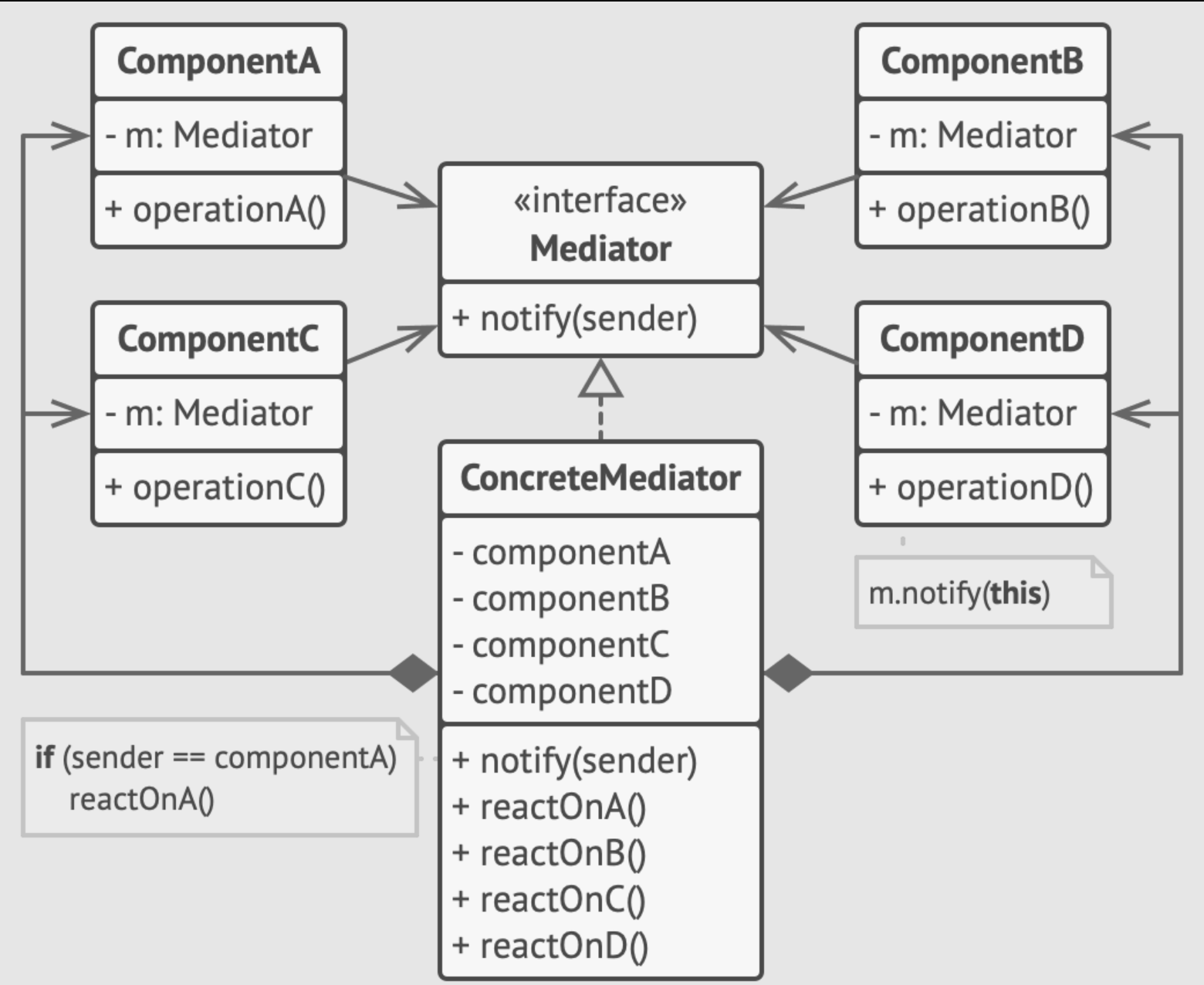
Krovinių pergabenimo kompanija turi sukurtą sistemą (pav1) kurioje automatizuojamos sistemos: Projektų vežimo kelio sudarymas, integracija su „Alps“ keltų sistema, klientų užsakymų svetainės servisas, integracija su darbuotojų buhalterijos programa, saugomi vilkikų, darbuotojų, užsakymų duomenys. Visa sistema veikia kompanijos patalpoje esančiame serveryje kaip viena ASP.NET aplikacija.

Vartotojai prašo naujų integracijų su keltų sistemomis ir nėra patenkinti užsakymų svetainės veikimo spartumu, kompanijos darbuotojai nepatenkint, jog buhalterijos programa lėtai veikia, nes ilgai traukia duomenis iš sistemos duomenų bazės. Kompanija nusprendė perdaryti esamą sistemą.

