Editor et al. (Hrsg.): Digital Ecosystems of the Future: Methods, Techniques and Applications (EMISA),
0 Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2019

Model-driven Runtime State Identification

Sabine Wolny[1], Alexandra Mazak[2], Manuel Wimmer[3], Christian Huemer[4]

Abstract: With new advances such as Cyber-Physical Systems (CPS) and Internet of Things (IoT), more and more discrete software systems interact with continuous physical systems. State machines are a classical approach to specify the intended behavior of discrete systems during development. However, the actual realized behavior may deviate from those specified models due to environmental impacts, or measurement inaccuracies. Accordingly, data gathered at runtime should be validated against the specified model. A first step in this direction is to identify the individual system states of each execution of a system at runtime. This is a particular challenge for continuous systems where system states may be only identified by listening to sensor value streams. A further challenge is to raise these raw value streams on a model level for checking purposes. To tackle these challenges, we introduce a model-driven runtime state identification approach. In particular, we automatically derive corresponding time-series database queries from state machines in order to identify system runtime states based on the sensor value streams of running systems. We demonstrate our approach for a subset of SysML and evaluate it based on a case study of a simulated environment of a five-axes grip-arm robot within a working station.

Keywords: Model-driven Engineering; Time-Series Database; State Identification; Runtime Queries; Process Mining

1 Introduction

Forecasts show that in the upcoming years most of the devices we interact with will be linked to a global computing infrastructure [BS14]. This tendency represents an infrastructure in which the physical environment is populated by interconnected and communicating objects (e.g., sensors, actuators and other smart devices) capable for autonomously interacting with each other and with the environment itself. In order to deal with the increasing complexity of cyber-physical systems (CPS), models are used in many research fields as abstract descriptions of reality. This means that a model serves as an abstraction for a specific purpose, as a kind of “blueprint” of a system, describing the system’s structure as well as desired behavior. However, often we recognize a discrepancy between these models and their real world correspondents [MW16b]. In other words, we experience deviations between design-time models and runtime models discovered from real data.

[1] JKU Linz, CDL-MINT, Linz, Austria sabine.wolny@jku.at
[2] JKU Linz, CDL-MINT, Linz, Austria alexandra.mazak-huemer@jku.at
[3] JKU Linz, CDL-MINT, Linz, Austria manuel.wimmer@jku.at
[4] TU Wien, BIG, Vienna, Austria huemer@big.tuwien.ac.at

This development raises new challenges for *Model-Driven Engineering (MDE)* approaches [MWP18]. While design models help in the engineering process by providing appropriate abstractions, data-driven approaches such as process mining [Aa16] may help to uncover some under-specified or unintended parts of these design models at runtime. For instance, on a high level of abstraction, behavioral modeling languages (e.g., state-machine-based languages) are used to describe the behavior of a physical asset by means of states and transitions. Such models define discrete states, which are represented by defined variable values. A system has to achieve/go through these states during its execution. However in reality, systems do not switch in a time discrete manner between states, but the values of the variables are continuously evolving to the intended values of the next state. This means, each variable undergoes a continuous series of changes that need to be continuously monitored, e.g., to be able to react immediately to a time delay in safety critical systems. The challenge is to continuously listening to value streams in order to determine whether a state has indeed occurred, i.e., if the specific combinations of variable values have occurred over all streams at the same time. In particular, the realization precision of systems as well as measuring inaccuracies complicate this process as false positives and false negatives may occur when matching state templates to data streams.

Based on first ideas presented in [Wo17], we address this challenge by introducing a novel approach where we automatically generate state realization event queries derived from state machines for an appropriate state identification at runtime. This approach enables us to continuously observe multiple data streams of distributed sensor devices for identifying a system's entire state at runtime. By applying the so-called *Model-driven Runtime State IdEntification (MD-RISE)* approach, we automatically transform behavioral models, i.e., state machines, into time-series queries to be able to match sensor value streams with pre-defined variable values of the design model to report identified states from execution. First evaluation results derived from a case study of a 5-axes grip-arm robot show the potential of the approach in terms of precision and recall of finding system states in sensor value streams. By this, state based monitoring is possible for instance, even if the systems are not able to provide a explicit state-based trace.

The remainder of this paper is structured as follows. In the next section, we present a motivating example for our approach. Section 3 presents the MD-RISE approach by describing the MD-RISE architecture and its prototypical implementation. Section 4 demonstrates the evaluation of MD-RISE based on a case study of a 5-axes grip-arm robot which is interacting with other components within a working station, like a pick-and-place unit. In Section 5, we discuss related work. Finally, we conclude this paper by an outlook on our next steps in Section 6.

