Systemy Rozproszone – Laboratorium

Technologie *middleware*

Łukasz Czekierda (luke@agh.edu.pl)
Zespół Systemów Rozproszonych (DSRG)
Katedra Informatyki AGH – Kraków

DSRG

Plan zajęć

- Dyskusja ważniejszych podstawowych zagadnień technologii middleware
- Przedstawienie wybranych funkcjonalności przykładowych rozwiązań
  - Zeroc ICE
  - Apache Thrift
  - Google RPC (kolejne zajęcia)
- Komunikacja rozproszona we współczesnej sieci Internet (kolejne zajęcia)

Distributed middleware

- Object-oriented middleware (OO RPC)
  - OMG CORBA
  - Zeroc ICE
  - RMI
  - . Net Remoting
- Message-oriented middleware
- Remote procedure call middleware (RPC)

Co woła klient?

- Metody?
- Procedury?
- Operacje?

Dlaczego middleware?

- Klasyka systemów rozproszonych
- „CORBA – matka wszystkich technologii”
- Ważna umiejętność – dobór właściwego rozwiązania w danym zastosowaniu

Czym jest (była) CORBA?

- = **C**ommon **ORB** **A**rchitecture
- ORB = Object Request Broker
- Technologia warstwy pośredniej (*middleware*)
- Umożliwia komunikację pomiędzy aplikacjami:
  - działającymi na różnych maszynach
  - działającymi pod różnymi systemami operacyjnymi
  - napisanymi w różnych językach programowania
- Dostarcza wielu usług (Naming, Trading, Event, Transaction,...)

DSRG

CORBA – inne spojrzenie

100 letnia technologia, kotrej się już prawie nie używa

czy naprawdę trzeba ludzi męczyć technologią CORBA?

zajęcia częściowo dotyczące wieloletniej wiedzy, przestarzała technologia.

Wymagający prowadzący ciekawy temat.

Praktyczne wykonywanie poznanych technologii.

DSRG

Czym jest ICE?

- = **I**nternet **C**ommunication **E**ngine
- Technologia warstwy pośredniej (*middleware*)
- Duże podobieństwa do CORBA
  - Wiele usprawnień i uproszczeń
  - Nacisk na wydajność i prostotę rozwiązania
- Wiele zaawansowanych mechanizmów
- Pozwala na budowę aplikacji na urządzenia *enterprise*, *desktop*, *mobile* i *embedded*

DSRG

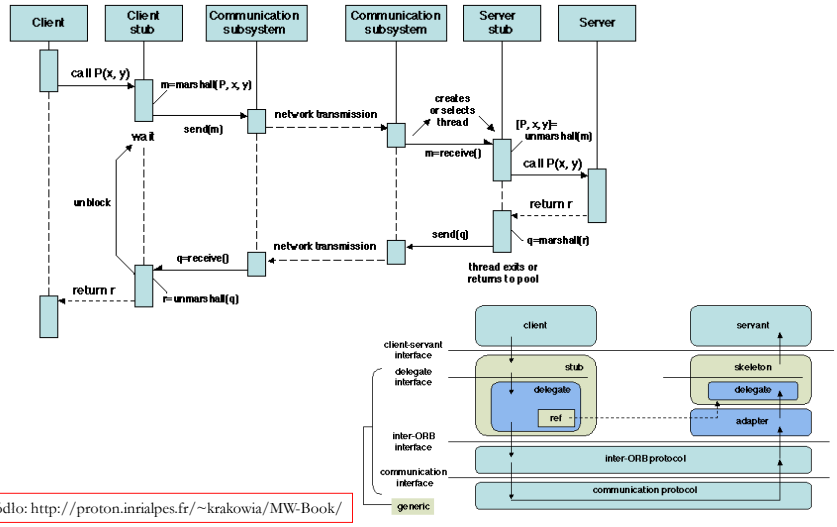
Czym są Thrift i gRPC?

- Rozwiązania podobne…
- … ale jednak nieco inne…

- Zobaczmy, porównajmy!

