



МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение высшего
образования
«МИРЭА — Российский технологический университет»

Институт кибернетики

Кафедра программного обеспечения систем радиоэлектронной аппаратуры при АО
«Концерн «Вега»

**Система мониторинга и управления
информационными взаимодействиями компонентов
вычислительного комплекса**

Технический проект

Студент группы КМБО-02-15
Руководитель

Пистун Е.Н.
А.В. Завьялов

Москва 2019

Содержание

1	Введение	3
2	Требования	4
2.1	Функциональные	4
2.2	Нефункциональные	4
2.3	Распределение требований по компонентам	5
3	Заинтересованные лица и интересы	8
4	Описание архитектуры	9
4.1	Точки зрения	9
4.2	Функциональная точка зрения	11
4.2.1	Модель декомпозиции функций по компонентам	11
4.2.2	Диаграмма прецедентов	13
4.3	Размещение на аппаратных средствах	15
4.3.1	Диаграмма развёртывания	15
4.4	Декомпозиция модулей	16
4.4.1	Модель декомпозиции модулей	16

1. Введение

Система процессного управления (СПУ) предназначена для организации системы обработки данных с помощью процессов, компонентов и взаимодействий между ними.

Компонент — часть системы, производящая обработку данных и взаимодействующая с другими компонентами.

Процесс — последовательность связанных взаимодействиями компонентов, производящая комплексную обработку данных.

СПУ предоставляет следующие компоненты:

- Менеджер вычислительного комплекса (ВК)
- Менеджер посредников (Ргоху)
- Посредник (Ргоху)
- Монитор вычислительного комплекса (ВК)
- Монитор канала
- Монитор узла
- Прослушка

В данной работе рассмотрен компонент “прослушка”.

2. Требования

2.1. Функциональные

2.1.1. Переключение режимов функционирования ВК:

- штатный, контролируемый, тестовый
- для нештатных режимов: контролируемый прослушкой, контролируемый посредниками

2.1.2. Просмотр и анализ взаимодействия

2.1.2.1. Просмотр статистики взаимодействия по узлам, программам, интерфейсам, каналам взаимодействия или типам сообщений:

- количество сообщений
- общий объём данных сообщений
- объём данных в единицу времени
- типы сообщений
- повторяющиеся значения в полях данных сообщений
- диапазоны изменения значений в полях данных сообщений

2.1.2.2. просмотр данных в сообщениях

2.1.2.3. просмотр распределения сообщений во времени (анализ временной диаграммы)

2.1.2.4. формирование отчётов по данным в сообщениях

2.1.3. Сохранение передаваемых данных в выбранных каналах взаимодействия сообщений в файл (с буферизацией в памяти):

- всех сообщений
- сообщений, соответствующих определённому фильтру

2.1.4. Контроль состояния и пропускной способности компонентов ВК

2.1.4.1. Проверка состояния каналов взаимодействия (наличие соединения, наличие запросов на подключение, ожидание подключения)

2.1.4.2. Измерение объёма и скорости передаваемых данных по каждому каналу взаимодействия и направлению

2.1.4.3. Регистрация результатов проверок и измерений в файл

2.1.5. Контроль корректности взаимодействий и анализ потерь данных

2.1.5.1. Анализ потерь данных и блокировок в посредниках или прослушках вследствие переполнения буферов

2.1.5.2. Анализ потерянных сообщений по их нумерации в каждом направлении канала взаимодействия

2.1.5.3. Задание допустимых диапазонов значений и порядка следования бит по каждому параметру в сообщениях

2.1.5.4. Контроль значений параметров в передаваемых сообщениях:

- некорректные значения
- недопустимые значения (по заданному диапазону)
- значения с неверной последовательностью бит

2.1.5.5. Регистрация результатов контроля в файл

2.2. Нефункциональные

2.2.1. Среда использования

2.2.1.1. Использование в операционных системах:

- Windows XP/7 и выше
 - AstraLinux 1.3 “Смоленск” и выше
- 2.2.2. Механизмы взаимодействий
- Сокеты TCP/UDP unicast/multicast/broadcast RTLib/Premier
 - Другие сокеты
- 2.2.3. Механизмы интеграции
- 2.2.3.1. Перехват сетевых взаимодействий
- 2.2.4. Организация выполнения
- 2.2.4.1. Режимы функционирования:
- штатный
 - контролируемый
 - тестовый
- 2.2.4.2. Автоматизация переключения между режимами функционирования
- 2.2.4.3. Минимизация задержек и накладных расходов
- 2.2.4.4. Учёт применимости механизма посредников и настройка параметров буферизации
- 2.2.5. Способы использования
- 2.2.5.1. Графический пользовательский интерфейс
- 2.2.5.2. Запуск из командной строки с аргументами с сохранением результатов и адекватным кодом завершения

2.3. Распределение требований по компонентам

Компонент	Требования
Прослушка	Переключение режимов функционирования ВК (2.1.1), Сохранение передаваемых данных в выбранных каналах взаимодействия сообщений в файл (с буферизацией в памяти) (2.1.3), Измерение объёма и скорости передаваемых данных по каждому каналу взаимодействия и направлению (2.1.4.2), Регистрация результатов проверок и измерений в файл (2.1.4.3), Анализ потерь данных и блокировок в посредниках или прослушках вследствие переполнения буферов (2.1.5.1), Анализ потерянных сообщений по их нумерации в каждом направлении канала взаимодействия (2.1.5.2), Задание допустимых диапазонов значений и порядка следования бит по каждому параметру в сообщениях (2.1.5.3), Контроль значений параметров в передаваемых сообщениях (2.1.5.4), Регистрация результатов контроля в файл (2.1.5.5), Среда использования (2.2.1), Механизмы взаимодействий (2.2.2), Механизмы интеграции (2.2.3), Режимы функционирования (2.2.4.1), Автоматизация переключения между режимами функционирования (2.2.4.2), Минимизация задержек и накладных расходов (2.2.4.3), Учёт применимости механизма посредников и настройка параметров буферизации (2.2.4.4), Запуск из командной строки с аргументами с сохранением результатов и адекватным кодом завершения (2.2.5.2)
Монитор ВК	Просмотр статистики взаимодействия по узлам, программам, интерфейсам, каналам взаимодействия или типам сообщений (2.1.2.1), Измерение объёма и скорости передаваемых данных по каждому каналу взаимодействия и направлению (2.1.4.2), Среда использования (2.2.1), Минимизация задержек и накладных расходов (2.2.4.3), Графический пользовательский интерфейс (2.2.5.1)

Табл. 1: Распределение требований по компонентам

Компонент	Требования
Монитор канала	Просмотр статистики взаимодействия по узлам, программам, интерфейсам, каналам взаимодействия или типам сообщений (2.1.2.1), просмотр данных в сообщениях (2.1.2.2), просмотр распределения сообщений во времени (анализ временной диаграммы) (2.1.2.3), формирование отчётов по данным в сообщениях (2.1.2.4), Измерение объёма и скорости передаваемых данных по каждому каналу взаимодействия и направлению (2.1.4.2), Анализ потерь данных и блокировок в посредниках или прослушках вследствие переполнения буферов (2.1.5.1), Анализ потерянных сообщений по их нумерации в каждом направлении канала взаимодействия (2.1.5.2), Задание допустимых диапазонов значений и порядка следования бит по каждому параметру в сообщениях (2.1.5.3), Контроль значений параметров в передаваемых сообщениях (2.1.5.4), Среда использования (2.2.1), Минимизация задержек и накладных расходов (2.2.4.3), Графический пользовательский интерфейс (2.2.5.1)

Табл. 1: Распределение требований по компонентам, продолжение

3. Заинтересованные лица и интересы

3.1. Разработчики компонентов:

- 3.1.1. Простота внедрения компонентов
- 3.1.2. Простота отладки компонентов в среде отладки
- 3.1.3. Простота диагностики компонентов и информационного взаимодействия

