Enhancing Business Process Mining with Distributed Tracing Data in a Microservice Architecture

Jochen Graeff (B.Sc.) | 21.08.2017 | Master thesis final presentation
Advisor: Martin Kleehaus

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

Motivation
Research questions

Approach
- Build sample architecture
- Instrument sample architecture
- Develop activity generation algorithm
- Set-up extended architecture
- Analysis creation

Live Demo

Evaluation
- Benefits
- Limitations
Motivation

Amazon found every 100ms of latency cost them 1% in sales!¹

Walmart saw up to a 2% increase in conversions for every 1 second of improvement in load time. Every 100ms improvement also resulted in up to a 1% increase in revenue.²

How does user behaviour and system behaviour influence each other?
Is there a gap between the layers?

- **Business Layer**
  - **Application Layer**
  - **Run-time behavior**
  - **Transparency**

- **Process discovery of multiple sessions**
  - **Analysis of single trace**

- **Distributed Tracing**

- **Search cars** -> **Search cars**
- **End rental** -> **Reserve car**
- **Book car**
- **Show Details**
- **Report Issue**
- **Show details**
- **Book car**
- **End rental**

- **Business transactions**

- **Span ID 1**
- **Span ID 2**
- **Span ID 3**
Research questions

RQ1. How can a relationship between business activities and a distributed application architecture be established?

RQ2. What data has to be extracted and how has it to be mapped to enable and store the relationship knowledge?

RQ3. How can business process mining be extended with technical aspects in order to uncover

   a) user and system throughput times for business activity executions and,
   b) correlations between business process performance and system behaviour?
Build sample architecture

- Car sharing platform
- Spring Cloud microservice architecture
- 3 x infrastructure services
- 6 x business services
Build sample architecture
Instrument sample architecture

1. Build sample architecture
   - Car sharing platform
   - Spring Cloud microservice architecture
   - 3 x infrastructure services
   - 6 x business services

2. Instrument sample architecture
   - Add Spring Cloud Sleuth to every business service
   - Append sessionID to span in controller of webui service
Instrument sample architecture

- Instrument every business service with spring cloud sleuth in order to generate span data in the applications
  - Add dependency spring-cloud-starter-zipkin
  - Set sampling rate

- Instrument service (as for business services)
- Append sessionID in every webui service endpoint
definition: tracer.addTag("sessionID", sessionID);
Develop activity generation algorithm

1. Build sample architecture
   - Car sharing platform
   - Spring Cloud microservice architecture
   - 3 x infrastructure services
   - 6 x business services

2. Instrument sample architecture
   - Spring Cloud Sleuth
   - Append sessionID to span in controller of webui service

3. Develop activity generation algorithm
   - Definition of required attributes and foreign for event log
   - Definition of healthy/failed activities
   - User and system activities
   - Inputs: spans, annotation & mapping table
### Activity generation

**spans table (zipkin)**

<table>
<thead>
<tr>
<th>TRAC_ID</th>
<th>SPAN_ID</th>
<th>PARENT_ID</th>
<th>NAME</th>
<th>TIMESTAMP</th>
<th>DURATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>a</td>
<td>null</td>
<td>http/bookcar</td>
<td>07/06/17 12:45:32.000</td>
<td>800 ms</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
<td>a</td>
<td>initialize</td>
<td>07/06/17 12:45:32.100</td>
<td>600 ms</td>
</tr>
<tr>
<td>a</td>
<td>c</td>
<td>b</td>
<td>http/initializebooking</td>
<td>07/06/17 12:45:32.200</td>
<td>500 ms</td>
</tr>
<tr>
<td>a</td>
<td>d</td>
<td>c</td>
<td>http/handlecarbooking</td>
<td>07/06/17 12:45:32.400</td>
<td>100 ms</td>
</tr>
<tr>
<td>b</td>
<td>b</td>
<td>null</td>
<td>http/opencar</td>
<td>07/06/17 12:37:32.400</td>
<td>3000 ms</td>
</tr>
</tbody>
</table>

