Formal Behavioural Models to Facilitate Distributed Development and Commissioning in Industrial Automation

James Harland, Jan Olaf Blech, Ian Peake and Luke Trodd
RMIT University, Melbourne, Australia
{*james.harland,janolaf.blech,ian.peake,luke.trodd*}*@rmit.edu.au*

Keywords: software engineering, spatial modeling, cyber-physical systems

Abstract: We aim to facilitate semi-automated collaborative distributed development, commissioning, operation and maintenance using formal behavioural models. We are interested in a whole-of-system context, where properties of the physical systems increasingly depend explicitly on software configuration, thus requiring validation. Our models are cyber-physical, comprising not only the control software itself but also mechatronic elements under control involving sensing/actuators, such as pneumatics, hydraulics, and motor drives. We discuss various issues relevant to the problems of collaboration, and we provide requirements for collaboration in a specific experimental context involving elements of a small-scale food packaging plant.

1 Introduction

Cyber-physical systems involve both physical aspects such as sensors, actuators, pneumatics, hydraulics and motor drives as well as various kinds of software for controlling them. Such systems often have several components which contain sensors and actuators, which allows the components to operate without any direct control based purely on their sensor inputs.

We are interested in modelling such systems using behavioural models in order to analyse and predict overall properties of the system. The increasing dependence of such systems on software control makes it increasingly important not only to validate such software but also to explore possible configurations of the system. The variety and sophistication of the technologies used means that the development and commissioning of such systems usually requires a team of varied skills, such as electrical engineering, mechatronics, hydraulics and software development. An appropriate formal model of the system will thus provide both a framework for co-operative work as well as a method of dividing work amongst the team.

Dividing tasks such as the development of a component into subtasks and assigning them to different teams as well as the identification of communication dependencies between the teams and its individual members (Figure 1) can benefit from automatic solutions. Here, we are targeting formal methods based solutions.

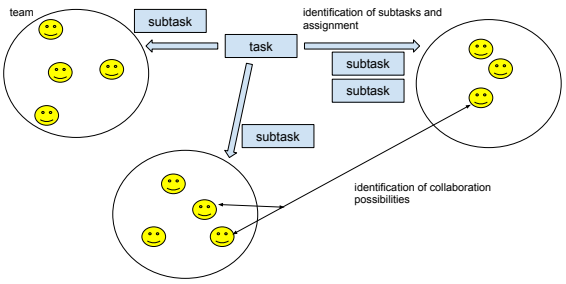

Figure 1: Collaboration overview

Our particular context is a small-scale demonstrator food plant. This has a number of components which are connected by programmable logic controllers (PLCs) and other devices (see Figure 2). We are particularly interested in the handling of bottle caps in preparation for placing caps onto bottles. Blue colour-coded elements and clear tubes are for pneumatic (air pressure) actuation. Black cables carry sensor/actuator signals. A *piston* feeds pushes caps from a *column* of caps (right of centre in Figure 2) to a pickup *plate*. A pneumatic *lever* rotates to the plate to pick up the cap and then rotates to a raised *conveyor* (centre of Figure 2). A mechanical *gripper* (left of centre in Figure 2) picks caps off the conveyor, lifts them and places them onto bottles. The operation of the level can be better seen in Figure 3; the lever moves a single cap (pushed out from the column of caps by the piston) onto the end of the conveyor (at the right of Figure 3). Once the cap reaches

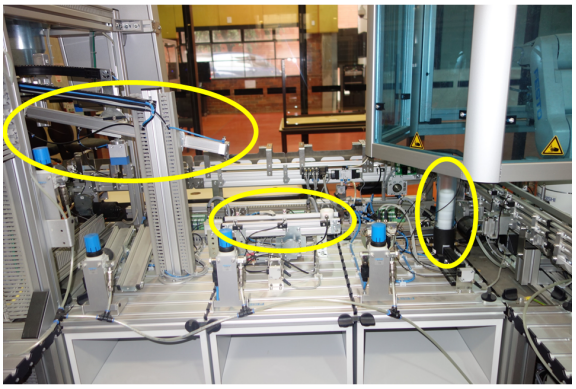

Figure 2: Experimental setup — mini food processing plant with relevant parts highlighted

