Ewaluacja narzędzia Apache Kafka jako brokera komunikacji w projekcie Platom

(Tomasz Jaworski, Tomasz Kowalski, Radosław Wajman, Andrzej Romanowski, Piotr Łuczak)

W celu przeprowadzenia ewaluacji możliwości wykorzystania narzędzia Apache Kafka[1] zebrano informację dotyczące przewidywanych wielkości pakietów danych generowanych przez poszczególne planowane komponenty systemu. Tabela poniżej przedstawia zebrane przewidywania.

Tabela 1. Przewidywane typy komunikacji w ramach systemu PLATOM

Nazwa kanału brokera Kafka	Opis	Rozmiar komunikatu [kB]	Częstotliwość [ramka/sec]	Liczba odbiorców	Liczba nadawców	Czas ważności danych [sec]	Reakcja na zgubienie jednej ramki	Reakcja na zanik strumienia	Metadane
ET3	Dane pomiarowe z ET3	(496 x 32bit) ~ 2 KB **float**	12	1 .. 4	1	0,5	pomijalne	konieczne do przywrócenia	• liczba elektrod czujnika **INT** • rozmiar wektora z danymi **INT** • oznaczenie modelu tomografu **CHAR[32]/String** • znacznik czasowy **long long (64bits)** • czy dane znormalizowane **Bool / INT/CHAR**

[1] https://kafka.apache.org/

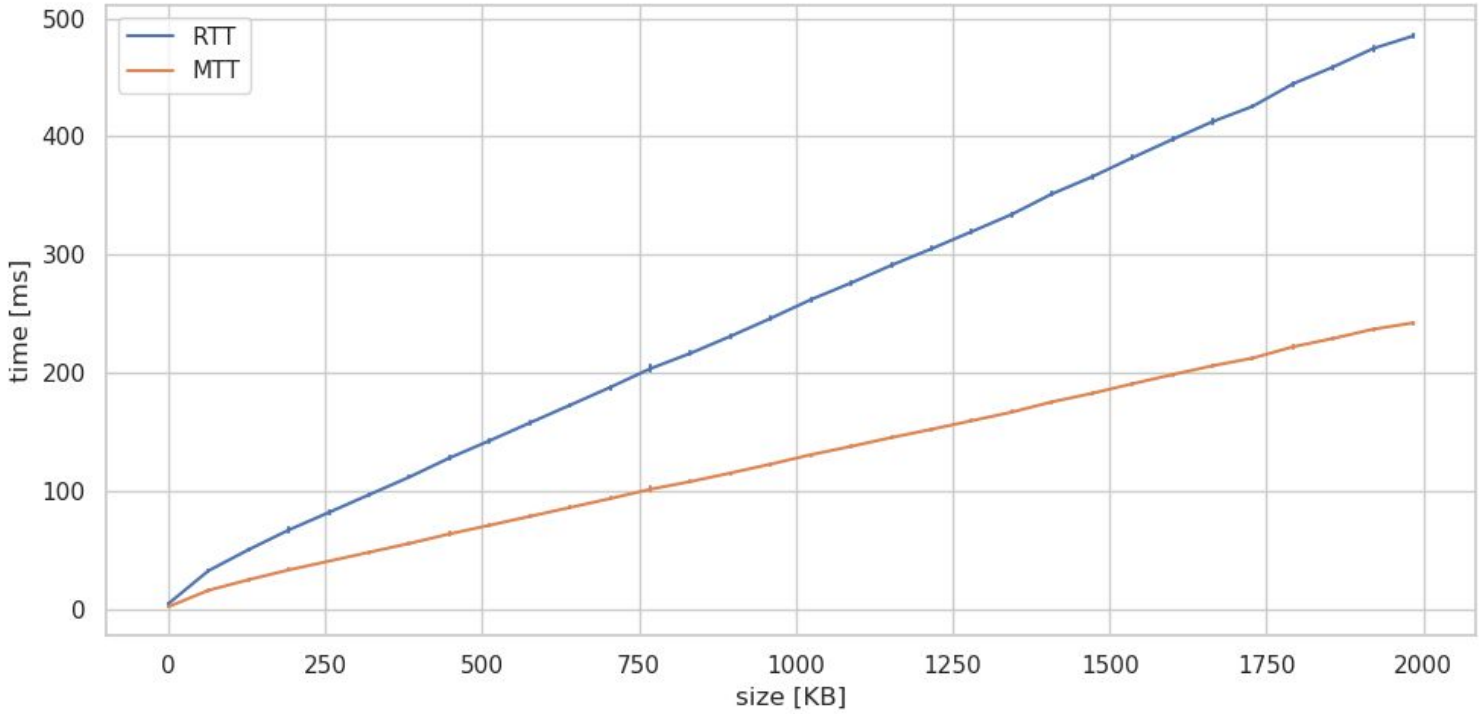
ImgLow	Zrekonstruowany obraz 3D ECT małej gęstości	(8500 x 32bit)
		~ 34 KB
		
		float
			12	1 .. 4	1	0,5	pomijalne	konieczne do przywrócenia	• liczba voxeli
									• liczba simpleksów
									• znacznik czasowy ramki pomiarowej, dla której zrekonstruowany obraz
									• algorytm rekonstrukcji
									  ○ liczba iteracji
									  ○ parametr relaksacji/regular.
									  ○ tresholding
									  ○ aktualizacja wrażliwości
									  ○ inne ???
									• czas rekonstrukcji ???
									• CPU / GPU

ImgHigh	Zrekonstruowany obraz 3D ECT dużej gęstości	(160000 x 32bit) ~ 640 KB **float**	12	1 .. 4	1	0,5	pomijalne	konieczne do przywrócenia	j.w.
CCD	Dane pomiarowe z kamery	4 KB	60	1 .. 4	1	0,5	pomijalne	konieczne do przywrócenia	• znacznik czasowy **long long (64bits)** • górna szerokość swirl-a • dolna szerokość swirl-a • wysokość swil-a • prędkość obiektu wewnątrz swirl-a
SPGas	Nastawy przepływu gazu	4 B	1 .. 12	1	1 .. 2	0,5	ważne można zadbać o potwierdzenie dostarczenia	konieczne do przywrócenia	
SPPump	Nastawy pracy pompy	4 B	1 .. 12	1	1 .. 2	0,5	ważne można zadbać o potwierdzenie dostarczenia	konieczne do przywrócenia	
SPValve	Komunikat zmiany stanu zaworu	4 B	1 .. 12	1	1 .. 2	0,5	ważne można zadbać o potwierdzenie dostarczenia	konieczne do przywrócenia	
Time	Znacznik czasowy	(2 x 64bit) **long long**	jednorazowo dla startującego modułu	???	1	1	powtórzenie do otrzymania odpowiedzi	powtarzanie	

Analiza komunikacji usług w systemie ze scentralizowanym brokerem (kafka):

