A Microservice Reference Case Study for Design-Runtime Interaction in MDE*

Daniele Di Pompeo, Michele Tucci, Alessandro Celi, and Romina Eramo

Department of Information Engineering, Computer Science and Mathematics,
University of L'Aquila, Italy
{firstname.lastname}@univaq.it

Abstract. Model-Driven Engineering techniques may achieve major support to the software development of nowadays complex systems when they allow managing relationships between a running system and its design models. These relationships can be exploited for different goals, such as the software evolution due to new functional requirements. In order to address this challenge, researchers need to better understand the nature of the available runtime information and related correspondences as well as how leveraging such knowledge. Typically, to this end, they rely on reference applications.
In this paper, we present a reference case study for design-runtime interaction in MDE. It is based on Train Ticket, a microservice-based web application, and its monitoring infrastructure. Also, the case study provides its software modeling artifacts designed in UML, a dataset of monitoring logs, and the definition of design-runtime correspondence as traceability links. We invite researchers to consider this case study as a reference for extending or new contribution to this topic.

Keywords: MDE · Design-runtime interaction · Microservice architecture

1 Introduction

Software is the foundation of today's life, managing everything from mobile/web applications to airplanes in flight, but ensuring increasingly complex and error-free systems has become a challenging task. While practitioners are forced to use and investigate different development techniques to tackle advances in productivity and quality, software engineering has to rely on automated approaches to keep low the development costs while tackling the rapid changes of software capabilities that expose different (non-)functional properties.

In order to manage software complexity, ever more companies are considering Model-Driven Engineering (MDE) [15] approaches, with the perceived benefit of enabling developers to work at a higher level of abstraction and to rely on automation throughout the development process. Nevertheless, MDE solutions

* This research was supported by the ECSEL-JU through the MegaM@Rt2 project (grant agreement No 737494).

need to be further developed to scale up for real-life industrial projects [8]. To this intent, one of the major challenges is to work on achieving a more efficient integration between the design and runtime aspects of systems. For instance, through observation and instrumentation, logs and metrics can be collected and related to the original software design in order to comprehend, extrapolate and analyze the inner behavior of running software system [10].

In support of this, a recent European project[1] has been founded and supported by both industry and academic partners. As part of its continuous system engineering approach [3], the project notably aims at providing a runtime-design time feedback loop that could be deployed and used in different industrial domains. Such a feedback from runtime to architectural design level can certainly be exploited to let the developers have some sort of control and manipulation possibilities over elements they would not be able to access otherwise.

Methods and tools have been proposed for monitoring system execution and measuring several aspects of the running systems (eg., performance). However, many of them do not envisage a solid integration with architectural design models [8]. In order to address this challenge, researchers need to better understand the nature of the available runtime information and related correspondences as well as how leveraging such knowledge. Typically, to this end, they rely on reference applications.

In this paper, we present a reference case study for design-runtime interaction in MDE. It is based on Train Ticket, a microservice-based web application, and its monitoring infrastructure. Also, the case study provides its software modeling artifacts designed in the UML language [2], a dataset of monitoring logs, and the definition of design-runtime correspondence as traceability links. We invite researchers to consider this case study as a reference for extending or new contribution to this topic.

2 The Train Ticket application

Recently, microservice architectures have become more relevant in the software engineering community due to their production flexibility. Among the few complex approaches available within the software engineering community, Xiang et al. [17] presented a benchmark that involves the `Train Ticket` system. Train Ticket is a microservice web-based application composed by 40 microservices, each one with a specific aim, developed with different programming languages, such as Java and Go. The application is composed of several scenarios (e.g., users can search specific destinations or can book a train ticket) and supports different kind of users with different kind of permissions (e.g., a guest user, or an admin one).

