Article:
Gleirscher, M., Calinescu, R., Douthwaite, J. orcid.org/0000-0002-7149-0372 et al. (5 more authors) (Submitted: 2021) Verified synthesis of optimal safety controllers for human-robot collaboration. arXiv. (Submitted)

Verified Synthesis of Optimal Safety Controllers for Human-Robot Collaboration*

Preprint, compiled June 15, 2021

Mario Gleirscher
University of Bremen, Germany
mario.gleirscher@uni-bremen.de

Radu Calinescu
University of York, UK
radu.calinescu@york.ac.uk

James Douthwaite
University of Sheffield, UK
j.douthwaite@sheffield.ac.uk

Benjamin Lesage
University of York, UK
benjamin.lesage@york.ac.uk

Colin Paterson
University of York, UK
colin.paterson@york.ac.uk

Jonathan Aitken
University of Sheffield, UK
jonathan.aitken@sheffield.ac.uk

Rob Alexander
University of York, UK
rob.alexander@york.ac.uk

James Law
University of Sheffield, UK
j.law@sheffield.ac.uk

Abstract

We present a tool-supported approach for the synthesis, verification and validation of the control software responsible for the safety of the human-robot interaction in manufacturing processes that use collaborative robots. In human-robot collaboration, software-based safety controllers are used to improve operational safety, e.g., by triggering shutdown mechanisms or emergency stops to avoid accidents. Complex robotic tasks and increasingly close human-robot interaction pose new challenges to controller developers and certification authorities. Key among these challenges is the need to assure the correctness of safety controllers under explicit (and preferably weak) assumptions. Our controller synthesis, verification and validation approach is informed by the process, risk analysis, and relevant safety regulations for the target application. Controllers are selected from a design space of feasible controllers according to a set of optimality criteria, are formally verified against correctness criteria, and are translated into executable code and validated in a digital twin. The resulting controller can detect the occurrence of hazards, move the process into a safe state, and, in certain circumstances, return the process to an operational state from which it can resume its original task. We show the effectiveness of our software engineering approach through a case study involving the development of a safety controller for a manufacturing work cell equipped with a collaborative robot.

Keywords risk-informed controller synthesis · formal verification · probabilistic model checking · code generation · collaborative robot safety · digital twins

1 Introduction

Effective collaboration between humans and robots [Nicolaisen, 1985, Jones, 1986] can leverage their complementary skills. But such collaboration is difficult to achieve because of uncontrolled hazards and because sensing, tracking, and safety measures are either still unexploited in practice [Santis et al., 2008] or they are difficult to validate following state-of-the-art safety regulations [Chemweno et al., 2020]. Since the 1980s, remote programming (also called tele-programming) and simulation have led to some reduction of hazard exposure. However, the effectiveness of human-robot collaboration is still limited because of frequent conservative shutdowns, simplistic emergency stops, and unspecific error handling procedures. Extensive guarding arrangements interfere with manufacturing processes and mobile robot applications. But effective work processes and complex tasks require continuous close human-robot interaction (e.g. mutual take-over of tasks), mutual clarification of intent, and trading off risk [Hayes and Scassellati, 2013, Villani et al., 2018]. From an operator's perspective, robot movements need to be predictable, and potential impacts on the human body need to be attenuated. From a control perspective, the confident monitoring and control of the robot speed and the separation between machines and humans require high-quality stereo vision and laser scanners to distinguish several safety zones. A decade after these issues were discussed in Alami et al. [2006] and Haddadin et al. [2009], Ajoudani et al. [2017] emphasise that the complex safety challenges of collaborative robots (cobots for short, Gillespie et al., 2001) remain largely unresolved. Increasingly complex robotic systems reduce the ability to understand and mitigate risks.

*This research has received funding from the Assuring Autonomy International Programme (AAIP grant CSI: Cobot), a partnership between Lloyd's Register Foundation and the University of York, and from the UKRI project EP/V026747/1 'Trustworthy Autonomous Systems Node in Resilience'.

Safety is a major barrier to the more widespread adoption of cobots, with organisations such as manufacturers having to resort to sub-optimal processes due to safety concerns. Methods of ensuring safe human-robot interaction will deliver $7.3bn of savings, reducing costs of US-manufactured goods by 1% [Anderson, 2016]. However, little such method and tool support is available for engineers that have to implement and confidently assure cobot safety requirements, that is, the results of cobot hazard analyses and risk assessments [Chemweno et al., 2020].

Problem. Among the measures for improving cobot safety, the monitoring of application processes, the handling of critical events, and the mitigation of operational risk are the responsibility of software-based *safety controllers*. To facilitate smooth human-robot collaboration with minimal interruption, the software engineers responsible for developing these controllers need to closely consider the process with its variety and complexity of adverse events, such as unusual operator behaviour and equipment failure modes. This leads to complex requirements and design spaces for the safety controllers, so these engineers must address questions in several areas:

1. *Risk assessment.* Which controller minimises the probability of incidents in the presence of human and sensor errors?
2. *Controller synthesis.* Which design minimises nuisance to the human, maximises productivity, etc. while maintaining safety?
3. *Controller verification.* Does a controller handle hazards when detected and return the system to a useful safe state?
4. *Controller validation.* Does an implementation of the synthesised controller exhibit the intended behaviour?

Preliminary solution. In Gleirscher and Calinescu [2020], we provide initial answers to the first three questions, by introducing a preliminary software engineering approach for the synthesis of discrete-event safety controllers that meet safety requirements and optimise process performance in human-robot collaboration. We model the application (e.g. a manufacturing process) as a Markov decision process (MDP), and select correct-by-construction controllers from an associated design space. The process model describes the behaviour of all involved actors including the controller. To describe critical events (e.g. hazards) and controller actions (e.g. safety mode changes), we employ the notion of *risk structures* [Gleirscher, 2017, Gleirscher et al., 2021] implemented in the risk-informed controller designer YAP [Gleirscher, 2021, 2020] used for risk structuring, qualitative risk analysis, and synthesis of risk-informed safety controllers. In particular, YAP supports risk modelling and controller design with a domain-specific language and automates the transformation of risk structures into guarded command language and, in turn, MDPs. The preliminary approach from Gleirscher and Calinescu [2020] facilitates the verification of the safety of the MDP and of probabilistic reach-avoid properties of selected MDP policies including the controller. A verified controller extracted from such a policy detects hazards and controls their mitigation by the execution of a safety function, a transition to a particular safety mode, or a safer process task or activity. Furthermore, in certain circumstances, the controller returns the process to an operational state from which it can resume its original task.

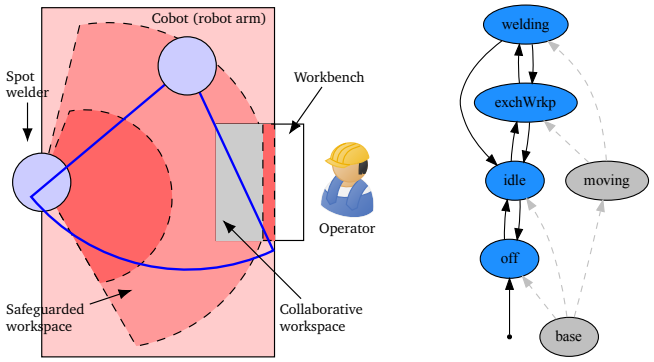
Contributions. In this paper, we extend our preliminary software engineering approach for safety controller synthesis [Gleirscher and Calinescu, 2020, Gleirscher, 2020] with multiple new capabilities, and we overcome several of its limitations, as explained below.

We refine the process and risk modelling to enable the synthesis of finer-grained controllers. For this, we provide a refined factor notion in Section 4.2.1. We improve and extend the verification stage by checking additional properties, increasing the confidence in the synthesised controllers. We employ the stochastic model synthesis tool EVOCHECKER [Gerasimou et al., 2018] in addition to the probabilistic model checker PRISM [Kwiatkowska et al., 2011], extending the verification capabilities of the optimal synthesis procedure to significantly more complex combinations of controller requirements.

We translate the controllers into executable code for a digital twin framework (DTF) to foster controller validation in a realistic environment. We assure the correctness of this translation by reusing properties from model checking in run-time verification. The DTF validation of the safety controller is based on randomised use-case tests that satisfy well-defined state-based coverage criteria.