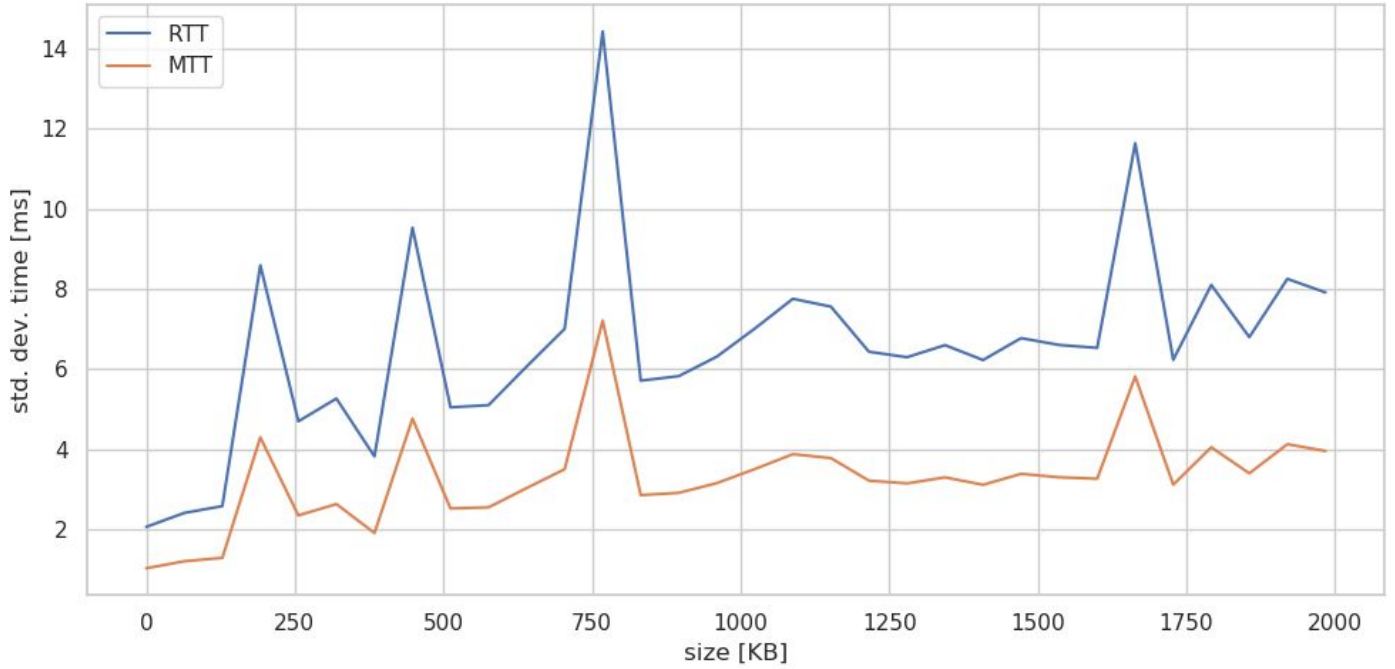
Schemat komunikacji:

Komputer A nadaje komunikat ze znacznikiem czasu do brokera komunikatów. Broker przesyła kopię komunikatu z powrotem. Komputer A sprawdza różnicę pomiędzy swoim czasem a czasem z komunikatu (wartość RTT). Uzyskany rezultat przedstawiono na Rys. 1 i 2.



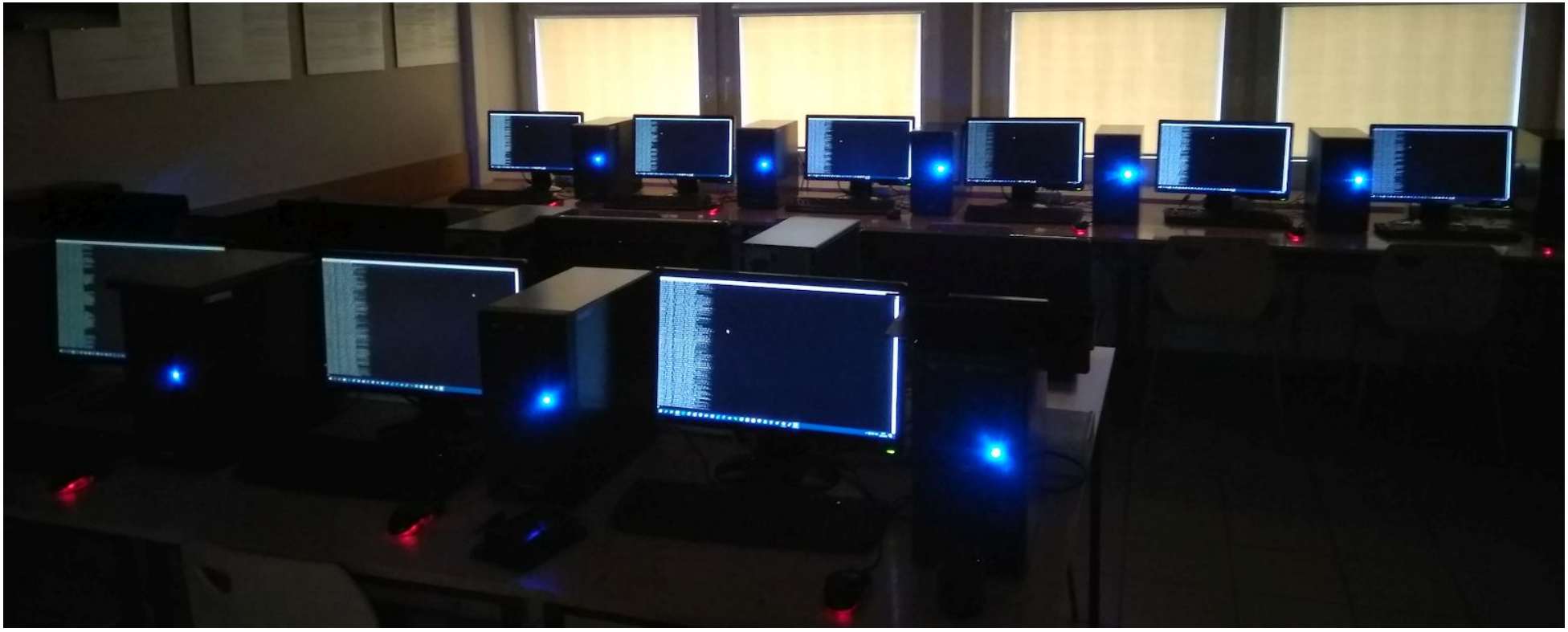
Rys 1. Czas transmisji komunikatów w funkcji ich długości
Legenda: RTT - round trip time (czas transmisji do oraz od brokera), MTT - średnia obu tych czasów.

Komunikat każdej długości przesłano 100 razy, wyznaczając wartość średnią RTT (rys 1) i odchylenie standardowe (rys 2).



Rys 2. Odchylenie standardowe czasu RTT w funkcji długości komunikatu

Analiza komunikacji usług w systemie ze scentralizowanym brokerem (kafka) i wieloma kolejkami:



Schemat komunikacji:

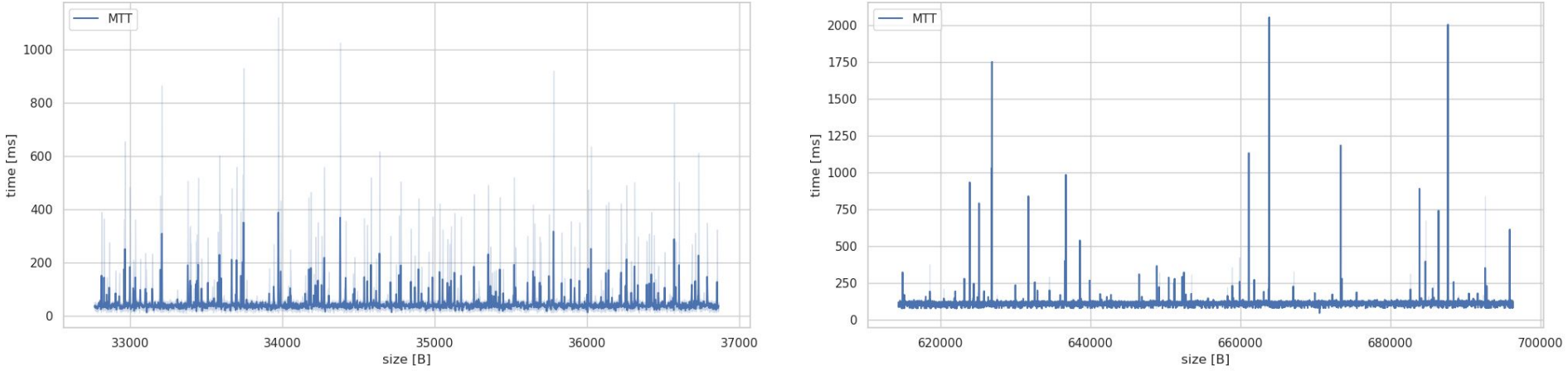
Cztery komputery (sala 311) nadają komunikaty, symulując planowane częstotliwości i wielkości paczek danych ze znacznikiem czasu do brokera komunikatów. Każda z przewidywanych kombinacji wielkości i częstotliwości jest testowana na każdej z maszyn. Schemat ten nieco odbiega od docelowego charakteru komunikacji - stanowi jego intensywniejszą odmianę.