Figure 1 shows the Train Ticket logical architecture. Each rectangle represents a microservice, whereas the topmost and the leftmost boxes depict infrastructural microservices, such as Service Discovery. In particular, `Gateway` is in charge of catching an HTTP request and sending the message to the right

[1] MegaM@Rt2 project: https://megamart2-ecsel.eu/

microservice, `Service Discovery` and `Registry` are in charge of managing microservices, whereas the `Load Balance` microservice is in charge of distributing the traffic by following pre-defined rules, e.g., round robin.

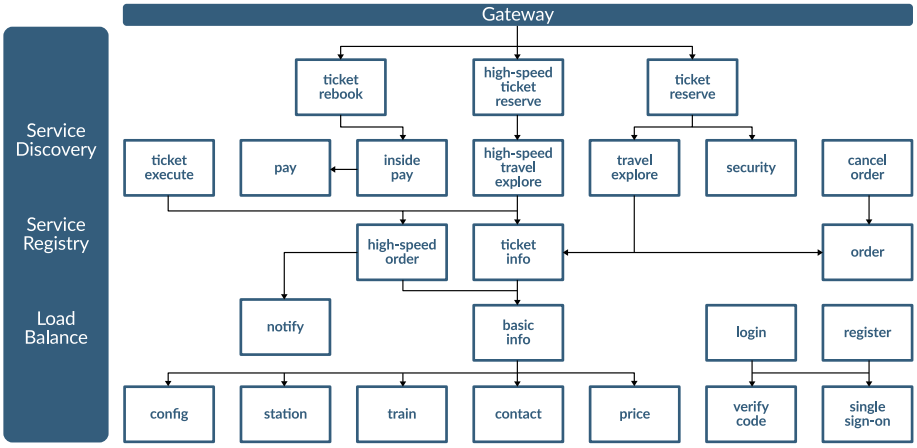


Fig. 1. Train Ticket architecture.

The Java-based part of the application were developed through the spring-cloud framework [2] that provides the needed features. Additionally, a distributed tracing system is used to collect real time information, for instance Xiang et al. used Spring Cloud Sleuth [3] distributed tracing solution, which is the official tracer within spring framework. In this work, we used Zipkin [4] and Elasticsearch [5] in our environment. Zipkin shows a graphical time-based representation of traces, whereas elastic search is a clustered database.

In the rest of the paper, we consider the Train Ticket design models and the trace/log artifacts to produce a reference case study for design-runtime interactions [6].

3 Train Ticket Design Modeling

The Train Ticket architecture and software design have been obtained by means of a both manual and automatic reverse engineering from the source code. It is modeled by means of UML; in particular, the multi-views UML modeling is composed of the *static views* (i.e., Component and Class diagrams) and the *dynamic views* (i.e., Sequence and State Machine Diagrams).

[2] Spring Cloud: https://spring.io/projects/spring-cloud
[3] Spring Cloud Sleuth: https://spring.io/projects/spring-cloud-sleuth
[4] Zipkin: https://zipkin.io/
[5] Elasticsearch: https://www.elastic.co/
[6] Please, refer to the following repository for the complete resources: https://github.com/SEALABQualityGroup/train-ticket

Altough in this case study the design models were reverse engineered from the source code, the same approach can be applied to models created at design-time, with the assumption that the implementation exactly follows the design.

In order to design the Train Ticket microservice architecture, we map each microservice into a UML Component. Figure 2 depicts a simplified version of the Component Diagram; in particular, we refer to the components that are involved in the `Rebook ticket` scenario (it allows to modify a ticket booking). As said, each UML Component describes a microservice, thus component's name maps one-to-one with the name of the microservice. Additionally, for each Component the operation discovered by the tracing analysis have been reported. We have also highlighted relations among microservices by means of UML Usage relations, i.e., the dashed arrows labelled with «use».

