DOI: 10.15514/ISPRAS-2019-31(5)-8

DOOR: Подход к реструктуризации распределенных объектно-ориентированных систем на основе нейронных сетей

А. Хан, ORCID: 0000-0003-4355-484X <soft_engr_isb@yahoo.com>
Университет COMSATS Исламабад,
Пакистан, 45550, Исламабад, Парк Роад, Тарлай Калан

Аннотация. Для развития распространенных программных систем инженерами и дизайнерами в настоящее время используется объектно-ориентированный подход, который помогает строить распределенные объектно-ориентированные системы. Отличительной особенностью распределенных объектно-ориентированных систем является компетентное распределение классов объектов по различным узлам. Обычно информация о распределении классов по серверам в приложениях отсутствует, что стимулирует проведение процедуры реструктуризации, позволяющей повысить производительность. В предлагаемом подходе реструктуризация программного обеспечения систем осуществляется с помощью адаптивного метода с использованием нейронной сети. На начальном этапе создается граф зависимости классов, узлы в котором представляют классы, а связи между узлами соответствуют зависимостям между классами. Затем извлекаются свойства классов, входящих в этот граф, которые передаются в качестве входных данных в нейронную сеть для ее обучения. После этого на основе учета зависимостей классов выполняется кластеризация, приводящая к разбиению множества классов распределенной объектно-ориентированной системы на слабосвязанные подмножества. Далее создается граф кластеров, вершины в котором соответствуют кластерам, а ребра – каналам связи, которые могут существовать между кластерами. К полученному графу применяется алгоритм k-medoids, который используется для сбора кластеров таким образом, что количество собранных групп кластеров становится равным количеству доступных узлов системы. Сформированные в результате группы кластеров оказываются слабосвязанными. В завершение группы кластеров приписываются различным доступным узлам в распределенной среде. Результаты моделирования показали, что предлагаемая работа дает более эффективные результаты по сравнению с существующими методами.

Ключевые слова: распределенные объектно-ориентированные системы; диаграмма зависимости классов; рекурсивная кластеризация графов; слабое связывание; нейронная сеть; распределенная архитектура

Для цитирования: Хан А. DOOR: Подход к реструктуризации распределенных объектно-ориентированных систем на основе нейронных сетей. Труды ИСП РАН, том 31, вып. 5, 2019 г., стр. 109–126. DOI: 10.15514/ISPRAS–2019–31(5)–8

DOOR: Distributed Object Oriented Software Restructuring Approach Using Neural Network

A. Khan, ORCID: 0000-0003-4355-484X <soft_engr_isb@yahoo.com>
COMSATS University Islamabad,
Park Road, Tarlai Kalan, Islamabad, 45550, Pakistan

Abstract: To develop common software systems, engineers and designers are currently using an object-oriented approach that helps build distributed object-oriented systems. A distinctive feature of distributed object-oriented systems is the distribution of classes of objects among various nodes. Typically, there is no information about the distribution of classes across servers in applications, which encourages a restructuring procedure that improves productivity. In the proposed approach, software restructuring of distributed object-oriented systems is carried out using the adaptive method using a neural network. At the initial stage, a graph of class dependencies is created, in which nodes represent classes, and relations between nodes correspond to dependencies between classes. Then, the properties of the classes included in this graph are extracted, which are transmitted as input to the neural network for its training. After that, based on the account of class dependencies, clustering is performed, leading to the partition of the set of classes of the distributed object-oriented system into loosely coupled subsets. Next, a graph of clusters is created, the vertices in which correspond to clusters, and the edges correspond to communication channels that may exist between clusters. The k-medoids algorithm is applied to the resulting graph, which is used to collect the clusters in such a way that the number of collected cluster groups becomes equal to the number of available system nodes. The resulting cluster groups are loosely coupled. Finally, cluster groups are assigned to various available nodes in a distributed environment. The simulation results showed that the proposed work gives more effective results compared to existing methods.

Keywords: distributed object oriented systems; class dependency graph; recursive graph clustering; low coupling; neural network; distributed architecture

For citation: Khan A. DOOR: Distributed Object Oriented Software Restructuring Approach Using Neural Network. Trudy ISP RAN/Proc. ISP RAS, vol. 31, issue 5, 2019 г., pp. 109-126 (in Russian). DOI: 10.15514/ISPRAS–2019–31(5)–8

1. Введение

С развитием технологии использование новейших программных приложений существенно способствовало и научному прогрессу, и экономическому росту. К сожалению, одновременно растут потери из-за намеренных или ненамеренных погрешностей в программном обеспечении, что ставит перед разработчиками программного обеспечения крайне актуальную задачу повышения качества создаваемых ими программных продуктов [1]. Проверки качества программных продуктов осуществляются на основе анализа исходных кодов программного обеспечения. Повысить качество программ удается совместным анализом качества структуры программ и ее функциональных возможностей. Обеспечение качества программ представляет собой заметную часть процесса разработки программного продукта [2].

Постоянный рост сложности, критической важности и безальтернативности использования программных систем способствует повышенному вниманию к качеству программного обеспечения при его разработке. Чтобы удостовериться в том, что система не имеет ошибок, она должна быть тщательно протестирована, хотя во многих случаях прямое тестирование затруднительно. Это приводит к созданию специальных групп разработчиков, занятых сопровождением программных систем, которые целиком сосредоточены на тестировании программных систем [3]. Рост скорости разработки программных продуктов приводит к серьезным конфигурационным изменениям,

программные функции меняются под влиянием сиюминутных обстоятельств, что приводит к столь же серьезным изменениям самой сути программных продуктов [4].

Конфигурация программного обеспечения наряду с его качеством часто становится жертвой ограниченного бюджета. Использование таких прекрасных языков программирования, как C#, Java и других не может отменить необходимости строить четкое и понятное описание структуры программы [8]. Как правило, методы реструктуризации приводят к улучшению исходного кода, повышая его качество, выявляя неописанные функции, иногда даже до первоначального выпуска продукта, делая его элементы более понятными и более пригодными для повторного использования [5].

