Orchestrating Digital Twins for Distributed Manufacturing Execution Systems[⋆]

Tomáš Fiedor, Martin Hruška, and Aleš Smrčka
{*ifiedortom,ihruska,smrcka*}@fit.vutbr.cz

Brno University of Technology, Faculty of Information Technology, Czech Republic

Abstract. We propose a generic framework for generating scenarios for orchestrating digital twins of distributed manufacturing execution systems (MES). The orchestration is a typical technique for configuring, managing and coordinating communication in various types of systems. We focus on its particular application for testing of distributed systems: e.g., when a new version of MES is about to be released, one may create the testing scenario observing the behaviour of the older version already deployed in the production and then use the scenario to orchestrate the digital twin to test the new version. Specifically, we build on communication logs captured from a manufactory with deployed MES and create an abstract model of communication inside the distributed system and abstract models of messages passed in communication. Based on these models, we then generate a scenario, which is further used to orchestrate the digital twin. Moreover, our approach can also automatically extrapolate the new testing scenarios from the derived models allowing efficient automated testing.

1 Introduction

One of the main challenges of Industry 4.0 is to develop secure and bug-free components, especially in distributed, manufacturing execution systems. These systems usually include manufacturing machines paired with controlling terminals, industrial control systems (ICS), and/or system that controls the whole manufactory process (manufactory execution system). Development and testing of such systems are quite complex because their components (1) work in a distributed environment, (2) use different communication protocols, (3) use different software ranging from low-level embedded software to complex information systems, (4) require interaction between humans and machines, and (5) often cannot be tested in a real-world environment during the common traffic.
Moreover, any bug or security issue may be quite costly, which can be substantiated by the expected growth of the market of ICS security up to $22.2 billions by 2025 [1]. Quality assurance teams usually utilize some form of test automation while keeping the effort spent on the testing itself. Unfortunately, test automation of distributed manufacturing systems is hard for two main reasons.
First, testing in a real-world environment (so-called out-of-the-lab testing) is expensive. Hence, we usually construct the so-called digital twin: a virtual environment where components (such as production machines) are emulated or simulated

[⋆] This is the updated paper accepted to EUROCAST'22.

to replicate the digital copy of the manufacturing process. Such a copy can then be used for testing in an environment as close as possible to a real system.
The other problem is how to model communication among a number of quite different components common in the manufacturing process. The communication within the system is often purpose-specific and requires strong domain knowledge. Hence, creating the automated test suite is complicated as it requires effort spent on precise test environment setup and deterministic test case description.
In this work, we propose a generic framework for creating automated test suites for digital twins of manufacturing execution systems (MES). The framework analyses the communication captured from a run of the real system, learns a model of the communication protocol, and models of data sent. Based on these models, we generate test scenarios that are used for orchestration of corresponding digital twin. In particular, we suggest using such scenarios for the automated testing of systems, e.g., when a new version of MES is being developed.

2 Framework for Generating Orchestration Scenarios

We propose a generic framework that can be applied to various settings of distributed MES systems. In this paper, however, we will demonstrate it on a particular use case consisting of various types of nodes communicating using various protocols. We assume the following infrastructure: the distributed system consists of an Enterprise Resource Planning (ERP) system, the MES system that controls the actual production, manufacturing machines and their corresponding terminals used by human operators. We expect that the communication between particular components uses different protocols and data structures, e.g.: (1) MES and ERP communicate using REST protocol with XML data, (2) MES and Terminal communicate using REST protocol with JSON data, and (3) MES communicates with machines using OPC-UA protocol, although some minor manual tweaks for understanding specific-purpose data might be necessary.
An overview of our framework is shown in Figure 1. Our framework requires logs of communication collected from a real-world system, e.g., with an older version of the MES system under testing: the collected log usually represents either expected communication in the system or a log of communication that led to some incidents. The log is in the form of a sequence of messages between pairs of communicating components logged with the timestamps of the communication and the data that were transferred. We derive two kinds of model based on this log: (1) model of the data transmitted in the messages, and (2) the model of the whole communication in the system. To model communication, we convert the log to a so called event calendar which provides efficient and direct manipulation with seen messages. Then, we eventually convert event the calendar to a finite automaton (where every event is a symbol) which is a more abstract representation but provides options for postprocessing (e.g., by applying length abstraction to generate new test cases) or analyses (e.g., by searching for a particular string representing an error behaviour). From the derived (abstract) models, we generate a so-called scenario: a sequence of concrete messages that will be sent in a real-world or simulated environment. A scenario is later used by a digital twin orchestrator to perform simulation of the real system in digital twins. In our case, we use the Cryton tool [2] to orchestrate the simulated environment. Other orchestrators that conform to our scenarios format can also be naturally used. Finally, based