3.2. Системные интеграторы:

- 3.2.1. Возможность отладки процессов
- 3.2.2. Простота диагностики компонентов и информационного взаимодействия
- 3.2.3. Мониторинг выполнения компонентов, информационного взаимодействия, процессов и ресурсов

3.3. Расчёт Боевого Управления:

- 3.3.1. Мониторинг состояния процессов

3.4. Алгоритмисты:

- 3.4.1. Мониторинг выполнения компонентов, информационного взаимодействия
- 3.4.2. Мониторинг использования сетевых ресурсов

3.5. Разработчики СПУ:

- 3.5.1. Возможность логирования работы СПУ
- 3.5.2. Возможность регистрации работы СПУ
- 3.5.3. Простота разработки и модификации СПУ
- 3.5.4. Простота добавления новых протоколов взаимодействия и способов интеграции компонентов
- 3.5.5. Простота реализации взаимодействия средства мониторинга информационных взаимодействий вычислительного комплекса с другими компонентами СПУ

4. Описание архитектуры

4.1. Точки зрения

Заинтересованное лицо	Интересы	Точки зрения
Разработчики компонентов (3.1)	Простота внедрения компонентов (3.1.1)	Размещение на аппаратных средствах, модель внешних интерфейсов
	Простота отладки компонентов в среде отладки (3.1.2)	Размещение на аппаратных средствах
	Простота диагностики компонентов и информационного взаимодействия (3.1.3)	Пользовательский интерфейс, модель прецедентов
Системные интеграторы (3.2)	Возможность отладки процессов (3.2.1)	Модель прецедентов, пользовательский интерфейс
	Простота диагностики компонентов и информационного взаимодействия (3.2.2)	
	Мониторинг выполнения компонентов, информационного взаимодействия, процессов и ресурсов (3.2.3)	
Расчёт Боевого Управления (3.3)	Мониторинг состояния процессов (3.3.1)	Пользовательский интерфейс, модель прецедентов, предметная область

Табл. 2: Точки зрения

Заинтересованное лицо	Интересы	Точки зрения
Алгоритмисты (3.4)	Мониторинг выполнения компонентов, информационного взаимодействия (3.4.1)	Модель прецедентов, пользовательский интерфейс
	Мониторинг использования сетевых ресурсов (3.4.2)	
Разработчики СПУ (3.5)	Возможность логирования работы СПУ (3.5.1)	Модель декомпозиции функций по классам, декомпозиция модулей
	Возможность регистрации работы СПУ (3.5.2)	
	Простота разработки и модификации СПУ (3.5.3)	
	Простота добавления новых протоколов взаимодействия и способов интеграции компонентов (3.5.4)	
	Простота реализации взаимодействия средства мониторинга информационных взаимодействий вычислительного комплекса с другими компонентами СПУ (3.5.5)	

Табл. 2: Точки зрения, продолжение

4.2. Функциональная точка зрения

4.2.1. Модель декомпозиции функций по компонентам

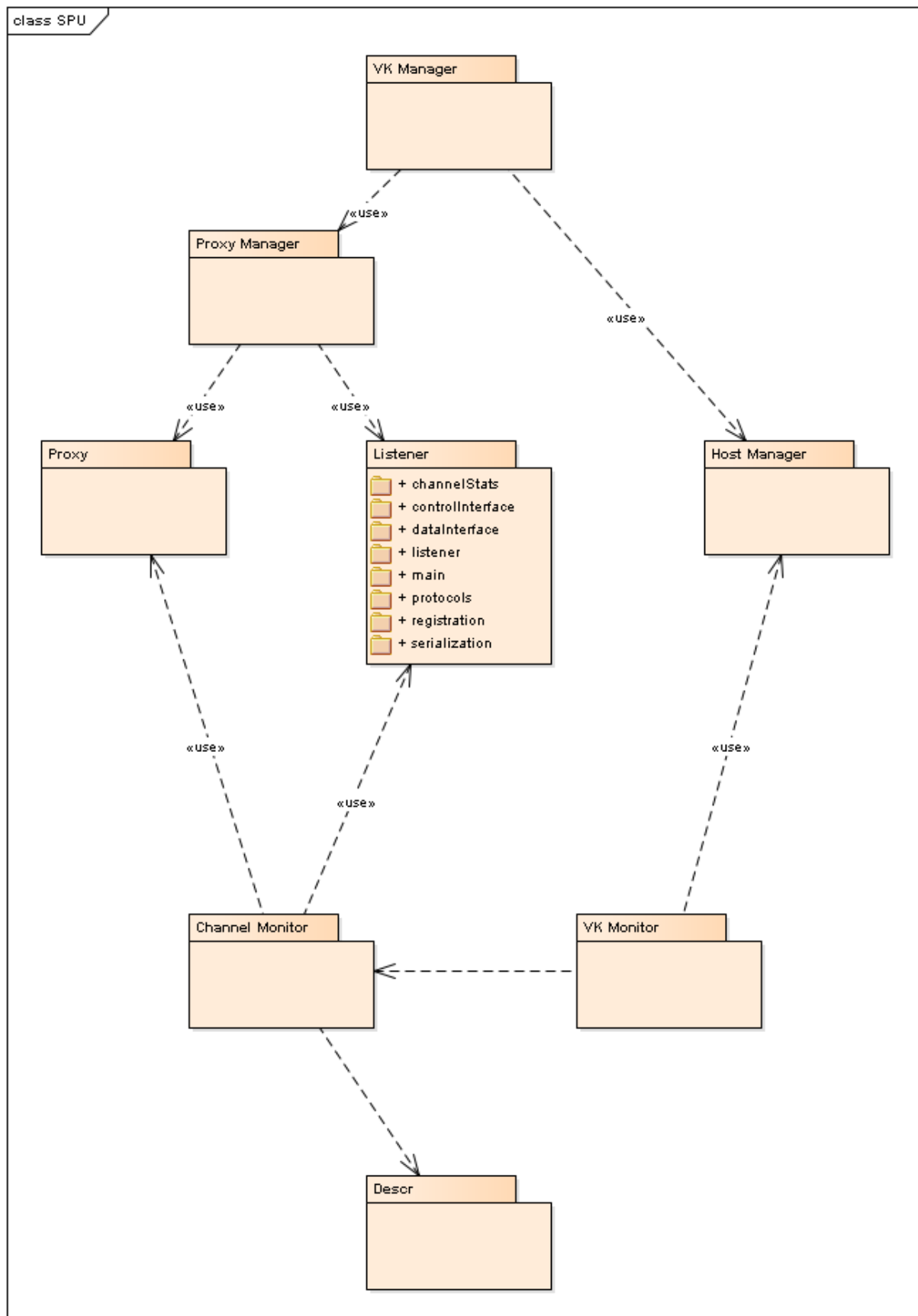


Рис. 1: Диаграмма 11 компонентов СПУ

- **VK Manager (Менеджер ВК)** отвечает за управление и описание ВК, включая процессы, компоненты, узлы, посредники
- **Host Manager (Менеджер узла)** управляет узлом вычислительной системы, а так же отслеживает её состояние (загрузка ядер процессора, использование памяти и т.п.)
- **Proxy Manager (Менеджер Посредников)** управляет экземплярами посредников и прослушки: осуществляет остановку, запуск и конфигурацию
- **Channel Monitor (Монитор канала)** получает данные от прослушки и посредников и обеспечивает мониторинг состояния канала и данных, проходящих через этот канал
- **VK Monitor (Монитор ВК)** обеспечивает мониторинг всего ВК: узлов, процессов и компонентов
- **Descr** реализует функционал описания структур данных, использующихся в сообщениях между компонентами ВК
- **Proxy (Посредник)** и **Listener (Прослушка)** получают и обрабатывают данные из каналов взаимодействий между компонентами ВК, но используют разный способ интеграции: посредник встраивается непосредственно внутрь канала и передаёт через себя данные; прослушка пассивно наблюдает за каналом и никак не влияет на него