Рефакторинг связан с перестройкой программного обеспечения. Этим термином обозначается изменение в программном обеспечении, выполняемое для улучшения его внутреннего качеств без изменения внешнего поведения. Основной целью этого процесса является повышение нефункциональных особенностей кода, некоторых его выбранных качеств, его понятности и гибкости [6].

Рефакторинг и поддержка программ – это два разных процесса. Рефакторинг в основном связан с повышением разумности уже существующего и действующего программного обеспечения. Рефакторинг отличается от сопровождения тем, что сопровождение призвано менять поведение программного обеспечения, то есть выявлять и устранять ошибки, ускорять работу приложения или как-то улучшать функциональность [7]. В то же время, рефакторинг программного обеспечения улучшает внутреннее качество программного обеспечения, не влияя на его внешнее поведение. Следовательно, он является эффективным и сравнительно безопасным способом улучшения качества программного обеспечения [9].

Совершенствование программной среды представляет собой длительный и непрерывный процесс. Программная среда, как правило, в течение некоторого времени претерпевает последовательность как небольших, так и больших изменений [10]. Чтобы найти, а потом и усовершенствовать методику разработки программного обеспечения и разрабатываемые программные продукты, были созданы и утверждены стандарты разработки программного обеспечения. Среди предложенных моделей в течение уже многих лет заметно выделяется объектно-ориентированный подход [11]. Эта парадигма рассматривает программу как совокупность взаимодействующих объектов, каждый из которых скорее заботится о хранении своего индивидуального частного состояния, чем о предоставлении средств хранения глобального состояния. Основу парадигмы объектно-ориентированного программирования составляют понятия динамического связывания, инкапсуляции, наследования и полиморфизма. Объекты стало возможность распределять по узлам системы, где потом они могли обрабатываться либо параллельно, либо последовательно, в зависимости от конкретики объектно-ориентированного приложения [12].

Революционная парадигма программного обеспечения, называемая объектно-ориентированным программированием (ООП), представляет собой метамодель, построенную на использовании понятия о классах и об экземплярах этих классов на основе теории абстракции, и механизмов инкапсуляции, наследования, полиморфизма и других. Эти теоретические построения способствуют созданию повторно используемых и способных к развитию программных модулей [13]. Объектно-ориентированный подход, первоначально возникший как парадигма программирования, в настоящее время используется специалистами и проектировщиками для решения сложных задач во многих научных областях.

Основным этапом моделирования распределенной объектно-ориентированной методологии (РОО) является выявление области видимости объекта. Этот этап уменьшает потребность в установлении связей с объектом в тех местах, где он недоступен [14].

Приложение РОО создается с использованием различных программных платформ, которые обеспечивают такие свойства, как мобильность, соответствие требованиям и способность к взаимодействию. На этих стадиях проявляются базовые свойства программных платформ и их конфигураций, которые влияют на архитектуру и полезность создаваемых систем [15]. Первые возникающие варианты приложения РОО, безусловно, не обладают достаточным качеством. Чтобы его добиться, можно действовать двояким образом: для взаимодействия с программными компонентами можно реконфигурировать оборудование, но чаще для взаимодействия с аппаратурой изменяется структура программ [14].

Настоящая работа построена следующим образом: в разд. 2 рассматриваются родственные работы, касающиеся предлагаемого нами метода, в разд. 3 кратко обсуждается предлагаемая методика, разд. 4 посвящен оценке результатов исследования, в разд. 5 обобщаются результаты, описанные в статье.

2. Обзор литературы

В работе Айхуа Гу и соавторов [16] предлагается использовать метрику связности сложных сетей (Cohesion Based on Complex Networks, CBCN), разработанную главным образом на основе расчета коэффициента средней кластеризации класса (Class Average Clustering, CAC) с помощью диаграмм, демонстрирующих причины объединения в кластере нескольких классов. Кроме того, метрика CBCN была оценена на гипотетическом примере сравнения с метрикой объединения классов (Class Union Measurement, CCM) по четырем свойствам (связывающие модули, неотрицательность и нормализация, монотонность, пустые и максимальные значения). Показатели метрики CBCN, основанные на корреляции информации с семнадцатью наилучшими из типичных CCM систем объединения классов, оказались наилучшими для всех остальных. Применение метрики CBCN к трем программам с открытым исходным кодом вычисления коэффициентов CAC, показало, что класс, требовавший модификации, усложнения и поддержки в системе с открытыми кодами, был протестирован хуже остальных. У упомянутых систем имелось степенное распределение коэффициентов CAC, что обеспечило дальнейшее понимание CBCN.

В работе Немитари Ажиенка с соавторами [17] была сделана попытка связать эффективность измерения семантического связывания (Semantic Coupling, SC) классов объектно-ориентированных программ, используя 1) простые методы, ориентированные на идентификаторы, и 2) полные корпуса классов в программной системе. Впоследствии они исследовали взаимодействие между семантическим и изменяемым связыванием (Change Coupling, CC). Что касается отношений между логическими и семантическими зависимостями, то в 79 объектно-ориентированных проектах и проектах с открытым кодом не было выявлено никакой связи между логическими и семантическими условиями и качеством программных продуктов. Тем не менее, они признали взаимовлияние семантических и логических условий. Было отмечено, что более семидесяти процентов семантически связанных классов регулярно развивались, а классы, связи которых, хотя бы в некоторой степени, менялись, обычно обладали семантическими связями. Результаты показали, что: (a) методы, ориентированные на использование идентификаторов классов, более эффективны, хотя и не во всех случаях (только при рассмотрении семантически очень похожих классов), (b) между семантическими и логическими зависимостями не было никакой линейной связи. Кроме того, они обнаружили, что (c) между ними существуют взаимоотношения, так как более семидесяти процентов семантически связанных классов обычно развиваются совместно, а классы, связанные через CC, обычно, хотя бы немного, связаны и семантически.