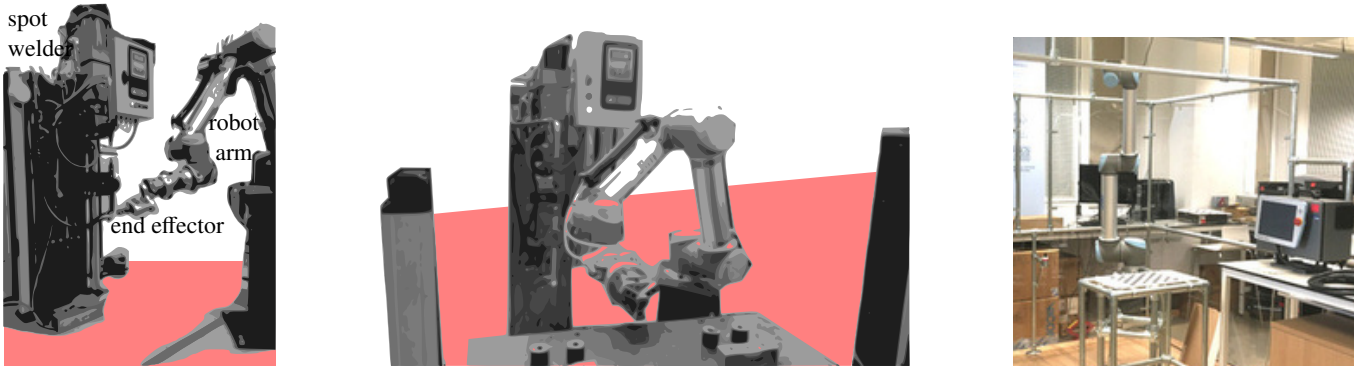
We demonstrate in the DTF how the synthesised safety controller can automatically resume the nominal procedure after the mitigation of hazards. We provide experimental results that confirm our safety controller's ability to achieve an increased process utility in addition to ensuring high levels of freedom from accidents.

Overall, we increase the level of technical detail, to improve the understanding and reproducibility of our results. Section 2 introduces our case study as a running example, and Section 3 provides the theoretical background. We describe our process modelling and controller design method in Section 4, our approach to verified optimal synthesis in Section 5, and explain in Section 6 how we implement, deploy, and validate the synthesised controllers in a realistic purpose built digital twin. We evaluate and discuss our approach in Section 7. Section 8 highlights related work. We conclude with a short summary in Section 9.



(a) Work cell with cobot (conceptual, top view) (b) Process activities

Figure 1: Work cell concept (a) and activities in the manufacturing process (b) performed by the operator, the robot, and the spot welder, classified by the activity groups *moving* and *base* (in light gray)



(a) Safeguarded area (company) (b) Workbench (company) with two handover locations (left and right receptacle) for part exchange (c) Replica (research lab)

Figure 2: Actual (a, b) and replicated (c) work cell with cobot

2 RUNNING EXAMPLE: MANUFACTURING COBOTS

Figure 2 shows a cobot-equipped manufacturing *work cell* at a UK company (with the pictures anonymised for confidentiality reasons) and replicated in a testbed at the University of Sheffield (Figure 2c). In the corresponding process, $\mathcal{P}$, an operator, a stationary collaborative robotic manipulator (robot arm for short), and a spot welder (Figure 1a) collaborate repetitively on several activities (Figure 1b).[2] Previous safety analysis (i.e., hazard identification, risk assessment, and requirements derivation) resulted in two sensors. The first one is a range finder using a rotating laser beam (indicated with the highlighted area at the bottom of Figure 2a) to determine the distance between the spot welder and a person or an object intruding into the highlighted area and triggering a slow down or an emergency stop if the intruder approaches the spot welder. The second one is a light barrier (the highlighted curtain indicated in Figure 2b) triggering such a stop if something like a person's arm reaches across the workbench while the robot or the spot welder are active. Table 1 shows our partial safety analysis of the cell following the guidance in Section 1. The right column specifies safety goals against each accident and controller requirement candidates (e.g. mode-switch requirements) handling each latent cause in the left column, and indicating how the hazard is to be removed. The running example is part of a case study organised around the AAIP project CSI: Cobot.[3] Due to Covid-19 restrictions limiting access to physical facilities during a critical phase of the project, we use a digital twin of the work cell as a target platform to deploy and validate synthesised safety controllers.

3 BACKGROUND

This section summarises the background of the presented approach, particularly, cobot safety, probabilistic model checking, system safety analysis, and risk-informed controller modelling.

[2] For the sake of simplicity, we use the notion of an *activity* as a hypernym describing a task, a situation, a use case, or a scenario.
[3] See https://www.sheffield.ac.uk/sheffieldrobotics/about/csi-cobot.

Table 1: Our partial safety analysis of the manufacturing cell referring to the measures recommended in ISO/TS 15066 [2016]

Id	**Critical Event** (risk factor)	**Safety Requirement**
	Accident (to be prevented or alleviated)	**Safety Goal**
RC	The Robot arm harshly Collides with an operator.	The robot shall *avoid* harsh active collisions with the operator.
WS	Welding Sparks cause operator injuries (skin burns).	The welding process shall *reduce* sparks injuring the operator.
RT	The Robot arm Touches the operator.	The robot shall *avoid* active contact with the operator.
	Latent Cause (to be mitigated timely)†	**Controller Requirement**‡
HRW	The Human operator and the Robot use the Workbench at the same time.	(m) The robot shall perform an appropriate mitigation (e.g. a safety-rated monitored stop) and (r) resume *normal operation* after the *operator* has left the *shared workbench*.
HW	The Human operator is entering the Workbench while the robot is away from the workbench.	(m) If the robot moves a workpiece to the workbench then it shall switch to *power & force limiting* mode and (r) resume *normal operation* after the *operator* has left the *workbench*.
HS	The Human operator has entered the Safeguarded area while the robot or the spot welder are active.	(m) The *spot welder* shall be *switched off*, the *robot* to *speed & separation monitoring*, and the operator be notified to leave. (r) Robot and spot welder shall resume normal mode after the operator has left.
HC	The Human operator is Close to the welding spot while the robot is working and the spot welder is active.	(m) The *spot welder* shall be *switched off*, the *robot* to *safety-rated monitored stop*. (r) Both shall resume *normal or idle mode with a reset procedure* after the operator has left.

†m: mitigation requirement, r: resumption requirement, ‡subjected to generalisation to define a controller design space

Table 2: Cobot safety measures associated to stages in the causal chain of events

Stage	Type of Measure	Examples
Hazard prevention	1. Safeguard/barrier	Fence, cage, interlock
	2. IT safety	*Verified*† *safety controller*
	3. IT security	Security-verified† (safety) controller
Hazard mitigation & accident prevention	4. Reliability	Fault-tolerant scene interpretation
	5. Workspace intrusion detection	Speed & separation monitoring, safety-rated monitored stop
	6. Shift of control	Hand-guided operation
Accident mitigation (alleviation)	7. Power & force limitation	Low weight parts, flexible surfaces; variable impedance, touch-sensitive, & force-feedback control
	8. System halt	Emergency stop, dead-man's switch

†avoidance of development or programming mistakes

3.1 Robot Safety: From Industrial to Collaborative

Hazards from robots have been studied since the advent of industrial robotics in the 1970s, resulting in risk taxonomies based on workspaces, tasks, and human body regions [Sugimoto, 1977, Jones, 1986, Alami et al., 2006, Haddadin et al., 2009, Wang et al., 2017, Kaiser et al., 2018, Matthias et al., 2011, Marvel et al., 2015]. The majority of hazards are *impact hazards* (e.g. unexpected movement, reach beyond area, dangerous workpieces, hazardous manipulation), *trapping hazards* (e.g. operator locked in cage), and *failing equipment* (e.g. valve, cable, sensor, controller). In the 1980s, robots were programmed interactively by operators being in the cage while powered, which had caused frequent accidents from trapping and collision. However, from the late 1990s on, the increased use of tele-programming contributed to the reduction of accidents related to such hazards.