Charakter indywidualnej komunikacji danego komputera z brokerem jest realizacją typowego pomiaru czasu transmisji. Komputer - nadawca - wysyła komunikat ze znacznikiem czasu wysłania. Broker komunikatów odbiera go a następnie, wykorzystując swoje wewnętrzne mechanizmy, zwraca do nadawcy. Nadawca ostatecznie porównuje otrzymany znacznik czasowy i porównujego ze swoim

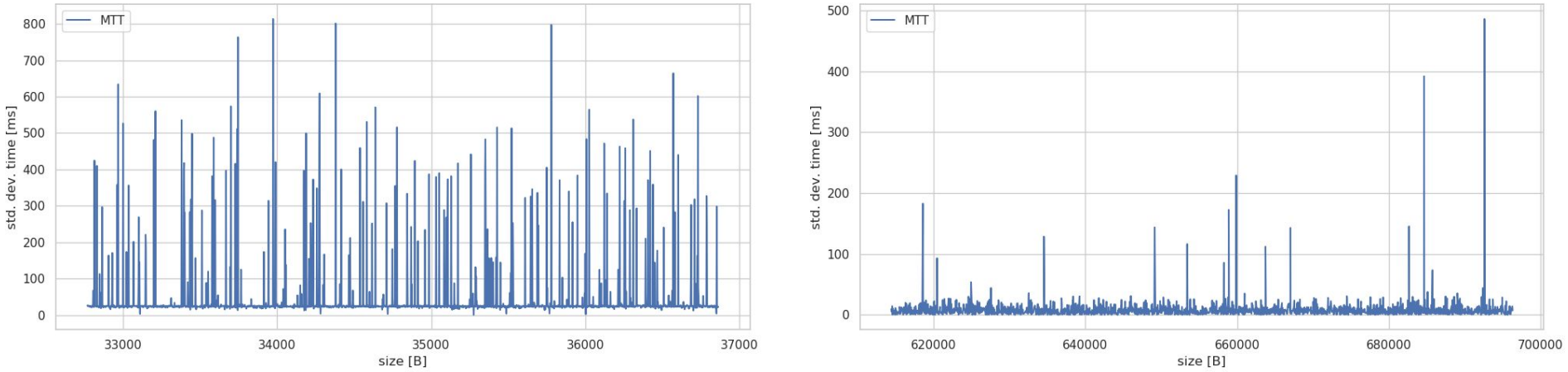
aktualnym czasem, wyznaczając róznicę. Różnica ta to czas transmisji do i z brokera RTT (ang. *round trip time*) a średnia czas transmisji (ang. *mean trip time*, MTT) to połowa tego czasu.

Uzyskane wyniki

Czas eksperymentu wynosił 10 minut. W ramach tego czasu uruchomionych było siedem kanałów komunikacji o ruchu odpowiadającemu temu z tabeli 1. Średni czas komunikacji dla tych kanałów przedstawiono na rys. 3 a dodatkowo odchylenie standardowe czasu komunikacji przedstawiono na rys. 4.



Rys 3. Średni czas komunikacji w funkcji wielkości pakietu dla kanału **ImgLow** oraz **ImgHigh** (Tab. 1)



Rys 4. Odchylenie standardowe średniego czasu komunikacji w funkcji wielkości pakietu dla kanału **ImgLow** oraz **ImgHigh** (Tab. 1)

Tab 2. Statystyka komunikacji czterech maszyn testowych i jednego węzła Kafka w ramach przeprowadzonego eksperymentu

Nazwa kanału komunikacji	Liczba pomiarów	Średnia wielkość pakietu [B]	Odchylenie standardowe średniej wielkości pakietu	Średni czas transmisji (MTT) [ms]	Czas ważności danych [ms]	Odchylenie standardowe średniego czasu transmisji
ET3	25577	2048	0	40.91	**500**	96.67
ImgLow	25452	34811	1182.09	44.71	**500**	96.79
ImgHigh	15242	655238.92	23727.79	112.08	**500**	36.91
SPGas	4385	4	0	63.48	**500**	98.82
SPPump	4339	4	0	62.90	**500**	94.88
SPValve	4360	4	0	63.34	**500**	98.22
Time	246	16	0	73.85	**1000**	188.05

Kod źródłowy (GIT https://gitlabplatom.cti.p.lodz.pl/platom/kafka-dotnet-core):

```
using KafkaNet;
using KafkaNet.Model;
using KafkaNet.Protocol;
using Newtonsoft.Json;
using System;
using System.Collections;
using System.Collections.Generic;
using System.Diagnostics;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading;

namespace ConsoleApp1
{
    class Content
    {
        public string Text { get; set; }
        public DateTime Timestamp { get; set; }
        public int ID { get; set; }
    }

    delegate void MeasurementAcquiredDelegate(string experiment_name, int size, double mtt);

    class PingerEntity
    {
        private Random rng;
        private string name;
        private BrokerRouter producer_connection_router;
        private ConsumerOptions consumer_options;

        private CancellationTokenSource producer_token_source;
        private CancellationTokenSource consumer_token_source;

        private Thread producer_thread;
        private Thread consumer_thread;

        public MeasurementAcquiredDelegate OnMeasurement { get; set; }

        public int DataLow { get; set; }
        public int DataHigh { get; set; }
```

```
public double FrequencyLow { get; set; }
public double FrequencyHigh { get; set; }

public string Topic { get; set; }

public PingerEntity(string experimentName, int dataLow, int dataHigh, double freqLow, double freqHigh, Random rng)
{
    this.rng = rng ?? new Random();
    this.name = experimentName;

    this.producer_token_source = new CancellationTokenSource();
    this.consumer_token_source = new CancellationTokenSource();

    this.DataLow = dataLow;
    this.DataHigh = dataHigh;
    this.FrequencyHigh = freqHigh;
    this.FrequencyLow = freqLow;

    this.SetupKafkaConnection();
}

private void SetupKafkaConnection()
{
    this.Topic = $"topic_" + Guid.NewGuid().ToString();

    try
    {
        KafkaOptions producerConnectionOptions = new KafkaOptions(new Uri("http://212.191.89.18:9092"));
        this.producer_connection_router = new BrokerRouter(producerConnectionOptions);

        KafkaOptions consumerConnectionOptions = new KafkaOptions(new Uri("http://212.191.89.18:9092"));
        BrokerRouter consumerConnectionRouter = new BrokerRouter(consumerConnectionOptions);
        this.consumer_options = new ConsumerOptions(this.Topic, consumerConnectionRouter);
        this.consumer_options.MinimumBytes = 1;
        this.consumer_options.MaxWaitTimeForMinimumBytes = new TimeSpan(0, 0, 0, 5);
    }
    catch (Exception ex)
    {
        //
        Console.WriteLine($"{this}: {ex.Message}");
    }
}

private void ProducerThread()
```

```
{
    CancellationToken ct = this.producer_token_source.Token;
    int id = 0;
    byte[] payload = new byte[this.DataHigh];
    rng.NextBytes(payload);

    using (Producer producer = new Producer(this.producer_connection_router))
        while (!ct.IsCancellationRequested)
        {
            // Losuj częstotliwość oraz dane
            int interval_high = (int)Math.Round(1000.0 / this.FrequencyHigh);
            int interval_low = (int)Math.Round(1000.0 / this.FrequencyLow);

            int interval = this.rng.Next(interval_high, interval_low + 1);
            int length = this.rng.Next(this.DataLow, this.DataHigh + 1);

            // Przygotuj blok danych
            Content content = new Content() { ID = id++, Timestamp = DateTime.Now };
            content.Text = String.Join("", payload.Take(length).Select(bt => bt.ToString("X2")));

            // Serializuj i wyślij
            string jcontent = JsonConvert.SerializeObject(content);
            Console.WriteLine($"{this} ID={content.ID}: Wysyłanie {jcontent.Length} bajtów, przerwa={interval} ms... ");
            producer.SendMessageAsync(this.Topic, new[] { new Message(jcontent) });//.Wait();
            Console.Out.Flush();

            // No i czekaj
            Thread.Sleep(interval);
        }
}

private void ConsumerThread()
{
    CancellationToken ct = this.consumer_token_source.Token;
    IEnumerator<Message> message_source = null;

    using (Consumer consumer = new Consumer(this.consumer_options))
        while (!ct.IsCancellationRequested)
        {
            // Odbierz oczekujący komunikat tak szybko, jak tylko się pojawi
            if (message_source == null)
                message_source = consumer.Consume().GetEnumerator();

            bool result = message_source.MoveNext();
```

```
            Debug.Assert(result, "Ale że jak to???");

            // Deserializuj
            Content content = JsonConvert.DeserializeObject<Content>(Encoding.UTF8.GetString(message_source.Current.Value));

            // Wyświetl czas
            // MTT - mean trip time
            // RTT - round trip time
            TimeSpan delta = DateTime.Now - content.Timestamp;
            double mtt = delta.TotalMilliseconds / 2.0;
            Console.WriteLine($"{this} ID={content.ID}: MTT={mtt:N3}, RTT={(delta.TotalMilliseconds):N3}");
            Console.Out.Flush();

            if (this.OnMeasurement != null)
                this.OnMeasurement(this.name, content.Text.Length, mtt);

        }
}

public void Start(MeasurementAcquiredDelegate measDelegate)
{
    this.OnMeasurement = measDelegate;

    //
    this.producer_thread = new Thread(new ThreadStart(ProducerThread));
    this.consumer_thread = new Thread(new ThreadStart(ConsumerThread));

    // Do dzieła
    this.producer_thread.Start();
    this.consumer_thread.Start();
}

public void Wait(int experimentTime)
{
    // Czekaj zadany czas i kończ zabawę
    Thread.Sleep(experimentTime * 1000);
    this.Terminate();
}

public void Terminate()
{
    // Zakończ wątki, NAJPIERW konsument, POTEM producent
    this.consumer_token_source.Cancel();
    if (this.consumer_thread.IsAlive)
```

```
            this.consumer_thread.Join();

        this.producer_token_source.Cancel();
        if (this.producer_thread.IsAlive)
            this.producer_thread.Join();
    }

    public override string ToString() => $"[{this.name}]";
}


class Program
{
    static void Main(string[] args)
    {
        Console.WriteLine("Aktualny czas: " + DateTime.Now.ToString());
        Console.WriteLine("Podaj czas startu eksperymentu (w formie HH:MM): ");
        string str = Console.ReadLine();
        DateTime start_time = DateTime.Parse(str);

        TimeSpan delta;
        do
        {
            delta = start_time - DateTime.Now;
            Console.WriteLine($"Czekam; zostało {Math.Max(delta.TotalSeconds,0.0):N0} sekund...");
            Thread.Sleep(1000);
        } while (delta.TotalSeconds > 0);

        Console.WriteLine("Do dzieła!");

        FileStream timeFile = File.Create("kafkaTime.csv");
        StreamWriter timeFileWriter = new StreamWriter(timeFile);
        timeFileWriter.WriteLine("name,payload,MTT");

        MeasurementAcquiredDelegate mad = new MeasurementAcquiredDelegate(
            (string expName, int contentSize, double meanTripTime) =>
        {
            timeFileWriter.WriteLine($"\"{expName}\",{contentSize},{meanTripTime}");
            timeFileWriter.Flush();
        });

        // Przygotuj eksperyment
        Random rnd = new Random();
        PingerEntity[] pingers = new PingerEntity[] {
```

```
                new PingerEntity("ET3", 1024, 1024, 12, 12, rnd),
                new PingerEntity("ImgLow", 16*1024, 18*1024, 12, 12, rnd),
                new PingerEntity("ImgHigh", 300*1024, 340*1024, 12, 12, rnd),
                new PingerEntity("SPGas", 2,2, 1, 12, rnd),
                new PingerEntity("SPPump", 2,2, 1, 12, rnd),
                new PingerEntity("SPValve", 2,2, 1, 12, rnd),
                new PingerEntity("Time", 8,8, 0.1, 0.1, rnd)
            };

            // Uruchom go
            foreach (PingerEntity pe in pingers)
                pe.Start(mad);

            // Poczekaj 10 minut
            Thread.Sleep(10 * 60 * 1000);

            // I ubij
            foreach (PingerEntity pe in pingers)
                pe.Terminate();

            timeFileWriter.Close();
        }
    }
}
```

