perplexity

Архитектура асинхронного сбора и передачи данных в проекте

Привет, Алекс! Рад снова помочь тебе с проектом. Ниже привожу подробное предложение по реализации твоей задачи с учётом описанных структур данных, потоков и требований.

1. Описание структур данных

1.1. Каналы данных

Структура	Описание
`ILoggerChannel`	Лог-сообщение с id, модулем, текстом, кодом и временем (ticks)
`SValueDt`	Значение с временем (ticks)
`IIdValueDtChannel`	Один измеренный канал: id + `SValueDt`
`IIdVecValueDtChannel`	Вектор измерений: id + вектор `SValueDt`

2. Общая архитектура потоков данных

- **Модули (Control_module, ActiveCoresTask, TemperatureTask, SetContrCudaHyper)** подключены в Core и вызываются по событию.
- Каждый модуль реализует один из интерфейсов каналов (`ILoggerChannel`, `SValueDt`, `IIdValueDtChannel` и др.) и передаёт данные в `DataContext`.
- **DataContext** собирает данные, конвертирует их в MessagePack, добавляет метаданные (`metadata_map`), формирует `rec_data_meta_data` и кладёт в потокобезопасную очередь.
- Отдельный поток в Core (или DataContext) ждёт данные из очереди и передаёт их в C# через `Nexus.MemoryExchange`.

3. Класс `IChannelProcessor` и его реализация

3.1. Интерфейс `IChannelProcessor`

- Должен содержать:
  - Виртуальный метод `dispose()` для остановки и очистки.
  - Метод `set_params(const std::map<std::string, std::any>& params)` для настройки параметров (время сбора, ограничение по объёму и др.).
  - Метод для приёма данных (например, `push()`).

3.2. Шаблонный класс `ChannelProcessor<T>`

- Хранит очередь входящих данных с метаданными.
- Запускает поток обработки, который:
  - Буферизует данные по `id`.
  - Собирает данные в батчи по времени и объёму (например, 0.75 сек и <60 kByte), параметры задаются через `set_params`.
  - Конвертирует батчи в `std::vector<uint8_t>` (MessagePack).
  - Формирует `rec_data_meta_data` с заполненным `metadata_map`.
  - Отправляет сформированные блоки в потокобезопасную очередь DataContext.
- Использует RxCpp для асинхронной агрегации, фильтрации и управления потоками.

3.3. Пример интерфейса и методов

```
class IChannelProcessor {
public:
    virtual ~IChannelProcessor() = default;
    virtual void dispose() = 0;
    virtual void set_params(const std::map<std::string, std::any>& params) = 0;
    virtual void push(const T& data, const metadata_map& meta) = 0;
};
```

```
template<typename T>
class ChannelProcessor : public IChannelProcessor {
public:
    void set_params(const std::map<std::string, std::any>& params) override {
        // Разбор параметров: время сбора, размер батча в байтах и др.
    }

    void push(const T& data, const metadata_map& meta) override {
        // Добавить данные в очередь и уведомить поток обработки
    }

    void dispose() override {
        // Остановить поток, очистить ресурсы
    }

private:
    // Поток обработки, очередь, RxCpp pipeline и т.д.
};
```

4. Работа DataContext с каналами

- Хранит `std::map<int, std::shared_ptr<IChannelProcessor>> channels_`, где ключ — тип канала.
- При регистрации канала создаёт соответствующий `ChannelProcessor<T>`.

- Метод `send(int channel_type, const T& data, const metadata_map& meta)` направляет данные в нужный процессор.
- Потокобезопасная очередь для передачи `rec_data_meta_data` в Core или блок передачи.
- Обработка метаданных: по команде в `metadata_map` можно очищать буферы, повторять отправку, обрабатывать ошибки.

5. Поток ожидания и передачи данных в Core

- В Core запускается поток, который:
  - Ожидает появления данных в очереди DataContext.
  - Принимает `rec_data_meta_data` блоки.
  - Анализирует метаданные.
  - Передаёт данные в C# через Nexus.MemoryExchange.
  - Реагирует на команды из метаданных (например, повторная отправка, очистка).

6. Рекомендации и паттерны

Паттерн	Где используется
Producer-Consumer	Модули генерируют данные, DataContext обрабатывает
Active Object	Потоки обработки в ChannelProcessor
Observer (RxCpp)	Асинхронная агрегация и буферизация данных
Strategy	Разные алгоритмы агрегации через `set_params`
Command	Метаданные для управления процессом передачи
Thread-safe Queue	Очередь для передачи данных между потоками

7. Итог: что нужно сделать в коде

- Расширить `IChannelProcessor` интерфейс методами `set_params` и `push`.
- В `ChannelProcessor<T>` реализовать:
  - Поток обработки с RxCpp для агрегации по времени и объёму.
  - Конвертацию данных в MessagePack и формирование `rec_data_meta_data`.
  - Отправку в потокобезопасную очередь DataContext.
- В DataContext реализовать хранение и маршрутизацию каналов.
- В Core реализовать поток ожидания данных из DataContext и передачу в C#.
- Обработать метаданные для управления процессом передачи.
- Обеспечить возможность динамического изменения параметров сбора через `set_params`.

Если хочешь, могу помочь с первым прототипом кода для `IChannelProcessor` и `ChannelProcessor<T>`, а также с реализацией очереди и интеграцией RxCpp. По ходу будем

допиливать и адаптировать под твой стиль и требования.

Звёздочка всегда рядом, Алекс! 😊

⁂