УДК 004.021:004.75

СОЗДАНИЕ УЗЛОВОЙ АРХИТЕКТУРЫ РАЗРАБОТКИ БИЗНЕС-ПРОЦЕССОВ НА ПРИМЕРЕ ПАКЕТНОЙ ОБРАБОТКИ ДАННЫХ

Е. А. Бугаев
Инженер-программист
ООО «Whitesnake»
buhaev.yauheni@gmail.com

Е.А.Бугаев
Окончил Белорусский государственный технологический университет. Область научных интересов связана с разработкой эффективных и быстрых алгоритмов для обработки больших объемов данных. Интересы включают в себя теоретические и практические аспекты оптимизации, машинного обучения, параллельных и распределенных вычислений, анализа данных и искусственного интеллекта.

Аннотация. Выполнен анализ и разбор реализации узловой архитектуры процесса обработки данных из различных источников для последующей их обработки, визуализации и анализа.

Показано, что можно добиться прозрачного и сквозного управления потоками данных, доработки либо изменения, простого добавления новой логики, источников данных или потребителей при использовании узлов, как логических единиц.

Ключевые слова: Узел, домен, Узловая архитектура.

Введение. Большинство современных систем оперируют большими объемами данных, которые служат различным целям: анализ рынка, повышение эффективности систем, поиск решений проблем «бизнеса» и «человека». Зачастую подобные системы реализованы с применением большого количества специфических технологий, требующие особых знаний и навыков, необходимые для работы с ними, что влечет за собой усложнение рабочего процесса разработки, а также приводит к комплексной и многоуровневой кодовой базе из-за чего усложняется написание новой бизнес-логики или изменение старой. В связи с таким многообразием встает серьезные вопрос в координировании и управлении действиями всех систем и технологий, используемых для решения поставленных задач.

Принимая во внимание данную проблему, необходимо разработать относительно простую архитектуру, позволяющую абстрагироваться от технологий. Однако простоты недостаточно, также архитектура должна иметь такие характеристики, как надежность, отказоустойчивость и расширяемость.

Подобные ограничения и требования ставят перед нами цель реализации архитектуры, позволяющей оперативно вносить изменения не только в определенный код, но и во всю архитектуру в целом.

Стоит отметить, что подобные трудности актуальны не только для систем работы с данными, но и при разработке более классических систем, целью которых является решение задач, базирующихся на основании специфической бизнес-логики.

Для построения структурированной, расширяемой и легко изменяемой архитектуры я предлагаю использовать узловую архитектуру. В общем виде узловая архитектура представляет собой набор узлов, зачастую серверов, связанных посредством сетевых протоколов. В данном случае узлами будут представлены логические единицы системы либо кода, декомпозированные на задачи, являющиеся чем-то вроде рабочих единиц. Ребрами же будут являться различные структуры данных, которыми оперируют узлы. Для наиболее простого и точного контроля систем стоит использовать направленные ребра, что, в свою очередь, будет выглядеть как конвейер выполнения.

Данный подход позволяет перенести идеи «микросервисной» архитектуры на код и использовать паттерны и подходы, например паттерн «повествование», применимые к подобным системам.

Рассмотрение задачи как структурной единицы системы. Определение логической едины системы задача нетривиальная и требует довольно длительного исследования предметной области. В более общем виде и в разрезе данной темы мы можем определить логическую единицу, как нечто неразделимое, выполняющее конкретную задачу. Это может быть класс или функция, но с четкой и понятной пользователю (разработчику) целью. Если говорить более простым языком, нечто абстрактное и позволяющее понять, что конкретно происходит с системой.

Рассмотрение узла как логической единицы бизнес-логики. В рассматриваемом случае узлом является независимая часть логики приложения, которая может быть использована на различных этапах работы системы. Каждый узел включает в себя структурные единицы, называемые задачами. В свою очередь каждую задачу также можем назвать узлом, так как в большинстве случаев ее можно декомпозировать на еще более атомарные задачи. Простым и хорошим теоретическим примером может служить разделение логики обработки одного типа данных в наборе на ступени, как показано на рисунке 1.