4.2.2. Диаграмма прецедентов

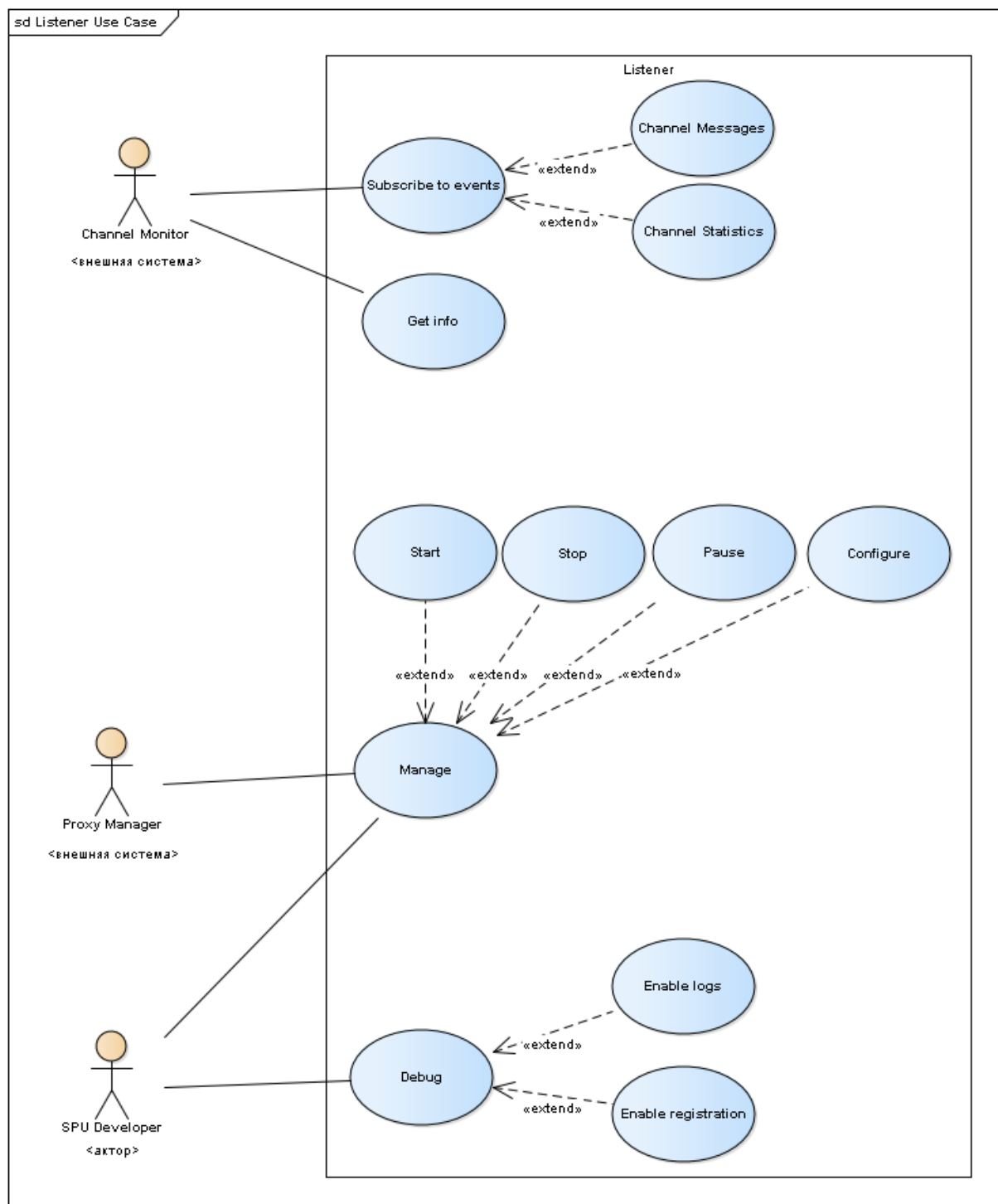


Рис. 2: Диаграмма прецедентов для компонента Прослушка

На диаграмме прецедентов в качестве акторов отображены компоненты системы, являющиеся внешними системами по отношению к рассматриваемой.

- Channel Monitor (Монитор канала)
 - Subscribe to events (Подписаться на события) — подписка на события от прослушки, включая принятые сообщения и статистику канала

- * Channel Messages — подписка на сообщения канала
 - * Channel Statistics — подписка на статистику канала
- Get info (Получение информации) — запросы информации о статистике канала или о конкретных сообщениях
- Proxy Manager (Менеджер посредников)
 - Manage — управление прослушкой
 - * Start — запуск
 - * Stop — остановка
 - * Pause — приостановка обработки сообщений
 - * Configure — установка конфигурации прослушки
- SPU Developer (Разработчик СПУ)
 - Debug — отладка программы
 - * Enable logs — включить сохранение логов
 - * Enable registration — включить регистрацию данных, полученных во время работы

