

A prototypical tool to discover architecture changes based on multiple monitoring data sources for a distributed system

Patrick Schäfer, 08.11.2017, Munich
Advisor: Martin Kleehaus

Chair of Software Engineering for Business Information Systems (sebis)
Faculty of Informatics
Technische Universität München
www.matthes.in.tum.de

Motivation

- Modeled & Runtime Architecture
- Problem Statement

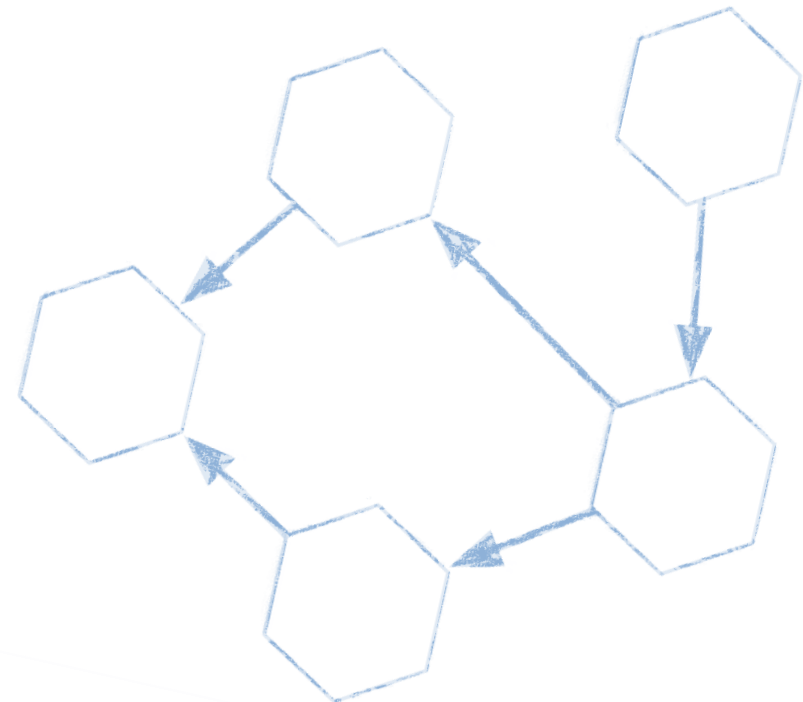
Research Questions

Prototypical Implementation

- Operating Environment
- Discovery Concept
- Implementation

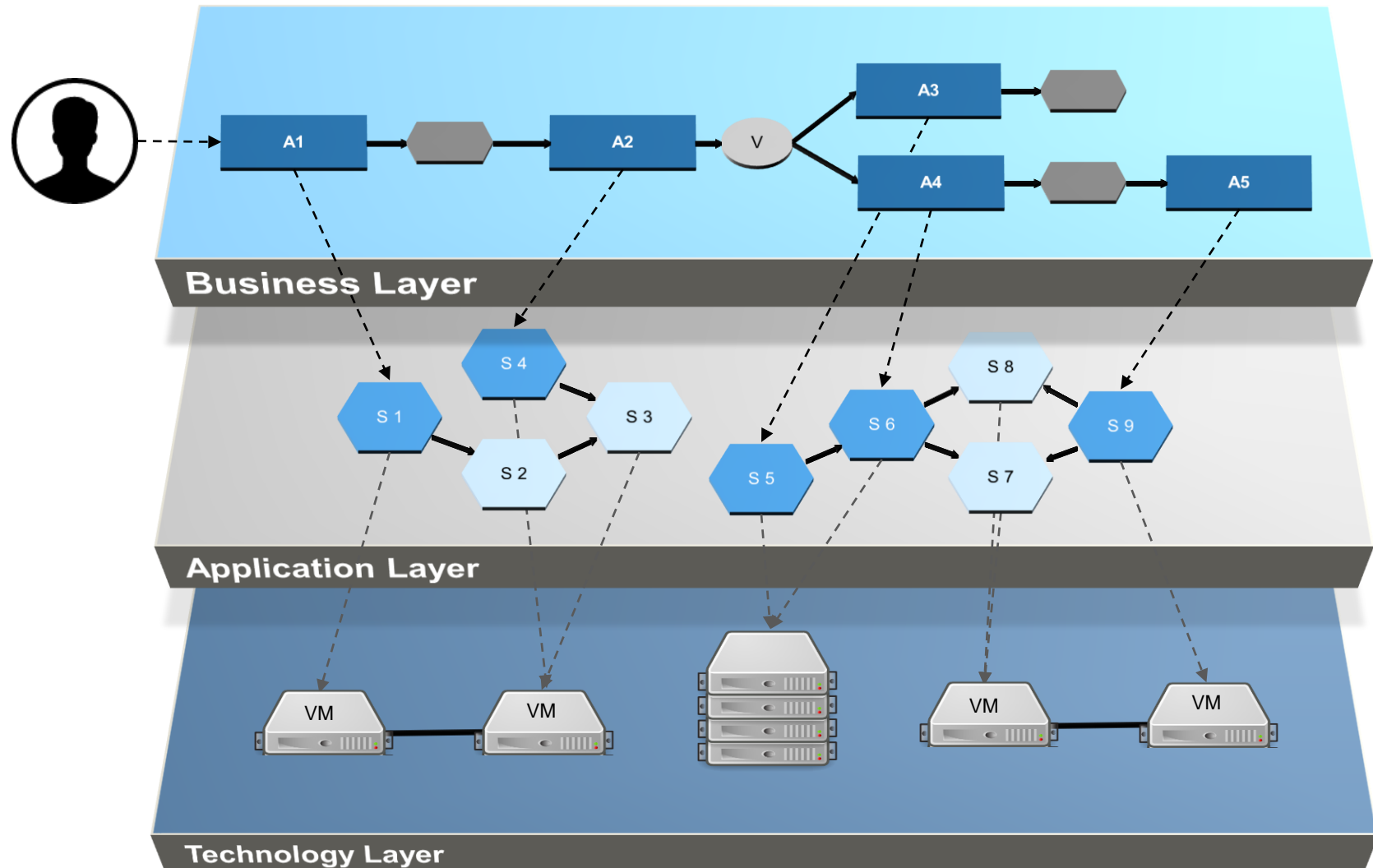
Live Demonstration

Limitations



Motivation

Modeled Enterprise Architecture

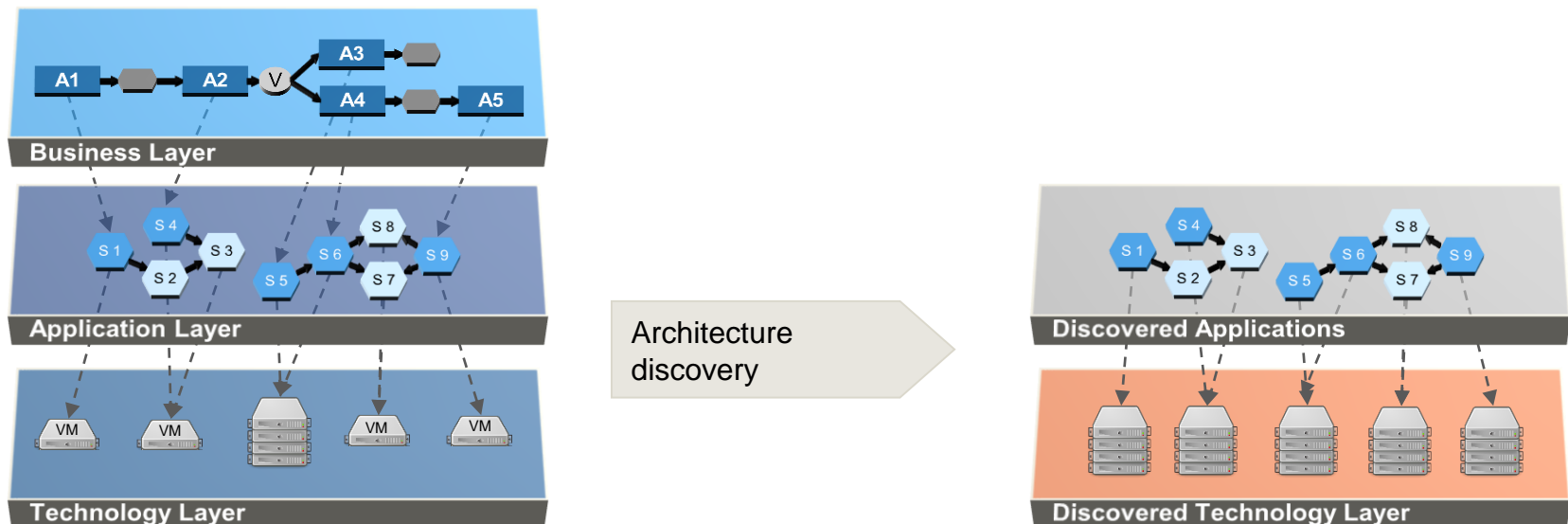


Necessity of knowledge about runtime architecture

- Detecting deviations from modeled architecture
- Compliance violations / SLA violations
- Semi-automated root cause analysis
- Semi-automated Impact analysis

State of the art solutions

Application Runtime Architecture Discovery



Problem:

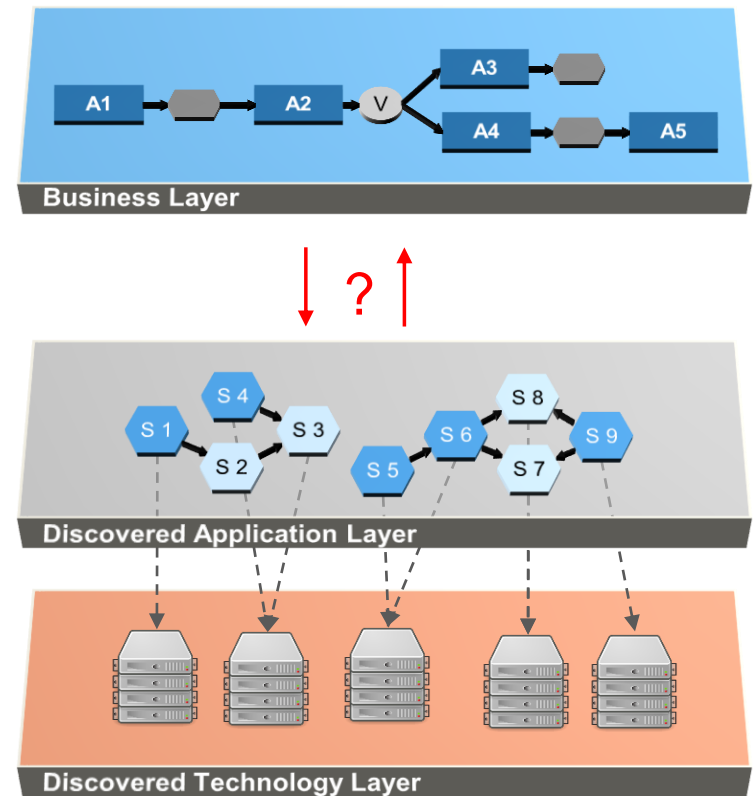
Current discovery solutions don't own knowledge about the entire architecture including dependencies between and within the architectural layers

Consequences

- comparison of modeled and runtime architecture not possible across all layers
- root cause analysis is challenging and time consuming
- Incomplete impact analysis

Objective

- a system, that owns knowledge about all realtime architecture components as well as their inter- and intralayer dependencies



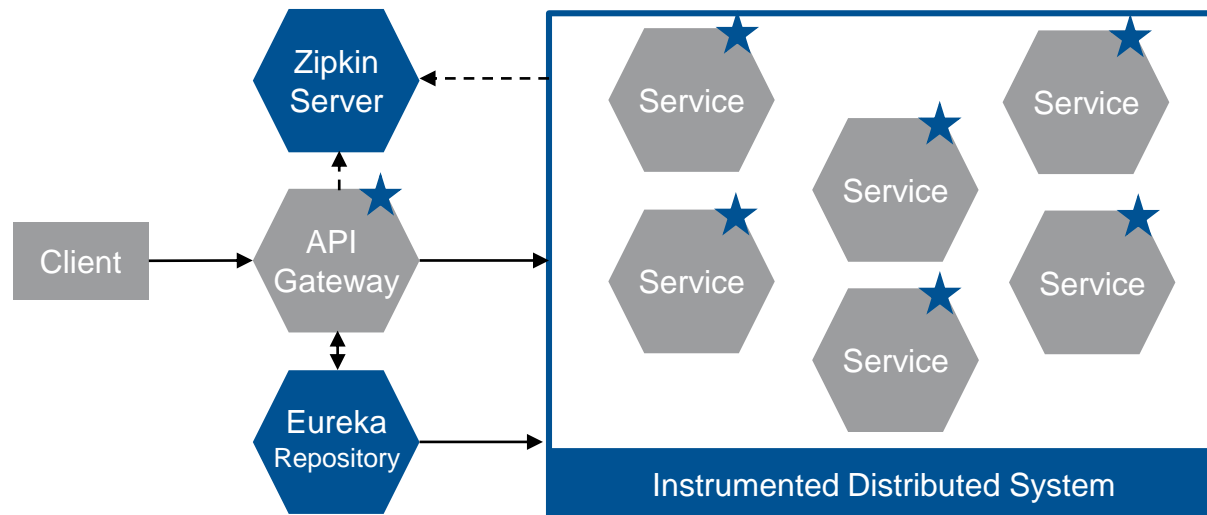
- 1 How to **discover the Microservice Architecture** by applying distributed tracing?
- 2 What are **component relationship types** and how to discover them automatically?
- 3 How to discover **concurrency and synchronization**?
- 4 How to **recognize changes** in the Microservice Architecture?
- 5 How to provide a **smart user interface** for adding **business semantic** on top of business services?

