perplexity

GoF-паттерны - а еще какие есть?

Ты задал очень хороший и глубокий вопрос!
GoF-паттерны (23 классических шаблона из книги "Design Patterns" 1994 года) — это только базис, фундамент. В современной архитектуре и промышленной разработке используются и другие паттерны, которые появились позже или были адаптированы под новые задачи и масштабы ПО.

Вот обзор **паттернов, которые идут "выше" или "вокруг" GoF** — архитектурные, интеграционные, инфраструктурные, современные паттерны для крупных систем и распределённых решений:

1. Архитектурные паттерны (Architectural Patterns)

Это шаблоны организации крупных подсистем и их взаимодействия.

- **Layered (Слоистая архитектура, N-tier)**
  Классика: Presentation → Business Logic → Data Access → Storage.
  Чёткое разделение ответственности, легко тестировать и поддерживать[1] [2].
- **Client-Server (Клиент-сервер)**
  Разделение на клиента (UI, запросы) и сервер (логика, данные).
- **Microservices (Микросервисы)**
  Система разбита на независимые сервисы, каждый — отдельный процесс, своя БД, API, масштабируется отдельно.
- **Event-Driven (Событийная архитектура, Event Bus)**
  Модули общаются через события, слабая связанность, легко расширять и масштабировать.
- **Microkernel (Плагинная архитектура, Plugin/Microkernel)**
  Ядро + плагины (расширения). Пример — редакторы, IDE, браузеры.
- **Pipe-and-Filter (Конвейер/фильтры)**
  Данные проходят через цепочку обработчиков (фильтров), каждый из которых выполняет одну задачу.
- **Broker (Посредник/брокер)**
  Для распределённых систем: брокер управляет взаимодействием между компонентами (пример — gRPC, RabbitMQ).
- **Blackboard (Доска объявлений)**
  Компоненты обмениваются данными через общее "чёрное поле", каждый модуль может читать/писать туда.

2. Современные паттерны для облака и распределённых систем

(часто называют "Cloud Patterns", "Enterprise Patterns", "Integration Patterns")[3] [1] [2]

- **CQRS (Command Query Responsibility Segregation)**
  Разделяет обработку команд (изменений) и запросов (чтения).
  Используется для повышения производительности и масштабируемости.
- **Event Sourcing**
  Все изменения состояния хранятся как последовательность событий.
  Позволяет "воспроизвести" историю, откатывать, строить сложную аналитику.
- **Circuit Breaker**
  "Предохранитель" — если внешний сервис недоступен, временно прекращаем попытки, чтобы не перегружать систему.
- **Sidecar**
  Вспомогательный процесс рядом с основным сервисом (например, для логирования, мониторинга, проксирования).
- **Backend-for-Frontend (BFF)**
  Для каждого типа клиента (web, mobile) — свой API backend, адаптированный под его нужды.
- **Strangler Fig**
  Постепенная замена старой системы новой: новые функции реализуются в новом сервисе, старые — постепенно "засыхают".

3. Интеграционные паттерны (Integration Patterns)

- **Message Bus/Event Bus**
  Все сервисы общаются через шину сообщений (RabbitMQ, Kafka, ZeroMQ).
- **Service Registry/Discovery**
  Динамический поиск сервисов (Consul, Eureka).
- **API Gateway**
  Единая точка входа для всех API-запросов, маршрутизация, аутентификация, агрегация данных.

4. Потоковые и реактивные паттерны

- **Reactor/Reactive**
  Асинхронная обработка событий, реактивные потоки (RxCpp, RxJS, Rx.NET).
- **Observer/Event Bus**
  Подписка на события, реакция на изменения (используется в UI, обмене данными между модулями).
- **Saga**
  Для управления распределёнными транзакциями (например, в микросервисах).

5. Примеры современных паттернов и их применение

Паттерн	Где применяется/для чего нужен
CQRS + Event Sourcing	Высоконагруженные системы, аналитика, undo/redo
Circuit Breaker	Безопасность и устойчивость интеграций
API Gateway	Микросервисы, мобильные и web-клиенты
Event-Driven Architecture	IoT, аналитика, интеграция сервисов
Sidecar	Kubernetes, сервис-меш, проксирование, мониторинг
Backend-for-Frontend	Мобильные/web приложения с разными сценариями
Microkernel/Plugin	IDE, редакторы, системы расширяемые плагинами

6. Сравнение: GoF vs. Архитектурные/современные паттерны

- **GoF-паттерны** — это "кирпичики" (решают локальные задачи проектирования классов и объектов)[4] [5].
- **Архитектурные паттерны** — это "архитектурные схемы" (определяют, как строить целые системы, их взаимодействие, масштабируемость, отказоустойчивость)[3] [1] [2].
- **Интеграционные и облачные паттерны** — отвечают за коммуникацию между сервисами, интеграцию, безопасность, мониторинг и т.д.

Вывод

GoF-паттерны — это классика ООП (23 шаблона), **Архитектурные/интеграционные/облачные паттерны** — это современные схемы построения крупных, распределённых, модульных систем.

Если нужно подобрать паттерн под конкретную задачу — просто скажи, и я помогу выбрать и реализовать его в твоём проекте!

Источники:

⁂

1. https://www.turing.com/blog/software-architecture-patterns-types
2. https://www.rishabhsoft.com/blog/enterprise-software-architecture-patterns
3. https://betterprogramming.pub/modern-day-architecture-design-patterns-for-software-professionals-9056ee1ed977
4. https://www.digitalocean.com/community/tutorials/gangs-of-four-gof-design-patterns
5. https://swimm.io/learn/system-design/the-top-7-software-design-patterns-you-should-know-about