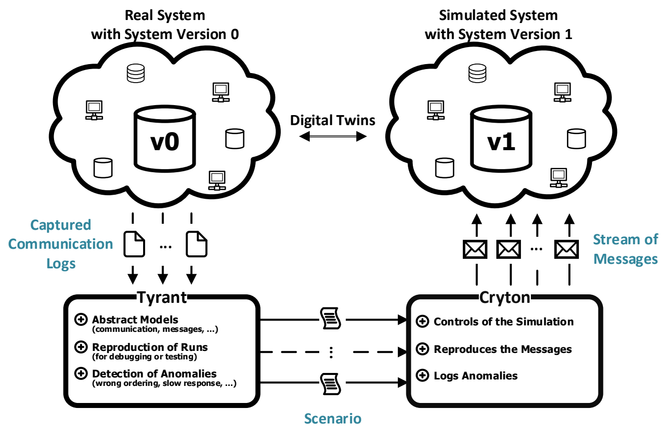

Fig. 1. Scheme of our solution.

on the result of orchestration, developers can observe whether the new version of a digital twin behaves as expected.
The framework can also be quite easily extended to support the performance testing of the digital twins. In particular, we propose to mine selected performance metrics (e.g., among others, the duration of communications) from the captured logs. The metrics are then used for comparison of runs from different environments or from different versions to detect, e.g., anomalies in the performance.

Related work. There have been several different approaches for modelling communication in manufacturing and deriving new test cases. [5] uses Finite Automata to model the communication in systems, however, their approach is limited to learning only fixed number of components. Another approach is process mining [10], a mature technique for modelling event-based systems. We see the technique as unsuitable for MES systems as it analyses one-to-one communications and is restricted to a single thread per node [8]. Modelling of communication for anomaly detection [9] implements an approach based on probabilistic automata. The usual communication in manufactory is, however, mostly deterministic, hence, no probabilistic transitions are created in a derived automaton. Finally, we can mention approaches of [6,7,4] which are research prototypes only and possibly not mature enough for the use case in real-world distributed systems.

3 Modelling Messages

In a distributed system, components usually communicate through messages. We assume that each message that was captured in the log has the following parts: (1) a timestamp (when the message was sent), (2) data (what was sent), and, (3) a type (what kind of message was sent). A suitable data representation of such messages can be a challenging task, especially, when modelling the communication among different components. The representation should be unified for different kinds of data formats (such as JSON, XML or YAML), should preserve the original semantics, and should allow generating new test cases from the observed data (e.g., extrapolating extreme values from the underlying domains).

Communication logs usually contain lots of subsets of messages that are structurally similar to each other and differ only in certain aspects (mainly in the data that were sent and the type of the message). Thus, we propose classification of the messages into groups of similar messages before creating abstract models. In particular, we classify the seen messages based on the so-called fingerprint of the message (i.e., the spanning tree of the nested structure with respect to the fields of the data) and based on the type of the message. The idea is that messages having similar structure (but that differ in, e.g., number of items in a list, or values in leaves) should have the same abstract model. For each such class, we construct an abstract model that represents all seen messages of the given class. Such a model can then be used not just to reproduce the communication but also to create new (potentially unseen) messages, e.g., by generating syntactically-similar messages.