DSRG

Zdalne wywołanie



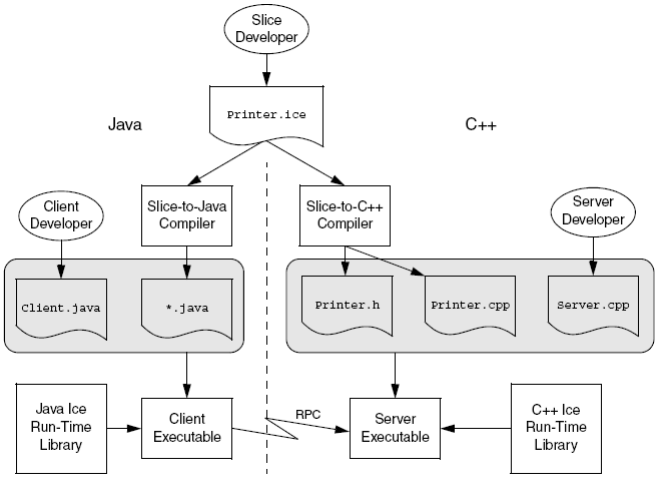
źródło: http://proton.inrialpes.fr/~krakowia/MW-Book/

DSRG

TWORZENIE APLIKACJI MIDDLEWARE

DSRG

Budowa i wykonanie aplikacji middleware (na przykładzie ICE)



Typowe kroki

1. Zdefiniowanie interfejsu (IDL)
2. Kompilacja interfejsu do danego języka programowania
3. Implementacja interfejsu
4. Implementacja i konfiguracja serwera
5. Implementacja i konfiguracja klienta
6. Kompilacja i uruchomienie

Języki definiowania interfejsów

- Języki z rodziny IDL
- Definiują kontrakt pomiędzy klientem a serwerem
- Rozwiązania
  - CORBA: CORBA IDL
  - Zeroc: SLICE (Specification Language for ICE) (.ice)
  - Thrift: (.thrift)
  - gRPC: (.proto)

Słowniczek trudniejszych pojęć

- Przeźroczystość
- *Marshaling*, *unmarshaling*
- Obiekt
- Serwer
- Serwant

Obiekt a serwant

- Obiekt (ICE/CORBA) – abstrakcja posiadająca jednoznaczną identyfikację oraz interfejs i odpowiadająca na żądania klientów
- Serwant – element strony serwerowej, implementacja funkcjonalności interfejsu w konkretnym języku programowania

- Relacje ilościowe pomiędzy obiektem i serwantem?

Komunikacja

- ICE
  - TCP, UDP, SSL/TCP, WebSocket
  - Serializacja binarna
- Thrift
  - TCP
  - Różne sposoby serializacji (binarna, JSON)
- gRPC
  - HTTP2/TCP
  - Serializacja binarna

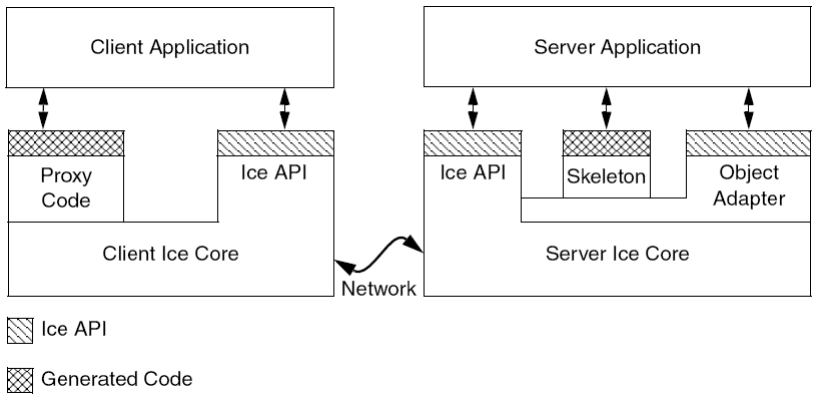
O odległości…

```
interface Person
{
  string getFirstName();
  string getLastName();
  string getNationalID();
  …
}
```

- Czy to jest dobry interfejs dla potrzeb komunikacji zdalnej? Nie – dlaczego?