2 Motivating Example

As motivating example for this paper, we consider a simple continuous automated system around a 3-axes grip-arm robot (gripper). This gripper is modeled by using by the *Systems*

Modeling Language (SysML) [FMS12], in particular by using the *block definition diagram (BDD)* and *the state machine diagram (SM)*. The BDD is used to define the structure of the gripper with its properties: `BasePosition (BP)`, `MainArmPosition (MAP)`, and `GripperPosition (GP)` (see Fig. 1(a) Design Models, `BDD System`). These properties describe the angle positions of the three axes of the gripper. Based on the machine operator's knowledge, these angle positions can be defined for different settings (e.g, drive down, pick-up) with pre-defined tolerance ranges. These ranges fix the accepted margin of deviation (e.g., ± 0.1) for the variable values of each property (`BP, MAP, GP`). The desired behavior of the gripper is described by various states and state transitions modeled by using the SM (see Fig. 1(a) Design Models, `SM Grip-arm robot`). These states are `DriveDown` and `PickUp` with assigned variable values specifying the respective angle position in these states. During operation (i.e., execution at runtime), the gripper as a continuous system moves in its environment (e.g., pick-and-place unit) on the basis of a workflow described by the SM. These movements are recorded by various axis sensors and returned as continuous sensor value streams on a log recording system. In our motivating example, we record three sensor value streams `BP, MAP, GP` (see Fig. 1(b) Runtime Data). These records show that the gripper does not "jump" from one discrete state into another as modeled in the SM, but is–of course–continuously moving. Thus, the challenge is to identify possible discrete states by analyzing the sensor value streams. For this purpose we have to raise raw sensor value streams on a higher level of abstraction. This enables, e.g., to better compare an initial model (e.g., SM) with its realization.

The state identification is done by matching the different raw sensor value streams to the pre-defined variable values defined in the SM (see Fig. 1(b) Runtime Data). It should be

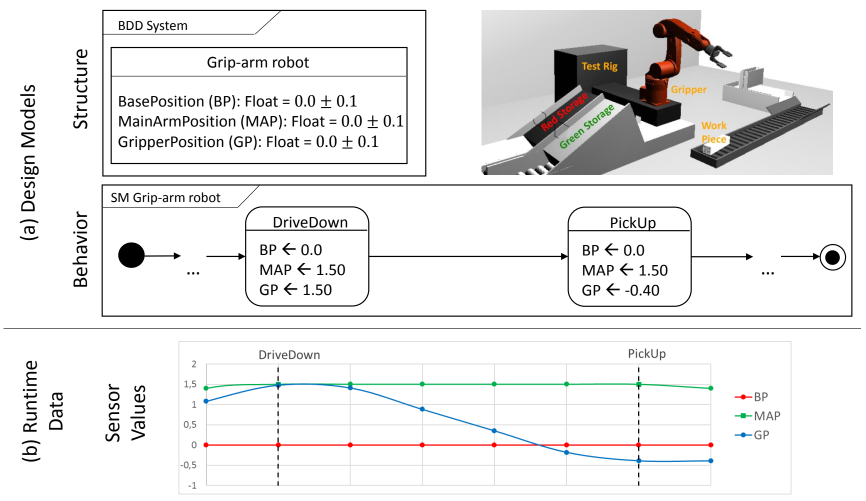

Fig. 1: Design time and runtime perspective of the motivating example

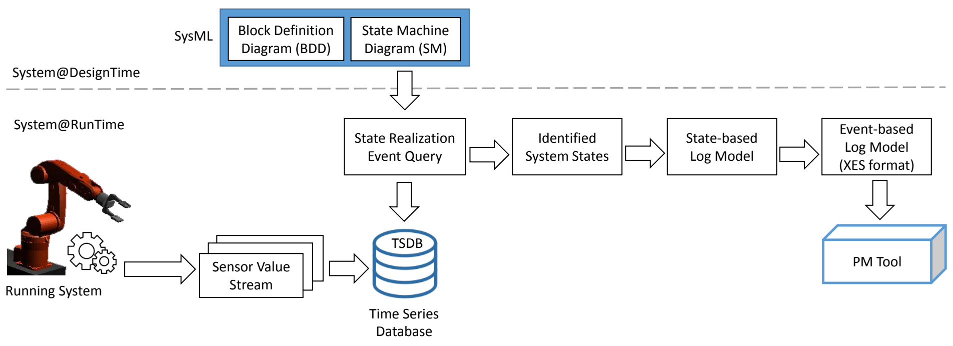

Fig. 2: Architecture for model-driven runtime state identification

considered that the pre-defined absolute variable values in the SM are not necessarily precisely measured in the real world because of, e.g., measuring inaccuracies. Such inaccuracies has to be taken into account by dealing with numerical values of objects of the physical world [MWV16]. Thus, in order to perform the state identification successfully, it is important to define appropriate tolerance ranges (see Section 4). For instance, the sequence of identified states can be used as input for further analysis (see Section 3).