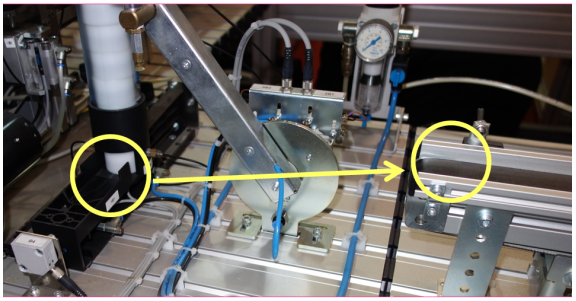

Figure 3: Experimental setup — mini food processing plant 2 with cap movement highlighted

the end of the conveyor, the gripper moves to a position above the cap, and then descends. Once the cap is in its grasp, the gripper reverses its path, moving the cap upwards and then across (see the motion path highlighted in Figure 4).

Components in such systems typically operate via sensors. When the sensor is activated (for example, by the piston detecting the presence of a bottle cap), the component is activated. The recent development of cheap miniature computers such as the Raspberry Pi (Raspberry Pi) means that it is now possible to introduce more intricate control mechanisms into environments such as this, which provide greater opportunities for exploring possible configurations of the bottling system, and particularly when it comes to coordination of concurrent activities. For example, the lever can only pick up the cap when one is present, which could be detected by a sensor. However, if the controller (i.e. the Raspberry Pi) knows that a cap has just been put into position, there is no need to sense

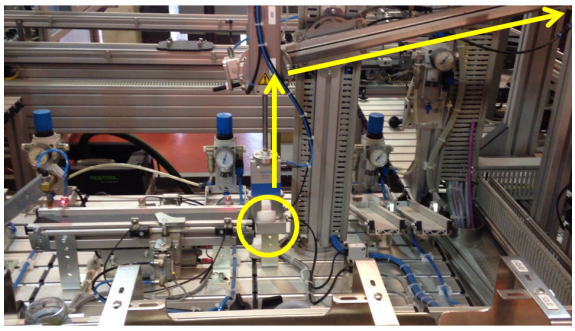

Figure 4: Experimental setup — gripper with motion path highlighted

whether one is present or not, and the lever can be activated independently of its sensor. This allows for greater variations of co-operation during the concurrent operation of the components by centralising some aspects of control in the Raspberry Pi. Raspberry Pis are now a well-known technology designed for experimentation and innovative applications. They are also cheap, based on open source software and are relatively well-documented. This makes Raspberry Pis an attractive platform for analysing the possibilities for controlling the bottling system.

In this position paper we discuss the ways in which we can apply the techniques of formal methods in order to analyse configurations of the bottling system. As noted above, there are a variety of skills needed to investigate end-to-end properties of this system, such as throughput, safety, energy efficiency and robustness. In particular, the presence of the Raspberry Pis means that it is possible to investigate tradeoffs between co-ordination via sensors and via explicit control. The naturally reactive nature of co-ordination via sensors has a greater amount of robustness and redundancy, but allowing programmed control will allow for potentially greater exploitation of global properties of the system.

It should be noted that this context requires an agile approach to configuration and deployment. This is due to not only the multi-disciplinary nature of the skills needed to deploy the system but also the dynamic nature of the environment in which it operates. Over the life of the plant, not only may components and other parts of the system change or be replaced, but also the requirements for the system may evolve. This means that we expect a cycle of continuous updates to the system, which will involve regular reconfiguration and refinement. Hence we are not using formal methods for the articulation and satisfaction

of deployment constraints, but as *a means of capturing properties of the system in a way which will allow regular reconfiguration and collaboration*. This will also allow us to explore "what if" scenarios, such as potential reconfigurations, and specific ideas about what constitutes optimality for the system (whether it be safety properties, numbers of sensors required, measures of concurrency, or an ability to recover from mishaps).

