Towards Engineering Digital Twins by Active Behaviour Mining

Tiziana Margaria[1,2,3](✉) and Alexander Schieweck[1,2,3](✉)

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

Abstract. In the context of Confirm, the Irish Research Centre on Smart Manufacturing, field demonstrators are used to show new techniques to industrial partners, various kinds of students, and the general public alike. Considering the robotics demonstrator for the Digital Thread concept used in Confirm, which is a small cyberphysical system based on the UR3 cobot and a web controller for it, we apply Active Automata Learning in order to obtain a Digital Twin for it. Behavior mining done in this fashion is nowadays uncommon, but it has various advantages over, e.g., models obtained with popular AI techniques in that the AAL models are accurate deterministic behavioural explanations for the system behaviour at the chosen level of abstraction, and they may be further amenable to formal verification, e.g., by model checking, in order to establish properties of interest.

This extension has the effect of showcasing the Digital Twin concept, the AAL technique, the use of model checking, and the importance of working with formal models that are amenable to these technologies. We then reflect on the nature of the models and their uses and meaning, from the point of view of the comments and questions we receive in the demonstrations. We also consider the use of a feature-based approach to modelling the systems and their interactions, which is a further aspect for which the demonstrator could be used, with a special attention to the aspects of this work, like AAL and the feature based and feature interaction research, that connect directly with the collaboration with and the research of Bengt Jonsson.

Keywords: Formal methods · Active automata learning · Model driven design · Digital twin · Digital thread · Industry 4.0 · Smart manufacturing

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

1 Digital Twins and CPSs

Engineering adequate Digital Twins for Cyber-Physical Systems is a complex, multidimensional challenge. While there are many definitions of what is a digital twin, we choose to refer to the recent, quite realistic and encompassing definition is by Ashtari et al. [50]: "*The Digital Twin is a virtual representation of a physical asset in a Cyber-Physical Production System (CPPS), capable of mirroring its static and dynamic characteristics. It contains and maps various models of a physical asset, of which some are executable, called simulation models. But not all models are executable, therefore the Digital Twin is more than just a simulation of a physical asset. Within this context, an asset can be an entity that already exists in the real world or can be a representation of a future entity that will be constructed.*"

In 2019 the Gartner group [43] listed digital twins in the top 10 strategic technology trends, next to blockchain, artificial intelligence, empowered edge, privacy and ethics, quantum computing, immersive experiences, augmented analytics, and autonomous things. While some of these technologies are evergreens, like quantum computing and the rather generic "autonomous things", digital twins are a new entry, and they start to play a role, at least conceptually, well beyond the smart manufacturing domain from which they originate. For example, one starts to hear about initiatives to co-create digital twins for (cancer) patients [1]. The digital twins of the future will be patient-tailored models that:

– Can be used to evaluate potential preventative and/or therapeutic plans,
– Incorporate information across length and time scales,
– Continually integrate new data and knowledge,
– Help clinicians and patients understand the risks and benefits of a particular treatment plan that best meets the patient's objectives.

Digital Twins are used as well for and within supply chains [21], in particular in connection with supply chain disruption for manufacturing, as the last year has acutely manifested. Most of these Digital Twin variants concern simulation models that arise distinctly from the physical thing, or more realistically, the real-world system, they model.

Cast in new words, a digital twin is an instance-level model of an entity (physical or not), that, as IBM's Chris O'Connor puts it, is *"simple, but detailed"*[1]. A digital twin implies a strong notion of **adequacy for purpose** (otherwise it is not a "twin", lacking sufficient sameness), a **context-dependency** of the purpose (there can be different digital twins for the same entity if this entity serves different purposes, and these differences of purpose matter, inducing difference of context), and it is required to have an ability to **support and guide the design, build and operation** phases of the entity it represents. It is therefore descriptive (like a design model), behavioural (as it must encompass what the physical twin can do), and predictive: during operations it must be able to

[1] https://youtu.be/RaOejcczPas.

predict maintenance needs as well as out-of-order behaviours, and serve as a baseline to figure out their prevention and repair.

Most frequently we see digital twins of some physical entities, like manufacturing machines and products that are more generally abstracted as Cyber-Physical Systems (CPSs) models. Most recently they started to include also any kind of Internet of Things (IoT) and Industrial Internet of Things (IIoT) devices.

At design time, the digital twin models serve the main purpose of increasing the expected dependability of their physical counterpart. At build time they serve to assess and monitor the faithfulness of the production processes for the physical twin. During operation, once the physical twin has been produced and installed, they find use to monitor the dependability of the products, for each individual piece with its individual characteristics, aging, and anomalies.

However, many engineers still associate the concept of digital twin to a quantitative, mathematical simulation based model, that allows (mostly mechanical) engineers to ask what-if questions that inform design, usage, and evolution decisions for a mechanical object or mechanical forming process [12]. Statistical models, Finite Element Analysis (FEA) as the simulation of a physical phenomenon using a numerical mathematical technique referred to as the Finite Element Method (FEM) are the most frequent types of models, and in some application domains they are still nearly synonyms of Digital Twins. The awareness that software plays a role in the "fullness" of modern devices, that the behavior of the software may go beyond a pure controller of the physical part, that there is inherently a heterogeneity, and thus an integration problem with discrete, finite state machine-like models are still not obvious today.

The most modern intuitive connection is with models derived by Machine Learning (ML) from data acquired from the physical twin, retrofitting the device (a black box) with models that are based on statistics, and incomplete knowledge and descriptions. They are therefore at best approximations of the real behaviours, thus themselves a black box model, and a blurry one.

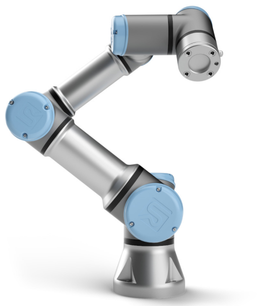
What we are trying to achieve, on the contrary, is the systematic, and possibly automatic, production of **behavioural models** for real devices, that describe precisely the observed behaviours and are congruent to such behaviours, thus can be used as faithful predictors, like a sosia, and are able to explain the predicted and the observed behaviour on an execution by execution basis, i.e., use case by use case, test run by test run.

The challenge towards this scenario is, how can one systematically enable the well-founded engineering of such digital twins for dependable CPSs?

We are going use the simple Confirm *Digital Thread* prototype consisting of 1) a commercial Universal Robots cobot and 2) a Web-based remote controller application as a small example for a concrete CPS of industrial relevance. This mini-system of systems case study includes a Cyber part (the web application), a Physical part (the robot), and a communication system, which is here the internet plus a TCP socket connection to the robot. Although simple, this system exhibits the essential traits of many CPSs, and it is simple and small enough (i.e., physically transportable) that it has been repeatedly used for many teaching and demonstration purposes.

We will show how already the simple Active Automata Learning for Mealy machines can be an effective technique to retrofit existing CPS systems with models that satisfy these characteristics. We then reflect on several aspects of the system, the model, the learning, and how the model, behavior and properties are expressed, that can be interesting from a research and practical point of view.

In the following, Sect. 2 presents the case study and Sect. 3 provides an overview of MDD and Active Automata Learning techniques. We then explain the setup for experiments of the Confirm Digital Thread prototype in conjunction with the robot simulator (Sect. 4), followed by a discussion of the learning results in Sect. 5 and an application of CTL model checking for property checking on the learned model in Sect. 1. In Sect. 7 we discuss the insights gained so far, the lessons learned and our reflections along various perspectives of past experience, collaboration with Bengt Jonsson and future work. Lastly, Sect. 8 concludes the paper.


(a) Universal Robots' UR3

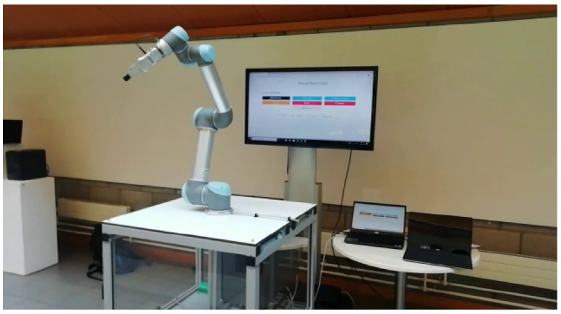

(b) Demonstrator Setup with UR 5

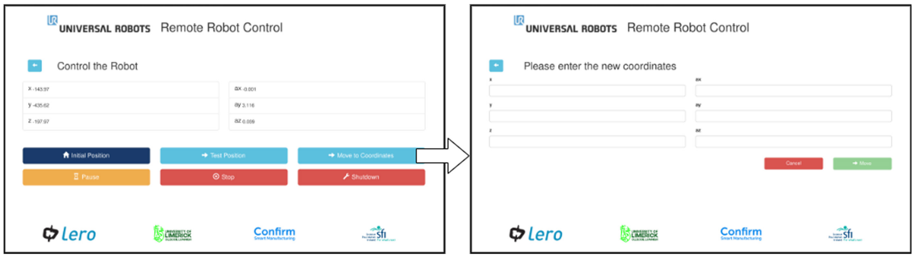
Fig. 1. The physical system

2 The Case Study: XMDD Steers a Cobot

The Confirm *Digital Thread* demonstrator, introduced in [32], showcases a MDD-based application in the smart manufacturing context. It is a handy example that brings together two worlds still culturally very distant and effectively disjoint: commercial collaborative robots and advanced Model Driven Development.