Prototypical Implementation

Operating Environment

Required components of the prototypes operating environment:

- Instrumented Services
- Service Repository Eureka
- Zipkin Server

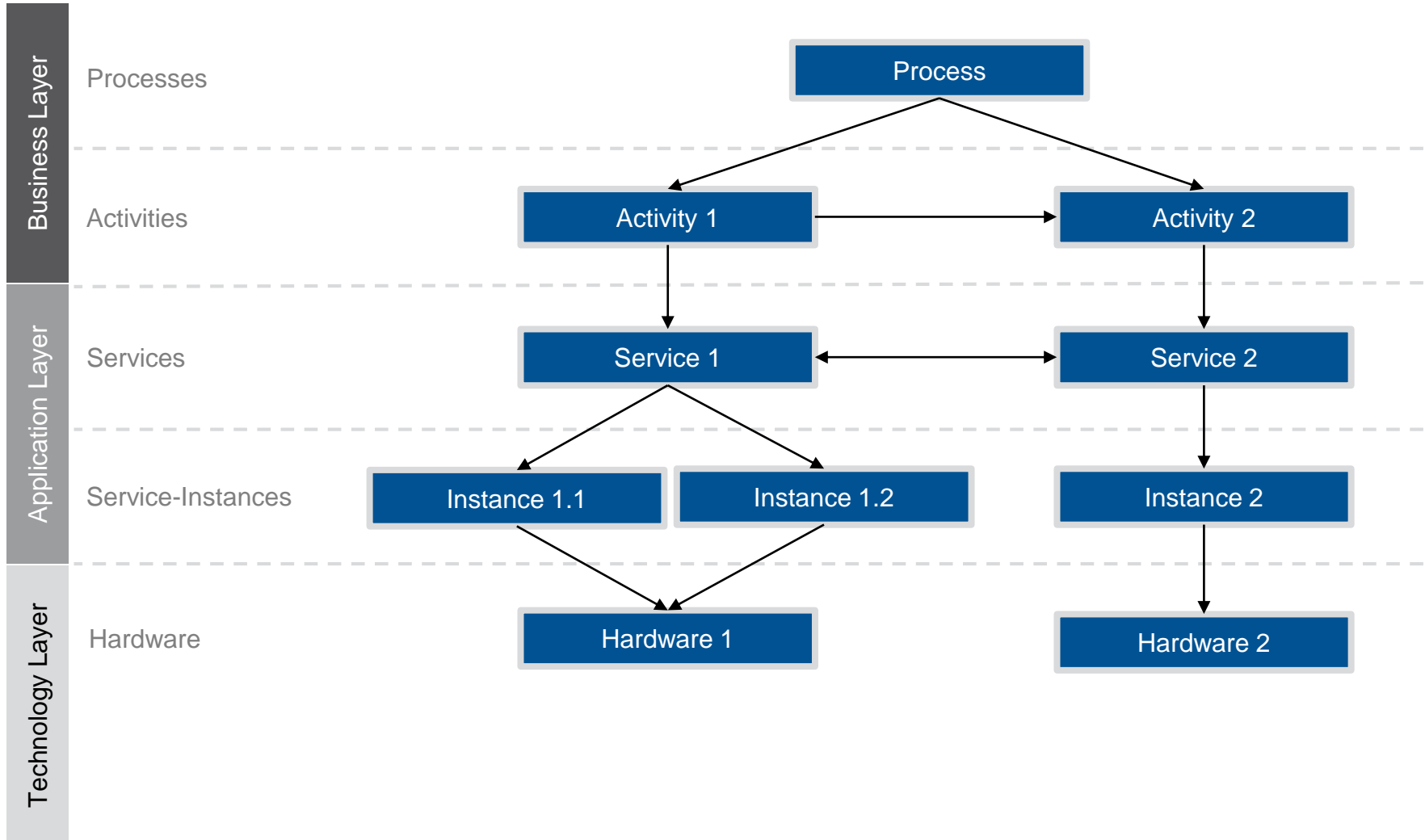


Legend: Required Component
 Optional Component

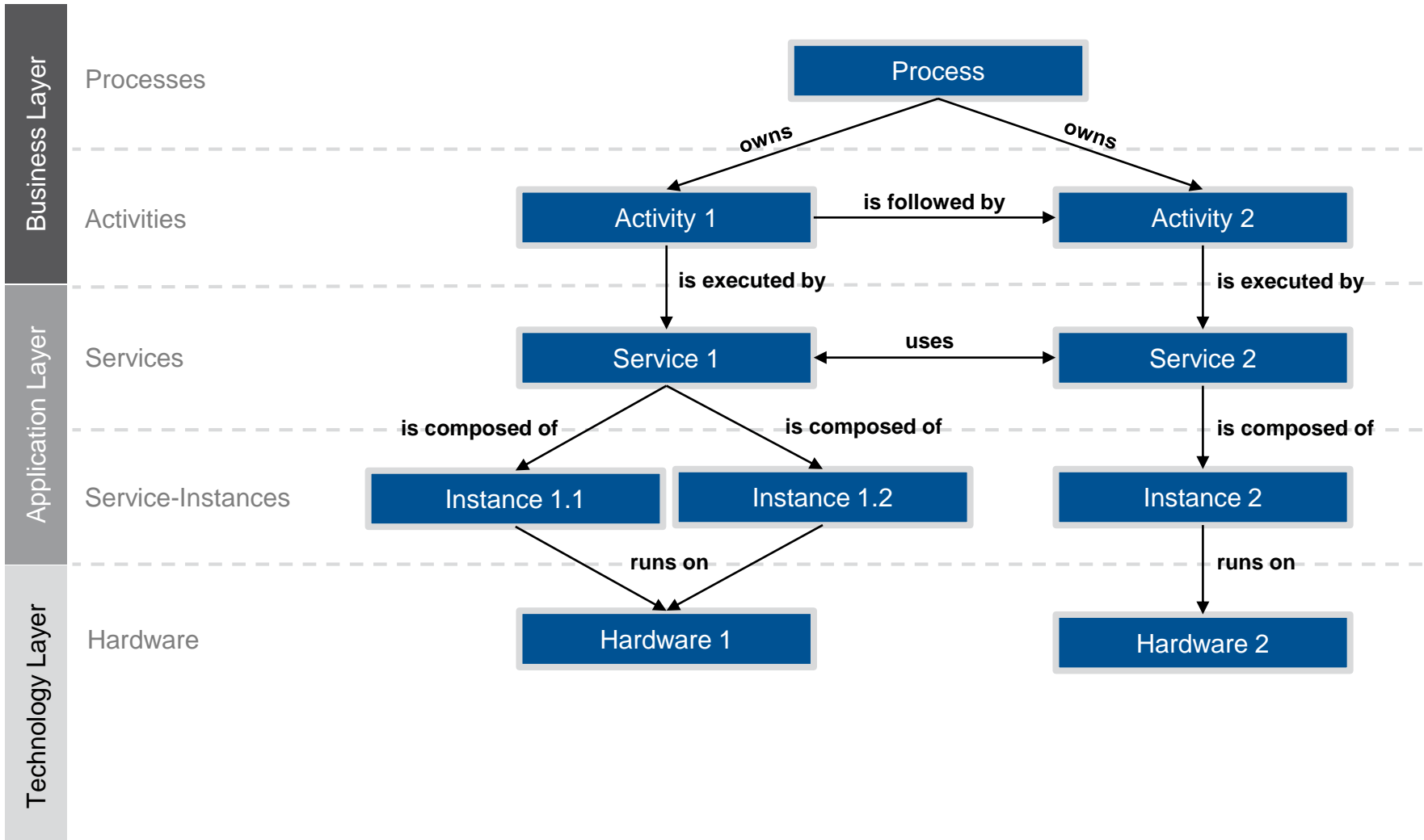
Instrumented Service
 Distributed Tracing Spans