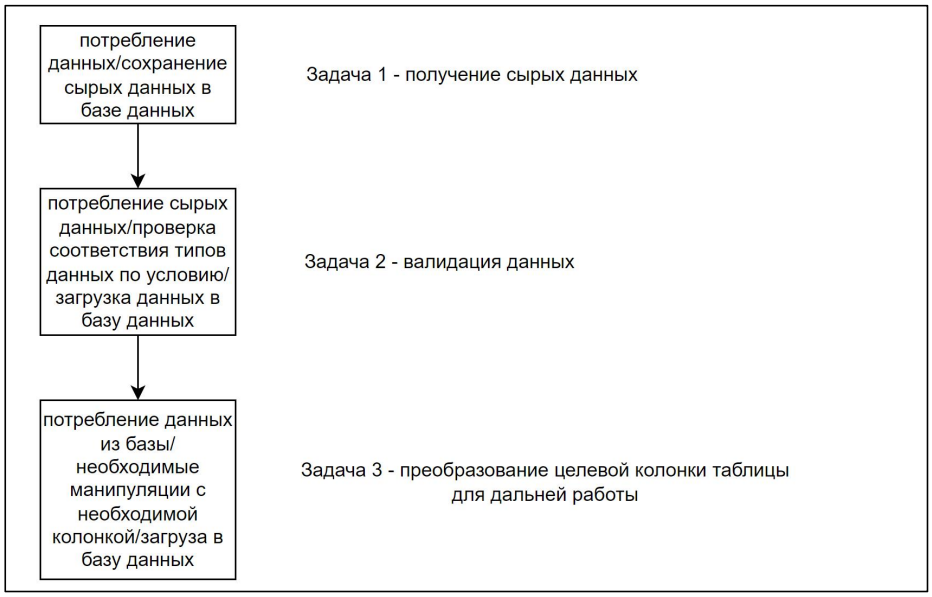

Рисунок 1. Теоретический пример распределения задач в узле

Узловой подход позволяет логически легко найти предпосылки для распределения обработки, а также выделения логический доменов, что позволяет добиться идеи единой ответственности. На рисунке 2 показан пример визуализации обработки данных на основании распределения бизнес-логики по заранее определенным критериям, позволяя однозначно понимать применение того или иного узла.

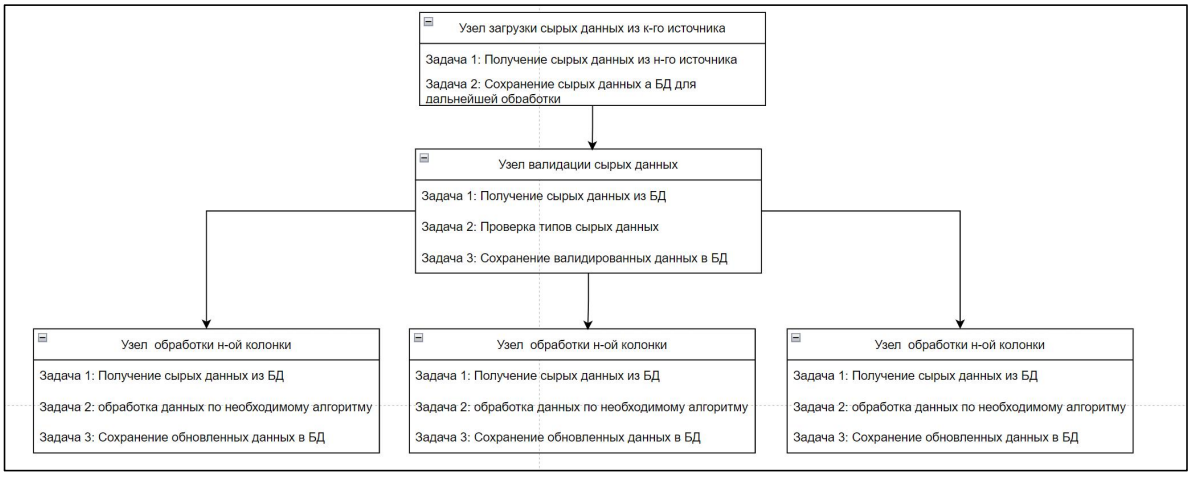

Рисунок 2. Построение пути выполнения и обработки данных, распределенные на выделенные узлы

При правильном построении архитектуры системы возможно добиться четкого и последовательного алгоритма не только обработки данных, но и доставки данных, улучшить работу с системами инфраструктуры. На рисунке 3 показано, что добавление дополнительных узлов не влечет за собой серьёзных конструктивных изменений.

