THESIS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

Preparation and control of intelligent automation systems

A goal-oriented automation framework

Martin Dahl

Department of Electrical Engineering
Chalmers University of Technology
Gothenburg, Sweden, 2021

Preparation and control of intelligent automation systems
A goal-oriented automation framework

Martin Dahl

ISBN: 978-91-7905-446-5
Series number: 4913
ISSN 0346-718X
This thesis has been prepared using LaTeX.

Department of Electrical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg, Sweden
Phone: +46 (0)31 772 1000
`www.chalmers.se`

Printed by Chalmers Digitaltryck
Gothenburg, Sweden, January 2021

To Elmer.

Abstract

In the automation systems of tomorrow, it is likely that the devices included have various degrees of autonomy, and include advanced algorithms for perception and control. Human operators will be expected to work together with collaborative robots as well as with roaming robots for material handling.

The volatile nature of the environment of such intelligent automation systems lead to an enormous amount of possible situations that can arise and which need to be suitably handled. This complexity makes development of control systems for intelligent automation systems difficult using traditional methods.

As an alternative, this thesis presents a model-based control framework, which uses a combination of formal specification and automated planning. The proposed framework allows for defining the intentions of the automation system on a high level, which enables decisions that influence when things should occur to be modeled using logical constraints, rather than programming. To achieve a modular framework, low level, reusable, resource models are composed by *1)* formal specification to ensure safety and *2)* applying an abstraction called an operation, which couples the reusable resources to the intentions of the system. By planning also the resources' detailed actions, the operations can, when possible, be completed regardless of the resources' current state. This eases error-recovery, as resources do not have to be reset when an error occurs.

Additionally, the thesis proposes an iterative and interactive workflow for integrating the proposed model-based control framework into a virtual preparation process, using computer based simulation as a tool for validating formal specifications. The control framework allows for adding new constraints to a running system, enabling an efficient and interactive preparation process.

The framework has been applied to a use case from final assembly, which features human-robot collaboration. Experimental results on the ability to handle unforeseen errors and planning performance are presented.

Keywords: Control Architectures, Automation, Planning and Coordination, Robotics.

List of Publications

This thesis is based on the following publications:

[A] **Martin Dahl**, Kristofer Bengtsson, Patrik Bergagård, Martin Fabian, and Petter Falkman, "Integrated virtual preparation and commissioning: supporting formal methods during automation systems development". Proceedings of the 8:th IFAC Conference on Manufacturing Modeling, Management & Control.

[B] **Martin Dahl**, Kristofer Bengtsson, Martin Fabian, Petter Falkman, "Guard extraction for modeling and control of a collaborative assembly station". To appear in the proceedings of the 15:th Workshop on Discrete Event Systems, WODES'20.

[C] **Martin Dahl**, Endre Erős, Kristofer Bengtsson, Martin Fabian, Petter Falkman, "Sequence Planner: A framework for control of intelligent automation systems". Submitted to Robotics and Computer-Integrated Manufacturing.

[D] **Martin Dahl**, Christian Larsen, Endre Erős, Kristofer Bengtsson, Martin Fabian, Petter Falkman, "Interactive formal specification for efficient preparation of intelligent automation systems". Submitted to CIRP Journal of Manufacturing Science and Technology.

[E] **Martin Dahl**, Endre Erős, Kristofer Bengtsson, Martin Fabian, Petter Falkman, "Application of the Sequence Planner control framework to an intelligent automation system with a focus on error handling". Submitted to Machines, special issue on Mechatronic System for Automatic Control.

Other publications by the author, not included in this thesis, are:

[F] **Dahl, M.**, Bengtsson, K., Bergagård, P., Fabian, M., and Falkman, P., "Sequence planner: Supporting integrated virtual preparation and commissioning.". *IFAC-PapersOnLine, 50(1), 5818-5823*, 2017.

[G] Virtual reality commissioning in production systems preparation., "**Dahl, M.**, Albo, A., Eriksson, J., Pettersson, J., and Falkman, P.". *In 2017 22nd*

IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), (pp. 1-7). IEEE., Sept. 2017.

[H] Automatic modeling and simulation of robot program behavior in integrated virtual preparation and commissioning., "**Dahl, M.**, Bengtsson, K., Fabian, M., and Falkman, P.". *Procedia Manufacturing*, 11, 284-291, 2017.

[I] Digital Twin for Legacy Systems: Simulation Model Testing and Validation, "Khan, A., **Dahl, M.**, Falkman, P. and Fabian, M.". *2018 IEEE 14th International Conference on Automation Science and Engineering (CASE)*, Munich, (pp. 421-426), 2018.

[J] A ROS2 based communication architecture for control in collaborative and intelligent automation systems., "Erős, E., **Dahl, M.**, Bengtsson, K., Hanna, A., and Falkman, P.". *Procedia Manufacturing*, 38, (pp. 349-357), 2019.

[K] Industrial Challenges when Planning and Preparing Collaborative and Intelligent Automation Systems for Final Assembly Stations., "Hanna, A., Bengtsson, K., **Dahl, M.**, Erős, E., Götvall, P. L., and Ekström, M.". *In 2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, (pp. 400-406). IEEE, Sept. 2019.

[L] Integrated virtual commissioning of a ROS2-based collaborative and intelligent automation system., "Erős, E., **Dahl, M.**, Hanna, A., Albo, A., Falkman, P., and Bengtsson, K. ". *In 2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, (pp. 407-413). IEEE, Sept. 2019.

[M] Control components for Collaborative and Intelligent Automation Systems., "**Dahl, M.**, Erős, E., Hanna, A., Bengtsson, K., Fabian, M., and Falkman, P.". *In 2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, (pp. 378-384). IEEE, Sept. 2019.

[N] A framework concept for data visualization and structuring in a complex production process., "Albo, A., Bengtsson, K., **Dahl, M.**, and Falkman, P.". *Procedia Manufacturing*, 38, 1642-1651, 2019.

[O] Development of an Industry 4.0 Demonstrator Using Sequence Planner and ROS2, "Erős, E., **Dahl, M.**, Hanna, A., Götvall, P. L., Falkman, P., and Bengtsson, K.". *In Robot Operating System (ROS)*, (pp. 3-29). Springer, Cham, Jun 2020.

[P] Towards compositional automated planning, "Erős, E., **Dahl, M.**, Falkman, P., and Bengtsson". *In 2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, (pp. 416-423). IEEE, Sept. 2020.

Acknowledgments

It has been almost a year now since we all took our laptops and said goodbye to the office. A strange way to end a five year PhD. Thankfully, discussions could continue over the Internet. I would like to give a big thank you to my supervisor Petter Falkman for remaining positive and encouraging all the way from start to finish. It really helped. *I thank* my co-supervisor Martin Fabian for never giving up on teaching me how to write. Though I never did learn to be as precise and unambiguous as only you can be. Perhaps we need to write more papers together. Special thanks to Kristofer Bengtsson, co-supervisor in all but on paper. We've had a lot of fun programming the last few years. We also learned a thing or two. Thanks to Christine Johansson for always being both super friendly and super helpful. Thanks to my office mates, past and present, and the ones in the offices next to ours. You know who you are! It's been a lonely year without the coffee breaks. Endre, you get a special mention for being such a genuinely nice person. Also, we still have work to do together! Another special mention goes to Sarmad Riazi, I know you would be disappointed to be left out of this section. I hope you are doing well!

Thanks to Atieh Hanna and Per-Lage Götvall at AB Volvo. It's been a wonderful collaboration. Thank you Christian Larsen at FCC, it is always a pleasure to work with you.

A special thank you goes to my parents Lennart and Marie for helping me out with daily life tasks during the last few months. Don't worry, you will still be welcome to do that.

Finally, thank you Josefin for your support and understanding. It may well be that you worked harder than me to get this thesis finished ♥.

Acronyms

AI:	Artificial Intelligence
CTL:	Computation Tree Logic
EFA:	Extended Finite Automata
FM:	Formal Methods
LTL:	Linear Temporal Logic
MC:	Model Checking
OLP:	Off-Line Programming
PDDL:	Planning Domain Definition Language
PLC:	Programmable Logic Controller
SCT:	Supervisory Control Theory
SP:	Sequence Planner
ROS:	Robot Operating System
VC:	Virtual Commissioning

Contents

Part I

Overview

CHAPTER 1

Introduction

If there was ever an interesting time to work in an *automation* research group, now is not a bad time! During my time as a PhD student collaborative robots have become commonplace and there have been significant advances in computer perception and robotics. At the same time, the hype around artificial intelligence (AI) has been increasing every year.

Given these current trends it seems clear that the automation systems of tomorrow will most likely look dramatically different from the installations that exist in industry today. The complexity in system size, software, algorithms, and know-how required to work with the automation systems could be orders of magnitudes higher than today to encompass these new technologies and require completely different skills.

In industrial automation, the motto of the engineering work has been, and still is, the KISS principle: *keep it simple, stupid.* The meaning of this design mantra, which has its origins in the engineering departments of the U.S. Navy in the 1960:s, is that a simple design usually results in a robust engineering solution. No-one cares for over-engineered and complex solutions that cannot be fixed with a screw-driver and pressing a "reset" button. So how can we accommodate the just mentioned new technologies, many of which *are* very

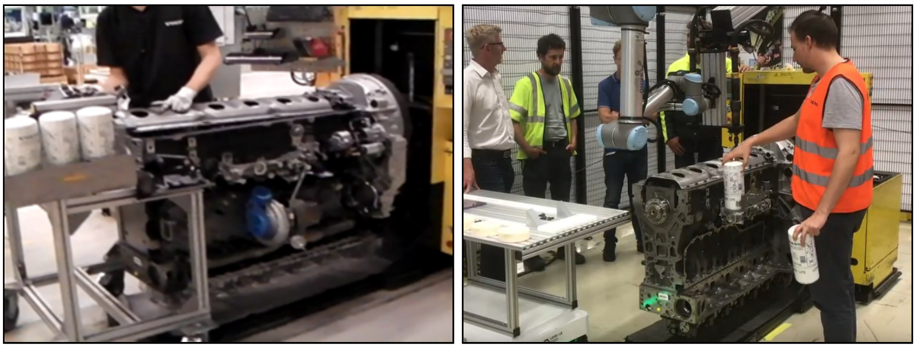

(a) The original manual assembly station **(b)** Collaborative robot assembly station. A video clip from the demonstrator: `https://youtu.be/TK1Mb38xiQ8`

Figure 1.1: Transformation of a manual assembly station into a collaborative one.

complex, into not only a simple-to-use system that can aid the users when something goes wrong, but also into a simple-to-develop system?

This thesis highlights our work on preparation and control of automation systems, especially in the context of production systems involving collaborative robots and autonomous subsystems. We classify this type of automation systems as *intelligent automation systems*. Specifically, the thesis proposes a framework for creating control systems for such intelligent automation systems, which tackles high complexity by relying on formal models of the behavior of the different devices in the system. Complexity can then, to some degree, be handled by computation rather than human reasoning. The framework is based on a notion of *goals* that are continuously distributed to all devices in the automation system, which lets some of the devices take their own decisions about how to reach these goals. The framework supports preparation of these intelligent automation systems in an iterative way, which relies on computer based simulation to validate the correctness of the formal specifications.

The work is motivated by the industrial need that these intelligent automation systems create, as well as the interesting research challenges that arise from trying to satisfy these needs. A good example of this is one of the use cases developed as part of the present research.

This particular case involves conversion of an existing manual assembly station from a truck engine final assembly line, shown in Figure 1.1a, into an intelligent and collaborative robot assembly station, shown in Figure 1.1b. The aim of the conversion work was to have a robot and an operator work together to perform assembly of the engine side by side, sharing tools hanging by wires from the ceiling. A dedicated camera system keeps track of operators, ensuring safe coexistence with machines. It is desired to let a human operator *be human*, and not simply used as an additional cog in the machine. For the automation system to be able to reach specific goals (in this case, completing assembly tasks) in an environment together with a human operator, it needs to be able to take the intentions and actions of the human operator into account. That is, the automation system may need to adapt the strategy used to reach its goals depending on what the human operator does. The system may *ask* the operator for aid, but should strive towards being independent and keep out of the way.

1.1 Preparation and control of automation systems

There is a large number of activities involved in the *preparation* of an automation system like the one described in the previous section, for example: identifying which processes that are necessary to perform, identifying the tools and machinery needed, physical layout of the devices in the system, and programming a computer system which can *control* the devices that make up the system. These activities can all benefit from computation and simulation support based on computer models [1]. In this work, these *computer assisted* activities are referred to as *virtual preparation* of automation systems.

Looking back at logical control performed by networks of electrical relays, *control* of man-made automation systems has been used in industry for over 100 years. In some ways, how to program such automation systems has stayed remarkably unchanged, owing to the fact that the main programming language for programmable logic controllers (PLC:s), called *ladder*, was designed with the explicit intention of emulating the existing control by relay logic [2].

The automation systems of today are expected to be modular, reconfigurable, and distributed. Many initiatives, both in academia and industry have pushed for “smarter” automation. Some very broad like Industry 4.0 [3], but also more focused standardization efforts like IEC 61499 [4] have tried to enable more intelligent automation.

1.2 Intelligent automation

To define what we mean by intelligent automation, it is useful to start to define what we mean by intelligence. The Merriam-Webster online dictionary defines “intelligence” as [5]:

in • tel • li • gence noun

1. the ability to learn or understand or to deal with new or trying situations : reason
   also: the skilled use of reason
2. the ability to apply knowledge to manipulate one’s environment or to think abstractly as measured by objective criteria (such as tests)

Essentially, it describes the ability to learn new things (knowledge), and to be able to apply this knowledge to solve tasks in one’s environment. Thomas Malone defines two categories of intelligence in terms of being able to reach goals [6]:

- **Specialized intelligence:** “the ability to achieve specific goals effectively in a given environment”
- **General intelligence:** “the ability to achieve a wide range of different goals effectively in different environments.”

When we write *intelligence* (often *intelligent*) in this thesis, we refer to Thomas Malone’s definition of *specialized intelligence.* Using this definition, we can define an intelligent automation as a system that can effectively reach specific goals in a given environment.

From this we can derive a few challenges and differences with traditional automation systems:

- In an intelligent automation system, goals of the system can change based on a changing environment. – In traditional automation systems they can not.

  The intelligent automation system is *adaptive.*
- An intelligent automation system should be able to reach its goals in many different ways. – In traditional automation systems there exist only a few ways (maybe one) to reach the goals.

The intelligent automation system is *flexible.*

- An intelligent automation system should be able to interact and cooperate with the environment, including reasoning about and planning future actions together with human operators. – A traditional automation system can only perform cooperative tasks that have been explicitly programmed.