3 Model-driven Runtime State Identification

In this section, we present our *Model-driven Runtime State IdEntification (MD-RISE)* approach which combines MDE-techniques with a *Time-Series Database (TSDB)* and *Process Mining (PM)*, for states identification, recording, abstraction, and analyses. Fig. 2 shows the architecture of MD-RISE as well as the interplay of design time and runtime artefacts.

3.1 MD-RISE Prerequisites

For prototypically realizing the approach, we have a number of prerequisites that must be met: (i) the system's workflow must be expressible by means of a state machine, (ii) the different states of the system must be unique in order that values describing a state are not identical for two different states, (iii) numeric values must be returned by sensors at runtime and must be storable in a TSDB, and (iv) it must be ensured that the time stamps are accessible.

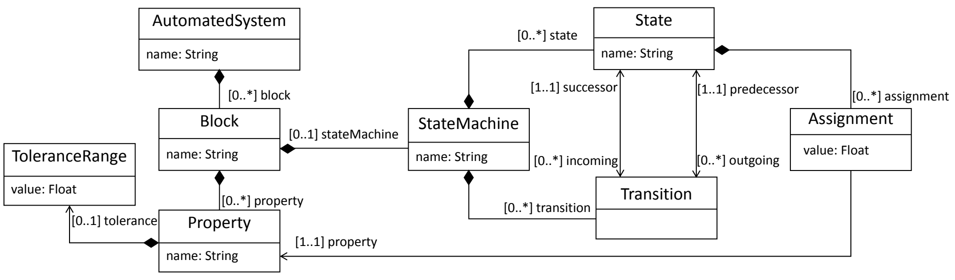

Fig. 3: Metamodel for describing a simple automated system

3.2 MD-RISE Architecture

Based on the motivating example of the gripper (see Section 2) and the mentioned prerequisites, we consider an automated system consisting of a controller, sensors, and actuators. At design time, we model the structure and behavior of this system by using a subset of SysML (see Figure 2: System@DesignTime, `BDD` and `SM`). Fig. 3 shows the simplified graphical metamodel used for modeling BDD and SM of the system. Every component of the system (`Block`) contains properties (`Property`) and can have a SM (`StateMachine`), which describes the behavior of this component. Each Property can have a specified tolerance range (`ToleranceRange`) that defines an acceptable deviation of the assigned property values, e.g., based on measurement inaccurancies. The SM consists of states (`State`) and transitions (`Transition`). Generally, a state can have multiple incoming and outgoing transitions. A transition must have a predecessor and successor state (see Fig. 3). Additionally, different values can be assigned to a state (`Assignment`). In this paper, we just focus on Float property values, since we are interested in value changes during execution (see Section 4).

Based on this metamodel, we automatically derive a query on the basis of the SM, a so-called “state realization event query” (see Fig. 2, System@RunTime). This query helps for identifying states based on the recorded sensor value streams in a TSDB. For this purpose we use a *Model-to-Text (M2T) transformation* to automatically transform model elements to query statements (i.e., text strings) (see Subsection 3.3). During runtime, the sensors of the running system continuously send data over a messaging system middleware. These sensor value streams (e.g., values of the angle positions of the gripper) are recorded in a TSDB (see Figure 2). A single log of the stream contains the following information: *timestamp* (the actual time in the granularity of seconds), *sensor* (the name of the specific sensor), *value* (the measured value). The number of log entries for one component varies depending on the number of sensors. The challenge is to continuously listening to value streams in order to determine whether a state has indeed occurred, i.e., if the specific combinations of variable values have occurred over all streams at the same time (see Fig. 1(b) Runtime Data). For this purpose we apply the aforementioned state realization event query for identifying states containing the following information: *timestamp* (the actual time in the granularity of seconds), *state* (the recognized state based on measured values).

However, the absolute values assigned in the SM at design time (see Fig. 1(a) Design Models) are necessarily not precisely identified as such during runtime due to measurement inaccuracies. For instance, we define for a certain state (e.g, `DriveDown`) a value of 1.50 for a certain angle position (e.g., `MAP`) at design time, but at runtime we measure a value for this position of 1.492. For this purpose we implement a tolerance range, assigned to the initial model (e.g., the SM), to define in which range such inaccuracies are still acceptable (see Figure 3, `ToleranceRange`). The definition of such a range is crucial. If the range is selected too small, the inaccuracies may result in too few or even no identified states. Otherwise, if the range is too large, too many states are identified. We examine this challenge in our case study presented in Section 4.

