УДК № 004.42

Современные перспективы развития компонентно-ориентированного подхода

М. Ю. Павлов, А. В. Боднар
Донецкий национальный технический университет, г. Донецк
кафедра программной инженерии им. Л.П. Фельдмана
E-mail: vigototheroad@mail.ru

Аннотация
В работе рассматривается компонентно-ориентированный подход, его первостепенные задачи и философия, выделяющая данную парадигму среди других. Основная часть работы посвящена описанию концепции подхода, рассмотрению преимуществ, выделение отличительных черт разработки, проанализированы сложности при введении концепции, оценены недостатки, трудность осуществления подхода, найдены проблемы в интегрировании парадигмы, рассмотрена реализация и перспективы дальнейшей разработки рассматриваемого подхода.

Введение

Программирование не стоит на месте, задачи порождают решения, а старое пересматривается под призмой современных технологий, что позволяет дополнить концепции новым пониманием для решения поставленных задач. Так, когда-то родился ООП подход, паттерны и другие концепции программирования. В данной статье мы рассмотрим одну из парадигм программирования, названную компонентно-ориентированный подход [1].

Компонентно-ориентированный подход (КОП) – подход в проектировании и программировании, основанный на рассматривании системы как набора объектов, состоящего из компонентов. КОП развивает ООТ путем устранения его недостатков. Хоть ООП и показывает себя очень хорошо, разработка сложных систем, моделирующих процессы окружающего мира, вызывает ряд проблем, которые можно решить лишь другим подходом.

Основную популярность КОП получил благодаря движкам разработки игр. Каждый движок имеет разные преимущества, которые тяжело реализовать в обычной системе ООП из-за жесткой взаимосвязи между классами, поэтому разбиение особенностей на компоненты позволяет оптимизировать реализацию как для программиста, так и для машины. Помимо этого, движки обладают собственными компонентами и особенностями: например, в Unity у нас есть компонент Animator, который способен самостоятельно запускать анимацию после перемещения; само взаимодействие возможно запустить путем использования компонента Input, который будет ловить захват клавиш. Это взаимодействие позволяет исключить постоянное переписывание кода для простой реализации, а также модернизировать уже имеющие компоненты новыми.

Но использование КОП не ограничивается использованием его в одних игровых движках. Считается, что КОП зародился в 1960-х годах. Тогда были разработаны первые компьютеры и концепция модульности, то есть разделение программ на разные модули, выполняющие определенные функции. После, более точная формулировка появилась примерно в середине 90-х [1], когда Никлаус Вирт предложил паттерн написания блоков. В то время данная парадигма расшифровывалась как компонентно-ориентированное программирование. Паттерн предполагал, что созданный компонент компилируется отдельно от остальной программы, а на стадии выполнения необходимые компоненты подключаются по мере необходимости. Всё это должно происходить динамически [2]. Суть подобного программирования была в том, чтобы избавиться от «хрупких» классов, возникающих в процессе создания иерархии. Такая концепция в будущем породила множество готовых решений, которыми пользуются до сих пор

В процессе своего развития КОП стал основным принципом разработки для популярных движков, таких как Unity Engine. Парадигма получила большое распространение в создании корпоративных приложений, где компоненты стали представлять из себя бизнес-логику, БД и интеграцию с другими системами; стали развиваться паттерны, нацеленные на описание и поддержание такого подхода, что позволило развивать идею. Тем не менее, использование КОП не является панацеей в разработке, поэтому при нем возможно использование других принципов для

упрощения, развития и улучшения этапов разработки. КОП можно считать относительно современным подходом, поскольку он подвергся переосмыслению и дополнению. Как и любой подход, КОП постоянно развивается. Чтобы эффективно обновляться, необходимо знать особенности КОП, выделяющие этот метод на фоне других.

Преимущества КОП

Одним из главных преимуществ КОП, является его легкое интегрирование в систему. Это достигается за счет главного свойства компонентов – независимости. Поскольку компоненты не зависят друг от друга, это позволяет гибко работать с системой и наделять объекты только необходимыми компонентами. За этим и следует такое преимущество: каждый объект обладает только теми компонентами, которые необходимы ему для реализации функционала.