В работе Нимиша Кумара и соавторов [18] представлена общая концепция метода управления качеством многочисленных свойств объектно-ориентированной структуры, который логически рассматривается как процедуры качества с вложенными в них процедурами, зависящие от перспектив развития сетевых приложений. Ключевым посылом выработки этих рекомендаций было изучение и распространение ранее разработанных процедур управления качеством на сетевые приложения. Этот метод был выбран из-за не всегда продуманного и очень разного характера сетевых приложений. Для управления структурой программной системы использовались рекомендации стандарта ИСО/МЭК 9126. Универсальность метода доказывается тем, что программирование обычных и сетевых приложений предлагается вести сходными средствами. При этом, конечно же, в зависимости от вида приложения (сетевое или обычное) можно добавлять, изменять или удалять различные его свойства. Еще больше повышает точность измерения показателя качества исследуемой структуры увеличение глубины иерархии и количества узлов в ней (сочетанием параметров качества с процедурами, вложенными процедурами, показателями и их отдельными частями). Таким образом, полученная модель оказалась достаточно гибкой и пригодной для использования со всеми программными приложениями и фундаментальными технологиями.

Аншу Парашар и Джитендер Кумар Чхабра [19] представили оригинальные оценки, предназначенные специально для классов: набор изменений-воздействий классов и индекс изменяемого связывания (CC). Сначала эти оценки используются для вычисления индекса CC классов, затем, рассчитываются показатели для классов. Предложенная структура прогнозирования изменчивости давала разработчикам большое количество различной информации, в частности, прогнозируя уровень изменчивости связывания класса. Полезность подхода была подтверждена путем выявления структурных особенностей классов и вычисления индекса CC, а также проведением достаточно точных замеров влияния измененных воздействий на изменения в классах. Предложенные меры были подтверждены эмпирически, были даны ответы на конкретные запросы исследователей, что подтвердило разумность потока изменений. Полученные результаты оказались достаточно надежными, они указывали на то, что предлагаемый прогноз потока потенциальных изменений крайне полезен при поддержке программных систем. Была подтверждена достоверность введенных мер изменчивости, что явилось реакцией на другие запросы, возникающие в ходе конкретных исследований. В частности, потоки информации об изменениях, зафиксированные как история изменений классов, могут быть задействованы для обнаружения изменяемого связывания, что помогает визуализировать предполагаемые архитектурные изменения.

Дарио Ди Нуччи с соавторами [20] исследовали программные компоненты, подвергавшиеся изменениям как со стороны специально выделенных для сопровождения разработчиков, так и со стороны других разработчиков, вносивших изменения в программы. В этой статье описано и другое исследование, в котором изучались ожидания разработчиков в отношении возможных видов ошибок, в частности, они выделяли 2 подхода i) упор на количество фрагментов кода, как основу для поиска ошибок (разработчик вносит изменения в те фрагменты, которые используются в большем числе распределенных узлов) ii) упор на количество соглашений, которые должны соблюдаться в тех или иных узлах (семантическое рассеяние). Указанные подходы, которые были названы предсказателями ошибок, были использованы в эксперименте с привлечением двадцати шести программных систем с открытым исходным кодом. Достигнутые результаты подтвердили действенность модели авторов, они также показали направления расширения и дополнения использованной стратегии. Была получена и затем протестирована на одиннадцати прогнозах гибридная предсказательная модель и пять альтернативных методов. Полученные результаты показали, что (а) гибридная модель продемонстрировала более высокую точность по сравнению с каждой отдельно взятой из

пяти стратегий; (б) разработанные авторами предсказатели ошибок вносят решающий вклад в самые эффективные «гибридные» предсказательные модели.

Вакар Аслам и Фара Иджаз [21] рекомендовали методологию, названную ими распределением задач (task allocation). Работа по этой методологии выполняется в 2 этапа: на этапе 1 определяются условия и перспективы, влияющие на приписку, на этапе 2 предлагается поддающаяся измерениям система, раздающая задания ближайшим коллегам пользователя наилучшим образом. Реквизиты задач формулируются как возможности, рассматриваемые в самых разных аспектах, учитывающих, например, специальные индивидуальные условия. Одним из важных решений считается распределение задач между членами группы разработчиков. Это решение обычно принимается субъективно, а не детерминированными средствами, поэтому связанные с ним риски могут сказываться на результатах работ, выполняющихся впоследствии. Предлагаемый авторами метод учитывает потребности каждой отдельной работы из тех, которые надо будет выполнять при реализации проекта. Для всех и каждой отдельной работы из членов команды выбирается наилучший исполнитель, причем выбор основывается на знании длительности аналогичных работ, на знании методов оценки окончания сравнимых работ, а также на предположениях об относительной безошибочности результата. Авторский метод напоминает методику фокусирования на целях и наилучшего отождествления. Другие предложения авторов приводят к улучшению соотношения цена-качество, а также к достижению многих других целей. Все это позволяет проводить апостериорную оценку качества распределения задач по разработчикам и, в конечном итоге, снижать опасность от связанных рисков.

3. Реструктуризация распределенных объектно-ориентированных программ с использованием нейронных сетей

Методы объектно-ориентированного программирования (ООП), предоставляя передовую платформу для разработки программного обеспечения, позволяют легко и просто создавать программные приложения. На основе подходов распределенных объектно-ориентированных (РОО) систем созданы многочисленные прикладные системы, позволяющие решать сложные задачи в различных научных областях. Центральным аспектом РОО систем является организация адекватного распределения классов, определенных в программе, по разным структурным узлам, то есть снижение риска возникновения несоответствия, которое может возникнуть, если структура программного обеспечения не соответствует структурной организации используемого оборудования.

Для повышения производительности РОО системы предлагается использовать адаптивный метод, использующий возможности нейронных сетей (НС), на основе которого может осуществляться реструктуризация программного обеспечения. Применение этого метода позволяет уменьшить общее число кластеров классов, которые изначально создаются при обучении нейронной сети, их уменьшение соответственно уменьшает потребность в выделенных вычислительных ресурсах.