ZEROC ICE

Architektura ICE



Slice

- Specification Language for Ice
- Deklaratywny język z rodziny IDL
- Opisuje kontrakt między klientem a serwerem ICE
- Niezależny od języka programowania
- Odwzorowania do konkretnych języków programowania: C++, C#, Java, Python, Ruby, PHP, JavaScript

Elementy języka Slice

- Moduł – *namespace*. Wszystkie interfejsy muszą być definiowane w module
- Interfejsy
- Typy proste
- Enumeracje
- Struktury
- Sekwencje
- Słowniki
- Stałe
- Wyjątki (możliwość dziedziczenia)

```
module ZeroC {
    module Client {
        // Definitions here...
    };
    module Server {
        // Definitions here...
    };
};
```

Type	Range of Mapped Type	Size of Mapped Type
`bool`	`false` or `true`	≥ 1bit
`byte`	–128–127[a]	≥ 8 bits
`short`	-2^{15} to $2^{15}-1$	≥ 16 bits
`int`	-2^{31} to $2^{31}-1$	≥ 32 bits
`long`	-2^{63} to $2^{63}-1$	≥ 64 bits
`float`	IEEE single-precision	≥ 32 bits
`double`	IEEE double-precision	≥ 64 bits
`string`	All Unicode characters, excluding the character with all bits zero.	Variable-length

DSRG

Przykład definicji i implementacji interfejsu

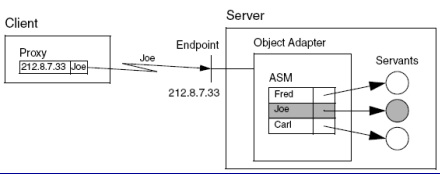
```
module Demo {  //slice
  sequence<long> seqOfNumbers;
  enum operation { MIN, MAX, AVG };
  interface Calc  {
    long add(int a, int b);
    long subtract(int a, int b);
  };
};

public class CalcI implements Calc { //java
  @Override public long add(int a, int b, Current __current)
  {
    return a + b;
  }
  …
}
```


DSRG

Identyfikacja obiektów Ice

- Obiekty Ice są identyfikowane przez strukturę Identity
- Kategoria może być pusta
- Reprezentacja w postaci łańcucha znaków **`kategoria/nazwa`** lub **`nazwa`**

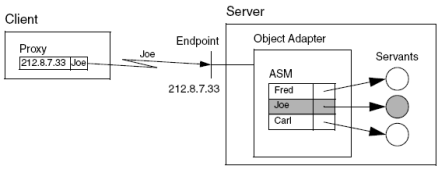
```
module Ice {
    struct Identity {
        string name;
        string category;
    };
};
```




```
module Ice {
    local interface ObjectAdapter {
        // ...

        Object* add(Object servant, Identity id);
        Object* addWithUUID(Object servant);
        Object  remove(Identity id);
        Object  find(Identity id);
        Object  findByProxy(Object* proxy);

        // ...
    };
};
```

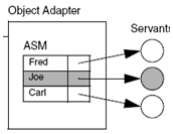
Adapter obiektu (OA) w ICE

- Odpowiednik POA w CORBA
- Komunikator tworzy jeden lub więcej adapterów obiektów
- OA odpowiada za
  - Kierowanie żądań adresowanych do obiektów do odpowiednich serwantów (odwzorowanie może być statyczne lub dynamiczne)
  - Zarządza cyklem życia obiektów
- Operacje add/remove dodają/usuwają skojarzenie obiekt-serwant zawarte w tablicy ASM (Active Servant Map)




```
module Ice {
    local interface ObjectAdapter {
        // ...

        Object* add(Object servant, Identity id);
        Object* addWithUUID(Object servant);
        Object  remove(Identity id);
        Object  find(Identity id);
        Object  findByProxy(Object* proxy);

        // ...
    };
};
```


Zarządzanie serwantami



- Proste (i często wykorzystywane) podejście
  - Każdy obiekt Ice odwzorowuje się na innego serwanta
  - Odwzorowanie obiekt-serwant jest zapewniane przez tablicę ASM
  - Brak skojarzenia powoduje zgłoszenie wyjątku ObjectNotExistException
- Bardziej zaawansowane podejścia
  - Servant Locator
  - Servant Evictor
  - Default Servant

Default Servant

- Dla każdej kategorii można zarejestrować jeden domyślny serwant
- Będzie on wykorzystany, gdy tablica ASM adaptera nie będzie zawierać indywidualnego wpisu dla poszukiwanego obiektu
- Strategia różne obiekty – wspólny serwant