In a next step, we generate a state-based log model that consists of the information of all identified states and, in addition, a case ID for identifying the corresponding process instance (see Fig. 2: System@RunTime, `State-based Log Model`). Such a case ID is required when using PM tools in order to be able to distinguish different executions of the same process. We employ this case ID in our approach to identify single runs of the SM during runtime. In a further step, the state-based log model is transformed to an event-based log model (see Fig. 2, `Event-based Log Model`) by applying a *Model-to-Model (M2M) transformation*, like presented in previous research work [MW16a]. Since, we use a PM tool for analyzing this model, the structure must be based on *eXtensible Event Stream (XES)* schema. This is a supported input format of ProM Lite[5] 1.1. For instance, by using this PM tool, the event-based log model can be analyzed, e.g., to uncover some under-specified or unintended events that were not considered in the SM.

In summary, by applying MD-RISE it is now possible to raise raw sensor value streams on a higher level of abstraction, namely the state level. MD-RISE bases on queries, so-called state realization event queries, which are automatically derived from an initial design model for the purpose of state identification at runtime. The identified system states can be automatically transformed into a state-based log model to make the outcome useable, e.g., for PM tools like ProMLite for further analyses.

3.3 MD-RISE Prototypical Realization

For a first prototypical realization, we use the defined metamodel (see Fig. 3) and implement it by using Ecore in the Eclipse Modeling Framework[6]. Based on this metamodel, we develop a M2T transformation by using Xtend[7] in order to automatically generate state realization event queries out of the SM for different states. The structure of this M2T

[5] `http://www.promtools.org/doku.php?id=promlite11`
[6] `https://www.eclipse.org/modeling/emf`
[7] `http://www.eclipse.org/xtend`

transformation depends on the used TSDB. In our implementation, we use InfluxDB[8] as TSDB. Therefore, the structure of our state realization event queries are similar to a SQL syntax, as shown in the following pseudo code example based on our metamodel:

```
«FOR s IN Block.stateMachine.state»
SELECT «FOR a IN s.assignment» «a.property.name», «ENDFOR» time
FROM «Block.name»
WHERE «FOR a IN s.assignment»
«a.property.name»>=«a.value-a.property.tolerance.value»
and «a.property.name»<=«a.value+a.property.tolerance.value»«ENDFOR»
«ENDFOR»
```

Based on the raw sensor value streams collected at runtime and stored in the TSDB, the queries are executed and the results are the identified states with their timestamps. In our prototypical implementation, we store the outcome as csv-file, which is then used as input for the state-based log model. This model is a Ecore model representation of the csv-file. In a next step, we use the *Atlas Transformation Language (ATL)*[9] as transformation tool to transform the state-based log model to an event-based log model for importing it into ProM[10] [MW16a]. The full implementation of MD-RISE can be found at our project website[11].

4 Case Study based on a CPPS-Simulation Environment

In this section, we present as well as discuss the accuracy and limitations of MDE-RISE on the basis of a case study of a CPPS-simulation environment around a 5-axes grip-arm. In doing so, we follow the guidelines for conducting empirical explanatory case studies by Runeson and Höst [RH09]. In particular, we report on applying our approach to detect states at runtime based on stored value streams in a TSDB.

4.1 Research Questions

The study was performed to quantitatively assess the completeness, correctness, and performance of MDE-RISE. More specifically, we aimed to answer the following research questions (RQs):

[8] `https://www.influxdata.com`
[9] `https://www.eclipse.org/atl`
[10] `http://promtools.org/doku.php`
[11] `https://cdl-mint.big.tuwien.ac.at/case-study-artefacts-for-emisa-2019/`

RQ1—Correctness: Are the identified states at runtime correct in the sense that all identified states are representing real states? If our approach identifies incorrect states, what is the reason for this?

RQ2—Completeness: Are the identified states complete in the sense that all expected states are correctly identified? If the set of identified states is incomplete, what is the reason for missed identifications?

RQ3—Performance: How strongly is the performance of the query execution influenced by the number of sensor value streams or the number of stored values per sensor?

4.2 Case Study Design

Requirements. As an appropriate input for our case study, we require an automated system such as a gripper integrated in a simulated environment where we are able to observe the behavior of the gripper during operation. We require access to multiple sensors of the gripper for log acquisition and a method to automatically identify states based on sensor value streams from simulation runs.

