Systemy rozproszone: Technologie middleware

Sprawozdanie do zadania: „Opcjonalne pola struktur danych i argumenty wywołania middleware”

Autor: Krzysztof Solecki

1. ICE:

W niektórych technologiach middleware takich jak ICE domyślnie wszystkie pola w strukturach i jako argumenty są wymagane. Aby oznaczyć pole jako opcjonalne, należy zdefiniować je jako „optional” wraz z tagiem. Gdy wartość opcjonalna nie jest ustawiona (ma wartość domyślną lub jest niezainicjowana), nie jest przekazywana wraz z przestrzenią nazw (np. w nagłówku komunikatu). W takim przypadku nie ma potrzeby przekazywania tagu do przestrzeni nazw, ponieważ wartość jest znana i stała. Jednak, jeśli wartość opcjonalna jest ustawiona, to jej tag jest przekazywany wraz z wartością do przestrzeni nazw. Dzięki temu odbiorca wiadomości wie, że wartość jest opcjonalna i może ją obsłużyć w odpowiedni sposób. Ten mechanizm pozwala na pominięcie nieustawionych pól opcjonalnych podczas serializacji i nie są one wysyłane przez sieć. Natomiast pola, których wartości opcjonalne są ustawione, dodają dodatkowy narzut, aby oznaczyć pole jako ustawione.

W programie Wireshark można zaobserwować różnice w rozmiarze pakietów sieciowych, gdy pole opcjonalne jest ustawione i gdy nie jest. Gdy wartość opcjonalna nie jest ustawiona, pole to nie jest wysyłane przez sieć, co prowadzi do mniejszego rozmiaru pakietu.

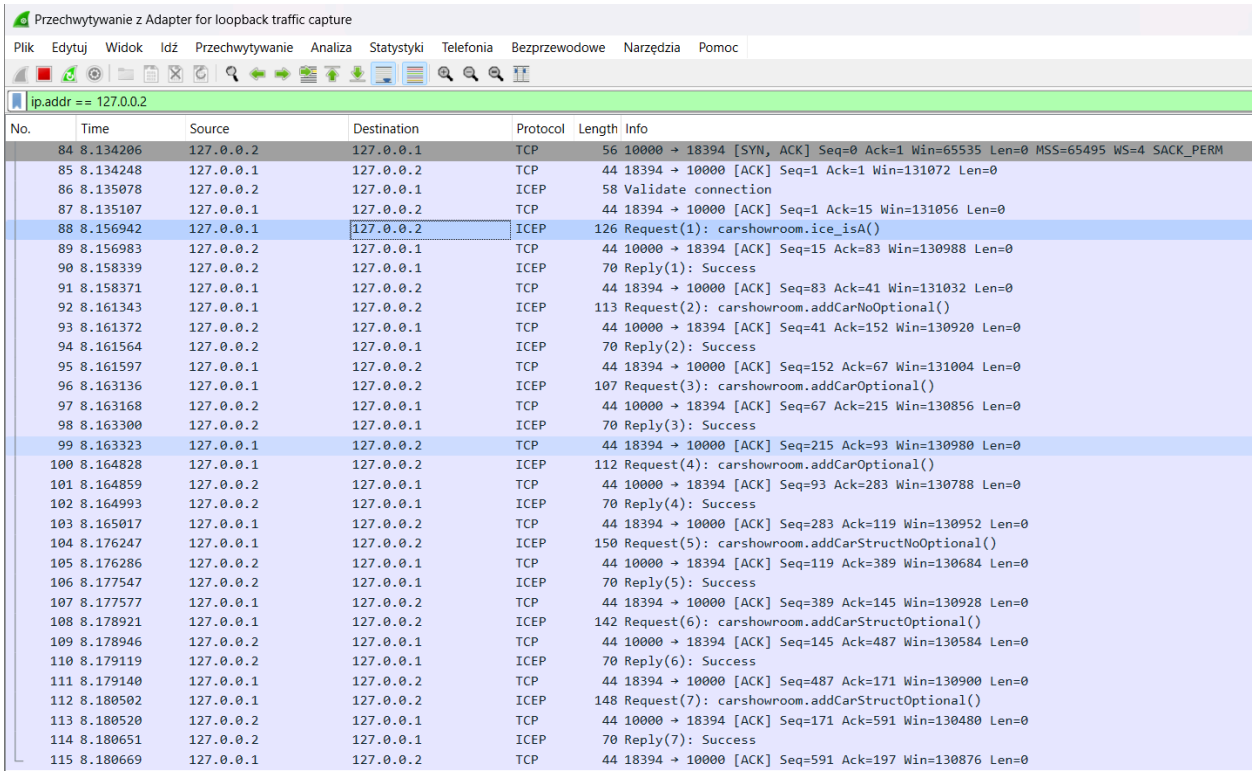
Przykładowy IDL stworzony na potrzeby zadania, który posłuży do przeprowadzenia testów:

```
module Carshowroom
{
    class CarOptional
    {
        string brand;
        string model;
        optional(1) int production_year;
    };

    class CarNoOptional
    {
        string brand;
        string model;
        int production_year;
    };

    interface CarshowroomService
    {
        bool addCarOptional(string brand, string model, optional(1) int production_year);
        bool addCarNoOptional(string brand, string model, int production_year);
        bool addCarStructOptional(CarOptional car);
        bool addCarStructNoOptional(CarNoOptional car);
    };
};
```