Protokoły komunikacyjne rozproszonego systemu PLATOM (wersja robocza)

Komunikacja sieciowa urządzeń kontrolno-pomiarowych w systemie PLATOM wykorzystuje centralny węzeł komunikacyjny w postaci brokera komunikatów Apache Kafka[2], działającego w trybie publikacji/subskrypcji. Oznacza to, że każda usługa publikuje swoje komunikaty, bez wskazania konkretnego odbiorcy. Natomiast to odbiorca decyduje, której usługi nadającej chce słuchać (subskrybować).

Komunikaty przesyłane są w formie wiadomości tekstowych, zorganizowanych w hierachiczną strukturę dokumentową z notacją obiektową, zapisaną w formacie JSON[3] (ang. *JavaScript Object Notation*). Każdy z komunikatów wskazuje na konkretny schemat walidacyjny (ang. schema), co daje możliwość walidacji struktury oraz zawartości przesyłanych komunikatów. Podejście takie pozwala na dowolną rozbudowę warstwy sprzętowej (kontrolno/pomiarowo/wykonawczej) o urządzenią i rozwiązania nie znane na etapie projektowania.

Zestawienie protokołów w formie struktur JSON, do wykorzystania w projekcie PLATOM. Poniżej przedstawiono podstawowe założenia oraz terminologię systemu komunikacji:

Komunikacja

- Podstawową jednostką wymiany danych jest **komunikat**.
- Podstawową platformą przekazywania danych jest **kanał**.
- Operację wysyłania/generowania komunikatów realizuje **nadajnik**.
- Operację odbierania/przetwarzania danych realizuje **odbiornik**.
- Do jednego kanału może być podpiętych **dowolnie wiele odbiorników** oraz tylko **jeden nadajnik** za wyjątkiem kanałów:
    - **status**
    - **log_trace**, **log_debug**, **log_info**, **log_warning**, **log_error**, **log_failure**.
- Kanałem statusowym całego systemu jest *status*, w którym poszczególne usługi informują o swojej dostępności.

Treść komunikatów

- Komunikat przesyłany jest w formie struktury JSON.
- Komunikat jest obiektem z wyłącznie dwoma polami *sequence* oraz *payload*.
- Struktura pola *sequence* jest stała a *payload* zależna od przeznaczenia komunikatu.
- Struktura komunikatu zależy od nadajnika/usługi. Ponieważ wyłącznie jeden nadajnik może generować komunikaty w ramach jednego kanału, to można przyjąć iż *nazwa kanału determinuje strukturę komunikatu*.
- Struktura komunikatu danego nadajnika/usługi musi posiadać opisujący ją schemat.

[2] https://kafka.apache.org/
[3] https://www.json.org/json-en.html

- Struktura pola *payload* jest opisana danym schematem.

Schematy treści komunikatów

- Schemat jest plikiem (tekstem) w formacie JSON z kodowaniem UTF8.
- Schemat musi być znany wszystkim odbiornikom zainteresowanym analizą/wczytywaniem odpowiadających mu komunikatów.
- Komunikat niezgodny ze schematem w jakimkolwiek zakresie należy uważać za uszkodzony i ignorować.

Usługi

- Komunikaty przesyłane są między **usługami**.
- Usługa może realizować **dowolnie wiele** nadajników i odbiorników.
- Każda usługa cyklicznie informuje o swojej obecności i swoich nadajnikach, wysyłając komunikat kanałem **status**.
- Brak cyklicznego informowania o obecności usługi oznacza brak jej dostępności.
- Każda usługa identyfikowana jest przez swoją **unikalną nazwę**.

Struktura komunikatów przesyłanych w systemie PLATOM

Poniżej przedstawiono przykładowy komunikat, przesyłany między działającymi usługami:

```
{
  "sequence": {
     "timestamp": "2009-02-15T00:00:00Z",
     "number": 1234,
     "channel": "et3_measurements",
     "schema": "et3_schema"
  },

  "payload": {
    .....
  }
}
```

Struktura pola ***sequence*** jest **obowiązkowa dla wszystkich usług/nadajników/odbiorników** pracujących w. Posiada ona identyczną strukturę, niezależną od nadajnika i usługi, generującej dany komunikat. Zadaniem struktury jest dostarczanie informacji o: a) źródle komunikatu, b) czasie jego wygenerowania, oraz c) metodzie jego walidacji.

