Plant Model Generator from Digital Twin for Purpose of Formal Verification

Midhun Xavier*, Johannes Håkansson*, Sandeep Patil*, Valeriy Vyatkin* †
* Department of Computer Science, Computer and Space Engineering, Lulea Tekniska Universitet, Sweden
†Department of Electrical Engineering and Automation, Aalto University, Espoo, Finland

Email: midhun.xavier@ltu.se, johhki-4@student.ltu.se, sandeep.patil@ltu.se, vyatkin@ieee.org

***Abstract*—This paper reports on a method of automatic generation of a formal model of plant from the behaviour traces recorded from its digital twin. The traces are observed from simulation in the loop of the digital twin in Visual Components connected with distributed automation software, developed in NxtSTUDIO according to IEC 61499. The generated modular formal model of the closed-loop system is transformed to the model of uncontrolled plant behaviour extended with non-determinism. The model is then combined in closed-loop with the formal model of controller, generated from its source code using the fb2smv tool. The verification and simulation is done by the symbolic model checker NuSMV tool, which verifies various CTL/LTL specifications of the system.**

***Index Terms*—Model synthesis from traces, Formal verification, Plant Modelling, State machine generation**

I. Introduction

Formal verification of automation systems is a promising technique to automatically verify the correctness and safety of automation systems. In particular, verification by model checking in closed-loop helps to identify the flaws in model's design via counter examples, but development of formal plant models is resource consuming. The model checking [1] is the efficient way to prove system's correctness for safety critical applications and it is also used because software used to design the industrial automation systems has less computation complexity and uses limited amounts of data structures.

Process mining [2] [3] [4] has been quite popular in the past decades which extracts process models from event logs. This paper [5] describes an algorithm which is used to produce the process model in the form of Petri nets from a recorded event log. The formal process model [6] can be used to identify the various specifications of the system. In this paper, a formal process model is constructed with the help of recorded event log.

The same time, simulation models are widely used for manufacturing systems, they are popular for purposes of illustration, virtual commissioning, maintenance, etc. The simulation model does not ensure 100% verification and validation automation systems. It would be great if it was possible to take advantage of their existence and automatically generate the formal models.

The author of [7], solves the problem of automatic plant model construction from existing specification, but it was focusing mainly on process systems. This paper aims at developing a similar method for discrete manufacturing systems.

This paper builds upon previous work from [8] but the key differences are related to state machine generation. The previous work is based on the state machine generation for a controller using traces from a real controller, whereas this paper takes a look at state machines and formal model generation of a plant using traces from a digital twin model of the plant. The generated model's behaviour is evaluated by verifying the functional properties of the plant. This paper explores an approach to automatically generate plant models of control systems from traces of digital twin and ensures that the generated formal model consists of desired plant behaviour.

II. Case Study

As a case study, the Energy Autarkic Actuators and Sensors[1] (EnAS), a test bed located at Aalto University, representing a small scale industrial production scenario and is used for the development and testing of various industrial automation techniques. Included with sets of pneumatic operators such as jacks and grippers, motor driven conveyors and laser sensors.

An accurate simulation model of EnAS has been developed using Visual Components 3D simulation environment. In this research, the simulation model was connected in the loop with the automation controllers. Therefore, every reference to the *plant* is referred to the digital twin version, unless specifically stated. Figure 1 shows the visual representation of the digital twin. The major components are the *main plant*, the *Automated Guided Vehicle (AGV)*, and the *IRB*. The following part briefly explains the major software applications used.

The controller for the plant is created using nxtSTUDIO and the controller is connected to the plant in Visual Components via OPC UA communication protocol, so that the controller can receive status updates from the plant, as well as control the plant through various signals.

The data collected by the plant is used for generating a formal model with the help of a proposed algorithm and it can be coded in any language. Here a Python script is used to restructure the data and create the desired model from the traces. The generated plant model is evaluated with the help of a symbolic model checker tool NuSMV, which is used to verify and check certain properties of a system.

[1]https://www.energieautark.com/

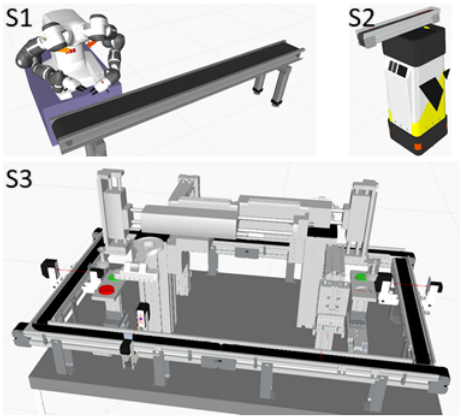

Fig. 1. S1) Isolated IRB-subsystem of plant, S2) Isolated AGV-subsystem of the plant, S3) Main section of the plant