Первым шагом предложенного метода является создание графа зависимостей классов (ГЗК), в котором узлы соответствуют классам, а связи между ними представляют зависимости между классами. После этого компоненты объектов, методов, переменных, файлов включения и объема кода, связанные с классами в ГЗК, извлекаются из их определений и передаются в качестве входных данных в нейронную сеть как исходные данные для ее обучения. Далее выполняется кластеризация, с помощью которой ОО-система сегментируется на слабосвязанные подсистемы с использованием метода кластеризации на основе зависимостей классов (Class Dependency Based Clustering, CDBP). Объединенные в кластеры классы включаются в кластерные графы с использованием техники k-medoids, и, наконец, используя рекурсивную кластеризацию k-

средних, созданные кластерные структуры сопоставляются с фиксированной конфигурацией доступных машин в целевой распределенной архитектуре. Структура предложенной методологии поясняется на рис. 1.

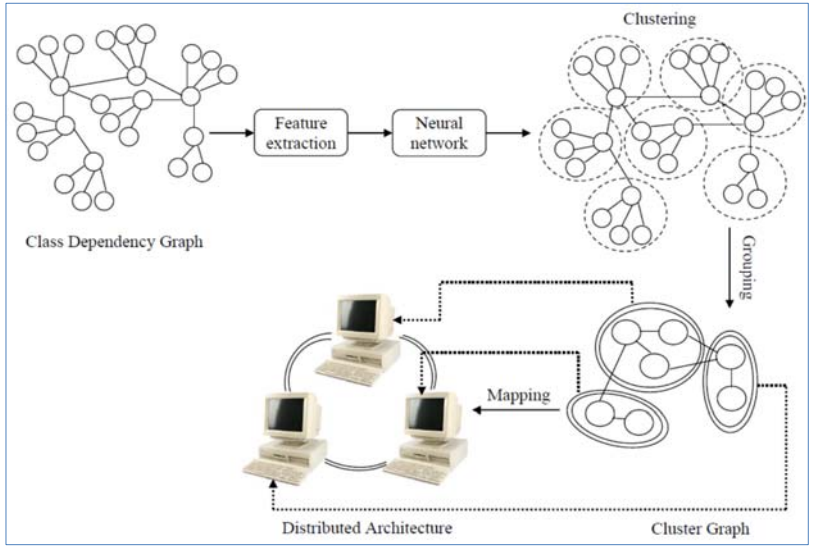

Рис. 1. Предлагаемый подход
Fig. 1. The Proposed Methodology

3.1 Граф зависимостей классов

Процесс первичной оценки связи между классами в РОО системе заключается в приписке каждого отдельного класса программы некоторому узлу распределенной системы. В совокупности связи между классами, размещенными в тех или иных узлах, составляют граф зависимостей классов объектно-ориентированной системы. Пример ГЗК показан на рис. 2.

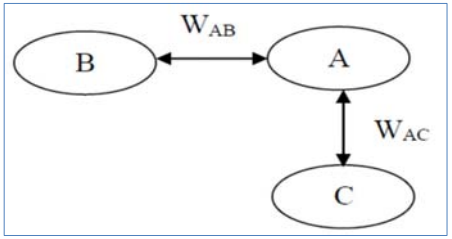

Рис. 2. Граф зависимостей классов.
Fig. 2. Class Dependency Graph

Каждая вершина графа ГЗК обозначает некоторый класс, а ребра на графе между классами A и B соответствуют наличию связи, образующей либо при передаче данных между объектами классов, либо при наличии некоторого отношения между классами. Вес ребра W_{AB} вычисляется как стоимость коммуникационных событий между классом A и классом B. Если между классами нет выявленных зависимостей или отношений, и между ними не никогда не осуществляется передача данных, на ГЗК между классами никаких ребер не возникает.

3.2 Извлечение свойств

После создания ГЗК на основе анализа текста программы выявляются свойства объектов, переменных, методов, файлов включения и объема программы, относящихся к выделенным классам в ГЗК.

- Объект: Объект выступает как центральная единица ООП. Обычная Java-программа содержит множество определений взаимосвязанных объектов, взаимодействующих друг с другом посредством определенных в программе методов классов. Каждый объект характеризуется состоянием, поведением и идентичностью.
- Переменные: Объектно-ориентированная парадигма, основанная на классах и объектах, как экземплярах классов, допускает наличие в классах «статических» переменных, которые существуют в единственном для всех объектов класса экземпляре и никогда не являются принадлежностью какого-либо одного из объектов. Такая переменная выступает в качестве отдельной формы атрибута класса (поля данных или, иначе, свойства класса, его информационной части).
- Метод: Процедура, которая является принадлежностью объекта и может генерировать сообщения, называется в объектно-ориентированной парадигме методом. На самом деле это фрагмент программы, который содержит последовательность операторов, выполняющих задачу. Методы обладают интерфейсом, который используется другими классами для изменения и доступа к свойствам данных объекта.
- Файлы включения: Файлы импорта используются для передачи пользовательских и встроенных системных определений в исходный файл Java, чтобы отдельный класс мог обращаться к классу другого кластера, напрямую используя его имя.
- Число строк: общее количество строк кода, содержащихся в программе.

3.3 Нейронная сеть

Выявленные на предыдущем этапе элементы передаются нейронной сети в качестве входных данных процесса обучения. Обучаясь, нейронная сеть корректирует собственные веса, базируясь на наборах входных данных и выходные с предсказанными целевыми значениями.

Веса синапсов постепенно меняют свои значения, пока сеть не стабилизируется. Каждая итерация выполняется в двух направлениях: прямая передача сигнала для достижения решения, а также обратное распространение ошибок, выполняемое для преобразования весов. Такие прямые и обратные передачи сигналов выполняются до тех пор, пока решение состояние нейронной сети не будет признано приемлемым с учетом ранее установленного допуска. Структура нейронной сети показана на рис. 3. Действия по обратному распространению ошибки в процессе обучения сети показаны на следующих рисунках.

Шаг 1: Выбрать случайные веса в интервале [0, 1] и передать их на выход, а также на нейроны скрытого слоя. Передать веса на все нейроны информационного слоя.

