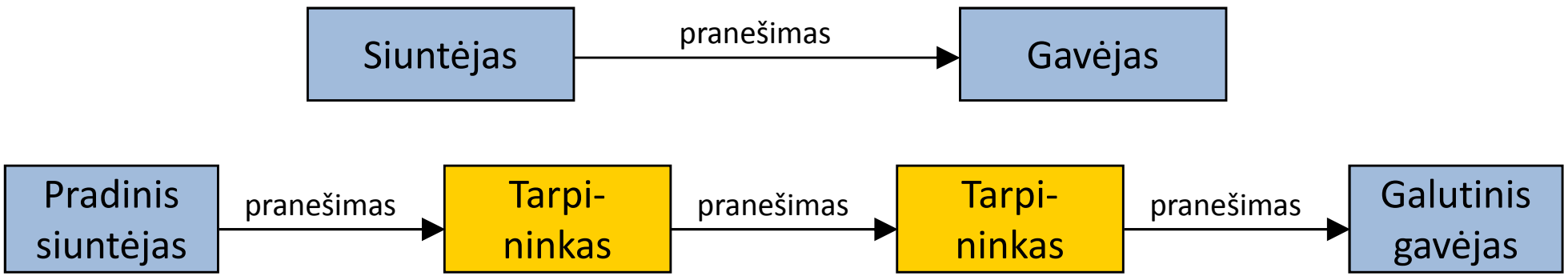
Web servisai

Pranešimų stiliai ir protokolai

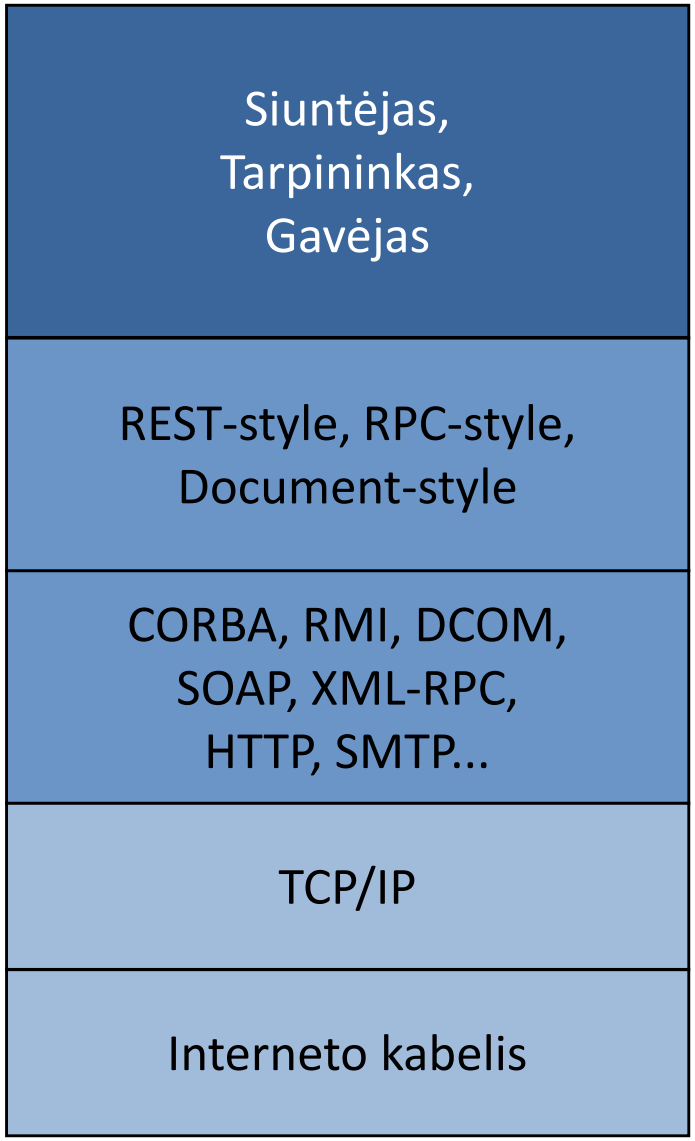
Osvaldas Grigas

Programų rolės

- Kompiuterių tinkle programos komunikuoja siųsdamos viena kitai pranešimus (*messages*)
- Tokios programos dar vadinamos mazgais (*nodes*) arba galiniais taškais (*endpoints*)
- Pranešimų kontekste programos įgyja tokias roles:
  - Pradinis siuntėjas (*initial sender*) – programa, kuri suformuoja ir išsiunčia pranešimą.
  - Galutinis gavėjas (*ultimate receiver*) – programa, kuriai skirtas pradinio siuntėjo suformuotas pranešimas.
  - Tarpininkas (*intermediary*) – programa, kuri persiunčia pranešimą galutiniam gavėjui arba sekančiam tarpininkui. Tarpininkas paprastai nekeičia paties pranešimo turinio, tačiau gali keisti jo meta-duomenis, įdėti jį į eilę (*message queue*), valdyti transakcijas ir pan.



Bendravimo stiliai ir protokolai



Tinkle bendraujančių programų **rolės**

Pranešimų siuntimo **stiliai**

Komunikacijos ir serializacijos **protokolai**
(kai kurie protokolai naudoja kitus)

Duomenų transportas

Pranešimų siuntimo stiliai

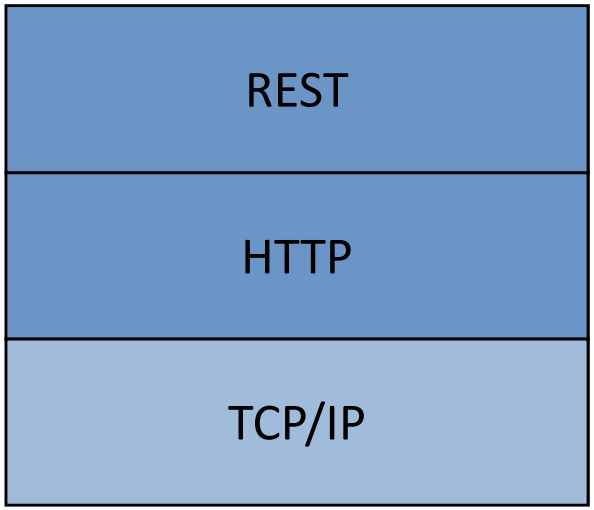
- REST-style
- RPC-style
- Document-style

REST-style

REST-style pranešimai

Stilius:

Protokolas:



REST-style

- Web servisas atstovauja **web resursus**
- Komunikacijos protokolas: HTTP