Kod testujący:

```
boolean res1 = service.addCarNoOptional( brand: "Tesla",  model: "X",  production_year: 2024);
System.out.println(res1);

boolean res2 = service.addCarOptional( brand: "Tesla",  model: "X", java.util.OptionalInt.empty());
System.out.println(res2);

boolean res3 = service.addCarOptional( brand: "Tesla",  model: "X",  production_year: 2024);
System.out.println(res3);

// Structs
boolean res4 = service.addCarStructNoOptional(new CarNoOptional ( brand: "Tesla",  model: "X",  production_year: 2024));
System.out.println(res4);

boolean res5 = service.addCarStructOptional(new CarOptional( brand: "Tesla",  model: "X"));
System.out.println(res5);

boolean res6 = service.addCarStructOptional(new CarOptional( brand: "Tesla",  model: "X",  production_year: 2024));
System.out.println(res6);
```

Przechwycone pakiety za pomocą Wiresharka:



No.	Time	Source	Destination	Protocol	Length	Info
84	8.134206	127.0.0.2	127.0.0.1	TCP	56	10000 → 18394 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=4 SACK_PERM
85	8.134248	127.0.0.1	127.0.0.2	TCP	44	18394 → 10000 [ACK] Seq=1 Ack=1 Win=131072 Len=0
86	8.135078	127.0.0.2	127.0.0.1	ICEP	58	Validate connection
87	8.135107	127.0.0.1	127.0.0.2	TCP	44	18394 → 10000 [ACK] Seq=1 Ack=15 Win=131056 Len=0
88	8.156942	127.0.0.1	127.0.0.2	ICEP	126	Request(1): carshowroom.ice_isA()
89	8.156983	127.0.0.2	127.0.0.1	TCP	44	10000 → 18394 [ACK] Seq=15 Ack=83 Win=130988 Len=0
90	8.158339	127.0.0.2	127.0.0.1	ICEP	70	Reply(1): Success
91	8.158371	127.0.0.1	127.0.0.2	TCP	44	18394 → 10000 [ACK] Seq=83 Ack=41 Win=131032 Len=0
92	8.161343	127.0.0.1	127.0.0.2	ICEP	113	Request(2): carshowroom.addCarNoOptional()
93	8.161372	127.0.0.2	127.0.0.1	TCP	44	10000 → 18394 [ACK] Seq=41 Ack=152 Win=130920 Len=0
94	8.161564	127.0.0.2	127.0.0.1	ICEP	70	Reply(2): Success
95	8.161597	127.0.0.1	127.0.0.2	TCP	44	18394 → 10000 [ACK] Seq=152 Ack=67 Win=131004 Len=0
96	8.163136	127.0.0.1	127.0.0.2	ICEP	107	Request(3): carshowroom.addCarOptional()
97	8.163168	127.0.0.2	127.0.0.1	TCP	44	10000 → 18394 [ACK] Seq=67 Ack=215 Win=130856 Len=0
98	8.163300	127.0.0.2	127.0.0.1	ICEP	70	Reply(3): Success
99	8.163323	127.0.0.1	127.0.0.2	TCP	44	18394 → 10000 [ACK] Seq=215 Ack=93 Win=130980 Len=0
100	8.164828	127.0.0.1	127.0.0.2	ICEP	112	Request(4): carshowroom.addCarOptional()
101	8.164859	127.0.0.2	127.0.0.1	TCP	44	10000 → 18394 [ACK] Seq=93 Ack=283 Win=130788 Len=0
102	8.164993	127.0.0.2	127.0.0.1	ICEP	70	Reply(4): Success
103	8.165017	127.0.0.1	127.0.0.2	TCP	44	18394 → 10000 [ACK] Seq=283 Ack=119 Win=130952 Len=0
104	8.176247	127.0.0.1	127.0.0.2	ICEP	150	Request(5): carshowroom.addCarStructNoOptional()
105	8.176286	127.0.0.2	127.0.0.1	TCP	44	10000 → 18394 [ACK] Seq=119 Ack=389 Win=130684 Len=0
106	8.177547	127.0.0.2	127.0.0.1	ICEP	70	Reply(5): Success
107	8.177577	127.0.0.1	127.0.0.2	TCP	44	18394 → 10000 [ACK] Seq=389 Ack=145 Win=130928 Len=0
108	8.178921	127.0.0.1	127.0.0.2	ICEP	142	Request(6): carshowroom.addCarStructOptional()
109	8.178946	127.0.0.2	127.0.0.1	TCP	44	10000 → 18394 [ACK] Seq=145 Ack=487 Win=130584 Len=0
110	8.179119	127.0.0.2	127.0.0.1	ICEP	70	Reply(6): Success
111	8.179140	127.0.0.1	127.0.0.2	TCP	44	18394 → 10000 [ACK] Seq=487 Ack=171 Win=130900 Len=0
112	8.180502	127.0.0.1	127.0.0.2	ICEP	148	Request(7): carshowroom.addCarStructOptional()
113	8.180520	127.0.0.2	127.0.0.1	TCP	44	10000 → 18394 [ACK] Seq=171 Ack=591 Win=130480 Len=0
114	8.180651	127.0.0.2	127.0.0.1	ICEP	70	Reply(7): Success
115	8.180669	127.0.0.1	127.0.0.2	TCP	44	18394 → 10000 [ACK] Seq=591 Ack=197 Win=130876 Len=0