Prototypical Implementation

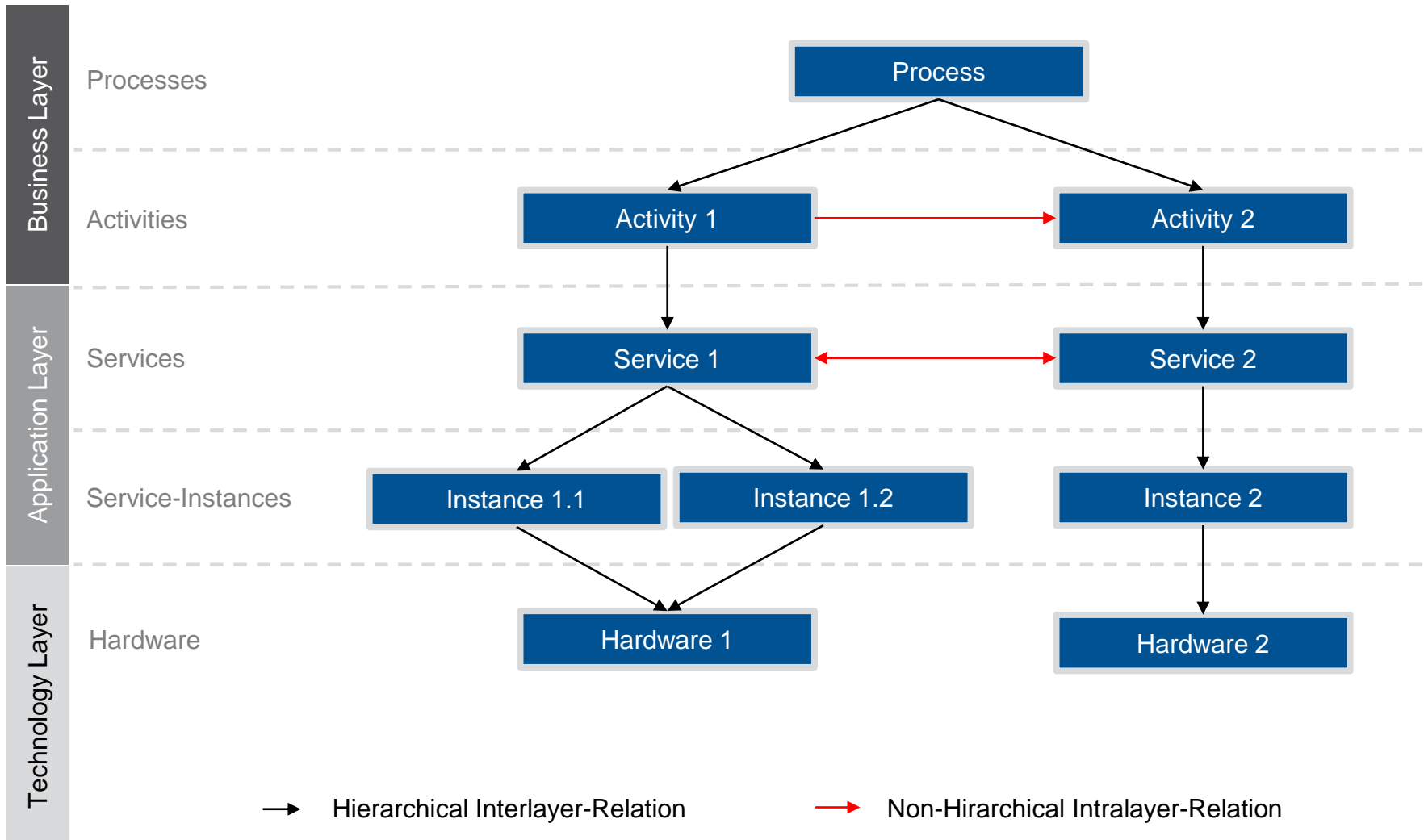
Discovery Concept: Architecture Model



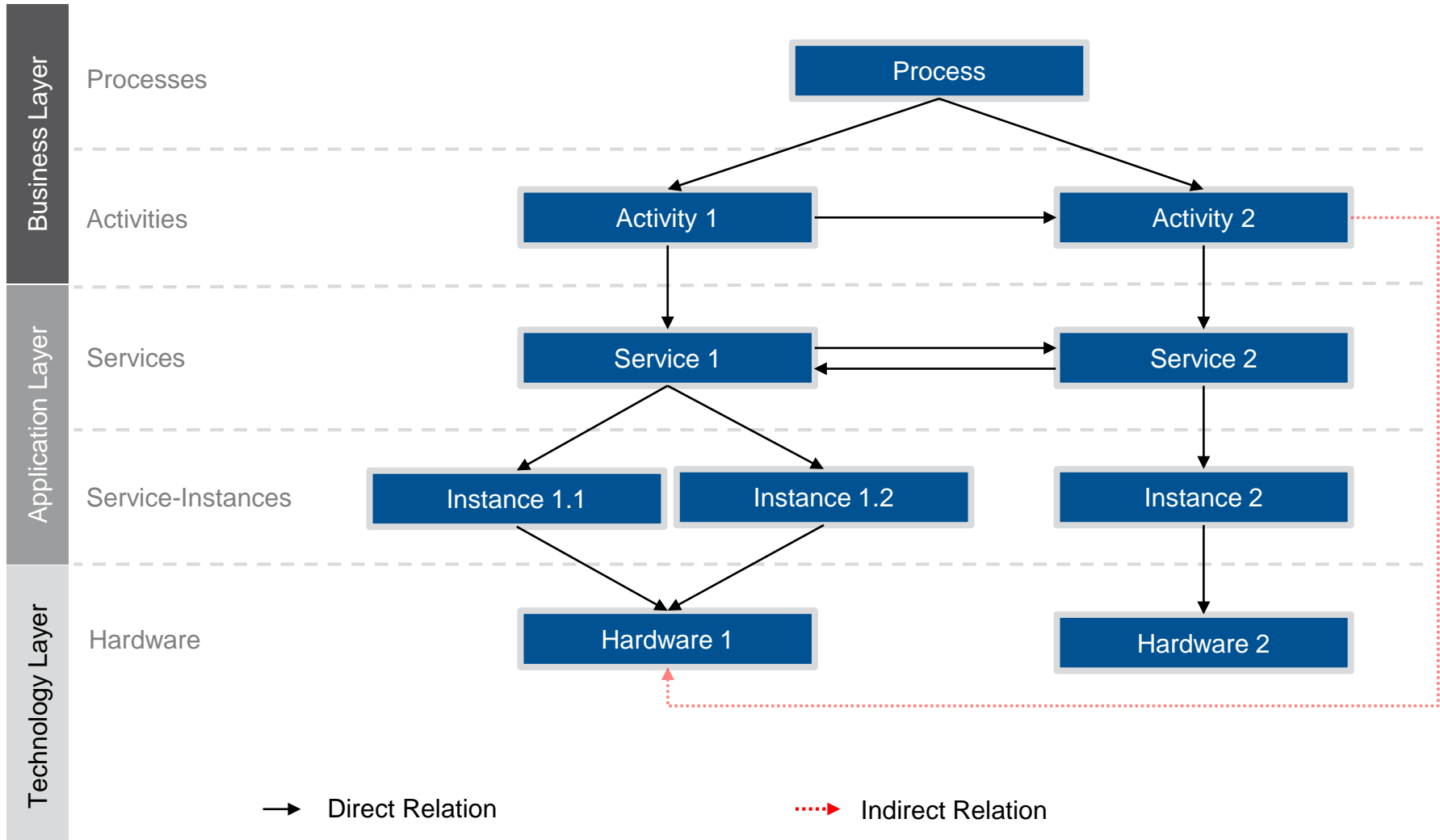
Implicit meaning of relations



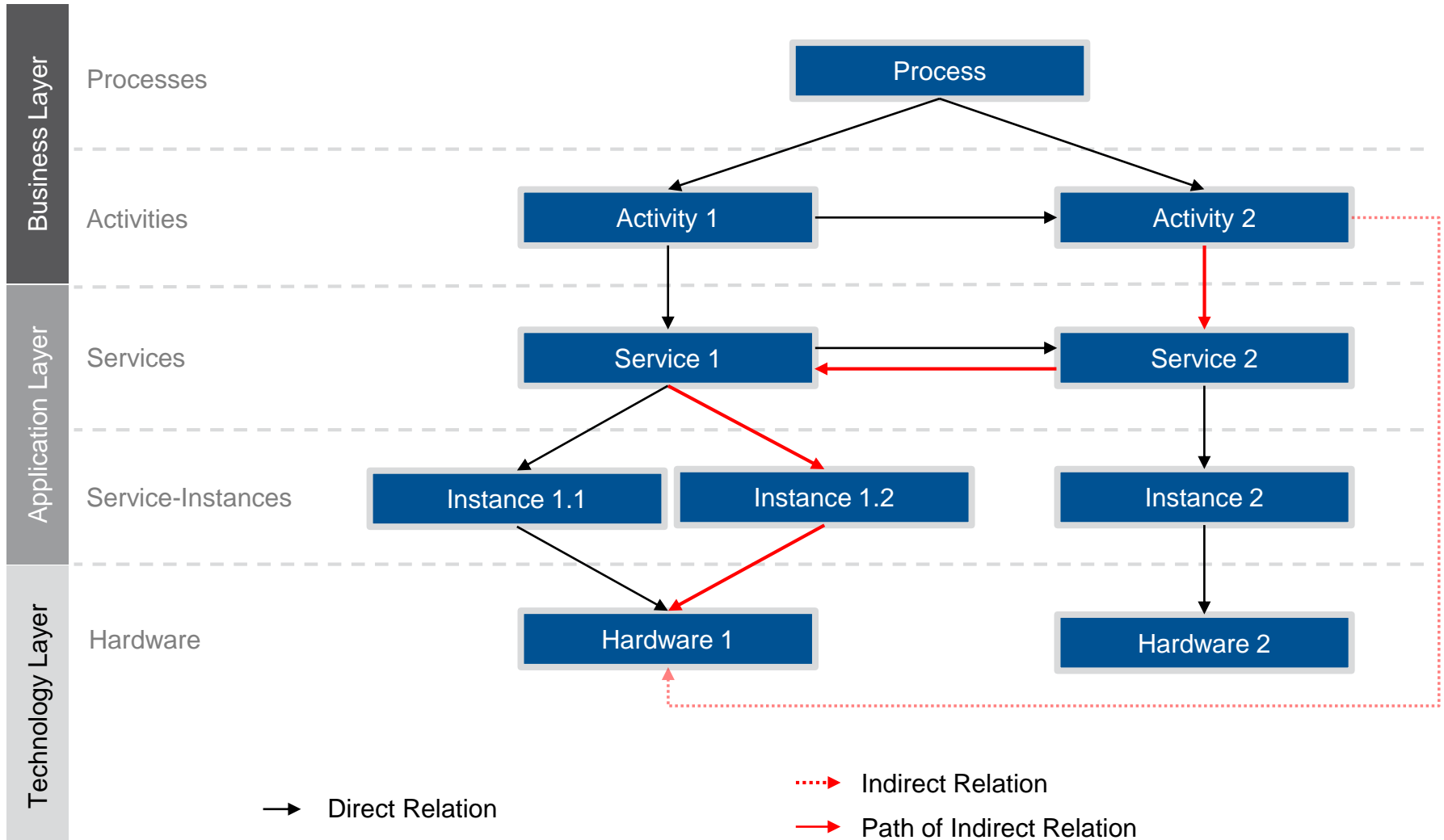
Relation classification: Interlayer- and Intralayer-Relations



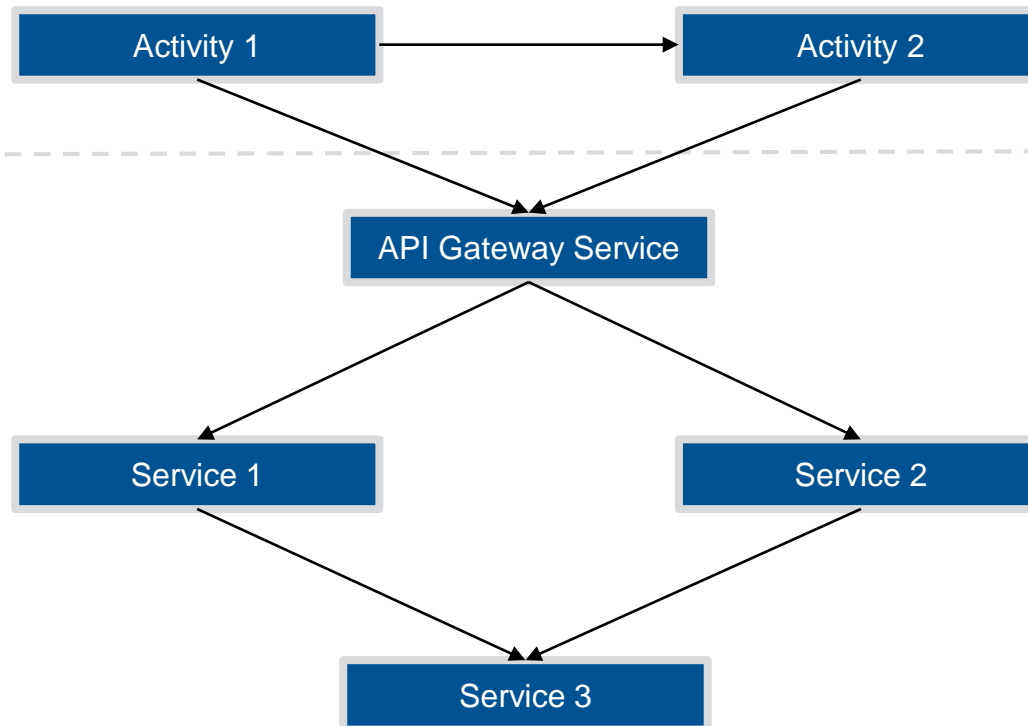
Relation classification: Direct and Indirect Relations



Relation classification: Direct and Indirect Relations



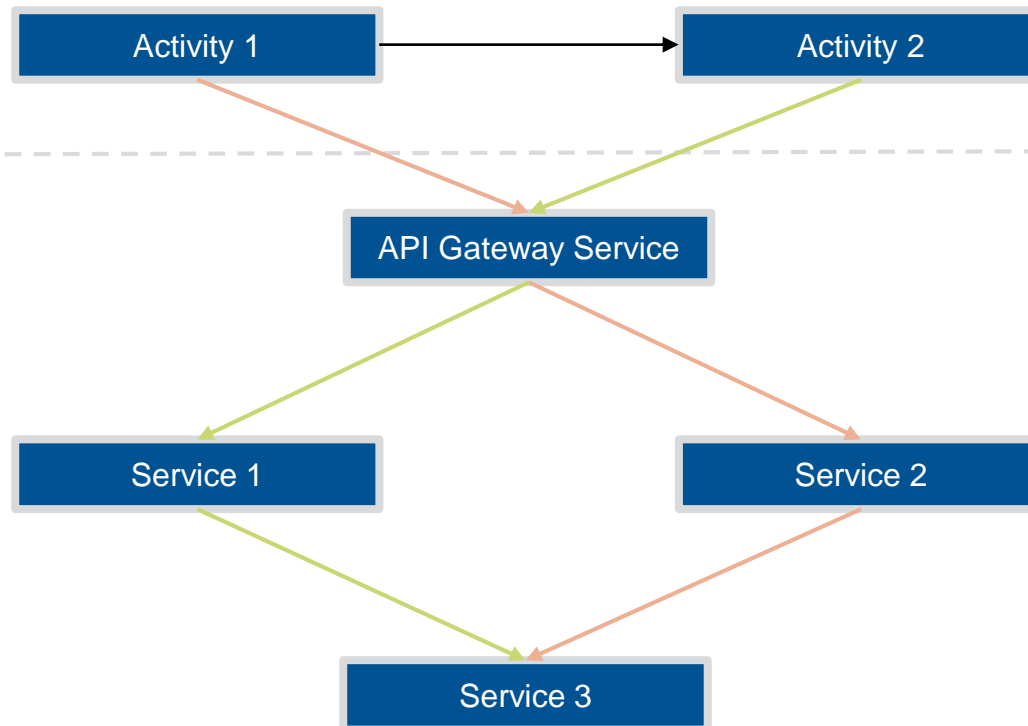
Indirect Relations are necessary



Prototypical Implementation

Discovery Concept: Architecture Model - Relations

Indirect Relations are necessary



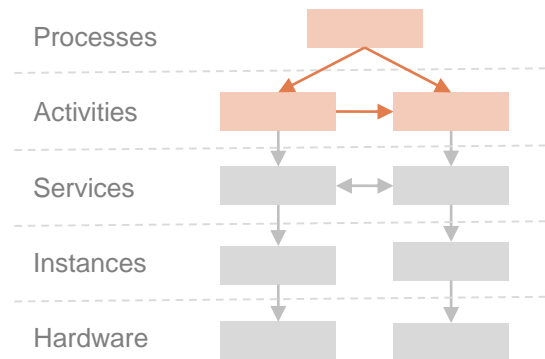
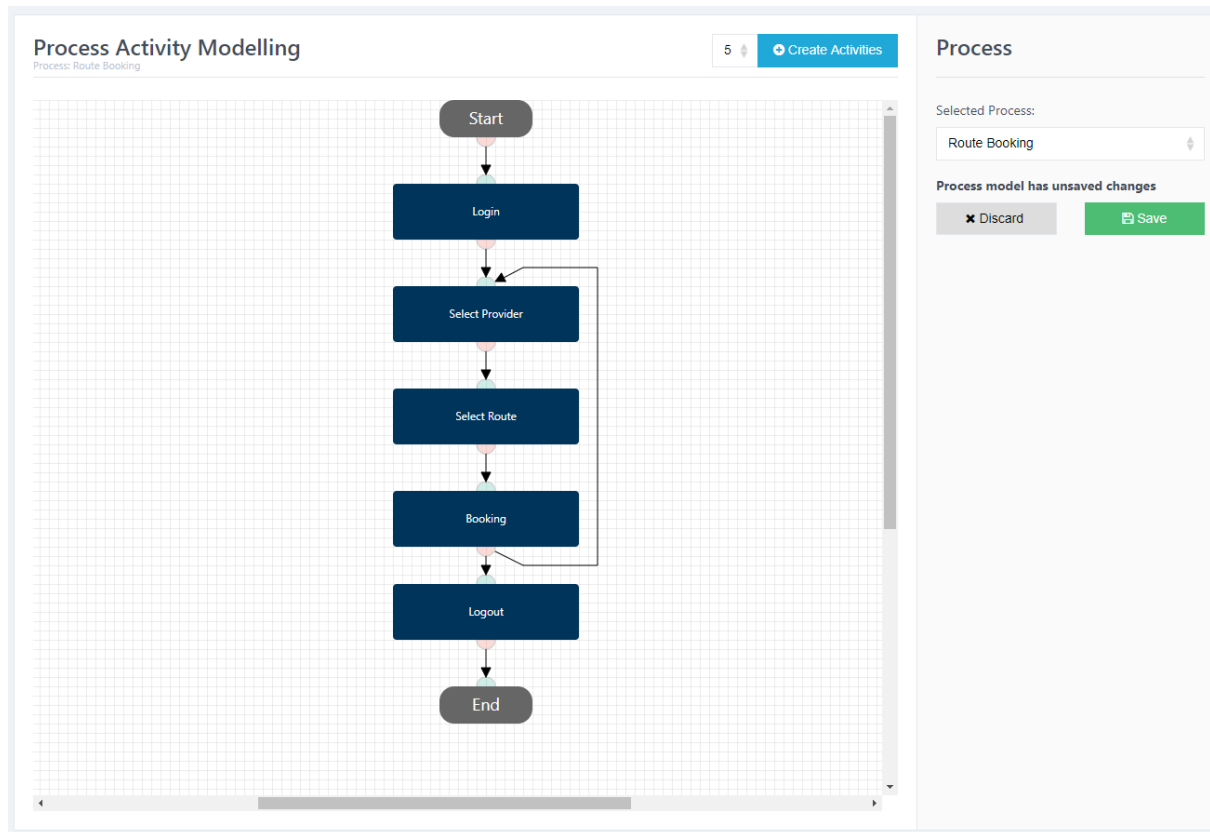
Prototypical Implementation

Discovery Concept: Business Layer

Modelling via Web UI:

- Process and Activity entities
- Process-flow

Added to the architecture model via REST API



Prototypical Implementation

Discovery Concept: Application / Technology Layer

Detection of

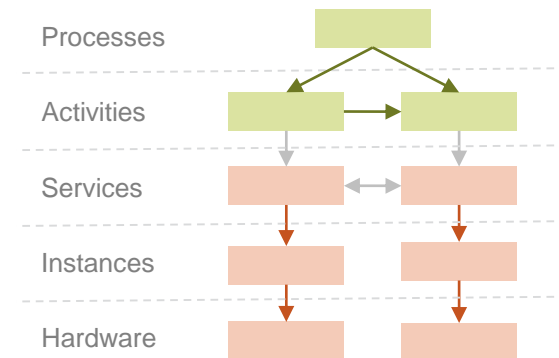
- Services, Instances, Hardware
- Interlayer-Relations of these types

In Realtime by analysing Distributed Tracing

- Detects new entities only

Cyclic by processing Eureka API

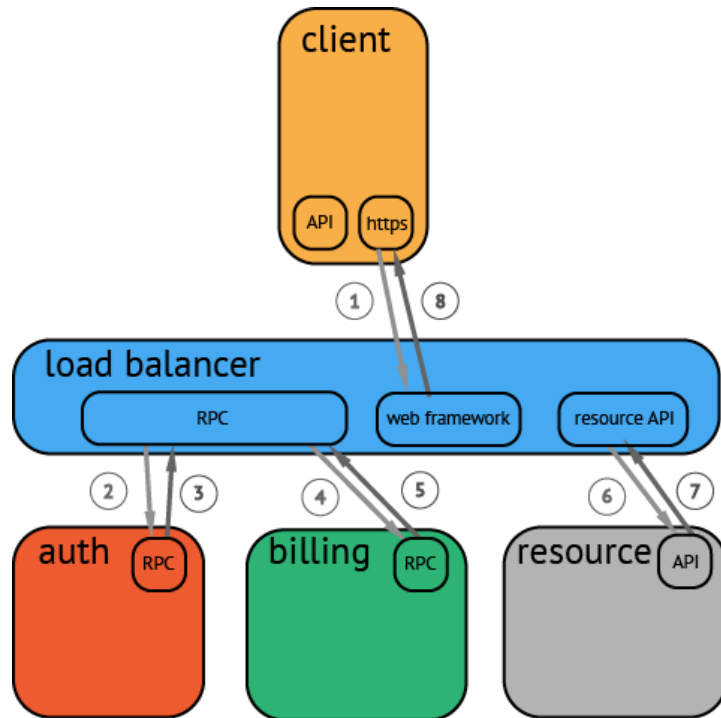
- Detects new, updated or removed entities