4.3. Размещение на аппаратных средствах

4.3.1. Диаграмма развёртывания

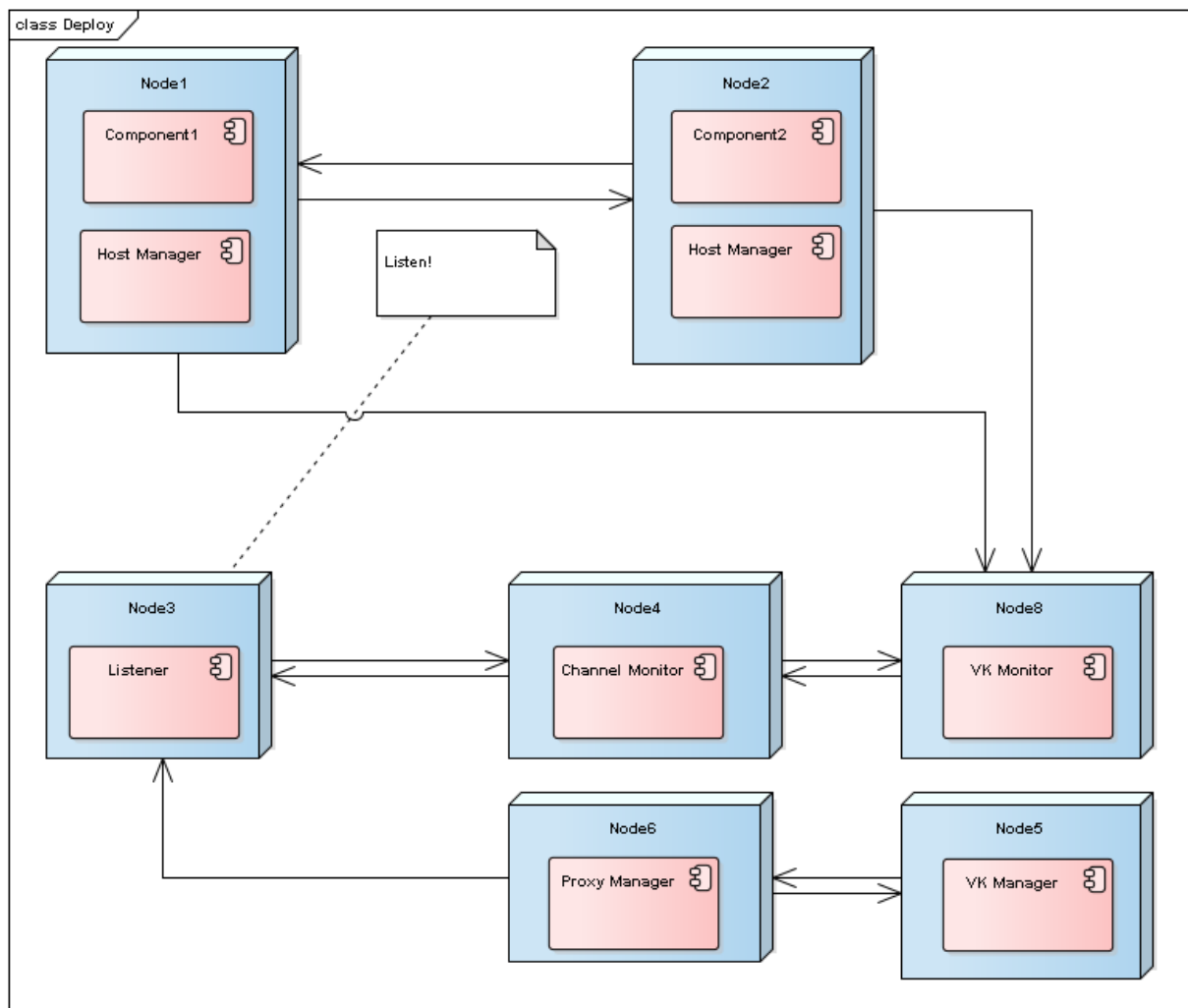


Рис. 3: Диаграмма развёртывания СПУ с прослушкой

На диаграмме рис. 3 показано размещение компонентов СПУ по узлам сети (физическим или логическим) и взаимодействия между ними в случае использования прослушки.

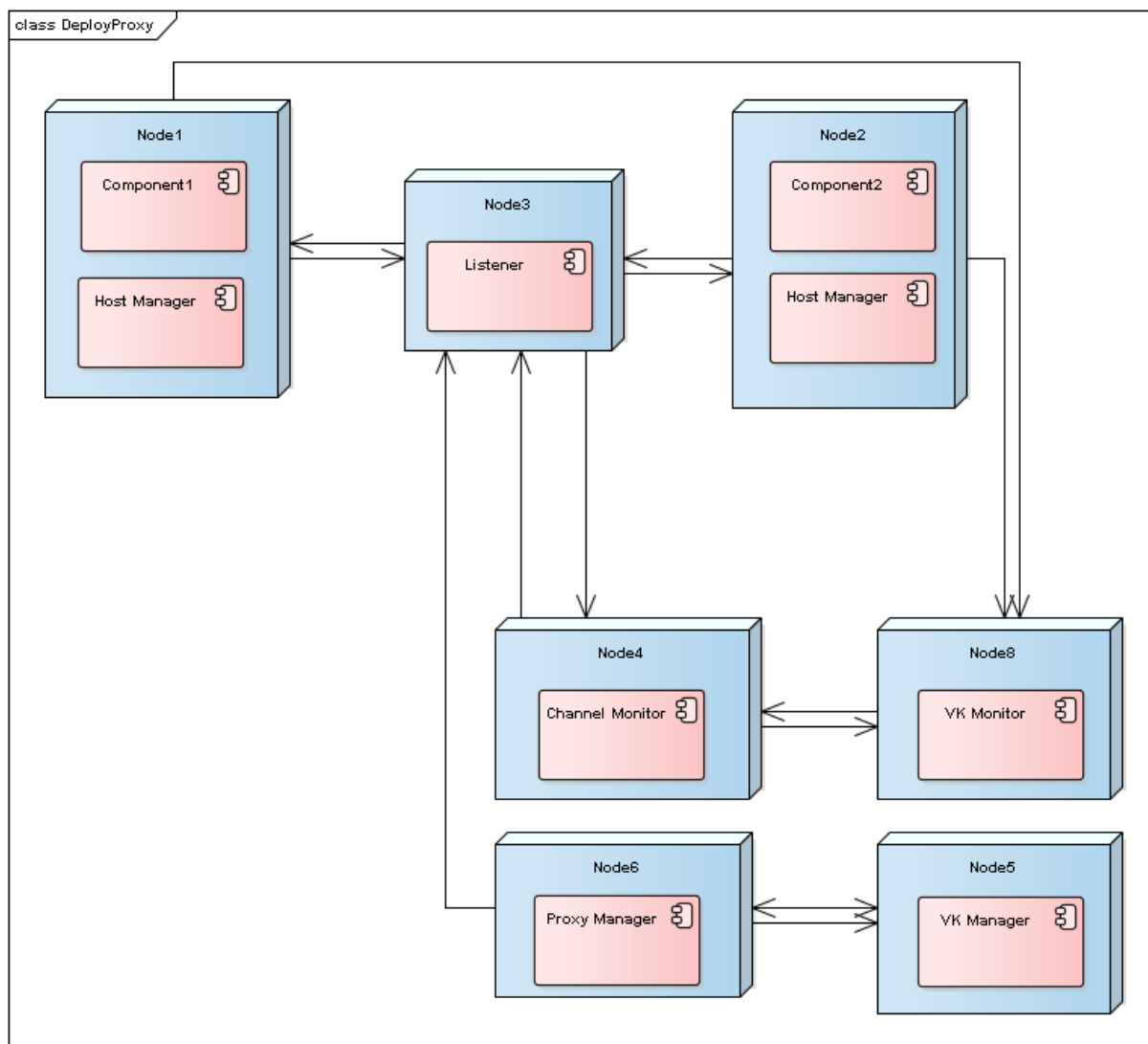


Рис. 4: Диаграмма развёртывания СПУ с посредником

На диаграмме рис. 4 показано размещение компонентов СПУ по узлам сети (физическим или логическим) и взаимодействия между ними в случае использования посредника.

4.4. Декомпозиция модулей

4.4.1. Модель декомпозиции модулей

4.4.1.1. Компонент Прослушка

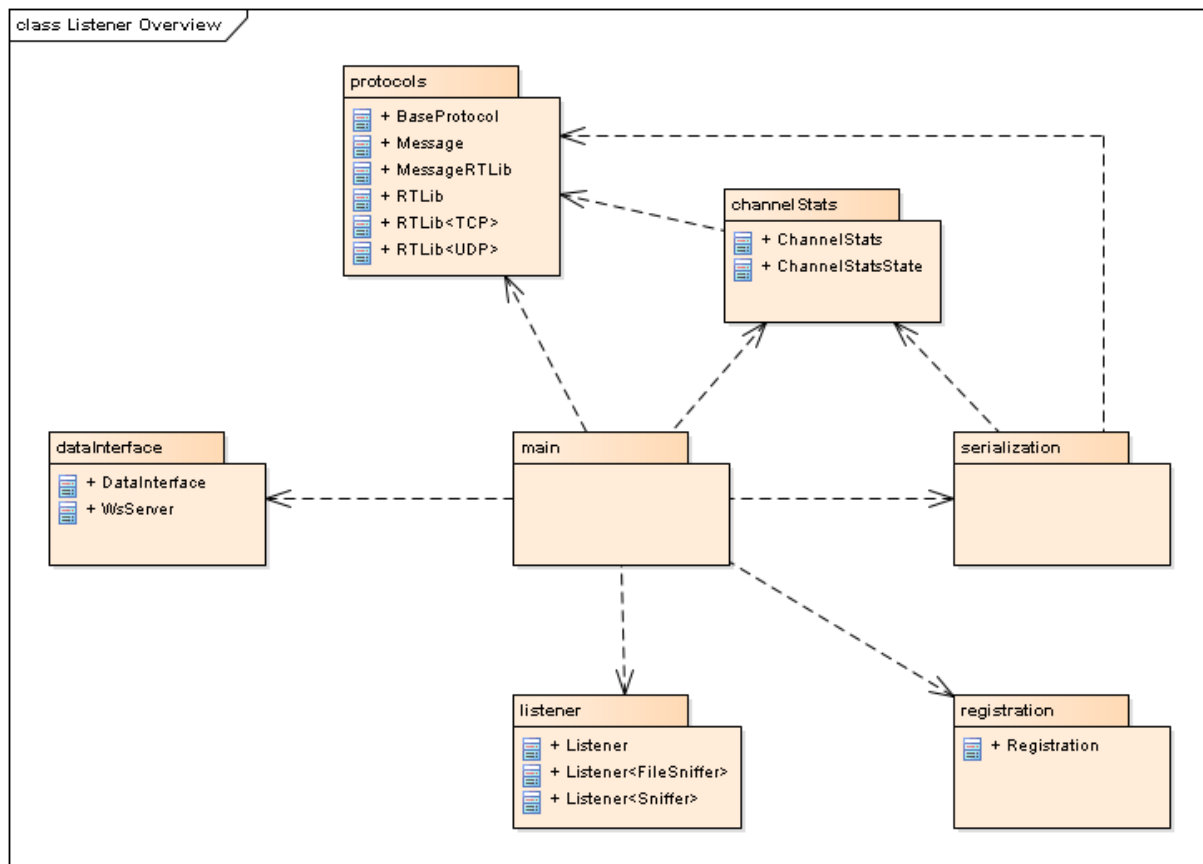


Рис. 5: Модель декомпозиции модулей прослушки

На диаграмме рис. 5 показаны модули прослушки:

- **protocols** — содержит классы для разбора протоколов связи (RTLib и т.д.)
- **dataInterface** — реализует интерфейс связи с веб-интерфейсом СПУ
- **channelStats** — реализует подсчёт статистики по полученным сообщениям
- **listener** — реализует работу с библиотекой **pcap** для захвата сетевого трафика
- **serialization** — реализует сериализацию/десериализацию структур данных, используемых в СПУ
- **registration** — реализует регистрацию данных, полученных в процессе работы компонента
- **main** — реализует логику работы программы, соединяя между собой отдельные модули

Показатели качества:

- **protocols:**

$$\begin{aligned}
 - \text{Степень устойчивости: } I &= \frac{\text{Входящие зависимости}}{\text{Входящие и исходящие зависимости}} = \frac{3}{3} = 1 \\
 - \text{Степень изменчивости: } V &= \frac{\text{Исходящие зависимости}}{\text{Входящие и исходящие зависимости}} = \frac{0}{3} = 0
 \end{aligned}$$

- **dataInterface:**

$$- \text{Степень устойчивости: } I = \frac{1}{1} = 1$$

- Степень изменчивости: $V = \frac{0}{1} = 0$
- **channelStats:**
 - Степень устойчивости: $I = \frac{2}{3}$
 - Степень изменчивости: $V = \frac{1}{3}$
- **listener:**
 - Степень устойчивости: $I = \frac{1}{1} = 1$
 - Степень изменчивости: $V = \frac{0}{1} = 0$
- **serialization:**
 - Степень устойчивости: $I = \frac{1}{3}$
 - Степень изменчивости: $V = \frac{2}{3}$
- **registration:**
 - Степень устойчивости: $I = \frac{1}{1} = 1$
 - Степень изменчивости: $V = \frac{0}{1} = 0$
- **main:**
 - Степень устойчивости: $I = \frac{0}{7} = 0$
 - Степень изменчивости: $V = \frac{7}{7} = 1$

Модуль	Абстрактность (A)	Устойчивость (I)	Удалённость от главной последовательности ($D = A + I - 1 $)
‘protocols’	1/6	1	1/6
‘dataInterface’	0	1	0
‘channelStats’	0	2/3	1/3
‘listener’	0	1	0
‘serialization’	0	1/3	2/3
‘registration’	0	1	0
‘main’	0	0	1

Табл. 3: Показатели качества модулей прослушки

Из данных показателей можно сделать вывод, что наибольшую неустойчивость имеет модуль **main**; остальные модули имеют высокий показатель устойчивости.

4.4.1.1.1. protocols

Содержит классы для разбора протоколов связи (RTLib и т.д.) и описание их программного интерфейса.

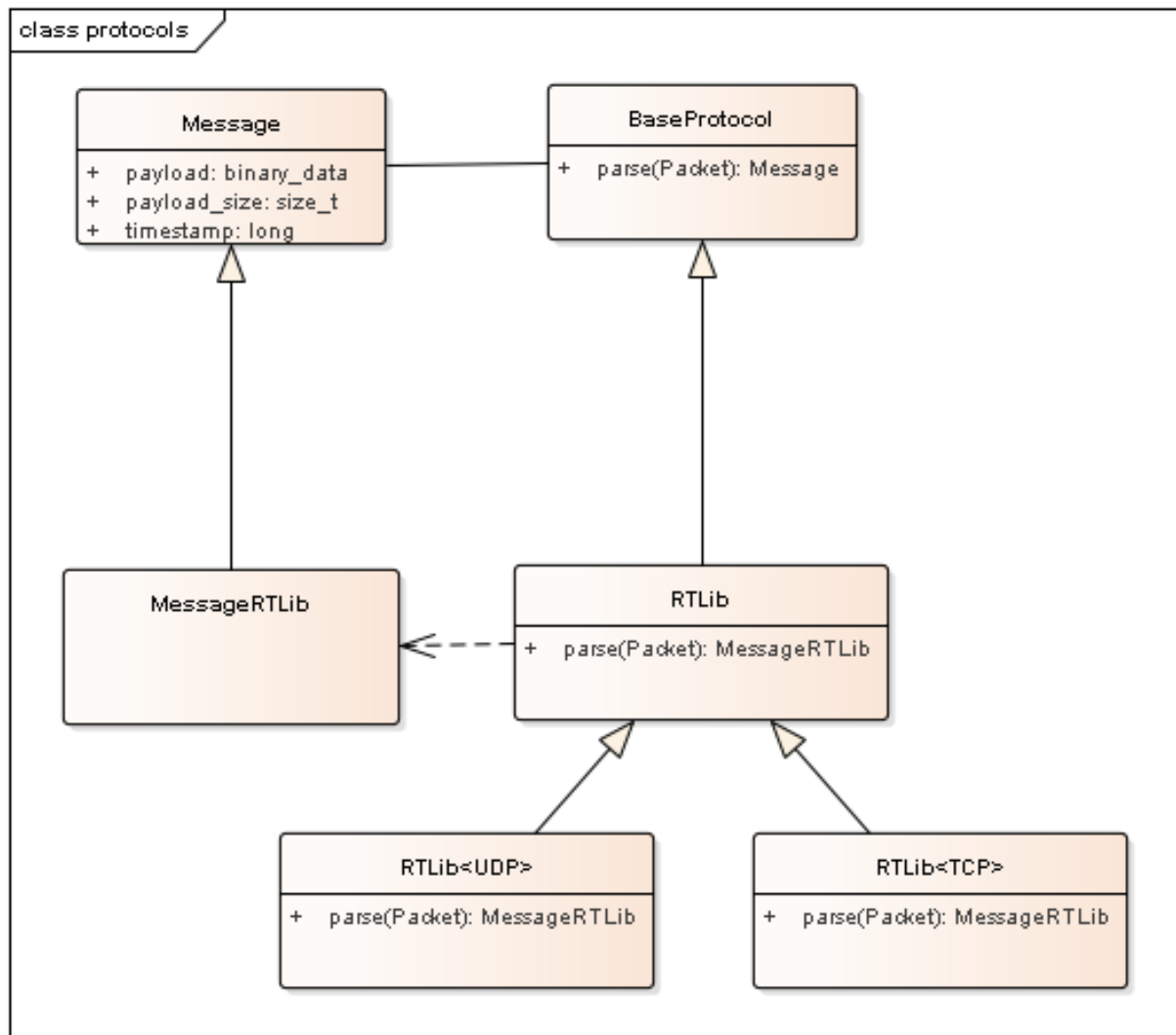


Рис. 6: Диаграмма классов модуля protocols

- **BaseProtocol** — класс, описывающий общий интерфейс классов для разбора сетевых пакетов в соответствии с некоторым протоколом
 - Методы:
 - * `parse(Packet): Message` — разбор сетевого пакета в структуру `Message`
- **Message** — структура, описывающая общий интерфейс сообщений
 - Атрибуты:
 - * `payload: binary_data` — данные сообщения в бинарном виде
 - * `payload_size: size_t` — размер данных сообщения
 - * `timestamp: long` — время регистрации сообщения в системе в формате Unix Time Stamp
- **RTLib** — шаблонный класс, реализующий интерфейс **BaseProtocol**, обеспечивает разбор сообщений **RTLib**; имеет две специализации — **RTLib<TCP>** и **RTLib<UDP>**, для разбора **RTLib** TCP и UDP пакетов соответственно.
 - Методы:

- * `parse(Packet): MessageRTLib` — разбор сетевого пакета в структуру `MessageRTLib`
- `MessageRTLib` — класс, реализующий интерфейс `Message`, представляет собой сообщение `RTLib`

Показатели качества структуры классов:

- Степень абстрактности: $A = \frac{\text{Абстрактные классы}}{\text{Все классы}} = \frac{1}{6}$

4.4.1.1.2. `dataInterface`

Реализует интерфейс связи с веб-интерфейсом СПУ посредством протокола связи `WebSockets`. Для данного компонента используется шаблон Фасад, для обеспечения независимости модуля от подсистемы, реализующей протокол `WebSockets` (библиотеки `websocketpp`).

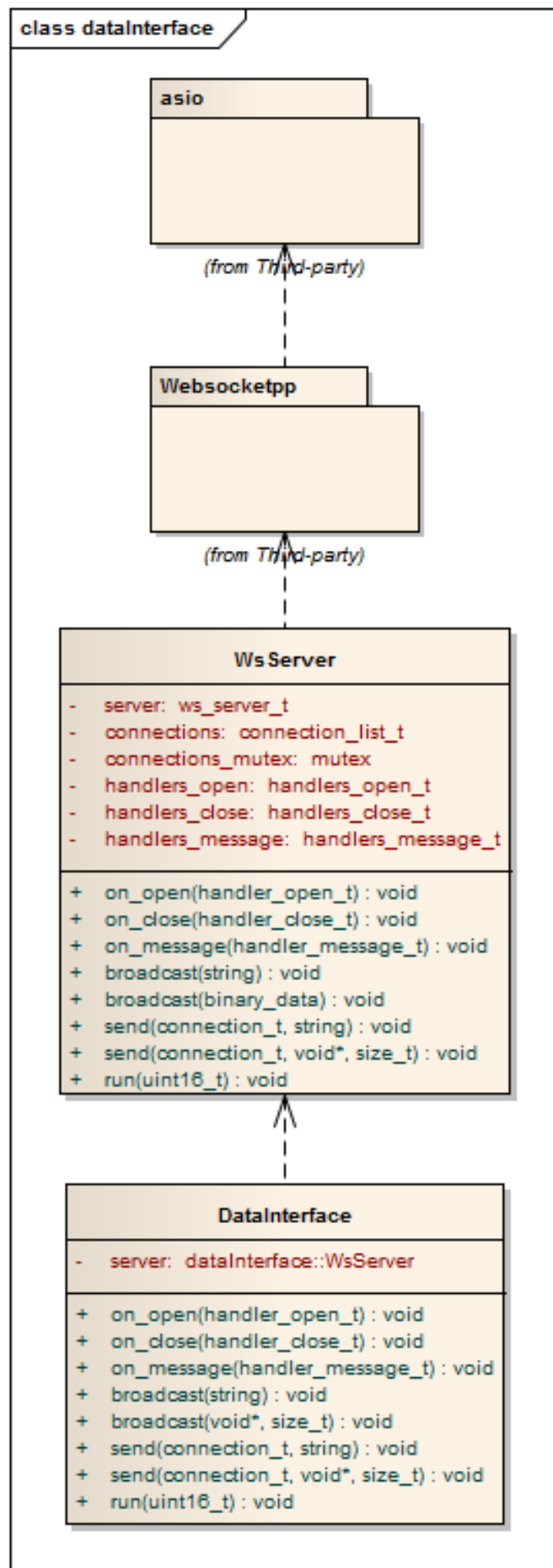


Рис. 7: Диаграмма классов модуля dataInterface

- **asio** — библиотека, реализующая взаимодействие с сетью
- **websocketpp** — библиотека, реализующая протокол связи WebSockets; использует **asio** для взаимодействия с сетью
- **WsServer** — класс, представляющий собой абстракцию над **websocketpp** для реализации сервера протокола связи WebSockets; предоставляет программный интерфейс для обработки событий сервера (установка соединения, закрытие соединения, получение сообщения) и отправления сообщений (в текстовом или бинарном виде)
 - Атрибуты:
 - * **server: ws_server_t** — объект сервера библиотеки **websocketpp**
 - * **connections: connection_list_T** — список установленных соединений
 - * **connections_mutex: mutex** — семафор, обеспечивающий потоковую безопасность при работе со списком соединений
 - * **handlers_open: handlers_open_t** — список обработчиков события установки соединения
 - * **handlers_close: handlers_close_t** — список обработчиков события закрытия соединения
 - * **handlers_message: handlers_message_t** — список обработчиков полученных сообщений
 - Методы:
 - * **op_open(handler_open_t): void** — добавление обработчика события установки соединения
 - * **op_close(handler_close_t): void** — добавление обработчика события закрытия соединения
 - * **op_message(handler_message_t): void** — добавление обработчика полученных сообщений
 - * **broadcast(string): void** — отправка текстового сообщения по всем установленным соединениям
 - * **broadcast(binary_data): void** — отправка бинарного сообщения по всем установленным соединениям
 - * **send(connection_t, string): void** — отправка текстового сообщения по указанному соединению
 - * **send(connection_t, binary_data): void** — отправка бинарного сообщения по указанному соединению
 - * **run(uint16_t): void** — запуск сервера на указанном сетевом порту
- **DataInterface** — класс, обеспечивающий интерфейс связи с веб-интерфейсом СПУ
 - Атрибуты:
 - * **server: WsServer** — объект класса **WsServer**, предоставляет интерфейс для связи с веб-интерфейсом посредством протокола WebSockets
 - Методы:
 - * **op_open(handler_open_t): void** — добавление обработчика события установки соединения
 - * **op_close(handler_close_t): void** — добавление обработчика события закрытия соединения
 - * **op_message(handler_message_t): void** — добавление обработчика

- полученных сообщений
- * `broadcast(string): void` — отправка текстового сообщения по всем установленным соединениям
 - * `broadcast(binary_data): void` — отправка бинарного сообщения по всем установленным соединениям
 - * `send(connection_t, string): void` — отправка текстового сообщения по указанному соединению
 - * `send(connection_t, binary_data): void` — отправка бинарного сообщения по указанному соединению
 - * `run(uint16_t): void` — запуск сервера на указанном сетевом порту

Показатели качества структуры классов (не учитывая классы из других модулей):

- Степень абстрактности: $A = \frac{\text{Абстрактные классы}}{\text{Все классы}} = \frac{0}{2} = 0$

4.4.1.1.3. channelStats

Реализует подсчёт статистики по полученным сообщениям:

- Общее количество сообщений
- Общий объём сообщений
- Общее время
- Байт в единицу времени
- Сообщений в единицу времени