Struktura pola *sequence* jest następująca:

- Pole ***sequence/timestamp*** zawiera znacznik czasowy uzyskania danych, będących przedmiotem transmisji (lub znacznikiem czasowym chwili wygenerowania komunikatu).
    - Przykładowo dla pomiaru temperatury na znacznik czasowy określa punkt w czasie, w którym realizowano ów pomiar. Oczywiście często podanie precyzyjnie czasu pomiaru nie będzie możliwe, choćby ze względu na sposób pomiaru lub czas jego trwania (np. tomograf). Należy wtedy zadbać, aby punkt czasowy był stały względem samego pomiaru, np. mierzony zaraz przed albo zaraz po pomiarze.
    - Znacznik zapisany jest w zgodnie ze standardem ISO 8601 (patrz typ danych `timestamp`).
- Pole ***sequence/number*** zawiera numer kolejnego komunikatu, liczony od startu usługi nadającej komunikaty w ramach danego kanału. Dopuszczalne są wartości jedynie większe od 0 (***number*** > 0). Wartość 0 jest zarezerwowana.
- Pole ***sequence/channel*** zawiera nazwę kanału w ramach którego komunikat zawierający daną sekwencję jest przesyłany. Ta redundantność (nazwa kanału, którym dostarczono dany komunikat == zawartość ***sequence/channel***) powinna ułatwić późniejsze uruchomienie/debugowanie systemu oraz logowanie przebiegu eksperymentów.
    - **Uwaga**: nazwa kanału jest ograniczona ze względu na wykorzystanie systemu Kafka - musi być zgodna z następującym wzorcem: “[A-Za-z0-9._-]”.
- Pole ***sequence/schema*** zawiera unikalną nazwę schematu walidacyjnego, pozwalającego na walidację zawartości komunikatu oraz na jej opisanie (np. w celach uruchomieniowych). Pole może mieć wartość ***null***. Oznacza ona brak schematu struktury komunikatu, a więc brak możliwości walidacji. Do stosowania jedynie w celach diagnostyczno/uruchomieniowych.
- Pole ***sequence/service*** zawiera nazwę usługi odpowiadającej za wygenerowanie danego komunikatu.

Stan dostępnych usług

Każda usługa przesyła cyklicznie komunikat kanałem **status** do innych usług, informując o aktywności swojej jak i kanałów na których ona nadaje. Poniżej przykład dla tomografu:

```
{
  "sequence": {
     "timestamp": "2019-07-29T14:35:21Z",
     "number": 123456,
     "channel": "status",
     "service": "ET3",
     "schema": "schema_status"
  },

  "payload": {
```

```
    "service": "ET3",
    "publishes": [ "et3_measurements","et3_messages" ],
    "subscribes": [ "et3_gains" ],
    "next_alive_interval": 3000,
    "timeout": 20000
  }
},
```

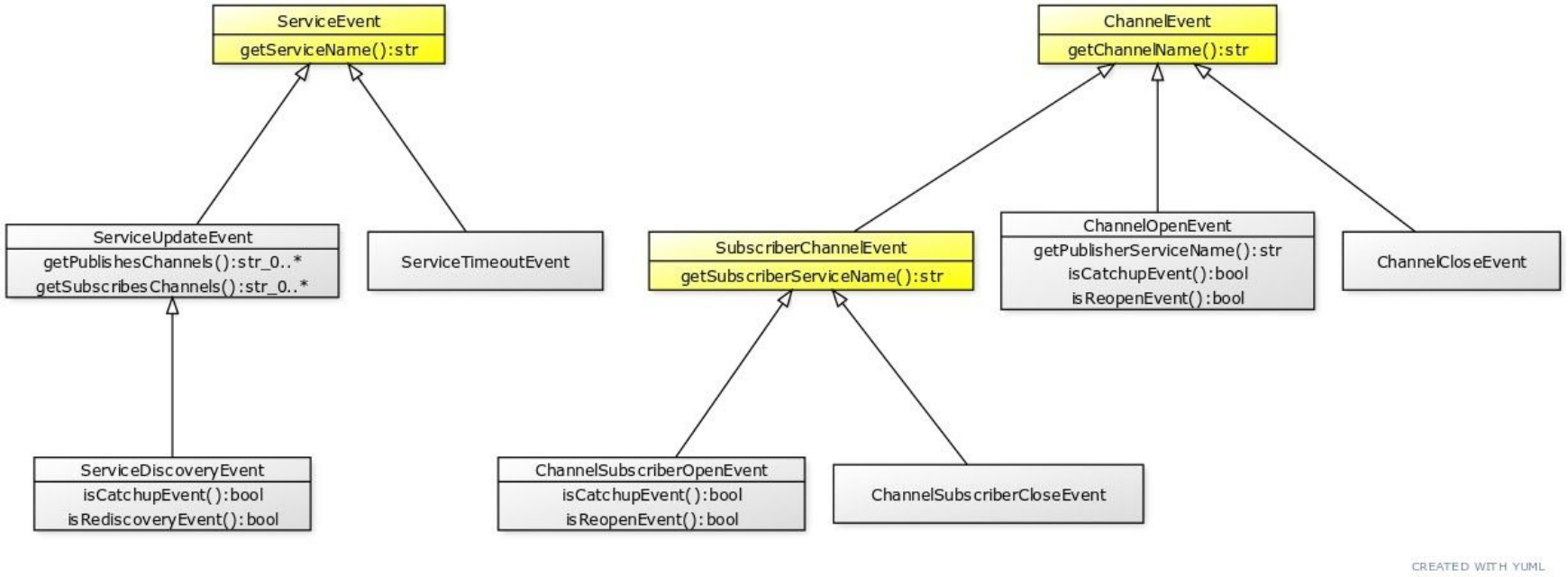
Struktura komunikatu jest stała, zgodna ze schematem o predefiniowanej nazwie **schema_status** i zawiera następujące pola:

- Podstruktura ***sequence*** została opisana powyżej.
- Pole ***payload/service*** przechowuje nazwę/identyfikator usługi (wartość unikalna w skali całego systemu).
- Pole ***payload/publishes*** przechowuje tablicę nazw kanałów, dla których dana usługa jest nadajnikiem.
- Pole ***payload/subscribes*** przechowuje tablicę nazw kanałów na których dana usługa nasłuchuje.
- Pola ***payload/next_alive_interval*** oraz ***payload/timeout*** pozwalają monitorować aktywność usługi ***service*** każdemu z odbiorników. Ich wartość określona jest w milisekundach i ma następujące znaczenie:
    - Pole ***payload/next_alive_interval*** zawiera opóźnienie względem czasu ***sequence/timestamp,*** po jakim usługa ***service*** wyśle kolejny komunikat kanałem *status*.
    - Pole ***payload/timeout*** zawiera czas względem ***sequence/timestamp***, po jakim usługę ***service*** można uznać za martwą.

W powyższym przykładzie usługa **ET3** informuje, że nadaje komunikaty (z dowolną częstotliwością) kanałami **et3_measurements** oraz **et3_messages** równocześnie nasłuchując komunikatów przychodzących kanałem **et3_gains.** Komunikat został wysłany o godzinie **14:35:21** z obietnicą, że kolejny zostanie wysłany o najpóźniej o godzinie **14:35:24**. Ponadto, jeżeli do godziny **14:35:41** usługa **ET3** nie prześle żadnego komunikatu kanałem *status*, to wszystkie pozostałe odbiorniki mogą uznać ją za nieaktywną.

Monitoring kanałów komunikacji

Mechanizmy, których cele wymagają monitoringu całej komunikacji w ramach platformy, mogą być zaimplementowane w oparciu wzorzec obserwator. Zdarzenia związane z komunikacją platformy można przedstawić w następujących hierarchiach: zdarzeń dot. usług i zdarzeń dot. kanałów (patrz Rys. 5)