  The intelligent automation system is *deliberative.*

- In an intelligent automation system things *are expected* to change unexpectedly, so the automation system needs be able to find other ways to reach the goal. – In traditional automation systems unexpected events result in failure and may require resetting the system back to an initial state.

  The intelligent automation system is *reactive.*

- An intelligent automation system can learn new things about the environment and use the new knowledge to adapt its strategies. – A traditional automation system does not learn.

  The intelligent automation system is *learning.*

1.3 Research questions

One way to formulate research questions is to synthesize an industrial need with a research challenge. Given the needs and research challenges described previously, three research questions are defined in this section.

RQ1 *How could a control framework for an intelligent automation system be designed?*

As we have seen before, the control system would have to be *adaptive, flexible, deliberative, reactive*, and *learning.* Exemplified by the use-case described earlier, it should be aware of the human operator and be able to change its behavior in response to what the operator is doing. For example, it should be able to reach its goals even though the human has performed tasks “out-of-sequence”.

RQ2 *How can we use model-based techniques to reduce complexity when preparing control systems for intelligent automation systems?*

The ability to reach the goals in an intelligent automation system in many different ways rules out traditional programming for the control systems, as the amount of cases to handle simply become too many. In the use-case described earlier, it should not matter where the tools are, or the robots, or the operator – the goals should be achievable regardless.

RQ3 *Could an interplay between simulation and formal methods provide a foundation for an iterative preparation methodology?*

In an intelligent automation system it will be difficult to anticipate (during preparation) the different situations that may or may not arise. The intelligent automation system is *complex*, and development of any complex system is inherently a creative and iterative process.

A missing keyword here is the *learning* of the automation system. Learning is not covered in this thesis, but it could be an excellent research topic for future work.

1.4 Research approach

The research described in this thesis is to a large extent application oriented. As such, much of the research activity is spent on implementing experiments to show the applicability of the proposed methods to modeling and control of automation systems. During this work the hypotheses often change – slightly different approaches are tried and tested in order to reach satisfying results. As such, at times the line between research and solving engineering challenges is hard to distinguish. It is the author's view however, that practical implementation constitutes an important part of the research method, the tweaks and changes to the approach that are made in order to make the demonstrator actually work is often where the largest insights are made – curiously these tend to coincide with the researcher being proven wrong.

1.5 Research journey

Integrated virtual preparation

The research journey of the author started with the VIRTCOM-II project, which started in 2014 and ended in the fall of 2017. According to the project web page [7]: *"The VirtCom project aims to result in a completely integrated work chain from virtual preparation, through automatically generated PLC code, to virtual and physical commissioning."* The description continues: *"Having such an integrated work chain, with software tools for optimization and verification, achieves sustainable production facilities adaptable to future requirements on flexibility, availability, product variety, human safety, etc."*.

One idea prevalent in industry on how to cope with these issues is to use computer based simulation for testing and validation of the automation system. If a production system is meticulously modeled on the level of sensors and actuators, it is possible to control a simulation of this model using the same hardware and software that will eventually control the real system. Pairing an actual industrial control system with such a simulation is called virtual commissioning (VC) [8]. During the past decade VC has matured and become a standard engineering task at many production facilities and automation solution companies. It has primarily been viewed as an extension of the current workflow – a final step in the automation engineering process that enables additional validation.

Another idea for increasing quality is to carefully verify and validate the developed control software, preferably automatically. Control of modern production systems, especially in the automotive industry, is still very much based on programmable logic controllers (PLC:s), due to the real time constraints of production and the need for high capacity I/O. Despite a large research body on formally verifying PLC software (e.g. [9]) industrial uptake of such methods is still at low levels [10].

By combining existing work on model based development of production system logic with VC validation, the contribution of this author towards such an integrated work chain is described in Paper A, and Paper D, included in this thesis. During the VITCOM-II project the author additionally contributed to development of VR-based VC [11], and applying the proposed work chain to automatically create formal models of existing robot station behavior [12], [13].

Human-robot collaboration, complex subsystems, and degrees of autonomy

After VIRTCOM-II had finished, the author subsequently started to work in a project called UNIFICATION, which was started in 2017 and ended in the fall of 2020. In project UNIFICATION, much of the work revolved around the previously described conversion of an existing assembly station in the AB Volvo Skövde plant, pictured in Figure 1.1. Volvo Group Trucks Operations has defined the following vision to better influence the research and development of next generation automation systems: Future Volvo factories [14], will be *re-configurable* and *self-balancing* to better handle rapid changes, production disturbances, and diversity of products. Collaborative robots and other machines can support operators at workstations, where they use the same smart tools and are interchangeable with operators when it comes to performing routine tasks. Operators are supported with mixed reality technology, and are provided with digital work instructions and 3D geometry while interacting intuitively with robots and systems. Workstations are equipped with smart sensors and cameras that feed the system with real-time status of products, humans, and other resources in the environment. Moreover, they are supported by advanced yet intuitive control, dynamic safety, online optimization and learning algorithms.

The work in UNIFICATION resulted in re-thinking how to design control systems that need to feature these new technologies. The work has resulted in a number of publications in addition to the ones included in this thesis, among them work on formalizing the requirements for human robot collaboration in assembly [15], defining the challenges in implementing such systems [16], designing communication between the different sub-systems [17], virtual preparation of the system [18], and application of model based control design [19].

1.6 Contributions

The thesis presents the following contributions, which we connect to the already mentioned research questions below.

Paper A introduces a framework for modeling discrete control systems in an iterative fashion, with VC models being used earlier in the preparation work for validation of formal specifications. While the paper has a small example, the ideas presented in that paper is used and built upon in all the remaining work, and is again given focus in Paper D. In Paper D, we apply the same workflow to help us write formal specifications for an intelligent automation system. These two papers both contribute to knowledge around *RQ3*.

Paper B shows, with the help of a practical example, how, given a formal representation of the possible actions that can be taken by the resources in an automation system, simple specifications can help ease the modeling workflow by moving the formal specifications out from the resources, increasing reusability of the involved resource components. This is also something which was alluded to as a goal in Paper A and expanded on in Paper C. These contributions relate to *RQ2*.

The main contribution of Paper C is the insight that, when working with intelligent automation systems, there is a need to merge traditional automation with automated planning and reasoning in a seamless way. While automated planning can help deal with complexity, current automated planning frameworks often deal with a single agent that takes decisions for themselves, or sending out commands to agents one at a time. In this paper we propose a framework for control of intelligent automation systems where we allow for intricate interdependencies between resources to be modeled using the formal specifications from Paper A, B, and D. Paper E adds additional insight into how the framework can handle various error situations. Paper C and Paper E contribute to knowledge that can help answer *RQ1*.

1.7 Outline

This thesis consists of two parts. Part I is a general introduction to the field and puts the appended papers into context. Part II contains the appended papers. Part I is organized as follows: Chapter 2 gives a background to discrete control of automation systems in general terms, but it also gives some important definitions that will be used throughout the thesis. Chapter 3 gives an overview of the computational tools used in this thesis and their semantics. In Chapter 4, we discuss system integration in intelligent automation systems. The main results of the appended papers are summarized in Chapter 5. Chapter 6 contains a summary of the appended papers. The thesis ends with some closing remarks and directions for future work in Chapter 7.

CHAPTER 2

Discrete control of automation systems

In this chapter we give a basic overview of the challenges of developing robust and flexible automation systems and try to give a sense of the trade-offs that need to be made during development. Additionally, an example is given which is used to explain different concepts throughout the thesis.

Automation systems engineering is fundamentally about managing *state*. In contrast to other types of computer programming, where we need to make sure that our algorithms are correctly implemented, when programming control systems for automation systems, we need to correctly manage the *physical* state of all resources involved in the system, such as industrial robots, motors on conveyor belts, sensors, as well as the state of all products in the system. Some state can be actively measured, while some needs to be estimated based on past actions and observations. The state of these resources will be distributed physically and needs to be communicated in some way to a *control system*.

We can view the problem of developing a robust automation *control system* as essentially computing the best next actions to be taken given the current state of the automation system. However, given that there may be an infinite number of different states the automation system can be in, this is quite difficult, if even possible.

So, we naturally start abstracting the problem. Consider a control system for the automation system illustrated in Figure 2.1, which shows a conveyor belt, a product of some kind that is transported on the conveyor, a robot, and a bin. Instead of representing where an industrial robot is in space and what the velocities of its joints are, perhaps we simply take note of which robot program is currently executing, and maybe if the robot has signaled an error. Instead of using an encoder to keep track of how far the conveyor belt has moved, perhaps we only keep track of a sensor which can detect if there is a at the end of the conveyor.

And so the system is abstracted and discretized, but still there are many more possible states than we can manage to deal with as human developers. As one last simplification, we decide on one particular state as our *initial* state. Finally! We can reason about the system; given that the conveyor is stopped, a product is at the beginning of the conveyor and the robot is idle, we can now determine the possible valid actions, and perhaps even the "best" action that we want the control system to take in this particular state.

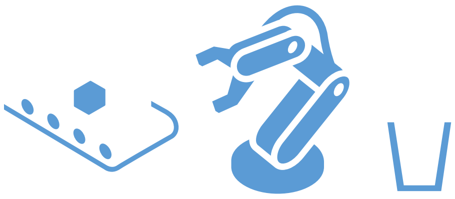


Figure 2.1: Example automation system consisting of a conveyor belt, a robot, and a bin. A product is being transported on the conveyor.

Let us imagine that one of the tasks of the automation system is to move the product on the conveyor from the beginning of the conveyor to the end of it, where the robot can pick it up and put it in the bin. One way to program this behavior is to define one action that moves the product to the end of the conveyor, one were the robot picks up the product, and one where the robot puts the product in the bin. Then we can say that whenever we start the system, we should run the sequence of actions *move product to end of conveyor*, followed by *robot pick up the product*, followed by *robot put product in bin*. Piece by piece the control system can then be built up by reasoning about different sequences that can begin in different starting states.

Or, instead of reasoning about sequences, we can reason directly about the underlying system state. For example: if a product is detected at the start of the conveyor, the conveyor is stopped, and the robot is idle: start conveyor. If a product is detected at the end of the conveyor and the robot is idle; run the program that picks up the product. When the robot has finished picking up the product, run the program that puts the product in the bin.

Both approaches are common and often a combination is used to solve the problem at hand. Control system engineers are skilled in this way of thinking, and combined with best practices and standardized interfaces developed over the years, it has been possible to develop the large and complex automation solutions used in industry today. Nevertheless, developing a robust automation system is a challenging task.

2.1 States and state variables

As the *state* of the system is of fundamental importance, let's define it a bit more formally. The convention used in this thesis is to represent the state of the control system in terms of *state variables*:

Definition 1: *A state variable v has a domain $V = \{x_1, \ldots, x_n\}$, where each element in V is a possible value of v.For simplicity, we will use the following notation when defining a variable: $v = \{x_1, \ldots, x_n\}$.*

In practice, an enumeration $\{x_1, \ldots, x_n\}$ can be represented by a combination of m binary variables to encode an integer so that $n \leq 2^m$.

State variables are used when describing the state of a system, where a state is defined as follows:

Definition 2: *A state S is a set of tuples $S = \{\langle v_i, x_i\rangle, \ldots, \langle v_j, x_j\rangle\}$, where v_i is a state variable and $x_i \in V_i$ is the current value of the state variable. Each state variable can only have one value at a time.*

Example 2.1: State of a control system

For the example in Figure 2.1 with a the product, a conveyor with a sensor for the product at each end and a motor, and robot which can pick up products from the conveyor and drop them in the bin, a simple representation of the state using only boolean variables could be:

- $\mathit{conveyor}_1 = \{\mathit{false}, \mathit{true}\}$. The sensor at the beginning of the conveyor, true means a product is detected.
- $\mathit{conveyor}_2 = \{\mathit{false}, \mathit{true}\}$. The sensor at the end of the conveyor, true means a product is detected.
- $\mathit{conveyor}_m = \{\mathit{false}, \mathit{true}\}$. Output to the motor, true means the motor is on, which runs the conveyor belt forward, false means the motor is off.
- $\mathit{pickup}_s = \{\mathit{false}, \mathit{true}\}$. An output to the robot which signals that it should run a program that picks up a product.
- $\mathit{dropoff}_s = \{\mathit{false}, \mathit{true}\}$. An output to the robot which signals that it should run a program that drops the product it is holding in the bin.
- $\mathit{pickup}_f = \{\mathit{false}, \mathit{true}\}$. An input from the robot which signals that the program that picks up the product has finished.
- $\mathit{dropoff}_f = \{\mathit{false}, \mathit{true}\}$. An input from the robot which signals that the program that drops off the product has finished.

Then the *initial state* of the system described above is $S_0 = \{\langle \mathit{conveyor}_1, \mathit{true}\rangle, \langle \mathit{conveyor}_2, \mathit{false}\rangle, \langle \mathit{conveyor}_m, \mathit{false}\rangle, \langle \mathit{pickup}_s, \mathit{false}\rangle, \langle \mathit{dropoff}_s, \mathit{false}\rangle, \langle \mathit{pickup}_f, \mathit{false}\rangle, \langle \mathit{dropoff}_f, \mathit{false}\rangle\}$.

In Example 2.1, there are $2^7 = 128$ possible states. If the size of the system is doubled (i.e. an additional conveyor and robot is added) there will be $128^2 = 16384$ possible states. It is this combinatorial growth that makes it hard to develop robust and flexible automation control systems. It is also the reason why choosing one or only a few initial states to start from reduces the complexity. Then the *reachable* states will only be those that can be reached from the initial states, which depends on what we allow the system to do in the different states. The fewer actions we allow the system to take in each state, the easier it is to reason about the system.

The trade-off here is quite clear: increasing flexibility (i.e. allowing the system to be in more states) is very costly in terms of reasoning about the system.

2.2 Actions and transitioning between states

In the above section, the *actions* taken by the system are very loosely defined. What do we mean by "start the conveyor" or "robot should pick up the product"?

In this thesis we reason about the actions in terms of how the actions update the system state. If the system is in some state S_1 it can *transition* to the state S_2 by the action a. Usually, an action can only happen in a subset of the system states. Consider the example in Figure 2.1 again: "if a product is detected at the start of the conveyor, the conveyor is stopped, and the robot is idle: start conveyor". To model this using actions, it is necessary to be able to "guard" the action so that it can only happen in the appropriate states. In this thesis we denote a *guard* and an *action* as *a transition.*

Definition 3: *A transition t has a guard g, which is a boolean function over a state, $g\colon S \to \{false, true\}$, and a set of action functions A where $a\colon S \to S$ which updates the assignments to the state variables in a state. We often write a transition as g/A to save space.*

Given a set of state variables and a set of transitions, it is possible to evolve a given state S into S', S'', etc. by repeatedly applying the actions of transitions whose guards evaluate to true in the current state. In the appended papers, we often write that a transition is *taken.* Transitions are taken one at a time.