- Pranešimai yra HTTP užklausos ir HTTP atsakymai
- REST stiliaus pranešimus priimantis web servisas dar vadinamas *RESTful service*
- Pranešimų turinys dažniausiai koduojamas XML formatu – toks stilius dar vadinamas *XML over HTTP*

REST-style

- Su RESTful web servisu bendraujama kaip su bet kuriuo kitu web resursu - vadovaujamasi REST (Representational State Transfer) architektūros principais:
  - Konkretų resursą identifikuoja URI, kuriuo kreipiamasi į web servisą
  - HTTP GET užklausa naudojama paimti resurso turiniui, kuris grąžinamas HTTP atsakyme
  - HTTP PUT arba POST užklausa naudojama resursui keisti arba naujam resursui sukurti. Užklausos kūne nurodomas naujasis resurso turinys
  - HTTP DELETE užklausa naudojama resursui trinti

REST-style pranešimų pavyzdys

GET užklausa:

```
GET /student?id=123 HTTP/1.1
Host: www.example.org
```

Atsakymas:

```
HTTP/1.1 200 OK
Content-Length: 158
Content-Type: application/xml

<?xml version=“1.0” encoding=“utf-8”?>
<student>
   <name>Petras Petraitis</name>
   <age>22</age>
</student>
```

REST-style pranešimų pavyzdys

PUT užklausa:

```
PUT /student?id=123 HTTP/1.1
Host: www.example.org
Content-Length: 111
Content-Type: application/xml

<?xml version="1.0" encoding="utf-8"?>
<student>
   <name>Petras Petraitis</name>
   <age>23</age>
</student>
```