We propose to model the messages using the following representation (simplified for the sake of presentation). We assume two types of nodes: (1) a leaf node is a quadruple $n = \langle k, l, u, V\rangle$, where k is a key associated with the node (e.g., as in JSON key-value pairs), l (resp. u) is the minimum (resp. maximum) number of occurrences of the node in the given part of the message, and V is the set of all seen values for the node; and (2) a composite node is a quadruple $n = \langle k, l, u, N\rangle$, where k, l, and u are defined the same as previously and N is a set of child nodes (leaves or composite). We assume that the root of every message is represented by the $root$ key. Note that we also support other types of nodes, e.g., the attribute node, used in XML format, but due to the limited scope of this paper, we omit their description.

To create abstract models, we process input log message by message (which are in XML or JSON format). Atomic values correspond to leaf nodes and composite values (lists, dictionaries, nested tags) correspond to composite nodes. Further, we will work with predicate c over node n written as $c(n)$ (e.g., representing that node n has a specific key). We denote the set of all possible predicates as C.

We define the function *reduce* over a set of leaves $\{n_1, \ldots, n_m\}$ with the same key k as $reduce(\{\langle k, l_1, u_1, V_1\rangle, \ldots, \langle k, l_n, u_n, V_n\rangle\}) = \langle k, \sum_1^n l_i, \sum_1^n u_i, \bigcup_1^n V_i\rangle$; similarly, we define the *reduce* of a set of composite nodes $\{n_1, \ldots, n_m\}$ corresponding to a key k as $reduce(\{\langle k, l_1, u_1, N_1\rangle, \ldots, \langle k, l_n, u_n, N_n\rangle\}) = \langle k, \sum_1^n l_i, \sum_1^n u_i, \bigcup_1^n N_i\rangle$. Then, for composite nodes, we define the *group and reduce* function as $grpreduce(\langle k, l, u, N\rangle, \langle c_1, \ldots, c_m\rangle) = \langle k, l, u, N'\rangle$, where $\langle c_1, \ldots, c_m\rangle$ are predicates and $N' = \bigcup_1^m \big\{reduce(\{n \in N \mid c_i(n)\})\big\}$. Basically, the operation groups the children nodes according to a given predicate (e.g., it groups children named with the same key), merges their values and aggregates their occurrences.

Finally, we define the *merge* of two nodes ($n \circ n'$) with the same key k as follows: (1) $\langle k, l_1, u_1, V_1\rangle \circ \langle k, l_2, u_2, V_2\rangle = \langle k, min(l_1, l_2), max(u_1, u_2), V_1 \cup V_2\rangle$, and (2) $\langle k, l_1, u_1, N_1\rangle \circ \langle k, l_2, u_2, N_2\rangle = \langle k, min(l_1, l_2), max(u_1, u_2), Merged(N_1, N_2) \cup Copy(N_1, N_2) \cup Copy(N_2, N_1)\rangle$, where $Merged(N, N') = \{n \circ n' \mid \exists c \in C\, \exists n \in N\, \exists n' \in N' : c(n) \wedge c(n')\}$ and $Copied(N, N') = \{n \mid \exists c \in C\, \exists n \in N\, \forall n' \in N' : c(n) \wedge \neg c(n')\}$. We choose values of the minimal and maximal number of occurrences to cover both nodes. In the composite nodes, we group the children satisfying the same criteria and recursively merge them. If a child node from N_1 (resp. N_2) with no node from

N_2 (resp. N_1) matches the same criterion, the first node is simply copied to the result. For simplicity, we assume that a criterion c is satisfied by maximally one node in one subtree. Finally, for each class and its messages with the root nodes $r_1, \ldots, r_n$, we compute the final abstract node n representing the class as $n = \mathit{grpreduce}(r_1, \langle c_1, \ldots, c_m \rangle) \circ \ldots \circ \mathit{grpreduce}(r_n, \langle c_1, \ldots, c_m \rangle)$.

