Electronic Communications of the EASST
Volume 079 (2020)

Interactive Workshop on the Industrial Application of Verification and Testing ETAPS 2020 Workshop (InterAVT 2020)

Testing Interconnected Systems with Behavior Mining

Alexander Schieweck, Tiziana Margaria

9 pages

Guest Editors: Stylianos Basagiannis, Goetz Botterweck, Anila Mjeda
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Testing Interconnected Systems with Behavior Mining

Alexander Schieweck[1], **Tiziana Margaria**[2]

[1] alexander.schieweck@ul.ie [2] tiziana.margaria@ul.ie
Department of Computer Science and Information Systems
University of Limerick, Ireland
Lero - The SFI Research Centre for Software
Confirm - The SFI Research Centre for Smart Manufacturing

Abstract: Modern software applications rely not only on a complex stack of technologies, but are more and more dependent on and connected to third party interfaces, Internet of Things devices and Industry 4.0 machines. One approach to tackle this complexity is Model Driven Design with custom interactions. But it is then still necessary to test the whole system to ensure that all parts work together as intended. This paper looks at the possibility of using Active Automata Learning as a systematic way to test interconnected systems.

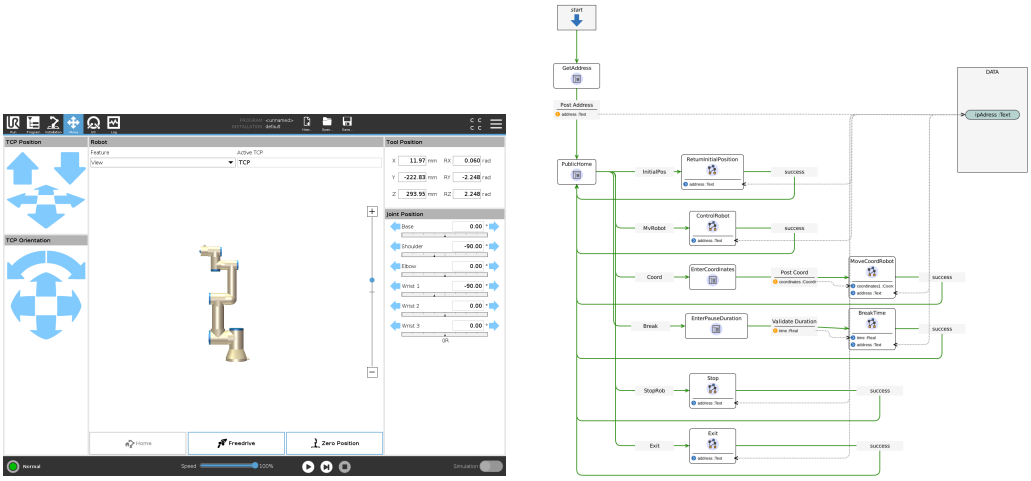
Keywords: Formal Methods, Active Automata Learning, Model Driven Design, Industry 4.0, Smart Manufacturing

1 Introduction

Model Driven Design (MDD) and Low-Code application development environments are getting increasingly popular. This is also true in the field of mechanical engineering, where new machines become more equipped with more sensors, computing power and connectivity. However, mechanical engineers are not specialized in programming, so instead of solely relying on software engineers they look for alternatives to help them configure and manage new machines on their own, reducing the programming needs. Most MDD tools are multi-purpose environments and need custom extensions to accommodate the needs of mechanical engineers. This raises the question of how to test and verify those combined systems.

The Confirm *Digital Thread* prototype will be used as a small example for a complex MDD based application in the smart manufacturing context. This prototype is a web application to remotely control collaborative robots (cobots) by Universal Robots (UR) (see Figure 1a). Cobots are equipped with special sensors and can detect if something is in their way. This ability allows them to operate without special work cages, opening the possibility of collaborative work with humans. UR, the first company to produce such robots, offers now four models: UR3, UR5, UR10 and UR16. The number indicates the maximum payload in kg of each model. The models grow in size and weight accordingly, but the core design and concepts are very similar for all models. They use for example the same API, which allows custom programs to be interchangeable [MS19].