Example 2.2: Actions of a control system

Continuing the example in 2.1, a transition that models "if a product is detected at the start of the conveyor, the conveyor is stopped, and the robot is idle: start conveyor", could be expressed as: $conveyor_1 \wedge \neg conveyor_m \wedge \neg pickup_s \wedge \neg dropoff_s \;/\; conveyor_m := true$, assuming that the robot is idle when both of $pickup_s$ and $dropoff_s$ are false.

We write $conveyor_1$ as shorthand for $conveyor_1 = true$ and $\neg conveyor_2$ as shorthand for $conveyor_2 = false$. Applying the transition to $S_0 = \{\langle conveyor_1, true\rangle, \langle conveyor_2, false\rangle, \langle conveyor_m, false\rangle, \langle pickup_s, false\rangle, \langle dropoff_s, false\rangle, \langle pickup_f, false\rangle, \langle dropoff_f, false\rangle\}$ yields a new state $S_1 = \{\langle conveyor_1, true\rangle, \langle conveyor_2, false\rangle, \langle conveyor_m, true\rangle, \langle pickup_s, false\rangle, \langle dropoff_s, false\rangle, \langle pickup_f, false\rangle, \langle dropoff_f, false\rangle\}$.

2.3 Effect transitions

In this thesis we make a distinction between transitions that the control system takes and transitions that are only there to inform us of what is expected to happen in the real world. We denote the later type of transition *effect transitions*. This is discussed more in Chapter 5. In other literature it is common to denote the set of effect transitions as a model of the *environment* or a *plant model*. We will continue to implement our example automation system to see why this is important.

Recall that we had the following control strategy in mind. *1)* If a product is detected at the start of the conveyor and the conveyor is stopped and the robot is idle: start conveyor. *2)* If a product is detected at the end of the conveyor and the robot is idle: run the program that picks up the product. *3)* When the robot has finished picking up the product, run the program that drops the product in the bin.

Given the transition defined in Example 2.2, the first part of implementing the strategy is already done. We introduce another transition, $conveyor_2 \wedge \neg pickup_s \wedge \neg dropoff_s \;/\; pickup_s := true$. The robot is considered idle if no start signal have been given. This allows the second sentence in the example to be handled.

Finally, $pickup_f \wedge \neg pickup_s \wedge \neg dropoff_s \;/\; dropoff_s := true$ implements the third part of the strategy. This starts the robot program that drops the product in the bin upon finished the pickup program.

Consider now that we would like to *test* that the logic of our automation system is correct, before deploying it in the real world. We can test the system by placing it in a certain state and checking if the intended action is taken, i.e. checking that no bugs where introduced while translating the description of the indented behavior into computer executable code. But how would we test whether the indented outcome is reached from the initial state, through the different actions that should be taken and ending up with the robot dropping the product in the bin? An assumption made when we defined the control actions above is that if the first sensor on the conveyor detects the product, and the motor driving the conveyor is on, the second sensor will eventually detect the product.

One way to test that the logic is correct is to attach the developed control system to a simulated version of the automation system, which would contain virtual mirrors of the sensors, conveyor, and the robot. The simulation would simulate the physics of the product moving on the conveyor belt as well as the robot motions. Then, after starting the conveyor motor, the physics simulation will eventually register that the product has moved to the second sensor. Now the control system can execute its next action, picking up the product with the robot. This requires detailed modeling of the physical reality on the level of sensors and actuators. Software tools for performing such simulations are described in Chapter 4.

Another way to test that the logic is correct is to model the movement of the product as a transition. Perhaps $conveyor_1 \wedge conveyor_m \ / \ conveyor_2 := true$? This is however lacking information about the time it takes for the product to travel across the conveyor to the other sensor – by applying this transition the product would instantly appear at the far end of the conveyor. While sufficient to test this simple scenario, it is not ideal. For a more complete example it is most likely important to know that a product is on the conveyor, *somewhere* between the sensors. To model this we add an additional boolean state variable to our control system, let's name it *product_between*, and modify the transition which starts the motor to set *product_between* to true in its actions. Then we define an *effect transition* as $product\_between \wedge conveyor_m \ / \ conveyor_2 := true, product\_between := false$, which models that if there is a product between the sensors and the motor is still running, the product will arrive.

Similarly, effect transitions can be modeled for the robot finishing its tasks (for example $pickup_s := true \ / \ pickup_f := true$), as well as modeling that the second sensor will stop detecting the product when it has been picked up.

Example 2.3: Actions of a control system

With the effect transitions it becomes possible to run a (discrete) simulation to see if it is always the case that given the initial state, we will end up with the robot holding the product. In the following table, $conveyor_1$, $conveyor_2$, and $conveyor_m$ are abbreviated to c_1, c_2, and c_m. $pickup_s$, $dropoff_s$, are abbreviated to p_s and d_s, as well as $pickup_f$ and $dropoff_f$, to p_f and d_f. Finally *true* is abbreviated to t and *false* to f.

c_1	c_2	c_m	p_s	d_s	p_f	d_f	pb	action
t	f	f	f	f	f	f	f	*start motor*
f	f	t	f	f	f	f	t	*sense at conveyor 2*
f	t	f	f	f	f	f	f	*robot start pick up*
f	f	f	t	f	f	f	f	*robot finish pick up*
f	f	f	t	t	f	f	f	*robot start drop off*
f	f	f	t	f	t	f	f	*robot finish drop off*
f	f	f	t	f	t	t	f	-

The type of check performed in Example 2.3 can be done using a variety of *formal methods*, which will be discussed further in Chapter 3. These methods can mathematically prove whether some property always hold. The example high-lights that in order to be able to reason about the system without simulating the dynamics, the *environment* needs to be properly modeled.

CHAPTER 3

Formal methods

This chapter continues where the previous chapter left off and gives a brief introduction to the different formal methods used in the appended papers. Specifically, we present supervisory control theory (SCT) and model checking (MC). Additionally, bounded model checking (BMC) is presented as well as how MC can be used also for automated planning.

Formal methods are a class of mathematical constructs that help reason about correctness of systems. There are generally two types of properties one would like to verify: that something bad cannot happen, which we call *safety properties*, and that something good will eventually happen, which we call *liveness properties*. Recall Example 2.3 where we wanted to check whether, from an initial state, we can always reach the state where the robot is holding the product. This is a typical liveness property. In our example, a safety property could be that the motor of the conveyor can never run when the robot is picking up the part.

In addition to verification, a formal model can also be used to directly *synthesize* restrictions to a system in order to achieve desired properties. When *composing* several resources that make up an automation system, synthesis allows keeping the original formal representations of the different resources of the automation system unchanged by instead relying on global specifications to affect the composed behavior. This decreases the need for modeling special cases in detail and increases the reusability of the formal models. Another way to achieve this is to apply automated planning, where we compute how to transition between states in the in the system in order to reach certain *goal* states. This reduces the need to model *control decisions*, for example steps 1-3 in Section 2.3.

3.1 Supervisory Control Theory

Supervisory Control Theory (SCT) [20] is a model-based framework for verification and synthesis of discrete event systems. The system to be verified is modeled by composing several finite automata, which models different parts of the system. Finite automata have *states*, and *events*, which transitions the automata between the states. Composition is done by the *synchronous composition* operator, where *shared* events happen either in all plants at the same time, or not at all. Terminal states can be *marked.*

Synthesis in SCT is performed by introducing additional *specification* automata, which may include *forbidden states.* Synthesis can be performed with respect to avoiding any forbidden states (ensuring *controllability*) and making sure that all marked states are reachable (ensuring *nonblocking*). The result of the synthesis is a *supervisor* that *disables* events in a way that leaves the most number of reachable states – we say that the supervisor should be *maximally permissive.*

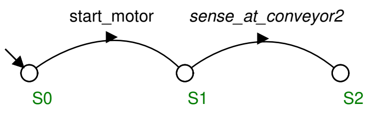


Figure 3.1: Events which transitions the example automation system between the states S_0, S_1, and S_2.

Synthesis w.r.t. controllability only makes sense if *the environment* is modeled – otherwise the solution is to simply disable all events that lead to forbidden states. In SCT, the environment is modeled as *uncontrollable events.* Uncontrollable events cannot be disabled during synthesis. Recall the example in Chapter 2 where a product is transported on a conveyor. Uncontrollable events can capture the fact that if the product is between the two sensors and the motor is on, the product *will eventually* appear in front of the second sensor. In this case the event that takes the plant model to the state where the product has arrived would be uncontrollable. In Figure 3.1 this part of Example 2.3 is modeled as a finite automaton where the "start motor" event is controllable and the "sense at conveyor2" is uncontrollable.

As the specifications are created from automata, it is possible to create very complex *dynamic* specifications, which can remember an arbitrary number of previous events.

Guard extraction

One extension to SCT is using *Extended Finite Automata* (EFA) [21] as the input "language". EFA:s are finite automata extended with variables of finite domain. This makes it possible to model the system with variables, as in Example 2.1. The events of the automata can have guards and actions over these finite domain variables, like the transitions defined in Definition 3.

For systems modeled with EFA:s, *guard extraction* [22] is a useful technique where, after synthesis, the supervisor is represented as additional guards on the controllable events.

Example 3.1: Guard extraction

Continuing the example in 2.3, we show how to write a *specification* instead of changing the guard predicate of a transition (using the terminology of Chapter 2 instead of events).

Consider that the conveyor (including the sensors) and the robot are two separate resources that should be *reusable* without one another. The transition defined in Example 2.2, $conveyor_1 \wedge \neg conveyor_m \wedge \neg pickup_s \wedge \neg dropoff_s \;/\; conveyor_m := true$, refer to both resources and as such it would be difficult to reuse in another automation system.

Instead the conveyor could be modeled independently: $conveyor_1 \wedge \neg conveyor_2 \;/\; conveyor_m := true, part\_between := true$. This would allow turning on the motor if there was something detected at the first sensor and nothing at the second. If it is important that the robot is idle when the conveyor is running, we can instead formulate a *specification* that should always be true: $conveyor_m \implies \neg pickup_s \wedge \neg dropoff_s$. The implication states that if the motor is on, the robot is in its home position. The synthesis algorithm will make sure that any states where this is not true are removed.

In this case, performing guard extraction from the supervisor would result in $\dots \land \neg \mathit{pickup}_s \land \neg \mathit{dropoff}_s$ automatically added to the transition of the conveyor. *Additionally*, it would result in $\dots \land \neg \mathit{conveyor}_m$ added to the transitions that start the robot programs.

3.2 Model checking

In model checking, properties are checked on a *model*. This model is represented by a Kripke structure. The summary here is mostly condensed from the book by Baier and Katoe [23].

Definition 4: *The Kripke structure M is a tuple $M = \langle S, I, R, AP, L\rangle$, where S is a set of states, $I \in S$ is a set of initial states, $R \in S \times S$ is a transition relation, AP is a set of atomic propositions, and $L\colon S \to 2^{AP}$ is a labeling function, defining which propositions are true in each state.*

The example in Chapter 2 can be represented by a Kripke structure as follows:

$$\begin{aligned}
S &= \{s_0, s_1, s_2, \dots\}\\
I &= \{s_0\}\\
R &= \{\langle s_0, s_1\rangle, \langle s_1, s_2\rangle, \dots\}\\
AP &= \{\mathit{conveyor}_1, \mathit{conveyor}_2, \mathit{conveyor}_m, \dots\}\\
L &= \{\langle s_0, \{\mathit{conveyor}_1\}\rangle, \langle s_1, \{\mathit{conveyor}_1, \mathit{product\_between}\}, \dots\rangle\}
\end{aligned}$$

The specifications that can be checked on this structure can be written in many different ways. Two of the most well known are both extensions to propositional logic: Linear Temporal Logic (LTL) [24] and Computation Tree Logic (CTL) [25].

In LTL, propositional logic is extended with temporal operators for expressing properties that relates to sequencing of states (i.e. discrete time). For example, there is an operator for expressing properties on the *next* state ($\bigcirc$), that some formula should *always* ($\Box$) hold, that it should *eventually* hold ($\Diamond$), and that one formula should hold until another one does ($\mathcal{U}$). For example, the formula $\Box(\mathit{conveyor}_1 \to \Diamond \mathit{pickup}_s)$ expresses that it is always the case that if the first conveyor sensor detects a product, the system will eventually start the robot pick-up procedure ($pickup_s = true$). A *LTL model checking problem* is to prove this given a set of valid initial states, or if it cannot, give a counterexample.

An algorithm for checking LTL formulas on a model defined by a Kripke structure is to translate the negation of the specification formula to a nondeterministic büchi automata (NBA). This NBA will contain all the behaviors that the LTL formula considers bad. If the intersection of the Kripke structure and the NBA is empty, the formula is proven valid, e.g. there exist no sequence of actions which violates the formula. Otherwise, counterexamples can be found by examining this intersection.

While LTL formulas can only reason about a single sequence of actions, in CTL, specifications can express branching. This is important since the Kripke stucture can itself express branching – a given state may have multiple successor states. This allows for checking whether, for example, a certain state *can* always be reached, something that is not possible to express in LTL.

There are many model checking tools which support both LTL and CTL, a couple of notable ones are Spin [26] and NuSMV [27]. While the models to be verified in NuSMV are described in a low-level input language, Spin supports a high level language called Promela from which the Kripke structure is derived.

3.3 SAT and bounded model checking

The disposition in this section is based on [28]. SAT is the classic computer science problem of proving whether there exists an interpretation (an assignment of true or false to all *atoms*) of a formula such that the formula is satisfied (i.e. evaluates to true). The *atoms* are boolean variables. For example, consider the following propositional formula

$$x_1 \wedge \neg x_2$$

This formula is satisfiable with the interpretation $x_1 = true$, $x_2 = false$.

A *SAT solver* reads an input file in conjunctive normal form (CNF) and returns either a satisfying interpretation or that the formula is unsatisfiable. CNF is a conjunction of disjunctions of *literals*. Literals are either positive or negative atoms, i.e. either an atom or a negated atom. Every propositional formula can be expressed in CNF, at some cost, either in the size of the formula (paying by introducing new literals), or by computing the canonical CNF representation (paying by solving a computationally difficult problem).

SAT was the first problem to be proven NP-complete (an important result since it means that all problems in the complexity class of NP can then be reduced to SAT). Even though the problem is NP-complete (implies that to solve it an algorithm must be of exponential complexity w.r.t. the problem size), modern SAT-solvers are tremendously powerful tools and can handle problem instances with millions of literals.

Taking advantage of this, in *bounded* model checking (BMC) [29], the Kripke structure together with the specifications are instead formulated as a SAT problem with a bounded size. The bound defines in how many steps from the initial state to look for counterexamples. The transition relation and the negated specifications are iteratively unrolled up to this bound. At each step a SAT problem is solved. If the problem is satisfiable, a specification has been violated, and the resulting assignment can be used to reconstruct a counterexample.