This portable demonstrator consists of a collaborative robot (cobot) produced by Universal Robots (UR) together with a web application designed to remotely control such cobots. In the world of robotics, this is a very small installation, easy to transport and set up for demonstration and outreach purposes. As shown in Fig. 1b, the UR5 is mounted on a portable rolling table, and the large display shows the Web Application, running on the laptop at the right.

The UR product line consists of flexible 7 joints robotic arms that can be equipped with a wide variety of mountable devices like a grip arm, a camera, and various sensors and actuators. It is widely customizable and retargetable for different applications with little effort and expense, by retooling and reprogramming. Cobots are particularly safe because they are equipped with special sensors to detect whether 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 four models: UR3 (see Fig. 1a), UR5 (see Fig. 1b), 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, like the joints, degrees of freedom and skills, are very similar for all models. They use for example the same API, which allows custom programs to be interchangeable [32].


UNIVERSAL ROBOTS Remote Robot Control

Control the Robot

X -143.97	ax -0.001
y -435.62	ay 3.116
Z -197.97	az 0.039

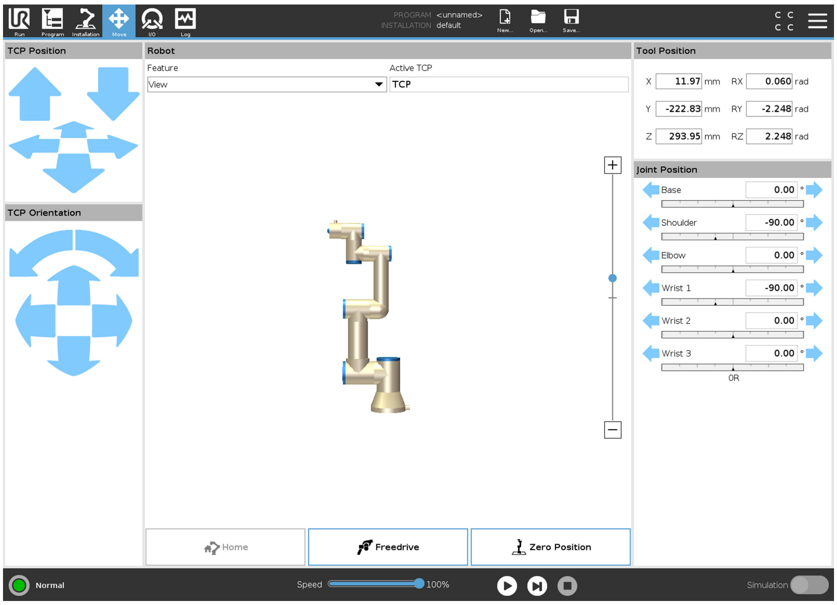
UNIVERSAL ROBOTS Remote Robot Control

Please enter the new coordinates

x, y, z, ax, ay, az

Fig. 2. The web application: The controller main page (left). Clicking the 'Move to Coordinates' button leads to the coordinates input page (right)

While this describes the Physical side of the CPS demonstrator, the user-level interactable view of the Cyber component is shown in Fig. 2. The main page of the controller offers a set of six predisposed operations: `Initial position` (dark blue button) `Pause` (yellow button), `Test position` and `Move to coordinates` (light blue buttons), `Stop` and `Shutdown` (red buttons). Each of them can be launched by clicking the corresponding GUI button. The `Initial position` and `Test position` skills are fully predefined: clicking the respective button brings the robot to a fixed position. For example, the `Initial position` button leads to the balanced vertical "zero" position shown in Fig. 3. The `Move to coordinates` button, however, allows a remote configuration of the robot: it



PROGRAM <unnamed>
INSTALLATION default

Feature: View
Active TCP: TCP

X	11.97	mm	RX	0.060	rad
Y	-222.83	mm	RY	-2.248	rad
Z	293.95	mm	RZ	2.248	rad

Base	0.00	°
Shoulder	-90.00	°
Elbow	0.00	°
Wrist 1	-90.00	°
Wrist 2	0.00	°
Wrist 3	0.00	°

0R

Normal
Speed 100%

Fig. 3. Interactive simulator by universal robots

leads to a second web page (Fig. 2(right)), where a mask allows the user to input target coordinates of the tip of the arm. The choice of these skills is intentional: it is a minimum set of skills that covers the categories *Home*, *Move*, *Timing* and *Manage*, associated with the colour of the button as well as with a small symbol near the skill name on the button. This is, in effect, a minimal Domain Specific Language in the application domain of the robots, what we would call an A-DSL for Application-specific DSL. The colours and symbols associated to the individual skills expose the internal structure of the A-DSL, which has a taxonomic structure.

Traditionally, the controller is programmed and tested either on-site, using the tablet physically tethered to the machine, or by means of a simulator software that behaves like the UR equipment, so that the program tested on the simulator can be then uploaded with confidence to the cobot. The simulator provided by Universal Robots shown in Fig. 3 covers the entire family of cobots. It can be installed on a Linux system or used in a virtual machine via a provided virtual machine image. For the purpose of this paper we choose the latter option, as this adds a layer of separation between the now virtual robot and the rest of our technology stack, similar to how a real cobot would be separated from the other technologies. The simulator also offers the option to change timing parameters. This allows the simulated robot to move like a real machine, but at accelerated speed. This feature is going to be very useful during the automata learning campaign that leads to the Digital Twin of this system.

The simulator is seen by many in robotics and manufacturing as "the" model, and effectively as the in-silico reification, of the Digital Twin in terms of software. However, there is much more that models can do to support the deeper understanding of a system.

With this small demonstrator we intend to showcase the use of models at many levels: to design and validate the controller, but also to represent in a different way the "real essence" of the Digital Twin, not just for the robot but also for the entire CPS, including the controller. To do so, we will use some XMDD concepts and technologies.

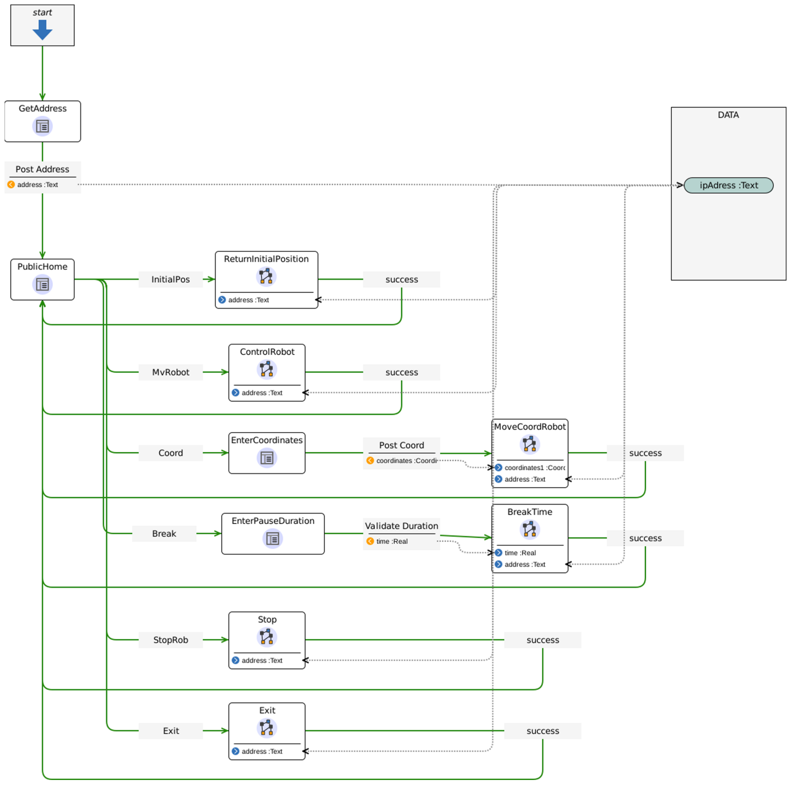


Fig. 4. Process model of the remote controller: App's main workflow in DIME

3 XMDD Concepts and Technologies

We adopt the eXtreme MDD paradigm of [33,34], and use the DIME [7] tool and platform first to model, and then to code-generate and deploy the Web application that controls the cobot. In this section we briefly introduce XMDD and provide a short introduction in Active Automata Learning, the approach we use to generate the Digital Twin. We assume that model checking is known.