Analiza pakietów:

a) Argumenty:
   - Podanie wszystkich argumentów do metody addCarNoOptional:
     TCP Payload: 69 bajtów
     Input Parameters: 18 bajtów
   - Nie podanie opcjonalnego argumentu do metody addCarOptional:
     TCP Payload: 63 bajtów
     Input Parameters: 14 bajtów
   - Podanie wszystkich argumentów do metody addCarOptional:
     TCP Payload: 68 bajtów
     Input Parameters: 19 bajtów

b) Struktury:
   - Podanie wszystkich argumentów do metody addCarStructNoOptional:
     TCP Payload: 106 bajtów
     Input Parameters: 49 bajtów
   - Nie podanie opcjonalnego argumentu do metody addCarStructOptional:
     TCP Payload: 98 bajtów

Input Parameters: 43 bajtów
- Podanie wszystkich argumentów do metody addCarStructOptional:
  TCP Payload: 104 bajtów
  Input Parameters: 49 bajtów

Użycie opcjonalnych pól w middleware, takim jak ICE, może pomóc zmniejszyć ilość danych przesyłanych po sieci, co sprawia, że transmisja staje się bardziej wydajna. Kiedy wartość opcjonalna nie jest ustawiona, pole jest po prostu pomijane podczas wysyłania danych przez sieć, co skutkuje mniejszymi pakietami. Natomiast gdy wartość opcjonalna jest ustawiona, pole jest przekazywane wraz z wartością do odbiorcy, co pozwala mu na odpowiednie jej przetworzenie. To podejście minimalizuje dodatkowy narzut na identyfikację ustawionego pola.

2. Thrift:

W Apache Thrift, opcjonalne wartości można stosować jedynie w strukturach danych zdefiniowanych w języku IDL. Aby je zaznaczyć, korzysta się z słowa kluczowego "optional". Jednakże, w przypadku argumentów, nie ma możliwości definiowania pól jako opcjonalnych. Bez względu na to, czy opcjonalne parametry są zdefiniowane, czy nie, rozmiar przesyłanych danych pozostanie niezmienny. W przeciwieństwie do ICE, w Apache Thrift nie używa się tutaj tagów.