Шаг 2: Классификатор сам выбирает набор обучающих данных в качестве входных, ошибка оценивается как

$$BP_{err} = Z_{tar} - Z_{out}. \quad (1)$$

В формуле (1) Z_{tar} является эталонным значением выхода, реальным выходом сети является значение Z_{out}, которое определяется как $Z_{out} = [Z_2^{(1)}\ Z_2^{(2)}\ \dots\ Z_2^{(N)}]$. Выходы сети $Z_2^{(1)}, Z_2^{(2)}, \dots, Z_2^{(N)}$ представляются как

$$Z_2^{(l)} = \sum_{r=1}^{N} W_{2rl} Z_l(r), \quad (2)$$

где

$$Z1(r) = \frac{1}{1+\exp(-W_{1lr}\, Z_{ln})}. \quad (3)$$

Формулы (2) и (3) вводят функции активации, реализованные в выходном и в скрытом слоях соответственно.

Шаг 3: Изменить все веса нейронов на величину ΔW, где ΔW вычисляется как

$$\Delta W = \gamma \times X2 \times BP_{err} \quad (4)$$

В формуле (4) величина γ обозначает скорость обучения, обычно она находится в диапазоне от 0.2 до 0.5.

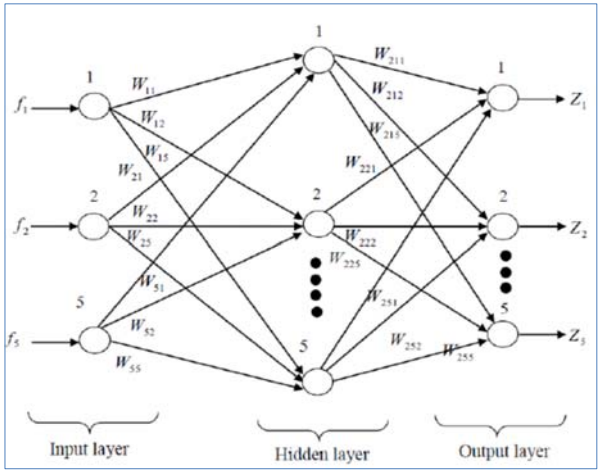

Рис. 3. Структура нейронной сети
Fig. 3. Structure of Neuron Network

Шаг 4: Повторять описанный процесс, начиная с шага 2, пока ошибка BP_{err} не будет уменьшена до минимального значения. Обычно в качестве критерия используется соотношение $BP_{err} < 0.1$.

Уточненные в процессе обучения функции, использовавшиеся для кластеризации, уменьшают начальную совокупность кластеров нейронной сети и, таким образом, уменьшают совокупность выделенных ресурсов.

3.4 Кластеризация системы классов

После завершения этапа обучения сети с помощью алгоритма CDBP выполняется разделение программной системы на слабосвязанные подсистемы. На рис. 4 показан псевдокод кластеризации на основе зависимостей классов. С точки зрения языка программирования Java кластер представляет собой пространство имен, в котором собраны близкие друг к другу классы и их интерфейсы. До некоторой степени кластеры напоминают специальные папки, размещенные в архиве персонального компьютера. Наличие кластеров стимулирует разработчиков группировать классы (а также интерфейсы) вместе. Все эти классы будут каким-то образом связаны, так что все они могут понадобиться при решении определенного набора задач. Поскольку целью является объединение зависимых классов, кластеризация с использованием алгоритма CDBC, показанная на рис. 4, может оказаться весьма полезной. Реализация алгоритма CDBC разбивает программу на несколько кластеров, в которые входят взаимодействующие и

схожие классы. Сначала производится идентификация классов, а затем формируются кластера.

Идентификация классов объектно-ориентированных программ: Перед выполнением процедуры кластеризации производится текстовый анализ программной системы и выделяются все использованные в программе классы. В нашем случае использовался программный продукт, созданный на языке Java, поэтому текстовый анализ выполнялся над файлами с расширением java.

Формирование кластера: После распознавания классов объектов алгоритм CDBC способен выявлять группы зависимых классов. Для очередного класса проверяется, содержатся ли в исходном коде операции создания объектов этого класса. Если да, класс помещается в кластер в соответствии со своими зависимостями. Если условие не выполняется, происходит переход к анализу следующего класса. Пополнение кластеров продолжается до тех пор, пока каждый класс, используемый в анализируемой программе, не будет отнесен к тому или иному кластеру.

```
Requirement: PackageContainer=0, PackageName=0,
ProgramClassList=0, NumOfObjects=0, NumOfImports=0

For each PackageContainer do
  For each File do
      If fileNameExtension= .Java then
        Add File to ProgramClassList
      End if
  End for
End for

For each ProgramClassList do
  Read Program File
  Search each class name in Header && SearchObject for ClassHeader
      If Class contains ClassHeader and its Object then
        Group class under the Dependency package
      Else
        Search in another ProgramClassList
      End if
End for

For each Package in PackageContainer do
  If NumOfClassInPackage<NumOfObjects then
      Add to Clusters
  End if
End for
```

Рис. 4. Псевдокод кластеризации на основе зависимости классов
Fig. 4. Pseudocode for Class Dependency Based Clustering

3.5 Группировка кластеров

Кластеры классов, созданные на предшествующем этапе, используются в дальнейшем в качестве кандидатов для распределения по узлам системы. Предлагаемый метод заключается в объединении кластеров в группы, что обеспечивает снижение затрат на организацию связей между ними. Чтобы добиться поставленной цели, создается граф кластеров, вершины в котором соответствуют кластерам, а ребра – каналам связи, которые могут существовать между кластерами. К полученному графу применяется алгоритм k-medoids, который используется для сбора кластеров таким образом, что количество собранных групп кластеров становится равным количеству доступных узлов системы. Сформированные в результате группы кластеров оказываются слабосвязанными. В

завершение группы кластеров приписываются различным доступным узлам в распределенной среде.

3.5.1 Алгоритм k-medoids

Группировка применяется для объединения кластеров классов с целью собрать в одной группе кластеры, классы которых некоторым образом более похожи друг на друга, чем на классы из других кластеров. Для группировки кластеров используется стратегия кластеризации k-medoids. Медоид – это элемент кластера, отличие которого от других элементов этого же кластера незначительно. Тем самым, медоиды, атрибуты которых сильно отличаются от средних значений, должны исключаться из наборов данных, в которых остаются только элементы, наиболее сильно приближенные к средним значениям. Псевдокод для алгоритма кластеризации k-medoids показан на рис. 5.