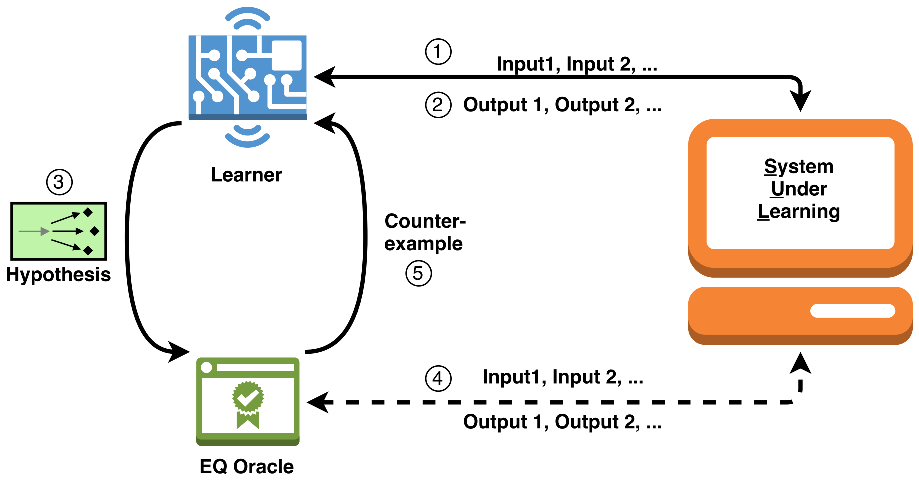


Fig. 5. Overview of the active automata learning loop

3.1 XMDD in DIME

The Model Driven Design (MDD) approach breaks with the paradigm that everything needs to be written in native code and puts instead models at the center of a software development project. Those models can be textual or graphical, they help the developer to describe what the software should be doing. Depending on the choice of models and modelling languages, they have different levels of expressiveness, automatic model analysis and transformation support. In most approaches, a key advantage is that they help delegate the worry about the "how" to a separate design granularity, and often to a separate professional profile [11,26–28].

From a generic MDD point of view, DIME is an Integrated Modelling Environment, i.e. a model driven design tool, specialized for the design, development and deployment of web applications. DIME is open source, provides flexibility, ease of extension, supports high-assurance software quality, agility, a service-oriented approach, and also containerization. For the specific low-code support, its model-driven approach is based on Domain Specific Languages (DSLs) at two levels:

– Language DSLs, as a mechanism to design and implement the application design environment itself, i.e., the Integrated Modeling Environment (IME),
– and a number of Application domain DSLs, at application design time. We want to use Native DSLs as the means to integrate and expose collections of capabilities offered by end devices and other sources of functionalities to the application designers, and Process DSLs (see Fig. 4) as the means to foster reuse of medium and large grained business logic across applications.

DIME's DSLs cover all layers of a modern web applications, e.g. the data model, the process models to describe the business logic, and the GUI front end in a way similar to a "What you see is what you get" editor. The Native DSLs extend its capabilities with new GDLSs: 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 [7,49].

DIME is itself created using the Cinco meta-modeling environment [37], and it is in fact the most sophisticated Cinco-product. Cinco allows the creation of further Eclipse based specialized editors for Language DSL tools without a deeper knowledge about the various Eclipse graphical tooling projects.

3.2 Active Automata Learning

Active Automata Learning (AAL) [2] 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,* $\delta : Q \times \Sigma \rightarrow Q$ *is the transition function, and* $\lambda : Q \times \Sigma \rightarrow \Lambda$ *is the output function.*

The core Active Automata Learning process is illustrated in Fig. 5. The learning algorithm, called the learner, interacts with the System Under Learning (SUL) via testing and observes its behavior. Those interactions are called Membership Queries. In a Membership Query, the learner sends inputs to the SUL, collects the corresponding observed outputs, and collects the resulting input/output behaviour traces, producing a hypothesis model of the internal states of the system. Once the learner reaches a point where it has seen enough behaviour, along a predefined notion of "enough", it passes the current hypothesis model to the Equivalence (EQ) Oracle. In an ideal world, the EQ Oracle would have perfect knowledge of the SUL and could decide this question directly. In the real world this is impossible: instead, the EQ Oracle applies another set of criteria to the model, and tells the learner whether the current hypothesis is correct, i.e. satisfies all those criteria, or not. If one or more counterexamples are found, they are passed back to the learner, which starts a new MQ campaign based on the new insights. This leads to successive evidence-based refinement cycles of the hypothesis model. When the deployed counter-example search strategies do not find any counter example anymore in reasonable time, the current hypothesis model is assumed to be correct and the learning process terminates with that learned model.

It is important to note that every membership and equivalence query needs to start with the same prerequisites, so a reset mechanism of the SUL to the same initial state is needed too.

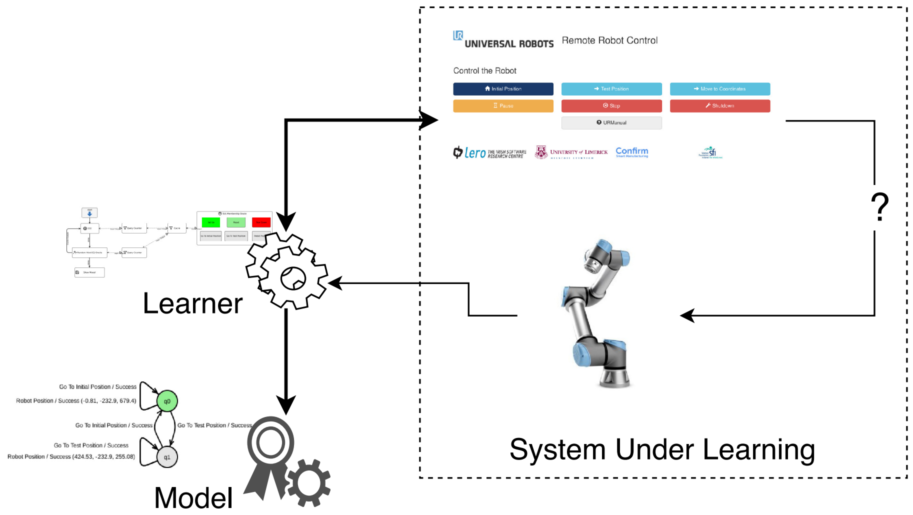

UNIVERSAL ROBOTS
Remote Robot Control
Control the Robot
Initial Position
Test Position
Move to Coordinates
Pause
Stop
Shutdown
URManual
?
Learner
System Under Learning
Go To Initial Position / Success
Robot Position / Success (-0.81, -232.9, 679.4)
q0
Go To Initial Position / Success
Go To Test Position / Success
Go To Test Position / Success
Robot Position / Success (424.53, -232.9, 255.08)
q1
Model

Fig. 6. The setup of the learning experiment: AAL with LearnLib Studio

LearnLib[2] [19,40] is a state of the art open source framework for AAL, which offers a wide set of algorithms, counter examples search strategies and infrastructure components in Java. Many tools have been designed to create customized learn experiments utilizing the LearnLib.

The Active Automata Learning Experience (ALEX) tool 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 process is fixed [19,40]. Because the UR robot itself does not offer a REST API, ALEX is unfortunately not applicable to this case. We use instead LearnLib Studio[3], a specialized *Cinco*-product for defining LearnLib experiments through a custom MDD editor.

4 Automata Learning Experiments: Set Up with Learnlib Studio

In our Digital Thread prototype, the UR Remote Control Web Application and the robot, here a UR simulator, constitute the SUL (see Fig. 6). 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 expected way, e.g. following the correct protocol, and if the controller application

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

is properly designed, i.e., it is doing exclusively what it is expected to do, in terms of sewuences of actions and reaction s to unexpected inputs or commands. We have the SUL as entire CPS on the right, and on the left we use LearnLib Studio as the Learner, extracting a model that is the Digital Twin of the SUL.

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

In the following we recall the preexisting components (Sect. 4.1), then we describe the set up of the learning experiment (Sect. 4.2). Section 4.3 describes in detail the alphabets we used, and finally Sect. 4.4 reports on performance issues and their resolution.