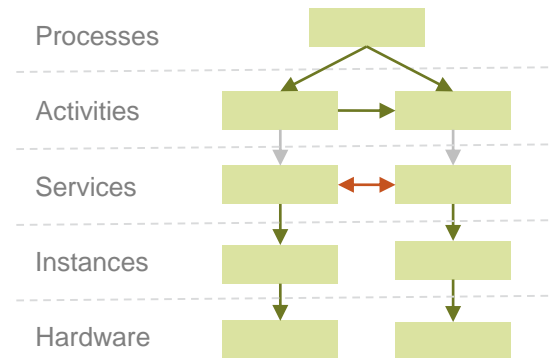
Prototypical Implementation

Discovery Concept: Service intralayer Relations

Detection of relations between Services
in Realtime by analysing Distributed Tracing Spans



Distributed Tracing Span production*



*Source: <http://opentracing.io/documentation/pages/instrumentation/instrumenting-large-systems.html>

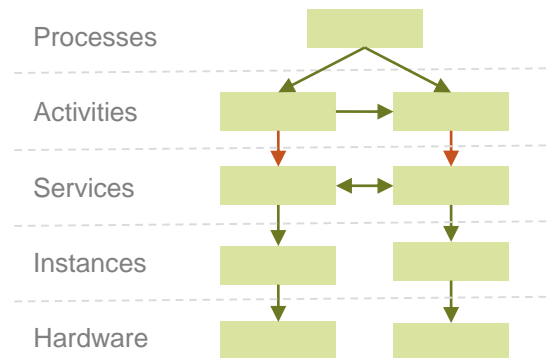
Prototypical Implementation

Discovery Concept: Business / Application Relations



Detection of Interlayer-Relations between Activities and Services in Realtime

- Set of regular expressions mapping to activities
- Set defined via Web UI and REST API
- Distributed Tracing Spans applied to ruleset

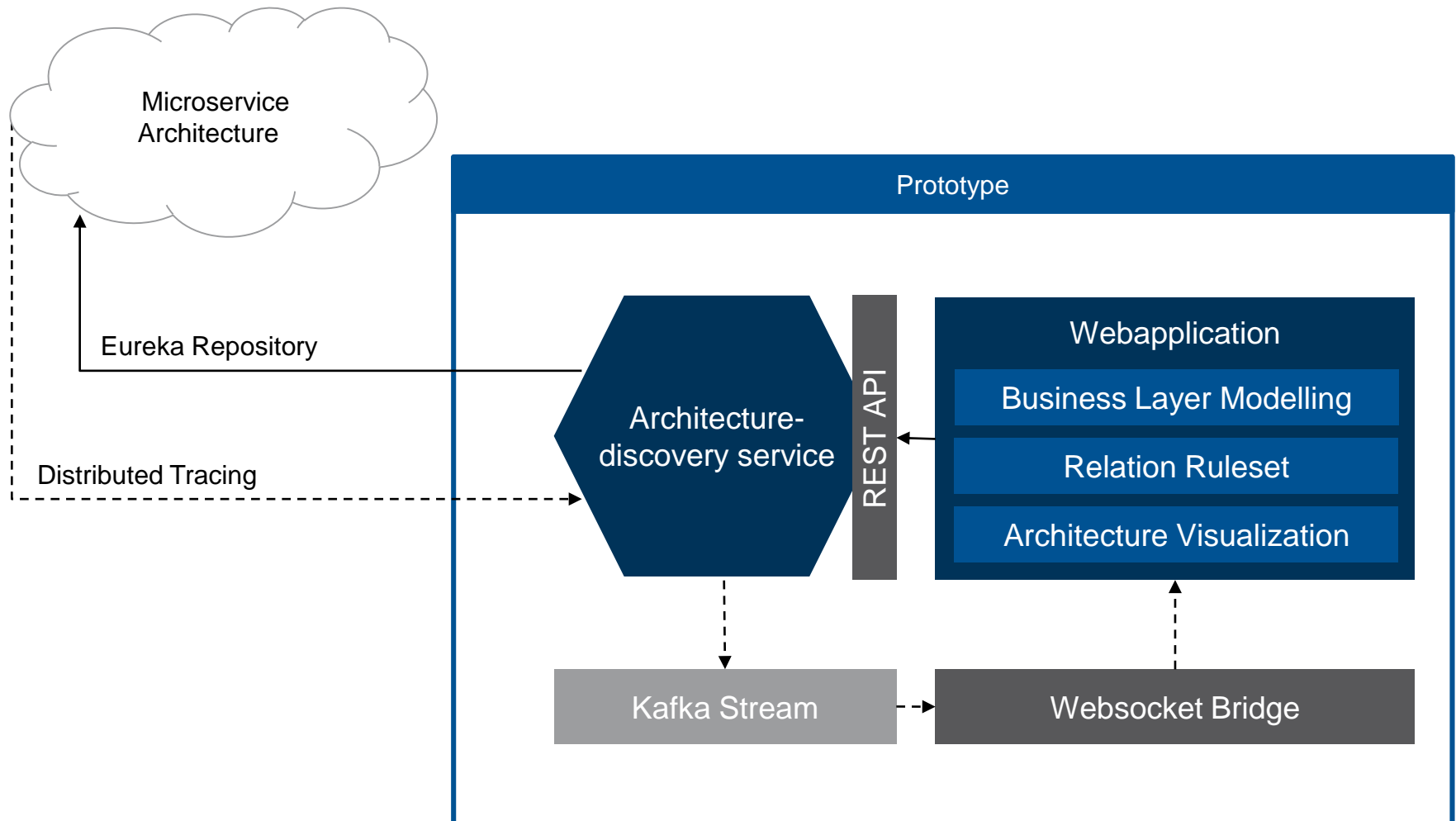


The screenshot shows a web interface with three tabs: "Simple Mapping", "Rule-based Mapping", and "Existing Mappings". The "Rule-based Mapping" tab is active. The interface is divided into three main sections:

- Unmapped paths:** A search bar labeled "Filter by path" is at the top. Below it, a list of paths is shown, each with a "GET" method and a "first seen" timestamp:
 - http/ (first seen: 2017-10-21 17:21:59)
 - http/business-core-service/businesses/list (first seen: 2017-10-21 17:21:59)
 - http/deutschebahn-mobility-service/routes (first seen: 2017-10-21 17:22:05)
 - http/deutschebahn-mobility-service/getroutes (first seen: 2017-10-21 17:22:12)
- Selected paths:** A list of paths that have been selected for mapping:
 - http/login (GET, first seen: 2017-10-21 17:21:59)
 - http/login (POST, first seen: 2017-10-21 17:22:03)
- Assign paths to activity:** A form for assigning the selected paths to an activity. It includes:
 - Selected Process: "Route Booking" (dropdown menu)
 - Selected Activity: "Login" (dropdown menu)
 - Buttons: "Discard" (with a close icon) and "Save" (with a save icon)

Prototypical Implementation

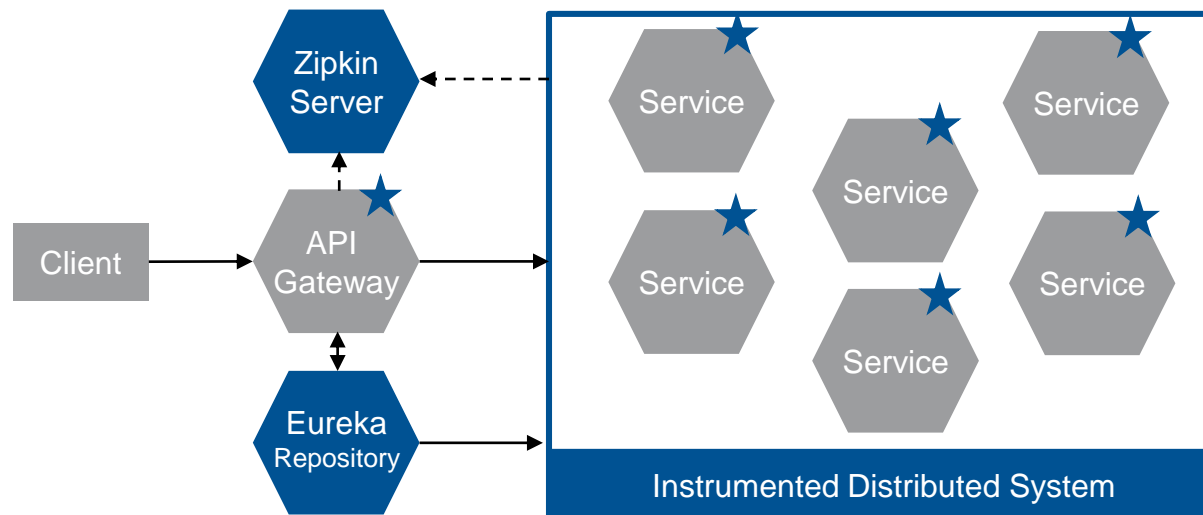
Implementation: System Components



Prototypical Implementation

Implementation: Environment

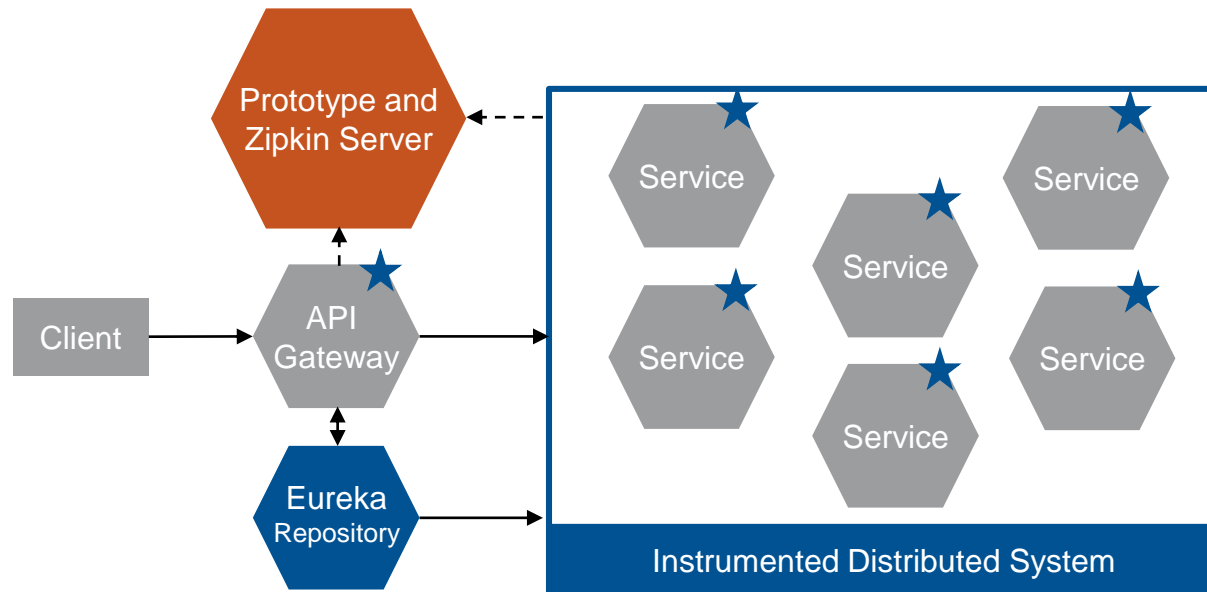
- Zipkin Server replaced by Prototype
- Prototype extends Zipkin implementation
- Zipkin Server Features stay complete and unmodified



Prototypical Implementation

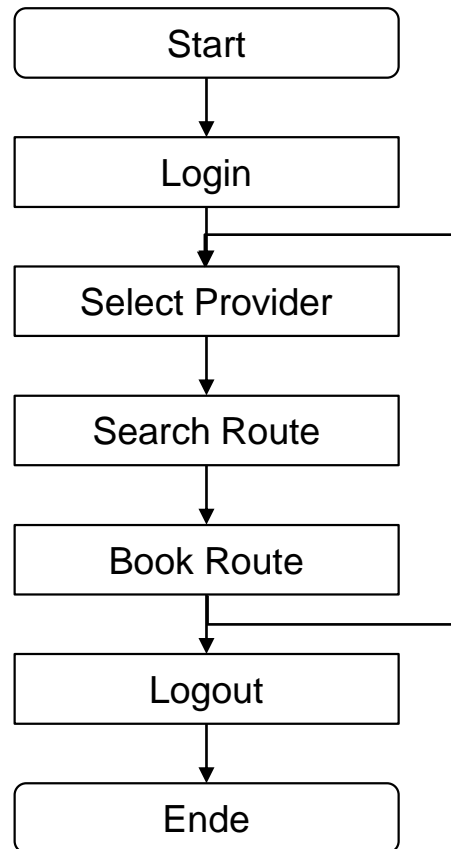
Implementation: Environment

- Zipkin Server replaced by Prototype
- Prototype extends Zipkin implementation
- Zipkin Server Features stay complete and unmodified