Setup. To fulfill these requirements, we implemented a CPPS-simulation of an autonomous acting production unit executed by using the open source tool Blender[12]. The simulation scenario considers a working station, like a pick-and-place unit, where a gripper takes work pieces from a conveyor belt, put them down on a test rig, and finally release them in a red or green storage box based on the information coded on each work piece by a QR-code. Each component communicates via a messaging system middleware with InfluxDB. This TSDB provides us to acquire raw sensor value streams. During simulation, the gripper enters several different states for processing the work pieces. To verify the correctness of our approach, we have chosen two very similar states (differ only in one sensor value stream) to determine if the detection works: `DriveDown` and `PickUp`. The assigned values of the axes `Base Position (BP), Main Arm Position (MAP), Second Arm Position (SAP), Wrist Position (WP), and Gripper Position (GP)` of the two states in the SM are shown in Tab. 1. Furthermore we need to define an acceptable tolerance range to determine when the state identification is as accurate and complete as possible. We use a tolerance range from a deviation of 0 to a deviation of 0.4 (in 0.01 steps). The upper bound is only set for evaluation purposes to show the distribution of precision and recall. In reality, a deviation of 0.4 may be already too large. The deviation values are added or subtracted to the respective SM values (see Tab. 1). We use the same tolerance ranges for all properties and do not vary them.

For our evaluation we use two different database settings in combination with different numbers of sensor value streams that are used for the states identification. We use a dataset

[12] `https://www.blender.org`

Tab. 1: Expected values for the gripper's axes for the states DriveDown and PickUp.

State / Gripper Axis	DriveDown	PickUp
Base Position (BP)	0.0	0.0
Main Arm Position (MAP)	1.50	1.50
Second Arm Position (SAP)	-0.12	-0.12
Wrist Position (WP)	0.0	0.0
Gripper Position (GP)	1.5	-0.40

with 156 rows and a dataset with 1,560 rows stored in the database. For the state identification we use a single sensor value stream (GP), three sensor value streams (GP, BP, MAP), and all five sensor value streams (GP, BP, MAP, SAP, WP). For the performance check we also extend our dataset up to 15,600, 156,000, and 1,560,000 rows.

For performance purpose of the state realization queries, we calculate the duration between start of the query execution and result return by System.nanoTime() in Java. The performance figures have been measured on an Acer Aspire VN7-791 with an Intel(R) Core(TM) i7-4720 HQ CPU @ 2.60 GHz 2.60 GHz, with 16 GB of physical memory, and running the Windows 8.1. 64 bits operating system. Please note that we measured the CPU time by executing each query 40 times for all different settings and calculated the arithmetic mean of these runs in milliseconds (ms).

Measures. In order to assess the accuracy of our approach, we calculate *precision* and *recall* as defined in [MRS08]. In the context of our case study, precision denotes the fraction of *correctly identified* states among the set of *all identified* states. Recall indicates the fraction of *correctly identified* states among the set of *all actually occurring* states. Precision denotes the probability that a identified state is correct and the recall is the probability that an actually occurring state is identified. Both values range from 0 to 1.

Precision is used to answer RQ1 and recall to answer RQ2. Furthermore, we calculate the so-called *f-measure* to avoid having only isolated views on precision and recall [MRS08]. To answer RQ3, we compute the duration of the query execution.

To check if our approach is accurate for a given scenario to identify system states, we have manually obtained the gold standard of state identifications for our given case study (156 rows: 3 expected states for DriveDown and PickUp, 1560 rows: 30 exepected states for DriveDown and PickUp). For computing precision and recall, we extract the true-positive values (TPs), false-positive values (FPs) and false-negative values (FNs), with the help of the expected state identifications. From the TP, FP and FN values we then compute *precision*, *recall* and *f-measure* metrics as defined by Olson and Delen [OD08, p. 138].

Tab. 2: Precision, recall and f-measure for a single sensor value stream (GP). Bold line marks the best fit.

	DriveDown			PickUp		
tolerance range	**precision**	**recall**	**f-measure**	**precision**	**recall**	**f-measure**
0	NaN	0	NaN	NaN	0	NaN
0.01	NaN	0	NaN	0.08	1	0.14
0.02	**1**	**1**	**1**	**0.08**	**1**	**0.14**
0.03-0.05	1	1	1	0.07	1	0.14
0.06-0.08	1	1	1	0.07	1	0.13
0.09-0.11	0.75	1	0.86	0.07	1	0.13
0.12-0.19	0.75	1	0.86	0.07	1	0.12
0.20-0.30	0.6	1	0.75	0.05	1	0.10
0.31-0.37	0.5	1	0.67	0.05	1	0.10
0.38-0.39	0.5	1	0.67	0.05	1	0.09

4.3 Results

We now present the results of applying our approach to the different settings of our gripper simulation. Tab. 2−Tab. 4 show the results for precision, recall and f-measure for the two different states in the different value stream settings. The values are valid for both database settings (156 rows, 1560 rows), since there were no differences with regard to precision, recall and f-measure. This can be explained by the fact that the queries are independent of the number of values in the database. As soon as the sensor value streams are in the accepted tolerance range, the state is returned.