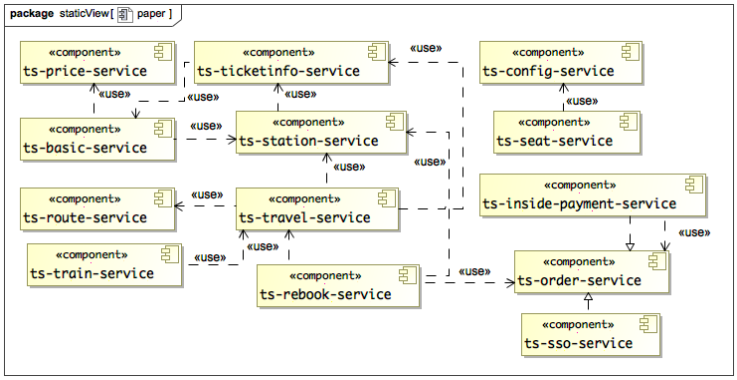


Fig. 2. Simplified Component Diagram of Train Ticket

A sample of the dynamic view of the application is depicted in Figures 3 and 4. In particular, Figure 3 shows a fragment of the Sequence Diagram of the Rebook ticket scenario, whereas Figure 4 shows a State Machine Diagram of the `Travel Service` microservice. In fact, each state machine represents the internal behavior of a component, thus the aforementioned state machine represents the internal behavior of the Travel Service microservice to accomplish the Rebook ticket request. Each state represents the invoked method, the arrows represent the event that have been triggered, while the `Stand by` state has been added to represent the microservice waiting state.

4 Runtime Information Mining

The monitoring infrastructure mentioned in Section 2 generates and collects runtime information in the form of *traces*, following the OpenTracing specification [1]. A trace consists of a series of casually related events that are triggered by a request as it moves through a distributed system. These events are called *spans* and they represent a timed operation occurring in a component. Spans contain references to other spans, which allow a trace to be assembled as a complete workflow.

According to the OpenTracing specification, a span always contains a set of basic information: the name of the operation, the name of the component providing the operation, the start timestamp and duration (or, alternatively, the finish timestamp), the role of the span in the request and a set of user-defined annotations called *tags*. In the case study we are considering, spans also contain additional information such as: the IP address and port number of the service, the Java class and method implementing the operation as well as the unique identifier of the Spring Cloud instance. Figure 5 reports an example of a raw log containing spans that are generated by the Train Ticket application and stored in the Elasticsearch database.