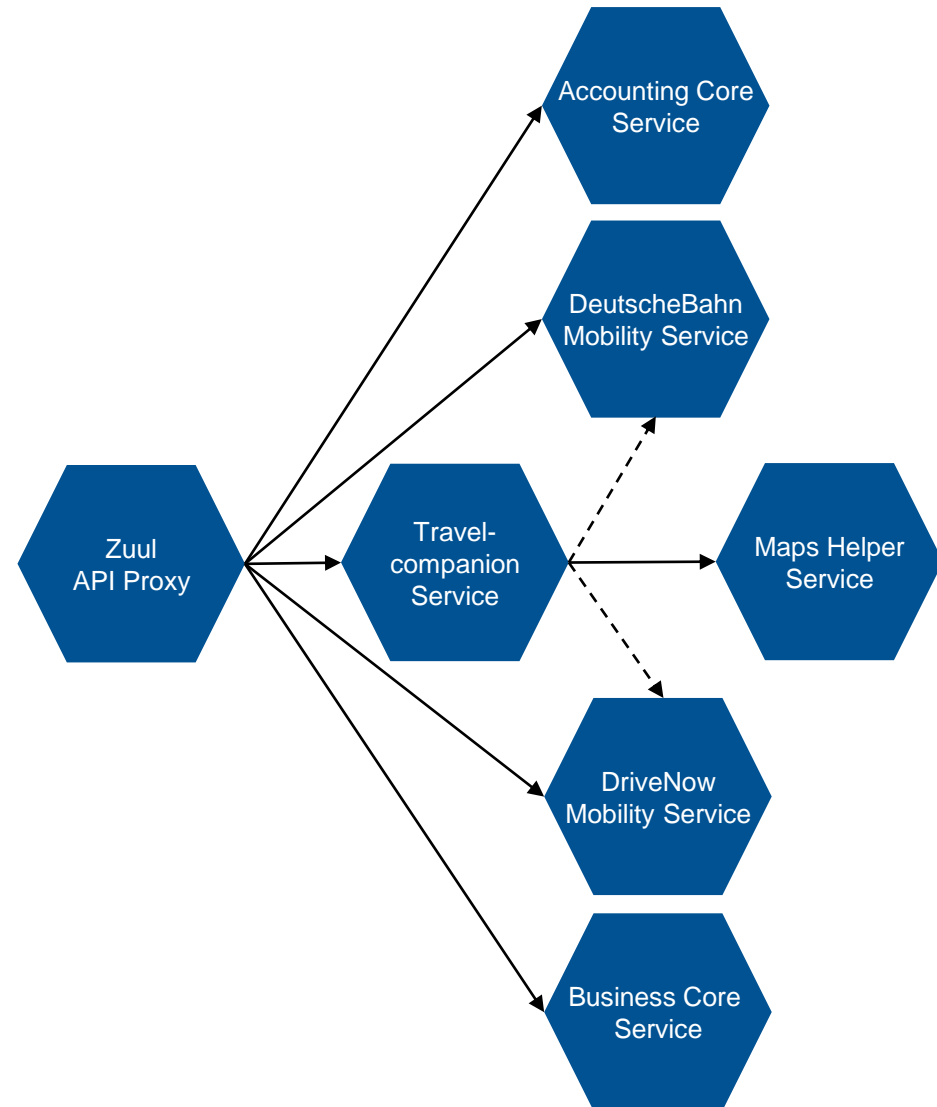
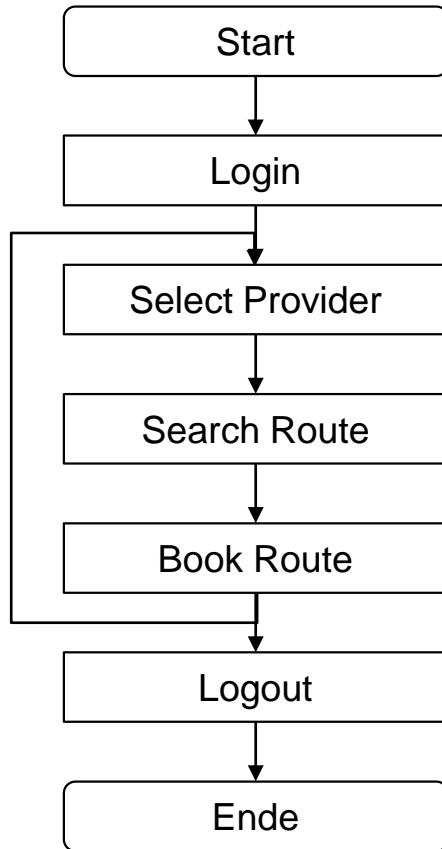
Live Demonstration