Addressing hazards outside the programming stage involves the examination of each *mode of operation* (e.g. normal, maintenance) for its hazardous behaviour, and the use of safety controllers to trigger mode-specific *safety measures* [Jones, 1986]. Malfunction diagnostics (e.g. fault detection, wear-out monitoring) can further inform these controllers. Table 2 shows a variety of measures [Santis et al., 2008] to prevent or mitigate hazards and accidents by reducing the probability of the occurrence and the severity of the consequences of these hazards. If these measures use electronic or mechatronic equipment, we speak

of *functional*[4] measures (e.g. safety modes as exemplified below) and of intrinsic measures otherwise (e.g. a fence around a robot, flexible robot surfaces). Functional measures focusing on the correctness and reliability of a controller (a programmable electronic or software system) are called dependability measures [Alami et al., 2006, Avizienis et al., 2004]. Functional measures are said to be passive if they focus on severity reduction (e.g. force-feedback control), *active* otherwise. In this work, we focus on the verified synthesis [Kress-Gazit et al., 2018] of safety controllers that realise active functional safety measures.

Standardisation of safety requirements for industrial robots [Sugimoto, 1977] culminated in ANSI/RIA R15.06, ISO 10218 [2011], 13482, and 15066. Following ISO 10218, such robot systems comprise a robot arm, a robot controller, an end-effector, and a work piece (see, e.g. Figure 1a). According to Helms et al. [2002] and Kaiser et al. [2018], one can distinguish four scenarios of human-robot interaction: (i) Encapsulation in a fenced robot work space, (ii) co-existence without fencing but separation of human and robot work space, (iii) cooperation with alternative exclusive use of shared work space, and (iv) collaboration with simultaneous use of shared work space and close interaction. Cooperation and collaboration are the two most interactive of these scenarios and motivate our work. In collaborative operation, the operator and the cobot [Gillespie et al., 2001] can occupy the collaborative workspace simultaneously while the cobot is performing tasks [ISO/TS 15066, 2016, Pt. 3.1]. The *collaborative workspace* has to be a subset of the *safeguarded workspace*.

Based on these definitions, ISO 15066 recommends four *safety modes*. First, a *safety-rated monitored stop* is an active functional measure realised as a mode where the robot is still powered but there is no simultaneous activity of the robot and the operator in the shared workspace. Second, *hand-guided operation* refers to a mode with zero-gravity control, that is, control without actuation beyond the compensation of gravity, solely guided by an operator. Hand guidance requires the robot to be in a compliant state, with control exerted by the operator through physical manipulation. Third, *speed & separation monitoring* as an active functional measure refers to a mode where speed is continuously adapted to the distance of the robot and an operator. Forth, *power & force limiting* is a mode with reduced impact of the robot on the human body and the robot's power and applied forces are limited. In this mode, a robot should not impact a human with more than a defined force, with acceptable forces mapped out for different impact points on the body. Heinzmann and Zelinsky [2003] propose such a mode always active during a collaborative activity described as a discrete-event controller. Long et al. [2018] propose a distance-triggered scheme to switch between nominal (max. velocity), reduced (speed limiting), and passive (hand-guided) operating modes. Kaiser et al. [2018] and Villani et al. [2018] describe and combine these modes with work layouts.

In addition to these safety modes, Alami et al. [2006] highlight the necessity of a general shift from robots whose motion is controlled only by following a pre-specified route of positions (also known as position control) to robots whose motion control minimises contact forces and energy (also known as interaction control). Overall, interaction-controlled robots, with less pre-planning and fewer assumptions on workspace structure and robot actions, can exploit the mentioned safety modes more effectively than position-controlled robots with extensive pre-planning and stronger assumptions.

3.2 Probabilistic Model Checking and Trace Checking

The proposed approach employs MDPs, and corresponding sets of discrete-time Markov chains (DTMCs), as a formal model of the process $\mathcal{P}$, and uses the policy space of an MDP, describing the degrees of freedom for decision making, to model the design space for verified controller synthesis.

Definition 1. *Markov decision process (MDP). Given all distributions $Dist(\cdot)$ over a sort (e.g. an action alphabet $A_\mathcal{P}$ of a process $\mathcal{P}$), an MDP is a tuple $\mathcal{M} = (S, s_0, A_\mathcal{P}, \delta_\mathcal{P}, L)$ with a set S of states, an initial state $s_0 \in S$, a probabilistic transition function $\delta_\mathcal{P} \colon S \times A_\mathcal{P} \to Dist(S)$, and a map $L \colon S \to 2^{AP}$ labelling S with atomic propositions AP [Kwiatkowska et al., 2007].*

Given a map $A \colon S \to 2^{A_\mathcal{P}}$, $|A(s)| > 1$ signifies non-deterministic choice in s. Choice resolution for S forms a policy.

Definition 2. *Policy. A* policy *is a map $\pi \colon S \to Dist(A_\mathcal{P})$ s.t. $\pi(s)(a) > 0 \Rightarrow a \in A(s)$. π is* deterministic *if $\forall s \in S\ \exists a \in A(s) \colon \pi(s)(a) = 1 \wedge \forall a' \in A_\mathcal{P} \backslash \{a\} \colon \pi(s)(a') = 0$.*

In this paper, we restrict our self to the consideration of deterministic policies.[5] Let $\Pi_\mathcal{M}$ be the space of all such policies for $\mathcal{M}$. Then, *action rewards* defined by a map $r^q_{action} \colon S \times A_\mathcal{P} \to \mathbb{R}_{\geqslant 0}$ allow the comparison of policies in $\Pi_\mathcal{M}$ based on a quantity q.

Verification of $\mathcal{M}$ can be done using probabilistic computation tree logic (PCTL) whose properties over AP are formed by

$$\phi ::= true \mid ap \mid \neg\phi \mid \phi \wedge \phi \mid \mathbf{E}\,\varphi \mid \mathbf{A}\,\varphi \mid \mathbf{P}_{\sim b|[\min|\max]=?}\,\phi \qquad \text{and} \qquad \varphi ::= \mathbf{X}\,\phi \mid \phi\,\mathbf{U}^{[\sim b]}\,\phi$$

with $ap \in AP$; an optional bound $b \in \mathbb{N}_+$ for $\mathbf{U}^{[\sim b]}$ with $\sim \in \{<, \leqslant, =, \geqslant\}$; the quantification operators $\mathbf{P}_{\sim b|[\min|\max]=?}\,\phi$ to verify (or with $=?$, to quantify) probabilities, and $\mathbf{S}_{\sim b|=?}\,ap$ to determine long-run probabilities of an atomic proposition ap.[6] States

[4] *Functional safety* (see IEC 61508, ISO 26262) deals with the dependability, particularly, correctness and reliability, of *critical* programmable electronic systems. Safety functions or "functional measures" are the archetype of such systems.

[5] More precisely, we only consider memoryless policies, with some restrictions on MDP policy synthesis, however, not relevant for our purposes.

[6] With $[\cdot]$ and $\cdot \mid \cdot$, we denote optional and alternative syntactic choice in sub- or superscript language elements. $[\cdot]$ is also a mandatory notational element to encapsulate a formula following a temporal operator.

correspond to valuations of state variables of type $\mathbb{B}$, $\mathbb{N}$, or $\mathbb{R}$. Hence, propositions in AP are of the form $x \sim f$ for a variable x and a function (or constant) $f\colon S \to \mathbb{R} \cup \mathbb{B}$. We use the abbreviations $\mathit{false} \equiv \neg \mathit{true}$, $\mathbf{F}\,\phi \equiv \mathit{true}\,\mathbf{U}\,\phi$, $\mathbf{G}\,\phi \equiv \neg\,\mathbf{F}\,\neg\phi$, and $\phi\,\mathbf{W}\,\psi \equiv \phi\,\mathbf{U}\,\psi \vee \mathbf{G}\,\phi$. The PCTL extension $\mathbf{R}^{q}_{\sim b|[\min|\max]=?}[\mathbf{F}\,\phi \mid \mathbf{C}^{[\sim b]}]$ calculates reachability rewards ($\mathbf{F}\,\phi$) and (optionally bounded) accumulative action rewards ($\mathbf{C}^{[\sim b]}$). With $\mathcal{M} \models \phi$ ($s \models \phi$), we state that the MDP $\mathcal{M}$ from state s_0 (the state $s \in S$) *satisfies* the property ϕ. Let $[\![\phi]\!]_S$ denote the largest subset $S' \subseteq S$ with $\forall s \in S'\colon s \models \phi$ for a state predicate ϕ.