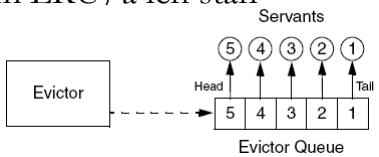
Servant Locator

- Servant Locator jest rejestrowany w adapterze dla konkretnej kategorii (można zarejestrować tylko jeden taki obiekt dla danej kategorii)
- Jeśli adapter nie znajdzie odwzorowania w tablicy ASM, przekaże żądanie do lokatora zarejestrowanego dla tej kategorii
- Lokator może
  - wskazać (np. stworzyć) serwanta – do niego zostanie skierowane to żądanie
  - zwrócić null – zgłaszany jest wyjątek ObjectNotExistException
- Możliwość realizacji różnych strategii, np. późna aktywacja serwantów, pula serwantów, współdzielony serwant, …

DSRG

Servant Evictor

- Odmiana Servant Locator, która utrzymuje *cache* serwantów
- Dba o nieprzekraczanie zadanej liczności aktywnych serwantów
- Serwanty nieużywane mogą być usuwane (np. w oparciu o algorytm LRU) a ich stan zachowywany



- Możliwość implementacji własnego ewiktora

DSRG

Referencje

- Obiekty middleware są zazwyczaj przekazywane przez referencję - przekazanie przez wartość to przekazanie stanu obiektu – jego pól!

Nie tylko klient-serwer

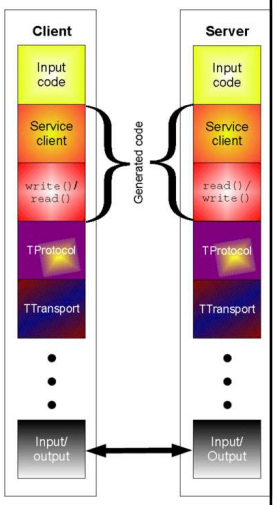
- Klient nie musi być „czystym" klientem, serwer nie musi być „czystym" serwerem
- Przydatne np. w aplikacjach potrzebujących natychmiastowych notyfikacji o zachodzących zdarzeniach - serwer jest aktywny (jest klientem)
- Decyzja o posiadaniu obiektów middleware także po stronie klienta implikuje konieczność instancjonowania tam również adaptera obiektów (OA)

APACHE THRIFT

DSRG

Wprowadzenie

- Stworzony w laboratoriach Facebook
- Obecnie projekt Apache
- Architektura warstwowa z możliwością różnej realizacji poszczególnych warstw
- Podejście bardziej usługowe niż obiektowe
- Kiepska dokumentacja ☹



DSRG

Definiowanie interfejsu - typy podstawowe

- bool (true / false)
- byte: 8-bit signed integer
- i16/i32/i64: 16/32/64-bit signed integer
- double: 64-bit floating point number
- string: UTF-8 encoding
- struct
- enum
- list<t1>: ordered list of elements of type t1. May contain duplicates.
- set<t1>: unordered set of unique elements of type t1.
- map<t1,t2>: map of strictly unique keys of type t1 to values of type t2.
- exception

DSRG

Przykład definicji interfejsu

```
struct Work {
   1: i32 num1 = 0,
   2: required i32 num2,
   3: optional string language = "english"
}
enum OperationType {  SUM = 1,  MIN = 2,  MAX = 3,   AVG = 4 }

service Calculator {
   i32 add(1:i32 num1, 2:i32 num2),
   i32 subtract(1:i32 num1, 2:i32 num2),
   i32 multiply(1:i32 num1, 2:i32 num2)
}
service AdvancedCalculator extends Calculator {
   double op(1:OperationType type, 2: set<double> val),
}
```


DSRG

Kompilacja i implementacja interfejsu (Handler = servant)

```
thrift --gen java   calculator.thrift
thrift --gen csharp calculator.thrift


public class CalculatorHandler implements Calculator.Iface
{
  @Override
  public int add(int n1, int n2) {
    return n1 + n2;
  }

  …
}
```

Protocol Layer

- TBinaryProtocol – serializacja binarna, efektywne kodowanie TLV
- TCompactProtocol – serializacja binarna, bardzo efektywne kodowanie
- TJSONProtocol – serializacja tekstowa, JSON