Scenario: Business Process



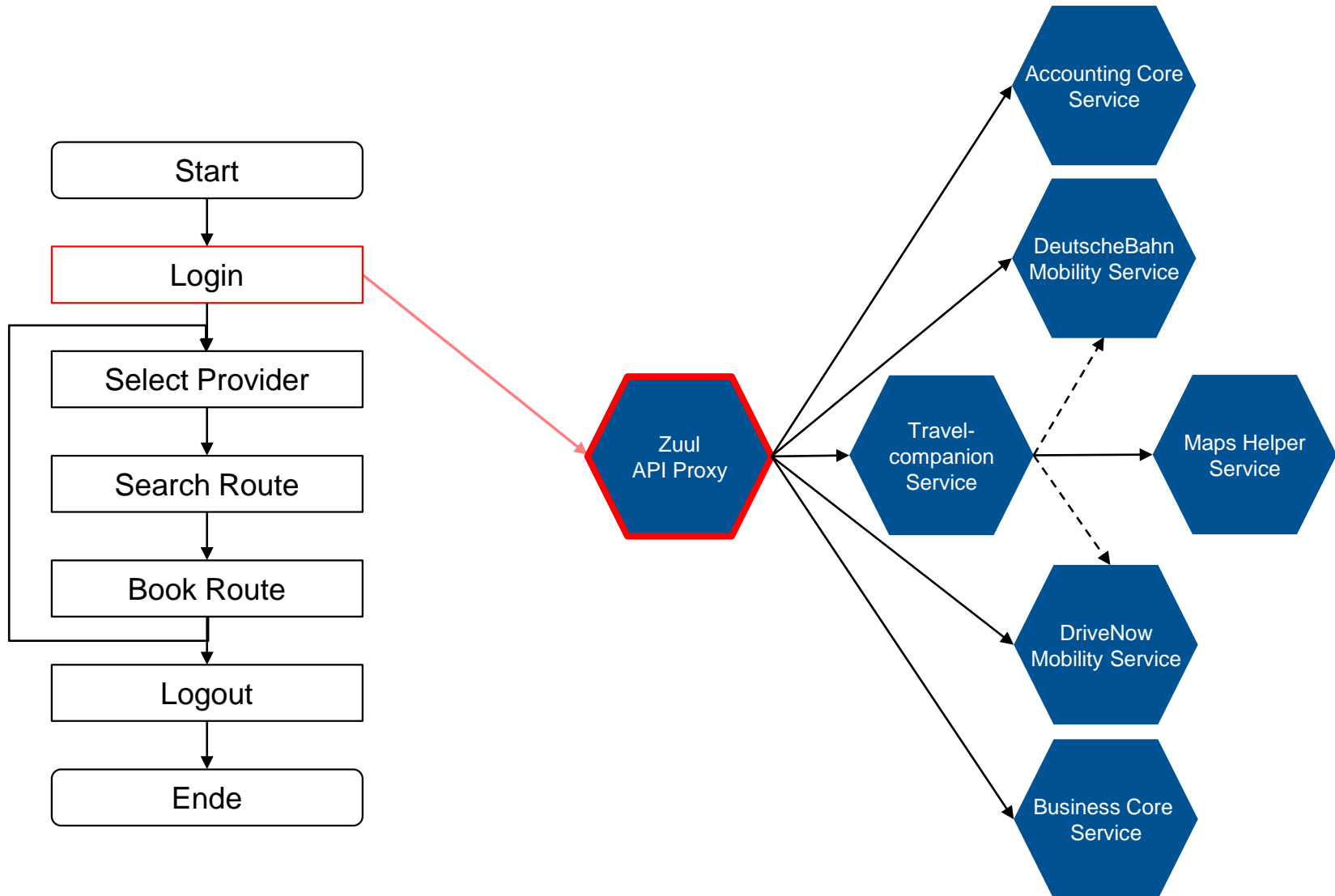
Live Demonstration

Scenario: Microservice Architecture



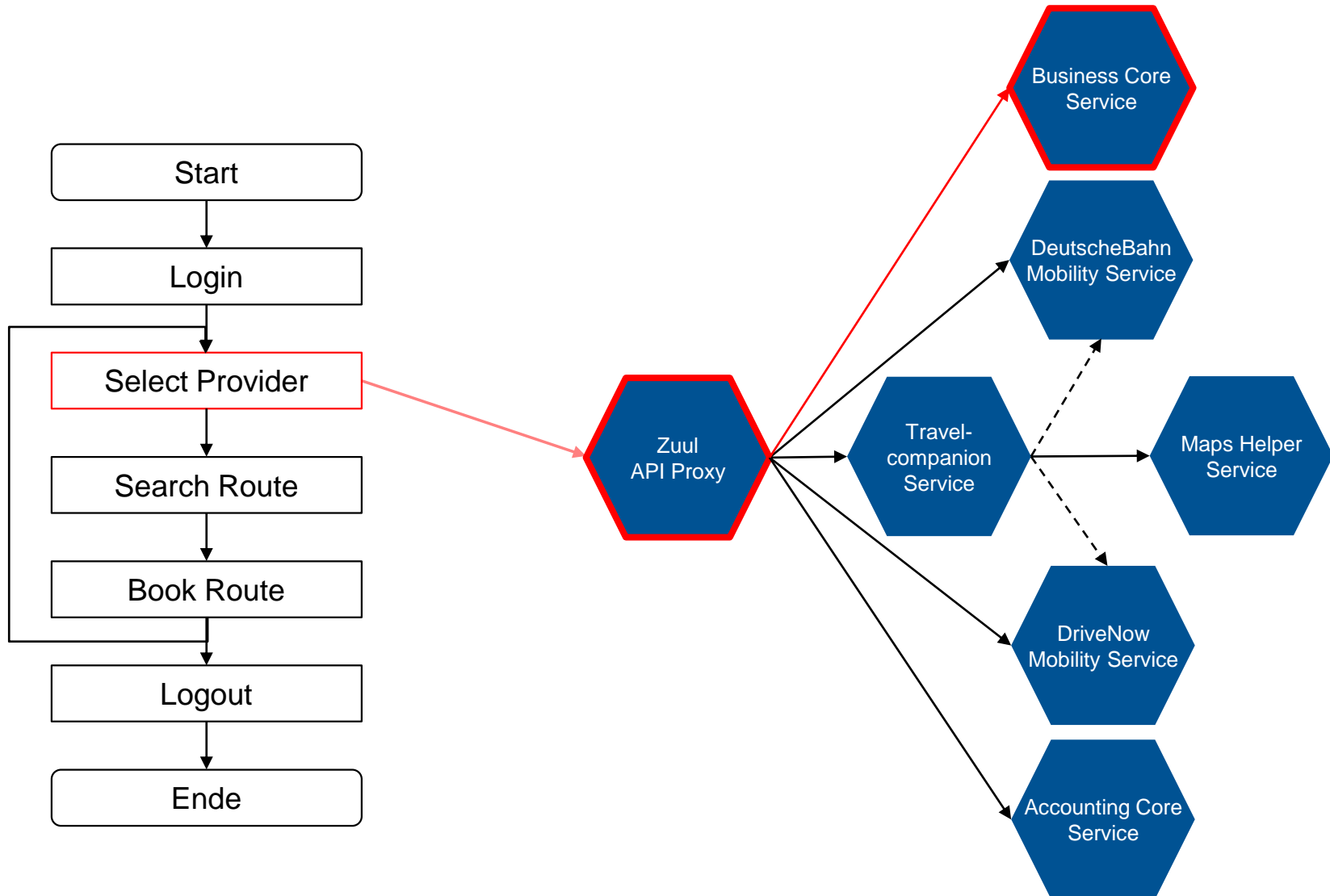
Live Demonstration

Scenario: Microservice Architecture



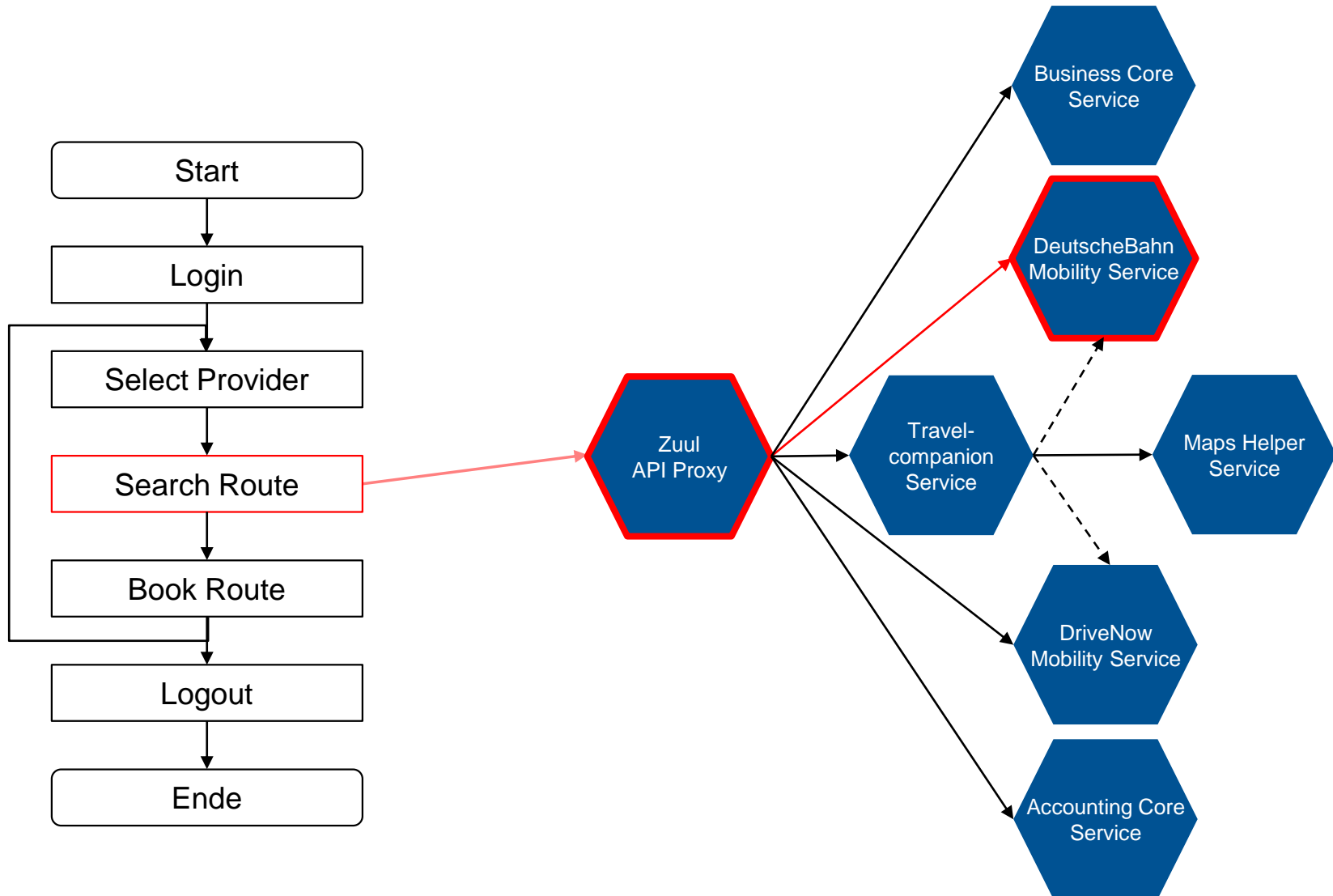
Live Demonstration

Scenario: Microservice Architecture



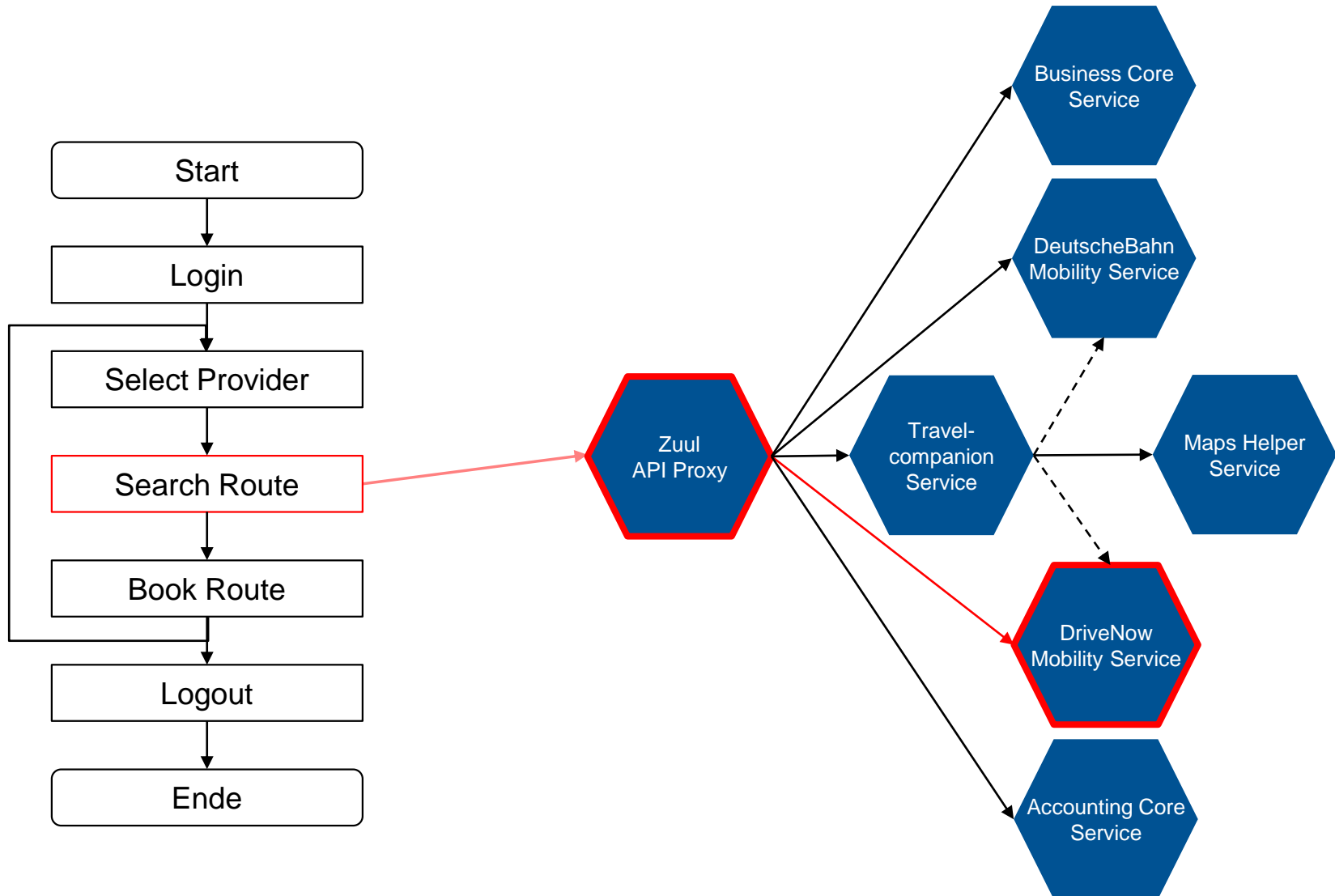
Live Demonstration

Scenario: Microservice Architecture



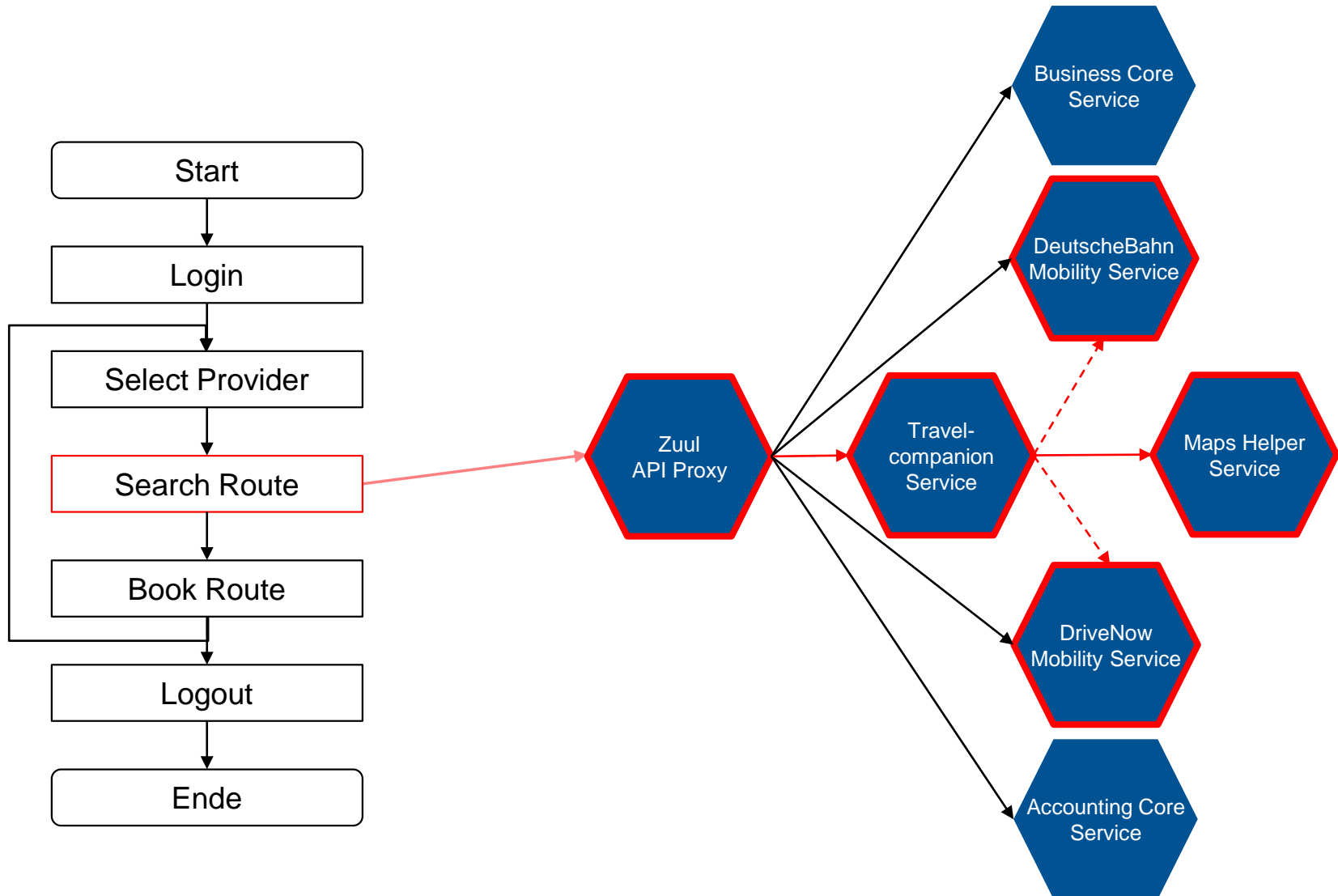
Live Demonstration

Scenario: Microservice Architecture



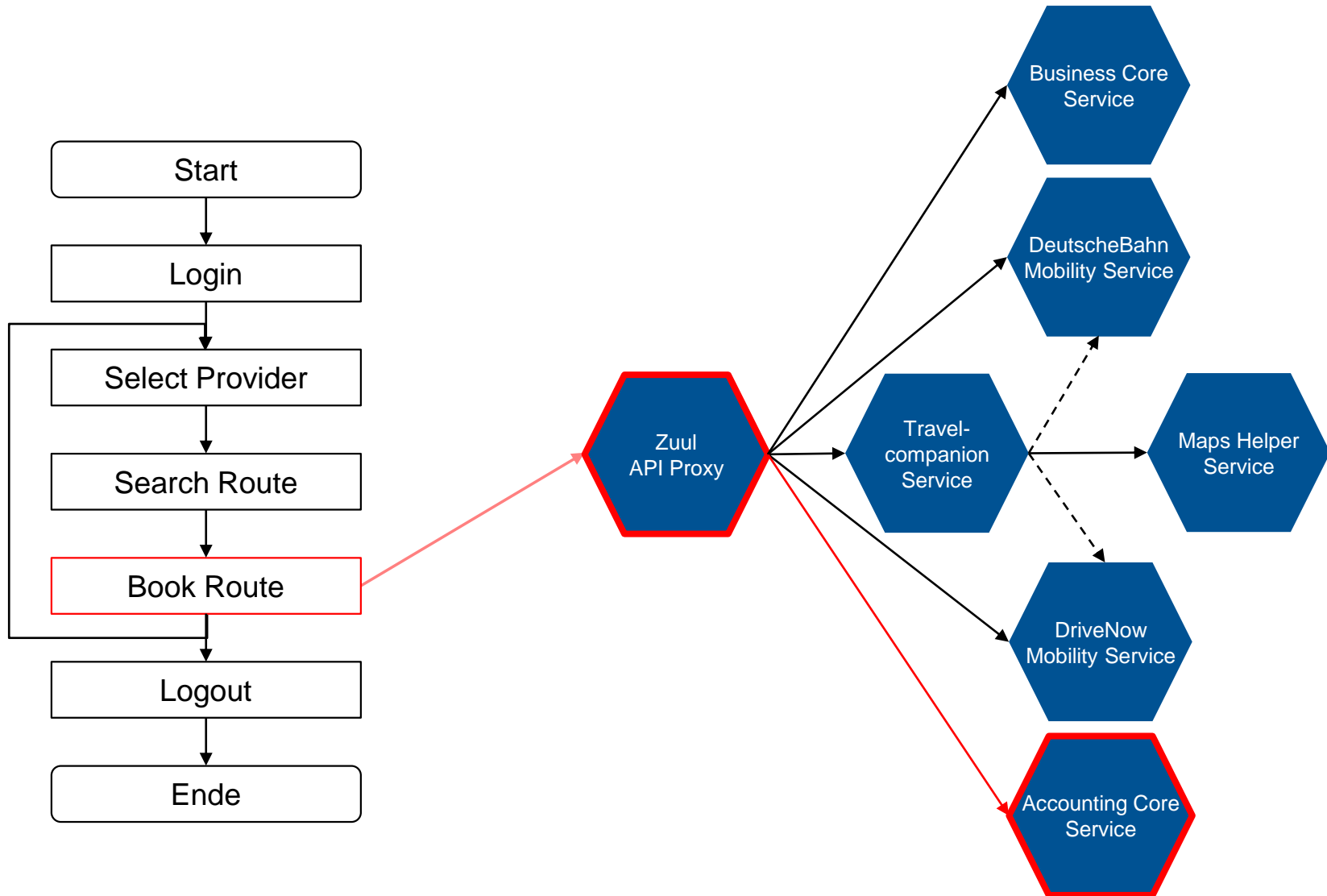
Live Demonstration

Scenario: Microservice Architecture



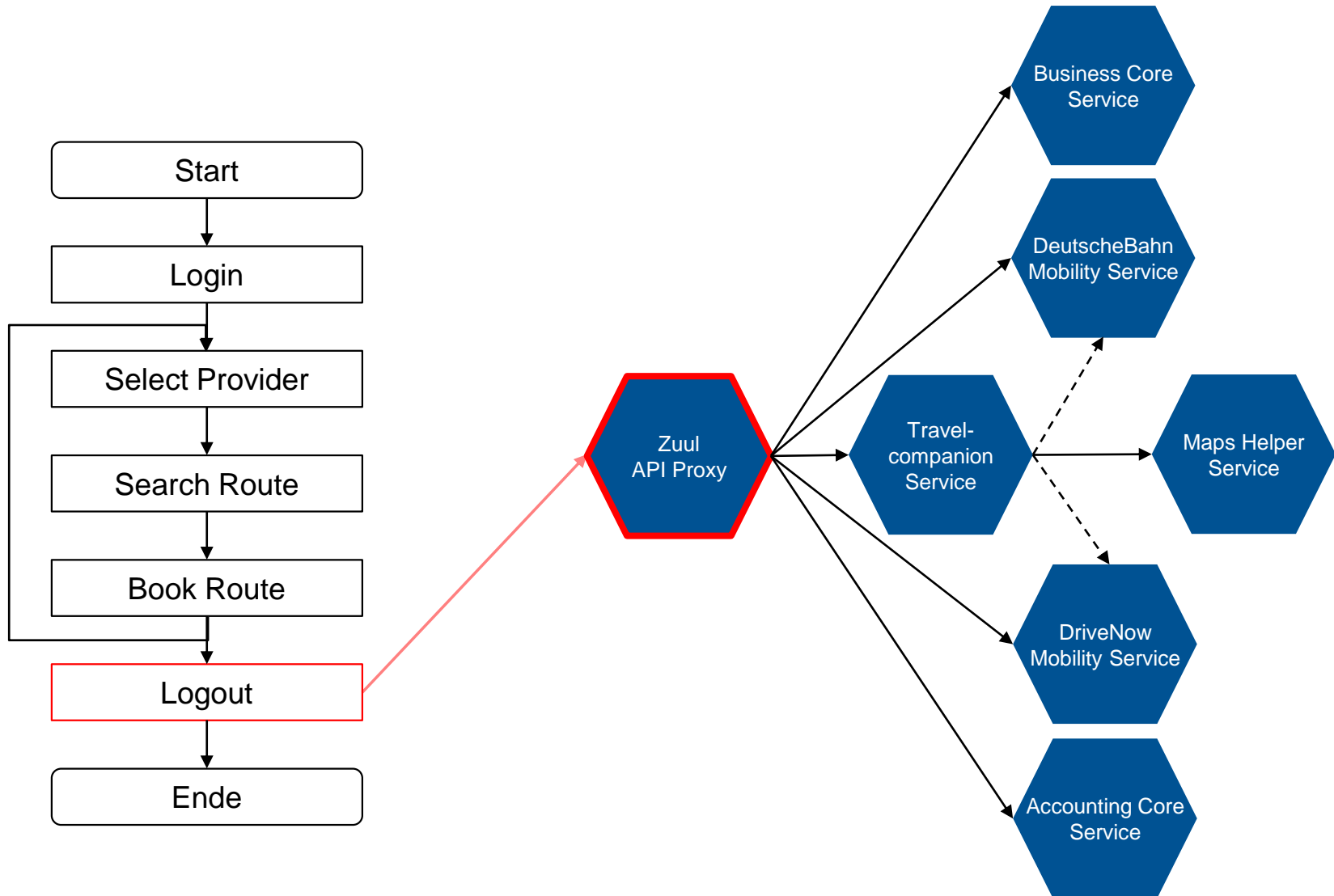
Live Demonstration