4 Modelling Communication of Monitored System

Once we derived the models of messages communicated in a system, we further learn the communication protocol used in the environment. We first use an intermediate data structure called the *event calendar* to represent messages in the monitored system where each event corresponds to one message. The messages are ordered chronologically in the calendar by their timestamps. This way, we can represent the communication using different protocols and data formats in a unified and regular manner, and we are not limited to a fixed number of components. That would not be possible with other representations, which need predefined topology of a represented system. The calendar is later used to generate scenarios precisely reproducing the learnt communication by transforming each event to a single step in a scenario for orchestrating digital twin.
Moreover, we want to generate new test cases allowing to experiment on scenarios which have not yet been seen but are similar to a real-world situations. Such scenarios sometimes bring more testing value since they are relatively easy to generate in contrast to the time-demanding process of writing tests manually. Hence, we propose to transform the event calendar to *finite automaton* and apply, e.g., length abstraction, which over-approximates language of the automaton. In the following paragraphs, we define our method in a formal way.
An event is a tuple $e = (t, s, r, \mathit{time}, m)$ where t is the type of communication protocol (i.e., OPC-UA or REST), s is the identification of the sender, r is the identification of the receiver, *time* is a timestamp, and m is an abstract representation of the sent message described in the previous section. Event calendar c is a list of events $c = (e_1, \ldots, e_n)$. A finite automaton is tuple $\mathcal{A} = (Q, \Sigma, \delta, I, F)$ where Q is a finite set of states, Σ is a finite alphabet, $\delta \subset Q \times \Sigma \times 2^Q$ is a transition relation, $I \subseteq Q$ is a set of initial states, $F \subseteq Q$ is a set of final states. A language $\mathcal{L}$ of automaton $\mathcal{A}$, denoted by $\mathcal{L}(\mathcal{A})$, is a subset of Σ^*. A run ρ of automaton $\mathcal{A}$ is a sequence of states $(q_1, \ldots, q_n)$ such that $\forall 1 \leq i \leq n-1 : \exists a \in \Sigma : q_{i+1} \in \delta(q_i, a)$. A word $w = a_1, \ldots, a_n$ is accepted by the automaton $\mathcal{A}$ iff there is a run $\rho = (q_1, \ldots, q_{n+1})$ of $\mathcal{A}$ such that $\forall 1 \leq i \leq n : q_{i+1} \in \delta(q_i, a_i)$ and $q_{n+1} \in F$. A language $\mathcal{L}_q$ of a state $q \in Q$ is a set $\{w = a_1, \ldots, a_n \mid w \textit{ is accepted by a run } \rho = q_1, \ldots, q_{n+1} \textit{ such that } q_1 \in I \wedge q_{n+1} \in F\}$.
Event calendar $c = (e_1, \ldots, e_n)$ is transformed to a finite automaton $\mathcal{A}_c = (Q^c, \Sigma, \delta^c, I^c, F^c)$ as follows: the set of states is $Q^c = \{q_1, \ldots, q_{n+1}\}$, the alphabet Σ is obtained by transforming each event $e = (t, s, r, \mathit{time}, m)$ to an unique symbol a^e by applying a hashing function over (t, s, r, time), i.e., giving away m, the set of initial states is $I^c = \{q_1\}$ and the set of final states is $F^c = \{q_{n+1}\}$. Finally, $\forall 2 \leq i \leq n+1 : (q_{i-1}, a^{e_i}, \{q_i\})$ is added to δ^c.
In order to create the new scenarios, we need to overapproximate the models. In particular, we propose to use the length abstraction to transform the automaton $\mathcal{A}$ to an abstracted automaton $\mathcal{A}^k$ by merging all states with the same language