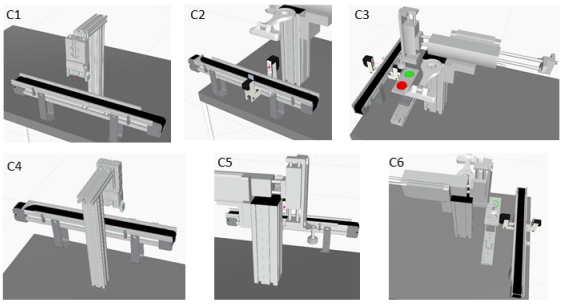

Fig. 2. Isolated C1-subsystem of the plant, Isolated C2-subsystem of the plant, Isolated C3-subsystem of the plant, Isolated C4-subsystem of the plant, Isolated C5-subsystem of the plant, Isolated C6-subsystem of the plant.

A. Digital Twin Component Isolation

Isolating components from the holistic view of the system enables us to explore parts of the system independently. The idea comes from partial order reduction [9]. Our knowledge of the system can be utilized to deduce plenty of information to remove irrelevant data.

The entire plant, presented in Visual Components, is divided into smaller subsystems. The first subsystem is the mobile robot ABB IRB 14000 (IRB) in the top-left corner. This can be viewed as an isolated system in figure 1 (S1).

This specific part of the system does the following actions: the robot moves from initial position to main plant, it fetches an object from a specified position from the main plant, then it returns to initial position with the object and feeds the object away from the process by placing the object on the conveyor belt. To this subsystem, interesting part from a global view, whether the position on the main line is vacant or not, anything other than this in the main system is irrelevant to this subsystem.

Similarly, Figure 1 (S2) is its own subsystem named AGV, which includes a moving robot with a conveyor belt on top. The task for this subsystem is to move from the initial position towards the main plant and connect to a set point of the main conveyor line. It may then feed an object into the system along the conveyor belt. Relevant information to this subsystem is whether the conveyor belt connected to is running or not. All other information is irrelevant. A full view of the main section of the plant can be seen in figure 1 (S3). As we dive deeper into the details of the main plant, we have split it into six parts, each with different components and functionality. They are depicted in their isolated form in the figure 2.

Each subsystem of the plant is kept as small as possible. This is due to the fact that it is easier to work with a smaller state space. If we consider the entire system, the state space will be huge and state space explosion may occur. Therefore, we divided the system into smaller sections, So we only need to consider a few parameters, hence the complexity will decrease substantially. The following points briefly explain about each of the different subsystems.

- The subsystem in figure 2 (C1), contains a black conveyor belt that runs from right-to-left, a gripper that grips objects from above and two sensors: one in the gripper and one on the conveyor belt below the gripper.
- The subsystem in figure 2 (C2) has a conveyor belt that runs from right-to-left, a sensor over the belt to detect objects, and an overhead camera.
- The subsystem in figure 2 (C3) consist of a conveyor belt that moves from bottom-to-top, a sensor across the belt to detect objects, a sledge that may hold details, and a jack that can move details between the sledge and the conveyor belt.
- The subsystem in figure 2 (C4) has a black conveyor belt that runs from left-to-right, A gripper that grips objects from above, and two sensors: one in the gripper and one on the conveyor belt below the gripper.
- The subsystem in 2 (C5) has a conveyor belt that runs from left-to-right, a sensor over the belt to detect objects, and an overhead camera.
- The subsystem in figure 2 (C6) has a conveyor belt that moves from top-to-bottom, a sensor across the belt to detect objects, a sledge that may hold details, and a jack that can move details between the sledge and the conveyor belt. Once an object leaves the bottom part of figure 2 (C2), it joins at the start of subsystem 2 (C3) again.

B. Proposed Solution Approach

The solution works as follows in Figure 3: the digital twin communicates with the controller which is designed in IEC 61499 standard with the help of NXT control software. Function block representation of controller in XML file is then fed to fb2smv tool, which generates SMV code for controller and also provides skeleton structure for plant model to incorporate closed loop verification of the whole system. The digital twin application records the behavior of the plant and the output from the application is added to a trace file. The trace file consists of a raw text describing the behavior of the plant. The model generator creates a model file depending on the order of recorded actions in the trace file. Formal model

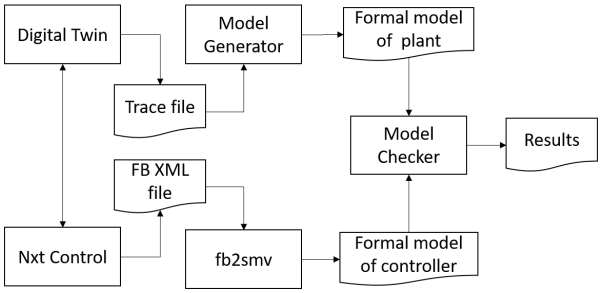

Fig. 3. Structure of the solution approach.

of plant is then added into the smv skeleton structure provided by the fb2smv tool. The functional properties of the system and safety properties are added to the model file prior to model checking. The updated model is fed to the symbolic model checker tool NuSMV which verifies the CTL/LTL properties of the system. The NuSMV determines whether the specifications are TRUE or return the counter examples which show where and how our properties do not hold true.