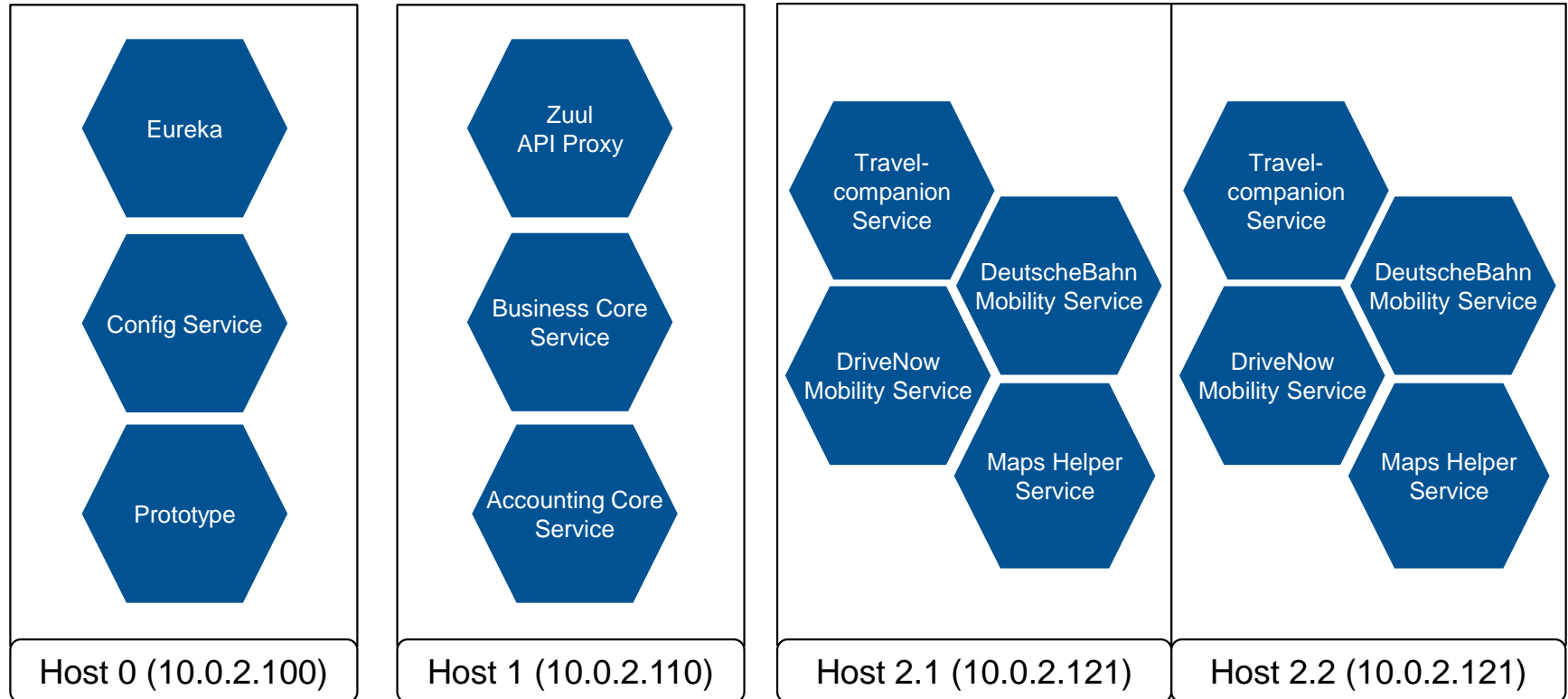
Scenario: Microservice Architecture



Live Demonstration

Scenario: Microservice Architecture





Live Demonstration

- Requires Zipkin-compatible instrumentation as well as Eureka
- Amount of meta-information depends on instrumentation
- Raw communications like Database-Access is not detected
- Virtual Hosts are not detected
- Business layer is modelled

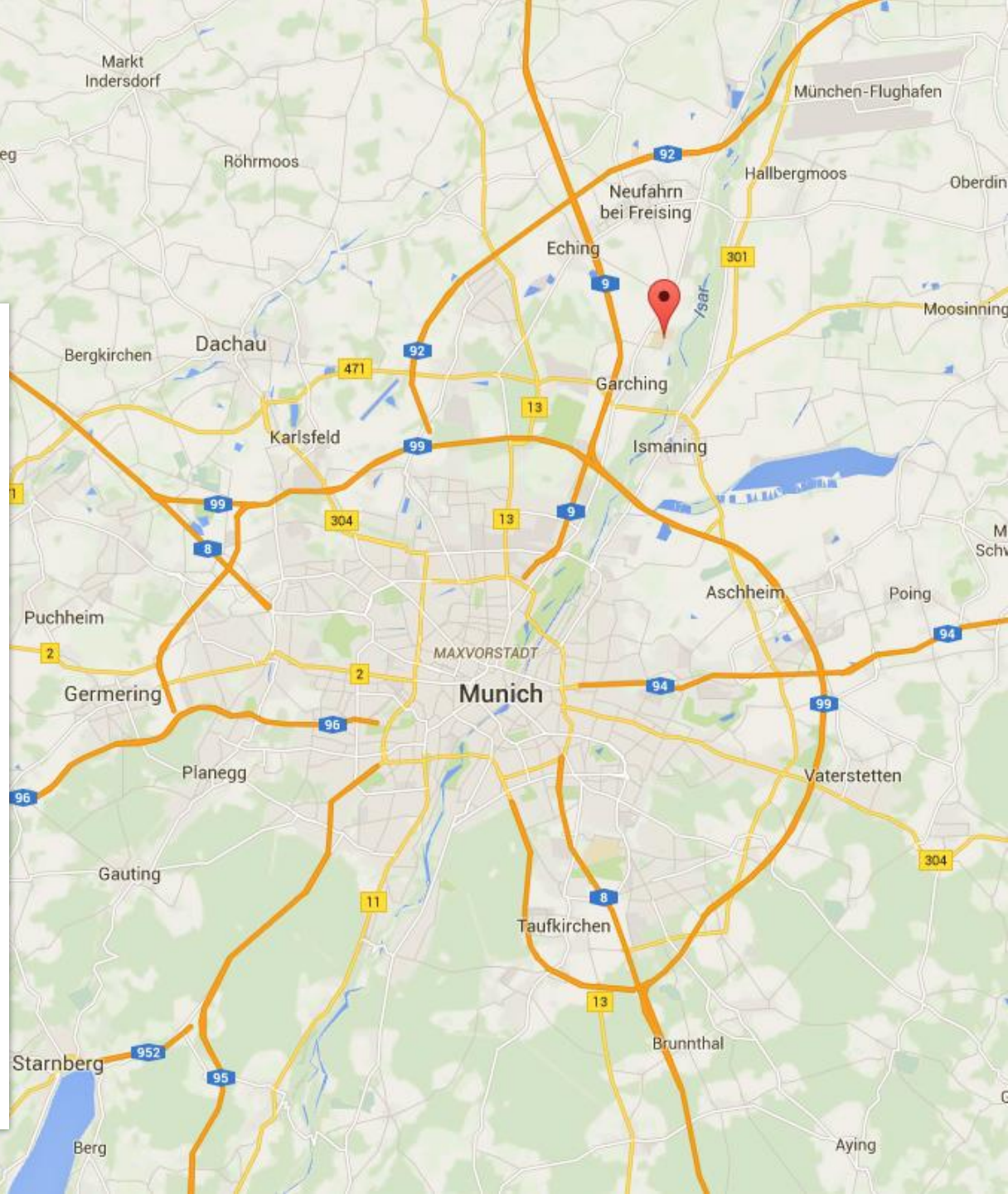


B.Sc.
Patrick Schäfer

Technische Universität München
Faculty of Informatics
Chair of Software Engineering for
Business Information Systems

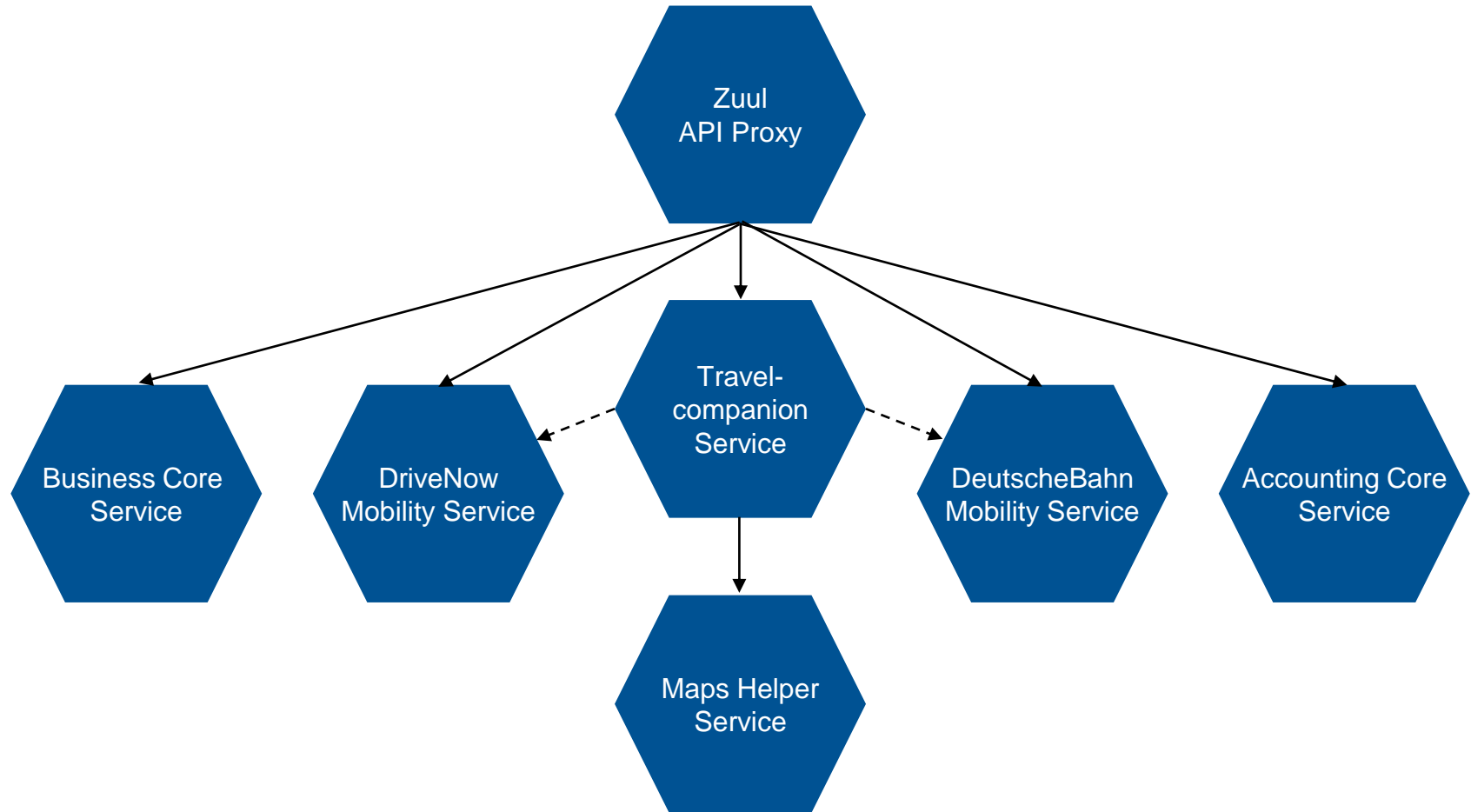
Boltzmannstraße 3
85748 Garching bei München