For the checking of a recorded trace t of a system, we use fragments of linear and metric temporal logic whose properties over AP are formed by the propositional fragment as defined above and by $\mathbf{F}$, $\mathbf{G}$, and $\mathbf{U}_I$, with a finite interval $I \subset \mathbb{R}_{\geqslant 0} \times \mathbb{R}_{\geqslant 0}$. Comprehensive treatments of PCTL, linear and metric temporal logic can be found in, for example, Kwiatkowska et al. [2007], Basin et al. [2015] and Baier and Katoen [2008].

The concise construction of $\delta_{\mathcal{P}}$, the behaviour of $\mathcal{P}$, can be facilitated by using a flavour of probabilistic guarded command language (pGCL), for example, as implemented in the PRISM tool [Kwiatkowska et al., 2007, 2011]. Guarded commands have the form $[\alpha]\ \gamma \longrightarrow \upsilon$ with an event (or action) label α and a probabilistic update υ applicable to $s \in S$ only if $s \models \gamma$, where γ is an expression in the propositional fragment of PCTL.[7] With $+$ for probabilistic choice and $\&$ to compose assignments, general updates are defined as $\upsilon ::= \pi_1\colon \upsilon_1 + \dots + \pi_n\colon \upsilon_n$ with $\Sigma_{i\in 1..n}\pi_i = 1$ and υ_i being a multiple assignment $\upsilon_i ::= x'_{i_1} = f_{i_1}\ \&\ \dots\ \&\ x'_{i_n} = f_{i_n}$ to state variables x_k based on functions f_j.

3.3 Risk Modelling for Controller Design

We view an application (e.g. Section 2) as a process, $\mathcal{P}$, monitored and influenced by a safety controller to mitigate hazards and prevent accidents. Critical events, such as accidents (or mishaps), their causes, and causal factors (e.g. hazards), are state properties. We express these properties as subsets of S. In particular, *mishaps* $\underline{F} \subset S$ are undesired states (e.g. all states where a person is injured by welding sparks). For a mishap $\underline{f} \subset \underline{F}$, we further define a subset $\Xi_{\underline{f}} \subset S$ from which $\underline{f}$ is reachable, for example, all states in S where the operator is near the spot welder while the latter is active. We call $\Xi_{\underline{f}}$ the *causes* of $\underline{f}$. Causes are intersections of *causal factors*, in particular, factors related to the subject of protection (e.g. the operator to be protected by the safety controller when being near the spot welder) and a *hazard* f as a causal factor related to the system (e.g. the spot welder being active; Leveson 1995, 2012). We call causes latent or *controllable*[8] if there are sufficient resources to prevent the accident (e.g. time for removing f by transition to $S \setminus \Xi_{\underline{f}}$). Controllability can be justified, for example, by assuming that if the spark flow is low there is a small time span left for the spot welder to be stopped or the operator to leave without leading to an accident. f can also refer to states in $S \setminus \Xi_{\underline{f}}$ being critical because certain events (e.g. an operator approaches the spot welder) cause a transition to $\Xi_{\underline{f}}$, and possibly $\underline{f}$, if f stays active, further conditions hold, and no safety measures are put in place promptly. Below, in Section 4.2.1, we will use state propositions to identify the discussed critical events.

Based on these notions, risk modelling can be facilitated by specifying risk factors and combining them into *risk structures* [Gleirscher et al., 2021]. A *risk factor* f is a labelled transition system (LTS) modelling the life cycle of a critical event in terms of phases. In its basic form, f has the phases inactive ($\not{f}$), active (f), mitigated ($\bar{f}$), and mishap ($\underline{f}$). Transitions between these phases signify endangerment events (e) as well as mitigation (m) and resumption (r) actions. A risk structure from a factor set F (e.g. column **Id** in Table 1) operates over a *risk (state) space* $R(F) = \times_{\mathsf{f}\in F} Ph_{\mathsf{f}}$ with $Ph_{\mathsf{f}} = \{\not{f}, f, \bar{f}, \underline{f}\}$. Furthermore, let $\Xi \subset S$ be the set of states labelled with at least one cause, describing the abstract state where any critical event has at least been sensed by the controller (e.g. HC with its handling about to start, i.e., the controller transitioning from $\not{HC}$ to HC). We call Ξ the *non-accident F-unsafe region*. One can hypothesise relationships between critical events using factor dependencies. Such relationships can be identified and justified by, for example, a hazard operability study, a failure mode effects analysis, or a fault tree analysis. Further details about risk structures will be introduced along with our approach explained below in Section 4.

3.4 Digital Twins

We focus on digital twins as a platform for controller deployment and validation. Digital twinning is a key Industry 4.0 technology for enabling industrial automation, smart processes, and process autonomy [Negri et al., 2017, Bolton et al., 2018]. Together with the Internet of Things and machine-to-machine communication, digital twinning enables both the creation of a better data infrastructure and the adoption of smart manufacturing technologies. A digital twin provides greater access to data relating to, and control over, a physical system, and has particular value in the design, implementation, and evaluation of processes and safety controllers.

Kritzinger et al. [2018] and Tao et al. [2018] define a digital twin as:

> "A digital twin is an integrated multi-physics, multi-scale, probabilistic simulation of a complex product and uses the best available physical models, sensor updates, etc., to mirror the life of its corresponding twin."

[7] We use $\longrightarrow$ to separate guard and update expressions and $\rightarrow$ both for logical implication and the definition of mappings.
[8] As opposed to immediate causes with limited or no risk handling controls.

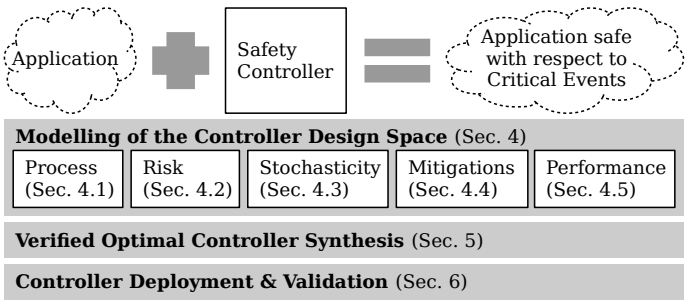


Figure 3: Stages of the proposed approach to safety controller synthesis

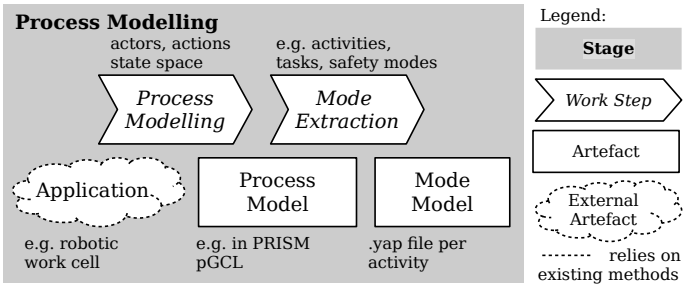


Figure 4: Overview of the work steps and artifacts of the process modelling stage

More specifically, a digital twin is a digital representation of a physical system that operates in parallel[9] with the real system. This concurrency can persist throughout the life-cycle of the physical system. Communication between the physical twin and its digital representation is bilateral, and as a result, both can be mutually informed by real-world or simulated sensor data, requested actions and decisions. As a means for safety verification and analysis, a digital twin presents: (i) A faithful representation of the process domain and state space, (ii) a means to interrogate and collect data that may not be readily available from the physical system (independent of hardware limitations), and (iii) an interface to the physical twin through which real-world responses to new safety procedures can be demonstrated.

4 MODELLING FOR THE SYNTHESIS OF SAFETY CONTROLLERS

Figure 3 provides an overview of the proposed approach to the synthesis of safety controllers. The main idea is that the control engineer designs a safety controller on top of an application, in this instance, comprising activities where humans and robots collaborate. The intention of the deployed controller is to increase safety with respect to the critical events under consideration.