Поскольку компоненты являются независимыми, то их легко можно заменить, что добавляет гибкости проекту в реализации [2]. Допустим, нам необходимо реализовать разные способы перемещения для не игровых персонажей. В этом случае мы используем разные компоненты, которые имеют реализацию через систему, или можем использовать паттерн «Стратегия».

Жизненный цикл компонента дополняется со временем [3], поскольку идея компонента в том, что он является частью модели и выполняется по необходимости. Следовательно, можно использовать разные компоненты для описания разных объектов, словно это свойства, участвующие во время взаимодействия. Данное отличие позволяет легко дополнять существующий функционал, притом никак не влиять на другой функционал и компоненты, связанные с ним.

Еще одной особенностью является взаимодействие, выполняющееся по необходимости. То есть, даже если объекты схожи, они могут выполнять разные функции, что повлияет на компоненты. При этом сами компоненты, участвующие во взаимодействии, могут и вовсе ничего не знать друг о друге, поэтому компоненты используют динамический обмен информацией при выполнении события. Это уменьшает нагрузку, ведь нам не нужно постоянно помнить обо всех компонентах – программа определяет, какие компоненты нужны в конкретный промежуток времени, позволяя облегчить управление памятью.

Изменяемость также является большим плюсом. При написании неправильного взаимодействия не будет проблемы найти причины ошибочного процесса, ведь связь между компонентами минимальна, а то и во все позволяет убрать данное взаимодействие без негативного влияния на продукт [2]. При этом совершенно не нарушится связь с другими компонентами.

Благодаря простой связи компоненты имеют великолепное свойство повторного использования. Это оптимизирует создание новых компонентов и и модернизацию уже существующих внутри одной компании или для определенного инженера. Совершенствование определенной механики позволяет упростить разработку и в свою очередь улучшить качество продукта, позволив сосредоточиться на новых важных элементах. К тому же, это снижает затраты на разработку, что благоприятно сказывается на финальном продукте.

Большим бонусом является облегчение тестирования [3]. Это преимущество связано с прошлым, поскольку вполне возможно создать компонент отдельно от общей разработки и спокойно протестировать его, так как это ограниченная область – значит, другие элементы программы не должны на неё влиять, что в свою очередь подтверждает уверенность программиста в написанном продукте и упрощает его презентацию другим членам разработки или заказчику.

Использование стандартных компонентов позволяет упростить обучение и поддержку системы, а также снизить порог вхождения новым разработчикам в проект [2]. Предполагается, что компоненты будут достаточно емкими, чтобы описывать сами себя. Они не требуют знания всего функционала для своей реализации.

Недостатки КОП

Каким бы продуманным не была парадигма программирования или создания кода, она не идеальна и имеет свои минусы, связанные с отличительными особенностями. Рассмотрим минусы КОП.

Начнем с минуса, который первым всплывает при начальной разработке продукта или попытки полного внедрения КОП. Данный подход может требовать больше времени на разработку, так как необходимо создать и настроить каждый компонент отдельно. Чем больше проект, чем сложнее внедрить КОП. Помимо этого, система будет очень громоздкой. Без опыта работы с подобным крайне сложно ориентироваться и планировать дальнейшую разработку [4].

Как и любой другой код, компоненты могут подвергаться неправильному проектированию, неумышленному усложнению или просто быть некачественно продуманы и реализованы. Это может приводить к усложнению системы, что негативно влияет на разработку и дополнение проекта. К тому же,

появляется шанс, что новые сотрудники запутаются и не смогут должным образом разобраться в работе. Негативные последствия подобного подхода могут выражаться в снижении производительности программы и увеличении количества багов.