Tab. 3: Precision, recall and f-measure for three sensor value streams (GP, BP, MAP). Bold line marks the best fit.

	DriveDown			PickUp		
tolerance range	**precision**	**recall**	**f-measure**	**precision**	**recall**	**f-measure**
0	NaN	0	NaN	NaN	0	NaN
0.01	NaN	0	NaN	1	1	1
0.02-0.08	**1**	**1**	**1**	**1**	**1**	**1**
0.09-0.10	0.75	1	0.86	1	1	1
0.11-0.12	0.75	1	0.86	0.6	1	0.75
0.13-0.16	0.75	1	0.86	0.5	1	0.67
0.17-0.18	0.75	1	0.86	0.43	1	0.6
0.19	0.75	1	0.86	0.25	1	0.4
0.20-0.21	0.6	1	0.75	0.25	1	0.4
0.22-0.30	0.6	1	0.75	0.23	1	0.375
0.31-0.39	0.5	1	0.67	0.23	1	0.375

It is noticeable that the states identification fails and no states are found if the tolerance range is too small. The larger the range, the more false states are detected and the precision decreases as expected. In Tab. 2 for the state `PickUp` it could be recognized that the precision value is really small (highest value 0.08), because of wrong states identification based on a single sensor value stream. This can be explained by the fact that the gripper moves during the simulation and opens and closes the gripper arm in various locations (e.g., conveyor, test rig). These states do not differ in the value of GP but have a different BP. Thus, this one axis GP is not enough to identify the state `PickUp`. Furthermore, it is interesting that the use of all gripper's axes for state identification `PickUp` leads to a lower recall for the tolerance range 0.01 (see Tab. 4).

Tab. 4: Precision, recall and f-measure for five sensor value streams (GP, BP, MAP, SAP, WP). Bold line marks the best fit.

	DriveDown			PickUp		
tolerance range	**precision**	**recall**	**f-measure**	**precision**	**recall**	**f-measure**
0	NaN	0	NaN	NaN	0	NaN
0.01	NaN	0	NaN	1	0.33	0.5
0.02-0.08	**1**	**1**	**1**	**1**	**1**	**1**
0.09-0.10	0.75	1	0.86	1	1	1
0.11-0.12	0.75	1	0.86	0.6	1	0.75
0.13-0.16	0.75	1	0.86	0.5	1	0.67
0.17-0.18	0.75	1	0.86	0.43	1	0.6
0.19	0.75	1	0.86	0.25	1	0.4
0.20-0.21	0.6	1	0.75	0.25	1	0.4
0.22-0.3	0.6	1	0.75	0.23	1	0.375
0.31-0.39	0.5	1	0.67	0.23	1	0.375

Figure 4 shows the results of our performance check. It could be determined that the number of sensor value streams and the number of rows in the database both have an influence on the execution time.

Interpretation of results. *Answering RQ1*: The recognition of correct states depends on the defined tolerance range and the differentiability of states. A precondition for our

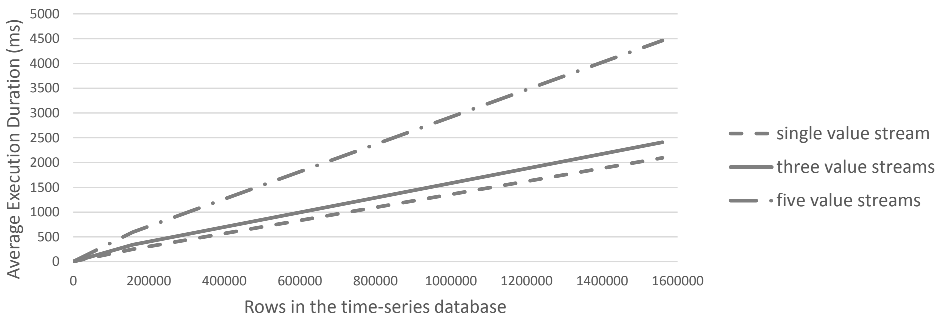

Fig. 4: Performance Results (average execution time in ms) according to sensor value streams and rows in the TSDB.

approach is the uniqueness of states. However, if the various states differ only slightly, the number of sensor value streams used for states identification is relevant for correctness.