This paper gives an overview of MDD and Active Automata Learning techniques in (Section 2). It then explains the set up for experiments of the Confirm Digital Thread prototype in



(a) Simulator by Universal Robots

(b) Model of the App Workflow in DIME

Figure 1

conjunction with the robot simulator (Section 3), followed by the results in Section 4. Lastly, it discusses future challenges (Section 5).

2 Concepts and Technologies

This section introduces the MDD tool called *DIME* that we used to create the applications and gives a short introduction in Active Automata Learning.

2.1 Model Driven Design

The Model Driven Design (MDD) approach breaks with the paradigm that everything needs to be written in code and puts models at the center of a software development project. Those models can be textual or graphical and help the developer to describe what the software should be doing, without worrying about the how.

DIME is an Integrated Modeling Environment, i.e. a model driven design tool, specialized for web applications. *DIME* offers various Graphical Domain Specific Languages (GDSLs) to describe all layers of a modern web applications, e.g. a data model GDSL, a process model GDSL to describe the business logic (see Figure 1b) and a GDSL for the GUI front end, similar to a "What you see is what you get" editor. For domains not covered by DIME in the default configuration, it is possible to extend its capabilities with new GDSLs: new libraries of Service Independent Blocks (SIBs) for the back or front end. The UR Control application makes use of this functionality by introducing robotics DSLs used to create a plugin that communicates with the UR robots [SMN+06] [BFK+16].

DIME itself is created using the *Cinco* meta-modeling environment [NLKS18]. In fact *DIME* is right now the most sophisticated *Cinco*-product. *Cinco* allows the creation of Eclipse based

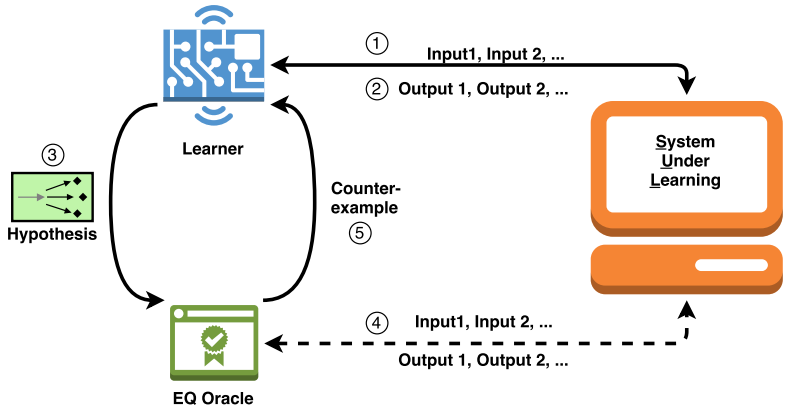


Figure 2: Overview of the Active Automata Learning Loop

specialized GDSL tools without a deeper knowledge about the various Eclipse graphical tooling projects.

2.2 Active Automata Learning

Active Automata Learning (AAL) uses observations to infer models of a system's internal states and behavior. In the case of reactive systems like web applications, those models are often Mealy machines.

Definition 1 (Mealy Machine) A *Mealy Machine* is defined as a tuple $(Q, q_0, \Sigma, \Lambda, \delta, \lambda)$, where Q is a finite set of states, $q_0 \in Q$ is the initial state, Σ is a finite set of input symbols, i.e. the input alphabet, Λ is a finite set of output symbols, i.e. the output alphabet, $\delta : Q \times \Sigma \to Q$ is the transition function, and $\lambda : Q \times \Sigma \to \Lambda$ is the output function.

