DOI: 10.17117/na.2016.10.03.205

<http://ucom.ru/doc/na.2016.10.03.205.pdf>

Поступила (Received): 02.10.2016

Малышев И.Е. Альтернативное программирование

Malyshev I.E.
Alternative programming

В статье рассматриваются самые современные парадигмы программирования, отличающиеся от наиболее используемых или дополняющие их. Описывается суть каждой из них и рассказывается о причинах и результатах их появления. Приведено несколько примеров языков, в которых новые подходы используются уже достаточно давно, обладают наибольшей популярностью и эффективностью

Ключевые слова: информационные технологии, программирование, парадигмы

The article deals with the most modern programming paradigms. They differ from most used or complement them. It describes the essence of each of them. It is told about the reasons and results of their appearance. There are several examples of languages in which the new approaches have been used for a long time. They are the most popular and effective

Key words: information technology, programming, paradigm

Малышев Илья Евгеньевич

Студент

Калужский филиал Московского государственного технического университета им. Н.Э. Баумана
г. Калуга, ул. Баженова, 2

Malyshev Ilya Evgenyevich

Student

Kaluga branch of Moscow state technical university named N.E. Bauman
Kaluga, Bazenova st., 2

Язык программирования – формальная знаковая система, предназначенная для записи компьютерных программ, они делятся по разным принципам, например, на высокоуровневые и низкоуровневые, на процедурные, декларативные, объектно-ориентированные, сетевые и т.д. Наиболее распространена на данный момент парадигма объектно-ориентированного программирования [1].

Объектно-ориентированное программирование (ООП) – методология программирования, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определенного класса, а классы образуют иерархию наследования [2].

Прототипное программирование – стиль объектно-ориентированного программирования, при котором отсутствует понятие класса, а наследование производится путём клонирования существующего экземпляра объекта – прототипа. Каноническим примером прототип-ориентированного языка является язык Self. В дальнейшем этот стиль программирования начал обретать популярность и был положен в основу таких языков программирования, как JavaScript, Lua, Io, REBOL и др.

В языках, основанных на понятии «класс», все объекты разделены на два основных типа – классы и экземпляры. Класс определяет структуру и функциональность (поведение), одинаковую для всех экземпляров данного класса. Экземпляр является носителем данных – то есть обладает состоянием, меняющимся в соответствии с поведением, заданным классом.

Сторонники прототипного программирования часто утверждают, что языки, основанные на классах, приводят к излишней концентрации на систематизации классов и на отношениях между ними. В противоположность этому, прототипирование заостряет внимание на поведении некоторого (небольшого) количества «образцов», которые затем классифицируются как «базовые» объекты и используются для создания других объектов. Многие прототип-ориентированные системы поддерживают изменение прототипов во время выполнения программы, тогда как лишь небольшая часть класс-ориентированных систем (например, Smalltalk, Ruby) позволяет динамически изменять классы.

Хотя подавляющее большинство прототип-ориентированных систем основаны на интерпретируемых языках с динамической типизацией, технически возможно добавить прототипирование и в языки со статической проверкой типов. Язык Omega является одним из примеров такой системы.

В класс-ориентированных языках новый экземпляр создаётся через вызов конструктора класса (возможно, с набором параметров). Получившийся экземпляр имеет структуру и поведение, жёстко заданные его классом.

В прототип-ориентированных системах предоставляется два метода создания нового объекта: клонирование существующего объекта, либо создание объекта «с нуля». Для создания объекта с нуля программисту предоставляются синтаксические средства добавления свойств и методов в объект. В дальнейшем, из получившегося объекта может быть получена полная его копия – клон. В процессе клонирования копия наследует все характеристики своего прототипа, но с этого момента она становится самостоятельной и может быть изменена. В число преимуществ данного подхода входит тот факт, что создатель копии может менять её, не опасаясь побочных эффектов среди других потомков своего предка. Также радикально нет необходимости искать подходящие слоты (метода или атрибута).

Компонентно-ориентированное программирование – парадигма программирования, существенным образом опирающаяся на понятие компонента – независимого модуля исходного кода программы, предназначенного для повторного использования и развёртывания и реализующегося в виде множества языковых конструкций (например, «классов» в объектно-ориентированных языках программирования), объединённых по общему признаку и организованных в соответствии с определёнными правилами и ограничениями.

Компонентно-ориентированное программирование, в применении к объектно-ориентированным языкам, подразумевает набор ограничений, налагаемых на объектные механизмы – и нацеленных, прежде всего, на повышение надёжности больших программных комплексов. При применении «неограниченного» объектно-ориентированного подхода, характерна проблема «хруп-

ких» базовых классов: ситуация, когда изменить реализацию типа-предка невозможно, не нарушив корректность функционирования типов-потомков (даже при прежних интерфейсах его методов). Компонентно-ориентированный подход, позволяющий через понятие «компонента» объединить в группу связанные конструкции и определяющий схемы и правила для них – позволяет избежать подобного рода проблем.

Компонентно-ориентированный подход может применяться во многих языках программирования, с помощью стандартных конструкций (таких как: классы, интерфейсы, пакеты, модули).

Существуют языки программирования, реализующие на конструктивном уровне компонентно-ориентированное программирование: «Оберон» (ограниченно), «Компонентный Паскаль». В рамках языка Java –компонентно-ориентированное программирование реализуется посредством компонентов, называемых JavaBeans, поддержанных в одной из ранних спецификаций языка. В платформе «.NET» –реализован компонентно-ориентированный подход, обеспечивающий создание и повторное использование компонентов для любого языка программирования, поддерживаемого платформой.