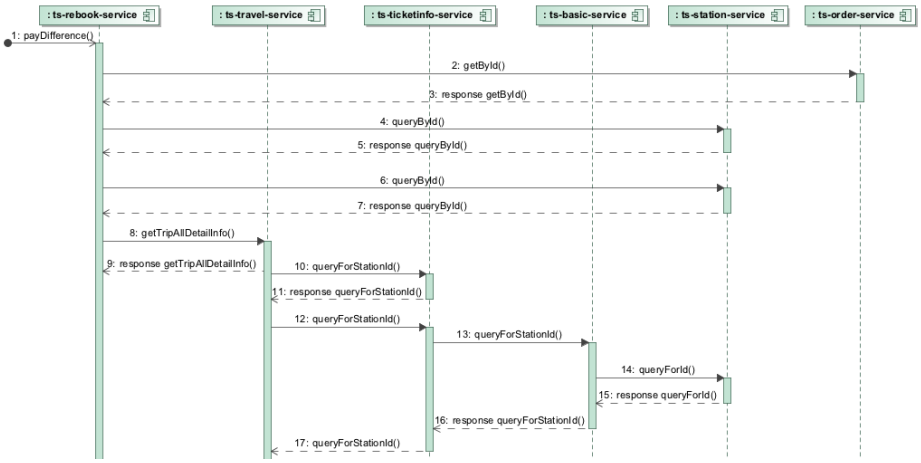


Fig. 3. A fragment of Sequence Diagram of the `Rebook Ticket` scenario

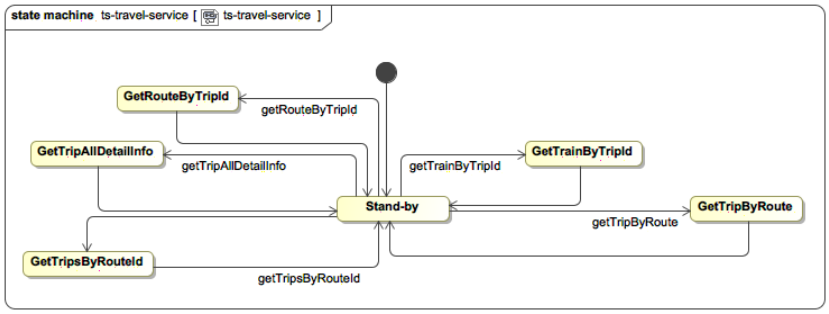
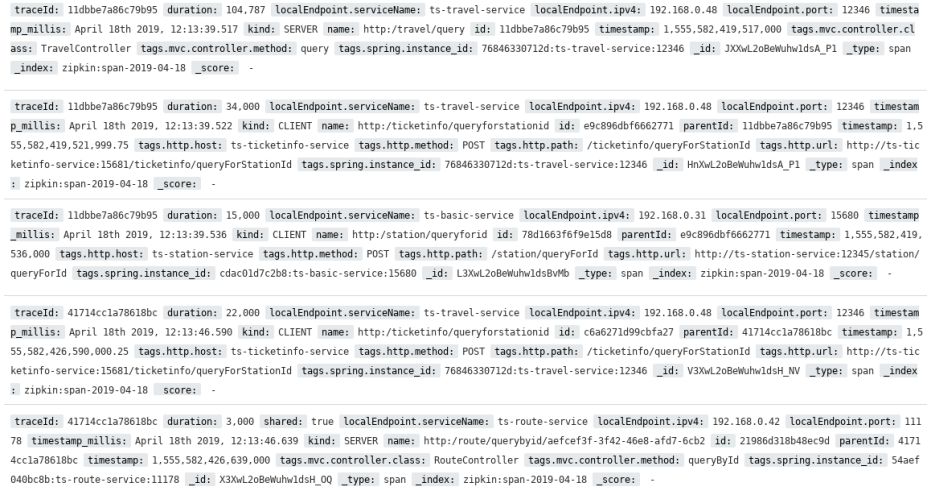


Fig. 4. State Machine Diagram of `Travel Service` microservice

4.1 Logs interpretation

Depending on the intended use, it may be necessary to analyze runtime information to compute complex metrics that the designer may want to link to design

models. For instance, when considering the duration of a span as the response time of an operation, it may be required to calculate the average response time in a time interval. Other examples of performance metrics may be the average response time of traces corresponding to a specific scenario or the number of times an operation is invoked in a time interval. In the field of dependability assessment, the designer may want to link errors and components in design models to establish their reliability and understand error propagation in the distributed system. To this extent, it is possible to include the HTTP status code of each request as a tag in the spans. Status codes corresponding to specific errors can be aggregated by error type or by time interval, obtaining, in this way, a metric that can be related to system components.


traceId: 11dbbe7a86c79b95 duration: 104,787 localEndpoint.serviceName: ts-travel-service localEndpoint.ipv4: 192.168.0.48 localEndpoint.port: 12346 timestamp_millis: April 18th 2019, 12:13:39.517 kind: SERVER name: http:/travel/query id: 11dbbe7a86c79b95 timestamp: 1,555,582,419,517,000 tags.mvc.controller.class: TravelController tags.mvc.controller.method: query tags.spring.instance_id: 76846330712d:ts-travel-service:12346 _id: JXXwL2oBeWuhw1dsA_P1 _type: span _index: zipkin:span-2019-04-18 _score: -