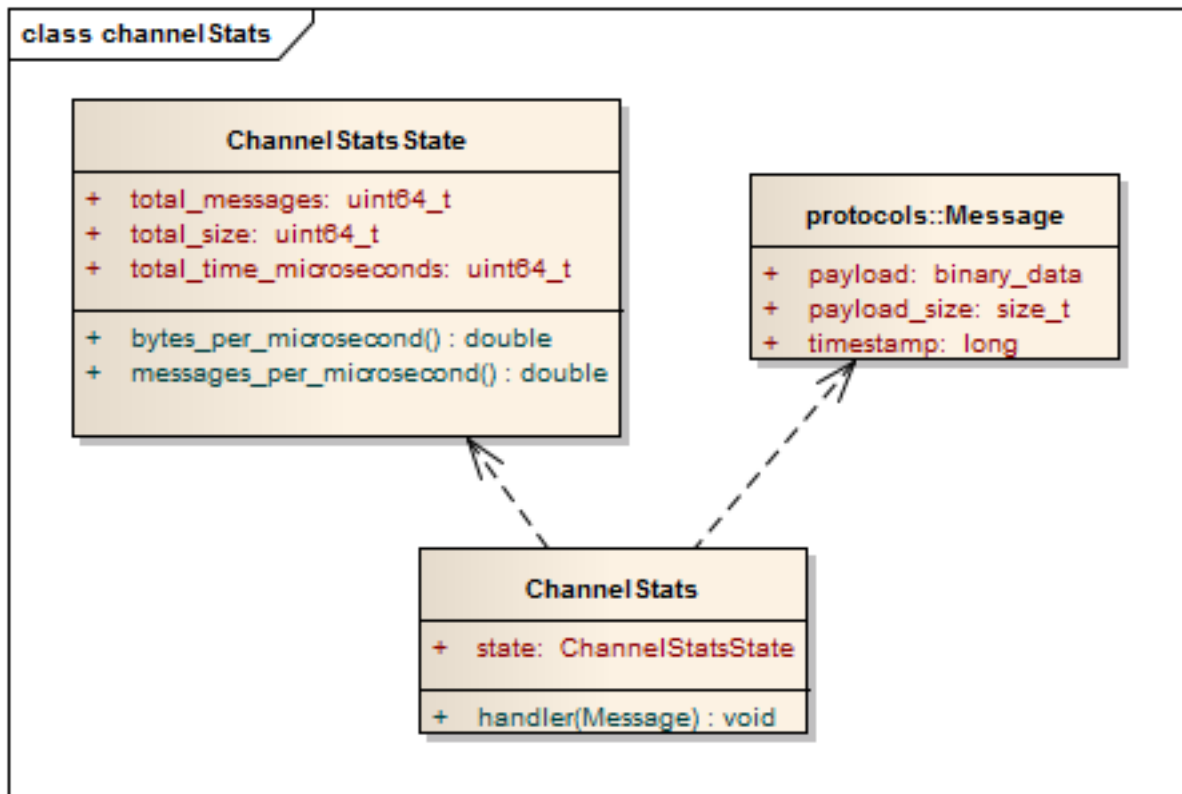


Рис. 8: Диаграмма классов модуля channelStats

- ChannelStats — класс для подсчёта статистики
 - Атрибуты:
 - * state: ChannelStatsState — контейнер для хранения статистики
 - Методы:
 - * handler(Message): void — обработчик сообщений; подсчитывает статистику

Показатели качества структуры классов (не учитывая классы из других модулей):

- Степень абстрактности: $A = \frac{\text{Абстрактные классы}}{\text{Все классы}} = \frac{0}{2} = 0$

4.4.1.1.4. listener

Реализует прослушку сетевого трафика. Для данного компонента используется шаблон Фасад, для обеспечения независимости модуля от подсистемы, реализующей прослушку сетевого трафика (библиотека libtins, которая также является Фасадом библиотеки libpcap).

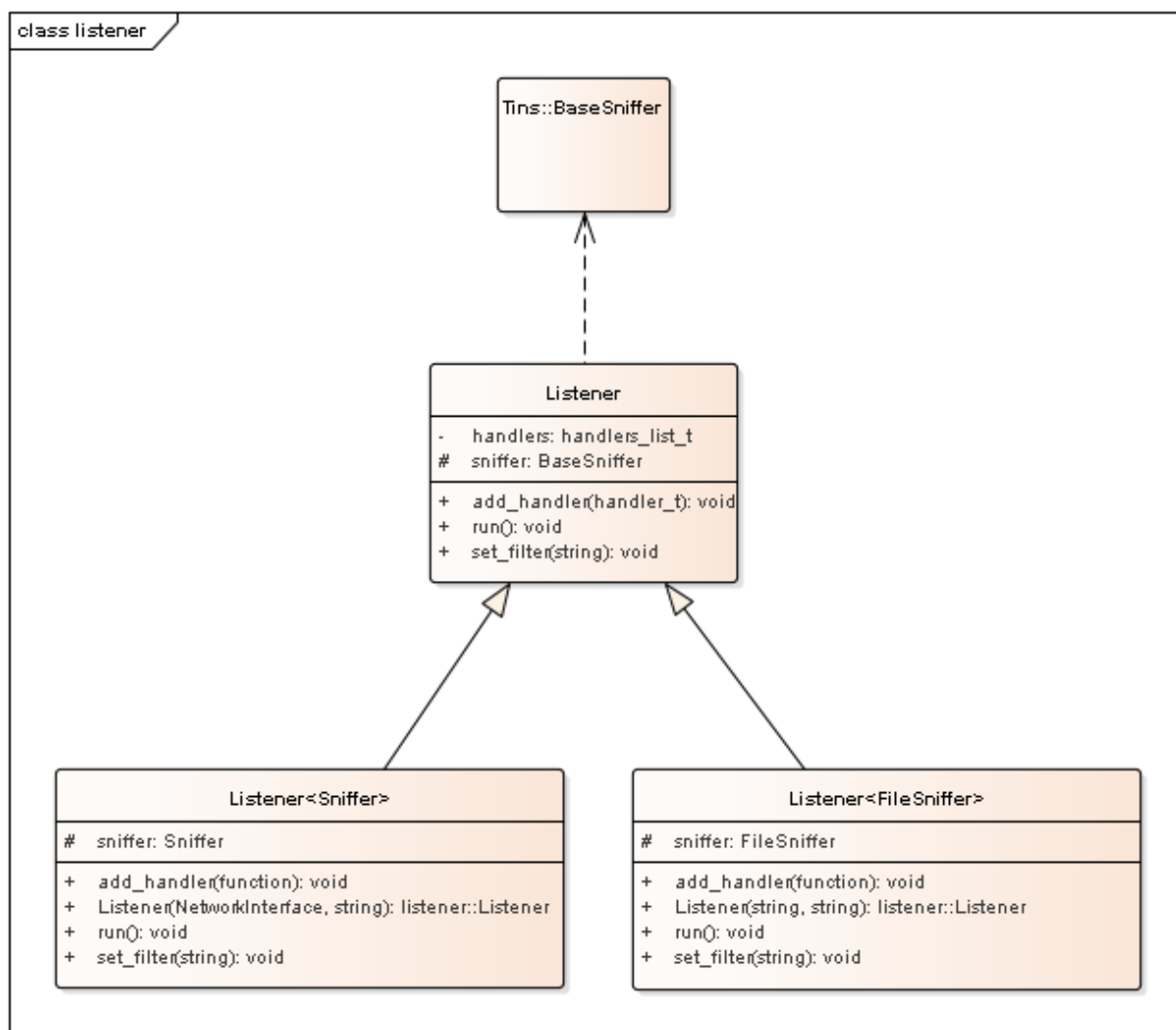


Рис. 9: Диаграмма классов модуля listener

- `Tins::Packet` — класс, представляющий сетевой пакет
- `Tins::BaseSniffer` — интерфейс прослушки сетевого трафика
- `Tins::Sniffer` — прослушка сетевого трафика
- `Tins::FileSniffer` — чтение сетевого трафика из предварительного записанного файла
- `Listener` — шаблонный класс, предоставляющий интерфейс для прослушки сетевого трафика; имеет 2 специализации — `Listener<Tins::Sniffer>` и `Listener<Tins::FileSniffer>` для работы с реальным и предварительно записанным в файл сетевым трафиком соответственно
 - Атрибуты:
 - * `handlers: handlers_list_t` — список обработчиков захваченных пакетов
 - * `sniffer: BaseSniffer` — объект прослушки
 - Методы:
 - * `add_handler(handler_t): void` — добавление обработчика захваченных пакетов
 - * `set_filter(string): void` — задание фильтра полученных пакетов; фильтр описывается синтаксисом BPF, например: `ip and udp and dst port 1000`
 - * `run(): void` — запуск прослушки

Показатели качества структуры классов (не учитывая классы из других модулей):

- Степень абстрактности: $A = \frac{\text{Абстрактные классы}}{\text{Все классы}} = \frac{0}{3}$

4.4.1.1.5. serialization

Модуль сериализации реализует сериализацию и десериализацию структур данных. Сериализация производит перевод структуры данных в последовательность битов; десериализация является обратной операцией. Сериализация применяется при передаче по сети или при сохранении данных на диск в виде, независимым от архитектуры системы (например, независимо от порядка байт). Также сериализация позволяет производить обмен данными между программами, использующими разные языки программирования (например, C++ и JavaScripts).