4.1 Preexisting Components

Instead of connecting a real robot to the system, we use the simulator provided by Universal Robots in a Virtual Box. As the robot is in reality also connected with an IP Address, the Virtual Box helps to create a realistic scenario. Within this setup, the simulator and the robot are interchangeable, as was confirmed through tests. We parameterized the simulator with the data and coordinates for the UR3 model, but the UR scripting language and the communication with the robot are identical for the whole UR family. To cover other models, which have different dimensions of the arm segments, the specific coordinates for predefined positions would need to be changed to those for the specific UR model. Using the simulator also allowed us to speed up the robot responsiveness, significantly reducing the overall time for our learning experiment. The robot is in fact mechanically quite slow. We used instead a simulator setting with a near immediate response to commands, preserving the execution traces but much faster than the real system. As we are not examining timed behaviour or performance, this difference did not impact the behaviour to be learned.

The UR Control Web Application, which is itself designed as a *DIME* application, once compiled and deployed runs in a Docker environment and it can be used independently of *DIME*. For the learning experiments, everything was thus executed on a single local machine.

4.2 The Learning Experiment Set up

The learning experiment was described graphically using *LearnLib Studio*'s models, as shown in Fig. 7. The left side of the *Learn Experiment Model* shows the definition of the experiment setup. It uses the TTT algorithm [18] and the Random Word Equivalence Oracle, which is a random word counterexample search parameterized with 20 random words with a length between 5 and 10 symbols. The right side of the model shows the graphical definition of the SUL in terms of the commands in its alphabet. The cycle between the TTT algorithm and the Random Word Equivalence Oracle represents the learn loop. Both these elements connect to the SUL via a query counter and a cache, which are filters defined

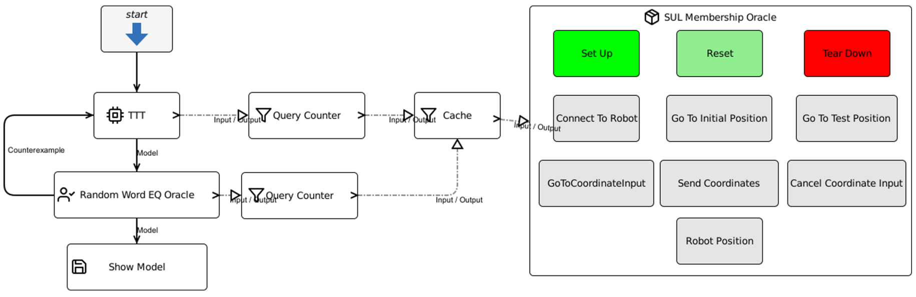


Fig. 7. Definition of the Learning Experiment as a *Learn Experiment Model*

as part of the LearnLib pipeline. While the query counter provides insights on the learning use of resources, the cache offers a potentially increased efficiency because it directly answers known queries instead of invoking the SUL anew. The SUL is represented as a set of 10 symbols: the learning alphabet consists of seven symbols, and the further three are special symbols which deal with the setup and tear down of the SUL and the connection to it.

4.3 The Learning Alphabet

The symbols defined for the learning experiment are described in Table 1. The seven symbols Connect to Robot, Go To Initial Position, Go to Test Position, Go to Coordinate Input, Send Coordinates, Cancel Coordinate Input and Robot Coordinates constitute the learning alphabet. Their names constitute the *input alphabet* to the algorithm. The *output alphabet* consists of their possible outputs: Success, Success (X, Y, Z) which includes 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 independently observe the robot coordinates in the learning process, and correlate them to interactions with the web application.

Beside the input and output symbols, the three helper symbols Set Up, Reset and Tear Down described in Table 2 help manage the experiment, for example to ensure a reliable reset.

These handling of the symbols is implemented in LearnLib Studio. It adopts a MDD approach with a Cinco GDSL similar to the process language of *DIME*, but modified to focus more on consistent outputs symbols. As an example, Fig. 8 shows the implementation of the *Go To Initial Position* symbol. This model is executed every time a learning component sends to the SUL a query containing this symbol. Starting from the *Start SIB*, a so-called WebDriver is needed to emulate the user behavior in a web browser. The *Start SIB* grabs it from a global context and passes it to the next SIB via the dotted data flow edge. As the *Start SIB* has only one control flow successor (the solid line), the *WaitForNode* is executed next. This SIB takes care of extra waiting time to ensure the page has properly loaded. Each SIB has input (blue, top) and output (orange, bottom) ports. Input ports can either be dynamic, i.e. they accept data flow from other

Table 1. Overview of the Learning Alphabet

Name	Outputs	Description
Connect To Robot	Success, ElementNotFound	Tries 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

SIBs, or static, i.e. the value is fixed and predefined when modeling the symbol. The next action in this symbol's workflow, the *Click SIB*, actually clicks the button, and then the symbol execution terminates with its *End SIB*, which in this case is the *Success* output. This *End SIB* also updates the *WebDriver* in the global context. Should any of those two execution SIBs fail, e.g., if the *WaitForNode* reaches a timeout while waiting for the button in, or if the *Click SIB* is unable to click the button, the alternative error path indicated by the red dotted lines is taken. These error paths lead to the *End SIB ElementNotFound*, which signals to the learn experiment that the button is missing.

The collections of symbols in Tables 1 and 2 are defined in this way. They use a newly created custom SIB library to interact with the web application, see Fig. 9. It is part of the example experiments included with *LearnLib Studio* and deals with buttons, numeric fields, and other interactable GUI elements of web application.

Another SIB library was used to interact with the robot directly.

4.4 Performance Issues

During the first learning experiment there were speed management issues with the network socket of the robot: even the sped-up robot in the simulator, much

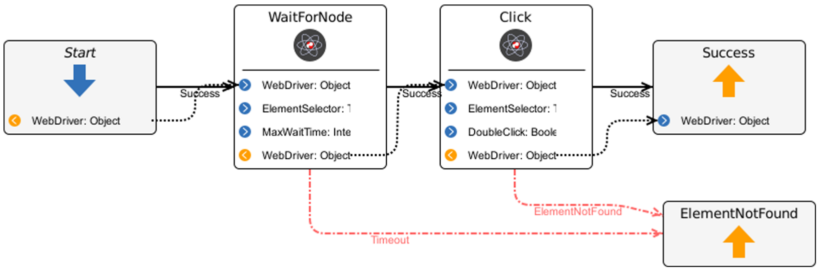


Fig. 8. Symbol definition in LearnLib studio for the click on the *Go to Initial Position* Button

faster to respond than the real robot, could not keep up with the amount of different commands automatically sent to it by the learning algorithm. The adopted solution was to introduce an artificial wait time of ten seconds before querying the coordinates from the robot, the SIB *WaitForNode*, even though this slowed down the overall learning process.

A different approach would have been to poll the robot multiple times until the reported coordinates stop changing, indicating that the robot has reached its final position, or alternatively to instruct the robot call back once the movement is finished. Both approaches would have requested multiple interactions and a more complex logic, so we preferred to opt for the simpler wait solution.

5 Results: The Learned Digital Twin

The Mealy machine shown in Fig. 10 is the behavioral Digital Twin of the UR Controller Web Application as learned through the AAL experiment. It has seven states, based on the seven input symbols introduced with the learning alphabet, the corresponding output symbols, i.e. *Success*, *ElementNotFound* and the different coordinates.

The state q_0 on the top 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 by reloading the application. This behavior is evident in the Digital Twin model's state q_0: it is the initial state and it only allows to move ahead with the Connect To Robot action.

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 three states q_0, q_1 and q_3 (in the 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 target state.