In the *modelling* stage, the control engineer creates several models to obtain a design space including controller candidates to select from during synthesis. When modelling the *process* of the application, the engineer as a domain expert describes the actions performed by any of the actors in the application. The engineer then performs a risk analysis resulting in a *risk model* that informs the process model with a notion of operational risk. The abstraction chosen for the model and a usual lack of knowledge about process details require the integration of *stochasticity* and *performance* estimates into the model. With the risk- and performance-informed stochastic process model, the engineer can specify controller behaviour in form of a *mitigation* model. In the *controller synthesis* stage, an abstract controller is automatically synthesised from the design space according to risk- and performance-based optimality criteria. Finally, in the *deployment and validation* stage, this controller is translated automatically into an executable form. The three stages are supported by the tools YAP, PRISM, EVOCHECKER, and the digital twin framework. These stages are detailed in the following sub-sections and illustrated with several examples from our case study introduced in Section 2.

4.1 Modelling Processes

For process modelling (Figure 4), we use pGCL to specify the actions of the process. The process model defines the state space to be manipulated by these actions and includes the actions of the cobot and the environment including human operators. We group actions into *activities* as abstractions of the process. Activities describe certain tasks and facilitate the transition to risk modelling by structuring hazard identification and the creation of a hazard list. The *mode model* resulting from this abstraction step is an abstract LTS.

[9] A digital twin must be able to operate synchronously with the physical system, but asynchronous operation is also permissible.

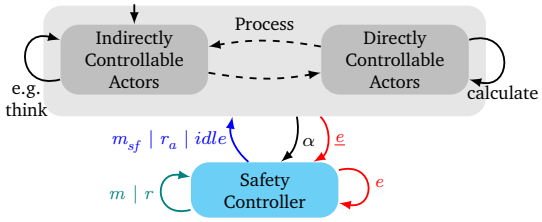


Figure 5: Execution from the viewpoint of the safety controller. The execution is coordinated by passing a single token between two groups of actors (dashed arcs) performing actions in the process. The controller can perform an update or issue control inputs to the process after every atomic event occurring in the process.

We describe the process, $\mathcal{P}$, as a set of guarded commands, distinguishing *actions* of relevant actors (e.g. a robot arm, a spot welder, an operator) and the safety controller from *events* of a sensor module and shared "manipulables" (e.g. workpiece support). Following Section 3.2, the structure of the guarded commands describing the behaviour in $\mathcal{P}$ follows the pattern

$$[\alpha] \ \cdots \wedge \gamma \longrightarrow \upsilon \ + \ \ldots$$

with an action label α, an action-specific condition γ and a generic update υ (see Section 3.2) being part of the overall guard and update expressions. The state space S is built from discrete variables (cf. Example 1) capturing the world state (e.g. robot location, workbench status), sensory inputs (e.g. range finder), control outputs (e.g. robot behaviour, notifications), user inputs (e.g. start button), and modes (e.g. activities, safety modes).

4.1.1 *Process Execution from a Controller's Perspective*

To account for the interaction of the safety controller with the process, we include a fair cyclic execution scheme into the process model. This scheme emulates the simultaneity of the controller and the process. Execution steps alternate between the controller and a set $\mathcal{A}$ of actors, ensuring in each cycle that each actor can take its turn (Figure 5). Regarding the way the controller can influence the process, we distinguish between directly controllable actors (e.g. cobot, spot welder) and indirectly or not controllable actors (e.g. human operator). Both kinds of actors can perform logical and physical actions following two corresponding command patterns:

$$[\alpha] \ ok_p \wedge \cdots \wedge \gamma \longrightarrow \upsilon \ + \ \ldots \qquad \text{(logical action)}$$

$$[\alpha] \ ok_p \wedge \cdots \wedge \gamma \longrightarrow \upsilon \,\&\, t' = sc \ + \ \ldots \qquad \text{(physical action)}$$

where ok_p guards the turn of an actor p and t stores the token passed between the safety controller and the other actors. $t' = sc$ denotes the safety controller's turn. ok_p can include a condition for terminating execution when reaching a final (or goal) state, resulting in $ok_p \equiv t = p \wedge \neg final$. The controller can stay idle or, most eagerly, intervene whenever one of the actors has completed an action. This scheme is more restrictive than the CSP-style[10] generalised parallel composition of concurrent processes available by default in tools such as PRISM [Kwiatkowska et al., 2011].

4.1.2 *Modelling Activities and Safety Modes*

To facilitate the design of a powerful class of safety controllers, we organise actions (e.g. grab work piece, move robot arm to spot welder) of $\mathcal{P}$'s controllable actors using mode variables, here, one variable for the engagement of each actor in an activity (e.g. ract) and one for the safety mode of the whole process (safmod). Mode variables group actions and, from a controller perspective, allow high-level process control by filtering enabled actions. Thus, the structure of the guarded commands for $\mathcal{P}$ is refined according to the pattern

$$[\alpha] \ ok_p \wedge \gamma_{sm} \wedge \gamma_a \wedge \gamma \longrightarrow \upsilon \,[\&\, t' = sc] \ + \ \ldots$$

where γ_a guards the enabling of actions in certain activities (e.g. exchange work piece, Figure 1b) and γ_{sm} in certain safety modes (e.g. speed & separation monitoring). We obtain safety- and task-aware actions by conjoining $\gamma_{sm} \wedge \gamma_a$. The update of t is optional to combine certain actor-internal updates into atomic actions.

Example 1. *The following listing describes part of the enumerations used to define the discrete state space S of $\mathcal{M}$.*

```
1 // spatial locations
2 const int atTable = 0; const int sharedTbl = 1; const int inCell = 2; const int atWeldSpot = 3;
3 // range finder signals
4 const int far = 0; const int near = 1; const int close = 2;
5 // notification signals
6 const int ok = 0; const int leaveArea = 1; const int resetCtr = 2;
```

[10] Following the synchronous interleaving semantics of Hoare's Communicating Sequential Processes.

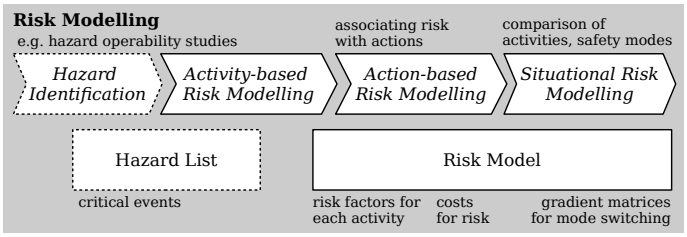


Figure 6: Overview of the work steps and artifacts of the risk modelling stage

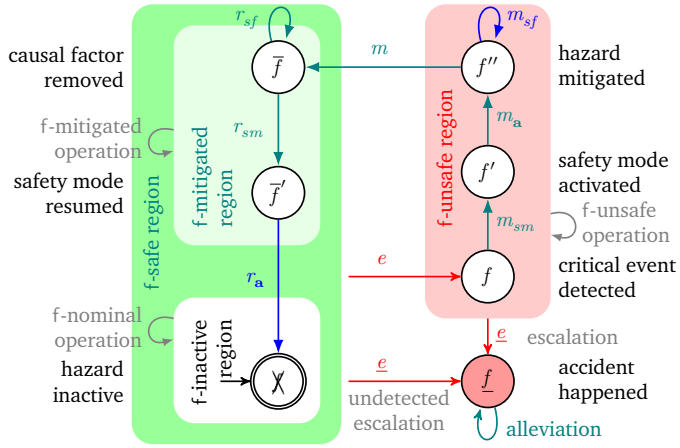


Figure 7: Refined notion of a risk factor f. The states of this LTS are called phases, the transitions are labelled with events and actions. Endangerment e (monitored) and accident $\underline{e}$ events; actions m_* and r_* of the safety controller with the actions m_{sf} and r_a where the controller interacts with the process waiting for a response; in gray, arbitrary process actions not controllable or observable by the safety controller.