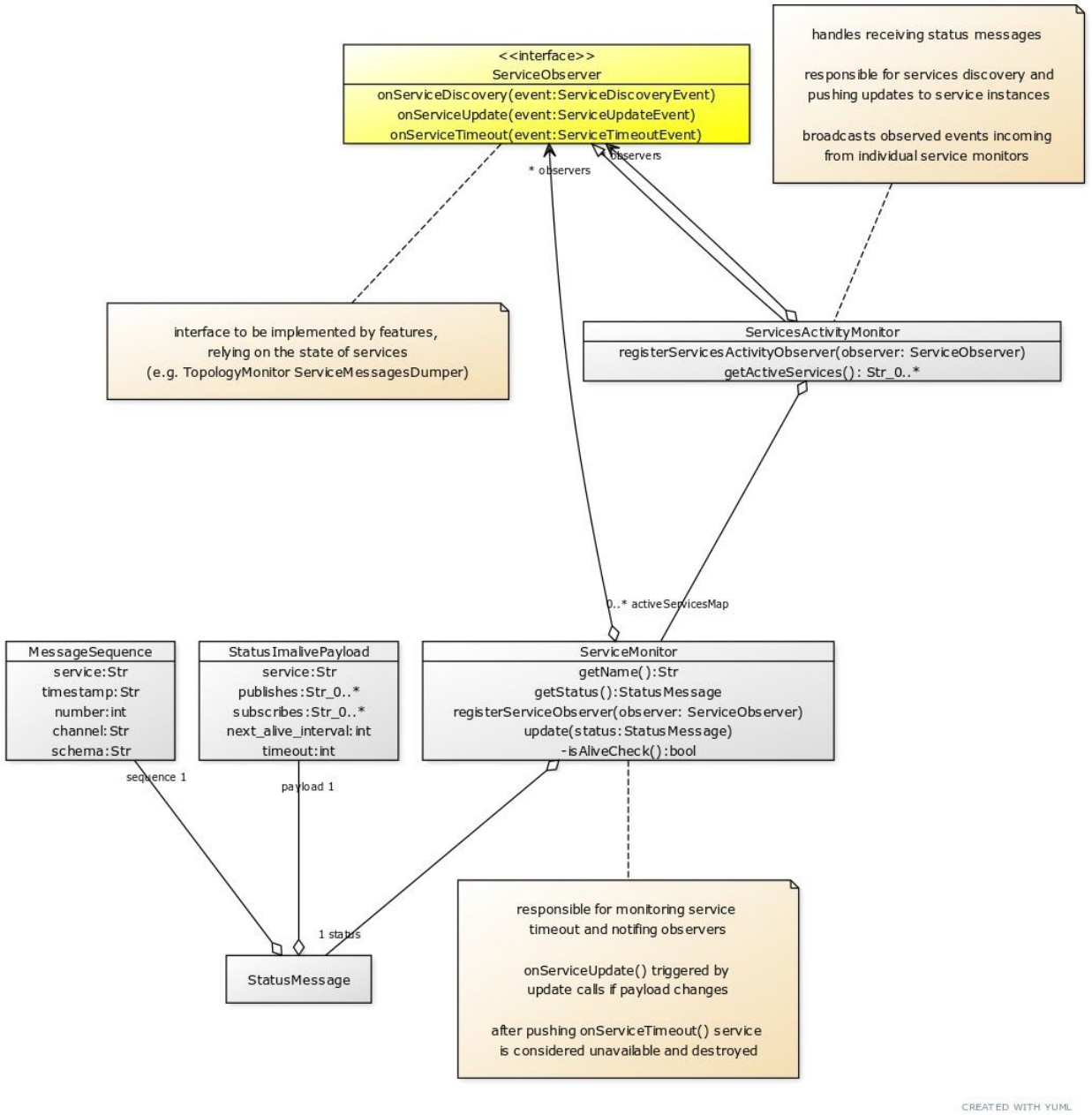


Rys 5. Hierarchia zdarzeń związanych z komunikacją platformy

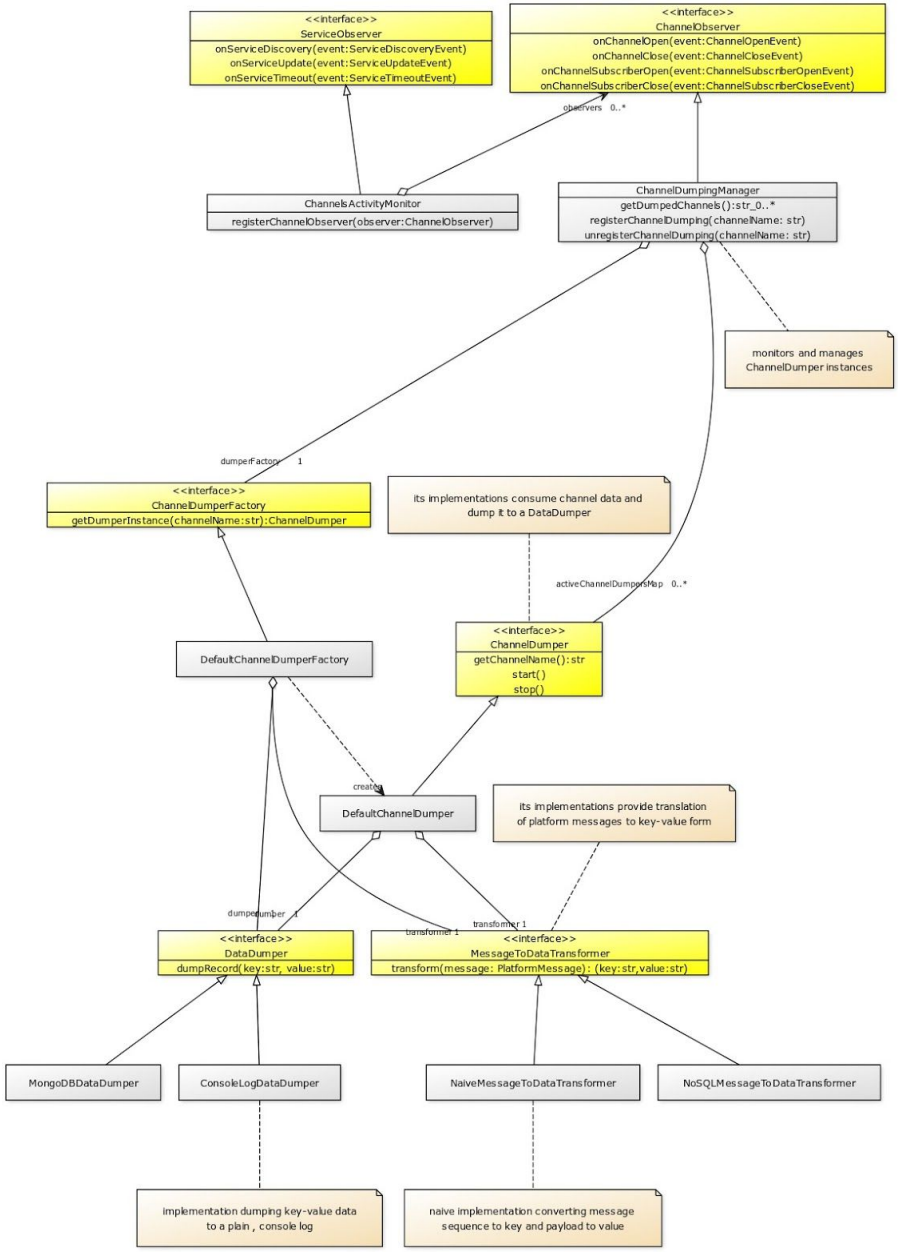
Projekt mechanizmów monitorowania w/w zdarzeń jest przedstawiony na Rys. 6 (monitoring usług) i Rys. 7 (monitoring kanałów). Przykładowym klientem (obserwatorem) mechanizmu monitorowania usług (*ServicesActivityMoniotr*) jest mechanizm monitorowania kanałów komunikacyjnych (*ChannelsActivityMonitor*). Z kolei obserwatorem zmian w nadawaniu i nasłuchiwaniu kanałów jest narzędzie do przechwytywania, przetwarzania i przekazywania komunikatów (*ChannelsDumpingManager*). Powyższe elementy mogą być wykorzystane do budowy narzędzi platformy takich jak:

- monitor topologii,
- walidator komunikacji,
- narzędzia wizualizacji pomiarów,
- mediator utrwalania komunikacji w bazie danych.
- itp.