III. Formal model generation from trace

In Visual Components, whenever an action occurs, it records the event into a trace file. Each event contains timestamp, subsystem, component and action. First events in a trace file are sorted based on timestamp and then classified to its corresponding subsystem level. While generating a model from the trace, we can either consider the complete trace of the system or we can specify two time slots to create the model only for a specific section. We can test each part of the system but we must know the time intervals when the desired section was executed.

The functionality for taking the traces and turning them into a model is done through an algorithm. The basic SMV code structure of the plant model is created with the help of model generator algorithm 1, which is a brief description of an actual script. Model generator sorts the trace according to the ascending order of timestamp, declares and initializes each component's variables and finally transitions of each variable is identified by analyzing each entry in the trace.

Algorithm 1 Model generator

```
0: function MODELGENERATOR
1: Set the initial values by running the method addInitialVal-
   ues
2: Set lower time limit
3: Set upper time limit.(Default time is between 0-3600
   seconds)
4: Sort the desired trace by running the method sort-
   Trace(filename, lower time, upper time).
5: Add "MODULE main" to model file.
6: Add "VAR" to model file.
7: for every component in components do
8:    Declare all variables in the component with specified
      data type
9: end for
10: Add "ASSIGN" to model file.
11: for every component in components do
12:    Add all variables in the component with initial value.
13: end for
14: for every component in components do
15:    Add "next(component):= case" to the model file
16:    for every entry in trace do
17:       if the entry in trace caused the change for the variable
          then
18:          Add the constraints as conditions and the action of
             the entry as action to the model file.
19:       end if
20:    end for
21:    Add "TRUE:component;" to the model file.
22:    Add "esac;" to the model file.
23: end for
```

IV. Closed loop verification of plant model from Digital Twin

The process to reach the goal is split into following parts, trace generation from the digital twin, model generation from the generated trace, embedding the plant model into SMV code structure provided by the fb2smv tool and updated smv code is given for verification using NuSMV. Through every iteration of these steps, more aspects are considered and issues are rectified, until a final working model is generated.

The formal verification can be done using the NuSMV tool but it is required to convert the controller and plant model to SMV format. The SMV code of the controller is created by the fb2smv tool and the plant model SMV code is generated by analyzing log trace with the help of model generator algorithm. In order to do formal verification of the system, we need to combine the controller smv code and plant model smv code. We created a function block skeleton structure for a plant model with simple ECC in NXT studio software and it is connected to the controller. Whenever the controller generates a control signal, the plant goes to the process state and updates the sensor values.

The following challenges occurred while combining the plant model into SMV code structure provided by the fb2smv converter.

A. Receiving control signals from controller to plant

The plant model goes to different states based on the control signals so these control signals should be given to the plant. It is possible to add the control signals by declaring, initializing and assigning control signal value from the controller. The following example shows how Jack control signals (extend signal) are transferred to the plant model.

```
VAR J1_extend_value : boolean;
init(J1_extend_value):= FALSE;
next(J1_extend_value):= case
        event_REQ : J1_extend_value_ ;
        TRUE : J1_extend_value;
esac;
```

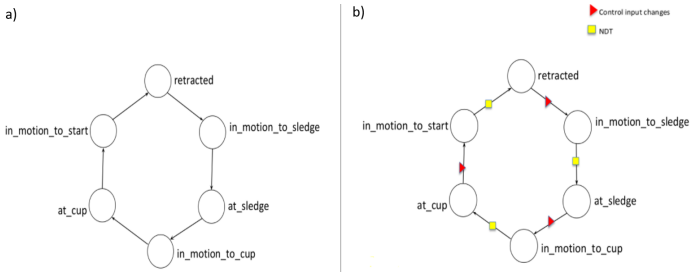

Fig. 4. a) state machine equivalent obtained after applying model generator algorithm b) Final jack model state machine with control signal and NDTs