Answering RQ2: The selected tolerance range and the number of sensor value streams are also decisive for the completeness of the states identification. The more sensor value streams are used, the more important individual sensor values become for the identification. In addition, the completeness of the identified states is better the larger the selected tolerance range is. In our evaluation we quickly achieve a good completeness. As soon as this is reached, the tolerance range should not be further increased, otherwise the correctness of the identified states suffers.

Answering RQ3: Our investigations of the execution time already show in this simple setting the influence of the number of data records in the database and the used number of different sensor value streams. However, the performance seems still promising for large cases as we experience a linear increase of execution time for all tested settings.

4.4 Threats to Validity

Internal Validity - Are there factors that can influence the results of the case study? At design time, values for our axis positions are assumed on the basis of, e.g., calibration values. At runtime, the same exact values are not always measured, but with a certain fluctuation range. Thus, a certain tolerance range must be defined at design time in which the values are accepted. In our case, we knew exactly which values to expect and were therefore able to keep our tolerance range small. However, this might not work with other settings.

External Validity - Is it possible to generalize the results? Our approach is based on queries automatically created by state machines. We focus on creating queries that are understood by the TSDB InfluxDB. Thus, the queries are currently in SQL syntax. If a different database query language is needed, only the Xtend code has to be adapted regarding syntax without changing the model in the background. At the moment our evaluation is based on a single case of a gripper simulation. For further and more detailed results the study has to be extended to other scenarios. Raw data from sensors are often noisy, incomplete and can contain erroneous records. This is not considered in our case study. In addition, the datasets for performance analysis are relatively small in relation to databases. Larger sets would be needed for further more detailed results.

5 Related Work

Discovering the behavior of running software. In [Li16], the authors utilize process mining (PM) techniques to discover and analyze the real behavior of software. By doing so, they discover behavioral models for each software component by considering hierarchies. In a first step of their approach, they identify component instances and construct event logs for each component from raw software execution data. In a second step, they recursively transform the logs to a hierarchical event log for each component by considering calling relations among method calls. Based on these hierarchical event logs, the authors discover a hierarchical process model to understand how the software is behaving at runtime. The authors' software component behavior discovery builds on the inter-disciplinary research field of *Software Process Mining (SPM)*, firstly introduced by Rubin et al. [Ru07]. Both approaches base their grounding on the well-established techniques and methods of the research field of PM [Aa16].

Applying reverse engineering for obtaining event logs. In [LA15], the authors present a reverse engineering technique based on PM for obtaining real event logs from distributed systems. Similar to [Li16], the authors present an inter-disciplinary approach based on PM techniques and reverse engineering. The aim of their approach is to analyze the operational processes of software systems when running. The formal definition, implementation, and instrumentation strategy of the approach bases on a joinpoint-pointcut model (JPM) known from the area of aspect-oriented programming [EFB01]. This JPM helps (*i*) by defining the parts of a system that are to be included, (*ii*) enables to quickly gain insight into the end-to-end process, and (*iii*) detects the main bottlenecks. The authors demonstrate the feasibility of their approach by two case studies.

Query-based process analytics. A query approach enabling business intelligence through query-based process analytics is presented by Polyvyanyy et al. [Po17]. In contrast to our approach they are focusing on PM techniques for the automated management of model repositories of designed and executed processes, and on the relationships among these processes. For this purpose the authors introduce a framework for specifying generic functionalities that can be configured and specialized to address process querying problems, such as filtering or manipulation of observed processes.

Finally, we would like to highlight two research works that underline our approach and discuss the differences. Mayr et al. [Ma17] critically note that models are mainly used as prescriptive documents. Therefore, the authors aim for a model-centered architecture paradigm to keep models and developed artefacts synchronized in all phases of software development as well as in the running system. In this context, our approach helps to lift raw sensor data through automated states identification during operation at a model level for enabling a comparison between prescriptive and descriptive models. Senderovich et al. [Se16] apply PM techniques for real-time locating systems. They solve the problem of mapping sensor data to event logs based on process knowledge since location data recordings do not relate to the process directly. Therefore, they provide interactions as an intermediate knowledge layer between the sensor data and the event log [Se16]. Contrary to our approach, their raw sensor log consists already of different business entities and they have to map interactions to activity instances, while the sensor logs in our approach consist only of numerical values which we first have to aggregate to events.

6 Conclusion and Future Work

In this paper, we presented an approach that automatically derives state realization event queries from the design model to identify system states of a continuous system based on sensor value streams at runtime. This enables to raise raw sensor data from the data layer on a higher model layer. At this model level, runtime processes can be analysed more quickly and possible unintended parts within the realized system may be identified more easily and time-saving. Since inaccuracies has to be taken into account by dealing with numerical values of objects of the physical world, additionally we implemented a tolerance range for defining in which range such inaccuracies are still acceptable for an identified state at runtime.