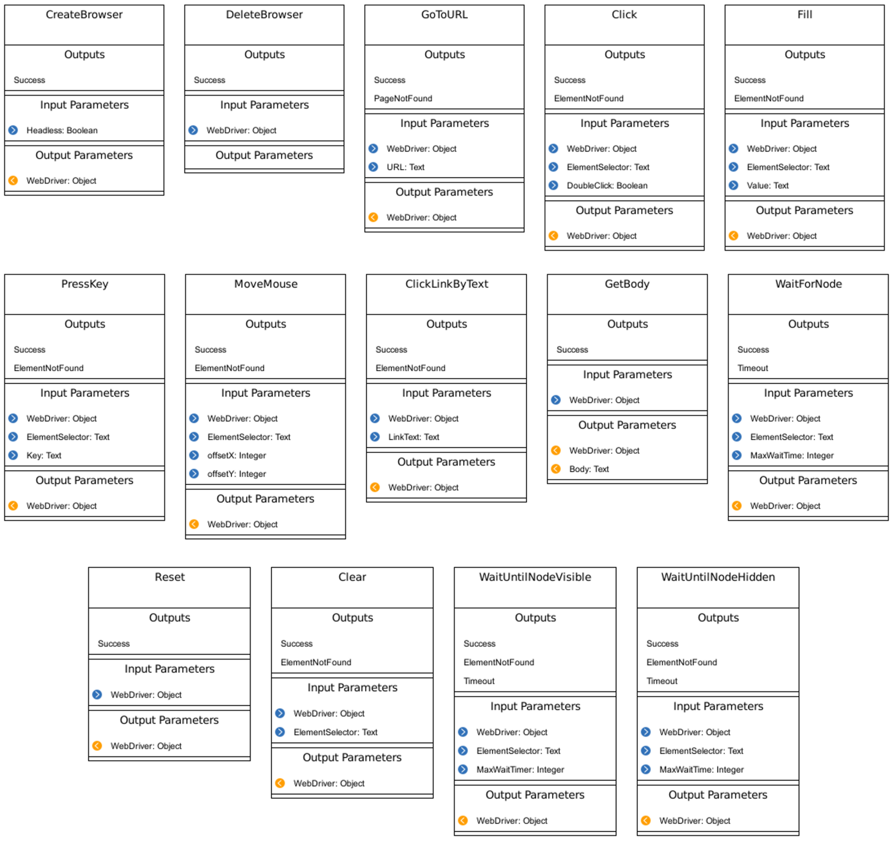


Fig. 9. Selenium SIB library for interactions with web applications.

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

Overall, the final model that emerged is structured 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* behaviour we expected.

The learning experiment was run on a Dell XPS 15 9560 (Intel Core i7-7700HQ, 32 GB RAM, Manjaro Linux) and took 80 min of execution time. Its production took four iterations of the learning loop, with algorithmic search and

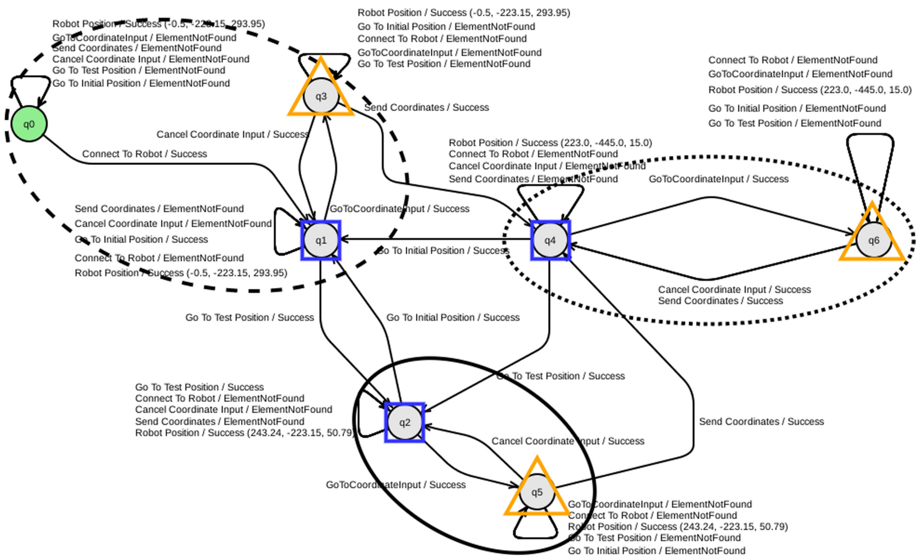

Robot Position / Success (-0.5, -223.15, 293.95)
GoToCoordinateInput / ElementNotFound
Send Coordinates / ElementNotFound
Cancel Coordinate Input / ElementNotFound
Go To Test Position / ElementNotFound
Go To Initial Position / ElementNotFound
q0
Connect To Robot / Success
Robot Position / Success (-0.5, -223.15, 293.95)
Go To Initial Position / ElementNotFound
Connect To Robot / ElementNotFound
GoToCoordinateInput / ElementNotFound
Go To Test Position / ElementNotFound
q3
Send Coordinates / Success
Cancel Coordinate Input / Success
Connect To Robot / ElementNotFound
GoToCoordinateInput / ElementNotFound
Robot Position / Success (223.0, -445.0, 15.0)
Go To Initial Position / ElementNotFound
Go To Test Position / ElementNotFound
Robot Position / Success (223.0, -445.0, 15.0)
Connect To Robot / ElementNotFound
Cancel Coordinate Input / ElementNotFound
Send Coordinates / ElementNotFound
GoToCoordinateInput / Success
Send Coordinates / ElementNotFound
Cancel Coordinate Input / ElementNotFound
Go To Initial Position / Success
Connect To Robot / ElementNotFound
Robot Position / Success (-0.5, -223.15, 293.95)
q1
GoToCoordinateInput / Success
Go To Initial Position / Success
q4
q6
Cancel Coordinate Input / Success
Send Coordinates / Success
Go To Test Position / Success
Go To Initial Position / Success
Go To Test Position / Success
Go To Test Position / Success
Connect To Robot / ElementNotFound
Cancel Coordinate Input / ElementNotFound
Send Coordinates / ElementNotFound
Robot Position / Success (243.24, -223.15, 50.79)
q2
Cancel Coordinate Input / Success
Send Coordinates / Success
GoToCoordinateInput / Success
q5
GoToCoordinateInput / ElementNotFound
Connect To Robot / ElementNotFound
Robot Position / Success (243.24, -223.15, 50.79)
Go To Test Position / ElementNotFound
Go To Initial Position / ElementNotFound

Fig. 10. The final model: the digital twin obtained by AAL

counterexample. The learner asked 218 Membership Queries and 34 Equivalence Queries were posed by the counter example strategy.

6 Property Checking on the Digital Twin

The Digital Twin extraction is interesting per se, but, as we see, its interpretation requires manual analysis and a good understanding of how the system under learning functions: its architecture, its components, both individually and in their communication patterns. This can be done for a small model. However, as soon as systems and models grow, for example in our case when adding a camera that observes from outside the real robot behaviour and steers adjustments to coordinates based on precise measurements, e.g., to limit the movements within a ‘virtual cage’, the model size grows as well, the interactions become more intricate, and a different method of analysis is needed in order to properly evaluate the model and its meaning. Property checking on the Digital Twin is a useful technique.

We use here CTL model checking, specifically model checking with the GEAR [3,4] tool, in order to a) express behavioural properties of the system that can be assessed on the model, and b) automatically check them on the produced Digital Twin model. The four properties we checked are:

Property 1: *If the* Go to Initial *button is clickable, clicking it leads to the robot being positioned in the initial position.*

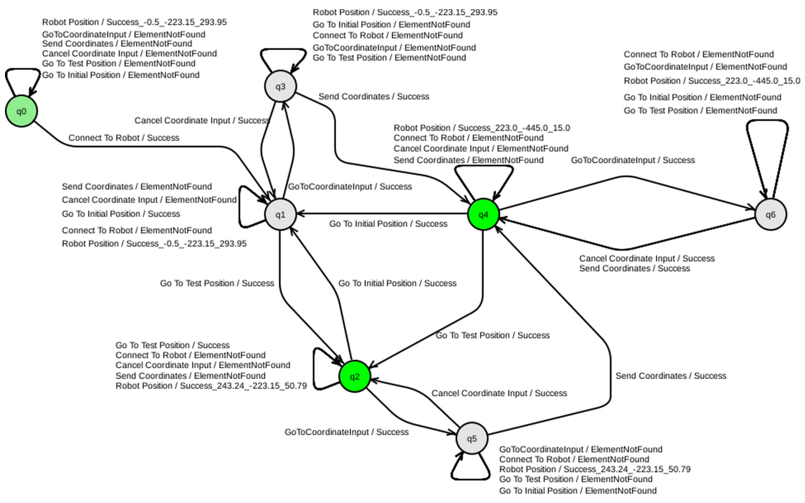

q0
q1
q2
q3
q4
q5
q6
Connect To Robot / Success
Cancel Coordinate Input / Success
GoToCoordinateInput / Success
Go To Initial Position / Success
Go To Test Position / Success
Send Coordinates / Success
Cancel Coordinate Input / Success
Send Coordinates / Success
GoToCoordinateInput / Success
Robot Position / Success_-0.5_-223.15_293.95
Robot Position / Success_223.0_-445.0_15.0
Robot Position / Success_243.24_-223.15_50.79

Fig. 11. GEAR results: States Q2 and Q4, highlighted in green, fulfill the constraint.

```
     (["Go_To_Initial_Position/Success"]
          "Robot_Position/Success (-0.5,_-223.15, 293.95)")
OR
     "Go_To_Initial_Position/Element_Not_Found"
```