The core learning loop of Active Automata Learning is illustrated in Figure 2 and follows the principles first described by Angluin [Ang87]. The learner interacts with the System Under Learning (SUL) via testing and observes its behavior. Those interactions are called Membership Queries. Once the learner thinks it has seen enough behavior, it creates a hypothesis model of the internal states of the system and passes it to the Equivalence (EQ) Oracle. The EQ Oracle then tells the learner if the hypothesis is correct or not. In an ideal world the EQ Oracle would have perfect knowledge and could decide this question directly. In the real world, instead, the EQ Oracle sends Equivalence Queries to the SUL in order to find a counter example to the hypothesis. If a counter example is found, the example is passed back to the learner to refine the hypothesis. If no counter example is found through the deployed counter-example search strategies in reasonable time, it is assumed that the hypothesis is correct.

Every membership or equivalence query needs to start with the same prerequisites, so a reset mechanism of the SUL is needed too.

LearnLib[1] is a state of the art open source framework for AAL, which offers a wide set of

[1] https://learnlib.de

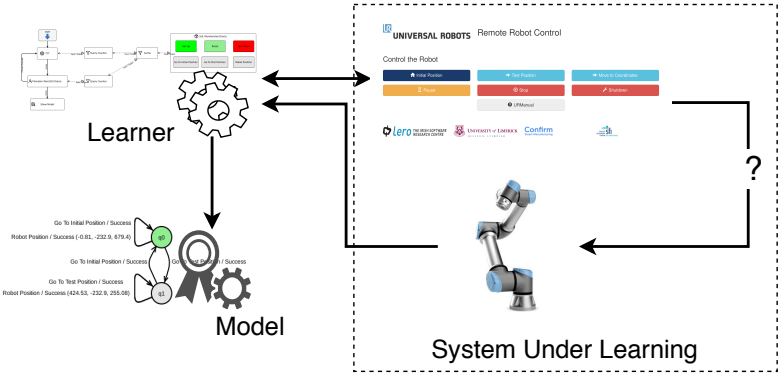


Figure 3: The Setup of the learning experiment (AAL with LearnLib Studio

algorithms, counter examples search strategies and infrastructure components in Java [IHS15] [HS18].

Many tools have been designed to create customized learn experiments utilizing the *LearnLib*.

The tool Active Automata Learning Experience (ALEX) is built upon *LearnLib* and allows a no-code way to learn web applications and even to mix them with REST APIs. *ALEX* is itself a web application. It offers a comfortable GUI to describe the interactions with a web application or a RESTful API. The learning can be parameterized, but the overall learning loop is fixed. [BSI+16] [BSSH17]. Because the UR robot itself does not offer a REST API, ALEX is not applicable and a different solution is needed.

We use LearnLib Studio[2], a specialized *Cinco*-product for defining LearnLib experiments through a custom MDD editor.

3 Experiment Set Up

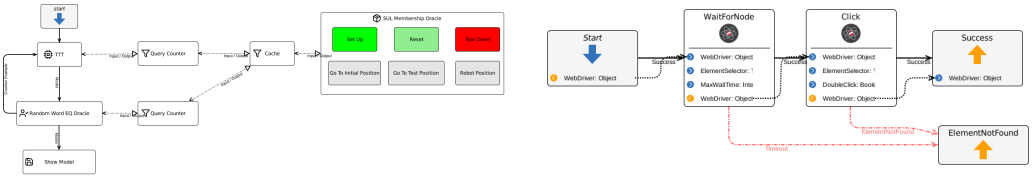
In our Digital Thread prototype, the UR Remote Control Web Application and the robot, here a UR simulator, constitute the SUL (see Figure 3). We wish to automatically extract a Digital Twin of the SUL in order to find out whether the web application interacts with the robot in the expected way. Concretely, we wish to find out if the native SIB libraries of the UR DSL are used in the correct way, and if the controller application is properly designed, i.e. it is doing exclusively what it is expected to do.

While this experiment can show the existence of a fault, it would not be able to determine whether the fault is in the implementation of the native SIBs (code) or if the SIBs are OK but not properly used in the process model (application logic).