Развитие компонентов может иметь не только позитивные последствия. При неправильном выборе вектора развития вероятно, что компоненты могут стать узконаправленными. Это порождает сложность внедрения в конкретный проект, а порой и вовсе указывает на то, что следует избавиться от уже готового решения в пользу нового. Хоть в этом и нет ничего страшного, нельзя забывать о направленности проекта. Компоненты и система должны решать поставленные перед проектом задачи и не вносить лишних взаимодействий. По этой причине многие рабочие компоненты переписываться под нужды проекта [5].

Использование слишком большого количества компонентов затрудняет понимание работы системы в целом. В ходе разработки системы могут появляться схожие компоненты, что повлечет копирование и, как следствие, появление ненужных затрат. Помимо этого, руководитель проекта должен знать досконально систему и уже имеющиеся в ней компоненты, чтобы грамотно определять новые задачи, которые в процессе разработке еще предстоит решить.

Хоть и компоненты являются основной силой КОП, они также являются и её слабостью. При разработке создание сложных компонентов откликается на стоимости продукта, даже если студия или компания ранее занимались развитием этого взаимодействия. Поскольку КОП не очень распространен, цены на отдельные компоненты в условиях рынка отражают эту нераспространённость.

Пример реализации КОП в Unity

В качестве примера реализации возьмем разработку игры на движке Unity и рассмотрим спроектированные компоненты с приведением подробного объяснения.

Первый пример будет связан с перемещением камеры и выполнением компонента Animator. Для начала рассмотрим схему на рисунке 1.

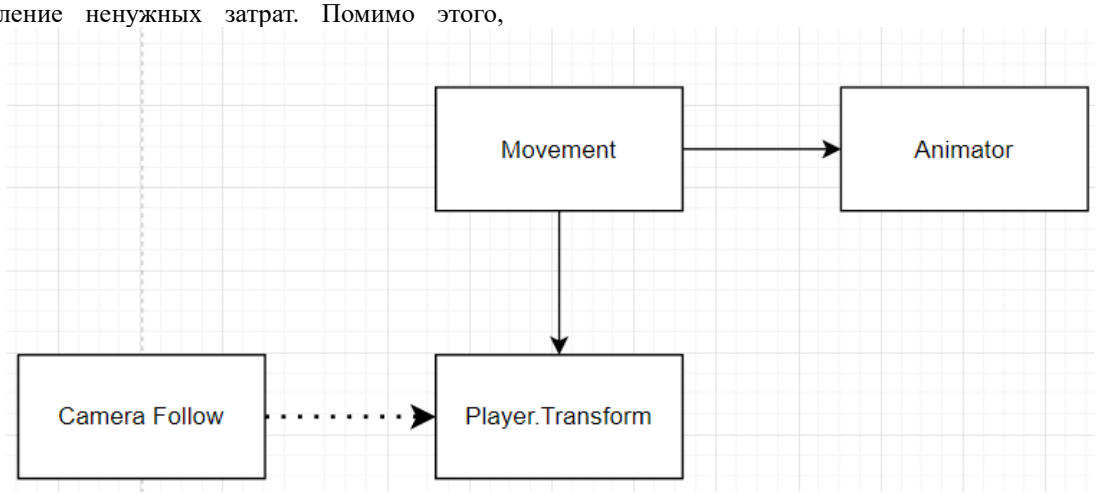

Рисунок 1 – Реализация компонентов перемещения камеры, передвижения и Animator

На данной схеме изображены 4-е компонента. Каждый компонент будет прикреплен к объекту, с которым он взаимодействует. Так, Movement будет прикреплен к Player, как и Transform, отображающий его текущую позицию на экране. Компонентная связь работает следующим образом: при совершении движения компонентом Movement изменяется компонент Transform, при этом Animator из-за изменения компонента Movement запускает анимацию движения, а поскольку компонент Movement изменил параметры Transform, то Camera Follow стала перемещаться за игроком.

Данный подход можно использовать комбинированно, что приведет к улучшению функционала компонентов, но усложнит разработку. Тем не менее, это позволит сделать отдельные компоненты более универсальными и расширит их использование. Также при большом количестве компонентов появляется один из главных минусов данного подхода, а именно постоянное усложнение общего курса системы. Пример приведен на рисунке 2.