- TDenseProtocol – bez metadanych, eksperymentalny
- TDebugProtocol – przydatny przy debugowaniu ☺

Transport Layer

- Podstawowe mechanizmy transportu:
  - TSocket - Uses blocking socket I/O for transport.
  - TFramedTransport - Sends data in frames, where each frame is preceded by a length. This transport is required when using a non-blocking server.
- Dodatkowe metody transportu:
  - Do pliku: TFileTransport
  - Do pamięci: TMemoryTransport
  - Z kompresją: TZlibTransport (używany w połączeniu z innym transportem)

Processor

- Pobiera dane z wejścia, przekazuje do odpowiedniego handlera, a odpowiedź przesyła na wyjście
- Generowany przez kompilator

Serwer

- TSimpleServer – jednowątkowy serwer, blocking I/O. Zasadniczo tylko do testowania aplikacji.
- TThreadPoolServer – wielowątkowy serwer, blocking I/O
- TNonblockingServer – wielowątkowy serwer, non-blocking I/O (Java: NIO channels), wymaga transportu TFramedTransport

Strona serwerowa

```
Calculator.Processor processor =
   new Calculator.Processor(new CalculatorHandler());

TServerTransport serverTransport = new TServerSocket(9090);

TProtocolFactory protocolFactory1 = new TBinaryProtocol.Factory();
TProtocolFactory protocolFactory2 = new TJSONProtocol.Factory();
TProtocolFactory protocolFactory3 = new TCompactProtocol.Factory();

TServer server = new TSimpleServer(
   new Args(serverTransport)
        .protocolFactory(protocolFactory1)
        .processor(processor));

server.serve();
```


Strona serwerowa

- Zazwyczaj serwer uruchamia tylko jedną instancję obiektu implementującego interfejs
- Wyjątkiem od tej reguły jest TMultiplexedProcessor

```
TMultiplexedProcessor multiplex = new TMultiplexedProcessor();
multiplex.registerProcessor("S1", processor);
multiplex.registerProcessor("S2", processor2);
```

- Wnioski?

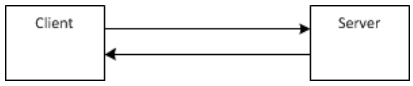
CIĄG DALSZY NASTĄPI…

CO NIECO O KOMUNIKACJI W INTERNECIE…

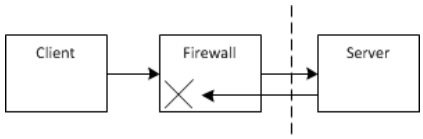
DSRG

Komunikacja dwukierunkowa

- Klient nie ma być „czystym” klientem, serwer nie ma być „czystym” serwerem…
- Brzmi dobrze, ale…



- Problemy: NAT, firewall,… - może się skończyć tak:



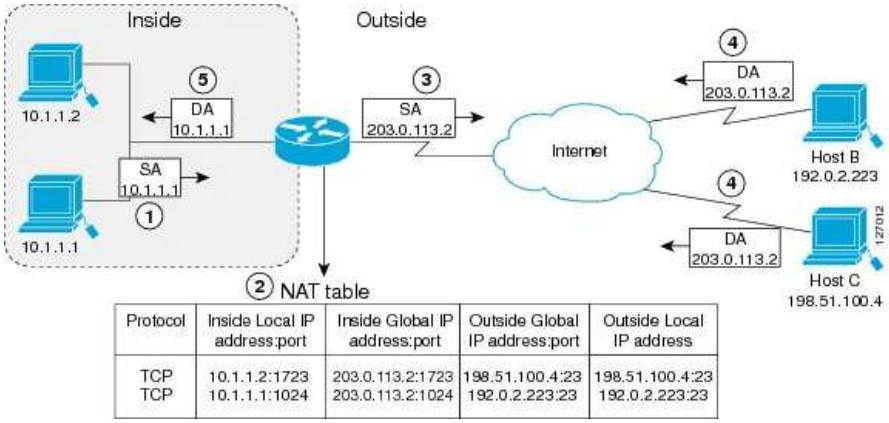
DSRG

Translacja adresów

- NAT, a tak naprawdę PAT
- Jak się skomunikować z komputerem za NAT?
- Jak działają aplikacje typu Team Viewer?
- STUN+TURN=ICE (choć nie to…)