traceId: 11dbbe7a86c79b95 duration: 34,000 localEndpoint.serviceName: ts-travel-service localEndpoint.ipv4: 192.168.0.48 localEndpoint.port: 12346 timestamp_millis: April 18th 2019, 12:13:39.522 kind: CLIENT name: http:/ticketinfo/queryforstationid id: e9c896dbf6662771 parentId: 11dbbe7a86c79b95 timestamp: 1,555,582,419,521,999.75 tags.http.host: ts-ticketinfo-service tags.http.method: POST tags.http.path: /ticketinfo/queryForStationId tags.http.url: http://ts-ticketinfo-service:15681/ticketinfo/queryForStationId tags.spring.instance_id: 76846330712d:ts-travel-service:12346 _id: HnXwL2oBeWuhw1dsA_P1 _type: span _index: zipkin:span-2019-04-18 _score: -

traceId: 11dbbe7a86c79b95 duration: 15,000 localEndpoint.serviceName: ts-basic-service localEndpoint.ipv4: 192.168.0.31 localEndpoint.port: 15680 timestamp_millis: April 18th 2019, 12:13:39.536 kind: CLIENT name: http:/station/queryforid id: 78d1663f6f9e15d8 parentId: e9c896dbf6662771 timestamp: 1,555,582,419,536,000 tags.http.host: ts-station-service tags.http.method: POST tags.http.path: /station/queryForId tags.http.url: http://ts-station-service:12345/station/queryForId tags.spring.instance_id: cdac01d7c2b8:ts-basic-service:15680 _id: L3XwL2oBeWuhw1dsBvMb _type: span _index: zipkin:span-2019-04-18 _score: -

traceId: 41714cc1a78618bc duration: 22,000 localEndpoint.serviceName: ts-travel-service localEndpoint.ipv4: 192.168.0.48 localEndpoint.port: 12346 timestamp_millis: April 18th 2019, 12:13:46.590 kind: CLIENT name: http:/ticketinfo/queryforstationid id: c6a6271d99cbfa27 parentId: 41714cc1a78618bc timestamp: 1,555,582,426,590,000.25 tags.http.host: ts-ticketinfo-service tags.http.method: POST tags.http.path: /ticketinfo/queryForStationId tags.http.url: http://ts-ticketinfo-service:15681/ticketinfo/queryForStationId tags.spring.instance_id: 76846330712d:ts-travel-service:12346 _id: V3XwL2oBeWuhw1dsH_NV _type: span _index: zipkin:span-2019-04-18 _score: -

traceId: 41714cc1a78618bc duration: 3,000 shared: true localEndpoint.serviceName: ts-route-service localEndpoint.ipv4: 192.168.0.42 localEndpoint.port: 11178 timestamp_millis: April 18th 2019, 12:13:46.639 kind: SERVER name: http:/route/querybyid/aefcef3f-3f42-46e8-afd7-6cb2 id: 21986d318b48ec9d parentId: 41714cc1a78618bc timestamp: 1,555,582,426,639,000 tags.mvc.controller.class: RouteController tags.mvc.controller.method: queryById tags.spring.instance_id: 54aef040bc8b:ts-route-service:11178 _id: X3XwL2oBeWuhw1dsH_OQ _type: span _index: zipkin:span-2019-04-18 _score: -

Fig. 5. A fragment of the raw log in Elasticsearch

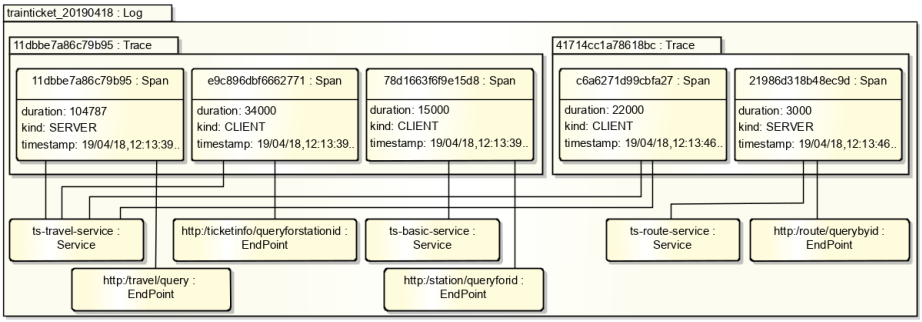


Fig. 6. A sample of Log Model

The detection of changes in the considered metrics may represent another use case for log analysis. Non-functional metrics, such as the ones previously described, may change over time due to a variety of reasons, including system

evolution and changes in deployment configurations. As an example, the detection of deviations from the average response time of a scenario can guide the designer in localizing the components that may have caused the deviation. In this respect, being able to link metrics to design elements may represent a pivotal asset to maintain and evolve complex distributed systems.