Atsakymas
(jei toks resursas jau buvo):

```
HTTP/1.1 200 OK
```

Atsakymas
(jei tokio resurso dar nebuvo):

```
HTTP/1.1 201 Created
```

REST-style pranešimų pavyzdys

DELETE užklausa:

```
DELETE /student?id=123 HTTP/1.1
Host: www.example.org
```

Atsakymas:

```
HTTP/1.1 200 OK
```

REST-style privalumai

- Paprastumas:
  - Komunikacijai naudojamas vienintelis HTTP protokolas
  - Pakanka keturių standartinių operacijų: GET, POST, PUT ir DELETE
  - Sąlyginai lengva, nenaudojant specializuotų įrankių, parašyti tokį web servisą ir jo klientą
- Universalumas
  - HTTP protokolą supranta bet kokia tinklo programinė įranga, jį palaiko praktiškai visi programavimo karkasai
  - Web servisą galima iškviesti su bet kuria naršykle, įrašius jo atstovaujamo resurso URL į adresų lauką
- Prieinamumas
  - Jei RESTful web servisu publikuojama XML informacija, ji tampa automatiškai pasiekiama:
    - HTML puslapiuose (per nuorodas)
    - XML dokumentuose (per XLink, XPointer, XInclude, XSLT)
    - kaip semantinio žiniatinklio dalis (RDF ir OWL ontologijose)

REST-style trūkumai

- Kol kas trūksta išbaigtų specializuotų įrankių RESTful web servisams kurti, nors situacija gėrėja
- Nėra paprasto būdo bendrauti asinchroniškai
  - HTTP serveris yra pasyvus, t.y. jis siunčia atsakymą tik tada, kai gauna užklausą
- Nesaugu, nes pranešimai keliauja pliku tekstu
  - jeigu reikia saugumo, vienintelis įprastas sprendimas - naudoti HTTPS
- Nėra įprasto būdo valdyti procesus (kombinuoti kelis web servisus) ir palaikyti transakcijas

RPC-style

RPC-style

- Web servisas traktuojamas kaip **modulis**
- Pranešimas yra modulio operacijos iškvietimas, t.y. Remote Procedure Call (RPC)
- Operacijos rezultatas taip pat yra pranešimas
- Komunikacijos ir serializacijos protokolai:
  - dvejetainiai: CORBA, RMI, DCOM
  - tekstiniai: XML-RPC
  - SOAP (serializuojamas įvairiai)

RPC privalumai

- Suprantama programuotojams
  - web serviso operacijos iškvietimas primena objekto metodo iškvietimą
- Dažniausiai naudojami specialūs API, kurie atlieka didelę dalį darbo ir pasirūpina detalėmis:
  - nereikia patiems serializuoti/deserializuoti duomenų tipų ir pan.

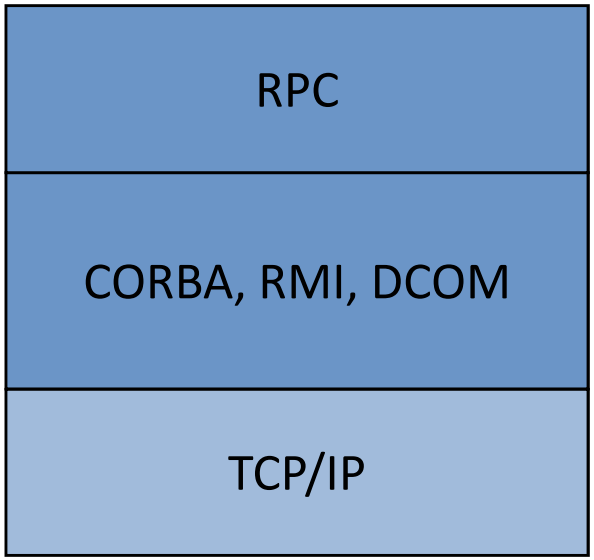
RPC trūkumai

- Nelankstu
  - pranešimo struktūra yra griežtai susieta su web serviso procedūromis ir vidiniais duomenų tipais
  - todėl pakeitimai serviso pusėje dažnai reikalauja atlikti atitinkamus pakeitimus kliento pusėje
  - tai pažeidžia vieną iš kertinių SOA principų – *loose coupling*
- Priklausomybė nuo specializuotų įrankių ir API
  - nes pačiam kurti ir nuskaityti RPC pranešimus sudėtinga, ypač jeigu jie koduojami dvejetainiu formatu
  - tokie įrankiai, skirtingai nuo programuotojo, neišmano pranešimų paskirties ir loginės struktūros, todėl rezultatas nebūna optimalus (dėl to nukenčia greitis)