**annotations table (zipkin)**

<table>
<thead>
<tr>
<th>TRAC_ID</th>
<th>SPAN_ID</th>
<th>KEY</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>a</td>
<td>http.method</td>
<td>GET</td>
</tr>
<tr>
<td>a</td>
<td>a</td>
<td>cs</td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>a</td>
<td>cr</td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>a</td>
<td>sessionID</td>
<td>12345</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
<td>ss</td>
<td></td>
</tr>
</tbody>
</table>

**technical_activity**

- http/bookcar: Book car
- http/initializebooking: http/initializebooking
- http/handlecarbooking: http/handlecarbooking
- http/opencar: Unlock car
- http/findroute: Find route

**calls_service**

- webui-service
- accounting-service
- cars-service
- webui-service
- webui-service

**activities table**

<table>
<thead>
<tr>
<th>CASE ID</th>
<th>ACTIVITY</th>
<th>START_TS</th>
<th>END_TS</th>
<th>DURATION</th>
<th>TYPE</th>
<th>SERVICE_NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>12345</td>
<td>Book car</td>
<td>07/06/17 12:45:32.000</td>
<td>07/06/17 12:45:32.800</td>
<td>800 ms</td>
<td>user</td>
<td>webui-service</td>
</tr>
<tr>
<td>12345</td>
<td>http/initializebooking</td>
<td>07/06/17 12:45:32.200</td>
<td>07/06/17 12:45:32.700</td>
<td>500 ms</td>
<td>system</td>
<td>accounting-service</td>
</tr>
<tr>
<td>12345</td>
<td>http/handlecarbooking</td>
<td>07/06/17 12:45:32.400</td>
<td>07/06/17 12:45:32.500</td>
<td>100 ms</td>
<td>system</td>
<td>cars-service</td>
</tr>
<tr>
<td>12345</td>
<td>Unlock car</td>
<td>07/06/17 12:37:32.400</td>
<td>07/06/17 12:37:35.400</td>
<td>3000 ms</td>
<td>user</td>
<td>webui-service</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Setup extended architecture

1. Build sample architecture
   - Car sharing platform
   - Spring Cloud microservice architecture
   - 3 x infrastructure services
   - 6 x business services

2. Instrument sample architecture
   - Spring Cloud Sleuth
   - Append sessionID to span in controller of webui service

3. Develop activity generation algorithm
   - Definition of required attributes and foreign for event log
   - Definition of healthy/failed activities
   - User and system activities
   - Inputs: spans, annotation & mapping table

4. Setup extended architecture
   - Deploy MySQL
   - Deploy Zipkin
   - Deploy Celonis
   - Build log generation service that executes the algorithm
Setup extended architecture
Analysis creation

1. Build sample architecture
   - Car sharing platform
   - Spring Cloud microservice architecture
   - 3 x infrastructure services
   - 6 x business services

2. Instrument sample architecture
   - Spring Cloud Sleuth
   - append sessionID to span in controller of webui service

3. Develop activity generation algorithm
   - definition of required attributes and foreign for event log
   - Definition of healthy/failed activities
   - user and system activities
   - inputs: spans, annotation & mapping table

4. Setup extended architecture
   - Deploy MySQL
   - Deploy Zipkin
   - Deploy Celonis
   - Build log generation service that executes the algorithm

5. Analysis Creation
   - Configure data model
   - Configure continuous data load
   - Create 4 analysis with
     - Business
     - Application
     - Cross-Domain
     - Single Activity views on the process

Jochen Graeff – Master thesis final presentation
Cross-Domain Analysis

Conversion Rate 41.38%  User Activity Failure Rate 0.95%  AVG clicks to conversion 6  Session Duration w/ c 140 s  Session Duration w/o c 2916 s

User Click Path

Conversion vs. Activity Failure

Duration: Session S...