In contrast to MC and SCT, BMC is not *complete* unless the bound is high enough. In practice, this upper bound is often too large to be practically feasible. However, errors can surprisingly often be found in a relatively short amount of steps. As such, BMC is more useful as a tool for *finding* errors than to *prove the absence* of errors, which is the goal of SCT and MC.

3.4 Automated planning

Recall that in MC, the procedure for checking whether a specification holds, the specification is first negated before it is transformed into a Büchi automata. What happens if the specification is not negated? Then instead the intersection between the model and the specificion BA is an accepting run, i.e. one particular (infinite) trace which satisfies the specification. Using a model checker as a backend for automated planning was first suggested in [30], and extended to non-deterministic effects in [31]. Planning with non-deterministic effects is often solved with *cyclic* plans, by solving a CTL problem where the goal should always be reachable under a fairness assumption (that eventually one of the nondeterministic effects will happen). This has a strong relation to controller synthesis [32].

When BMC is used for planning, the maximum length of the plan is bounded by the maximum number of steps. As the planning problem is iteratively encoded into SAT problems of increasing length, the first satisfiable problem instance will constitute the *shortest* (in the number of steps) way to reach the goal state under a given set of specifications. Essentially this arrives at a planning problem formulated as a SAT problem, which has been used in the planning community since [33], but with a powerful temporal specification language for constraining the plans.

In the field of automated planning, planning problems are often written in the Planning Domain Definition Language (PDDL) [34], but in this work, we are interested in computing plans which fit well with the automata-centric approach for modeling the automation system. It is interesting to see that also the input languages for model checking is of interest to the planning community [35] as an alternative to PDDL, citing human-robot collaboration as one of the cases where this makes sense.

One downside to using a BMC solver to perform the planning is that the counterexample trace is a *totally ordered* plan. In contrast to a *partial order plan*, totally ordered plans have a fixed sequential execution order [36]. The partial order plan can thus execute some steps in parallel, where the ordering of actions do not matter. This is important in automation where it is common that different resources are able to perform their tasks in parallel. To efficiently use BMC as a planner, post-processing of the produced plans must be performed.

By using automated planning we can further simplify the modeling activity. It allows us to remove the parts of the model that drives the system forward and instead automatically find our way through the state space towards a given goal. Additionally, the planner can be used to chose suitable parameters automatically.

Example 3.2: Automated planning

Continuing the example in 3.1, we show how to simplify our system further by applying automated planning. In Example 3.1, we used formal *specification* to avoid having to write resource-specific details into the transitions manually. After guard extraction, we were left with the transition (let's call it t_1) $conveyor_1 \wedge \neg conveyor_2$ / $conveyor_m :=$

$true, product\_between := true$ together with a new specification. In Section 2.3 we discussed modeling *effects* and introduced the transition $product\_between \wedge conveyor_m$ / $conveyor_2 := true, product\_between := false$. Let's call this transition t_2.

If we instead of writing t_1 as above, which is essentially a manually written *rule* or *control policy*, only describe the *effects* of turning on the motor, like we did with t_2, t_1 could be replaced with $conveyor_1 \wedge \neg part\_between \wedge conveyor_m$ / $conveyor_1 := false, part\_between := true$. This takes the *control* (i.e. when to turn it on and off) of the motor ($conveyor_m$) out of the system model. We instead leave it to an automated planner to take the decision at run-time, depending on the *goal* of the system. To move the product from the beginning to the end of the conveyor, we can let the planner find the correct sequence of actions resulting in $conveyor_2$ going high. From the initial state S_0 from Example 2.1, this involves turning on the motor, which the planner knows will be followed by t_1 and t_2, which reaches the goal.

Already now we are able to *cancel* the act of moving the product to the end of the conveyor, *without writing anything new*, by simply changing the goal to $\neg conveyor_2$, which will make the planner stop the motor to avoid t_2 from being taken.

CHAPTER 4

Preparation and system integration

Developing a control system, as discussed in Chapter 2, is one of the tasks of preparing an automation system. As we will see in Paper A and Paper D, we suggest that developing the discrete control should start early, and run in parallel to other preparation activities. This chapter will give a brief introduction to the virtual preparation tools used in industry today.

In order to achieve early iterations of the discrete control design work, there is a need to be able to integrate the different resources in the system in a *plug-and-play* manner. With a large number of algorithms in the loop required to achieve intelligent automation, system integration becomes a challenging task in itself. As the appended papers concerns a framework for working with composing resources based on the Robot Operating System (ROS), an introduction to ROS is also given in this chapter.

4.1 Virtual preparation and commissioning

Industry leading software tools supporting virtual preparation, such as *Process Simulate* [37] and *Delmia* [38], combine several different aspects, including 3D workstation layouts, device kinematics and logic for simulation, robot OLP, and material flows, into a single complex software package.

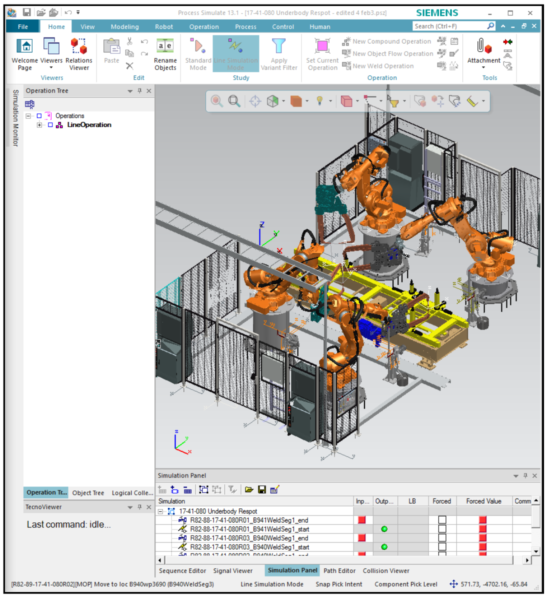
Figure 4.1 shows the user interface of *Process Simulate*, simulating a production station with four robots. Historically these tools were sprung from robot off-line programming (OLP) tools, which have been a big success – robot programs are routinely generated in a 3D simulation environment and downloaded onto the robot controllers on the factory floor, needing very little manual adjustment. To support OLP, the simulation tools include support for validation of production feasibility (e.g. reachability and collision clearance), optimization of robot motions, and they also include support for different robot vendors. More features have been added over the years to enable the simulation of complete assembly sequences, including modeling of other devices (e.g. clamping devices, turn tables), transportation of materials, and simulation of human workers to evaluate ergonomics. To be able to simulate the complete operation of a production station, assembly sequences have historically been entered manually. During the past decade, virtual commisioning (VC) support has been added, allowing the tools to expose the simulation model to a real control system.

VC defines the act of letting a candidate control system implementation, running on its intended hardware, control a software simulation of the production system in what is often called a hardware-in-the-loop setup [39]. The motivation is to reduce physical commissioning time (in [40], savings of 50% is reported) by finding and fixing errors well ahead of the the physical commissioning activity. In addition, VC enables tests that would be prohibitively expensive (virtual products are cheap) or even impossible to run on a physical system (e.g. performing tests which could endanger staff or risk damaging expensive equipment).

4.2 A new automation paradigm

The general trend in automation is clearly moving in the direction of integrating more intelligent robotics that can perform tasks dynamically in a changing environment [41], [42], as well as integrating collaborative robots [43], [44] and other semi or fully autonomous machines such as AGVs [45], [46].

The application described in Section 1 (to be described in more detail in Section 5.2), and intelligent automation in general, features interesting challenges from an integration perspective. Some parts of the system can be classified



Simulation	Inp...	Outp...	LB	Forced	Forced Value	Comm
17-41-080 Underbody Respot						
R82-88-17-41-080R01_B941WeldSeg1_end	🟥			☐	🟥	
R82-88-17-41-080R01_B941WeldSeg1_start		●		☐	🟥	
R82-88-17-41-080R03_B940WeldSeg1_end	🟥			☐	🟥	
R82-88-17-41-080R03_B940WeldSeg1_start		●		☐	🟥	

[R82-89-17-41-080R02][MOP] Move to loc B940wp3690 (B940WeldSeg3)   571.73, -4702.16, -65.84

Figure 4.1: Siemens *Process Simulate* simulating a production station with four robots.

as more traditional automation, such as signal handling for tool connectivity, execution of "special" maneuvers with the robot for attaching the different tools, and controlling the different tools by setting inputs and outputs. Other parts of the application, such as robot motion planning and the tracking of human objects and products, belong more to the fields of robotics, computer perception, and artificial intelligence. The intelligent automation system is a combination of all these fields.

Supporting intelligent automation with the traditional virtual preparation tools will be a challenge, as the system does not always behave in predicable ways. Also, the monolithic structure with the tools containing everything needed for simulation will be a hindrance from an integration perspective.

From a control perspective there are other challenges. Computer perception and manipulation based on machine learning is not perfect, which means that there will be more error states that needs to be handled compared to today. An intelligent automation system described in [47] for example, reports 95%

success rate for picking up parts. Other research on grasping reports number between 84%–94% [48], 91% [49]. This means more execution paths and thus more complex control systems. Combined with the existing challenges of automation software development, such as safety, reliability, and efficiency, new tools and development processes are required.

Another challenging task is how to achieve long-term robustness without sacrificing flexibility. If the semi-autonomous robots of an industrial automation system cooperate with human operators [50], and can ask for help in intuitive ways, either using natural language [51], or projecting intentions and desires [52], the existence of the human operators in the system becomes an *enabler* for robust intelligent automation systems, as they can intervene when something goes wrong [53]. This means that is needs to be possible for operators to influence the automation system to a large degree.

4.3 The robot operating system (ROS)

In order to ease integration and development of different types of online algorithms for sensing, planning, and control of the hardware, various platforms have emerged as middle-ware solutions, of which one stands out, the so-called Robot Operating System (ROS) [54]. ROS has been incredibly successful, with the original paper having almost 8000 citations according to Google Scholar [55]. In the current version of ROS (ROS2 [56]), the communication architecture is based on the Data Distribution Service (DDS) [57] to enable large scale distributed systems to be built on top of it. Even though ROS2 is not hard real-time [58], the decision to build upon DDS paves the way for the use of ROS2-based architectures in real-world industrial automation systems, as we have previously shown in [59].

ROS systems are composed of a set of nodes communicating by sending typed messages over named topics using a publish/subscribe mechanism. This enables a quick and semi-standardized way to introduce new drivers and algorithms to a system. Figure 4.2 shows an early prototype of the application simulated in the ROS visualization tool *RViz*. The simulated robot in the figure can immediately be used with the motion planning framework *MoveIt!* [60], which builds on the Open Motion Planning Library [61] that provides a multitude of state of the art motion planning algorithms. The open-source and collaborative nature of ROS creates a great network effect. For example, with robot motion planning readily available, this also opens up for building on top of that, for example the *ROS2 Grasp Library* [62] which provides plug-in object grasping based on machine learning. This composability of eases prototyping. By providing *driver* software, ROS also makes it easy to move from simulation to reality. As these libraries continue to mature, and with DDS as an improved communications layer, ROS has the potential to provide a good base for developing future automation solutions.

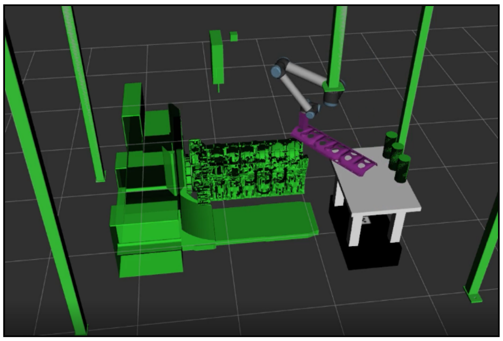


Figure 4.2: Protoyping application described in Section 1. A scene in RViz for testing robot motion planning. A virtual robot is hanging from a pillar in the ceiling, moving a part from an autonomous kitting vehicle onto a diesel engine mounted on an automated guided vehicle. Obstacles can be moved in or out in this environment dynamically.

When composing a system of heterogeneous ROS nodes, care needs to be taken to understand the behavior of each node. While the interfaces are clearly separated into typed messages on named topics, knowledge about the workings of each node is not readily available. This is especially true when nodes have internal states that are not visible to the outside world.

To coordinate different ROS nodes in an automation system, one can think about using a *control system* designed to work with ROS nodes. To do this, a control system needs to know both *how* to control the node, as well as how the nodes *behave.*

4.4 Existing control frameworks for intelligent automation

There exist several frameworks in the ROS community for helping the user with composing and executing (mainly robot) tasks. For example, the framework ROSPlan [63] that uses PDDL-based models for automated task planning and dispatching, SkiROS [64] that simplifies the planning with the use of a skill-based ontology, eTaSL/eTC [65] that defines a constraint-based task specification language for both discrete and continuous control tasks or CoSTAR [66] that uses Behavior Trees for defining the tasks. Below is a summary of how they work.

ROSPlan

ROSPlan features an elegant interface to structure *knowledge*, *planning* and *execution* of plans which uses standard ROS interfaces, allowing any ROS nodes to query the planner easily. Both sequential and temporal plans can be produced and executed. While originally supporting PDDL2.1 with deterministic and temporal planning, it has recently been extended to also support probabilistic planning [67], which enables not only non-determinism but also allow effects which have probabilities associated with them.

SkiROS

The core concept in SkiROS is the composition of reusable robot *skills*. These skills are essentially parameterized computer programs that should be easy to adapt to new situations by changing the parameters. Together with a representation of the world state, the skills can be sequenced autonomously using a PDDL-based planning system to reach goal states, but they can also be used to compose manually defined *tasks*. An ontology is used to organize the knowledge about the world needed by the robots, which includes for example the relation between objects, their locations, if they are graspable etc. SkiROS has been tested with some success in kitting applications in an industrial setting [68].

CoSTAR

CoSTAR uses Behavior Trees (BTs) [69] for defining and executing tasks coupled with a novel approach to integrate computer vision systems. Rather than relying on planning, BTs allow for a more manual, but easy way to instead *program* what should happen in different situations. For example, it is possible to define sequences of subtrees, where the different subtrees can in turn define actions that should be performed in order of importance until one of them succeeds. Because there is no planning system involved, BTs are very efficient to execute. However, this also means that it may be difficult recover from a failure using BTs – when all manually defined options have been exhausted, the BT enters an error state.

CoSTAR allows for easy composition of computer vision algorithms by defining *pipelines* that transform output from machine learning algorithms into predicates that can be referenced in the BT execution. This, combined with the easy programming, allows the system to quickly react to changes in the perceived state of the world.

CHAPTER 5

Preparation and control of intelligent automation systems

With the three chapters of preliminaries out of the way, this chapter is intended to give an overview of the contents of the appended papers and highlight the main results of the thesis. It also aims to highlight how the different papers connect to each other.