This also means that the semantics will need to be able to express *contracts* for each component. For example, the piston that moves the bottle caps forward may vary the way in which its internal hydraulics work, but as long as there is no change in the frequency at which the caps can be moved or the way in which it is triggered, the rest of the system need not be concerned. However, if a change to the hydraulics affects, say, the power consumption, or the timing at which the caps can be moved, then there is a need to collaborate in order to reconfigure the system appropriately. This means that the formal description of the system becomes an important element of the collaboration, as it is the place where interactions between the components are expressed and analysed. This will also provide a means for standardising the terminology used.

This means that our point of distinction from other approaches is to use the food processing plant as a concrete case study of how to organise collaboration between a diverse group of people. In particular, supervision of the plant is often where the need for collaboration arises. For example, a new manager may need to have an overview of what the plant does and how the various components work together. This overview may result in some changes in the management of the plant, such as prioritising electronics over hydraulics due to the availability of local expertise. In general, there may be mismatches between skills required for the plant and the local availability of qualified personnel, or imperfections in some components that require some subtle changes to the way the plant operates. Suppliers may change over the lifetime of the plant, or come from various different parts of the world, which, together with local market factors (taxes, exchange rates etc.) may vary the way in which components are valued. Naturally recalling and exchanging components is much less preferable than adapting existing ones to changing conditions.

It is also worth noting that collaboration may come in a number of forms. One is collaboration across disciplines, but there is also a need to collaborate over temporal differences (e.g. components that change over time), distance (e.g. mining operations controlled from hundreds of kilometres away), or across phases as in the well-known V-model (ISO26262). There may be many forms of collaboration, and it is important to allow as wide a variety of collaboration as possible.

A further pragmatic issue is that the semantics must be sufficiently simple so that it can be readily understood by a wide variety of professionals. This means that temporal logics and other semantically rich formalisms are not appropriate in our context. Whilst such formalisms are well-known in software engineering, it is highly likely that people outside the formal methods community will find them too difficult to use. This means that our semantics will be based around finite state automata, and particular variants such as timed automata (Alur and Dill 1994). This also has the advantage of being able to switch between levels of abstraction. For example, if we wish to focus our attention on the gripper, rather than the system of the piston, lever, conveyor and gripper, then it is relatively straightforward to consider the gripper as the configuration of a number of components, such as the gripper arm itself, the pin that moves vertically up and down, and the mechanism that moves the gripper horizontally.

A key observation is that we are interested in validation of the current configuration, no matter what the current stage of development may be. This is in the same spirit as the V-model, in that at any point in the process, we are able to identify validation issues, and potentially address them. This is another reason for keeping the semantics relatively simple, as it is more focused on the interaction between components than on the particular properties of the components themselves.

While the number of components in our example is small, and hence can be manually reconfigured as need be, in general it will be a lot more difficult to revise and validate a particular configuration of the plant. For this reason we anticipate using *satisfiability modulo theory* (SMT) solvers and related technologies in order to analyse configurations of the plant. This seems particularly appropriate given the potentially global nature of the control mechanisms made possible by the Raspberry Pis, and in particular the way in which the overall properties of a configuration can be inferred from the properties of individual components. The problem of determining an appropriate architectural configuration can then be transformed into a problem of determining whether a specific formula is satisfiable or not, which can then be provided as input to an SMT solver.