Most critical services

User Process Activity Performance

<table>
<thead>
<tr>
<th>Activity</th>
<th>Activities count</th>
<th>Duration</th>
<th>Act. Conversion Impact</th>
</tr>
</thead>
<tbody>
<tr>
<td>Book car</td>
<td>13</td>
<td>1593 ms</td>
<td>100.00%</td>
</tr>
<tr>
<td>Find route</td>
<td>6</td>
<td>651 ms</td>
<td>100.00%</td>
</tr>
<tr>
<td>End car rental failed</td>
<td>1</td>
<td>3512 ms</td>
<td>100.00%</td>
</tr>
<tr>
<td>End car rental</td>
<td>13</td>
<td>5340 ms</td>
<td>92.31%</td>
</tr>
<tr>
<td>Unlock Car</td>
<td>15</td>
<td>345 ms</td>
<td>86.67%</td>
</tr>
<tr>
<td>List available cars</td>
<td>18</td>
<td>1757 ms</td>
<td>61.11%</td>
</tr>
<tr>
<td>Reserve car</td>
<td>16</td>
<td>1014 ms</td>
<td>50.00%</td>
</tr>
<tr>
<td>Show balance</td>
<td>4</td>
<td>122 ms</td>
<td>50.00%</td>
</tr>
</tbody>
</table>
Live Demo

Carsharing Mockup App

Enhancing Business Process Mining with Distributed Tracing Data in a Microservice Architecture

This frontend is intended for log data creation. A user (1 - 10) receives a session ID for each session and can perform multiple activities. These activities trigger a variety of microservices in the background. When a new user is created, he or she gets a new session ID assigned.

The data model can be initialized by clicking on Initialize datamodel. The activity generation can be triggered manually through the “Generate activities & reload data model” button. After a new activity log is created, a new entry is written in the reload_trigger_table that triggers a new data load in the process mining tool.

- New User: User 1
- New Session: q6qjf01hpuv7c1cdse5as6ncss

List available cars  Reserve car  Book car  Unlock car  End car rental
Show my balance  Book package  Show driving history  Report issue  Find route

sessionID = q6qjf01hpuv7c1cdse5as6ncss || WebUI --> Zuul Service --> Cars Service

Generate activities & reload data model
## Benefits

<table>
<thead>
<tr>
<th>Cross-domain analysis</th>
<th>Resource-efficient data source</th>
<th>Portability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Provides a more holistic view between the business and application layer</td>
<td>Resource-efficient (easy to implement) input source for process mining in microservice architectures</td>
<td>Approach transferable to different architectures with limited effort (i.a. due to OpenTracing standard)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Ubiquity</th>
<th>Flexibility on process perspectives</th>
<th>Bottom up process discovery</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ubiquitous through distributed tracing becoming a standard tool for microservice debugging</td>
<td>Process scope flexibility through appending <em>spans</em> with arbitrary IDs</td>
<td>Bottom up process discovery in legacy systems</td>
</tr>
</tbody>
</table>
Limitations

System under survey (SUS)
- Approach only tested on SUS
  - Single architecture
  - Data volume
  - Data contents

Process visualisation of system activities
- Petri nets not a suitable visualisation method for system activities

Performance overhead
- Necessity for SamplingRate=1.0 for capturing whole process instances leads to performance overhead

Real-time event handling
- Presented prototype only generates event log and reloads data model every 5 min
Backup

- Workflow of activity generation and data reloading
- Related work
- Process Mining
- Inputs for activity generation
- Distributed tracing
- User request and span/trace context
- 'End car rental' user activity sequence diagram
Workflow of activity generation and data reloading

<table>
<thead>
<tr>
<th>User</th>
<th>Log generation service</th>
<th>MySQL</th>
<th>PMT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Perform clicks in front-end</td>
<td>Trigger scheduled activity generation</td>
<td>Write activities table</td>
</tr>
<tr>
<td></td>
<td>Trigger manual activity generation</td>
<td>Execute activity generation algorithm</td>
<td>Write technical_activities table</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Write trace_span table</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Insert new entry into reload_trigger table</td>
</tr>
</tbody>
</table>

- User
- Log generation service
- MySQL
- PMT

- ~5 min

Jochen Graeff – Master thesis final presentation
Table 5.2: Classification of related work