Input:
K, Number of cluster groups
Output:
K Set of Cluster Groups

Begin
Arbitrarily choose K objects as the initial representative objects
Repeat
- Assign each remaining object to the cluster with the nearest representative object.
- Randomly select a non-representative object o_{rand}
- Compute total cost S of swapping representative object o_j with o_{rand}

If $S<0$ **then**
- Swap o_j with o_{rand} to form the new set of K representative objects

Until No change
End

Рис. 5. Псевдокод алгоритма k-medoids
Fig. 5. Pseudocode for k-medoids Algorithm

3.6 Распределение классов по узлам сети

Построенные группы кластеров, используемых в прикладной ООО-системе классов теперь надо сопоставить с узлами серверной сети, что должно способствовать достижению более высокой производительности. Процедура сопоставления пишется таким образом, чтобы максимально ограничить зависимость классов от классов других групп и снизить любой обмен информацией между ними. Обычно распределенная система, для которой выполняется такая работа, однородна, то есть состоит из одинаковых процессоров, взаимодействующих в сети на основе единых принципов обмена данными. В предлагаемой стратегии сопоставление классов и узлов серверной сети выполняется на основе алгоритма рекурсивной кластеризации k-средних.

3.6.1 Рекурсивный алгоритм кластеризации k-средних

Предложена новая итеративная стратегия, использующая алгоритм k-средних и рекурсивно строящая последовательности кластеров. Цель этих действий заключается в том, чтобы получить решение k-средних для полного набора данных путем рекурсивной актуализации взвешенной формы k-средних над растущим, но все же небольшим числом

представителей кластера. Псевдокод для алгоритма рекурсивной кластеризации k-средних показан на рис. 6.

На начальном этапе итерационного процесса набор данных разбивается на различные непересекающиеся подмножества, называемые блоками, которые собраны в гиперкубах одинакового размера. Для каждого блока затем создается представитель и вычисляется характеристика, называемая весом. Наконец, над группой представителей выполняется адаптированный вариант алгоритма Ллойда с весами. Шаг за шагом, начиная с полученного разбиения, создаются уточнения, в которых каждый гиперкуб заменяется на другой гиперкуб с тем же весом.

Выполнение алгоритма позволяет заменять подмножества кластеров предыдущей итерации другими кластерами так, чтобы общая ошибка уменьшалась. Этот процесс повторяется до тех пор, пока не будет выполнено заранее заданное число шагов, либо пока очередная итерация не будет отличаться от предыдущей слишком незначительно. Выполняемые вычисления полностью следуют варианту алгоритма Ллойда с весами элементов, называемого взвешенным алгоритмом Ллойда, при работе которого для каждой группы представителей определенного сегмента исследуется вес, рассчитываемый для каждой группы в ее совокупности. По завершении работы алгоритма построенные в результате группы кластеров отображаются на доступные серверные узлы.

Input: Number of Clusters K, Integers $\{q_{\min}, q_{\max}\}$, threshold η
Output: Initial set of centroid

Begin
For $q = q_{\min} : q_{\max}$ **do**
- Construct the partition P_q
- Define the weight set W_q
- Update the centroid set approximation $C_q = \{c_j^q\}_{j=1}^{K} : C_q = Weighted\_Lloyd(W_q, K, C_{q-1})$
- Compute the centroid set displacement measure $d(C_{q-1}, C_q) = \max_{j=1,2,\dots,K} \left\| c_j^q - c_j^{q-1} \right\|^2$

If $d(C_{q-1}, C_q) \le \eta$ **then**
Return C_q
End if
End for
Return C_q
End

Рис. 6. Псевдокод рекурсивной k-средней кластеризации
Fig. 6. Pseudocode of Recursive k-means Clustering

4. Обсуждение результатов

В этом разделе будет проведена сравнительная оценка производительности известного метода реструктуризации ООО-системы без нейронной сети (DOOR) и предлагаемого метода, в котором используется нейронная сеть (DOOR_NN). Оценка проводится сравнением затрат на организацию связей по совокупности кластеров. В системе моделирования MATLAB имеется возможность провести симуляцию такой реструктуризации. Симулятор включает дружественный пользовательский интерфейс, позволяющий пользователю указывать системные узлы и ребра, а также строить граф зависимостей.

Табл. 1. Результаты моделирования для предлагаемой распределенной объектно-ориентированной реструктуризации с нейронной сетью и без нее.
Table 1. Simulation Results for the Proposed DOOR with Neural Network and the Existing technique without Neuron Network.

Число классов	Число кластеров		Затраты на взаимодействие					
			Кластеризация		Группирование		Сопоставление с узлами сети	
	DOOR	DOOR_NN	RGC	CDBC	k-Partitioning	k-medoids	Двойная k-кластеризация	Рекурсивный k-средних
51	6	4	2045	1916	1899	1714	1680	1540
62	7	6	2632	2243	2160	2095	2124	2024
79	9	7	2600	2415	2185	2060	2097	1988
107	11	9	3196	2818	3009	2698	2483	2184
153	15	13	4090	3825	3784	3515	3143	3005

В табл. 1 приведены результаты моделирования предложенной распределенной объектно-ориентированной реструктуризации с нейронной сетью (DOOR_NN) и известного метода без нейронной сети (DOOR). Предлагаемый метод DOOR_NN по всем позициям выигрывает у существующей стратегии DOOR, ориентированной на количество кластеров, кластеризацию, группировку и отображение. Для кластеризации в стратегии DOOR используется метод рекурсивной графовой кластеризации (Recursive Graph Clustering, RGC), а предложенный новый подход базируется на CDBC. Для группировки кластеров предложенная стратегия использует подход k-medoids, в то время как существующая процедура использует подход k-секционирования (k-Partitioning). Для отображения кластеров на серверные узлы в предложенной стратегит использовалась рекурсивная кластеризация k-средних, в то время как в существующем методе используется подход двойной k-кластеризации (Double k). Для любого количества классов и кластеров предлагаемая процедура с нейронной сетью демонстрирует превосходство по производительности по сравнению с существующим подходом без нейронной сети.