Kursinio darbo tikslas įvertinti panašios į pateiktą pavyzdį, monolitinės sistemos trūkumus, ir pateikti perdarytos sistemos į microservisus pranašumus ir trūkumus naudojant skirtingus mikroservisų komunikavimo būdus. Atsižvelgiama į REST, GRAPHQL, žinučių eilė komunikacijos būdus. Darbas pagrįstas iš .Net programavimo perspektyvos.

Komunikavimo būdai:

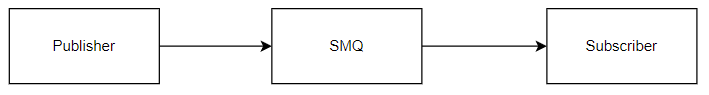
Monolitinėje aplikacijoje komunikacija vyksta programos viduje, dažniausiai naudojant dam tikrą komunikavimo stilių. Paprasčiausias, kai metodas kviečia kitą metodą tiesiogiai, tačiau yra komunikavimo modelių kuria naudoja pagalbines klases, pavyzdžiui mediatoriaus modelis:

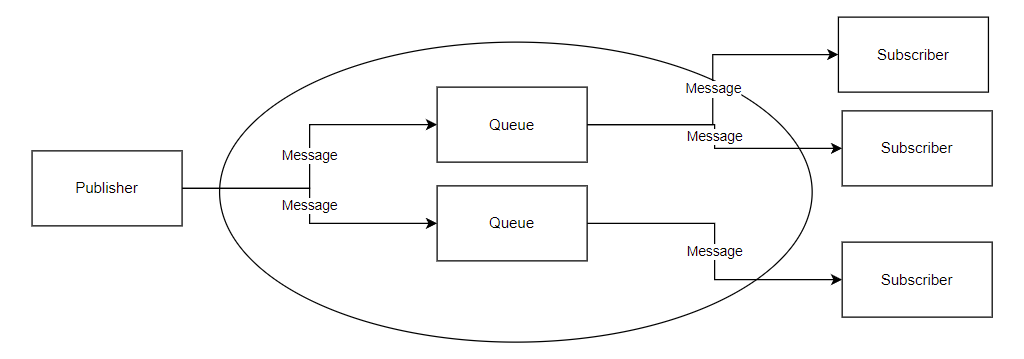


Mediatorius žino, kuris komponentas atsakingas už kurio įvykio apdorojimą ir išsiunčia įvykį atitinkamiems komponentams. Mediatoriaus komunikacijos stilius labai panašus į žinučių eilės komunikaciją tarp mikroservisų.

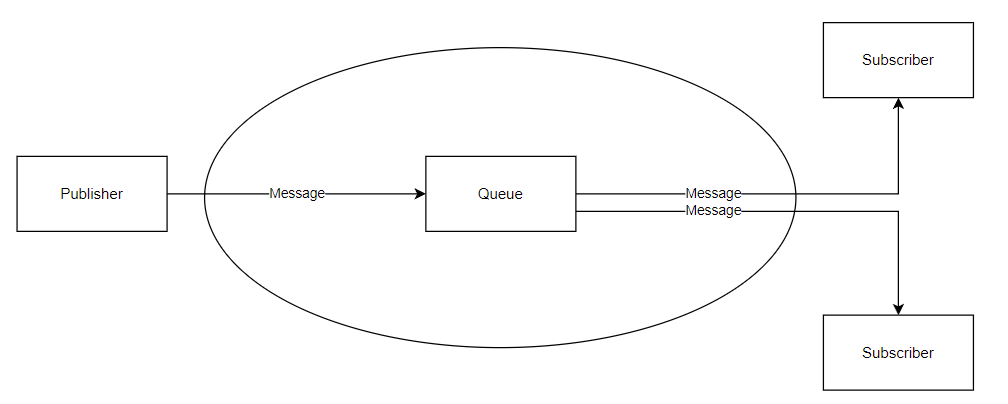
**Žinučių eilė**

Žinučių eilė ­­­tai yra atskiras servisas, kuris valdo žinučių siuntimą tarp servisų. Vienas servisas išsiunčia žinutę į žinučių eilę, o eilė išsaugo gautas žinutes savo duombazėje (priklausimai nuo žinučių eilės įgyvendinimo, duomenų bazė gali būti programos atmintyje arba išliekanti). Kitas servisas prenumeruoja žinutes. Jei žinučių eilė turi žinučių, ji žinutes perduoda prenumeratoriui.