with respect to a given length. Formally, a length abstraction over an automaton $\mathcal{A} = (Q, \Sigma, \delta, I, F)$ is an equivalence relation $\alpha^k \subseteq Q \times Q$ such that $(p, p') \in \alpha^k$ iff $\mathcal{L}_p^n = \mathcal{L}_{p'}^n$ where $\mathcal{L}_q^n = \{w' \mid \exists w \in \mathcal{L}_q : w'$ *is a prefix of* $w \wedge$ *length of* w' *is up to* $n\}$. We denote an equivalence class of $q \in Q$ by $[\![q]\!]$. An abstracted automaton $\alpha^k(\mathcal{A}) = (Q_\alpha, \Sigma, \delta_\alpha, I_\alpha, F_\alpha)$ is obtained using α^k in the following way: $Q_\alpha = \{[\![q]\!] \mid q \in Q\}$, for each $q \in \delta(p, a)$ there is $([\![p]\!], a, X) \in \delta_\alpha$ such that $[\![q]\!] \in X$, and finally, $I_\alpha = \{[\![q]\!] \mid q \in I\}$ and $F_\alpha = \{[\![q]\!] \mid q \in F\}$. The length abstraction overaproximates language of the original automata, i.e., $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{A}_\alpha)$ meaning that there may exist a word $w = a_1, \ldots, a_n$ such that $w \in \mathcal{L}(\mathcal{A}_\alpha) \wedge w \notin \mathcal{L}(\mathcal{A})$. Both automata have the same alphabet originally derived from a set of events. Therefore, it is possible to convert the word w to a series of actual messages. Supposing that w is not in the language of the original automaton, we thus obtain a series of events not present in the original system that can be used as a new test case for testing the MES system in a digital twin.

5 Generating Scenario

Finally, we generate a scenario that will be used for the orchestration of the digital twin of the tested system. We iterate over the event calendar, and, for each event, we generate one step in the scenario. Each step consists of sending messages in the digital twin. The concrete messages sent during the orchestration are generated from the abstract representation. By default, we support exact replication of the seen communication, however, we provide also experimental support for, e.g., generating syntactically or semantically similar messages.
We implemented our framework in our tool *Tyrant* [3] which generates scenarios for the orchestrating tool *Cryton*. Cryton uses as an input a configuration for creating the digital twin (i.e., a description of digital twin components) and a scenario generated by Tyrant in the YAML format.
In the following, we will illustrate the transformation of communication logs from a real system to a YAML scenario. We remark we consider a system consisting of an ERP system, MES system, and manufacturing machines and their corresponding terminals used by human operators. Listing 1.1 shows a message between ERP and MES. The message is stored in the file `20211207-125952.xml` which has a timestamp encoded in its name. The message is in XML format and its semantics is that there are 42 items of Material 1 in stock. Listing 1.2 shows a message between a machine and MES. The message was sent one second after the previous one. The message semantics is that the value of the node 0 should be set to 99 in Machine 001. Finally, Listing 1.3 shows a generated scenario consisting of two steps. The first step is executed in (logical) time 0 hours, 0 minutes, 0 seconds: the orchestrator will send a message from ERP to MES using the REST protocol. The message has the XML data attached. The second step is executed one second after the first step: the orchestrator will send another message from MES to Machine 001 using the OPC-UA protocol. The message says that the value of the node with path 0 should be set to 99.
We implemented the proposed framework in the tool called the *Tyrant*. We tested our approach on the captured communication provided by a partner company that offers the MES as their product. We were able to successfully generate valid scenarios for the Cryton tool which then subsequently orchestrated a digital twin using our scenarios. However, our Tyrant is still a prototype and it would need extensive collaboration with the company to deploy it to production.