Агентно-ориентированный подход к программированию – разновидность представления программ или парадигма программирования, в которой основополагающими концепциями являются понятия агента и его ментальное поведение, зависящее от среды, в которой он находится.

Агентом является всё, что может рассматриваться как воспринимающее свою среду с помощью датчиков и воздействующее на эту среду с помощью исполнительных механизмов.

Агентно-ориентированный подход, представляется, как новая парадигма, в которой слово агент может использоваться в различных близких смыслах. И хотя под агентом обычно понимается программный агент, он базируется на теории искусственного интеллекта и отталкивается от понятия интеллектуальный агент, которое уже существовало ранее. В итоге понятие агента, в отличие от простого объекта, наделяется рядом ментальных конструкций, таких как вера, обязанности и способности. Поэтому в языке программирования будут появляться различные ментальные категории, а семантика программирования будет связана с семантикой ментальных конструкций:

– Объект – Программная сущность заданной структуры и конкретизированных механизмов взаимодействия с другими объектами посредством передачи сообщений. Сообщения формируются и отсылаются в ответ на пришедшие сообщения. Формирование сообщений происходит процедурами на основании данных.

– Актор – Программная сущность заданной структуры и механизмов взаимодействия, содержит данные и процедуры, обладает инкапсуляцией, отношениями, наследованием и может порождать сообщения.

– Агент – Программная сущность для выполнения поставленных задач. Обладает поведением, а именно: взаимодействует с внешней сложной и динамично-развивающейся средой, способной модифицироваться или быть модифицированной другими агентами в зависимости от конкретных условий.

Взаимодействие подразумевает: восприятие динамики среды; действия, изменяющие среду; рассуждения в целях интерпретации наблюдаемых явлений, решения задач, вывода заключений и определения действий.

Агентно-ориентированный подход является частным случаем (специализацией) объектно-ориентированного программирования (ООП). В ООП вычислительный процесс понимается достаточно широко как система, собранная из объектов, которые взаимодействуют друг с другом через сообщения. АОП специализирует эти понятия, устанавливая состояние (называемое психическим состоянием) объектов (называемых агентами), состоящих из компонентов таких как верование (убеждения) (включая убеждения о мире, о себе, и о друг друге), способности, и решения, каждое из которых обладает определенным синтаксисом. Кроме того, различные ограничения помещены в психическое состояние агента. Вычисление состоит из информирования агентов, выполнения их требований, выполняя их предложения, принимая, отклоняя, конкурируя, и помогая друг другу.

Аспектно-ориентированное программирование (АОП) – парадигма программирования, основанная на идее разделения функциональности для улучшения разбиения программы на модули. Методология АОП была предложена группой инженеров исследовательского центра Xerox PARC. Ими же было разработано аспектно-ориентированное расширение для языка Java, получившее название AspectJ – (2001 год).

Существующие парадигмы программирования предоставляют определённые способы для разделения и выделения функциональности: функции, модули, классы, но некоторую функциональность с помощью предложенных методов невозможно выделить в отдельные сущности. Такую функциональность называют сквозной, так как её реализация распределена по различным модулям программы. Сквозная функциональность приводит к рассредоточенному и запутанному коду, сложному для понимания и сопровождения.

Ведение лога и обработка исключений – типичные примеры сквозной функциональности. Другие примеры: трассировка; аутентификация и проверка прав доступа; контрактное программирование. Для программы, написанной в парадигме ООП, любая функциональность, по которой не была проведена декомпозиция, является сквозной. Однако, как утверждают некоторые авторы, АОП может успешно применяться и для решения задач защиты, многопоточности, управления транзакциями и многих других.

Все языки АОП предоставляют средства для выделения сквозной функциональности в отдельную сущность. Так как AspectJ является родоначальником этого направления, используемые в этом расширении концепции распространились на большинство языков АОП.

Основные понятия АОП:

– Аспект – модуль или класс, реализующий сквозную функциональность. Аспект изменяет поведение остального кода, применяя совет в точках соединения, определённых некоторым срезом.

– Совет – средство оформления кода, которое должно быть вызвано из точки соединения. Совет может быть выполнен до, после или вместо точки соединения.

– Точка соединения – точка в выполняемой программе, где следует применить совет. Многие реализации АОП позволяют использовать вызовы методов и обращения к полям объекта в качестве точек соединения.

– Срез – набор точек соединения. Срез определяет, подходит ли данная точка соединения к данному совету. Самые удобные реализации АОП используют для определения срезов синтаксис основного языка и позволяют их повторное использование с помощью переименования и комбинирования.

– Внедрение – изменение структуры класса и/или изменение иерархии наследования для добавления функциональности аспекта в инородный код. Обычно реализуется с помощью некоторого метаобъектного протокола.

В итоге можно сказать, что существует множество парадигм программирования дополняющих объектно-ориентированное программирование, позволяя программисту использовать в разработке новые эффективные инструменты.

Список используемых источников:

1. Голицына О.Л. Попов И.И. Основы алгоритмизации и программирования. М.: ФОРУМ, 2008. 342 с.
2. Иванова Г.С., Ничушкина Т.Н. Объектно-ориентированное программирование. М.: Изд-во МГТУ им. Н.Э. Баумана, 2014. 455 с.