DSRG

Translacja adresów



Protocol	Inside Local IP address:port	Inside Global IP address:port	Outside Global IP address:port	Outside Local IP address
TCP	10.1.1.2:1723	203.0.113.2:1723	198.51.100.4:23	198.51.100.4:23
TCP	10.1.1.1:1024	203.0.113.2:1024	192.0.2.223:23	192.0.2.223:23

DSRG

Tablica translacji

- UDP a TCP
- Co z innymi protokołami?
- Co daje połączeniowość protokołu?
- Cisco: domyślnie 24h dla TCP (chyba, że połączenie zostanie zamknięte lub przerwane – wówczas minuta) i 5 min. dla UDP, ale często te wartości są znacznie zmniejszane

O czym warto pamiętać?

- Urządzenie NAT/PAT zazwyczaj nie podmienia adresów i portów przesyłanych wewnątrz wiadomości
- Zniknięcie wpisu w tablicy translacji: wiele powodów
  - restart urządzenia
  - administracyjne usunięcie wpisów
  - przekroczenie czasu życia wpisu

Ice – połączenie dwukierunkowe

- Pomysł: wykorzystać istniejący, ustanowiony przez klienta kanał komunikacyjny (TCP, SSL) do komunikacji serwera z klientem
- Klient uruchamia OA i w jego tablicy ASM rejestruje obiekt odbierający żądania/wiadomości od serwera
- Czy to będzie działać?
- Czy to będzie działać niezawodnie?

Jak obsługiwać komunikację?

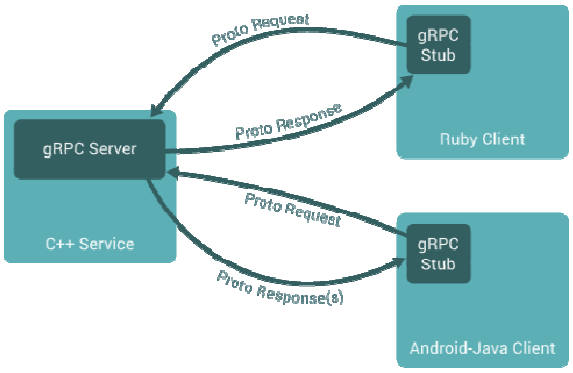
- Czy klient wie, że serwer stracił z nim łączność?
- Co może zrobić serwer?
- Przywracanie łączności

Czy możliwe jest nawiązanie bezpośredniej łączności?

- Nawiązanie bezpośredniej łączności pomiędzy dwiema aplikacjami działającymi na urządzeniach z adresami prywatnymi może być… niemożliwe
- Może być konieczny pośrednik
- Warto spojrzeć: STUN, TURN

GRPC

Wprowadzenie



Istotne cechy

- Usługi – nie obiekty
- Komunikacja z wykorzystaniem transferu wiadomości

- Serializacja: Protocol Buffers
- Komunikacja: HTTP/2

- gRPC nie może być nazwane *OO middleware*

Serializacja: Protocol Buffers

- Serializacja – cechy
  - Tekstowa lub binarna
  - Zawierająca metadane lub nie
  - Opisana schematem lub nie
  - Ograniczona do języka, platformy itp. lub nie
- Jej realizacje: XML, JSON, Ice, Thrift, Protocol Buffers

- Jakie cechy ma serializacja Protocol Buffers?
- Jakie ma zalety? Jakie ma wady?
- Gdzie jest wykorzystywana?

DSRG

proto – podstawowe informacje

- Wersja (np. syntax = "proto2"), istotniejsze różnice:
  - Proto2:
    - pola muszą być otagowane: optional/required
    - możliwość określenia domyślnej wartości pola
  - Proto3:
    - wszystkie pola są opcjonalne (optional)
    - pola nie mogą mieć deklarowanej domyślnej wartości
- Podstawowy element: wiadomość (message)
- Typy: int32, int64, …, bytes, string, bool, enum, sekwencje (repeated), message