3.1 Preexisting Parts

Instead of connecting a real robot to the system we used of the simulator provided by Universal Robots in a Virtual Box. As the real robot is also connected with an IP Address, the Virtual

[2] https://github.com/learnlib/learnlib-studio



(a) Description of the Learning Experiment as a *Learn Experiment Model*

(b) *Symbol Definition* of the Click on Initial Position Button

Figure 4: Examples of the Models in *LearnLib Studio*

Box helps to create a realistic scenario. Within this setup the simulator and the robot are interchangeable, which was confirmed through tests. We parameterized the simulator for the UR 3 model, but the UR scripting language and ways to communicate with the robot are identical for the whole UR family. Using the simulator also allowed us to speed up the robot a little bit, bringing down the overall time for our learning experiment.

The UR Control Web Applications runs in a Docker environment and can be used without *DIME*. Everything was executed on a single local machine.

3.2 The Learning Set Up

The learning experiment was described graphically using *LearnLib Studio*'s models. some models can be seen in Figure 4.

The experiment uses the TTT Algorithm and a random word counterexample search, which is parametrized with 20 random words with a length between 5 and 10. The set up can be seen in Figure 4a.

A list of created symbols in described in Table 1. The Symbols 'Go To Initial Position', 'Go to Test Position' and 'Robot Coordinates' are the learning alphabet. Their names are the *input alphabet*. The *output alphabet* consists of their possible outputs, i.e. 'Success', 'Success (X, Y, Z)', with the actual robot coordinates at time of calling, and the additional symbol 'ElementNotFound'. The use of the robot coordinates in this set up allows to observe the robot coordinates in the learning independently and correlate them to interactions with the web application.

Beside those input symbols there are three helper symbols (see Table 2) to manage the experiment, e.g. to ensure a reliable reset.

The symbols use a custom SIB library to interact with the web application and another SIB library to interact with the robot directly. The SIB library to interact with the robot was newly created. The one to interact with the web application is standard (buttons, numeric fields) and preexisted.

During the first learning experiment there were some issues with the network socket of the robot: even the sped-up robot (in the simulator) could not keep up with the amount of different commands send to it. This was solved by introducing artificial wait times.

Table 1: Overview of the Learning Alphabet

Name	Outputs	Description
Connect To Robot	Success, ElementNotFound	Tries to to enter the IP address and click 'Connect'.
Go to {Initial, Test} Position		Tries to click the button {initial, test} position button, which should move the robot accordingly.
Go to Coordinate Input		Tries to click the button in the web application to navigate to the coordinate input page.
Send Coordinates		Tries to enter custom coordinates and click the move button on the coordinate input page.
Cancel Coordinate Input		Tries to click the cancel button on the coordinate input page.
Robot Coordinates	Success (X, Y, Z)	Connects to the robot and receives the current robot coordinates, which are part of the output.

Table 2: Overview of the Helper Symbols

Name	Description
Set Up	Starts the web browser. It is called only once at the beginning of the learn experiment.
Reset	Opens the web app in a fresh environment. Moves the robot to the initial position. Called before every query.
Tear Down	Closes the web browser. It is only called once at the end of the learning experiment.

4 Results

The final Mealy machine shown in Figure 5 is the behavioral Digital Twin of the UR Controller Web Application.

The state q_0 on the left shaded in green is the initial state. The very first page of the web application asks for the IP address of the robot and is otherwise only reachable through a reload of the application. This behavior is evident in the Digital Twin model's state q_0: it is the initial state and only allows to move ahead with 'Connect To Robot'.

Upon closer inspection one notices that the final model is a product of the possible states of the web application, i.e. main 'button' page and coordinate input page, and the three possible robot positions from the app, i.e. initial position, test position, and custom coordinates. In the states q_0, q_1 and q_3 (dashed oval) the robot is in the initial position. The states q_2 and q_5 (solid oval) represent the robot in the test position. And in states q_4 and q_6 (dotted oval) the robot is in the custom coordinates position. Between those areas there are only the 'Go to Initial Position', 'Go to Test Position' and 'Send Coordinates' transitions, and they lead always successfully to the according state.