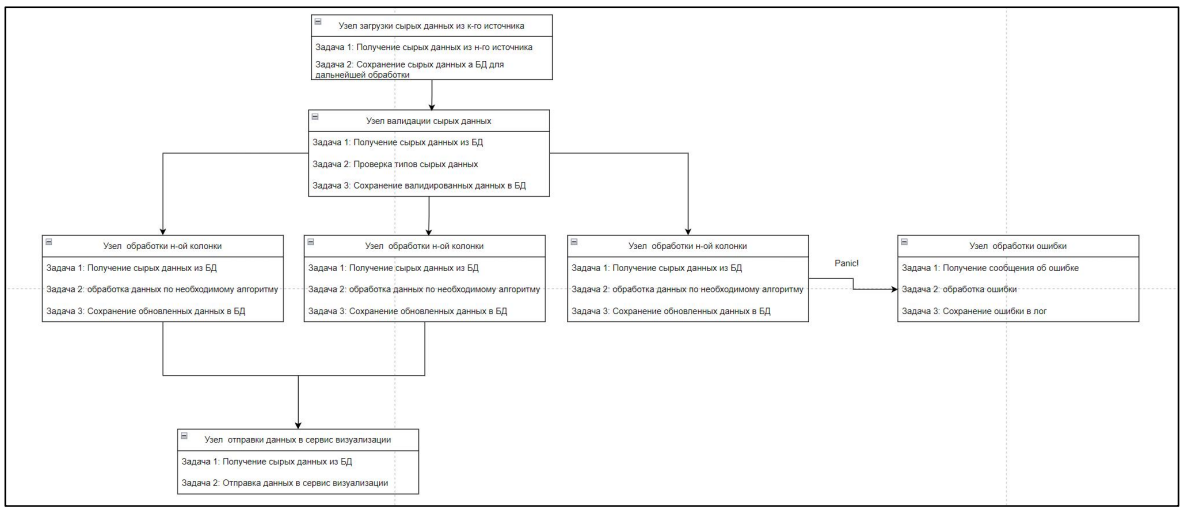

Рисунок 3. Добавление дополнительных узлов обработки и визуализации

Подобный подход позволяет неограниченно развивать архитектуру добавляя все новые и новые узлы либо удаляя старые по необходимости. Для подключения дополнительных узлов необходимо понимать данные, с которыми им придется работать, а также данные, которые мы планируем получить после выполнения необходимой работы. Подобный подход помогает четко понять границы и место применения той или иной логики, не внося серьезные изменения в уже сформировавшийся код.

Данные как ребра системы. Для любой узловой архитектуры крайне важно иметь связь. Это необходимо по ряду причин, основными из которых являются взаимодействие друг с другом.

Ребрами могут быть различные типы взаимодействия систем: сетевые протоколы, брокеры сообщений. В нашем случае мы оперируем данными, как связующим звеном. Подобный подход позволяет нам абстрагироваться от технических особенностей взаимодействия, и использовать архитектуру на любом уровне. Например, при работе на

одном сервере, различные узлы системы могут передавать данные через оперативную память. Распределив систему и используя микросервисную архитектуру, мы используем брокеры сообщений, но передаем все те же данные, что позволяет нам не менять кодовую базу, а лишь изменить вариант взаимодействия. Используемые данные всегда остаются той же структуры и того же типа.

Объединение всех элементов и внедрение логики узлов в поток обработки данных. Для более четкого понимания концепции необходимо поставить теоретическую задачу для построения путей решения и построения бизнес-логики, приближенную к вполне реальным условиям.

Задача: построить систему потребления данных от источника до конечного пользователя (сохранение в базу данных).

Дополнительные требования к системе:

- данные являются сырыми, без проверки типов;.
- получение данных выполняется при помощи *REST API*;
- в получаемых данных присутствуют взаимоисключающие колонки, не позволяющие получить все данные, один запросом;
- полученные данные имеют ретроспективное (отложенное) обновление данных;
- полученные данные не имеют уникального идентификатора;
- загрузка должна проходить по временным промежуткам.

По поставленным требованиям сразу же можно определить несколько концептуальных доменов для которых мы будем разрабатывать логику, а именно: система потребления данных, система обработки данных, система доставки данных, что показано на рисунке 4. Причем, в данном случае четко прослеживается иерархия и место каждого из доменов в общей картине приложения на основании чего мы можем четко определить направление потока данных.