The following commands specify two actions, r_moveToTable *and* r_grabLeftWorkpiece*, for the actor* robotArm *in the activity* exchWrkp.

```
1 [r_moveToTable] OK_wc
2   & (safmod=normal|safmod=ssmon|safmod=pflim)
3   & ract=exchWrkp & (rloc != sharedTbl)
4   & (((wps!=right) & reffocc=1) | (wps=left & (reffocc=0)))
5   -> (rloc'=sharedTbl)&(turn'=sc);
6 [r_grabLeftWorkpiece] OK_wc
7   & (safmod=normal|safmod=ssmon|safmod=pflim|safmod=hguid)
8   & ract=exchWrkp & rloc=sharedTbl & reffocc=0 & wps=left
9   -> (reffocc'=1)&(wpfin'=0)&(wps'=empty)&(turn'=sc);
```

4.2 *Modelling Risk*

For our synthesis approach to lead to correct and effective controllers, we need an expressive risk model (Figure 6). To obtain such a model, we translate the hazard list into a set of risk factors (Section 4.2.1). For this, we transcribe safety analysis results into factor LTSs (Section 3.3). Then, we define a risk profile for each of the process actions (Section 4.2.1). For each hazard in the hazard list, the *risk profile of an action* describes the risk that a performance of the action results in an accident related to this hazard. The final step of risk modelling consists of capturing risk-related situational change (i.e., a change of activity or safety mode) in the process with a *risk gradient* (Section 4.2.2). For this, we associate a numerical measure with each activity transition (i.e., each situational change), that describes the change in overall risk level. We proceed analogously with the definition of safety modes and the corresponding risk assessment.

4.2.1 *Refined Risk Factors*

Shown in Figure 7, we develop and use a refinement of the notion of a risk factor introduced in Section 3.3. As before, for a factor f, we refer to any state of $\mathcal{P}$ where f is inactive as the f-*inactive region* or, equivalently, as $\not{f}$. Any state of $\mathcal{P}$ where f has occurred and any causal factor is still active is subsumed by the f-*unsafe region*, which includes three phases: hazard detected (f), safety mode activated (f'), and hazard mitigated (f''). The f-*mitigated region* refers to any state deviating from $\mathcal{P}$'s nominal state but with any causal factor of f removed. The f-mitigated region includes the phase $\bar{f}$, reached when causal factors of f have been removed, and $\bar{f}'$, reached after deactivating a safety function and resuming from the current safety mode. The phases $\bar{f}$, $\bar{f}'$, and $\not{f}$ together constitute the f-*safe region* of $\mathcal{P}$.

When a critical event e is detected in the f-safe region, f is switched to f. For a critical event, we distinguish its ground truth predicate χ (i.e., the cause) from the detector (or monitoring) predicate ζ (i.e., the sensor) where $\chi \Leftrightarrow \zeta$ in case of perfect sensing.

Critical event detection follows the pattern

$$[e]\ ok_S \wedge \zeta \wedge rel(a,\mathsf{f}) \wedge \neg(f \vee \underline{f}) \longrightarrow f$$

where $rel(a,\mathsf{f})$ is a relevance indicator determining whether to react on ζ in an activity a or, more generally, in a particular situation. With $\neg(f \vee \underline{f})$, we ignore re-occurrences while f is active or after an accident. *Idling* of the safety controller (Figure 5) is captured by

$$[idle]\ ok_S \wedge \neg\chi \wedge (\not{f} \vee \bar{f}) \longrightarrow t' = next(p)$$

where $next(p)$ passes the token from the actor p to the next actor in $\mathcal{P}$ (Figure 5). The other actions will be explained in Section 4.4.

Activity-based Risk. Factor LTSs guide the formalisation of hazards, their causes, and mishaps and the events in the causal chain (e.g. a mishap event leads to a mishap state). This way, factors support the design of mitigations to reduce accidents, and alleviations to reduce consequences. Hence, all critical events related to an activity should be translated into a factor set.

Action-based Risk. We define an action multi-reward structure $r^{risk}_{action}\colon S \times A_{\mathcal{P}} \times F \to \mathbb{R}_{\geqslant 0}$ over $\mathcal{M}$ to measure overall and factor-specific risk in $\mathcal{P}$. Action rewards are guarded, requiring f being active. For example, if $r^{risk}_{action}(s,a,\mathsf{f}) > 0$ then f is active in state s.

4.2.2 Situational Risk

The decision space of the safety controller for mitigations and resumptions includes choices from sets of safety modes (S_{sm}) and activities (S_a) to switch to from a particular mode and activity. To simplify the design space in $\mathcal{M}$, we resolve this choice during generation (Section 5.1) using a *categorical risk gradient* $\nabla r = [\partial r/(a \to a'), \partial r/(sm \to sm')]^T$. For categorical variables x, we allow the convention $\partial y/(x \to x')$ denoting the change of y when x changes its value from x to x'.

We implement ∇r with two skew-diagonal matrices $\mathbb{R}^{|S_{sm}|\times|S_{sm}|}$ and $\mathbb{R}^{|S_a|\times|S_a|}$, assuming that they can be (manually) derived from safety analysis based on the following justification. Assume that, in the activities $a, a' \in S_a$, actors vary in physical movement, force, and speed. If a means more or wider movement, higher force application, or higher speed than in a', then a change from a to a' will likely reduce risk. Hence, $\partial r/(a \to a') \geqslant 0$. Similarly, assume that the safety modes $sm, sm' \in S_{sm}$ vary $\mathcal{P}$'s capabilities by relaxing or restricting the range and behavioural shape of the permitted actions. If sm relaxes capabilities more than sm', then a change from sm to sm' will likely reduce risk. Again, $\partial r/(sm \to sm') \geqslant 0$. Skew-diagonality of the matrices provides that $\partial r/(sm' \to sm) \leqslant 0$. The matrices for ∇r can be specified as part of a YAP model.

Example 2. *We instantiate* f *according to Figure 7 for the hazard* HC *from Table 1. When an operator approaches an active spot welder, an event* e^{HC} *is detected, activating* HC *by a transition to a state where the predicate HC holds, a state in* $[\![HC]\!]_R \subset R(F)$. *The safety controller will then start with performing mitigations to reach phase* $\overline{HC}$. *The handling of* HC *includes the switching to a speed & separation monitoring mode, issuing an operator notification, and waiting for the operator's response. From* $\overline{HC}$, *the controller can continue with resumptions (e.g. switching from speed & separation monitoring back to normal) to finally return to phase* ~~HC~~ *where both HC and* $\overline{HC}$ *are false. Further endangerments (e.g. erroneous robot movement) may reactivate* HC *from phase* $\overline{HC}$ *or* $\overline{HC}'$. *An accident* $\underline{e}^{\mathsf{HC}}$ *with the spot welder or robot arm, leading to phase* $\underline{HC}$, *can occur, for example, because of a faulty range finder responsible for the detection of* e^{HC} *or too slow mitigations* m^{HC}_{sm} *and* m^{HC}_{a}. *In* $\underline{HC}$, *alleviations (e.g. flexible robot surfaces, protective goggles) can reduce certain consequences. We start encoding* HC *in a* YAP *model below.*

```
1 HC desc "(H)uman (C)lose to active spot welder and cobot working"
2   requiresOcc (HS) // imposes relationship between cause/guard conditions of HS and HC, e.g. not HS => not HC
3   mitPreventsMit (HS)
4   guard "hSM_PERM & hACT_WELDING & hloc=atWeldSpot"
```

As a factor dependency, we identify HC requiresOcc (HS), *expressing the assumption that the factor* HS *must have occurred prior to the activation of* HC. *Using the* guard *directive, we specify the cause* χ, *the ground truth predicate for the activation of* HC. *Figure 8 shows the resulting risk structure for our case study as a risk graph.*