Property 2: *If the* Go to Test *button is clickable, clicking it leads to the robot being positioned in the test position.*

```
     (["Go_To_Test_Position/Success"]
          "Robot_Position/Success (243.24, -223.15, 50.79)")
OR
     "Go_To_Test_Position/Element_Not_Found"
```

Property 3: *If the* Go to Coordinate Input *button is clickable, clicking it does not change the position of the robot.*

```
 (
     ("Robot Position/Success (-0.5, -223.15, 293.95)"
      ["Go To Coordinate Input/Success"]
           "Robot Position/Success (-0.5, -223.15, 293.95)"
     )
  AND
     ("Robot Position/Success (243.24, -223.15, 50.79)"
      ["Go To Coordinate Input/Success"]
            "Robot Position/Success (243.24, -223.15, 50.79)"
     )
  AND
     ("Robot Position/Success (223.0, -445.0, 15.0)"
      ["Go To Coordinate Input/Success"]
           "Robot Position/Success (223.0, -445.0, 15.0)"
```

```
        )
    )
OR
    "Go To Coordinate Input/Element Not Found"
```

Property 4: *If the* Cancel Coordinate Input *button is clickable, clicking it does not change the position of the robot.*

```
 (
    ("Robot Position/Success (-0.5, -223.15, 293.95)"
     ["Cancel Coordinate Input/Success"]
          "Robot Position/Success (-0.5, -223.15, 293.95)"
    )
  AND
    ("Robot Position/Success (243.24, -223.15, 50.79)"
     ["Cancel Coordinate Input/Success"]
           "Robot Position/Success (243.24, -223.15, 50.79)"
    )
  AND
    ("Robot Position/Success (223.0, -445.0, 15.0)"
     ["Cancel Coordinate Input/Success"]
          "Robot Position/Success (223.0, -445.0, 15.0)"
    )
 )
OR
    "Cancel Coordinate Input/Element Not Found"
```

As we see from the GEAR screenshot in Fig. 11, the model checker highlights the states satisfying a property, providing good visual feedback to the user. All these properties are fulfilled by this model. Property checking on the digital twin can be a useful way to validate the model, making sure that it captures those phenomena that are known to the designers.

7 Reflections and Lessons Learned

Several considerations come upon reflection on this even simple case study, especially in the context of our conversations with people interested in Digital Twins for Cyberphysical Systems that are not computer scientists themselves. In this group fall the recurring questions about the role of AI (Sect. 7.1), and the fact that there are many different kinds of software models, like design models and behavioral models (Sect. 7.2). We follow then with some reflections on two lines of work that are central to Bengt's and Tiziana's connection: the genesis and evolution of the specific AAL technology we used here (Sect. 7.3), and their work on features and feature interactions (Sect 7.4).

7.1 AAL vs. AI

When using AAL to extract digital twins of (software) systems, we are frequently asked:

Can't you use AI instead?

Resorting to AI as the all-encompassing solution to any unknown seems to become a reflex response to any question concerning systems and their analysis. There are at least two fundamental differences between the models produced by AAL and those produced in AI:

– Many popular AI techniques essentially retrofit (mostly) numerical and/or probabilistic models based on a data set interpreted as an input/output relation, without necessarily a relation with the real system. Even Grammatical Evolution [44], which is based on a BNF-style description of a system's potential actions, essentially tries to match an I/O behaviour provided in a file by first guessing and then recombining populations of alternative "programs" that approximate the real system.
  Instead, AAL systematically explores the real system (or the part of the system one decides to observe, as this is steered through the Learning Alphabet) and provides the minimal model that reflects faithfully all the observations. In this sense, there is an aspect of tightness to the system that the AAL approach has and the AI one does not.
– AI models that replicate systems are themselves mostly black boxes, and even the typical tools used for Explainable AI, like SHAP [25], provide percentages of correlation between certain inputs and certain outputs, but do not provide an analyzable, even enactable model of the system itself like the model in Fig. 10. The value of this Mealy machine as an explanation model, for any What-If analysis, is in a totally different class of confidence and evidence.

7.2 Design Models vs. Behavioural Models

Other frequent questions are:

Isn't the software the model itself?
Isn't the process diagram the model?
Why do you need a digital twin of software?

There is still a widespread belief that software does not need models, that software "is" per se modelling (because it is inherently immaterial, in contrast to the tangible things in manufacturing and production, or even communications, as communication tools come with apparati like transmitters, receivers, etc. In some circles, software is met in the form of simulators, and the simulation software is then identified itself as "the model". The fundamental distinctions between the simulation tool, the simulation run, the data, and the aggregated model from many runs are more or less unconsciously blurred.

In the same line of thought:

If the software is not the model, then the diagrams "are" the model, right?

The graphical presentation of process models and workflows induces some to see them as fundamentally different from code, and thus from software. A recipe like a DIME or BPMN process model is then taken as "being the model", which is true in a behaviour design sense, but not in the sense of exposing the semantic effects of the execution.

The difference between design-time models of behaviour and semantic and runtime or execution models in this guise was showcased and discussed in [38], on the case study concerning a prior version of the Online Conference System (OCS) published as part of the FMICS Working Group state of the art book [10]. There, we used the AAL technologies described in [15] in order to extract by learning the behaviour of a Web application that implements a conference management system. In that case, AAL was applied to a purely cyber system, i.e., not physical at all, and used to show that various properties that were requirements at design time were indeed satisfied by the implemented and deployed system.

This brings us to the specific learning technology used, then and now, in our work. This also brings us to the connection and collaboration with Bengt.

7.3 Learning Technology

We use here a new version of the LearnLib Studio which is now a specialized Cinco-product for defining LearnLib experiments through a custom MDD editor. We specifically used the TTT algorithm of [18]. However, the AAL technology has a long history and it has benefited greatly from Bengt Jonsson's research and work. The original LearnLib [40] was greatly enhanced in collaboration with Bengt and his group during the Connect EU project [41], leading to the first applications within FMICS [31] and to dynamic testing [42]. It was then followed by the Next Generation LearnLib (NGLL) [36], with full details described in [46], and more recently by the Open Source Learn Lib of [19]. This LearnLib has made school, becoming one of the most downloaded and widely used tools for Automata Learning, with the ALEX tool and others like LearnLib Studio as further derivatives. The LearnLib materialized in a very successful tool set the first observations about knowledge based relevance filtering for an efficient use of testing in order to save order of magnitudes of tests when we started to explore model extraction based on systematic execution as a way to extract compact models from deployed systems [17,29,30].

Specifically in AAL, or regular inferences, Bengt and his group contributed greatly to the development of algorithms and optimizations, e.g. exploring the connection between the connection between conformance testing and regular inference [5] back in 2005, then working on the inferring semantic interfaces for data structures [13] and on learning canonical register automata [16], and here most recently combining black-box and white-box techniques [14], until the more recent works on active learning for extended finite state machines [9], in the quest to solve the CONNECT challenges [20] and more.

7.4 Features and Feature Interaction

Also without addressing the topic of humans in the loop and human-machine collaboration that cobots address, it is clear that manufacturing systems deal with system interactions. In the language of coordination models, some of these interactions can be orchestrated, i.e. the interactions are designed in their entirety and can be to a good extent linearized to workflow-deterministic projections onto the individual actors, but this is not always the case. Already two UR3 that collaborate, e.g., to hold and weld a piece could give rise to more diverse interactions if they are considered independent systems that happen to cooperate. This is the realm of choreographies, akin to collections of independent (state) machines that only loosely coordinate. Sometimes one even observes emergent behaviours, that, from the point of view of the modelling, appear spontaneous. In this context, it is frequent to observe so called feature interactions, where independent behaviours that are individually correct and consistent become inconsistent and face ambiguous choices if they have to coexist. In a sense, coupling due by the fact that these behaviours are brought in the same context, often being one the context of the other, exposes inconsistencies for which there is often no good decision policy.

The phenomenon is observed in the robotics domain, where collisions happen because the individual rules and policies driving one of the moving components clash with the others. Early discovery of such interactions is one of the uses of digital twins, ideally with the ability to identify whether such situations are possible, and even better with the ability to reduce the model to a small or even minimal model that models precisely and only the interference potential.