2 Example System

As shown in Figures 2 and 3, we focus on a particular part of the Festo installation. A column of bottle caps (which can be seen in the left of Figure 3) is supplied, and the bottom cap in this column is pushed forward by the piston so that it is underneath the pneumatic lever. The lever then descends and collects the cap, rotates through around 135° before halting and dropping the cap on the conveyor. The conveyor then moves the cap around 30cm to the other end, where the gripper then descends, collects the cap and places it on a rotating platform (to the right of Figure 4), from which the caps are placed onto bottles (by another part of the plant). The cap's journey from the initial column to the rotating platform takes about 20 seconds.

One point to note is that in the configuration in the plant, the lever can only rest when it is at the position where it drops the cap onto the conveyor belt. However, in order for the cap to move along the conveyor, the lever has to be moved from this position, because otherwise the end of the lever remains inside the cap. This means that an extra iteration of the lever movement is required (so that the lever has to make two movements in order to move one cap) to allow the tip of the lever to be out of the way when the cap is moved by the conveyor. This is an artefact of the cyber-physical nature of the system.

We construct a possible abstract specification of the physical process. In this example our method is based on composition and synchronisation of cyclic state machines of local processes corresponding to underlying cyber-physical components including machinery and their associated sensors and/or actuators, following the approach in (Schmidt et al 2003). State machines capture that local physical processes are in contact with neighboring processes via synchronisation such that output actions of one process correspond to input actions of a contacting process. For several components, actions imply synonyms for labels on corresponding input or output transitions linking connecting states: The piston is initially *home*, then *push(ing)*, at *limit* of extension (output: limit), *retracting* (input: retract) or in the *home* position. The lever and gripper are either *carrying*, *waiting* to drop, *dropping*, or *(return)ing*. The conveyor is either *moving* or *stopped*. Sensors on the conveyor indicate via transitions when a cap at the *entry* (placed by lever) and *exit* (lifted by gripper) positions *arr*ived or *dep*arted, hence for example the overall cyclic sequence: entry.arr, entry.dep, exit.arr, exit.dep. The *(rotating) platform* is either *moving* or *stopped*.

The overall behaviour of the system is a synchronised composition of all these. The description of the system can be derived incrementally by pointwise-synchronising pairs of component processes. In composition, for example, the conveyor states are guarded by actions *move* and *stop* which are paired with actions in a synchronisation process which interacts with the lever and the gripper.

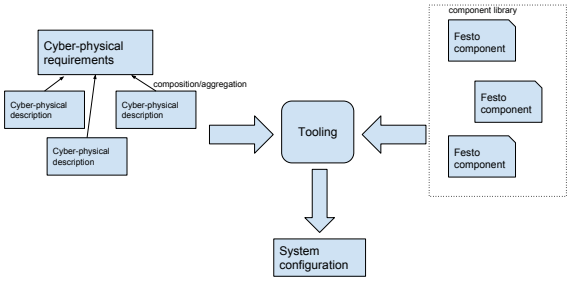

Figure 5: Overview on our methodology

3 Collaborative Scenarios

Figure 5 provides an overview of our methodology. The Festo equipment is designed to be flexible, and hence generally requires tailoring to specific needs. The introduction of the Raspberry Pis means that there are greater possibilities for co-operation than when the components are operated by sensors alone. This means that we may be able to alter some of the end-to-end properties of the configuration by utilising more explicit control mechanisms. For example, it takes around 20 seconds for the cap to travel from the initial column to the place where it is put down by the gripper. We can potentially increase the throughput of the system by utilising the conveyor as a buffer for caps. From the images, it seems that around 6 caps would fit on the conveyor at one time (note that there is an arm on the conveyor which means the caps cannot proceed past a certain point without an explicit command from a controller). The time taken for the piston to move the cap is about 1 second, and the time for one cycle of the lever and the gripper is about 3 seconds and 8 seconds respectively. It takes the conveyor around 3 seconds to move the cap from one end to the other. This means that it would be possible to have up to 3 caps on the conveyor at any one time, with the gripper (being the slower of the two moving arms) working continuously while the lever can work at a slower rate, as it only needs to ensure that there are 3 caps on the conveyor at any time.