The frameworks described in Section 4.4, do support intelligent automation as defined in Section 1.2 in that they can reach the goal from many different states (they are *flexible*), they can *adapt* to a changing environment, and they can *react* to unexpected changes. They support development in different ways; SkiROS focuses on knowledge capture and generalization of robot skills, ROSPlan allows for easy integration with powerful PDDL-based planning engines, and CoSTAR features a novel way to incorporate computer perception algorithms based on machine learning.

A general theme is that they are more focused on *robotics* rather than automation. They tend to ignore the challenges of traditional automation by, for example, not offering sufficient support for detailed control of the execution of actions and the management of IO state. Usually the *actions* to be performed are implemented by computer programs, which can succeed or fail.

There is also work in the automation community to try to have planners being integral parts of the automation system, notable work being [70] which also proposes a hierarchical architecture, and the work described in [71] which uses a planner to continuously compute production plans while being able to react to unexpected events.

The PDDL-based execution environments of Skiros and Rosplan can indeed automatically retry actions that have failed, but it is difficult to have them perform detailed control of *interdependent* resources, which are common in automation. It is our desire to be able to control individual state machines of the resources in the automation system, and also be able to express constraints between them, as we saw in Chapter 3. These constraints must be known to the control system in order to be able to start the system from many different states, for example after unforeseen errors have occurred.

In contrast to the ROS frameworks discussed above, the control framework to be presented in this chapter is less focused on robotics and instead allows a number of *highly interdependent* resources to be modeled independently and composed using a combination of formal specification and planning. To allow for the possibility to restart the automation system, the framework promotes lifting details about the resource state up to the control system. To conclude: in order to take advantage of the tremendous amount of work put into the ROS ecosystem for intelligent automation, there is a need for yet another control framework.

5.1 The Sequence Planner control framework

This section presents the control framework that is the main contribution of this thesis. In this thesis, as well as in the appended papers, we refer to the framework as Sequence Planner (SP). The framework is implemented in the latest iteration of the *software* Sequence Planner. Since Sequence Planner has existed in many shapes over the years, this section clarifies the history of SP. Initially [72], the focus of SP was on supporting engineers in developing control code for programmable logical controllers (PLCs). With time, algorithms to handle product and automation system interaction [73], and to visualize complex operation sequences using multiple projections [74] were developed, as well as integration of formal verification and synthesis using *Supremica* [75]. Online monitoring of production systems [76] and emergency department online planning support [77] have also been developed under the Sequence Planner name. The work in this thesis is implemented in the latest iteration of the software. To avoid confusion, from now on SP refers to the control framework presented in this thesis.

SP is based on a notion of *goals*, where a goal defines a set of states the automation system is desired to be in. For example, consider a product that can be in an "unassembled", or "assembled" state. Then, a goal of the system can be that this product should be in its "assembled" state. To reach this state, the automation system may need to take many different actions – it should follow a recipe or a *plan*. In an intelligent automation system, these actions will most likely be taken in an uncertain environment. This means that plans need to be able to change. SP handles this by constantly replanning to find a suitable set of actions to take given the state of the system.

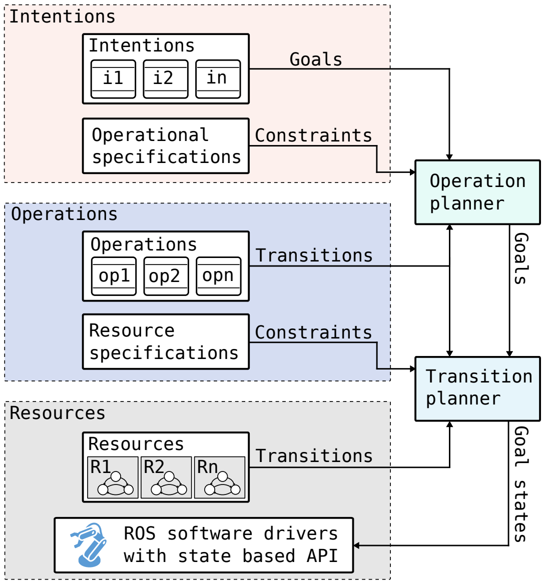


Figure 5.1: Overview of the SP control framework.

Figure 5.1 shows an overview of the proposed framework. Fast replanning is achieved by planning on two levels. On the high level with the *operation planner* that determines a rough course of actions, and on the low level with the *transition planner* that issues detailed goal states to the resources in the system. Reusable *resources* are composed into a global model and constrained via *specifications* to avoid undesired behavior. *Operations* define how the resources can interact to solve individual tasks of the automation system by defining *low-level goals* for the transition planner as predicates over the resource state. The operations abstract away the detailed resource state, which makes it possible for *intentions* to define *high-level goals* for the operation planner of the system. The high-level goals are predicates over *decision variables*, which are the link between the two layers. Additional *operational specifications* model high level constraints, such as in which order operations are allowed to be executed.

This allows complexity during preparation to be reduced, as programming is replaced with defining *goals* for the automation system, combined with *safety constraints* and *operational constraints*. Preparation efforts shift from writing and testing software to writing and testing goals and specifications.

Combined with the fact that the current valuations of the state variables define the current state, the ability to continuously re-plan enables a great deal of flexibility w.r.t. state variables changing in an unexpected way. For example, if parts in the system are continuously tracked, the system can change its plan on the fly if parts have been moved.

State variables

The current valuations of all state variables in the automation system make up the current system state, as in Definition 2. SP uses a state-based control design, where resources continuously receive *goal states* from SP, and continuously update *measured states*, which are inputs to SP. Since the framework is aimed at controlling devices with a varying degree of autonomy, it becomes necessary to be able to control both low-level "dumb" devices (e.g. an actuator or a light) and "intelligent" devices, which may be able to take their own decision on how to reach their goals. For example, consider a simple indicator light, it may have a goal state $\in \{\mathit{off}, \mathit{on}\}$ and a measured state with the same domain. In contrast, a robot may have a more complex goal state that includes a position to in space, perhaps with additional parameters such as maximum joint velocity. For one particular set of valuations of the system variables, there is no notion of how we arrived at the state in question – this means that *memory* needs to be explicitly added to the control system. Memory is modeled using *estimated state variables*. As in Definition 1, the variables in SP are of finite domain.

Example 5.1: Conveyor resource state

Consider again the example in Figure 2.1. The conveyor and the robot would be different *resources*. The resources have states defined by resource variables: *goal state*, *measured state*, and *estimated state*. For example, the table below describes how the conveyor resource could be modeled and controlled in SP.

Variable	Domain	Type
$conveyor_1$	$\{false, true\}$	measured
$conveyor_2$	$\{false, true\}$	measured
$conveyor_m$	$\{false, true\}$	goal
$conveyor_e$	$\{false, true\}$	estimated

Estimated state variables model state that the control system can not directly measure, but needs to keep track of. A part being on the conveyor ($conveyor_e$) is an example of such a state – there is no sensor for it so a memory needs to be introduced. The estimated position of the product is a variable internal to SP.

Transitions

Transitions are defined as in Definition 3, with the distinction that they, just like the state variables, have a type. The type defines in which context the transition can be taken. The types of transitions are:

Controlled transitions are taken when their guard condition evaluates to true, only if they are also activated by the planning system.

Automatic transitions are always taken when their guard condition evaluates to true, regardless of if there are any plans active or not. All automatic transitions are taken before any controlled transitions can be taken. This ensures that automatic transitions can never be delayed by the planner.

Effect transitions define how the *measured state* is updated, and as such they are not used during control like the control and automatic transitions. They are important to keep track of however, as they are needed for online planning and formal verification algorithms (recall the discussion on modeling the environment in Chapter 2). They are also used to determine if the plan is correctly followed – if expected effects do not occur it can be due to an error.

Decision variables and operations

An automation system can generally be thought of as moving around and applying transformations to some item of interest. In the example described in Chapter 2, this item is clearly the product. In other cases it may be less clear, for example in an automation system for controlling an elevator, but there is almost always some notion of a *goal* for the system. In the case of the elevator, we might model it so that the position of the elevator is such an item of interest. A goal state could then express that the elevator should be on a certain floor.

We model the state of such items of interest as estimated variables called *decision variables*. These can be synchronized with resource states at points in time when it is possible to measure them, but they can also be pure memory variables, like the estimated variables. The decision variables can change suddenly by external, not modeled, events, which may or may not trigger a need for replanning.

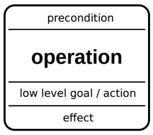


Figure 5.2: We use this graphical notation to visualize operations. An operation consist of a precondition, which is a predicate over the decision variables, a goal state which is a predicate over the resources variables, actions with which to update the state variables upon reaching the goal, and a set of effects to the decision variables of completing the operation.

Describing what can, and should, happen to these items of interest is a natural starting point for modeling an automation system. To define this we introduce the *operation*. The operation is defined in Definition 5 of Paper C, but at a glance, operations bind combinations of *resource variables* to *decision variables*. Figure 5.2 shows a graphical representation of the operation. An operation consist of a precondition, which is a predicate over the decision variables, a goal state which is a predicate over the resources variables, actions with which to update the state variables upon reaching the goal, and a set of effects to the decision variables of completing the operation. To the transition planner, an operation exist as an automatic transition between the goal state and the effect actions. In this way, the operations can be state-less during execution, and manual interventions that change the state will keep the control system state in synchrony.

Example 5.2: The operation abstraction

Consider again the case of the conveyor belt and robot discussed in Chapter 2. To model the product, we introduce a *decision variable*, $p \in \{s_1, s_2, r\}$, where s_1 and s_2 are represent the start and end of the conveyor belt, and r that it is being held by the robot. With the product starting at the beginning of the conveyor, initially $p = s_1$.

An *operation*, O_m, for moving the product over the conveyor belt can then be defined with the guard predicate (p_m) $p = s_1$, effect (e_m) $p = s_2$, goal predicate (g_m) $conveyor_2$, and no actions $g_j = \varnothing$.

When the operation is active, the transition planner tries to reach the goal state $p = s_2$. If the product is placed in front of the second sensor, the control system will assign $p := s_2$, regardless of whether the operation is activated by the operation planner or not, due to the automatic transition created by the operation.

An operation can refer to the state of more than one resource in its goal. This, combined with resource specifications are the main way resources are *composed* in the proposed framework.

Resource specifications

Resource specifications define invariant propositions that must always hold. Because the effects and automatic transitions occur outside of the control of the transition planner, these transitions must be taken into account before taking any controlled transitions, so that states that break the specification are not reached. Paper B shows how we use a technique called *guard extraction* to ensure that such propositions always hold. This synthesizes new guard expressions for the controlled transitions, which ensure that the planner cannot visit any undesired states.

As discussed in Example 3.1, this allows for separating the resource models and the logic, which depends on the specific requirement of the automation system under development. Combined with the operation abstraction for defining how the system progresses, which is also dependent on the specific requirements, resources can be reusable. This contributes towards answering *RQ2* in Section 1.3.

Intentions and operational specifications

Intentions (defined in Definition 6 of Paper C) are used to define high level planning problems w.r.t. the decision variables. For example, an incoming production order could trigger an intention that has as its goal that a certain product should be produced.

There are often constraints that need to be adhered to w.r.t. the decision variables. In production systems, for example, it is common that assembly tasks must occur in a certain sequence. This is modeled using operational specifications. These specifications influence how the operation planner plans the sequences of operations required to reach the current goals. An example of this is described in Section 5.6 of Paper D, where the order of tightening bolts during assembly is important.

Operation and transition planners

The transition runner, (top part in Figure 5.3) keeps track of the current state of all resource states, decision variables, operations, and intentions. It continuously applies transitions (controlled and automatic), which update the system state, reacting to any changes to incoming state from the ROS nodes on the network.

When the guard expression of a transition is evaluated to true in the current state, the transition is taken and the state is updated by the transition's action functions. The state variables relating to resource goals are continuously published to the appropriate ROS2 topics.

Controlled transitions are given additional guard expressions every time a new plan is computed, which defines the execution order and what external state changes that needs waiting for. Intentions and operations are active based on their state. This state can safely be changed arbitrarily in order to cancel running operations or active intentions, as the planner will not allow any forbidden states to be reached.

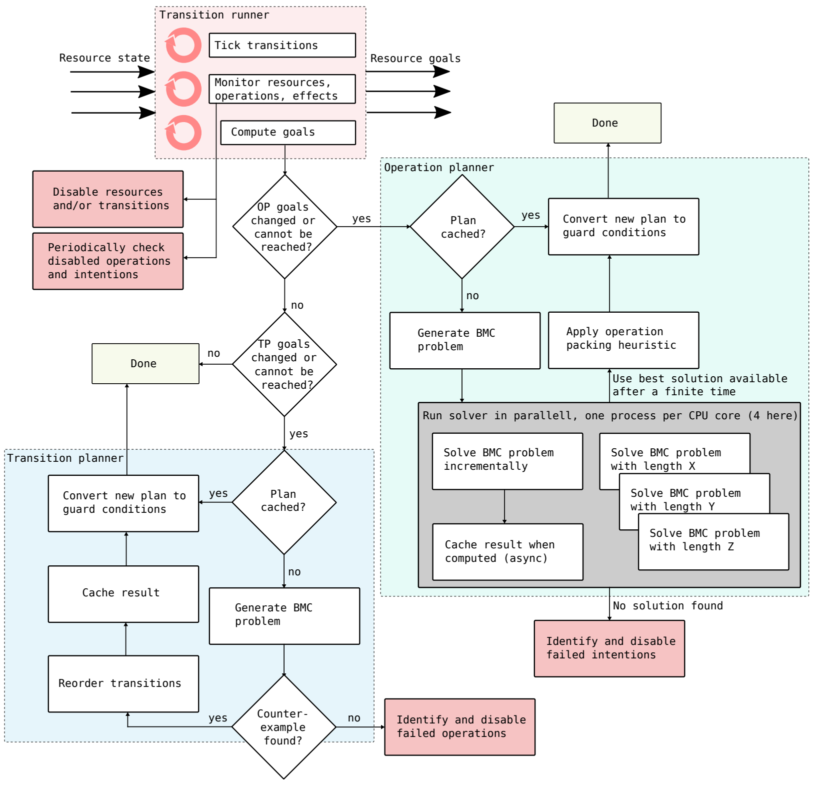

Transition runner
Tick transitions
Monitor resources, operations, effects
Compute goals
Resource state
Resource goals
Disable resources and/or transitions
Periodically check disabled operations and intentions
OP goals changed or cannot be reached?
yes
no
Operation planner
Plan cached?
yes
Convert new plan to guard conditions
Done
no
Generate BMC problem
Apply operation packing heuristic
Use best solution available after a finite time
Run solver in parallell, one process per CPU core (4 here)
Solve BMC problem incrementally
Cache result when computed (async)
Solve BMC problem with length X
Solve BMC problem with length Y
Solve BMC problem with length Z
No solution found
Identify and disable failed intentions
TP goals changed or cannot be reached?
no
Done
yes
Transition planner
Plan cached?
yes
Convert new plan to guard conditions
no
Generate BMC problem
Cache result
Reorder transitions
Counter-example found?
yes
no
Identify and disable failed operations

Figure 5.3: Illustration of how the control system continuously deliberates new plans for both the operations and resources.