4.2 Logs representation

In order to obtain a model-based representation of the previously described raw logs, we used a specific metamodel [4] that allows to specify Log models. A Log is characterized by a set of *Traces* representing the requests the system served during the monitoring period. Each Trace consists of a set of *Spans* representing execution events. A Span is defined by the following attributes: a *timestamp* at which the execution started, its *duration* in microseconds, and *kind* specifying the role of the call in the scenario (*SERVER*, *CLIENT*, or undefined for internal calls). Each Span has a reference to a *Service*, that is the component providing the operation, and a reference to the mapped *EndPoint*. Figure 6 shows a sample of a Log Model obtained from the raw log in Figure 5.

Log Models conforming to the Log Metamodel are automatically generated by means of a Java transformation. Such transformation queries Elasticsearch to access raw logs in text format and subsequently transforms them in XMI-encoded EMF [7] models.

5 Design-Runtime Interactions via Traceability Links

In the previous section, we showed how it is possible to obtain runtime log models in a suitable format. These models can now be used along with the UML model to define design-runtime interactions. While different approaches exist to specify and implement such interactions, we propose a solution based on the JTL (Janus Transformation Language) framework [9]. It is an Eclipse EMF-based framework designed to mantain consistency between software artefacts., furthermore its traceability engine [12] allows to specify correspondences between elements that are defined at metamodel level as a set of relations between two domains.

```
1  transformation Log2UML (log:Log, uml:UML) {
2  ...
3  top relation Trace2UseCase {
4  checkonly domain log t : Log::Trace { spans = s:Log::Span{} };
5  checkonly domain uml uc : UML::UseCase {
6  ownedBehavior = ob : UML::Interaction { message = m:UML::Message{} }
7  };
8  where { Span2Message(s, m); }
9  }
10 relation Span2Message {
11 checkonly domain log s : Log::Span { endpoint = ep:Log::EndPoint{} };
12 checkonly domain uml m : UML::Message { signature = s:UML::Operation{} };
13 where { EndPoint2operation(ep, s); }
14 }
15 relation EndPoint2Operation {
```

[7] Eclipse Modeling Framework: https://www.eclipse.org/modeling/emf/

```
16  n : String;
17  checkonly domain log ep : Log::EndPoint { name = n };
18  checkonly domain uml s : UML::Operation { name = n };
19  }
20  top relation Service2Component {
21  n : String;
22  checkonly domain log s : Log::Service { name = n };
23  checkonly domain uml c : UML::Component { name = n };
24  checkonly domain uml l : UML::Lifeline { name = n };
25  }
26  ...
27  }
```

Listing 1. Log2UML correspondences specification

For instance, Listing 1 reports an excerpt of the correspondences between Log and UML metamodels in the JTL syntax.; in detail:

- the top relation *Trace2UseCase* (Lines 3-9) creates a correspondence between a trace and a scenario by relating Trace elements in the Log domain and UseCase elements in the UML domain.
- the *Span2Message* relation (Lines 10-14) maps a Span element to the corresponding UML Message by relating variables *ep* and *s* of type EndPoint and Operation, respectively.
- the *EndPoint2Operation* relation (Lines 15-19) creates a correspondence between an EndPoint of a Span and the UML Operation matching the same name;
- the top relation *Service2Component* (Lines 20-25) maps a Service element in Log to both the UML Component and Lifeline with the same name.