In each robot position, one state $q1$, $q2$ and $q4$ (blue squares) represents the main button page of the website. Furthermore, these areas include the states $q3$, $q5$ and $q6$ (orange triangles) representing the coordinate input page. Between pairs of those states there are only the transitions with 'Go to Coordinate Input' and 'Cancel Coordinate Input': they are present and successful. The only exception is the 'Send Coordinates' transition between $q6$ and $q4$, which can be explained as this is the reflexive edge within the robot position area.

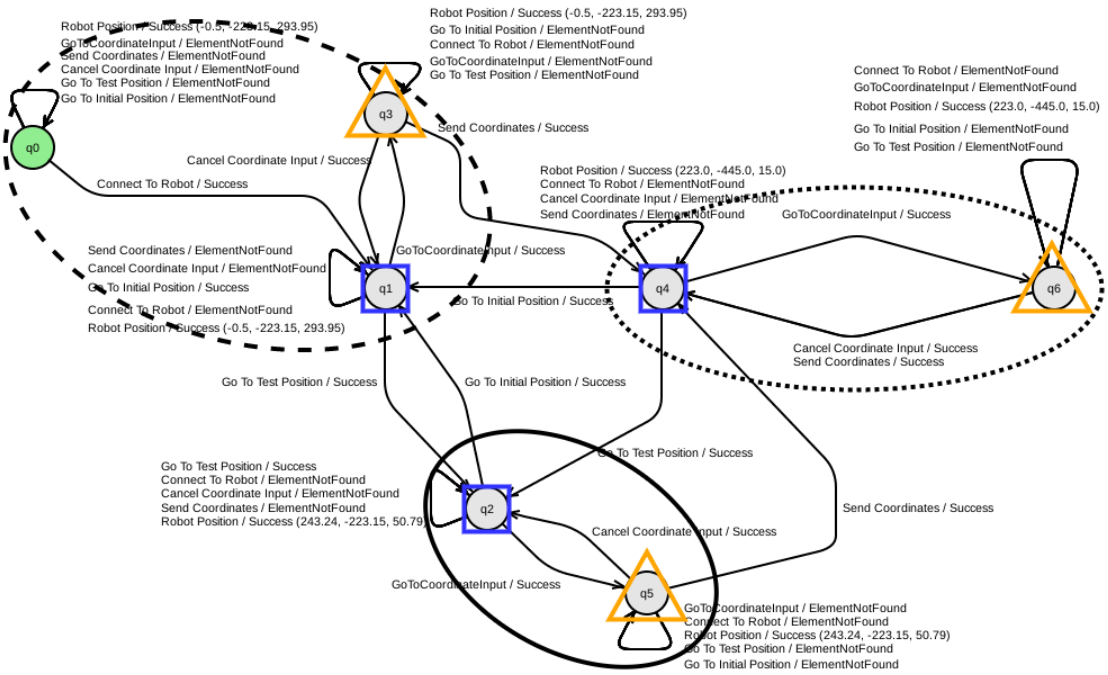


Figure 5: The Final Model: A Digital Twin Obtained by AAL

Overall the final model can be seen as a product between the states of the web application and the three chosen robot positions, with a network of correct transitions according to the ”good machine” behavior we expected.

The learning experiment was run on a Dell XPS 15 9560 (Intel Core i7-7700HQ, 32GB RAM, Manjaro Linux) and took 80 min. It took four iterations of the learning loop. The learner asked 218 Membership Queries and the counter example strategy issued 34 Equivalence Queries.

5 Conclusion & Challenges

In this paper we showed how to extend the classical class of System Under Learning for Active Automata Learning experiments to more complex and connected scenarios like robotics. While this is a small example, the capability to retrofit Digital Twin models to the behavior of cyberphysical systems can potentially pave the way to capturing the behavior of legacy (control) applications in smart manufacturing by means of models amenable to formal analysis and model based testing.