Listing 1.1. 20211207-125952.xml

```
<DataSource>
 <Data>
   <Name>Material1</Name>
   <Value>42</Value>
 </Data>
</DataSource>
```

Listing 1.2. MES and Machine message

```
SampleDataTime          ; Value ; Name        ; Path
2021-12-07 12:59:53.617 ;  99   ; Machine 001 ; 0
```

Listing 1.3. Generated scenario

```
---
timestamps:
- delta:
    seconds: 0
    minutes: 0
    hours: 0
  steps:
  - type: ERP
    host: ERP
    target: MES
    args:
      xml: |-
        <DataSource>
          <Data>
            <Name>Data1</Name>
            <Value>42</Value>
          </Data>
        </DataSource>
- delta:
    seconds: 1
    minutes: 0
    hours: 0
  steps:
  - type: OPC-UA
    host: MES
    target: Machine 001
    args:
     value: 99
     node: 0
```

6 Conclusion

In this work, we proposed a generic framework for orchestrating the digital twins of distributed systems in manufacturing environments. The main challenges of generating scenarios suitable for orchestrating digital twins are finding suitable models of (1) the communication in the systems, and (2) messages sent during the communications. We proposed to use the simple approach: finite automata for communication and simple abstract representation for messages.
However, in our experience applying the framework in practice requires much more effort. Lots of testing scenarios and components require specific preparation before the orchestration: e.g., setting of the initial database or sending a specific sequence of (hard-coded) messages to prepare the system that are usually not being captured in the communication log. Currently, our tool supports a concrete use case in a concrete manufacturing environment. Hence, extending our solution to a broader class of manufacturing environments is our future work.

Acknowledgement. The work was supported by the project No. 20-07487S of the Czech Science Foundation, by the ECSEL JU under grant agreement No 876852., and by the FIT BUT internal project FIT-S-20-6427.

References

1. MarketsandMarkets Research Pvt. Ltd. ”Industrial Control Systems (ICS) Security Market worth $22.2 billion by 2025”. https://www.marketsandmarkets.com/PressReleases/industrial-control-systems-security-ics.asp, Last accessed 11. Nov 2021.
2. Repository of Cryton. https://gitlabdev.ics.muni.cz/beast-public/cryton, Last accessed 11. Nov 2021.
3. Repository of Tyrant. https://pajda.fit.vutbr.cz/tacr-unis/tyrant, Last accessed 11. Nov 2021.
4. Christopher Ackermann. Recovering views of inter-system interaction behaviors. In *Working Conference on Reverse Engineering 2009, Lille, France*, pages 53–61, 10 2009.
5. Ivan Beschastnikh, Yuriy Brun, Michael D. Ernst, and Arvind Krishnamurthy. Inferring models of concurrent systems from logs of their behavior with csight. In *ICSE ’14, Hyderabad, India*, pages 468–479. ACM, 2014.
6. Paolo Milani Comparetti, Gilbert Wondracek, Christopher Krügel, and Engin Kirda. Prospex: Protocol specification extraction. In *S&P 2009, Oakland, California, USA*, pages 110–125. IEEE Computer Society, 2009.
7. Yating Hsu, Guoqiang Shu, and David Lee. A model-based approach to security flaw detection of network protocol implementations. In *ICNP 2008, Orlando, Florida, USA*, pages 114–123. IEEE Computer Society, 2008.
8. Maikel Leemans and Wil M. P. van der Aalst. Process mining in software systems: Discovering real-life business transactions and process models from distributed systems. In *MODELS, Ottawa, Canada*, pages 44–53. IEEE, 2015.
9. Petr Matoušek, Vojtêch Havlena, and Lukáŝ Holík. Efficient modelling of ICS communication for anomaly detection using probabilistic automata. In *IM 2021, Bordeaux, France*, pages 81–89. IEEE, 2021.
10. Wil M. P. van der Aalst. *Process Mining - Data Science in Action, Second Edition*. Springer, 2016.