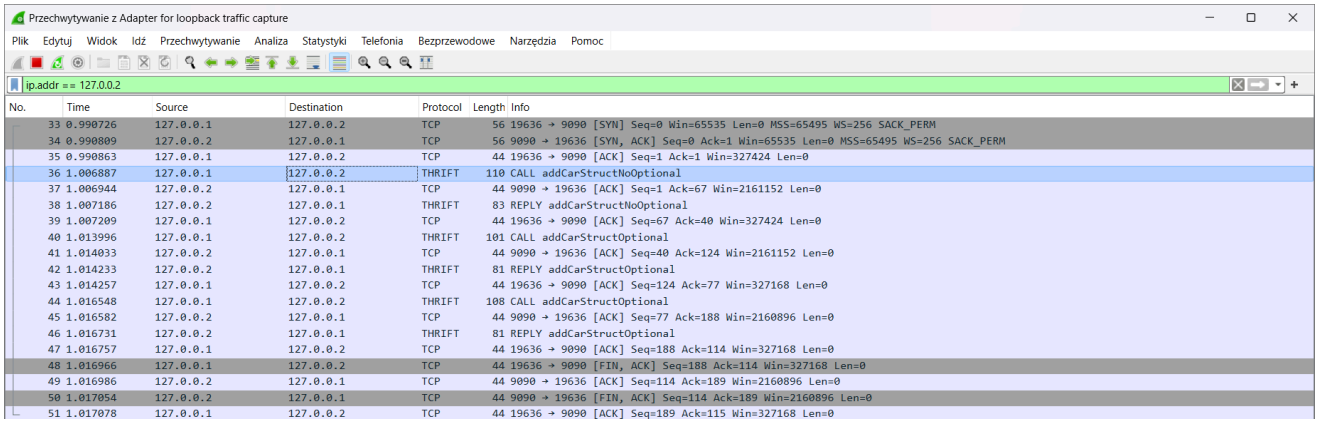
Przykładowy IDL:

```
namespace java genjava
namespace py genpython

struct CarOptional {
  1: string brand,
  2: string model,
  3: optional i32 year,
}

struct CarNoOptional {
  1: string brand,
  2: string model,
  3: i32 year,
}

service CarshowroomService {
   bool addCarOptional(1:string brand, 2: string model, 3: optional i32 year);
   bool addCarNoOptional(1:string brand, 2: string model, 3: i32 year);
   bool addCarStructOptional(1: CarOptional car);
   bool addCarStructNoOptional(1: CarNoOptional car);
}
```

Kod testujący:

```java
CarNoOptional carNoOptional = new CarNoOptional();
carNoOptional.setBrand("Tesla");
carNoOptional.setModel("X");
carNoOptional.setYear(2024);

boolean res1 = client.addCarStructNoOptional(carNoOptional);
System.out.println(res1);

CarOptional carOptional = new CarOptional();
carOptional.setBrand("Tesla");
carOptional.setModel("X");

boolean res2 = client.addCarStructOptional(carOptional);
System.out.println(res2);

CarOptional carOptionalSet = new CarOptional();
carOptionalSet.setBrand("Tesla");
carOptionalSet.setModel("X");
carOptionalSet.setYear(2024);

boolean res3 = client.addCarStructOptional(carOptionalSet);
System.out.println(res3);
```

Przechwycone pakiety za pomocą Wireshark:



Analiza pakietów:

Struktury:

- Podanie wszystkich argumentów do metody addCarStructNoOptional:
  TCP Payload: 66 bajtów
  Thrift Protocol Call Length: 22 bajtów
- Nie podanie opcjonalnego argumentu do metody addCarStructOptional:
  TCP Payload: 57 bajtów
  Thrift Protocol Call Length: 20 bajtów
- Podanie wszystkich argumentów do metody addCarStructOptional:
  TCP Payload: 64 bajtów
  Thrift Protocol Call Length: 20 bajtów

3. gRPC:

W przypadku gRPC opcjonalne wartości w strukturach danych (messsages) definiowane są przy pomocy słowa kluczowego “optional” w proto3, który pozwala na definiowanie pól jako opcjonalnych. Opcjonalne pola nie muszą być ustawione, co oznacza, że nie będą one serializowane i przesyłane w sieci, jeśli ich wartość nie została ustawiona.