Dvejetainiai RPC pranešimai

Stilius:

Protokolai:



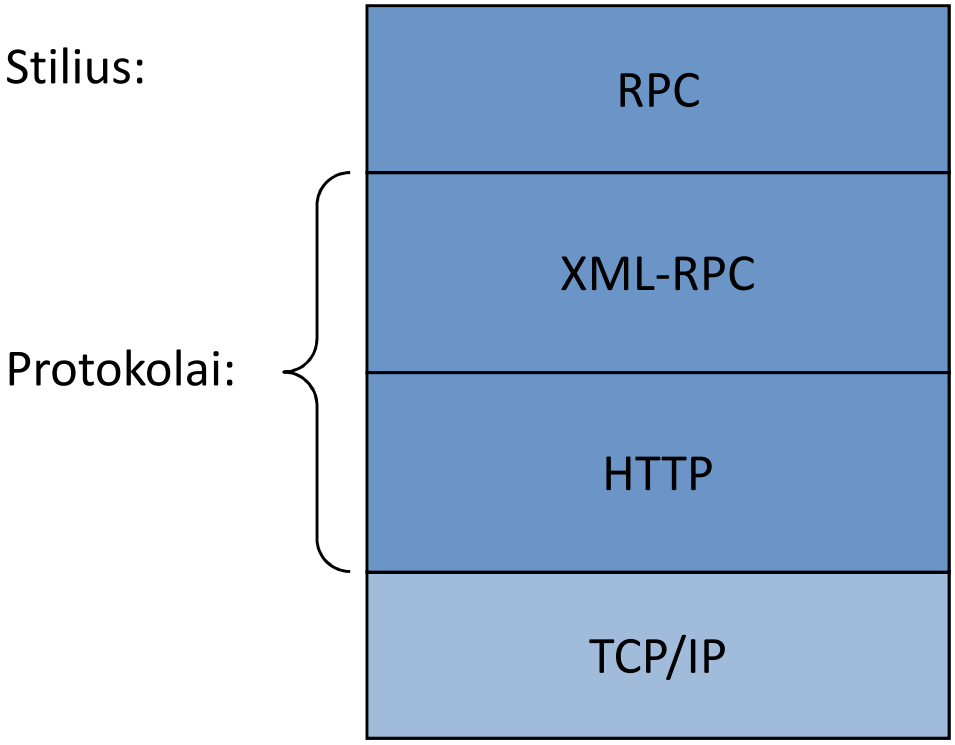
Dvejetainių RPC protokolų privalumai

- Greitis: dvejetainiai pranešimai greičiau keliauja tinklu ir yra greičiau apdorojami nei tekstiniai
- + Galioja bendri RCP privalumai

Dvejetainių RPC protokolų trūkumai

- Prisirišimas prie konkrečių technologijų susiaurina jų panaudojimo galimybes, pvz:
  - DCOM veikia tik tarp Windows programų
  - RMI veikia tik tarp Java programų
- Dvejetainius pranešimus, keliaujančius ne HTTP kanalu, dažnai blokuoja įmonių ugniasienės
  - todėl kyla sunkumų bandant komunikuoti tarp organizacijų
- + Galioja bendri RCP trūkumai

XML-RPC pranešimai



XML-RPC

- Naudoja HTTP protokolą XML pranešimų siuntimui
- Web serviso iškvietimas yra HTTP POST užklausa
- Rezultatas grįžta HTTP atsakyme
- Ir užklausos, ir atsakymo kūnas koduojamas tekstiniu XML formatu
- Parametrų duomenų tipai, struktūros ir masyvai nurodomi elementų vardais

XML-RPC užklausos pavyzdys

```
POST /studentService HTTP/1.1
Host: www.example.org
Content-Length: 111
Content-Type: application/xml

<?xml version="1.0"?>
<methodCall>
  <methodName>students.service.getStudent</methodName>
  <params>
     <param><value><i4>123</i4></value></param>
  </params>
</methodCall>
```