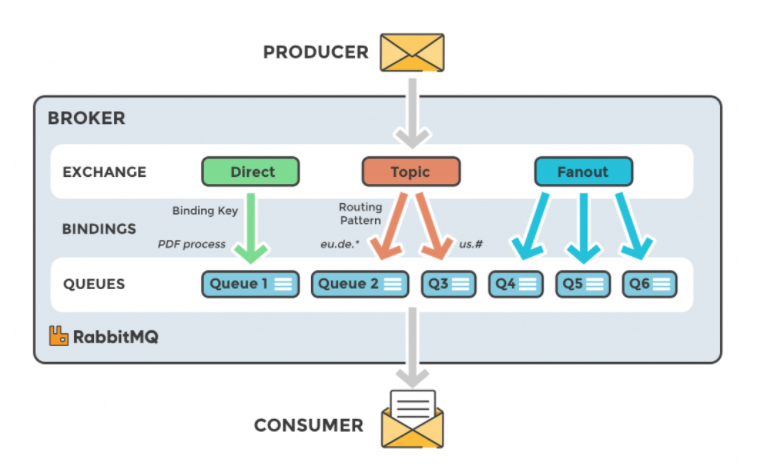




Žinučių eilės (pavyzdžiui AWS SQS arba RabbitMQ) turi ir galimybę žinutes siųsti visiems prenumeratoriam vienu metu. Tai reikalinga, pavyzdžiui, jeigu yra poreikis, kad visi prenumeratoriai gautų informaciją apie įvykį.



Tokiu atveju reikia pasirūpinti jog įvykiai nesusidubliuotų, jeigu naudojamas LOAD BALANCER daugiau nei vienu servisu viduje. Apie tai daugiau: Sqs and load balancing.

Dar gal kalbėti ir apie rabbit mq exchanges, reikia išsiaiškint ar aws toki turi

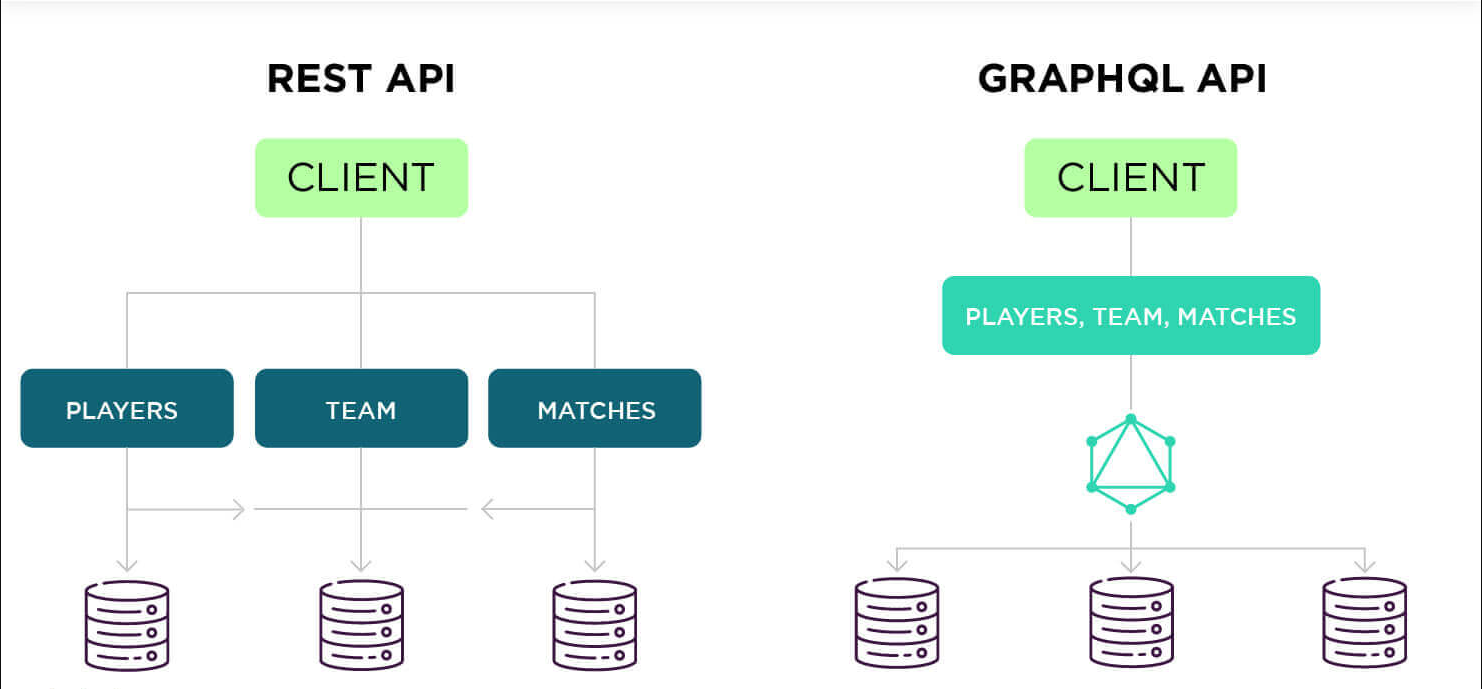
**Iš vidinio komunikavimo į tarp servisų komunikavimą**