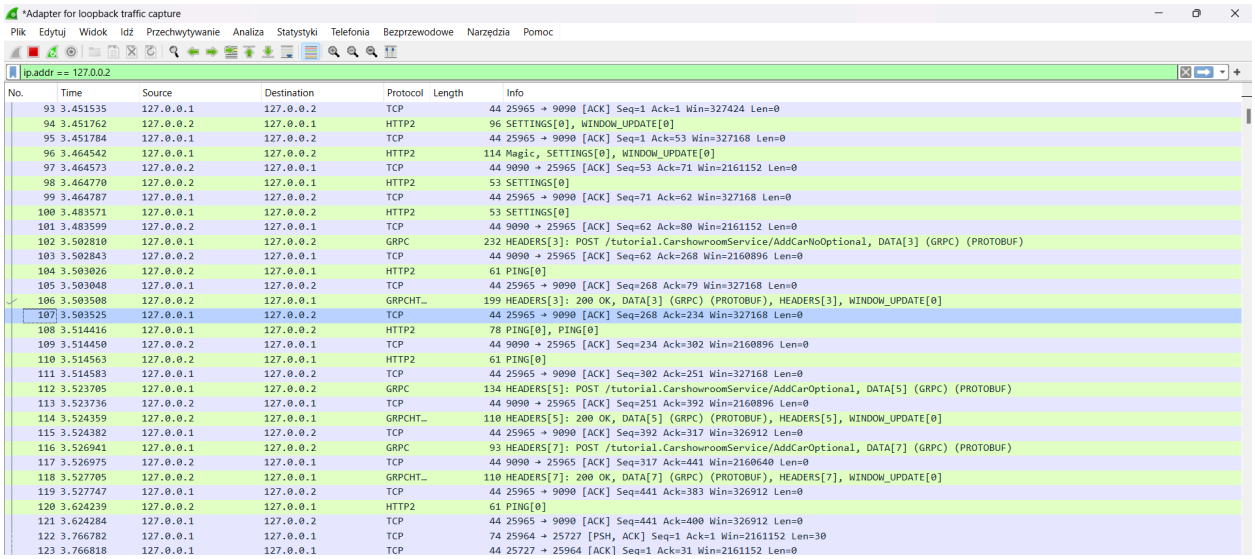
Przykładowy IDL:

```
syntax = "proto3";

package tutorial;

option java_package = "sr.proto";
option java_outer_classname = "CarshowroomServiceProtos";

message AddCarOptionalRequest {
  string brand = 1;
  string model = 2;
  optional int32 year = 3;
}

message AddCarNoOptionalRequest {
  string brand = 1;
  string model = 2;
  int32 year = 3;
}

message AddCarResponse {
  bool response = 1;
}

service CarshowroomService  {
  rpc AddCarOptional(AddCarOptionalRequest) returns (AddCarResponse);
  rpc AddCarNoOptional(AddCarNoOptionalRequest) returns (AddCarResponse);
}
```

Kod testujący:

```
CarshowroomServiceProtos.AddCarNoOptionalRequest requestNoOptional = CarshowroomServiceProtos.AddCarNoOptionalRequest.newBuilder()
        .setBrand("Tesla")
        .setModel("X")
        .setYear(2024)
        .build();
CarshowroomServiceProtos.AddCarResponse responseOptional = stub.addCarNoOptional(requestNoOptional);
System.out.println("Response received: " + responseOptional.getResponse());

CarshowroomServiceProtos.AddCarOptionalRequest requestOptionalNotSet = CarshowroomServiceProtos.AddCarOptionalRequest.newBuilder()
        .setBrand("Tesla")
        .setModel("X")
        .build();

CarshowroomServiceProtos.AddCarResponse responseOptionalNotSet = stub.addCarOptional(requestOptionalNotSet);
System.out.println("Response received: " + responseOptionalNotSet.getResponse());

CarshowroomServiceProtos.AddCarOptionalRequest requestOptionalSet = CarshowroomServiceProtos.AddCarOptionalRequest.newBuilder()
        .setBrand("Tesla")
        .setModel("X")
        .setYear(2024)
        .build();

CarshowroomServiceProtos.AddCarResponse responseOptionalSet = stub.addCarOptional(requestOptionalSet);
System.out.println("Response received: " + responseOptionalSet.getResponse());
```

Przechwycone pakiety za pomocą Wireshark:



Analiza pakietów:

Wiadomości:

- Podanie wszystkich argumentów do metody addCarStructNoOptional:
  TCP Payload: 188 bajtów
  GRPC Message Data: 13 bajtów
- Nie podanie opcjonalnego argumentu do metody addCarStructOptional:
  TCP Payload: 90 bajtów
  GRPC Message Data: 10 bajtów
- Podanie wszystkich argumentów do metody addCarStructOptional:
  TCP Payload: 49 bajtów
  GRPC Message Data: 13 bajtów

Wnioski końcowe:

Opcjonalne wartości w strukturach danych oraz argumentach wywołania middleware umożliwiają elastyczną komunikację między aplikacjami. Dzięki nim można uniknąć przesyłania zbędnych informacji, co skutkuje mniejszymi rozmiarami pakietów sieciowych w przypadku Ice, czy gRPC. Natomiast w przypadku Thrift, rozmiar pakietów sieciowych pozostaje niezmienny, niezależnie od tego, czy parametry opcjonalne są ustawione, czy nie.