XML-RPC atsakymo pavyzdys

```
HTTP/1.1 200 OK
Content-Length: 111
Content-Type: application/xml

<?xml version="1.0"?>
<methodResponse>
 <params>
  <param>
    <value>
      <struct>
        <member>
          <name>foo</name>
          <value><i4>1</i4></value>
        </member>
        <member>
          <name>bar</name>
          <value><i4>2</i4></value>
        </member>
      </struct>
    </value>
  </param>
 </params>
</methodResponse>
```

XML-RPC privalumai

- Paprastumas:
  - pranešimų struktūra yra aiški ir nesudėtinga
  - naudoja programuotojams pažįstamas sąvokas: struct, array...
  - tekstinius pranešimus lengviau stebėti ir debug’inti
- + Galioja bendri RCP privalumai

XML-RPC trūkumai

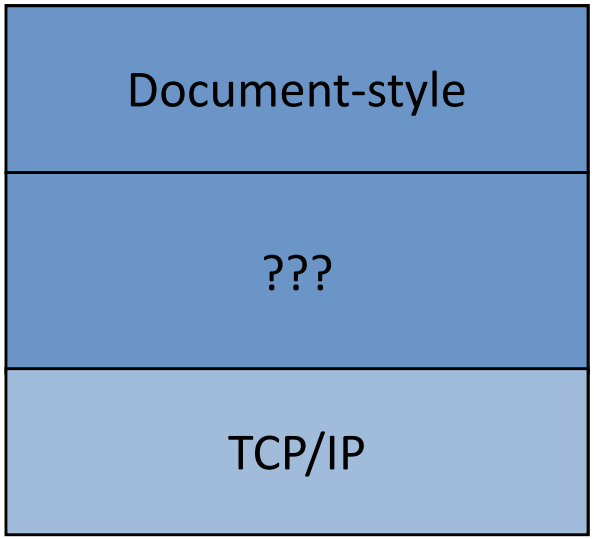
- XML-RPC nėra labai populiarus, todėl trūksta įrankių
- Tekstiniai XML-RPC pranešimai užima daug vietos, ilgiau užtrunka juos suformuoti, persiųsti tinklu ir nuskaityti
- + Galioja bendri RCP trūkumai

Document-style

Document-style pranešimai

Stilius:

Protokolai:



Document-style

- Web servisas traktuojamas tiesiog kaip **galinis taškas (*endpoint*)**, kuris priima ir siunčia pranešimus
- Pranešimas yra XML *dokumentas*, kurio struktūra yra sutartinė - ją supranta tik siuntėjas ir gavėjas
- Pranešimas yra savarankiškas, t.y. logiškai atsietas nuo transporto protokolo
- Tokiais pranešimais bendraujantis web servisas dar vadinamas *message-oriented service*
- Komunikacijos protokolas: teoriškai bet koks
  - paprastai naudojamas SOAP – dėl savo lankstumo

Document-style privalumai

- Lankstumas:
  - pranešimų struktūrą lemia išorinis *kontraktas* tarp siuntėjo ir gavėjo
  - lengviau “atsirišti” nuo serviso procedūrų ir vidinių duomenų tipų, taigi lengviau įgyvendinti *loose coupling* principą
- Paprasčiau realizuoti asinchroninį bendravimą, nes pranešimas atsietas nuo transporto protokolo:
  - gavėjas gali nesiųsti atsakymo arba išsiųsti jį vėliau
  - siuntėjas gali nelaukdamas atsakymo išsiųsti kelis pranešimus iš karto
  - atsiranda daugiau galimybių įsiterpti trečioms šalims, panaudoti tarpininkus

Document-style trūkumai

- Abstraktumas
  - Siuntėjas su gavėju turi kaip nors patys susitarti dėl siunčiamų dokumentų struktūros, t.y. reikalinga bendra schema
- Daugiau darbo programuotojui
  - Pranešimų konstravimas ir analizė gula ant programuotojo pečių, nes tik jis supranta siunčiamų dokumentų semantiką
  - Gudrūs įrankiai palengvina šią užduotį, susiedami dokumento struktūras su programos duomenų tipais. Aišku, tuomet sumažėja lankstumas (bet ne tiek, kiek RPC-style servisuose)