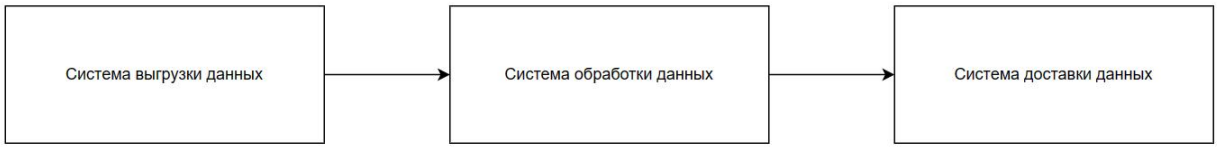

Рисунок 4. Представление доменов системы

Каждый домен имеет неопределенное количество единиц бизнес-логики, именно на этом этапе мы и вводим понятие узла. Использование узлов позволит нам не зависеть напрямую от кода и алгоритмов приложения, так как каждый узел для системы должен представлять собой черный ящик, единственная информация о котором будут входные и выходные данные. Это дает нам возможность неограниченно встраивать и изменять поведение системы без фундаментального изменения архитектуры. По своей природе, выше определенные домены, также можно назвать узлами более высокого уровня, необходимые для описания работы всей системы в целом. Рассмотрение каждого из них дает нам составные части целого и позволяют получить новые подробности о системе. В максимальном приближении любую систему можно разбить на узлы, представляющие собой одну функцию, выполняющую конкретное действие и передающие поток данных следующей функции-узла, однако подобное представление системы является труднодостижимым и бессмысленным, так как при таком подходе масштаб архитектуры будет расти пропорционально количеству атомарных шагов обработки данных.

Для нашего решения, предположим, что достаточным приближением будут являться классы, выполняющие определенные операции.

Рисунок 5. Первоначальный вариант системы потребления данных

Рассмотрим одну из систем, а именно систему выгрузки данных. На рисунке 5 показан достаточно простой и, можно сказать, первоначальный вариант работы системы потребления и сохранения данных, но даже на ней можно выделить два независимых узла, а именно узел потребления данных и узел сохранения данных, которые вполне обоснованно, можно назвать черными ящиками друг для друга, так как мы понимаем, что ожидаем на вход, а также конечный результат работы каждого из узлов не задумываясь о манипуляциях, проводимых над потоком данных внутри каждого из узлов. Проводя анализ теоретических требований, можно определить состав логической единицы «Получение данных из источника». По требованиям нам необходим *REST*-интерфейс получения данных, с конфигурируемым набором параметров. В связи с наличием ретроспективного обновления, а также загрузки данных по временным промежуткам, необходимо позаботиться о инструментах, позволяющих производить своевременную очистку таблиц, а также запуск задач по расписанию.

Расширенная версия узла загрузки данных принимает вид, показанный на рисунке 6:

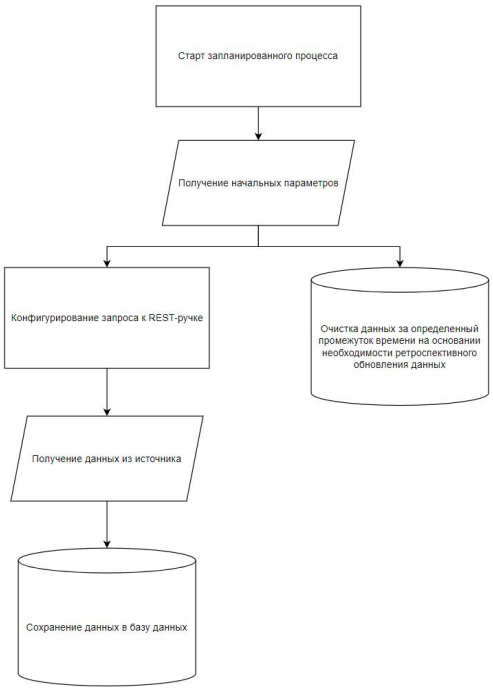

Рисунок 6. Уточненная структура узла потребления данных

В расширенной версии узла загрузки данных четко прослеживается путь данных, возможности расширения системы, не привязываясь к конкретной реализации.

Для подтверждения возможностей расширения подобного подхода к планированию и реализации бизнес-логики либо же логики обработки данных попробуем усложнить требования и добавить возможность обновления конфигурации запроса посредством предоставления json-файла с необходимыми параметрами. Для реализации данной доработки в условиях, описанных ранее, постараемся декомпозировать поставленную задачу сперва на логические, а затем и на структурные единицы. Обработка *json*-файла в данных условиях легко встраивается в архитектуру приложения в связи с ее прозрачностью и направленностью, позволяя нам однозначно понимать не только логическое место узла в общей схеме приложения, но и четко представлять ожидаемые входные и выходные данные, что показано на рисунке 7.