B. Jack Logic Design

The figure 4 (a) shows the state machine obtained from SMV code of its plant model. This state machine needs to be modified in order to attain the accurate behavior of the jack. Initially the state machine was moving directly from ”retracted” state to ”at_sledge” via an intermediate state ”in_motion_sledge”. Non-deterministic transitions in motion states helps to achieve the similar behavior of the model which is used to identify the flaws that exist in the system via verifying CTL/LTL properties. The figure 4 (b) shows the final structure of the state machine with non-deterministic transitions. The following SMV code represents jack logic with NDTs.

```
next(C5_jack_position) := case
C5_jack_position = retracted & J1_down_value:
    in_motion_to_sledge;
C5_jack_position = in_motion_to_sledge & NDT:
    at_sledge;
C5_jack_position = at_sledge & J1_extend_value
     : in_motion_to_cup;
C5_jack_position = in_motion_to_cup & NDT:
    at_cup;
C5_jack_position = at_cup & (!(J1_down_value))
     & (!(J1_extend_value)):
    in_motion_to_start;
C5_jack_position = in_motion_to_start & NDT:
    retracted;
TRUE: C5_jack_position;
esac;
```

C. Passing sensor values to controller

The sensor values of the plant need to be transferred to the controller, These sensor values help the controller to make the next decision according to the plant condition. The following example shows how jack's top sensor value is given to the controller.

```
next(J1_top_value_):= case
    C5_jack_position=retracted: TRUE;
    !(C5_jack_position=retracted): FALSE;
    TRUE : J1_top_value_;
esac;
```

V. CONCLUSION AND FUTURE PLAN

We implemented the SMV model of the conveyor subsystem and connected to the controller for closed loop verification. Testing and validation is done with the help of NuSMV by checking various properties of the system.The analysis of the solution yields a promising result. The model generator is consistent in generating a formal model of a plant because the specification of the formal model provides the expected output for more than one trace file. The resulting general solution is derived from one digital twin. To improve the legitimacy of the solution, it should be applied to another system to see if it actually works.

We need to make sure that the digital twin records all possible traces in the plant otherwise the verification process may fail. Ensuring the digital twin contains all problematic traces that are possible in the plant could be the next step in future.

The implementation of a tool which automates analysis and verification of system properties requires an extensive development process. However, it is possible to come a long way with automatic generation, with minor additions to finalize the process. Another point of future work that was not explored at all, but might be an area of interest, is simultaneous running of analysis alongside the actual plant. In this approach we need to generate the model during run-time instead of generating a model using traces from a specified time. The benefits of this approach makes system analysis in real time instead of after-the-fact.

ACKNOWLEDGMENT

This work was sponsored, in part, by the H2020 project 1-SWARM co-funded by the European Commission (grant agreement: 871743). We are also thankful to Visual Componenets (https://www.visualcomponents.com/) for providing with licenses for the 3D simulation tool.

REFERENCES

[1] E. M. Clarke, E. A. Emerson, and A. P. Sistla, “Automatic verification of finite-state concurrent systems using temporal logic specifications,” *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 8, no. 2, pp. 244–263, 1986.

[2] J. E. Cook and A. L. Wolf, “Discovering models of software processes from event-based data,” *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 7, no. 3, pp. 215–249, 1998.

[3] R. Agrawal, D. Gunopulos, and F. Leymann, “Mining process models from workflow logs,” in *International Conference on Extending Database Technology*. Springer, 1998, pp. 467–483.

[4] J. E. Cook and A. L. Wolf, “Event-based detection of concurrency,” *ACM SIGSOFT Software Engineering Notes*, vol. 23, no. 6, pp. 35–45, 1998.

[5] W. van der Aalst, T. Weijters, and L. Maruster, “Workflow mining: discovering process models from event logs,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 16, no. 9, pp. 1128–1142, 2004.

[6] V. Vyatkin, H.-M. Hanisch, C. Pang, and C.-H. Yang, “Closed-loop modeling in future automation system engineering and validation,” *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 39, no. 1, pp. 17–28, 2008.

[7] I. Buzhinsky and V. Vyatkin, “Plant model inference for closed-loop verification of control systems: Initial explorations,” in *2016 IEEE 14th International Conference on Industrial Informatics (INDIN)*, 2016, pp. 736–739.

[8] D. Chivilikhin, S. Patil, K. Chukharev, A. Cordonnier, and V. Vyatkin, “Automatic state machine reconstruction from legacy programmable logic controller using data collection and sat solver,” *IEEE Transactions on Industrial Informatics*, vol. 16, no. 12, pp. 7821–7831, 2020.

[9] D. Peled, *Partial-Order Reduction*. Cham: Springer International Publishing, 2018, pp. 173–190. [Online]. Available: https://doi.org/10.1007/978-3-319-10575-8$_6$