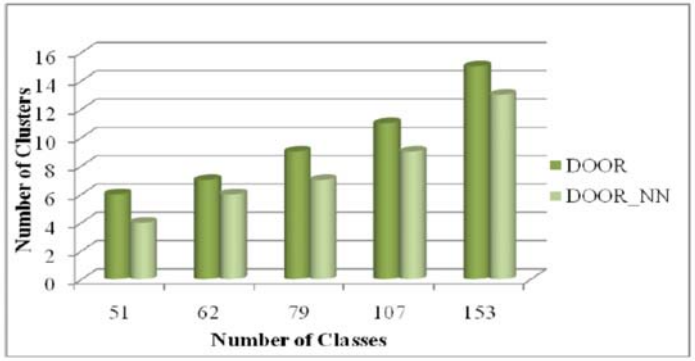

Рис. 7. Анализ производительности предложенного метода в зависимости от числа кластеров
Fig. 7. Performance analysis of the proposed technique based on the number of clusters

На рис. 7 анализируется предлагаемая производительность DOOR_NN и эффективность существующей процедуры DOOR, сконцентрированной на количестве кластеров, сформированных в процессе кластеризации. Он демонстрирует результаты моделирования с четырьмя узлами. Он также показывает совокупное количество кластеров, сформированных для заданного количества классов. Здесь можно безошибочно заметить, что предлагаемый метод DOOR_NN производит меньшее количество кластеров, чем существующая стратегия, что показывает большую эффективность предложенной стратегии.

4.1 Кластеризация

На рис. 8 иллюстрируется сравнение затрат на установление связей между кластерами для предложенной процедуры кластеризации на основе CDBC и для существующего метода RGC. По оси X на рис. 8 отмечается количество построенных кластеров, а по оси Y – затраты на установление связи в единицах времени, учитывающих расположение классов в различных узлах сети. Видно, что для четырех кластеров расходы на связь примерно одинаковы для процедуры RGC и предлагаемого метода CDBC. Однако по мере увеличения количества кластеров расходы на связь увеличиваются, при этом наблюдается различие в стоимости для существующей и предлагаемой стратегий. Предложенный метод CDBC демонстрирует наименьшие затраты при любом количестве кластеров.

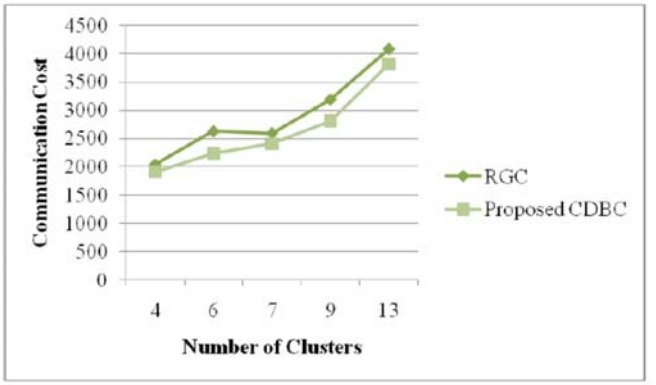

Рис. 8. Анализ эффективности кластеризации в терминах затрат на взаимодействие для известного метода RGC и предлагаемого метода CDBC
Fig. 8. Performance analysis of Clustering in terms of Communication Cost for the existing RGC and the proposed CDBC

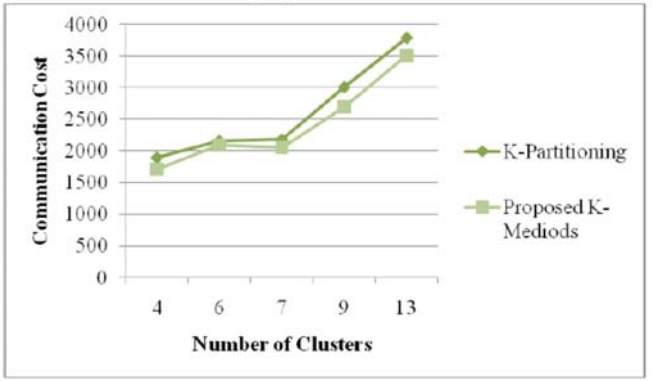

Рис. 9 Анализ эффективности группирования в терминах затрат на взаимодействие для известного метода k-секционирования и предлагаемого метода k-medoids
Fig. 9 Performance analysis of Grouping in terms of Communication Cost for the existing k-Partitioning and the proposed k-medoids

4.2 Группировка

На рис. 9 иллюстрируется сравнение затрат на группирование кластеров для предложенного к использованию метода кластеризации k-medoids и часто используемой

стратегии кластеризации k-секционирования. Для проведения группирования кластеров предлагается использовать метод кластеризации k-medoids. При числе кластеров 6 и 7 затраты на связь примерно одинаковы как для метода k-секционирования, так и для предлагаемого метода k-medoids. Однако сравнение стоимости обоих методов показывает, что предлагаемый метод k-medoids имеет более низкую стоимость установления связей при любом количестве кластеров.

4.3 Отображение

Рис. 10 иллюстрирует сравнение затрат на установление связей при использовании для распределения кластеров по узлам сети предлагаемой системы рекурсивной кластеризации k-средних и часто используемой методики двойной k-кластеризации.

Отображение кластеров на узлы предлагается проводить на основе стратегии рекурсивной кластеризации k-средних. Разница в стоимости связи для предлагаемой и существующей систем особенно хорошо видна, когда количество кластеров равнялось 4, 6 и 7. Наибольшее различие между методами выявлено для девяти кластеров.

Рис. 10. Анализ эффективности отображение в терминах затрат на взаимодействие для известного метода двойной k-кластеризации и предложенного рекурсивного метода k-средних
Fig. 10. Performance analysis of Mapping in terms of Communication Cost for the existing Double k-Clustering and the proposed Recursive k-means

5. Заключение

Использование методов объектно-ориентированного программирования позволяет заметно упростить создание приложений, предоставляя передовую платформу для разработки приложений. Реструктуризация программного обеспечения (РОО-систем) осуществляется с помощью предложенного адаптивного метода с использование нейронной сети, который способен еще больше повысить производительность системы. Благодаря использованию этого метода, можно уменьшить совокупность кластеров, которая первоначально возникает при обучении нейронной сети, и, следовательно, уменьшить совокупность выделенных ресурсов.