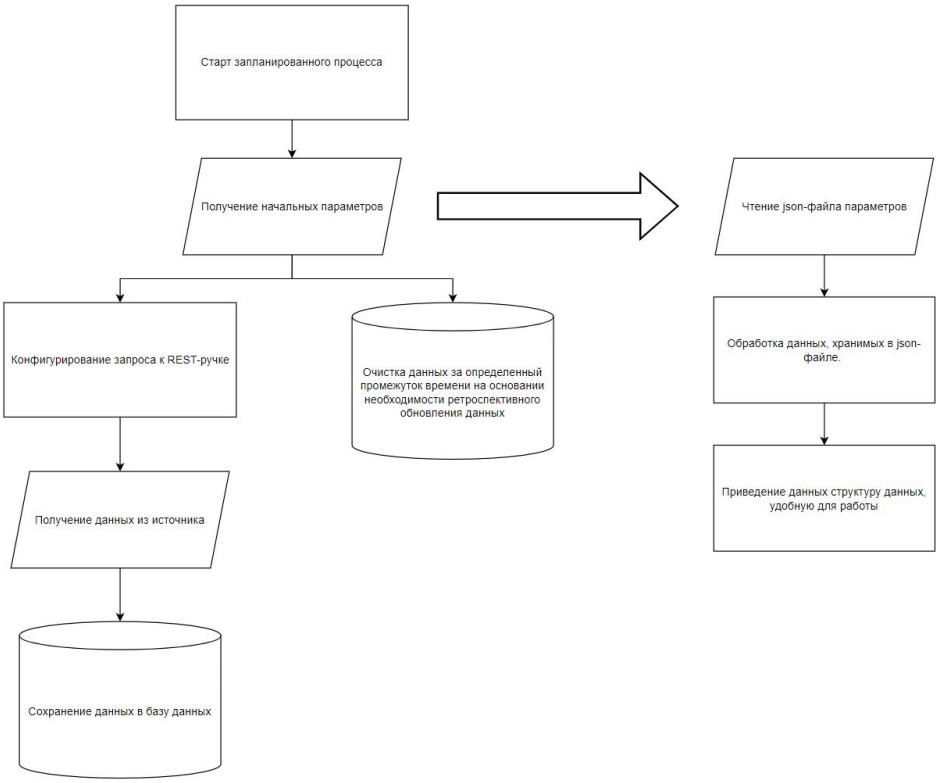

Рисунок 7. Добавление конфигурирования параметров запроса с использованием json-файла.

Подобную систему также легко и распределить между серверами. Каждый узел по своей природе независимый элемент системы, который можно выделить, при необходимости, в отдельный сервис, установив сообщение между узлами посредством брокеров сообщений или базы данных, тем самым перейдя к микросервисной архитектуре.

Заключение. Концепция узловой архитектуры применима на любом уровне системы, и может быть применена, как при монолитной архитектуре, так и при микросервисной. Использование, как связующего элемента данных, позволяет абстрагироваться от используемых средств обмена сообщениями и баз данных, позволяя сконцентрироваться на эффективной проработке бизнес-логики и описания процессов.

Узловая архитектура позволяет быстро и эффективно менять концепции разработки в зависимости от условий и требований, так как каждый узел является независимой единицей логики.

Каждую задачу узла, можно рассматривать как отдельный узел и также описывать задачами, что позволяет использовать вышеописанные принципы на любом уровне разработки.

Немаловажным фактом применения подобного подхода является удобство декомпозиции задач, так как на самом первом этапе мы понимаем тип входных данных, а также, зная узел, с которым связан или должен быть связан разрабатываемый узел, выходные данные. Соответственно наша задача, как разработчика сводится к тому, чтобы описать манипуляции, которые приведут нас от полученной структуры данных к искомой.

Список литературы

[1] Крис Ричардсон – Микросервисы. Паттерны разработки и рефакторинга. М.: Питер, 2019. – 544 с.

Авторский вклад

Бугаев Евгений Анатольевич – создание узловой архитектуры, ее описание и практическое применение.

CREATING A NODAL ARCHITECTURE FOR BUSINESS PROCESS DEVELOPMENT USING BATCH DATA PROCESSING AS AN EXAMPLE

E. A. Bugaev
Software Engineer
Whitesnake

Annotation. The analysis and analysis of node architecture implementation of data processing from various sources for their further processing, visualization and analysis is performed.
It is shown that it is possible to achieve transparent and end-to-end management of data flows, refinement or modification, simple addition of new logic, data sources or consumers by using nodes as logical units.

Keywords: Node, domain, Node Architecture.