Because the system is modeled with effect transitions, it is possible for the transition runner to continuously monitor if an effect does not occur within some time bound, or if the "wrong" effect occurs. When an effect does not occur within a (user specified) timeout period, the effect is disabled in the generated planning problem. For example, there may exist an effect which transitions between a "request" state and a "response" state for a particular resource. Keeping track of the duration an effect has been enabled provides a general way to specify behavior for timeouts. If the system leaves the state where an effect is active, the timestamp for that effect is reset.

A special case of effects not occurring in time is if a resource fails to communicate with SP, for example due to a network, hardware, or software error. This leads to the resource being marked as unavailable, which disables all transitions related to it.

The transition runner also keeps track of which operations that are in an error state, and keeps this information updated based on continuously checking if the operation can be completed from the current state. This is done by periodically executing a one-off planning request in the background. If an operation can be completed, the error state is automatically removed. This allows the system to continue automatically when errors have been cleared.

The *operation planner* (to the right in Figure 5.3) continuously computes sequences of operations in order to reach the goals of the currently active intentions while adhering to the operational specifications described in Section 5.1. This is done by generating a BMC problem based on the preconditions and effects of the operations, the operation specifications, and the current state of all decision variables. Then, a model checker is called using a heuristic that tries to solve for different plan lengths in parallel [78]. The model checker tries to prove that the desired goal state is *not* reachable under the operational specifications. If the goal state *is* reachable, a counterexample is produced, which is converted into a plan. The computed plan is of total order, but a post-processing step is applied where operations can be started in parallel if there are no dependencies between them. More about the operation planner can be read in Section 2.6 of Paper C.

In the same way, the *transition planner* (bottom left Figure 5.3) tries to reach the goal of all currently executing operations. It has constraints defined by the invariant formulas (and synthesis results) described in Section 5.1.

The operations and the decision variables naturally define a hierarchy, where the operations define an abstraction of how products and important abstract information about resource state can change in the system. In order to produce correct plans this abstraction needs to be ordered monotonic [79], which means that the transition planner must make sure not to change any other decision variables than the ones in the current goal. Otherwise, the transition planner may end up reversing progress already made by the operations.

In SP the strategy is to let the planner decide on the “correct” outcome whenever there is non-determinism in the model, and instead re-plan upon encountering an unexpected result. This means that non-determinism at the transition planner level must be properly modeled by the operations. This is done by creating *variants* of operations, which are copies of the same operation, but with different actions and goal states. When an operation that has a variant is executing, the goal of the transition planner is set to the disjunction of the operation variants' goals. For example, consider a sensor ($sens_?$) that scans the color of a product, determining that the product is either red or green. An operation "scan" could exist in two variants, one with a goal predicate $sens_? = red$ and an outcome that a decision variable is assigned "red" and one with a goal predicate $sens_? = red$ and the outcome that a decision variable is assigned "green". The goal for the transition planner would then be $sens_? = red \lor sens_? = green$. When $sens_?$ receives one of the values, the decision variable is updated accordingly. This may trigger re-planning of the operation planner if the "correct" outcome was not achieved.

Example 5.3: Planning and error handling

In Example 5.2, an operation O_m was introduced, to move the product, modeled by the decision variable p, from one end of the conveyor to the other.

Consider what would happen if, on its way from one end to the other, we lift the product off from the conveyor. The control system now expects $conveyor_2$ to go high, due to the effect transition $product\_between$ / $conveyor_2 := true, part\_between := false$ that we added in Section 2.3, since this transition will be part of the plan to reach the goal.

Because we interfered and removed the product, there is now a mismatch between the estimated state of the control system and reality. Eventually we time out on the effect transition, which will cause the next replan to fail due to the goal of O_m being unreachable. The failed plan puts O_m in an error state and it will no longer be included by the operation planner. Any unrelated operations will continue their execution in the mean time.

Figure 5.4 shows an example of how an error can propagate in the hierarchy. The figure shows an intention, "tighten all bolts", which has a goal that all "bolts" in the system should be tightened. The bolts can be tightened by a robot holding a tool built for this purpose (Paper E includes a more thorough description of the system). An operation plan has been computed, which reaches the goal that all bolts should be tightened. The plan includes locating the proper tool using a 3d camera, attaching the robot to the tool, and then using the tool to tighten the bolts.

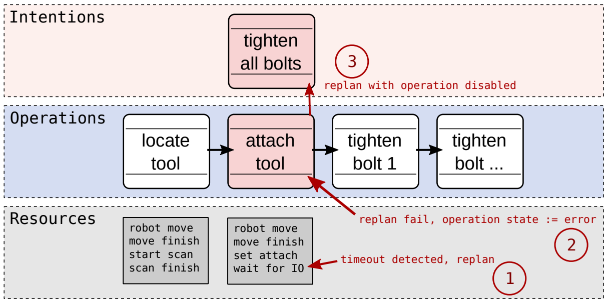


Figure 5.4: Propagating errors in the planning hierarchy.

During execution, a low level effect fails to occur (1). This disables the problematic effect transition, which triggers a replan with the transition planner. If this replan fails (2), the operation that posted the goal is identified and is put into its error state (pink color). Thus, the system *learns* which operations are currently not possible to complete. It may or may not be able to continue execution with this new knowledge. A new operation plan is computed, (3) in Figure 5.4, with the operation disabled. In this case, no plan was found, and the intention is also disabled.

Maintenance transitions (MTs)

In the case that execution cannot continue, the exact cause of the problem needs to be possible for an operator to pin-point. The planner can be used to aid the operator by suggesting a solution to the problem. This is done with the help of *maintenance transitions* (MTs).

MTs are controlled transitions that are only active when checking if the disabled operations can be restarted. Their purpose is to give the planning system more freedom to operate in, by encoding specific actions that the operator can take (in the physical world) to solve problems.

For example, clearing an error state off a resource can be modeled as an MT if clearing the error cannot, or should not, be done automatically by the control system. It is common that resources include a `reset` maintenance transition, which sets the state of the resource to some predefined (safe) state.

By letting the planning system correct errors via MTs, safety is ensured – either the goals can be reached without violating any safety constraints, or they cannot be reached at all.

5.2 Applications and results

We end this thesis by taking a closer look at the application in the UNIFICATION project. The application concerns transforming an existing manual assembly station from a truck engine final assembly line, shown in Figure 1.1a, into an intelligent and collaborative robot assembly station, shown in Figure 1.1b.

In the application, diesel engines are transported from station to station in a predetermined time slot on Automated Guided Vehicles (AGVs). Material to be mounted on a specific engine is loaded by an operator from kitting facades located adjacent to the line. An autonomous mobile platform (MiR100) carries the kitted material to be mounted on the engine, to the collaborative robot assembly station.

In the station, a robot and an operator work side by side to mount parts on the engine, sharing tools suspended from the ceiling. A dedicated camera system keeps track of operators, ensuring safe coexistence with machines. The camera system can also be used for gesture recognition.

Before the collaborative mode of the system starts, an authorized operator has to be verified by a RFID tag. After verification, the operator is greeted by the station and operator instructions are shown on a screen. If no operator is verified, some operations can still be executed independently by the robot. However, violation of safety zones triggers a safeguard stop.

After the AGV and the kitting material have arrived, a Universal Robots (UR10) robot and an operator together lift a heavy ladder frame on to the engine. After placing the ladder frame on the engine, the operator informs the control system with a button press on a watch, or with a gesture, after which the UR10 leaves the current end-effector and attaches itself to the nutrunner used for tightening bolts. During this tool change, the operator starts to place 24 bolts, which the UR10 will tighten with the nutrunner.

During the tightening of the bolts, the operator can mount three oil filters. If the robot finishes the tightening operation first, it leaves the nutrunner in a floating position above the engine and waits for the operator. When the operator is done, the robot attaches a third end-effector and starts performing the oil filter tightening operations. During the same time, the operator attaches two oil transport pipes on the engine, and uses the same nutrunner to tighten plates that hold the pipes to the engine. After executing these operations, the AGV with the assembled engine and the empty MiR100 leave the collaborative robot assembly station.

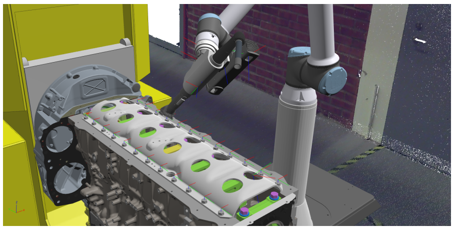
The transformation has undergone several iterations during the course of the project. The first iteration was a semi-static one, with both the UR10 robot and the nutrunner hanging in a fixed positions from the ceiling. An early simulation of this setup can be seen in Figure 4.2. In the setup described in Paper D, the fixed position of the robot is replaced with a robot on a movable pallet and tools that hang from wires in the ceiling. The robot has a 3d camera mounted on it for localizing tools and products.

Applying IVPC for an intelligent automation system

When trying to solve the complexities of intelligent automation using a planning framework, traditional preparation work such as optimizing robot programs off-line can no longer be done with the expectation that these programs will run uninterrupted. The control system may take other decisions due to external events, or there may exist temporary obstacles in the environment. It is still important to perform off-line preparation to ensure an efficient automation system. Even though a robot is semi-autonomous, in our use case it is still required to work with placement of the robot, product geometries, etc. Additionally, there is a need to simulate the system to study the sometimes hard to predict actions of the planner.

An interesting feature of the control framework is that, with the exception of the invariant expansions described in Section 5.1, there is no off-line computation necessary when changing the components of the control system. This means that we can add, remove, or change both resources, operations, and state variables, without needing to compile or download anything.

Paper D describes how we worked with virtual preparation to compute sequence constraints for the control system with a dedicated software for robot motion planning called IPS [80], as well as how we used virtual validation of our control design, which can be built step by step during the process. The preparation process follows a concept introduced in Paper A, called Integrated Virtual Preparation and Commissioning (IVPC). In IVPC virtual preparation


(a) No collision when locating the engine.

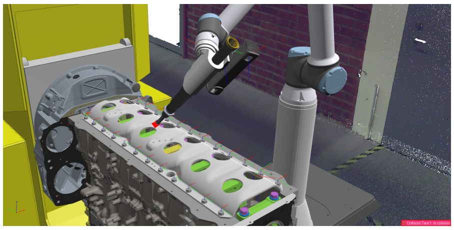

(b) Red triangles highlight the collision between the new tool and the workpiece when locating the engine.

Figure 5.5: Changing tools todo.

is extended to include formalized control system development and VC. IVPC features an iterative workflow that is centered around VC for validation and a model based control system design. If this workflow includes a two-way communication between the VC (which we assume also contains tools for virtual preparation) simulation software and the control system, not only can the simulation software be used to *validate* the control system, but also provide information from which it is possible to *automatically generate constraints* for the control system model. This enables a truly interactive workflow that enables rapid iterations at the same desk, reducing both handovers between engineering disciplines and mental context switches.

Paper D describes the process of importing resource models, and then step by step adding operations that can initially be completed very easily by the planner. Resource specifications are added while a simulation is running, which forces the planner to start avoiding certain states. Similarly, operational specifications, which define sequencing of the assembly tasks are added to the live automation system. While the original concept for IVPC described in Paper A had in mind control *code generation* based on formal models, how IVPC was applied in Paper D provides a more tightly integrated and pleasant development experience.

In Paper D we describe how the operational specifications can be used to define preferred orderings of operations in an interactive way. Consider for example the case where multiple resources have operations that can complete some task, but the different tasks need to happen in a certain order. The simulation allows us to experiment with what will happen if the operator performs tasks “out of sequence”, or if certain resources become unavailable.

Another example of the iterative workflow shown in Paper D revolves around wanting to change to a different type of nutrunner tool. If preparation and control design is performed as parallel activities, they both influence each other. While it was, with the old tool, possible to scan the engine even with the tool attached, the control system entered an error state on the *scan engine* operation once the tool had been changed. Figure 5.5 illustrates the behavior. The connection with the planner also allows traditional model checking to be performed on the models. Any counterexamples found can then, because they are just plans, be visualized in the simulation software as done in [81].

This interactive formal modeling, presented in Paper A and Paper D, contributes towards *RQ3* in Section 1.3.

Investigating error recovery

As discussed in Section 5.1, both transitions, operations, and intentions can be disabled during runtime. This can be used to handle, or work around, errors, dynamically changing what the system is allowed to do.

Paper E investigates how the control framework handles a variety of error situations. The use case from Paper D was extended with an additional robot and a number of different error scenarios were manually triggered. The errors all fell into one of the following categories.

- *Unresponsiveness* A resource does not respond in time. For example, communication failure or equipment malfunction (e.g. a sensor that does not give the expected result).
- *Task failure* A resource fails to perform its current task. This may be both expected and unexpected failures.
- *Unexpected events* The state unexpectedly changes. For example, consider the case of a product slipping away from the grip of a robotic gripper.
- *Specification errors* The automation system breaks safety specifications or cannot make progress. This can occur due to mistakes in programming, or as a consequence of previous errors.

Figure 5.6 shows one example of an error situation. In this experiment the robot to the left, *r2*, has stopped responding for an unknown reason.

However, it is still attached to the smart tool (orange), which is needed to complete assembly. Additionally, *r2* is physically blocking the space around the bolts. In Paper E we are interested to know if this particular situation can

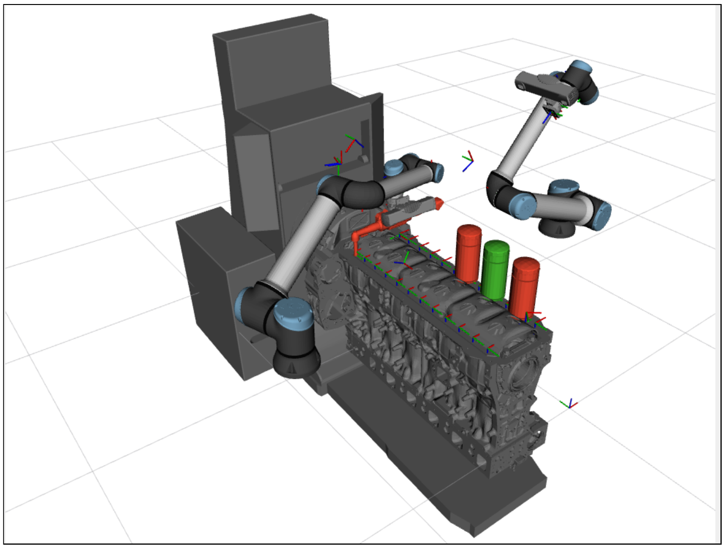
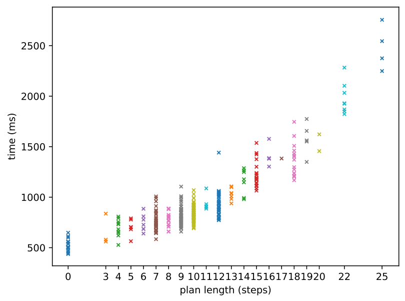


Figure 5.6: Simulation showing one of the experiments performed in Paper E. In this experiment the robot to the left, *r2*, has stopped responding for an unknown reason. However, it is still attached to the smart tool (orange), which is needed to complete assembly. Additionally, *r2* is physically blocking the space around the bolts.