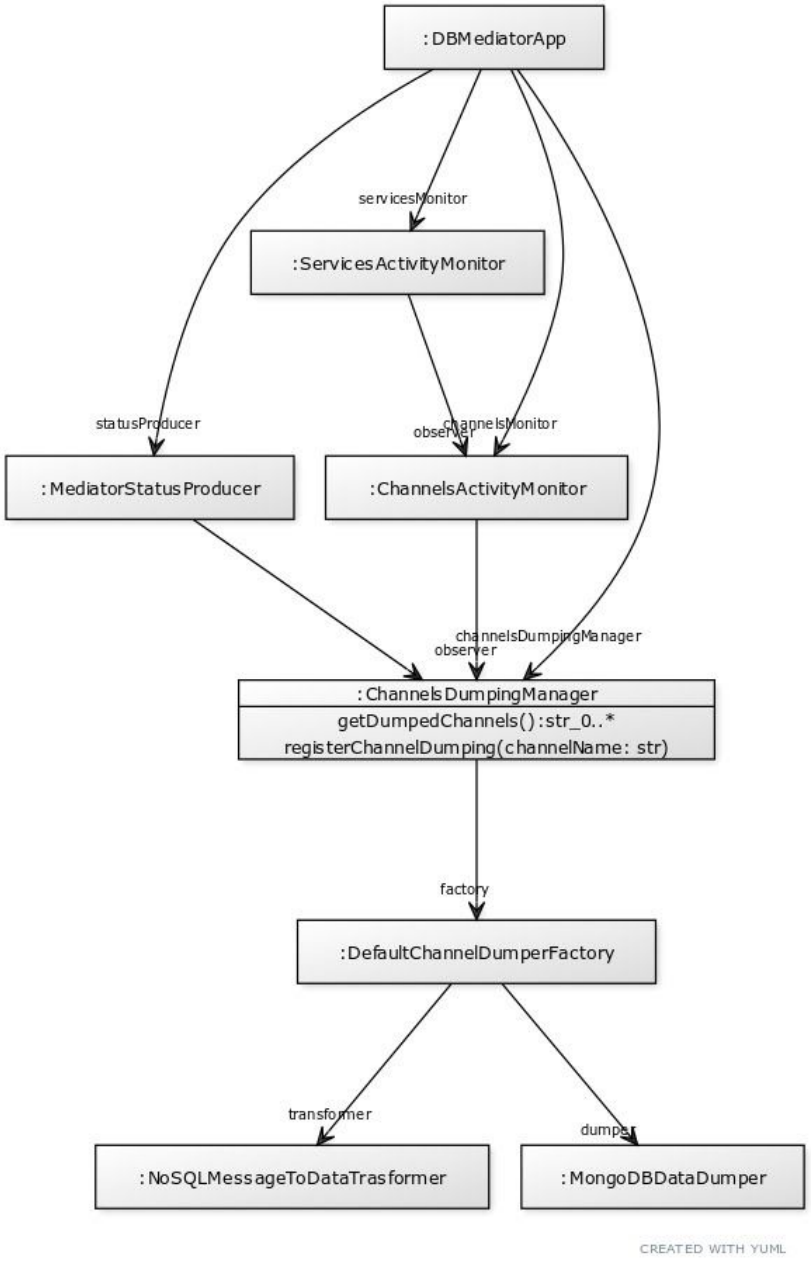
Przykładowy projekt aplikacji mediatora dla bazy danych MongoDB został przedstawiony na diagramie obiektów (Rys. 8).



Rys 6. Projekt monitoringu stanu usług.



Rys 7. Projekt monitoringu nadawania i nasłuchiwania kanałów przez usługi z przykładowym klientem *ChannelsDumpingManager*.



Rys 8. Projekt aplikacji mediatora utrwalającego komunikację platformy do bazy danych MongoDB.

Informacja o sytuacjach awaryjnych -- logi dla doraźnego monitorowania stanu systemu

Każda z usług może dodatkowo nadawać komunikaty kanałami logowania stanu systemu PLATOM. Struktura komunikatów z logownymi informacjami musi być zgodna ze schematem **logentry**.
Dostępne kanały logowania:

- Kanał **log_trace** - nieistotne informacje generowane przez usługi podczas pracy,
    - Przykład: informowanie o starcie/stopie jakiegoś podmoduły.
- Kanał **log_debug** - wszelkiej maści informacje uruchomieniowe.
    - Przykład: Zapytania do bazy danych, aktywności CPU jakiejś usługi, itd...
- Kanał **log_info** - Ogólne informacje o działaniu systemu.
    - Przykład: zmiana konfiguracji jakiejś usługi
- Kanał **log_warning** - Miała miejsce nieoczekiwana sytuacja, ale stabilność systemu nie jest zagrożona. Jego praca będzie kontynuowana.
    - Przykład: Poziom ciśnienia cieczy jest niższy od nominalnego.
    - Przykład: Temperatura silnika pompy ponad normę.
    - Przykład: Zmierzona wartość pojemności jednej z par elektrod nie mieści się w zakresie.
- Kanał **log_error** - Jakaś operacja zakończyła się niepowodzeniem. Praca systemu może zostać przerwana.
    - Przykład: Nie udało się uzyskać zadanej temperatury krystalizacji w ustalonym czasie, choć temperaturą da się sterować.
    - Przykład: Brak danych pomiarowych z tomografu.
- Kanał **log_fatal** - Coś kompletnie padło/uszkodziło się i dalsza praca systemu nie jest możliwa.
    - Przykład: zawór się nie zamknął, woda się leje, padła sprężarka, pożar reaktora, pierwszy dzień zajęć.

Komunikaty wysyłane w ramach tych kanałów mają następującą strukturę:

```
{
  "sequence": {
     "timestamp": "2019-11-12T11:22:33Z",
     "number": 1423234,
     "channel": "log_failure",
     "service": "scheduler",
     "schema": "logentry"
  },
  "payload": {
    "service": "scheduler",
    "message": "Kawa się skończyła"
```

```
  }
},
```

- Pole ***payload/service*** zawiera nazwę usługi, która jest źródłem danego komunikatu.
- Pole ***payload/message*** zawiera opis/treść komunikatu o błędzie/ostrzeżeniu/informacji.

Przykład: Komunikat z danymi tomografu ET3

Wartość przykładowego komunikatu dla tomografu, przy konfiguracji 8 elektrod:


```
{
  "sequence": {
     "timestamp": "2019-11-12T11:22:33Z",
     "number": 1423234,
     "channel": "et3_measurements",
     "service": "ET3",
     "schema": "ect_potentials"
  },
  "payload": {
     "electrodes": 8,
     "potentials": {
       "rows": 7,
       "columns": 7,
       "data": [
          [ 12.00, 13.00, 14.00, 15.00, 16.00, 17.00, 18.00 ],
          [ 23.00, 24.00, 25.00, 26.00, 27.00, 28.00 ],
          [ 34.00, 35.00, 36.00, 37.00, 38.00 ],
          [ 45.00, 46.00, 47.00, 48.00 ],
          [ 56.00, 57.00, 58.00 ],
          [ 67.00, 68.00 ],
          [ 78.00 ]
     ]
     }
  }
}
```

Opisy pól komunikatu:

- Pole ***sequence*** pozostaje bez zmian względem pierwszej części dokumentu.
- Pole ***payload*** zawiera dane, zgodne z formatem komunikatu, danym schematem **ect_potentials**.
    - Pole ***payload/electrodes*** to liczba elektrod (int)
    - Pole ***payload/potentials*** to macierz liczb zmiennoprzecinkowych o wymiarach 7x7 elementów.

Powyższy komunikat nadawany jest przez oprogramowanie pomiarowe tomografu i zawiera wartości pojemności/potencjału/cokolwiek zmierzone przez urządzenie. Komunikat ten nadawany jest kanałem kanale **et3_measurements**.

Oprogramowanie tomografu może również odbierać informacje o wzmocnieniach, jakie ma ustawić w urządzeniu. Źródłem komunikatów może być TomokisStudio, a całość wymiany informacji odbywa się kanałem **et3_gains**.

```
{
  "sequence": {
     "timestamp": "2019-11-12T20:22:33Z",
     "number": 666,
     "channel": "et3_gains",
     "service": "tomokis_studio"
     "schema": "etc_gains"
  },
  "payload": {
    "electrodes": 8,
    "gains": {
      "size": 7,
      "data": [ 1111, 222, 33, 4, 55, 666, 7777 ],
    }
  }
},
```

Schemat opisu protokołu wymiany danych

Każdy komunikat, przesyłany w ramach rozproszonego systemu PLATOM, musi być zgodny z wybranym schematem, opisującym jego strukturę. Nazwa schematu jest unikalnym ciągiem znaków i dana jest polem ***sequence/schema***. Schemat odpowiada za

charakterystykę pól bloku ***payload*** analizowanego komunikatu. Brak zgodności w tym zakresie (komunikatu ze schematem) oznacza błąd protokołu i musi kończyć się odrzuceniem komunikatu przez odbiorcę.