We may also wish to experiment with the frequency of the lever action, as it may be better (for

hydraulic, electrical or maintenance reasons) to move the lever for three continuous cycles (to fill an otherwise empty conveyor) and then remain idle for three cycles (allowing the conveyor's store of caps to emptied before moving again), rather than to initially fill the conveyor (for three cycles) and then work one cycle at a time to maintain three caps on the conveyor (i.e. synchronise the movement of the lever with the gripper). As mentioned above, we also need to take into account the property that caps cannot move on the conveyor when the lever is at rest at 135°, meaning that it may be necessary to move the lever even if there is no cap to collect.

Alternatively, we may wish to maximise the possibility of recover from an error, such as the lever placing the caps on the conveyor in slightly different ways each time, which may mean it is safer to limit the number of caps on the conveyor at any time to 2. We may also want to minimise the number of active sensors (or other parts) in the system, or to focus control on the most critical aspect of the system (such as the number of caps on the conveyor at any one time) and allow the other components to be operated purely by sensors.

We may also wish to reconfigure the lever itself, so that it can come to rest at a vertical position after dropping the cap. This will avoid the problem of having to move the lever so that the conveyor can move the cap, but requires more sophisticated hydraulic management, and will presumably mean that the lever moves at a significantly slower pace. The change of the properties of the lever will be reflected in the semantics, which can then be used as a tool for communication and hence collaboration between the hydraulics expert, the conveyor expert and the software engineer in charge of the Raspberry Pis.

The semantics could also be used as a specification of what behaviour is desired, which can then be used to investigate the potential behaviour of the system. In the above example with the lever, the software engineer may be interested to know if the lever can be made to rest in a vertical position with a cycle time (including picking up the cap from the piston and placing it on the conveyor) within 9 seconds; the hydraulics expert can then analyse whether this is possible or not, and what other consequences there may be. This is how we envisage the semantics assisting with collaboration, in that it provides a means of stating precisely what the (ideal or actual) behaviour of a specific component may be, as well as how the various components interact. This also means that the expression of a contract, i.e. the properties of one component of interest to another, will also need to be a central part of the semantics.

4 Approach

Our approach is to develop the semantics for each component (in our running example, the piston, lever, conveyor, and gripper) and then to compose the semantics of the overall system from those for each component. This will involve both the internal states for each component (with the associated transition conditions between states) and the interfaces between components. Our running example is fairly small, and in general it seems possible to handle the semantics for this system by hand. However, for a system with dozens of components, it is likely that some automated tools will be required to manage the scale involved. For this reason we anticipate using tools such as SMT solvers to manage the overall configuration of the system. In particular, we envisage a specification of a desired configuration of the system being mapped to a specific physical configuration, with the mapping between the two being found by the SMT solver.

It should be noted that we do not necessarily expect this process to be fully automatic; in fact, it is likely that this will only be feasible with some human input at critical points, especially for larger systems. In the extreme, there may even be a completely human specification that is checked by the solver, but in many cases a human expert may provide some partial input, such as the configuration for a particularly intricate or critical component, with the SMT solver being used to fill in the rest of the configuration task. It may also be that the solver will provide some choices, from which a team of humans may choose the most appropriate one. In this way the use of formal models with appropriate tool support can provide opportunities for collaboration and distributed development.

The use of a solver in this way means that it is possible for collaboration to take place across distances, with the semantics for each component being input to a solver in the cloud. It also means that it is not required that any one person in the team know the global properties of the system, but only the properties of the components for which they are responsible.

5 Related Work

Our demonstrator is based on Festo equipment, customized using Raspberry Pis as controllers running on the 4DIAC (Strasser et al. 2008) runtime environment. A similar demonstrator setup has been used in (Wenger et al. 2015) where formal models (behavioral types) serve as a specification basis for monitoring of an IEC 61499 based system. Moreover, similar specifications have been extended for cyber-

physical systems (Blech and Herrmann 2015). Ultimately, our work is intended to complement a framework that supports other aspects of collaboration such as spatio-temporal models and decision support based on them. Such a framework has been introduced in (Blech et al. 2015). (Blech et al. 2014) describes usages for specifications for similar demonstrators.