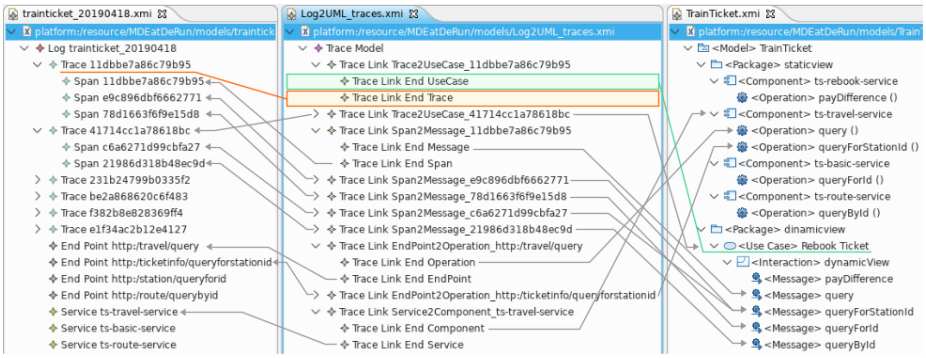


Fig. 7. Traceability model between the Train Ticker Log and UML model

The JTL traceability engine is able to produce a *design-runtime traceability model* by executing the correspondences specification in Listing 1. The resulting traceability model is conform to the Traceability Metamodel presented in [12]. Figure 7 shows a fragment of the traceability model (in the middle) connecting a Train Ticket Log model (left-hand side) and the UML model (right-hand side). A dataset containing logs and traceability models obtained from extensive monitoring is publicly available [8]. Links among elements on both sides are depicted as arrows in the figure and implemented as EMF inter-model references.

[8] https://github.com/SEALABQualityGroup/traceability-datasets

6 Related Work

Achieving an efficient integration between the design and runtime aspects of complex systems proved to be an interesting challenge for MDE methods and tools, as advocated during the first edition of the Model-Driven Engineering for Design-Runtime Interaction in Complex Systems (MDE@DeRun 2018) workshop [8]. In this context, we identify such lack of existing approaches that explicitly exploit design-runtime interaction with the aim at improving software design.

As mentioned in Section 1, the European project MegaM@Rt2 aims at providing a runtime-design time feedback loop in the development of complex systems. Within the project, we presented the following contributions on this topic. In [4] design-runtime relationships have been defined and used to support the performance improvement of a running system; the approach has been applied to an e-commerce web application designed by means of UML software models profiled with MARTE. In [11] the authors present a model-based approach that analyzes runtime data and automatically infers potential design issues that might need to be fixed in order to solve detected system malfunctioning. The approach is applied in the context of a real railway industrial system.

With the aim to extend the applicability of software models produced in MDE approaches to the runtime environment, the modeling community proposed the use of models at runtime (known as Models@run.time) [7, 6] . Such models should represent the system and its current/updated state and behavior to support adaptive systems, e.g., to drive subsequent adaptation decisions, to fix design errors or to explore new design decisions.

In the software engineering community, other approaches have introduced reference case studies with specific aims, e.g., for performance analysis [16]. Von Kistowsk et al. [16] have presented a micro-service reference application for performance benchmarking, and modelling. In particular, Teastore is a Java microservice oriented application and the authors exploit the Palladio Component Model (PCM) [5] as modelling formalism; thus they can apply different non-functional analyses by exploiting runtime data information. Herold et al. [13] have presented a UML reference model for a bank system. They have produced different diagrams, such as a Component Diagram, Sequence Diagrams for different scenarios as Larman described in [14]. Similarly to the above mentioned works, we provided the modeling of a given application. While they focused on performance analysis scope, we reversed the software architecture by exploiting runtime traces (i.e., log) and we defined traceability links between design elements and runtime data.