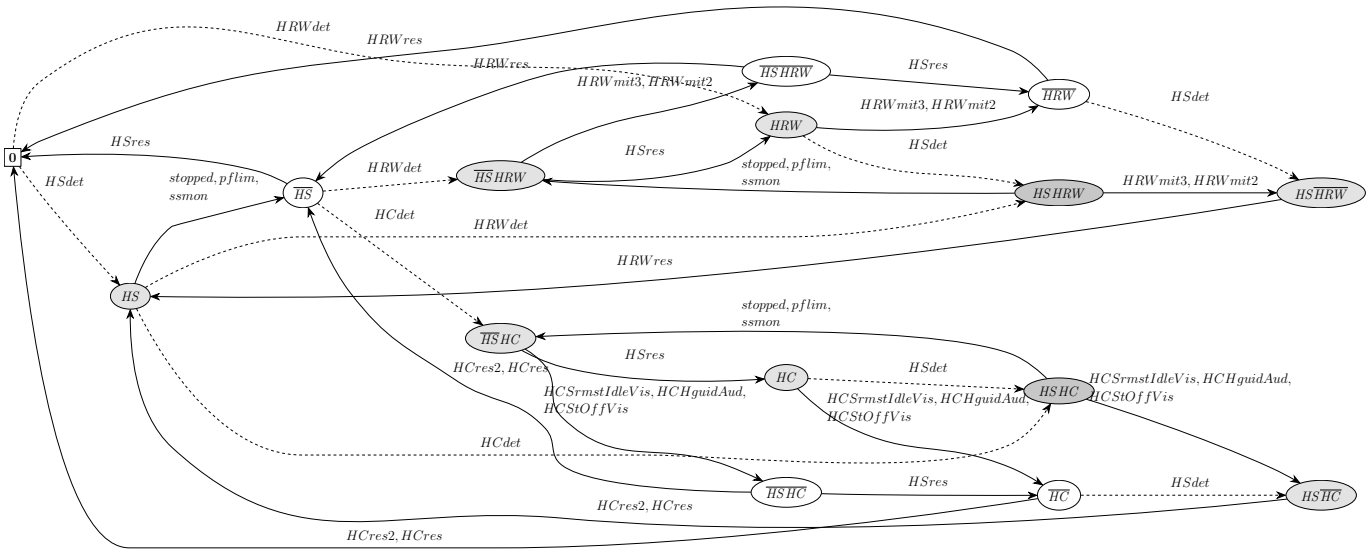


Figure 8: Risk graph for Example 2 from factors HC, HRW, and HS, with 15 risk states, serving as a specification to be refined by a synthesised controller. Dependencies "HC prevents HRW" and "HRW prevents HC" encode the assumption of one operator in the work cell. Detectors *HSdet* or *HRWdet* (dottet arcs) instantiate endangerments, and *stopped*, *pflim*, *ssmon*, and *HSres* exemplify mitigations and resumptions (solid arcs). The darker shaded a risk state, the more dangerous it is, qualitatively.

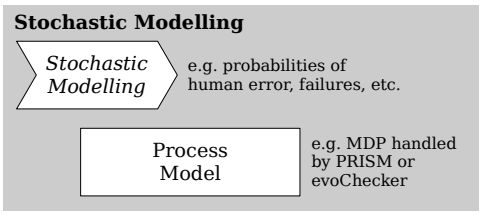


Figure 9: Overview of the work steps and artifacts of the stochastic modelling stage

The risk profile for the robot arm actions and the factors HC *and* HS *is encoded in the* YAP *model in an intuitive and compact manner.*

```
1                          guard  risk_HC  risk_HS;
2  // actor: robotArm
3  r_moveToTable:          ""     "5"      "9";
4  r_grabLeftWorkpiece:    ""     "0"      "3";
5  r_placeWorkpieceRight:  ""     "0"      "3";
6  r_moveToWelder:         ""     "9"      "5";
```

Based on the activity automaton in Figure 1b, we encode ∇r *in our* YAP *model with the two distance matrices* act *and* safmod.

```
1  distances act {
2    off :        0;
3    idle :       1 0;
4    exchWrkp:    3 2 0;
5    welding:     5 4 2 0;
6  }
7
8
```

```
9  distances safmod {
10   normal:    0;
11   hguid:     -2  0;
12   ssmon:     -1  1     0;
13   pflim :    -2  0     -1  0;
14   srmst:     -3 -1     -2 -1  0;
15   stopped:   -4 -2     -3 -2 -1  0;
16 }
```

4.3 Modelling Stochastic Adversarial Phenomena

In this stage (Figure 9), we integrate adversarial stochastic phenomena into the process model. Probabilistic choice in $\mathcal{M}$ can be used to model various phenomena, such as accidents, human error, and sensor failure.

Mishaps. In the refined factor model (Figure 7), a mishap $\underline{e}$ leading to phase $\underline{f}$ is always possible, assumed to happen more likely in the f-unsafe region than in the f-safe region. *Accident-prone physical actions* follow the two command patterns:

$$[\alpha]\ ok_p \wedge \gamma \wedge \neg\chi \longrightarrow \upsilon\ \&\ t' = sc$$

$$[\underline{e}_\alpha]\ ok_p \wedge \gamma \wedge \chi \wedge \neg\underline{f} \longrightarrow pr_{\underline{f}} : \underline{f}\ \&\ \upsilon_a\ \&\ t' = sc\ +\ (1 - pr_{\underline{f}}) : \upsilon_n\ \&\ t' = sc$$

with the probability $pr_{\underline{f}}$ of a mishap if the cause χ of the critical event has occurred, independent of whether or not f's activation (e^{f}) was detected. υ_a and υ_n can include specific updates if an accident occurs respectively if it does not. $pr_{\underline{f}}$ can be inferred from observations, experiments, or accident statistics. We keep using γ and υ for action-specific preconditions respectively updates.

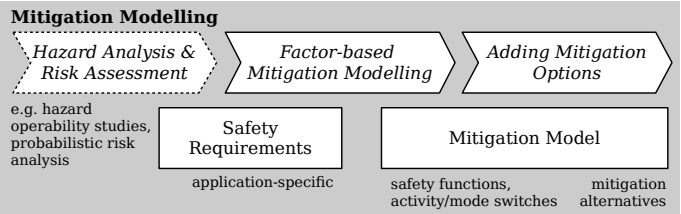


Figure 10: Overview of the work steps and artifacts of the mitigation modelling stage

Human Error. A human error model can be informed by hierarchical task analysis [Stanton, 2006]. We introduce a particular class of human errors into $\mathcal{M}$ using the pair of probabilistic commands

$$[\alpha_l]\ ok_{op} \wedge \neg\delta \wedge \gamma \longrightarrow (sp?1 : pr_{he})\colon \delta' = true \ +\ (sp?0 : 1 - pr_{he})\colon t' = sc$$
$$[\alpha_p]\ ok_{op} \wedge \delta \wedge \gamma \longrightarrow \upsilon \,\&\, \delta' = false \,\&\, t' = sc$$

with a predicate sp specifying when action α_p is safe or permitted to be performed, the probability pr_{he} of an operator to commit a specific error when this action is not safe or not permitted, and a deontic flag δ controlling whether the logical action α_l is to be reified into the physical action α_p with a potentially dangerous update υ.

Sensor Failure. Informed by a fault tree or failure mode effects analysis, one can consider sensor and actuator faults in a way similar to human error. To model *fault behaviour*, we employ the pattern

$$[\alpha]\ ok_p \wedge \gamma \longrightarrow (1 - pr_s)\colon \upsilon_{corr} \,\&\, t' = sc \ +\ pr_s\colon \upsilon_{fail} \,\&\, t' = sc$$

with a probability pr_s of a sensor failing to detect a specific event implied by γ, an update υ_{corr} modelling the correct behaviour of the sensor, and an update υ_{fail} modelling its failure behaviour.

Example 3. *We model the accident that, with a 20% chance, HC follows HC′ (i.e.,* HC *remains undetected because of a sensor fault) or HC (i.e., the safety controller is not reacting timely). For the encoding of* HC*,* YAP*'s input language supports the specification of the actions with the mishap HC as a bad outcome if HC is undetected or not mitigated timely, the probability of HC under these conditions, and the severity of the expected consequences from HC. Furthermore, we model the human error that, with a 10% chance, the operator enters the cell, knowing that the* robotArm *and the* spotWelder *are active. Finally, we specify as a sensor failure that the range finder as the detector of* e^{HC} *fails in 5% of the cases when the operator enters the cell.*

The command patterns explained in this section can be combined, offering many degrees of modelling freedom not further discussed here. For practical examples, see also Gleirscher [2020].

4.4 Modelling Mitigations

For mitigation modelling (Figure 10), we complete the factor LTSs introduced in Section 4.2.1 with mitigation actions. The ability of stochastic models, such as MDPs, to express nondeterminism supports the modelling of alternative *mitigation options*. To extend the controller design space, we can thus specify several such options for each factor. Differences in the quality of these options (e.g. expected nuisance and effort) can be quantified using reward structures.