Here, the ability to produce faithful digital twins though AAL, together with the ability to analyze the models by means of model checking is a real asset. Having fully fledged formal models like the Mealy Machine of our case or richer models, like the register automata, may for example help identify certain language elements of the learning alphabet as irrelevant, and produce, either by re-learning or by model abstraction, smaller versions that still correctly characterize the problem. This is difficult and expensive to achieve both by traditional testing and by AI. Some reinforcement learning approaches start to bear fruits, and it is noteworthy in this context that the principle of RL, with an omniscient teacher (mostly simulated by annotations) is similar to AAL's oracle, with its implementations by query-answering mechanisms.

The feature interaction problem was first discovered in the late1990 s in the telecommunications domain. Also here we can look back at joint contributions with Bengt and some of his students. Started with their modular specification of telephone services within first-order linear-time temporal logic [6] and their formalization of Service Independent Building Blocks [39] in that style, this line of work brought us to an intense collaboration in 1998–2001, leading to a sabbatical semester in Uppsala. The approach for incremental requirement specification for evolving systems [22] based on that modular modelling style [23]. Since 1994 we were already working on projects with Siemens concerning the use of formal models for Intelligent Network services. They included model hierarchy [47] and constraint techniques to specify service properties . In particular we produced what today would be called a DSL-based, model driven and generative design environment for the agile development of IN Services [48]. Soon later we added an automated service evolution technology based on constraint-driven modification of the business logic [8], so that at the time of the cooperation with Bengt in Uppsala both groups happened to have independent yet fully aligned modelling styles and technology stacks. This made the collaboration possible, easier, and productive. Eight years later, at the onset of service oriented computing, an overview paper cheekily stated that we had 10 year experience in a field that was 3 years old [35], a claim substantiated by all this experience in research and industrial products with real impact.

Thinking in features has played a significant role in two other contexts: when modelling, then learning and analyzing the Online Conference System already mentioned [24], and later in the approach to constraint-based variability modeling framework [45]. This framework has a much wider applicability than the original feature models, as it targets generic (software and systems) product lines. Families of artifacts are first described in terms of collection of features, then composed, selected and analaysed in terms of their behavioral properties specified as temporal logic constraints. It is this last evolution of our feature-oriented thinking that is closest to the approach we intend to adopt and apply to both the digital twins, in terms of behavioural models, and to the digital thread in terms of end-to-end integration and composition of software and systems.

8 Conclusion and Outlook

In this paper we applied active automata learning experiments to a scenario in remote robotics control, based on the UR family of collaborative robots. While this is a small example, the capability to retrofit Digital Twin models to the behaviour of cyberphysical systems can potentially pave the way to capturing the behaviour of legacy (control) applications in smart manufacturing by means of models amenable to formal analysis and model based testing. We used here the LearnLib Studio for the Automata learning, and the GEAR model checker to verify a few properties of the Mealy machine obtained in the learning phase.

We are currently extending the functionality of the case study by connecting the robot also via ROS, the Robot Operating System popular in the education and research community for CPS, with the goal to provide many versions of the demonstrator and to be able to profile the various technologies involved, sometimes as alternatives like ROS vs. TCP socket. Thinking in features and dealing with feature interaction when having more than one independent subsystem is a further direction of research.

We also reflected on many dimensions of modelling and technology choice stemming from the two contexts in which we see us immersed: the Confirm Research Centre on Smart Manufacturing, with its challenges as a multicultural, multidisciplinary and multisectoral community of research and practice, and the personal history and experience. The Confirm context, in particular the conversation with excellent partners with very different background, lead us to have to rethink, explicitly formulate, and many times tangibly demonstrate rather than explain assumptions and facts that are not common knowledge nor obviously accepted outside of computer science. Here we count the omnipresence of AI as the go-to solution, with the need to justify why one is adopting a different approach, and the understanding of software, its facets and its role in a wider context of production floors. On the personal experience, having worked with many industry sectors and with many collaborators turns out to be a great asset. Beside the familiarity with automata learning, particularly the experience in modelling of telecommunication systems and reasoning about feature interactions has proven useful in recent collaborations also in Confirm, and is a nice benefit that we now draw, stemming from the pleasant and productive collaboration with Bengt and his group in the past 23 years. It is nice to see that there is a long term effect also for collaborations in a research that seemed to be a niche topic at some point.

References

1. Nci-doe collaboration 2020 ideas lab: Toward building a cancer patient "digital twin." an ideas lab to shape the future of predictive modeling across scales from biology to clinical care. Information Age (2020). https://www.imagwiki.nibib.nih.gov/news-events/relevant-meetings/nci-doe-collaboration-2020-ideas-lab-toward-building-cancer-patient
2. Angluin, D.: Learning regular sets from queries and counterexamples. Inf. Comput. **75**(2), 87–106 (1987)
3. Bakera, M., Margaria, T., Renner, C., Steffen, B.: Verification, diagnosis and adaptation: tool-supported enhancement of the model-driven verification process. In: Revue des Nouvelles Technologies de l'Information (RNTI-SM-1), pp. 85–98, December 2007
4. Bakera, M., Margaria, T., Renner, C., Steffen, B.: Tool-supported enhancement of diagnosis in model-driven verification. Innovations Syst. Softw. Eng. **5**, 211–228 (2009). https://doi.org/10.1007/s11334-009-0091-6
5. Berg, T., Grinchtein, O., Jonsson, B., Leucker, M., Raffelt, H., Steffen, B.: On the correspondence between conformance testing and regular inference. In: Cerioli, M. (ed.) Fundamental Approaches to Software Engineering, FASE 2005. LNCS, vol. 3442, pp. 175–189. Springer, Heidelberg (2005). https://doi.org/10.1007/978-3-540-31984-9_14
6. Blom, J., Jonsson, B., Kempe, L.: Using temporal logic for modular specification of telephone services. In: In Feature Interactions in Telecommunications Systems, pp. 197–216. IOS Press (1994)
7. Boßelmann, S., et al.: DIME: a programming-less modeling environment for web applications. In: Margaria, T., Steffen, B. (eds.) Leveraging Applications of Formal Methods, Verification and Validation: Discussion, Dissemination, Applications, ISoLA 2016. LNCS, vol. 9953, pp. 809–832. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-47169-3_60
8. Braun, V., Margaria, T., Steffen, B., Yoo, H., Rychly, T.: Safe service customization. In: Intelligent Network Workshop, 1997. IN 1997. IEEE, vol. 2, p. 4, May 1997. https://doi.org/10.1109/INW.1997.601576

9. Cassel, S., Howar, F., Jonsson, B., Steffen, B.: Active learning for extended finite state machines. Formal Aspects Comput. **28**(2), 233–263 (2016)
10. Gnesi, S., Margaria, T.: Formal Methods for Industrial Critical Systems: a Survey of Applications. Wiley, Hoboken (2013). http://eu.wiley.com/WileyCDA/WileyTitle/productCd-0470876182.html
11. Graf, S.: Building correct Cyber-Physical Systems - can we improve current practice? In: Proceedings of the 23rd International Conference on Formal Methods in Industrial Critical Systems (FMICS 2018). LNCS, vol. 11119 (2018)
12. Hinchy, E., Carcagno, C., O'Dowd, N., McCarthy, C.: Using finite element analysis to develop a digital twin of a manufacturing bending operation. Procedia CIRP **93**, 568–574 (2020)
13. Howar, F., Isberner, M., Steffen, B., Bauer, O., Jonsson, B.: Inferring semantic interfaces of data structures. In: Margaria, T., Steffen, B. (eds.) Leveraging Applications of Formal Methods, Verification and Validation. Technologies for Mastering Change, ISoLA 2012. LNCS, vol. 7609, pp. 554–571. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-34026-0_41
14. Howar, F., Jonsson, B., Vaandrager, F.: Combining black-box and white-box techniques for learning register automata. In: Steffen, B., Woeginger, G. (eds.) Computing and Software Science. LNCS, vol. 10000, pp. 563–588. Springer, Cham (2019). https://doi.org/10.1007/978-3-319-91908-9_26
15. Howar, F., Merten, M., Steffen, B., Margaria, T.: Practical Aspects of Active Automata Learning, chap. 11, pp. 235–267. Wiley, Hoboken (2012). https://doi.org/10.1002/9781118459898.ch11
16. Howar, F., Steffen, B., Jonsson, B., Cassel, S.: Inferring canonical register automata. In: Kuncak, V., Rybalchenko, A. (eds.) Verification, Model Checking, and Abstract Interpretation, VMCAI 2012. LNCS, vol. 7148, pp. 251–266. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-27940-9_17
17. Hungar, H., Margaria, T., Steffen, B.: Test-based model generation for legacy systems. In: Test Conference 2003 Proceedings ITC 2003, International, vol. 1, pp. 971–980, October 2003. https://doi.org/10.1109/TEST.2003.1271205
18. Isberner, M., Howar, F., Steffen, B.: The TTT algorithm: a redundancy-free approach to active automata learning. In: Bonakdarpour, B., Smolka, S.A. (eds.) Runtime Verification, RV 2014. LNCS, vol. 8734, pp. 307–322. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11164-3_26
19. Isberner, M., Howar, F., Steffen, B.: The Open-Source LearnLib. In: Kroening, D., Păsăreanu, C.S. (eds.) Computer Aided Verification, CAV 2015. LNCS, vol. 9206, pp. 487–495. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21690-4_32
20. Issarny, V., et al.: CONNECT challenges: towards emergent connectors for eternal networked systems. In: ICECCS, pp. 154–161. IEEE Computer Society, June 2009
21. Ivanov, D., Dolgui, A.: A digital supply chain twin for managing the disruption risks and resilience in the era of industry 4.0. Prod. Planning Control **32**(9), 775–788 (2021). https://doi.org/10.1080/09537287.2020.1768450
22. Jonsson, B., Margaria, T., Naeser, G., Nyström, J., Steffen, B.: Incremental requirement specification for evolving systems. Nordic J. of Computing **8**, 65–87 (2001). http://dl.acm.org/citation.cfm?id=774194.774199
23. Jonsson, B., Margaria, T., Naeser, G., Nyström, J., Steffen, B.: On modelling feature interactions in telecommunications. In: Proceedings of Nordic Workshop on Programming Theory 1999–008. http://www.it.uu.se/research/publications/reports/1999-008/nwpt99/proceedings/