W ramach schematu każde pole w komunikacie ma swój typ danych (prymitywny lub złożony), zakresy walidacyjne oraz opis. Informacje te pozwolą na generowanie czytelnych informacji i stanie oraz przepływie danych w systemie, w razie jego awarii lub w trakcie uruchamiania. Pola podane w schemacie walidacyjnym są obowiązkowymi w komunikacie.

Typy pól w protokole PLATOM

Dopuszczalne są następujące typy danych - prymitywne:

- **boolean** - wartość logiczna; może przyjmować tylko liczby całkowite [0, 1]
- **string** - wartość tekstowa
  - Specyfikator **maxlength** - górna granica długości tekstu; długość podawana w znakach a nie w bajtach.
    - Brak specyfikatora = brak sprawdzania długości.
  - Format: ciąg znaków z kodowaniem UTF8.
- **integer** - liczba całkowita ze znakiem, w zakresie od -9 223 372 036 854 775 808 do +9 223 372 036 854 775 807 (`int64_t`).
  - Specyfikator **minimum** - najniższa poprawna wartość pola.
    - Brak specyfikatora = dolna granica typu `int64_t` (2^63-1).
  - Specyfikator **maximum** - najwyższa poprawna wartość pola.
    - Brak specyfikatora = górna granica typu `int64_t` (-2^63).
  - Format zapisu: ciąg liczb cyfr z ewentualnym znakiem na początku (42; -42; +42).
- **real** - liczba zmiennoprzecinkowa o precyzji zgodnej z typem `double` (`float64`).
  - **minimum** - najniższa poprawna wartość pola.
    - Brak specyfikatora = dolna granica typu `double`.
  - **maximum** - najwyższa poprawna wartość pola.
    - Brak specyfikatora = dolna granica typu `double`.
  - Format zapisu:
    - naukowy: 2e12; 23.42e-45
    - dziesiętny: -123.456; -.3; +444.
    - Niezależnie od konfiguracji lokalizacji systemu operacyjnego część dziesiętna liczby jest rozdzielona od części ułamkowej kropką.
    - Dopuszczalne znaki: `[0-9eE.+-]`.

Typy złożone:

- **vector** - wektor elementów o podanym typie

    - Specyfikator **minlength** - minimalna liczba elementów, uznawana za poprawną
        - Brak specyfikatora = 1 element
    - Specyfikator **maxlength** - maksymalna liczba elementów, uznawana za poprawną
        - Brak specyfikatora = brak granicy
        - Jeżeli **minlength** == **maxlength** == *n* to wektor musi mieć dokładnie i elementów.
    - Specyfikator **type** - typ danych elementów wektora
        - Specyfikator jest obowiązkowy
    - ~~Specyfikator **default** - wartość domyślna w przypadku niepełnej liczby wartości w wektorze.~~
    - W komunikacie wektor jest implementowany jako tablica (uporządkowana lista) z elementami o typie **type**.
    - ~~W komunikacie wektor posiada dwa pola:~~
        - ~~*size* - liczba całkowita określająca liczbę elementów,~~
        - ~~*data* - tablica danych~~
- **matrix** - macierz wartości o podanym typie
    - Specyfikator **minrows/mincols** - minimalna liczba wierszy/kolumn, uznawana za poprawną
        - Brak specyfikatora = 1 element
    - Specyfikator **maxrows/maxcols** - maksymalna liczba wierszy/kolumn, uznawana za poprawną
        - Brak specyfikatora = brak granicy
    - Specyfikator **type** - typ danych elementów wektora
        - Specyfikator jest obowiązkowy
    - Specyfikator **default** - wartość domyślna w przypadku niejednorodnej liczby elementów w wierszach lub samych wierszy.
        - Brak specyfikatora - liczba wierszy oraz kolumn musi być zgodna z polami ***rows***/***columns*** w komunikacie.
    - W komunikacie macierz posiada trzy pola:
        - ***rows*** - liczba całkowita określająca liczbę wierszy
        - ***cols*** - liczba całkowita określająca liczbę kolumn
        - ***data*** - tablica tablic danych macierzy; pierwszy wymiar to wiersze.

Typy pomocnicze:

- **timestamp** - Wartość daty/czasu
    - Format: zgodnie ze standardem ISO 8601, przykład: `2019-11-12T11:22:33Z`.

Specyfikatory dla wszystkich typów danych:

- **nullable** - stwierdza, czy pole danego typu może przyjąć wartość **null** (brak wartości). Domyślnie: **false**.

- ~~**description** - opis znaczenia danego pola/typu danych.~~
- **mapping** - określa mapowanie pola na kolumnę w bazie danych oraz sposób serializacji
  - ~~**column** - nazwa kolumny w tabeli danych~~
  - ~~**serialization** - mechanizm serializacji, zależny od typu danych danego pola~~

Mapowanie pól do bazy danych będzie realizowane dopiero po zakończeniu prac nad mechanizmem przechowywania danych.

Przykład: Schemat dla komunikatu z danymi tomografu ET3

```
{
  "name": "ect_potentials" // nazwa schematu walidacyjnego
  "service": "ET3", // nazwa usługi generującej komunikaty
  "channel": "et3_measurements", // nazwa kanału, jakim dany komunikat jest przesyłany
  "description": "jakiś opis tekstowy; dowolny",
  "mapping": {
     //"table": "nazwa tabeli w bazie danych"
  },

  "fields": [
     {
          "name": "electrodes",
          "description": "Liczba elektrod wykorzystana w danym pomiarze",
          "type": {
               "name": "integer",
               "minimum": 2,
               "maximum": 64
          },
          "mapping": {
               //"column": "electrodes",
               //"serialization": "tostring"
          }
     },
     {
```

```
            "name": "potentials",
            "description": "Macierz wartości pojemności dla danego pomiaru",
            "type": {
                  "name": "matrix",
                  "minrows": 1,
                  "mincols": 1,
                  "maxrows": 63,
                  "maxcols": 63
                  "default": 0.0,
                  "type": {
                        "name": "real"
                  }
            },
            "mapping": {
                  //"column": "measurements",
                  //"serialization": "multiline_with_spaces"
            }
      }
   ]
}
```

Przykład: Schemat komunikatów dla kanału *status*:

```
{
   "name": "schema_status" // nazwa schematu walidacyjnego
   "service": null, // nazwa usługi generującej komunikaty
   "channel": "status", // nazwa kanału, jakim dany komunikat jest przesyłany
   "description": "jakiś opis tekstowy; dowolny",
   "mapping": {
      //"table": "nazwa tabeli w bazie danych"
   },

   "fields": [
      {
```

```
        "name": "service",
        "description": "Nazwa usługi generującej komunikat, nadawany kanałem status",
        "type": {
            "name": "string",
            "nullable": false,  // nazwa usługi musi być zawsze podana
        },
        "mapping": {
            // ???? do ustalenia
        }
    },
    {
        "name": "publishes",
        "description": "Lista nazw kanałów, którymi dana usługa publikuje dane (np. pomiary)",
        "type": {
            "name": "vector",
            "minlength": 0,
            "type": {
                "name": "string",
                "nullable": false,  // nazwa usługi musi być zawsze podana
            }
        }
        "mapping": {
            // ???? do ustalenia
        }
    },
    {
        "name": "subscribes",
        "description": "Lista nazw kanałów, na których dana usługa nasłuchuje",
        "type": {
            "name": "vector",
            "minlength": 0,
            "type": {
                "name": "string",
                "nullable": false  // nazwa usługi musi być zawsze podana
            }
```

```
            },
            "mapping": {
                // ???? do ustalenia
            }
        },
        {
            "name": "next_alive_interval",
            "description": "Czas [ms] do następnego wygenerowania komunikatu na kanale status, przez tę
    usługę",
            "type": {
                "name": "integer",
                "min": 0,
                "max": 0xFFFFFFFF,
                "nullable": false  // wartość wymagana
            },
            "mapping": {
                // ???? do ustalenia
            }
        },
        {
            "name": "timeout",
            "description": "Czas [ms] jaki musi minąć od punktu czasowego, wynikającego z parametru
    next_alive_interval aby usługę nadawczą uznać za wyłączoną.",
            "type": {
                "name": "integer",
                "min": 0,
                "max": 0xFFFFFFFF,
                "nullable": false  // wartość wymagana
            },
            "mapping": {
                // ???? do ustalenia
            }
        }
    ]
}
```