First results of our case study indicate that a high precision and recall of system state identification may be achieved if an appropriate tolerance range for the runtime values was defined. Nevertheless, the uniqueness and distinctiveness of the individual states determine whether the state identification works well or not. If states are very similar, enough different sensor value streams must be used for state identification to obtain a good precision and recall. The approach is a step towards a better integration of model-driven software development to all the operations within a system's life cycle in order to continuously deploy stable versions of application systems.

There are several lines for future work we are going to explore in more detail. First, we plan to apply and validate our approach in a real-world setting, instead of a simulation. Second, we want to extend our approach to monitor different components with a larger set of sensor value streams. Third, we only used identically tolerance ranges for the properties. In a further investigation, we want to find out if there are automated techniques possible to estimate good guesses for the tolerance ranges of different properties. Finally, we want to find out if we could extend our approach for state estimation and detection of possible hidden states.

Acknowledgment

This work has been supported by the Austrian Federal Ministry for Digital and Economic Affairs, the National Foundation for Research, Technology and Development, and by the FWF in the Project TETRABox under the grant number P28519-N31.

References

[Aa16] van der Aalst, W. M. P.: Process Mining-Data Science in Action. Springer, 2016.

[BS14] Broy, M.; Schmidt, A.: Challenges in Engineering Cyber-Physical Systems. Computer 47/2, pp. 70–72, 2014.

[EFB01] Elrad, T.; Filman, R. E.; Bader, A.: Aspect-oriented Programming: Introduction. Commun. ACM 44/10, pp. 29–32, Oct. 2001.

[FMS12] Friedenthal, S.; Moore, A.; Steiner, R.: A Practical Guide to SysML. Morgan Kaufmann, 2012, ISBN: 9780123852069.

[LA15] Leemans, M.; van der Aalst, W. M. P.: Process mining in software systems: Discovering real-life business transactions and process models from distributed systems. In: MODELS. Pp. 44–53, 2015.

[Li16] Liu, C.; van Dongen, B.; Assy, N.; van der Aalst, W. M. P.: Component behavior discovery from software execution data. In: SSCI. Pp. 1–8, 2016.

[Ma17] Mayr, H. C.; Michael, J.; Ranasinghe, S.; Shekhovtsov, V. A.; Steinberger, C.: Model Centered Architecture. In: Conceptual Modeling Perspectives. Springer, pp. 85–104, 2017.

[MRS08] Manning, C. D.; Raghavan, P.; Schütze, H.: Introduction to information retrieval. Cambridge University Press, 2008.

[MW16a] Mazak, A.; Wimmer, M.: On Marrying Model-driven Engineering and Process Mining: A Case Study in Execution-based Model Profiling. In: SIMPDA. Pp. 78–88, 2016.

[MW16b] Mazak, A.; Wimmer, M.: Towards Liquid Models: An Evolutionary Modeling Approach. In: CBI. Pp. 104–112, 2016.

[MWP18] Mazak, A.; Wimmer, M.; Patsuk-Bösch, P.: Execution-Based Model Profiling. In: Data-Driven Process Discovery and Analysis. Springer, pp. 37–52, 2018.

[MWV16] Mayerhofer, T.; Wimmer, M.; Vallecillo, A.: Adding uncertainty and units to quantity types in software models. In: SLE. Pp. 118–131, 2016.

[OD08] Olson, D. L.; Delen, D.: Advanced Data Mining Techniques. Springer, 2008, ISBN: 978-3-540-76916-3.

[Po17] Polyvyanyy, A.; Ouyang, C.; Barros, A.; van der Aalst, W. M.: Process querying: Enabling business intelligence through query-based process analytics. Decision Support Systems 100/, pp. 41–56, 2017.

[RH09] Runeson, P.; Höst, M.: Guidelines for conducting and reporting case study research in software engineering. Empirical Software Engineering 14/2, pp. 131–164, 2009.

[Ru07] Rubin, V.; Günther, C. W.; van der Aalst, W. M. P.; Kindler, E.; van Dongen, B. F.; Schäfer, W.: Process Mining Framework for Software Processes. In: Software Process Dynamics and Agility. Springer, pp. 169–181, 2007.

[Se16] Senderovich, A.; Rogge-Solti, A.; Gal, A.; Mendling, J.; Mandelbaum, A.: The ROAD from Sensor Data to Process Instances via Interaction Mining. In: CAiSE. Pp. 257–273, 2016.

[Wo17] Wolny, S.; Mazak, A.; Konlechner, R.; Wimmer, M.: Towards Continuous Behavior Mining. In: SIMPDA. Pp. 149–150, 2017.