип данных** (АТД) - это объединение структуры данных и набора функций с целью создания нового типа.

Для интерпретации абстрактного типа данных используется **виртуальная машина АТД**.

6. Обобщение

При добавлении к АТД свойства наследования получим понятие *класса*.

- **Класс** - это абстрактный тип данных, который обладает свойством *наследования*.

Наследование позволяет указывать, что создаваемый класс наследует функции и данные базового класса и может добавлять свои функции и данные. Экземпляр класса называется *объектом*.

Для обслуживания классов необходимо использовать **объектно-ориентированную виртуальную машину**.

Введение наследования позволяет использовать операцию обобщения. При этом ищутся общие черты у разных сущностей, а различия между ними не учитываются. Это позволяет для существующих классов вводить базовый класс, общий для обеих классов.

Выполняя обобщение, программист может предоставлять пользователю классов только ту информация, которая в данный момент имеет для него значение. Остальная информация является недоступной и скрыта от глаз пользователя. В качестве примера можно привести понятие защиты данных. Структура описания класса в объектно-ориентированных языках может состоять из закрытых областей, доступных только для методов данного класса, защищенных областей, доступным методам производных классов и открытых областей, доступных любым методам [6].

На основе обобщения строятся *полиморфные функции*. Такая функция полагает, что объекты, которыми она манипулирует, имеют базовый тип. Это означает, что все специфические свойства объекта отбрасываются. Таким образом достигается повышение абстрактности реализуемого функцией кода.

7. Специализация

Операция *специализации* является обратной к операции обобщения. При ее использова-

нии идет поиск различий между схожими явлениями.

Специализация позволяет породить от одного класса несколько производных, отличающихся друг от друга только частично. Например, исследуя различные типы автомобилей можно прийти к выводу, что класс автомобиль может быть *специализирован*, а в результате специализации могут появиться такие классы, как спортивный автомобиль, грузовой автомобиль и др.
Операция специализации используется при проведении *классификации* [1].

Заключение

Исходя из вышеизложенного, современный язык программирования должен обладать следующими свойствами:

- основным средством для реализации понятия типа является класс;
- класс должен поддерживать концепции контейнеризации и наследования;
- класс должен иметь механизмы для сокрытия в случае необходимости своих компонент;
- классы и экземпляры классов (объекты) могут располагаться в разных модулях;
- механизм модульности должен поддерживать произвольное число уровней вложенности;
- объекты должны иметь возможность одновременного существования (для реализации динамических моделей);
- виртуальная машина языка должна поддерживаться явным образом.

Чтобы добиться максимального эффекта от объектно-ориентированного подхода, необходимо его использование не только при написании программ, но и во всех остальных случаях:

- при разработке архитектуры компьютера;
- при построении операционной системы;
- при построении пользовательского интерфейса;
- при разработке универсальных программ (например, при разработке баз знаний и баз данных, редакторов, компиляторов и др.).

Из всех современных языков максимально перечисленным выше требованиям соответствует язык Java [6]. Он является полностью объектно-ориентированным языком и поддерживает объектную ориентацию уровня пользовательского интерфейса и универсальных программ. Виртуальная машина поддерживается явным образом. Допускается параллельное существование разных объектов. В будущем запланировано реализовывать виртуальную машину Java на аппаратном уровне (выпускаются специальные Java-процессоры). Современные операционные системы начинают поддерживать язык Java на уровне операционной системы (OS\2). Кроме этого, разработана операционная система, специально разработанная для приложений, написанных на языке Java (JavaOS 1.0). Таким образом, предлагается комплексное решение, охватывающее все возможные уровни.

Литература

1. Буч Г. Объектно-ориентированное проектирование с примерами применения. - М.: Конкорд, 1992.- 519 с.
2. Страуструп Б. Язык программирования С++. - М.: Радио и связь, 1991.- 352 с.
3. Гарднер М. Крестики-нолики. - М.: Мир, 1988. - 352 с.
4. Дьюхарст С., Старк К. Программирование на С++. - Киев: "ДиаСофт", 1993.-272с.
5. Вирт Н. Программирование на языке Модула-2.-М.: Мир, 1987.- 224 с.
6. Нейл Барлет, Алекс Лесли, Стив Симкин. Программирование на Java. Путеводитель. - Киев: "ДиаСофт", 1996.