Future challenges might evolve around the questions of more complex behaviors, e.g., how to handle complex paths of the robot arm, combined with actual work of the robot. Another challenge is the scalability of this approach to multiple robots or machines, who are doing different jobs. The AAL techniques have been shown to scale very well - here the time limitation was due to the slow simulator response.

Through Confirm and its (future) outreach partners we will hopefully have the possibility to

extend the Confirm Digital Thread prototype to different kinds of robots, i.e. bigger industrial robots, typically caged, as well as smaller robots used by makers as in the FabLab Limerick. An extended prototype and a working Digital Thread library of models for various robot families will open possibilities to investigate those issues further.

Acknowledgements: This work was supported, in part, by Science Foundation Ireland grant 16/RC/3918 to Confirm, the SFI Research Centre for Smart Manufacturing (www.confirm.ie) and 13/RC/2094 to Lero - The SFI Research Centre for Software (www.lero.ie).

Bibliography

[Ang87] D. Angluin. Learning Regular Sets from Queries and Counterexamples. *Inf. Comput.* 75(2):87–106, 1987.
doi:10.1016/0890-5401(87)90052-6

[BFK+16] S. Boßelmann, M. Frohme, D. Kopetzki, M. Lybecait, S. Naujokat, J. Neubauer, D. Wirkner, P. Zweihoff, B. Steffen. DIME: A Programming-Less Modeling Environment for Web Applications. In Margaria and Steffen (eds.), *ISoLA 2016, Proceedings, Part II*. LNCS 9953, pp. 809–832. 2016.
doi:10.1007/978-3-319-47169-3_60

[BSI+16] A. Bainczyk, A. Schieweck, M. Isberner, T. Margaria, J. Neubauer, B. Steffen. ALEX: Mixed-Mode Learning of Web Applications at Ease. In Margaria and Steffen (eds.), *ISoLA 2016, Proceedings, Part II*. LNCS 9953, pp. 655–671. 2016.
doi:10.1007/978-3-319-47169-3_51

[BSSH17] A. Bainczyk, A. Schieweck, B. Steffen, F. Howar. Model-Based Testing Without Models: The TodoMVC Case Study. In Katoen et al. (eds.), *ModelEd, TestEd, TrustEd - Essays Dedicated to Ed Brinksma on the Occasion of His 60th Birthday*. LNCS 10500, pp. 125–144. Springer, 2017.
doi:10.1007/978-3-319-68270-9_7

[HS18] F. Howar, B. Steffen. Active Automata Learning in Practice - An Annotated Bibliography of the Years 2011 to 2016. In Bennaceur et al. (eds.), *Machine Learning for Dynamic Software Analysis: Potentials and Limits - International Dagstuhl Seminar 16172, 2016, Revised Papers*. LNCS 11026, pp. 123–148. Springer, 2018.
doi:10.1007/978-3-319-96562-8_5

[IHS15] M. Isberner, F. Howar, B. Steffen. The Open-Source LearnLib. In Kroening and Păsăreanu (eds.), *Computer Aided Verification*. Pp. 487–495. Springer International Publishing, Cham, 2015.

[MS19] T. Margaria, A. Schieweck. The Digital Thread in Industry 4.0. In Ahrendt and Tarifa (eds.), *IFM 2019, Proceedings*. LNCS 11918, pp. 3–24. Springer, 2019.
doi:10.1007/978-3-030-34968-4_1

[NLKS18] S. Naujokat, M. Lybecait, D. Kopetzki, B. Steffen. CINCO: a simplicity-driven approach to full generation of domain-specific graphical modeling tools. *STTT* 20(3):327–354, 2018.
doi:10.1007/s10009-017-0453-6

[SMN+06] B. Steffen, T. Margaria, R. Nagel, S. Jörges, C. Kubczak. Model-Driven Development with the jABC. In Bin et al. (eds.), *HVC 2006, Revised Selected Papers*. LNCS 4383, pp. 92–108. Springer, 2006.
doi:10.1007/978-3-540-70889-6_7