7 Conclusion

In this paper, we present a reference case study for design-runtime interaction in MDE. We presented the Train Ticket microservice-based web application and its monitoring infrastructure. The proposed case study is composed of a set of software modeling artifacts designed in UML, a dataset of monitoring logs, and a set of design-runtime correspondence defined as traceability links. The complete

resources of the case study have been made available with the scope to provide a reference case both for a better understanding of the related challenges and for extending or new contribution to this topic.

References

1. The OpenTracing project, https://opentracing.io/
2. Unified modeling language. OMG (2015), http://www.omg.org/spec/UML/2.5/, version 2.5
3. Afzal, W., Brunelière, H., Di Ruscio, D., Sadovykh, A., Mazzini, S., Cariou, E., Truscan, D., Cabot, J., Gómez, A., Gorroñogoitia, J., Pomante, L., Smrz, P.: The megam@rt2 ECSEL project: Megamodelling at runtime - scalable model-based framework for continuous development and runtime validation of complex systems. MICPRO **61**, 86–95 (2018)
4. Arcelli, D., Cortellessa, V., Di Pompeo, D., Eramo, R., Tucci, M.: Exploiting architecture/runtime model-driven traceability for performance improvement. In: ICSA. pp. 81–90 (2019)
5. Becker, S., Koziolek, H., Reussner, R.H.: The Palladio component model for model-driven performance prediction. Systems and Software **82**(1), 3–22 (Jan 2009)
6. Bencomo, N., Götz, S., Song, H.: Models@run.time: a guided tour of the state of the art and research challenges. Software & Systems Modeling (2019)
7. Blair, G., Bencomo, N., France, R.B.: Models@run.time. Computer **42**(10), 22–27 (2009)
8. Bruneliere, H., Eramo, R., Gomez, A., Besnard, V., Bruel, J.M., Gogolla, M., Kastner, A., Rutle, A.: Model-Driven Engineering for Design-Runtime Interaction in Complex Systems: Scientific Challenges and Roadmap - Report on the MDE@DeRun 2018 workshop. In: Proc. of STAF Collocated Workshops (2018)
9. Cicchetti, A., Di Ruscio, D., Eramo, R., Pierantonio, A.: JTL: A bidirectional and change propagating transformation language. In: SLE Proc. pp. 183–202 (2010)
10. Cito, J., Leitner, P., Bosshard, C., Knecht, M., Mazlami, G., Gall, H.C.: PerformanceHat: augmenting source code with runtime performance traces in the IDE. In: Proc. of ICSE Companion. pp. 41–44 (2018)
11. Eramo, R., Marchand de Kerchove, F., Colange, M., Tucci, M., Ouy, J., Bruneliere, H., Di Ruscio, D.: Model-driven design-runtime interaction in safety critical systemdevelopment: an experience report. In: (ECMFA) (2019), to appear
12. Eramo, R., Pierantonio, A., Tucci, M.: Improved traceability for bidirectional model transformations. In: MDETools, MODELS. vol. 2245, pp. 306–315 (2018)
13. Herold, S., Klus, H., Welsch, Y., Deiters, C., Rausch, A., Reussner, R., Krogmann, K., Koziolek, H., Mirandola, R., Hummel, B., Meisinger, M., Pfaller, C.: CoCoME - The Common Component Modeling Example, pp. 16–53 (2008)
14. Larman, C.: Applying UML and patterns: an introduction to object oriented analysis and design and interative development. Pearson Education India (2012)
15. Schmidt, D.C.: Model-driven engineering. IEEE Computer **39**(2), 25–31 (2006)
16. von Kistowski, J., Eismann, S., Schmitt, N., Bauer, A., Grohmann, J., Kounev, S.: Teastore: A micro-service reference application for benchmarking, modeling and resource management research. In: (MASCOTS). pp. 223–236 (Sep 2018)
17. Zhou, X., Peng, X., Xie, T., Sun, J., Xu, C., Ji, C., Zhao, W.: Benchmarking microservice systems for software engineering research. In: ICSE Companion Proceedings. pp. 323–324 (2018)