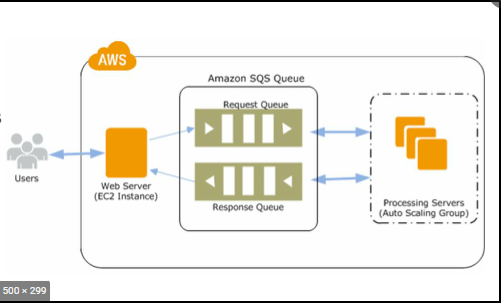
Tiesioginis komunikavimas tarp monolite paslėptų kontekstų ir vidinio komunikavimo nelabai skiriasi, tačiau kai perkeliame komunikavimą iš vienos aplikacijos į skirtingus servisus pagal kontekstus atsiranda problema kaip servisam komunikuoti tarpusavyje. Pagrinde yra du komunikavimo stiliai:

1. REST: Reprezentacinės būsenos perkėlimas yra architektūrinis stilius, nurodantis apribojimus, pvz., vienodą sąsają, kurią pritaikius žiniatinklio paslaugai, atsiranda pageidaujamų savybių, pvz., našumo, mastelio ir modifikavimo, leidžiančių paslaugoms geriausiai veikti Žiniatinklis.
2. GraphQL: yra API užklausų kalba, skirta šioms užklausoms atlikti naudojant esamus duomenis. GraphQL pateikia išsamų ir suprantamą API duomenų aprašymą, suteikia klientams teisę prašyti būtent to, ko jiems reikia ir nieko daugiau, palengvina API plėtrą laikui bėgant.

REST sukurtas anksčiau nei GRAPHQL, ir kol kas turi didesnę

|  |  |  |  |
| --- | --- | --- | --- |
| Ypatybės | REST | GRAPHQL | Žinučių eilė |
| Mokymosi greitis | Greitas | Vidutinis | Greitas |
| Sudėtingumas | Lengvas | Vidutinis | Lengvas/Priklauso nuo naudojimo serviso |
| Pradinis funkcionalumo įgyvendinimo laikas | Labai greitas | Vidutinis | Greitas |
| Kaip lengva išplėsti esamą funkcionalumą | Lengva | Labai lengva arba net nereikalauja keisti kodo iš serviso pusės. | Lengva |
| Pranašumai | Greitas žinių įsisavinimas, lengva naudoti, daug programuotojų jau žino šį komunikacijos būdą ir nereikia papildomai leisti resursus apmokinimui, pakankamai greitai gali pridėti naujo funkcionalumo | Panaikina „Overfetch“ problemą, gerai sukonfigūravus papildomus laukus kliento servisas gali pasiimti be papildomo serviso programavimo, sumažina interneto komunikacijos išlaidas, nes paima tik tiek resursų kiek reikia, leidžia lengvai sukurti GRAPHQL komunikacijos tinklą į kurį leidžia prijungti ir REST servisus. | Asyncroninis komunikavimas tarp servisų leidžia efektyviai išnaudoti servisų resursus, nulūžusieji servisai neturi tiek įtakos mikroservisų tinklui, kaip sinchroninis REST arba GRAPHQL servisų komunikavimas. |
| Trūkumai | „Overfetch“ problema, papildomas funkcionalumas tame pačiame endpointe, gali turėti didelį poveikį jau integruotoms sistemoms, sulėtėjęs serviso darbas tam tikrais atvejais\* | Gan naujas komunikavimo būdas, gali tekti apmokyti programuotojus, gal lėtas įsisavinimo lygis palyginus su REST arba žinučių eile, pradžioje konfigūravimas užtrunka ir pradinis funkcionalumas įgyvendinamas vėliau nei kiti komunikavimo būdai. | Papildomas sudėtingumas naudojant asinchroninį komunikacijos būdą. |





Sqs and load balancing

Literatūra

<https://refactoring.guru/design-patterns/mediator>

<https://geekflare.com/rabbitmq-hosting-platform/>

https://docs.oracle.com/javaee/6/tutorial/doc/gijqy.html