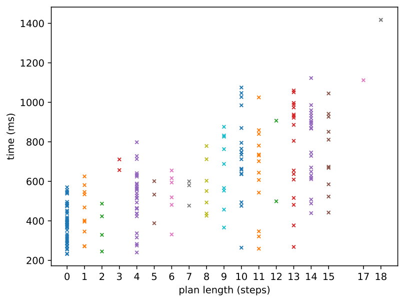
be recovered from, even if *r2* stays disconnected. The control system looks for a plan that includes MT:s and suggests a solution to the operator that it should manually put the tool back to its initial hanging position (which is an MT). Out of nine situations tested, the operator had to intervene in four of them. After manual intervention, restarting production was possible in all cases.

Planning performance

Paper C includes some benchmarks on the performance of the two different planners that are included here. Because the operation planner and the transition planner effectively define a hierarchy, there exist a trade-off in how much



(a) Planning times for the transition planner.



(b) Planning times for the operation planner.

Figure 5.7: Planning performance.

information should be encoded in the decision variables. For example, it is not necessary to model the knowledge of which tool the robot is holding as a decision variable, because when the time comes to bolt, the transition planner would figure out that it is necessary and perform the necessary resource transitions. However, changing the tool is a costly (in time) operation and so the location of the tool should be encoded as a decision variable.

The transition planning model benchmarked includes 6 resources and has 84 transitions (of which 30 are auto transitions coming from operations), 58 specifications, and 1.02434e22 reachable states. The operation planning model is orders of magnitudes smaller with 1.61248e9 reachable states, with 30 transitions modeling the operations.

The plots in Figure 5.7 show the planning times for the transition planner and operation planning ordered by plan length. Each cross is one plan computed. The plans were computed on a consumer grade laptop computer. In this system, the responsiveness is enough at around 1 second, which suggests that for a system of this size, the transition planner can comfortably search between 15-20 steps into the future while keeping adequate responsiveness. Even though the system can handle great flexibility, for the most part things progress nominally, in which case the system does not need to replan. For a system of this size, planning performance does not appear to be a hindrance. Combined with the usability also after unexpected situations as described in Section 5.2, this contributes to answering *RQ1* in Section 1.3.

CHAPTER 6

Summary of included papers

This chapter provides a summary of the included papers.

6.1 Paper A

Martin Dahl, Kristofer Bengtsson, Patrik Bergagård, Martin Fabian, and Petter Falkman
Integrated virtual preparation and commissioning: supporting formal methods during automation systems development
8:th IFAC Conference on Manufacturing Modeling, Management & Control,
IFAC-PapersOnLine,
49(12), pp. 1939-1944, 2016.

This paper proposes a framework, *Integrated Virtual Preparation and Commissioning*, where virtual commissioning models are used as a base for preparation and control system implementation assisted by formal methods. The extensive use of simulation in virtual commissioning allows computation results from formal methods to be continuously validated by visual inspection and using existing analysis tools (e.g. collision detection methods). The framework is applied in a case study, where the combination of a simulation model and a formal model is used as an aid in generating operation sequences for validation during production preparation.

6.2 Paper B

Martin Dahl, Kristofer Bengtsson, Martin Fabian, Petter Falkman
Guard extraction for modeling and control of a collaborative assembly station
Workshop on Discrete Event Systems, Nov. 2020,
To appear in conference proceedings.

This paper presents an earlier iteration of the proposed control framework, which is why it uses different notation than the following Papers. It focuses on how invariant propositions over the resource descriptions can be expressed by extending the original guards using guard extraction. Planning and verification can then be performed directly on the system with additional guards. While the specification language is limited to invariant propositions, we find that in practice many common safety specifications can be expressed like this when combined with the notion of uncontrollability.

6.3 Paper C

Martin Dahl, Endre Erős, Kristofer Bengtsson, Martin Fabian, Petter Falkman
Sequence Planner: A framework for control of intelligent automation systems
Submitted to Robotics and Computer-Integrated Manufacturing.

Paper C presents a control framework for automation systems featuring collaborative robotics and other machines with some degree of autonomy. To develop robust automation solutions, there is also a need for reliably *coordinating* a number of complex subsystems. The proposed framework helps with this complex task by relying on supporting algorithms for control logic synthesis and online planning. The framework is applied to an industrial use case.

6.4 Paper D

Martin Dahl, Christian Larsen, Endre Erős, Kristofer Bengtsson, Martin Fabian, Petter Falkman
Interactive formal specification for efficient preparation of intelligent automation systems
Submitted to CIRP Journal of Manufacturing Science and Technology.

The intelligent automation systems of the future will consist of an increasing amount of complex resources, such as collaborative robots or and autonomous roaming robots for material handling. To be used in an automation system, these complex resources need to be coordinated, both with each other and with respect to high level specification on the goals of the automation system. In Paper D we apply the framework for virtual preparation with a model-based control system in the loop presented in Paper A to a case study of an intelligent and collaborative automation system. This allows for new resources, operations, and constraints to be integrated into a running virtual system, enabling a truly integrated preparation approach.

6.5 Paper E

Martin Dahl, Endre Erős, Kristofer Bengtsson, Martin Fabian, Petter Falkman
Application of the Sequence Planner control framework to an intelligent automation system with a focus on error handling
Submitted to Machines, special issue on Mechatronic System for Automatic Control.

In the new, intelligent automation systems discussed in this thesis, various error states are expected to occur more often, highlighting the need for good support in error handling. This paper studies, using a case study, how well the control framework presented in Paper B can recover after errors. It presents the key insights that were gained during implementation.

CHAPTER 7

Concluding remarks and future work

This thesis introduced a framework for model based control of distributed devices, with *intelligent automation* in mind. We have used it to control robots, tools, human-machine interfaces, and have successfully been able to restart after errors. We have chosen techniques which do not require long times spent recomputing synthesis problems to guarantee safety specifications. This allows for very fast iteration times, which, combined with interactive simulation, supports the creative process. This Chapter provides some concluding remarks and directions for future work. The conclusions are grouped around the research questions introduced in Section 1.3.

RQ1 *How could a control framework for an intelligent automation system be designed?*

As we noted in the discussion about intelligent automation in Section 1.2, the control system needs to be able to adapt to its surroundings and be able to find multiple ways to reach the goals. To this end a *goal-oriented* control framework based on automated planning is proposed. The control system continuously communicate *goal states* for each individual resource. What these goal states should be is computed with the help of automated planning. This makes it possible to change the goals of the system on the fly, even half-way through another process. This satisfies the requirements of being adaptive, deliberate and to some extent reactive. As a system designed around online planning cannot be real-time, we cannot reach high levels of reactivity on the coordination level – this must be handled in the resources.

RQ2 *How can we use model-based techniques to reduce complexity when preparing control systems for intelligent automation systems?*

In the classic paper about software engineering *"No silver bullet"*[82] by by Brooks and Kugler, the phrase *silver bullet* became popularized. Brooks and Kugler's argument is that there will be no programming language, or new methodology, that can give an order of magnitude increase in productivity (this would be the silver bullet), because the essential problem of software *is fundamentally difficult* and even though many different representations can be used in search of an engineering solution, the essential problems are still there.

Using model-based techniques allows for declarative specification of the intentions of the system, rather than focusing on the *how*. Formal specification and synthesis, combined with automated planning can essentially eliminate "if-then-else" programming.

It is however the authors view that the model-based techniques are not a silver bullet. The essential problems are fundamentally in the *details* of the automation system. When we shift towards a model-based approach, we shift the problem from writing correct code to modeling the problem properly. When things go wrong, abstractions break and we are anyway forced to understand what is happening in the system. *Someone* needs to know the details.

What the model-based techniques do offer is a structured way to tackle complexity, which could (some day) be condensed into engineering work tasks.

RQ3 *Could an interplay between simulation and formal methods provide a foundation for an iterative preparation methodology?*

Continuing the reasoning from *RQ2*, with the following quote, also from Brooks and Kugler: *The hardest single part of building a software system is deciding precisely what to build. No other part of the conceptual* *work is to difficult as establishing the detailed technical requirements, including all the interfaces to people, to machines, and to other software systems. No other part of the work so cripples the resulting system if done wrong. No other part is more difficult to rectify later.*

The earlier errors can be found, the less costly it is to fix them. As the previous quote hints at, it is also possible to build something *correctly* even though what was built turns out to be the wrong thing. The advent of agile methodologies that focus on quick iterations and changing requirements is a response to this aspect.

The interactive framework proposed in this thesis can help with this point. Different resources and specifications can be tested out in a live environment to gain an understanding of details in the system that may not be apparent in earlier planning phases. This is especially true for an intelligent automation system, where both resources and operators may behave in unexpected ways. Using the framework presented in this thesis, the control system can evolve into a full-grown prototype already during early preparation.

Future work

One weakness of the presented framework is that it does not provide a general way to avoid deadlock situations. While it is possible to perform model checking w.r.t. liveness, this is limited to small systems due to the state space explosion problem. This means that *testing* becomes a crucial activity, but this has not been worked on within the scope of this thesis.

The framework as it is presented in this thesis is limited to variables with a boolean or enumeration type domain. Today there exist mature solvers for problems of the class Satisfiability Modulo Theories (SMT), where *theories* are integrated into the core of the SAT solving algorithm. Examples of such theories are arithmetics with real numbers, pointer and array logic, and bitvectors. This allows for a much more expressive modeling language as well as the possibility to perform *optimization* with a cost function compared to just finding the shortest plan in number of steps taken.

As an example, consider the case shown in Paper D where the user is interactively entering sequence specifications. It is not difficult to envision, that if operations had a cost related to them (or an execution time), that the user could be entering constraints for a mathematical programming or constraint satisfaction problem instead. This would enable on-line *optimization* of the operations instead of just finding one particular plan. Combined with *learning* how long an operation takes (which can vary depending on the state of the involved resources), some interesting results could potentially emerge.

Another interesting thing to *learn* are the effect transitions. Given a set of resources and formal models of their behavior it could be possible to learn what happens when they interact in a simulated environment. We could find out that some states are not reachable, or what happens to the products in the system by observing such a simulation.

References

[1] G. Chryssolouris, D. Mavrikios, N. Papakostas, D. Mourtzis, G. Michalos, and K. Georgoulias, “Digital manufacturing: History, perspectives, and outlook,” *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture*, vol. 223, no. 5, pp. 451–462, 2009.

[2] A. Rullán, “Programmable logic controllers versus personal computers for process control,” *Computers & Industrial Engineering*, vol. 33, no. 1, pp. 421–424, 1997, Proceedings of the 21st International Conference on Computers and Industrial Engineering, ISSN: 0360-8352.

[3] H. Lasi, P. Fettke, H.-G. Kemper, T. Feld, and M. Hoffmann, “Industry 4.0,” *Business & information systems engineering*, vol. 6, no. 4, pp. 239–242, 2014.

[4] V. Vyatkin, “Iec 61499 as enabler of distributed and intelligent automation: State-of-the-art review,” *IEEE transactions on Industrial Informatics*, vol. 7, no. 4, pp. 768–781, 2011.

[5] *Merriam-Webster Online Dictionary*, `https://www.merriam-webster.com/dictionary/intelligence`, [Online; accessed 18-Nov-2020], 2020.

[6] T. W. Malone, “How human-computer ’superminds’ are redefining the future of work,” *MIT Sloan Management Review*, vol. 59, no. 4, pp. 34–41, 2018.

[7] The VirtCom project. (2020). About the virtcom project. [Online; accessed 20-Nov-2020].

[8] C. G. Lee and S. C. Park, “Survey on the virtual commissioning of manufacturing systems,” *J. Comput. Des. Eng.*, vol. 1, no. 3, pp. 213–222, 2014, ISSN: 2288-4300.

[9] T. Ovatman, A. Aral, D. Polat, and A. O. Ünver, “An overview of model checking practices on verification of PLC software,” *Softw. Syst. Model.*, pp. 1–24, 2014.

[10] O. Ljungkrantz, K. Åkesson, and M. Fabian, *Practice of industrial control logic programming using library components.* INTECH Open Access Publisher, 2010.

[11] M. Dahl, A. Albo, J. Eriksson, J. Pettersson, and P. Falkman, “Virtual reality commissioning in production systems preparation,” in *2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, Sep. 2017, pp. 1–7.

[12] M. Dahl, K. Bengtsson, P. Bergagård, M. Fabian, and P. Falkman, “Sequence planner: Supporting integrated virtual preparation and commissioning,” *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 5818–5823, 2017, 20th IFAC World Congress, ISSN: 2405-8963.

[13] M. Dahl, K. Bengtsson, M. Fabian, and P. Falkman, “Automatic modeling and simulation of robot program behavior in integrated virtual preparation and commissioning,” *Procedia Manufacturing*, vol. 11, 284–291, 2017.

[14] *Volvo GTO Vision*, `https://www.engineering.com/PLMERP/ArticleID/18868/Vision-and-Practice-at-Volvo-Group-GTO-Industry-40-and-PLM-in-Global-Truck-Manufacturing.aspx`, [Online; accessed 14-Apr-2019].

[15] A. Hanna, P.-L. Götvall, M. Ekström, and K. Bengtsson, “Requirements for designing and controlling autonomous collaborative robots system-an industrial case,” *Advances in Transdisciplinary Engineering*, pp. 139–144, 2018.

[16] A. Hanna, K. Bengtsson, M. Dahl, E. Erős, P. Götvall, and M. Ekström, “Industrial challenges when planning and preparing collaborative and intelligent automation systems for final assembly stations,” in *2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, Sep. 2019, pp. 400–406.

[17] E. Erős, M. Dahl, A. Hanna, and K. Bengtsson, “A ROS2 based communication architecture for control in collaborative and intelligent automation systems,” in *29th International Conference on Flexible Automation and Intelligent Manufacturing (FAIM2019)*, Jun. 2019.