The V-model (as e.g., incorporated in the ISO 26262 standard (ISO26262) in the automotive domain) uses a divide and conquer approach for identifying and developing components. This provides some suggestion on possible teams, tasks and dependencies in the development of cyber-physical systems.

Interface automata (Alfaro and Henzinger 2001) are one form of component-based specifications that inspired the modeling formalisms proposed in this paper. Component descriptions in (Alfaro and Henzinger 2001) are based on timed automata. The focus of interface automata is on interactions between components and as in our work can serve as a basis for facilitating collaboration between different development teams. Behavioral types from the Ptolemy framework (Lee and Xiong 2004) are a modeling mechanism for components and are primarily aimed towards the software part of real-time systems such as execution time of code. External State Machines (Kraemer and Herrmann 2009) are based on UML state machines describing the interface behavior of software components or building blocks for the control of cyber-physical systems. Tangible user interfaces (Shaer and Hornecker 2010) provide another way of exemplifying underlying principles of cyber-physical specifications. Cyber-physical characteristics can be described using a behavioral specification, e.g., using a domain specific language.

Ways to use SMT solvers for generating PLC functionality have been described in (Cheng et al. 2014) (Cheng et al. 2012). In contrast to this, we are investigating solutions to facilitate the collaboration between different stakeholders. We propose the use of solvers to facilitate this. Formula is a framework for specifying systems and using SMT-based constraint solving to ensure properties [1] . The framework is specifically tailored towards the use of the z3 SMT solver (Moura and Bjøner). Modelica is another logic-based specification framework [2] which has been applied in a variety of contexts.

Several industry-academia collaborations have developed frameworks for component interfaces and aim at facilitating collaboration between different stakeholders for different system components. The combest project (Bensalem et al.) [3], Recomp [4], d-mils [5], and Aramis [6] all follow these goals providing some sort of component description language. Application domains include industrial automation, automotive engineering, avionics and trains.

Formula (Jackson et al. 2009) provides a generic modeling language for specifying properties of components and means to reason about them. Reasoning is based on SMT solving. The framework targets non-functional requirements of software systems.

Modelica (Fritzson 2010) is a language aimed towards the modeling of cyber-physical systems. Specification of industrial automation systems using a theorem prover has been investigated, e.g., in (Blech and Ould Biha 2011). The specification is based on the IEC 61131–3 standard for programming programmable logic controllers. In our work, we are not addressing a particular programming standard, but are experimenting with IEC 61131–3, IEC 61499 and Python code running on our Raspberry Pi based controllers.

6 Conclusion

This position paper is an outline of our plans for developing an appropriate semantics for reasoning about the properties of the Festo system. We will further develop the semantics for the Festo system described above, and to use it to investigate various properties. This will provide us with a specific scenario for the discussion of reconfiguration problems and related issues. We envisage using tools such as the SMT solver z3 for this purpose. We also plan to empirically derive requirements for collaborative scenarios based on an observational study of a project involving the Festo system.

REFERENCES

L. de Alfaro, T.A. Henzinger (2001). Interface Automata. In Symposium on Foundations of Software Engineering, ACM.

[1] http://research.microsoft.com/en-us/um/people/ejackson/

[2] https://www.modelica.org/documents/ModelicaSpec33Revision1.pdf http://book.xogeny.com

[3] http://combest.imag.fr/home

[4] https://artemis-ia.eu/project/21-recomp.html

[5] http://www.d-mils.org/page/publications-1

[6] http://www.projekt-aramis.de/projekt/ziele

R. Alur, D. Dill (1994, April). A theory of timed automata. Theoretical Computer Science 126(2):183-235.

S. Bensalem, M. Bozga, T.-H. Nguyen, J. Sifakis (2010, June). Compositional verification for component-based systems and application. In Software, IET, vol.4, no.3, pp.181-193. doi: 10.1049/iet-sen.2009.0011

J. O. Blech, M. Spichkova, I. Peake, H. Schmidt (2014). Cyber-Virtual Systems: Simulation, Validation & Visualization. Evaluation of Novel Approaches to Software Engineering. SciTePress ISBN 978-989-758-030-7.