<table>
<thead>
<tr>
<th>Study</th>
<th>Log origin</th>
<th>Activity types</th>
<th>Captured behaviour</th>
<th>Type of work</th>
<th>Evaluation environment</th>
<th>System architecture</th>
<th>Language independence</th>
<th>(Near) real-time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Poggi et al. [53]</td>
<td>Web logs</td>
<td>User Activities</td>
<td>Business</td>
<td>Algorithm evaluation</td>
<td>Real-life event logs</td>
<td>No</td>
<td>n/a</td>
<td>No</td>
</tr>
<tr>
<td>Abe &amp; Kudo [7]</td>
<td>Web logs</td>
<td>User activities</td>
<td>Business</td>
<td>Framework</td>
<td>Real-life event logs</td>
<td>n/a</td>
<td>n/a</td>
<td>No</td>
</tr>
<tr>
<td>Bruckmann et al. [13]</td>
<td>n/a</td>
<td>User Activities, system activities</td>
<td>Business and system</td>
<td>Architecture proposal</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>Yes</td>
</tr>
<tr>
<td>Leemans &amp; van der Aalst [41]</td>
<td>Joinpoint-pointcut model instrumentation</td>
<td>User activities</td>
<td>System</td>
<td>Instrumentation strategy, implementation</td>
<td>Real-life event logs</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Rubin et al. [59]</td>
<td>Custom instrumentation</td>
<td>User activities, system activities</td>
<td>User and system</td>
<td>Experimental</td>
<td>Real-life event logs</td>
<td>No</td>
<td>n/a</td>
<td>No</td>
</tr>
<tr>
<td>Proof-of-concept prototype of this work</td>
<td>Distributed tracing instrumentation</td>
<td>User activities, system activities</td>
<td>Business, user and system</td>
<td>Instrumentation strategy, architecture description, implementation</td>
<td>Simulated user requests on testing system</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
“The idea of process mining is to **discover, monitor and improve real processes** (i.e., not assumed processes) by extracting **knowledge from event logs** readily available in today's (information) systems.”

IEEE CIS Task Force on Process Mining

There are three Classes of Process Mining:

1. Process Discovery
2. Conformance Checking
3. Extension

---

**Event Log**

<table>
<thead>
<tr>
<th>TIMESTAMP</th>
<th>ACTIVITY</th>
<th>CASE ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>2016-03-05</td>
<td>SEARCH CARS</td>
<td>#1234</td>
</tr>
<tr>
<td>2016-03-05</td>
<td>REPORT ISSUE</td>
<td>#5678</td>
</tr>
<tr>
<td>2016-03-05</td>
<td>RESERVE CAR</td>
<td>#1234</td>
</tr>
<tr>
<td>2016-03-05</td>
<td>SHOW DETAILS</td>
<td>#9012</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

---

**Diagram**
Develop activity generation algorithm

spans table (zipkin)
- name
- span_id
- trace_id
- start_ts
- parent_id
- duration
  ...

1 has 1..*

annotations table (zipkin)
- span_id
- trace_id
- a_key
- a_value
- a_timestamp
  ...

mapping table
- technical_activity
- pretty_name
- type
- calls_service

activities table
- sessionID
- activity
- start_ts
- end_ts
- duration
- trace_id
- span_id
- service_name
- failure
- sorting
- type
  ...

1 has 1..*

1..* has 1
The path taken through a simple servicing system on behalf of user request X.

The causal and temporal relationship between four spans of a trace.

Sigelman et al. (2010). *Dapper, A Large Scale Distributed Systems Tracing Infrastructure*. Google Research
User request and span/trace context
'End car rental' user activity

GET /endRental
GET /lockCar
GET /finalizeBooking
GET /handlePayment
PUT /notifyUser

User
ZUUL SERVICE
CARS SERVICE
ACCOUNTING SERVICE
PAYMENT SERVICE
NOTIFICATION SERVICE

locks car
finalizes booking
handles payment
notifies user