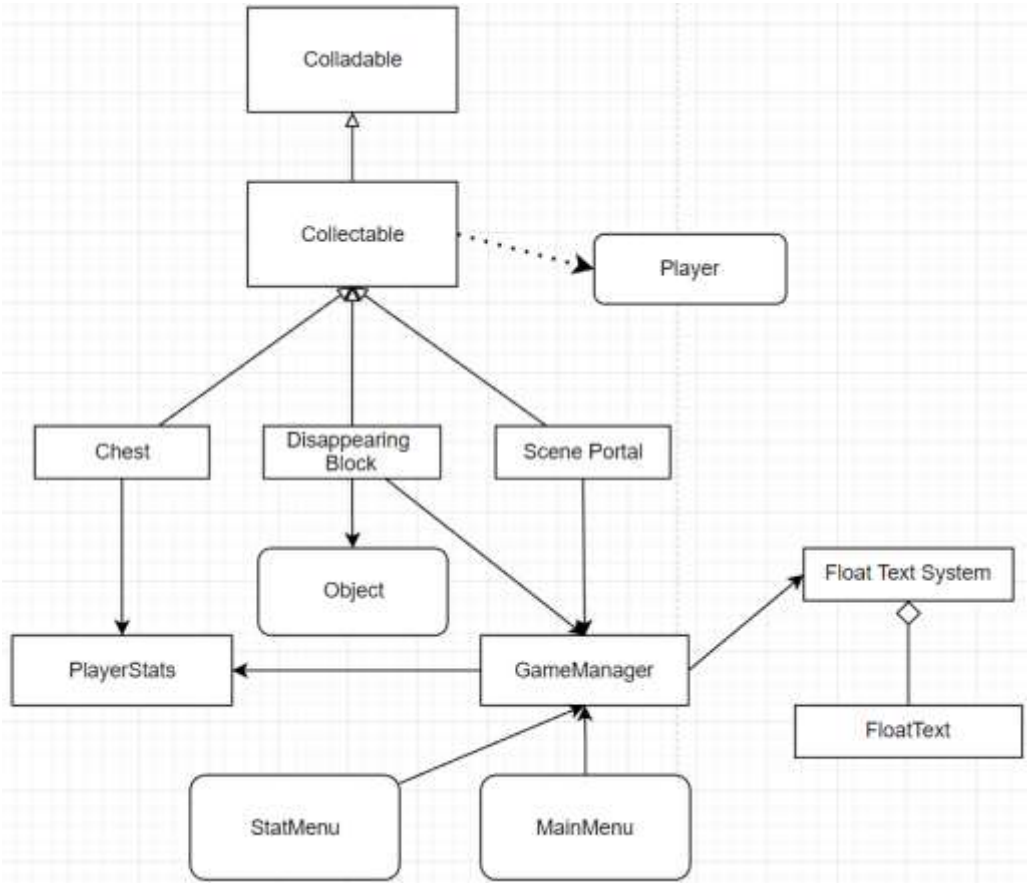

Рисунок 2 – Реализация множества компонентов в разработке

Компоненты, которые пишет программист, называются скриптами. Помимо скриптов, Unity обладает собственной огромной библиотекой компонентов, которые облегчают создание многих объектов. Они разделяются по назначению: так, существуют отдельные компоненты для 2D или 3D-разработки. Помимо стандартных компонентов, облегчающих разработку, существуют те, которые напрямую взаимодействуют с движком. Например, компонент Animator [6].

Перспективы КОП

Перспективы такого подхода вытекают из его преимуществ.

Одной из главных перспектив КОП является возможность повторного использования компонентов. Это позволяет сократить время на разработку новых проектов и снизить затраты на создание новых функций. Кроме того, использование готовых компонентов позволяет разработчикам быстрее освоиться в новой системе и начать работать над своими задачами.