patrick.schaefer@tum.de
www.matthes.in.tum.de



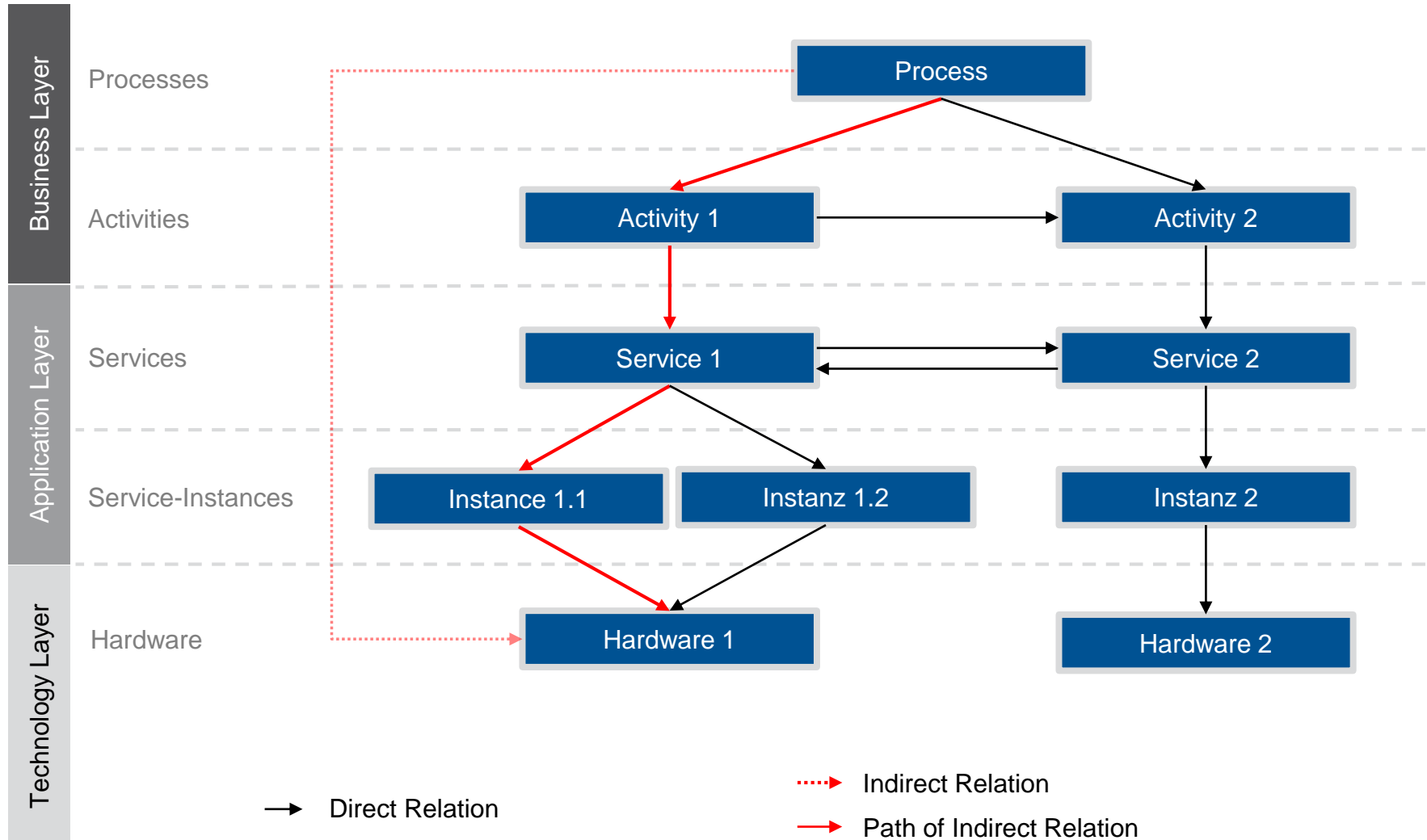
Live Demonstration

Scenario: Microservice Architecture



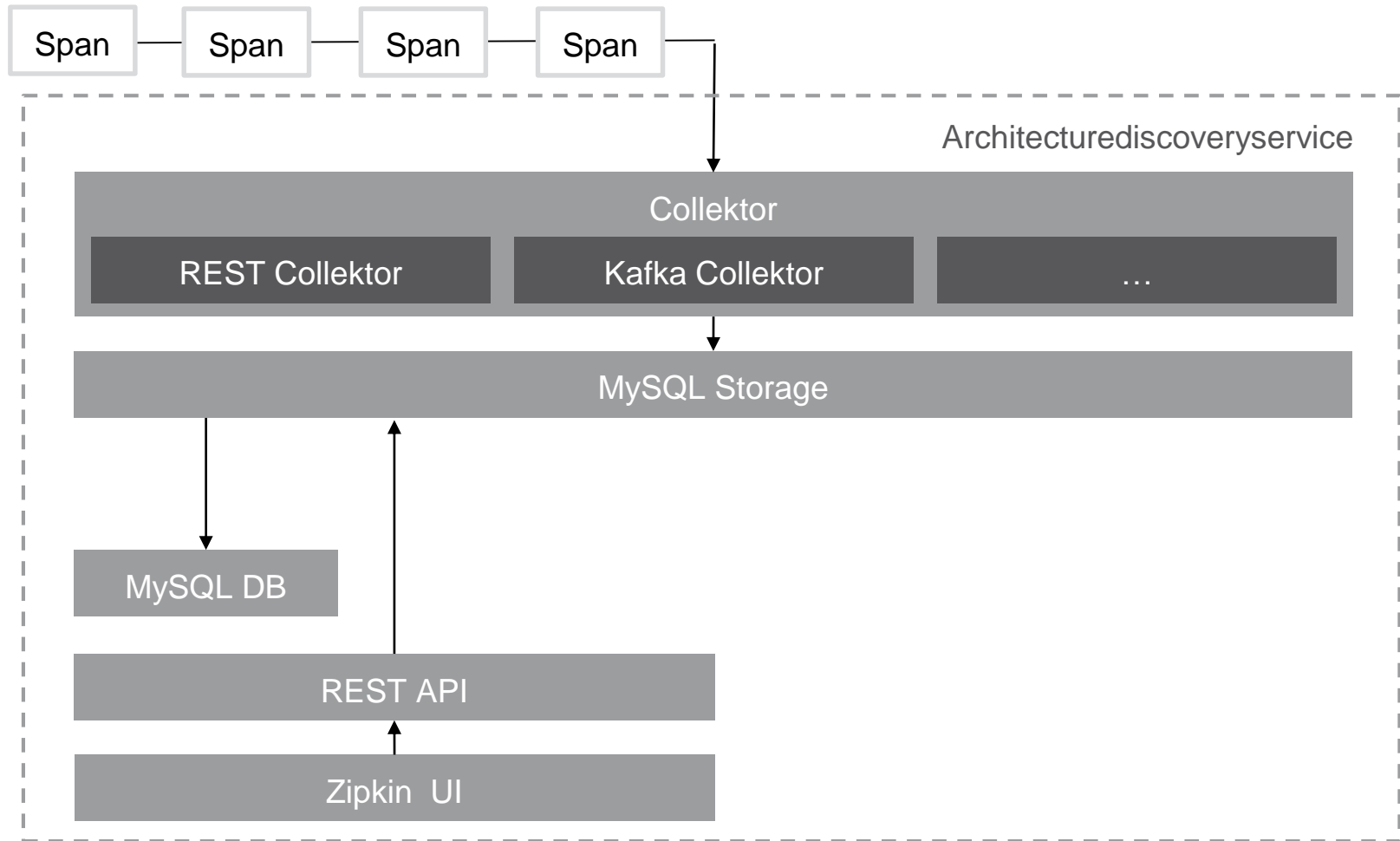
- ← Synchronous communication
- ←-- Asynchronous communication

Relation classification: Direct and Indirect Relations



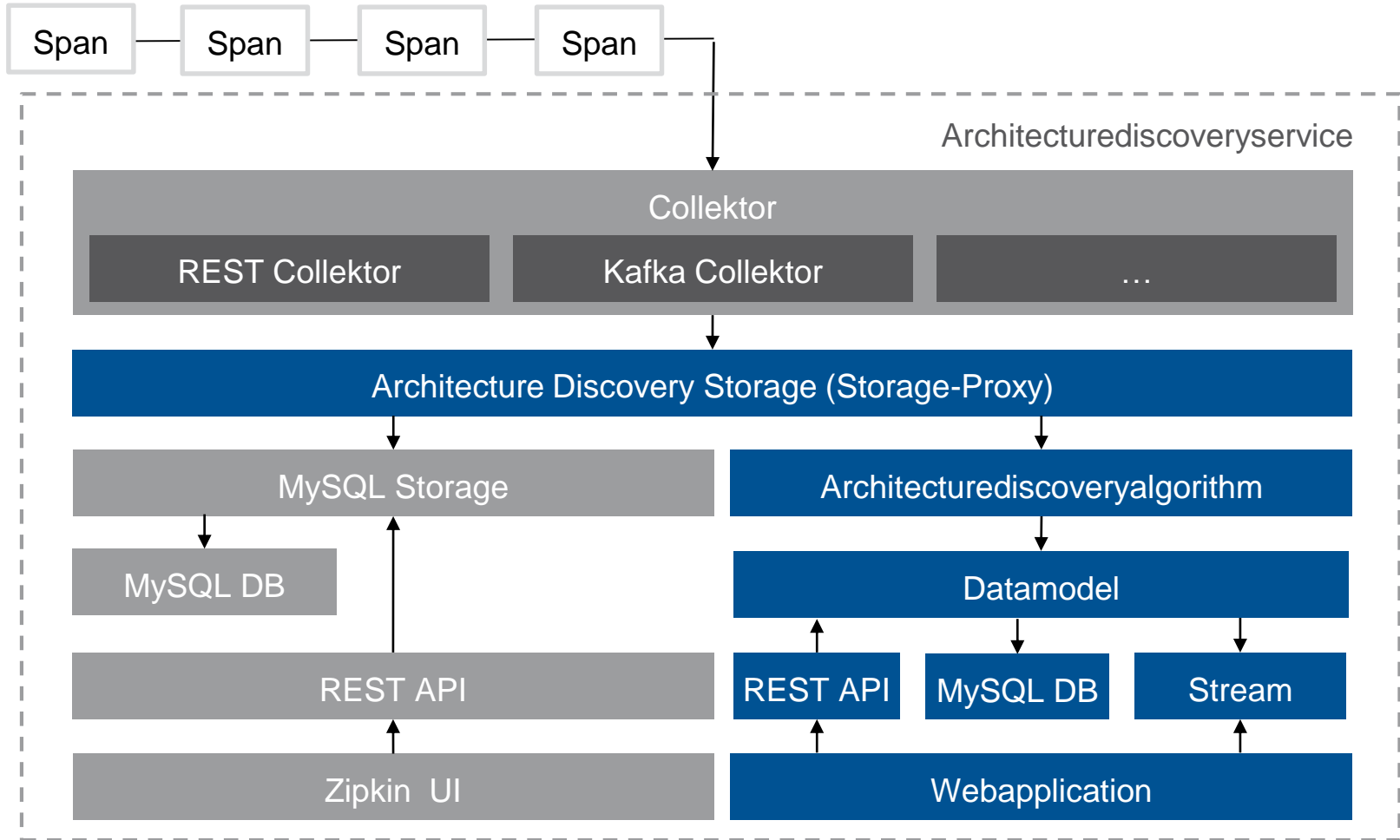
Prototypical Implementation

Implementation: Zipkin Server Extension



Prototypical Implementation

Implementation: Zipkin Server Extension



Architecture Adjacency Matrix

Time of snapshot: **LIVE**

Successors ->


	A4	A5	S1	S2	S3	S4	S5	S6	S7	S8	I1	I2	I3	I4	I5	I6	I7
A1		x															
A2		x								x							
A3																	
A4					x	x		x	x	x			x	x	x	x	
A5		-															
S1			-								x						
S2				-								x					
S3					-								x	x			
S4						-									x	x	
S5							-										x
S6								-									
S7					x	x		x	-				x		x		
S8			x	x	x	x		x	x	-	x	x	x	x	x	x	
I1											-						
I2												-					

< - Predecessors

Architecture Time

Architecture from

NOW

 Fetch architecture

Matrix Filters

Layers

Process ON

Activity ON

Service ON

Instance ON

Hardware ON

Relations

Indirect Relations ON

Process Activity Modelling

Process: Route Booking

1 [Create Activities](#)

Process

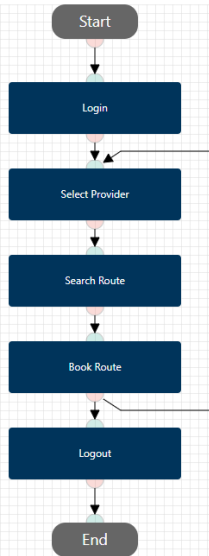
[Create Process](#)

Selected Process:

Route Booking

[Rename](#)

[Delete](#)



Unmapped paths

http:/

GET first seen: 2017-11-08 00:56:38

http:/login

POST first seen: 2017-11-08 00:56:47

http:/business-core-service/businesses/list

GET first seen: 2017-11-08 11:46:48

http:/logout

GET first seen: 2017-11-08 11:47:40

Rule definition

GET
 POST
 PUT
 DELETE

Regex: book\$

Parsed: book\$

Matching paths:

http:/accounting-core-service/deutsche-bahn/5/book

GET first seen: 2017-11-08 11:47:00

http:/accounting-core-service/drive-now/3/book

GET first seen: 2017-11-08 11:47:21

Assign paths to activity

Selected Process:

Route Booking

Selected Activity:

Book Route

Discard

Save

	P1	A1	A2	A3	A4	A5	S1	S2	S3	DEUTSCHEBAHN-MOBILITY-SERVICE					I4	I5	I6	I7	I8	I9	I10	I11	I12	H1	H2	H3	H4
Route Booking	P1	-	x	x	x	x	x		x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
Book Route	A1	-		x		x																					
Login	A2		-			x																	x		x		
Logout	A3			-																							
Search Route	A4		x		-				x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
Select Provider	A5				x	-																					
ACCOUNTING-CORE-SERVICE	S1						-																		x		
BUSINESS-CORE-SERVICE	S2							-																	x		
DEUTSCHEBAHN-MOBILITY-SERVICE	S3								-																	x	x
DRIVENOW-MOBILITY-SERVICE	S4																									x	x
EUREKA-SERVICE	S5																							x			
MAPS-HELPER-SERVICE	S6																									x	x
TRAVELCOMPANION-MOBILITY-SERVICE	S7																										
ZUUL-SERVICE	S8						x	x	x														x		x	x	x
ACCOUNTING-CORE-SERVICE (10.0.2.110:5000)	I1																										
BUSINESS-CORE-SERVICE (10.0.2.110:5000)	I2																										
DEUTSCHEBAHN-MOBILITY-SERVICE (10.0.2.121:6003)	I3																									x	
DEUTSCHEBAHN-MOBILITY-SERVICE (10.0.2.122:6003)	I4																										x
DRIVENOW-MOBILITY-SERVICE (10.0.2.121:6003)	I5																									x	
DRIVENOW-MOBILITY-SERVICE (10.0.2.122:6003)	I6																										x
EUREKA-SERVICE (10.0.2.100:8761)	I7																							x			
MAPS-HELPER-SERVICE (10.0.2.121:7000)	I8																									x	
MAPS-HELPER-SERVICE (10.0.2.122:7000)	I9																										x
TRAVELCOMPANION-MOBILITY-SERVICE (10.0.2.110:6003)	I10																									x	
TRAVELCOMPANION-MOBILITY-SERVICE (10.0.2.121:6003)	I11																										x
ZUUL-SERVICE (10.0.2.110:9000)	I12																								x		
10.0.2.100	H1																										
10.0.2.110	H2																										
10.0.2.121	H3																										
10.0.2.122	H4																										

Relation Details

Service **TRAVELCOMPANION-MOBILITY-SERVICE**
uses
Service **DEUTSCHEBAHN-MOBILITY-SERVICE**

Annotations

mvc.controller.class DeutscheBahnController

mvc.controller.method findDistance

spring.instance_id 10.0.2.121:deutschebahn-mobility-service:6003

http.path /getroutes

http.url http://localhost:6003/getroutes?origin=1&destination=3

First seen 2017-11-08 12:31:28

Asynchronous true

http.method GET

http.host localhost