[18] E. Erős, M. Dahl, A. Hanna, A. Albo, P. Falkman, and K. Bengtsson, “Integrated virtual commissioning of a ros2-based collaborative and intelligent automation system,” in *2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, Sep. 2019, pp. 407–413.

[19] M. Dahl, E. Erős, A. Hanna, K. Bengtsson, M. Fabian, and P. Falkman, “Control components for collaborative and intelligent automation systems,” in *2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, Sep. 2019, pp. 378–384.

[20] P. J. Ramadge and W. M. Wonham, “Supervisory control of a class of discrete event processes,” *SIAM J. Control Optim.*, vol. 25, no. 1, pp. 206–230, 1987.

[21] M. Sköldstam, K. Åkesson, and M. Fabian, “Modeling of discrete event systems using finite automata with variables,” in *Decision and Control, 2007 46th IEEE Conference on*, IEEE, 2007, pp. 3387–3392.

[22] S. Miremadi, K. Akesson, and B. Lennartson, “Symbolic computation of reduced guards in supervisory control,” *IEEE Transactions on Automation Science and Engineering*, vol. 8, no. 4, pp. 754–765, Oct. 2011.

[23] C. Baier and J.-P. Katoen, *Principles of model checking.* MIT press, 2008.

[24] A. Pnueli, “The temporal logic of programs,” in *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*, IEEE, 1977, pp. 46–57.

[25] E. M. Clarke and E. A. Emerson, “Design and synthesis of synchronization skeletons using branching time temporal logic,” in *Workshop on Logic of Programs*, Springer, 1981, pp. 52–71.

[26] G. J. Holzmann, “The model checker spin,” *IEEE Transactions on software engineering*, vol. 23, no. 5, pp. 279–295, 1997.

[27] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella, “NuSMV Version 2: An Open-Source Tool for Symbolic Model Checking,” in *Proc. International Conference on Computer-Aided Verification (CAV 2002)*, ser. LNCS, vol. 2404, Copenhagen, Denmark: Springer, Jul. 2002.

[28] D. Kroening and O. Strichman, *Decision procedures.* Springer, 2016.

[29] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu, “Symbolic model checking without bdds,” in *International conference on tools and algorithms for the construction and analysis of systems*, Springer, 1999, pp. 193–207.

[30] F. Giunchiglia and P. Traverso, “Planning as model checking,” in *European Conference on Planning*, Springer, 1999, pp. 1–20.

[31] M. Pistore and P. Traverso, “Planning as model checking for extended goals in non-deterministic domains,” in *IJCAI*, vol. 1, 2001, pp. 479–486.

[32] N. D’Ippolito, N. Rodríguez, and S. Sardina, “Fully observable non-deterministic planning as assumption-based reactive synthesis,” *Journal of Artificial Intelligence Research*, vol. 61, pp. 593–621, 2018.

[33] H. A. Kautz, B. Selman, *et al.*, “Planning as satisfiability.,” in *ECAI*, Citeseer, vol. 92, 1992, pp. 359–363.

[34] M. Fox and D. Long, “Pddl2. 1: An extension to pddl for expressing temporal planning domains,” *Journal of artificial intelligence research*, vol. 20, pp. 61–124, 2003.

[35] J. Hoffmann, H. Hermanns, M. Klauck, M. Steinmetz, E. Karpas, and D. Magazzeni, “Let’s learn their language? a case for planning with automata-network languages from model checking,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, 2020, pp. 13 569–13 575.

[36] D. S. Weld, “An introduction to least commitment planning,” *AI magazine*, vol. 15, no. 4, pp. 27–27, 1994.

[37] *Process Simulate*, `https://www.plm.automation.siemens.com/global/en/products/tecnomatix/`, [Online; accessed 14-Apr-2019].

[38] *Delmia*, `https://www.3ds.com/products-services/delmia/`, [Online; accessed 14-Apr-2019].

[39] F. Auinger, M. Vorderwinkler, and G. Buchtela, "Interface driven domain-independent modeling architecture for "soft-commissioning" and "reality in the loop"," in *Proc. 31st Conf. Winter Simul. Simulation—a Bridg. to Futur. 1*, ACM, 1999, pp. 798–805, ISBN: 978-1-4244-9864-2.

[40] N. Shahim and C. Møller, "Economic justification of virtual commissioning in automation industry," in *Winter Simulation Conference (WSC), 2016*, IEEE, 2016, pp. 2430–2441.

[41] R. Alterovitz, S. Koenig, and M. Likhachev, "Robot planning in the real world: Research challenges and opportunities," English, *AI Magazine*, vol. 37, no. 2, pp. 76–84, Jun. 2016.

[42] L. Perez, E. Rodriguez, N. Rodriguez, R. Usamentiaga, and D. F. Garcia, "Robot guidance using machine vision techniques in industrial environments: A comparative review," *Sensors*, vol. 16, no. 3, 2016, ISSN: 1424-8220.

[43] M. Parente, G. Figueira, P. Amorim, and A. Marques, "Production scheduling in the context of industry 4.0: Review and trends," *International Journal of Production Research*, vol. 58, no. 17, pp. 5401–5431, 2020.

[44] A. Bauer, D. Wollherr, and M. Buss, "Human-robot collaboration: A survey," *International Journal of Humanoid Robotics*, vol. 05, no. 01, pp. 47–66, 2008.

[45] H. Andreasson, A. Bouguerra, M. Cirillo, D. N. Dimitrov, D. Driankov, L. Karlsson, A. J. Lilienthal, F. Pecora, J. P. Saarinen, A. Sherikov, *et al.*, "Autonomous transport vehicles: Where we are and what is missing," *IEEE Robotics & Automation Magazine*, vol. 22, no. 1, pp. 64–75, 2015.

[46] L. Sabattini, M. Aikio, P. Beinschob, M. Boehning, E. Cardarelli, V. Digani, A. Krengel, M. Magnani, S. Mandici, F. Oleari, *et al.*, "The pan-robots project: Advanced automated guided vehicle systems for industrial logistics," *IEEE Robotics & Automation Magazine*, vol. 25, no. 1, pp. 55–64, 2017.

[47] E. Solowjow, I. Ugalde, Y. Shahapurkar, J. Aparicio, J. Mahler, V. Satish, K. Goldberg, and H. Claussen, "Industrial robot grasping with deep learning using a programmable logic controller (plc)," *arXiv preprint arXiv:2004.10251*, 2020.

[48] D. Morrison, P. Corke, and J. Leitner, “Learning robust, real-time, reactive robotic grasping,” *The International Journal of Robotics Research*, vol. 39, no. 2-3, pp. 183–201, 2020.

[49] S. James, P. Wohlhart, M. Kalakrishnan, D. Kalashnikov, A. Irpan, J. Ibarz, S. Levine, R. Hadsell, and K. Bousmalis, “Sim-to-real via sim-to-sim: Data-efficient robotic grasping via randomized-to-canonical adaptation networks,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 12 627–12 637.

[50] E. Karpas, S. Levine, P. Yu, and B. Williams, “Robust execution of plans for human-robot teams,” in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 25, 2015.

[51] R. A. Knepper, S. Tellex, A. Li, N. Roy, and D. Rus, “Recovering from failure by asking for help,” *Autonomous Robots*, vol. 39, no. 3, pp. 347–362, 2015.

[52] R. S. Andersen, O. Madsen, T. B. Moeslund, and H. B. Amor, “Projecting robot intentions into human environments,” in *2016 25th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*, IEEE, 2016, pp. 294–301.

[53] B. Sankaran, B. Pitzer, and S. Osentoski, “Failure recovery with shared autonomy,” in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2012, pp. 349–355.

[54] M. Quigley, J. Faust, T. Foote, and J. Leibs, “Ros: An open-source robot operating system,” *ICRA workshop on open source software*, vol. 3, no. 2, May 2009.

[55] *Google Scholar: Metrics for ros paper.* `https://scholar.google.com/scholar?cluster=143767492575573826&hl=sv&as_sdt=0,5&sciodt=0,5`, [Online; accessed 26-Nov-2020], 2020.

[56] *ROS 2*, `https://index.ros.org/doc/ros2/`, [Online; accessed 25-Feb-2019], 2019.

[57] G. Pardo-Castellote, “Omg data-distribution service: Architectural overview,” in *23rd International Conference on Distributed Computing Systems Workshops, 2003. Proceedings.*, May 2003, pp. 200–206.

[58] H. Fischer, M. Vulliez, P. Laguillaumie, P. Vulliez, and J.-P. Gazeau, “Rtrobmultiaxiscontrol: A framework for real-time multiaxis and multirobot control,” *IEEE Transactions on Automation Science and Engineering*, vol. 16, no. 3, pp. 1205–1217, 2019.

[59] E. Erős, M. Dahl, A. Hanna, P.-L. Götvall, P. Falkman, and K. Bengtsson, “Development of an industry 4.0 demonstrator using sequence planner and ros2,” in *Robot Operating System (ROS)*, Springer, pp. 3–29.

[60] I. A. Șucan and S. Chitta, *MoveIt!* `http://moveit.ros.org`, [Online; accessed 26-Feb-2019], 2018.

[61] I. A. Șucan, M. Moll, and L. E. Kavraki, “The Open Motion Planning Library,” *IEEE Robotics & Automation Magazine*, vol. 19, no. 4, pp. 72–82, Dec. 2012, `http://ompl.kavrakilab.org`.

[62] *ROS2 Grasp Library*, `https://github.com/intel/ros2_grasp_library`, [Online; accessed 23-Nov-2020], 2020.

[63] M. Cashmore, M. Fox, D. Long, D. Magazzeni, B. Ridder, A. Carreraa, N. Palomeras, N. Hurtós, and M. Carrerasa, “Rosplan: Planning in the robot operating system,” in *Proceedings of the Twenty-Fifth International Conference on International Conference on Automated Planning and Scheduling*, ser. ICAPS’15, Jerusalem, Israel: AAAI Press, 2015, pp. 333–341, ISBN: 978-1-57735-731-5.

[64] F. Rovida, M. Crosby, D. Holz, A. S. Polydoros, B. Großmann, R. P. A. Petrick, and V. Krüger, “Skiros—a skill-based robot control platform on top of ros,” in *Robot Operating System (ROS): The Complete Reference (Volume 2)*, A. Koubaa, Ed. Cham: Springer International Publishing, 2017, pp. 121–160, ISBN: 978-3-319-54927-9.

[65] E. Aertbeliën and J. De Schutter, “Etasl/etc: A constraint-based task specification language and robot controller using expression graphs,” in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sep. 2014, pp. 1540–1546.

[66] C. Paxton, A. Hundt, F. Jonathan, K. Guerin, and G. D. Hager, “Costar: Instructing collaborative robots with behavior trees and vision,” in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, May 2017, pp. 564–571.

[67] G. Canal, M. Cashmore, S. Krivić, G. Alenyà, D. Magazzeni, and C. Torras, “Probabilistic Planning for Robotics with ROSPlan,” in *Towards Autonomous Robotic Systems*, Springer International Publishing, 2019, pp. 236–250, ISBN: 978-3-030-23807-0.

[68] V. Krueger, F. Rovida, B. Grossmann, R. Petrick, M. Crosby, A. Charzoule, G. M. Garcia, S. Behnke, C. Toscano, and G. Veiga, “Testing the vertical and cyber-physical integration of cognitive robots in manufacturing,” *Robotics and Computer-Integrated Manufacturing*, vol. 57, pp. 213–229, 2019.

[69] M. Colledanchise and P. Ögren, *Behavior trees in robotics and AI: An introduction.* CRC Press, 2018.

[70] C. Legat and B. Vogel-Heuser, “A configurable partial-order planning approach for field level operation strategies of plc-based industry 4.0 automated manufacturing systems,” *Engineering Applications of Artificial Intelligence*, vol. 66, pp. 128–144, 2017, ISSN: 0952-1976.

[71] B. Wally, J. Vyskočil, P. Novák, C. Huemer, R. Šindelář, P. Kadera, A. Mazak-Huemer, and M. Wimmer, “Leveraging iterative plan refinement for reactive smart manufacturing systems,” *IEEE Transactions on Automation Science and Engineering*, vol. 18, no. 1, pp. 230–243, 2020.

[72] P. Falkman, B. Lennartson, and K. Andersson, “Specification of production systems using PPN and sequential operation charts,” in *2007 IEEE International Conference on Automation Science and Engineering*, Sep. 2007, pp. 20–25.

[73] K. Bengtsson, B. Lennartson, and C. Yuan, “The origin of operations: Interactions between the product and the manufacturing automation control system,” *IFAC Proceedings Volumes*, vol. 42, no. 4, pp. 40–45, 2009, 13th IFAC Symposium on Information Control Problems in Manufacturing.

[74] K. Bengtsson, P. Bergagard, C. Thorstensson, B. Lennartson, K. Akesson, C. Yuan, S. Miremadi, and P. Falkman, “Sequence planning using multiple and coordinated sequences of operations,” *IEEE Transactions on Automation Science and Engineering*, vol. 9, no. 2, pp. 308–319, Apr. 2012.

[75] P. Bergagård and M. Fabian, "Deadlock avoidance for multi-product manufacturing systems modeled as sequences of operations," in *2012 IEEE International Conference on Automation Science and Engineering: Green Automation Toward a Sustainable Society, CASE 2012, Seoul, 20-24 August 2012*, 2012, pp. 515–520.

[76] A. Theorin, K. Bengtsson, J. Provost, M. Lieder, C. Johnsson, T. Lundholm, and B. Lennartson, "An event-driven manufacturing information system architecture for industry 4.0," *International Journal of Production Research*, pp. 1–15, 2016.

[77] K. Bengtsson, E. Blomgren, O. Henriksson, L. Johansson, E. Lindelöf, M. Pettersson, and Å. Söderlund, "Emergency department overview - improving the dynamic capabilities using an event-driven information architecture," in *IEEE International Conference on Emerging technologies and factory automation (ETFA)*, 2016.

[78] J. Rintanen, K. Heljanko, and I. Niemelä, "Planning as satisfiability: Parallel plans and algorithms for plan search," *Artificial Intelligence*, vol. 170, no. 12-13, pp. 1031–1080, 2006.

[79] C. A. Knoblock, J. D. Tenenberg, and Q. Yang, "Characterizing abstraction hierarchies for planning.," in *AAAI*, 1991, pp. 692–697.

[80] *Industrial Path Solutions*, `https://industrialpathsolutions.com/`, [Online; accessed 10-Nov-2020], 2020.

[81] S. Patil, V. Vyatkin, and C. Pang, "Counterexample-guided simulation framework for formal verification of flexible automation systems," in *2015 IEEE 13th International Conference on Industrial Informatics (INDIN)*, IEEE, 2015, pp. 1192–1197.

[82] F. Brooks and H. Kugler, *No silver bullet.* April, 1987.