Производительность существующего метода реструктуризации распределенной объектно-ориентированной программной системы (распределения кластеров классов системы по узлам сети) без нейронной сети (DOOR) и предлагаемого метода реструктуризации с использованием нейронной сети (DOOR_NN) оценивается с помощью метрик производительности, например, количества кластеров и стоимости

связи. Построение кластеров с использованием нейронной сети значительно уменьшает количество кластеров, генерируемых во время кластеризации, и, следовательно, снижает стоимость связи. Результаты моделирования показали, что предлагаемая работа дает более хорошие результаты по сравнению с существующими методами.

References

[1]. Wang J., Ai J., Yang Y., Su W. Identifying key classes of object-oriented software based on software complex network. In Proc. of the 2nd International Conference on System Reliability and Safety, 2017, pp. 444-449.
[2]. Tarwani S., Chug A. Prioritization of code restructuring for severely affected classes under release time constraints. In Proc. of the 1st India International Conference on Information Processing, 2016, pp. 1-6.
[3]. Boucher A., Badri M. Predicting Fault-Prone Classes in Object-Oriented Software: An Adaptation of an Unsupervised Hybrid SOM Algorithm. In Proc. of the IEEE International Conference on Software Quality, Reliability and Security, 2017, pp. 306-317.
[4]. Rajdev U., Kaur A. Automatic detection of bad smells from excel sheets and refactor for performance improvement. In Proc. of the International Conference on Inventive Computation Technologies, vol. 2, 2016, pp. 1-8.
[5]. Kaya M., Fawcett J.W. A new cohesion metric and restructuring technique for object oriented paradigm. In Proc. of the IEEE 36th Annual Computer Software and Applications Conference Workshops, 2012, pp. 296-301.
[6]. Mourad B., Badri L., Hachemane O., Ouellet A. Exploring the Impact of Clone Refactoring on Test Code Size in Object-Oriented Software. In Proc. of the 16th IEEE International Conference on Machine Learning and Applications, 2017, pp. 586-592.
[7]. Amirat A., Bouchouk A., Yeslem M.O., Gasmallah N. Refactor Software architecture using graph transformation approach. In Proc. of the Second International Conference on Innovative Computing Technology, 2012, pp. 117-122.
[8]. Bhatti M.U., Ducasse S., Huchard M. Reconsidering classes in procedural object-oriented code. In Proc. of the 15th Working Conference on Reverse Engineering, 2008, pp. 257-266.
[9]. Liu H., Li G., Ma Z.Y., Shao W.Z. Conflict-aware schedule of software refactorings. IET software, vol. 2, issue 5, 2008, pp. 446-460.
[10]. Nongpong K. Feature envy factor: A metric for automatic feature envy detection. In Proc. of the 7th International Conference on Knowledge and Smart Technology, 2015, pp. 7-12.
[11]. Tomyim J., Pohthong A. Requirements change management based on object-oriented software engineering with unified modeling language. In Proc. of the 7th IEEE International Conference on Software Engineering and Service Science, 2016, pp. 7-10.
[12]. Hamad S.H., Ammar R.A., Khalifa M.E., Fergany T. Randomized Algorithms for Mapping Clustered Object-Oriented Software onto Distributed Architectures. In Proc. of the 7th IEEE International Symposium on Signal Processing and Information Technology, 2018, pp. 426-431.
[13]. Sugandhi R., Srivastava P., Sanyasi A., Awasthi L.M., Parmar V., Makadia K., Patel I., Shah S. Implementation of object oriented software engineering on LabVIEW graphical design framework for data acquisition in large volume plasma device. In Proc. of the 7th International Conference on Cloud Computing, Data Science & Engineering, 2017, pp. 798-803.
[14]. Faheem M.T., Ammar R.A., Sarhan A.M., Ragab H.A.M. A hybrid algorithm for restructuring distributed Object-oriented software. In Proc. of the International Symposium on Signal Processing and Information Technology, 2010, pp. 202-208.
[15]. Cosma D.C. Reverse engineering object-oriented distributed systems. In Proc. of the IEEE International Conference on Software Maintenance, 2010, pp. 1-6.
[16]. Gu A., Zhou X., Li Z., Li Q., Li L. Measuring Object-Oriented Class Cohesion Based on Complex Networks. Arabian Journal for Science and Engineering, vol. 42, no. 8, 2017, pp. 3551-3561.
[17]. Ajienka N., Capiluppi A., Counsell S. An empirical study on the interplay between semantic coupling and co-change of software classes. Empirical Software Engineering, 2017. pp. 1-35.
[18]. Kumar N., Dadhich R., Shastri A. MAQM: a generic object-oriented framework to build quality models for Web-based applications. International Journal of System Assurance Engineering and Management, vol. 8, no. 2, pp. 2017, pp. 716-729.

[19]. Parashar A., Chhabra J.K. Mining software change data stream to predict changeability of classes of object-oriented software system. Evolving Systems, vol. 7, no. 2, 2016, pp. 117-128.
[20]. Nucci D.D., Palomba F., Rosa G.D., Bavota G., Oliveto R., Lucia A.D. A developer centered bug prediction model, IEEE Transactions on Software Engineering, vol. 44, no. 1, 2018, pp. 5-24.
[21]. Aslam W., Ijaz F. A Quantitative Framework for Task Allocation in Distributed Agile Software Development. IEEE Access, vol. 6, 2018, pp. 15380-15390.

Информация об авторе / Information about author

Ахмед ХАН работает преподавателем на факультете компьютерных наук. Его научные интересы включают искусственный интеллект, параллельные вычисления и теорию информации.

Ahmed KHAN is a lecture at the Department of Computer Science. His research interests include artificial intelligence, parallel processing, and information theory.