SEArch: an execution infrastructure for service-based software systems

Carlos G. Lopez Pombo  Pablo Montepagano  Emilio Tuosto

Coordination 2024
Tool track
June 19, 2024
– Prelude –
What is this talk about?

Service Execution Architecture in a nutshell

A PoC platform for semantic-based service composition

Bisimilarity as a semantic notion of compliance to search for and compose distributed service with support for multi-language programming (language-independence via choreographic models)
What is this talk about?

**Service Execution Architecture in a nutshell**
A PoC platform for semantic-based service composition

**Bisimilarity** as a semantic notion of **compliance**

to \{ search for and compose \}

distributed service with support for

multi-language programming

(language-independence via choreographic models)
Plan of the talk

An bird-eye watch of SEArch’s choreographic model
Plan of the talk

An bird-eye watch of SEArch’s choreographic model

An overview of SEArch and it’s design
Plan of the talk

An bird-eye watch of SEArch’s choreographic model

An overview of SEArch and it’s design

A “meta-demo”
Plan of the talk

An bird-eye watch of SEArch’s choreographic model

An overview of SEArch and it’s design

A “meta-demo”

Conclusions
– The underlying theory of SEArch –

& it’s architecture
A theory of software architectures of SOAs featuring provide and required interfaces

search and binding hinge on contracts
Asynchronous Relational Networks [FL:TCS (503) 2013]

A theory of software architectures of SOAs featuring provide and required interfaces

Contracts as CF-SMs [BZ:JACM 1983] according to [PVT:PLACES 2015,Vis:PhD 2018]
Asynchronous Relational Networks [FL:TCS (503) 2013]

A theory of software architectures of SOAs featuring provide and required interfaces

Contracts as CF-SMs [BZ:JACM 1983] according to [PVT:PLACES 2015,Vis:PhD 2018]
SEArch, conceptually
Service providers register their contract and URI to a broker
a client’s request specifies the (set) of contracts partners should fulfil
SEArch, conceptually

The broker searches for compatible providers...

Diagram:
- Broker
- Repository
- Middleware
- Client
- Pps
- Srv
SEArch, conceptually

...collects the compatible providers...
SEArch, conceptually

...and returns them to the client
SEArch, conceptually

All components can now interact
A provider may require other services; just ask the broker...
– From theory to practice –
async def main(grpc_channel):
    stub = search.PrivateMiddlewareServiceStub(grpc_channel)
    registered = False
    logger.info("Connected to middleware. Waiting for registration...")
    async for r in stub.register_app(
        search.RegisterAppRequest(
            provider_contract=search.LocalContract(
                format=search.LocalContractFormat.LOCAL_CONTRACT_FORMAT_FSA,
                contract=PROVIDER_CONTRACT,
            )
        )
    )
    if registered and r.notification:
        logger.info(f"Notification received: {r.notification}")
        # Start a new session for this channel.
        asyncio.create_task(session(grpc_channel, r.notification))
    elif not registered and r.app_id:
        # This should only happen once, in the first iteration.
        registered = True
        logger.info(f"App registered with id {r.app_id}\")
        # Create temp file for Docker Compose healthcheck.
        with open("/tmp/registered", "w") as f:
            f.write("OK")
    else:
        logger.error(f"Unexpected response: {r}. Exiting.")
        break

grpc_channel.close()
```go
const ppsContract = `.
.outputs PPS
.state graph
q0 ClientApp ? CardDetailsWithTotalAmount q1
q1 ClientApp ! PaymentNonce q2
q2 Srv ? RequestChargeWithNonce q3
q3 Srv ! ChargeOK q4
q3 Srv ! ChargeFail q5
.marking q0
.end
` // the CFSM in ChorGram syntax

func main() {
    flag.Parse()
    var opts []grpc.DialOption
    opts = append(opts, grpc.WithTransportCredentials(insecure.NewCredentials()))
    conn, err := grpc.Dial(*middlewareURL, opts...)
    if err != nil {
        logger.Fatalf("Error connecting to middleware URL %s", *middlewareURL)
    }
    defer conn.Close()
    stub := pb.NewPrivateMiddlewareServiceClient(conn)

    // Register provider contract with registry.
    req := pb.RegisterAppRequest{
        ProviderContract: &pb.LocalContract{
            Contract: []byte(ppsContract), // passed to the broker upon registration
            Format: pb.LocalContractFormat_LOCAL_CONTRACT_FORMAT_FSA,
        },
    }
    streamCtx, cancel := context.WithCancel(context.Background())
    defer cancel()
    stream, err := stub.RegisterApp(streamCtx, &req)
    if err != nil {
```
```
public class Main {
    public static void main(String[] args) {
        // get book selection and shipping address from the user
        ByteString contractBytes = null; // Load file contract.fsa into a GlobalContract
        try {
            contractBytes = ByteString.readFrom(new FileInputStream("contract.fsa"));
        } catch (IOException e) {
            e.printStackTrace();
        }
            GlobalContractFormat.GLOBAL_CONTRACT_FORMAT_FSA
        ).setInitiatorName("ClientApp").build();
    }
}

where in contract.fsa we find:

```
.outputs ClientApp
.state graph
q0 Srv ! PurchaseRequest q1
q1 Srv ? TotalAmount q2
q2 PPS ! CardDetailsWithTotalAmount q3
q3 PPS ? PaymentNonce q4
q4 Srv ! PurchaseWithPaymentNonce q5
q5 Srv ? PurchaseOK q6
q6 Srv ? PurchaseFail q7
.marking q0
.end

.outputs Srv
.state graph
q0 ClientApp ? PurchaseRequest q1
q1 ClientApp ! TotalAmount q2
q2 ClientApp ? PurchaseWithPaymentNonce q3
q3 PPS ! RequestChargeWithNonce q4
q4 PPS ? ChargeOK q5
q5 PPS ? ChargeFail q6
.marking q0
.end

.outputs PPS
.state graph
q0 ClientApp ? CardDetailsWithTotalAmount q1
q1 ClientApp ! PaymentNonce q2
q2 Srv ? RequestChargeWithNonce q3
q3 Srv ! ChargeOK q4
q3 Srv ! ChargeFail q5
.marking q0
.end
```
– A meta-demo –

courtesy of Pablo Montepagano
Bookkeeping

Middleware
Client

Broker
Repository

Middleware
Pps

Middleware
Srv
Bookkeeping

\[ \text{req} (R = R_{\text{Client}} + R_{\text{Srv}} + R_{\text{Pps}}, \text{client}) \]
Bookkeeping

Middleware

Client

Broker

Repository

URIs

Middleware

Pps

Middleware

Srv
Bookkeeping
Check out the paper for the details about the protocols that generate these logs in the implementation of **SEArch**.
– Epilogue –
SEArch combines

- SOAs
- semantic models (ARNs + CFSMs)
- and tools for choreographic development (D. Senarruzza’s extension of ChorGram)

to enable dynamic and semantic-based discovery and composition of distributed services

There’s space for improvement

- data-aware CFSMs
- decouple broker and service repository
- \[ \Rightarrow \text{distributed bisimulation checks!} \]
- parameterise the compliance check
- what about mistakes/attacks?
Thanks to

Carlos is the main driver behind the design SEArch
Thanks to

Carlos is the main driver behind the design SEArch

Pablo realised SEArch
Thanks to

Carlos is the main driver behind the design SEArch

Pablo realised SEArch

Research partly supported by the EU H2020 RISE programme under the Marie Skłodowska-Curie grant agreement No 778233

Dipartimento di Eccellenza
Many thanks to the reviewers and to Ilaria & Francesco for helping us addressing a reviewer’s concerns.