J. O. Blech, P. Herrmann (2015). Behavioral Types for Space-aware Systems. Model-based Architecting of Cyber-Physical and Embedded Systems. CEUR proceedings, vol. 1508.

J. O. Blech, S. Ould Biha (2011). Verification of PLC Properties Based on Formal Semantics in Coq. 9th International Conference on Software Engineering and Formal Methods (SEFM), Montevideo, Uruguay, vol. 7041 of LNCS, Springer.

J. O. Blech, I. Peake, H. Schmidt, M. Kande, A. Rahman, S. Ramaswamy, S.D. Sudarsan, V. Narayanan (2015). Efficient Incident Handling in Industrial Automation through Collaborative Engineering. Emerging Technologies and Factory Automation (ETFA), IEEE.

C.H. Cheng, C.H. Huang, H. Ruess, S. Stattelmann (2014, January). G4LTL-ST: Automatic generation of PLC programs. In Computer Aided Verification (pp. 541-549). Springer International Publishing. ISO 690

C.H. Cheng, M. Geisinger, H. Ruess, C. Buckl, A. Knoll (2012, January). MGSyn: Automatic synthesis for industrial automation. In Computer Aided Verification (pp. 658-664). Springer Berlin Heidelberg.

P. Fritzson (2010). Principles of object-oriented modeling and simulation with Modelica 2.1. ohn Wiley & Sons. ISO 690

ISO 26262 standard (2011), ISO, Geneva.

E.K. Jackson, D. Seifert, M. Dahlweid, T. Santen, N. Bjørner, W. Schulte (2009, January). Specifying and composing non-functional requirements in model-based development. In Software Composition (pp. 72-89). Springer Berlin Heidelberg.

F. A. Kraemer and P. Herrmann (2009). Automated Encapsulation of UML Activities for Incremental Development and Verification. In Model Driven Engineering Languages and Systems (MoDELS), vol. 5795 of LNCS, pages 571–585. Springer-Verlag.

E. A. Lee, Y. Xiong (2004). A Behavioral Type System and its Application in Ptolemy II. Formal Aspects of Computing, 16(3):210–237.

L. Moura, N. Bjørner (2008). Z3: An Efficient SMT Solver. In Proceedings of the 14 International Conference on Tools and Algorithms for the Construction and Analysis of Systems (pp. 337-340).

Raspberry Pi. `https://www.raspberrypi.org`. Accessed December 2015.

H. Schmidt et al. (2003). Modelling predictable component-based distributed control architectures. In the Ninth IEEE International Workshop on Object-Oriented Real-Time Dependable Systems, 2003. WORDS 2003 Fall.

O. Shaer, E. Hornecker (2010). Tangible user interfaces: past, present, and future directions. Foundations and Trends in Human-Computer Interaction 3(!-2):1-137.

T. Strasser, M. Rooker, G. Ebenhofer, A. Zoitl, C. Sünder, A. Valentini, A. Martel (2008, July). Framework for distributed industrial automation and control (4DIAC). In Industrial Informatics, 2008. INDIN 2008. 6th IEEE International Conference on (pp. 283-288). IEEE.

M. Wenger, J. O. Blech, A. Zoitl. Behavioral Type-based Monitoring for IEC 61499 (2015). Emerging Technologies and Factory Automation (ETFA), IEEE.