В рассматриваемой системе (СПУ), для реализации компонентов могут быть использованы различные языки программирования; для передачи данных между компонентами следует использовать *универсальный* формат данных, с которым могут работать все компоненты, независимо от конкретной реализации (архитектуры системы, используемого языка программирования, т.п.).

В приведённой диаграмме (рис. 10) показана схема сериализации данных в формат JSON. Модуль сериализации зависит от библиотеки `json` и структур, сериализация которых будет производиться. Конкретный формат для сериализации не влияет на архитектуру модуля и компонента в целом.

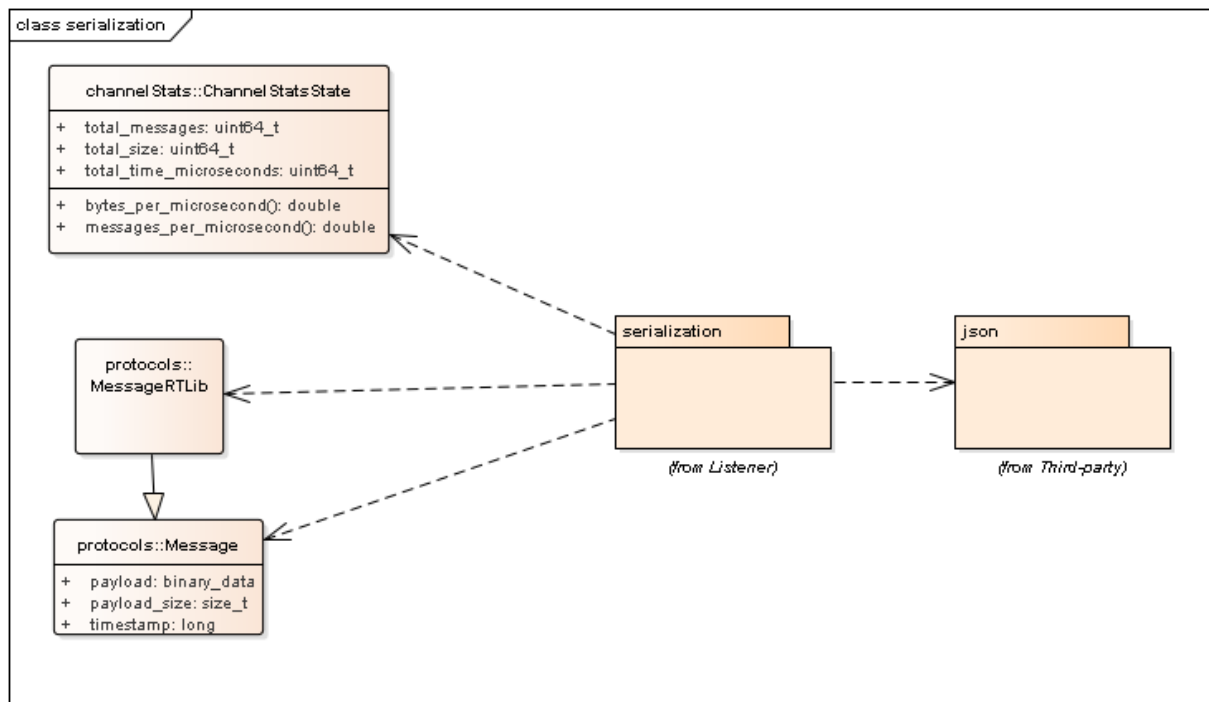


Рис. 10: Диаграмма классов модуля serialization

Показатели качества структуры классов (не учитывая классы из других модулей):

- Степень абстрактности: $A = \frac{\text{Абстрактные классы}}{\text{Все классы}} = \frac{0}{1} = 0$

4.4.1.1.6. registration

Реализует регистрацию данных, полученных в процессе работы компонента. Для реализации данного компонента используется порождающий шаблон “Одиночка” (Singleton): так как модуль обеспечивает регистрацию данных конкретного экземпляра программы, он должен существовать так же в единственном экземпляре.

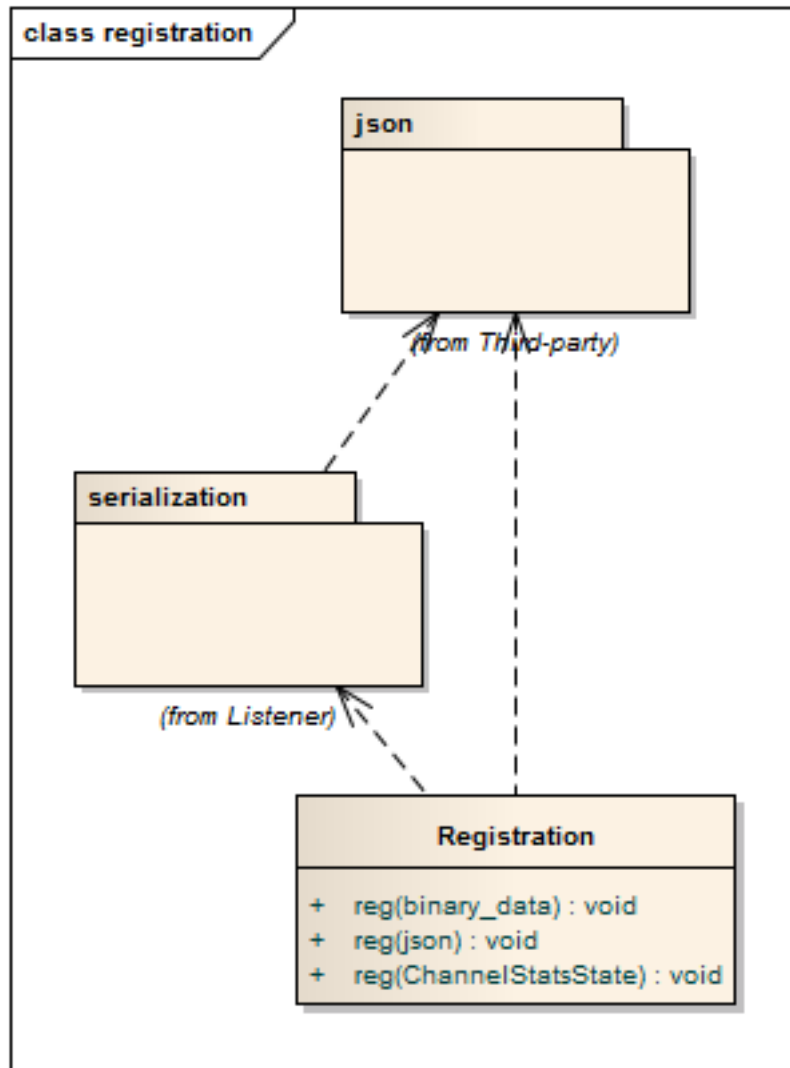


Рис. 11: Диаграмма классов модуля registration

Показатели качества структуры классов (не учитывая классы из других модулей):

- Степень абстрактности: $A = \frac{\text{Абстрактные классы}}{\text{Все классы}} = \frac{0}{1} = 0$