24. Karusseit, M., Margaria, T.: Feature-based Modelling of a Complex, Online-Reconfigurable Decision Support Service. ENTCS (2), 101–118. https://doi.org/10.1016/j.entcs.2005.12.049
25. Lundberg, S.M., Lee, S.I.: A unified approach to interpreting model predictions. In: Proceedings of the 31st International Conference on Neural Information Processing Systems, pp. 4768–4777. NIPS 2017, Curran Associates Inc., Red Hook, NY, USA (2017)
26. Margaria, T.: Knowledge management for inclusive system evolution. In: Steffen, B. (ed.) Transactions on Foundations for Mastering Change I. LNCS, vol. 9960, pp. 7–21. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46508-1_2
27. Margaria, T.: Generative model driven design for agile system design and evolution: a tale of two worlds. In: Howar, F., Barnat, J. (eds.) FMICS 2018. LNCS, vol. 11119, pp. 3–18. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-00244-2_1
28. Margaria, T.: Making sense of complex applications: constructive design, features, and questions. In: Margaria, T., Graf, S., Larsen, K.G. (eds.) Models, Mindsets, Meta: The What, The How, and The Why Not? LNCS, vol. 11200, pp. 129–148. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-22348-9_9
29. Margaria, T., Niese, O., Raffelt, H., Steffen, B.: Efficient test-based model generation for legacy reactive systems. In: HLDVT 2004: Proceedings of the High-Level Design Validation and Test Workshop, 2004. Ninth IEEE International, pp. 95–100. IEEE Computer Society, Washington, DC, USA (2004). https://doi.org/10.1109/HLDVT.2004.1431246
30. Margaria, T., Raffelt, H., Steffen, B.: Knowledge-based relevance filtering for efficient system-level test-based model generation. Innovations Syst. Softw. Eng. **1**(2), 147–156 (2005)
31. Margaria, T., Raffelt, H., Steffen, B., Leucker, M.: The LearnLib in FMICS-jETI. In: ICECCS 2007: Proceedings of the 12th IEEE International Conference on Engineering Complex Computer Systems, pp. 340–352. IEEE Computer Society, Washington, DC, USA (2007). https://doi.org/10.1109/ICECCS.2007.43
32. Margaria, T., Schieweck, A.: The digital thread in industry 4.0. In: Ahrendt, W., Tapia Tarifa, S.L. (eds.) Integrated Formal Methods, IFM 2019. LNCS, vol. 11918, pp. 3–24. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-34968-4_1
33. Margaria, T., Steffen, B.: Service-Orientation: Conquering Complexity with XMDD. In: Hinchey, M., Coyle, L. (eds.) Conquering Complexity, pp. 217–236. Springer, London (2012). https://doi.org/10.1007/978-1-4471-2297-5_10
34. Margaria, T., Steffen, B.: eXtreme Model-Driven Development (XMDD) Technologies as a Hands-On Approach to Software Development Without Coding, pp. 1–19. Springer International Publishing, Cham (2020). https://doi.org/10.1007/978-3-319-60013-0_208-1
35. Margaria, T., Steffen, B., Reitenspieß, M.: Service-oriented design: the roots. In: Benatallah, B., Casati, F., Traverso, P. (eds.) Service-Oriented Computing - ICSOC 2005, ICSOC 2005. LNCS, vol. 3826, pp. 450–464. Springer, Heidelberg (2005). https://doi.org/10.1007/11596141_34
36. Merten, M., Steffen, B., Howar, F., Margaria, T.: Next generation LearnLib. In: Abdulla, P.A., Leino, K.R.M. (eds.) Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2011. LNCS, vol. 6605, pp. 220–223. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-19835-9_18
37. Naujokat, S., Lybecait, M., Kopetzki, D., Steffen, B.: CINCO: a simplicity-driven approach to full generation of domain-specific graphical modeling tools. Int. J. Softw. Tools Technol. Transf. **20**(3), 327–354 (2017). https://doi.org/10.1007/s10009-017-0453-66

38. Neubauer, J., Margaria, T., Steffen, B.: Design for verifiability: the OCS Case Study. In: Formal Methods for Industrial Critical Systems: A Survey of Applications, chap. 8, pp. 153–178. Wiley-IEEE Computer Society Press (2013)
39. Nyström, J., Jonsson, B.: A formalization of service independent building blocks. In: Proceedings of the International Workshop on Advanced Intelligen Networks, pp. 1–14 (1996)
40. Raffelt, H., Steffen, B.: LearnLib: a library for automata learning and experimentation. In: Baresi, L., Heckel, R. (eds.) Fundamental Approaches to Software Engineering, FASE 2006. LNCS, vol. 3922, pp. 377–380. Springer, Heidelberg (2006). https://doi.org/10.1007/11693017_28
41. Raffelt, H., Steffen, B., Berg, T., Margaria, T.: LearnLib: a framework for extrapolating behavioral models. Int. J. Softw. Tools Technolo. Transf. (STTT) **11**(5), 393–407 (2009). https://doi.org/10.1007/s10009-009-0111-8
42. Raffelt, H., Steffen, B., Margaria, T.: Dynamic testing via automata learning. In: Yorav, K. (ed.) Hardware and Software: Verification and Testing, HVC 2007. LNCS, vol. 4899, pp. 136–152. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-77966-7_13
43. Ross, A.: The top 10 strategic technology trends for 2019, according to gartner. Information Age (2019). https://www.information-age.com/strategic-technology-trends-123475549/
44. Ryan, C., O'Neill, M., Collins, J.J.: Introduction to 20 years of grammatical evolution. In: Ryan, C., O'Neill, M., Collins, J.J. (eds.) Handbook of Grammatical Evolution, pp. 1–21. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-78717-6_1
45. Schaefer, I., Lamprecht, A.L., Margaria, T.: Constraint-oriented Variability Modeling. In: Rash, J., Rouff, C. (eds.) 34th Annual IEEE Software Engineering Workshop (SEW-34), pp. 77–83. IEEE CS Press (2011). https://doi.org/10.1109/SEW.2011.17
46. Steffen, B., Howar, F., Isberner, M.: Active automata learning: from DFAs to interface programs and beyond. J. Mach. Learn. Res.-Proc. Track, **21**, 195–209 (2012)
47. Steffen, B., Margaria, T., Braun, V., Kalt, N.: Hierarchical service definition. Ann. Rev. Commun. ACM **51**, 847–856 (1997)
48. Steffen, B., Margaria, T., Claßen, A., Braun, V., Nisius, R., Reitenspieß, M.: A Constraint-oriented service creation environment. In: TACAS, pp. 418–421 (1996)
49. Steffen, B., Margaria, T., Nagel, R., Jörges, S., Kubczak, C.: Model-driven development with the jABC. In: Bin, E., Ziv, A., Ur, S. (eds.) HVC 2006. LNCS, vol. 4383, pp. 92–108. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-70889-6_7
50. Talkhestani, B.A., Jung, T., Lindemann, B., et al.: An architecture of an intelligent digital twin in a cyber-physical production system. Automatisierungstechnik, **67**(9), 762–782 (2019). https://doi.org/10.1515/auto-2019-0039