Способность компонентов к быстрому обновлению хорошо подходит к разному роду приложений, но лучшими среди них будут корпоративные. За счет длительной разработки и постоянного наращивания функционала КОП показывает себя максимально эффективно. Так, новые компоненты могут обогащать функционал, добавляя необходимые возможности и интеграцию с другими системами и приложениями [7].

Примером может служить использование готовых компонентов в системах управления базами данных. Вместо того, чтобы разрабатывать новую систему с нуля, компания может использовать существующие компоненты и адаптировать их под свои нужды, что позволит без дополнительных расходов открыть новый филиал или расширить сферу влияния на другой рынок [8].

Другой важной перспективой КОП является повышение производительности системы. Благодаря разделению границ между функционалом можно создавать все более корректную систему. Этим уже давно пользуется разработка игр в своих движках. Данный тренд задал движок Unity Engine. Unity развивали эту идею в собственном движке путем создания собственных компонентов для упрощения разработки и более тесного взаимодействия между движком и играми. Теперь же у Unity есть собственная документация, связанная с компонентами, способом их написания. Все опции мануала представлены на рисунке 3.

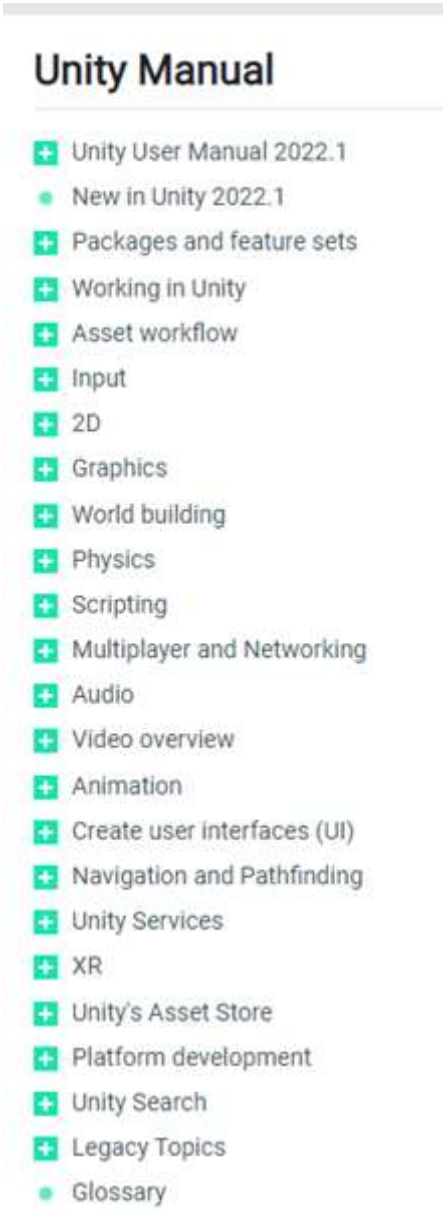

Рисунок 3 – Мануал компонентов Unity

У КОП есть существуют перспективы в области безопасности благодаря разделению системы на более мелкие и более управляемые компоненты. Это может помочь уменьшить вероятность возникновения уязвимостей в системе. КОП может помочь в создании более защищенных систем, поскольку он позволяет лучше контролировать взаимодействие между компонентами и снижает вероятность проникновения злоумышленников [9].

Это далеко не все области, в которых КОП имеет перспективы и возможности. Поскольку это пересмотренный подход к программированию, он все больше и больше находит возможности реализации и упрощения разработки, что уже указывает на огромные амбиции подхода [10].

Заключение

В заключении стоит сделать вывод, что компонентно-ориентированный подход является следствием развития концепций программирования, которые улучшают как собственный подход, так и направление в общем. В результате образовался перспективный подход, ориентирующийся в первую очередь на слабую связь между отдельными компонентами, из чего образовались и другие важные преимущества, позволяющие повторно, без каких-либо усилий, использовать уже написанные и проверенные компоненты; упростить разработку схожих продуктов; рассматривать объекты как предмет определенного функционала; уменьшать потребление памяти за счет слабой связи между компонентами; облегчить обучение и поддержку системы, а так же снизить порог входа новым сотрудникам. Одним из верных способов интегрирования данной концепции в свой продукт, является правильное понимание необходимости такой парадигмы. Многие проекты, способные развиваться, возможно, уже достигли такого этапа, из-за чего не стоит вводить подобные изменения, ведь это приведет к краху продукта. Концепция до сих пор развивается, образуя новые методики разработки, исправляя свои недостатки, постоянно изменяясь и обрастая новыми особенностями.