więcej na: https://developers.google.com/protocol-buffers/docs/proto

DSRG

proto – przykładowe wiadomości

```
message SearchRequest {
  required string query = 1;
  optional int32 page_number = 2;
  optional int32 result_per_page = 3 [default = 10];
  enum Corpus {
    UNIVERSAL = 0;
    WEB = 1;
    IMAGES = 2;
    LOCAL = 3;
    NEWS = 4;
    PRODUCTS = 5;
    VIDEO = 6;
  }
  optional Corpus corpus = 4 [default = UNIVERSAL];
}
```

```
message SearchResponse {
  message Result {
    required string url = 1;
    optional string title = 2;
    repeated string snippets = 3;
  }
  repeated Result result = 1;
}
```

```
message Outer {                 // Level 0
  message MiddleAA {  // Level 1
    message Inner {   // Level 2
      required int64 ival = 1;
      optional bool  booly = 2;
    }
  }
```

więcej na: https://developers.google.com/protocol-buffers/docs/proto

DSRG

gRPC – rozszerzenie definicji proto

```
message ArithmeticOpArguments {
  int32 arg1 = 1;
  int32 arg2 = 2;
}
message ArithmeticOpResult {
  int32 res = 1;
}
service Calculator {
  rpc Add (ArithmeticOpArguments) returns (ArithmeticOpResult) {}
}
message ComplexArithmeticOpArguments {
  OperationType optype = 1;
  repeated double args = 2;
}
service AdvancedCalculator {
  rpc ComplexOperation (ComplexArithmeticOpArguments) returns
   (ComplexArithmeticOpResult) {}
```


DSRG

Kilka uwag

- Brak możliwości rozszerzania usług przez dziedziczenie
- Brak wyjątków
- Obsługa błędów – statusy wywołań, m.in.:
  - GRPC_STATUS_UNIMPLEMENTED
  - GRPC_STATUS_UNAVAILABLE
  - GRPC_STATUS_DEADLINE_EXCEEDED

DSRG

Komunikacja strumieniowa

```
service StreamTester {
  rpc GeneratePrimeNumbers(Task) returns (stream Number) {}
  rpc CountPrimeNumbers(stream Number) returns (Report) {}
}
```

- Sposoby komunikacji
  - Simple (unary) RPC
  - Server-side streaming RPC
  - Client-side streaming RPC
  - Bidirectional streaming RPC
- Strumieniowanie - możliwość dostarczenia wielu wiadomości przed zakończeniem wywołania
- Strumienie są zawsze inicjowane przez klienta

DSRG

Komunikacja strumieniowa - przykład

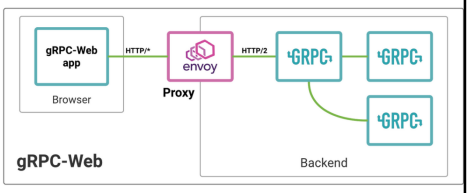
```
@Override
public void generatePrimeNumbers(Task request, StreamObserver<Number> responseObserver)
{
    System.out.println("generatePrimeNumbers");
    for (int i = 0; i < request.getMax(); i++) {
        if(isPrime(i)) { //zwłoka czasowa - dla obserwacji procesu strumieniowania
            Number number = Number.newBuilder().setValue(i).build();
            responseObserver.onNext(number);
        }
    }
    responseObserver.onCompleted();
}
```

```
Task request = Task.newBuilder().setMax(15).build();
Iterator<Number> numbers;
try {
    numbers = streamTesterBlockingStub.generatePrimeNumbers(request);
    while(numbers.hasNext())
    {
        Number num = numbers.next();
        System.out.println("Number: " + num.getValue());
    }
} catch (StatusRuntimeException ex) {
    logger.log(Level.WARNING, "RPC failed: {0}", ex.getStatus());
    return;
}
```

DSRG

gRPC w aplikacjach webowych

- Komponenty
  - Klient: gRPC-Web, biblioteka JavaScript
  - Strona serwerowa: service gateway (Envoy, Nginx) + zasadnicze usługi gRPC



- Nie cała funkcjonalność gRPC jest obecnie obsługiwana

więcej na: https://grpc.io/blog/state-of-grpc-web/, https://grpc.io/blog/grpc-web-ga/

DSRG

Podsumowanie

- Czy wiem, co to jest to middleware?
- Czy wiem, na czym polega specyfika i wartość dodana technologii middleware w stosunku do omawianych wcześniej rozwiązań?
- Architektura rozwiązań middleware
- Definiowanie interfejsów
- Adresacja w systemach middleware

KONIEC