Тем не менее, нельзя считать, что КОП – панацея программирования. Как и все методы программирования, необходимо подстраиваться под конкретный проект выбирая лучший способ решения поставленных задач для итогового продукта. Стоит отметить, что данный подход можно комбинировать с другими, что позволит развить направленность проекта.

Литература

1. Николаенко, Д. В. Программная реализация функциональной части методов классов в объектной модели интеллектуальной системы управления транспортом // Современные тенденции развития и перспективы внедрения инновационных технологий в машиностроении, образовании и экономике, 2016. - Т. 1. - № 1. - С. 151-154.

2. Вирт, Н. Программирование на языке Modula-2 / Н. Вирт. – перевод с английского. – Москва: Мир, 1987. – 224 с.

3. Введение в компонентно-ориентированный подход к программированию habr / Интернет-ресурс. - Режим доступа: https://habr.com/ru/articles/243479/

4. Медведев, В. И. NET компонентно-ориентированное программирование / В. И. Медведев. - 2-е издание. – Казань: Республиканский

центр мониторинга качества образования, 2013. – 248 с.

5. Тепляков, С. Паттерны проектирования на платформе .NET / С. Тепляков. – СПБ: Питер, 2015. - 320 с.

6. Backer, J. W. Software systems: principles of design and construction. / J. W. Backer. - England : Pearson Education Limited, 2013. – 763 с.

7. Гамма, Э. Паттерны объектно-ориентированного проектирования / Э. Гамма, Р. Хелм, Р. Джонсон, Дж. Влиссидес. - СПб.: Питер, 2020. — 448 с.

8. Heinman, G. T. Component-based software engineering : putting the pieces together / G. T. Heinman, W. T. Concill. - Boston : Addison-Wesley, 2001. – 818 с.

9. Эванс, Э. Предметно-ориентированное проектирование (DDD): структуризация сложных программных систем. / Э. Эванс. - Москва : 000 "И.Д. Вильямс", 2011. - 448 с.

10. Зыков, С. В. Программирование: учебник и практикум для академического бакалавриата / С. В. Зыков. — 2-е издание. — Москва : Издательство Юрайт, 2023. — 285 с.

Павлов М. Ю., Боднар А. В. Современные перспективы компонентно-ориентированного подхода. *В работе рассматривается компонентно-ориентированный подход, его первостепенные задачи и философия, выделяющая данную парадигму среди других. Основная часть работы посвящена описанию концепции подхода, рассмотрению преимуществ, выделение отличительных черт разработки, проанализированы сложности при введении концепции, оценены недостатки, трудность осуществления подхода, найден проблемы в интегрировании парадигмы, рассмотрена реализация и перспективы дальнейшей разработки рассматриваемого подхода.*

Ключевые слова: *компонентно-ориентированный подход, компоненты, разработка, проектирование, парадигма, система.*

Pavlov M., Bodnar A. Modern perspectives of the component-oriented approach. *The paper considers the component-oriented approach, its primary tasks and the philosophy that distinguishes this paradigm from others. The main part of the work is devoted to describing the concept of the approach, considering the advantages, highlighting the distinctive features of the development, analyzing the difficulties in introducing the concept, assessing the disadvantages, the difficulty of implementing the approach, finding problems in integrating the paradigm, considering the implementation and prospects for further development of the approach in question.*

Keywords: *component-oriented approach, components, development, design, paradigm, system*.

Статья поступила в редакцию 05.10.2023
Рекомендована к публикации профессором Мальчевой Р. В.