The capabilities of actors in $\mathcal{P}$ determine the controllability of critical events. To restrict the controller design space, we allow three kinds of actions: *action filters* (i.e., safety modes, cf. Section 1), *activity changes* (e.g. change from welding to off), and *safety functions* (e.g. interacting with the operator through warnings). These mitigations are mirrored by corresponding resumptions. We continue with our discussion of how mitigation and resumption actions, according to the refined factor LTS in Figure 7, are translated into pGCL.

Let $\mathsf{f} \in F$ be the risk factor under consideration for the rest of this section. Given the ground truth predicate χ and the corresponding detector predicate ζ for f (Section 4.2.1), we assume to have identified a causal factor κ such that $\neg\kappa \Rightarrow \neg\chi \wedge \neg\zeta$ (i.e., an absent causal factor eliminates the cause) and $\zeta \Rightarrow \kappa$ (i.e., the causal factor is detectable). Factor dependencies, such as requiresOcc in Example 2, can be used to automatically derive part of ζ (cf. YAP, Gleirscher 2020).

Let a *mitigation option* $(a_t, sm_t, sf) \in S_a \times S_{sm} \times SF$ for f with a target activity a_t, a target safety mode sm_t, and sf picked from a set SF of safety functions. For each combination $(sm_{cur}, a_{cur}) \in S_{sm} \times S_a$ possible in a state $s \in S$, the controller provides a *safety mode switch*

$$[m_{sm}]\ ok_S \wedge f \wedge sm_{cur} \longrightarrow sm_{new} \,\&\, f'$$

and an *activity switch*

$$[m_a]\ ok_S \wedge f' \wedge a_{cur} \longrightarrow a_{new} \,\&\, f''$$

where sm_{new} and a_{new} are determined according to the scheme

$$x_{new} = \begin{cases} x_t, & \text{if } \partial r/(x_{cur} \to x_t) \geqslant 0 \\ x_{cur}, & \text{otherwise} \end{cases} . \tag{1}$$

We use the non-strict order $\geqslant$ because a switch to a desired target within the same risk level should be allowed. Then, the controller activates a *safety function sf* through the commands

$$[m_{sf}]\ ok_S \wedge f'' \wedge \kappa \wedge \neg sf \longrightarrow sf \,\&\, t' = p$$
$$[m_{sf}]\ ok_S \wedge f'' \wedge \kappa \wedge sf \longrightarrow t' = p \quad .$$

Reaching phase f'' constitutes the first set of logical controller actions. The performance of these actions is followed by an interaction with the process. If this interaction results in the elimination of κ, the controller finalises the mitigation stage with the command

$$[m]\ ok_S \wedge f'' \wedge \neg\kappa \longrightarrow \bar{f} .$$

The controller subsequently moves into the resumption stage, continuing with the *deactivation of the safety function* through

$$[r_{sf}]\ ok_S \wedge \bar{f} \wedge \neg\kappa \wedge sf \longrightarrow sf^{-1}$$

and the *resumption from the current to a more progressive safety mode* from any risk state $rs \in R(F)$ by

$$[r_{sm}]\ ok_S \wedge \bar{f} \wedge \neg\zeta \wedge \neg\kappa \wedge rs \wedge sm_{cur} \wedge \neg sf \longrightarrow sm_{new} \,\&\, \bar{f}' .$$

Beyond its basic enabling condition ($ok_S \wedge \bar{f}$), the controller checks whether both the cause of the critical event and, particularly, the causal factor subject of mitigation have been removed ($\neg\zeta \wedge \neg\kappa$).

Analogously, the *resumption of the current activity to a more productive activity* from any risk state $rs \in R(F)$ is accomplished with

$$[r_a]\ ok_S \wedge \bar{f}' \wedge \neg\zeta \wedge \neg\kappa \wedge rs \wedge a_{cur} \wedge \neg sf \longrightarrow a_{new} \,\&\, t' = p .$$

Given a *resumption option* $(a_t, sm_t) \in S_a \times S_{sm}$ and the set $am(rs) \subseteq F$ of factors active or mitigated in risk state rs, the reaction of the controller follows the scheme

$$(a_{new}, sm_{new}) = \arg_{(\mathrm{f}.a_t, \mathrm{f}.sm_t)} \max_{\mathrm{f} \in am(rs)} \left\{ \nabla r \mid \nabla r < [a_t, sm_t]^T \right\} . \tag{2}$$

This scheme determines the most permissive yet allowed combination of activity and safety mode to switch to among all activated or mitigated factors in rs, that is, the combination with the maximum acceptable risk as seen from (a_{cur}, sm_{cur}) and rs according to ∇r.

In order for the controller to be able to safely deal with a factor set F, we assume that each of the controller actions is idempotent. Note how phase indicators (e.g. $\bar{f}$) ensure that the controller is performing in the right context (e.g. if $rs \in [\![\bar{f}]\!]_R \subset R(F)$). The discussion of alleviations is out of scope of this synthesis approach.

Example 4. *Continuing with Example 2, we specify mitigation actions in our* YAP *model as shown below for the factor* HC *with three mitigation and two resumption options.*

```
1   detectedBy (.HCdet)
2   mitigatedBy (.HCHguidAud,.HCStOffVis,.HCSrmstIdleVis)
3   resumedBy (.HCres,.HCres2) ...;
4
5  mode HCdet desc "light barrier"
6    guard "hSM_PERM & hACT_WELDING & rngDet=close";
7  mode HCStOffVis desc "emergency stop / cobot+welder turned off / visual notif."
8    cf "hST_HOinSGA" // causal factor
9    update "(notif'=leaveArea)" // safety function
10   target (act=off, safmod=stopped); // target activity /safety mode
11 mode HCSrmstIdleVis desc "safety–rated mon. stop / cobot+welder idle / visual notif."
12   cf "hST_HOinSGA" update "(notif'=leaveArea)" target (act=idle, safmod=srmst);
13 mode HCStOffAud desc "emergency stop / cobot+welder turned off / audio–vis. notif."
14   cf "hST_HOinSGA" update "(notif'=leaveArea)" target (act=off, safmod=stopped);
15 mode HCHguidAud desc "hand–guided welding / moderate vis. notif."
16   cf "hST_HOinSGA" update "(notif'=leaveArea)" target (safmod=hguid);
17 mode HCres desc "exchange workpiece and start over"
18   cf "hST_HOinSGA"
19   update "(notif'=ok)" // deactivate safety function
20   target (act=exchWrkp, safmod=normal);
21 mode HCres2 desc "continue welding if workpiece still unfinished"
22   cf "hST_HOinSGA" update "(notif'=ok)" target (act=welding, safmod=normal);
```

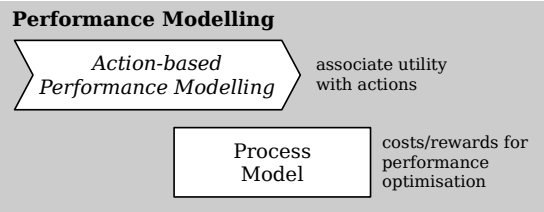


Figure 11: Work step and artifact for performance modelling

The directive detectedBy *defines the sensor predicate* ζ*, stating that "the human operator is in the safeguarded area" (*hST_HOinSGA*). The factor attribute* mitigatedBy *associates* HC *with three mitigation options, and the attribute* resumedBy *with two resumption options. For example, in the action* HCStOffVis*, (i)* update *models a safety function, issuing a notification to the operator to leave the safeguarded area, and (ii)* target *switches the manufacturing cell to the activity* off *and to the safety mode* stopped,[a] *all triggered by the range finder (*rngDet=close*). The* guard *and* detectedBy *attributes are translated into a pair of predicates for* $\mathcal{M}$*,* RCE_HC *(*χ*) describing world states, and* CE_HC *(*ζ*) signifying states monitored by the range finder.*

```
1 // HC:monitor "(H)uman (C)lose to active spot welder and cobot working"
2 formula CE_HC = (hSM_PERM & hACT_WELDING & rngDet=close) & (HSp=act | HSp=mit1 | HSp=mit2 | HSp=mit | HSp=res) & (HRWp!=act);
3 formula RCE_HC = (hSM_PERM & hACT_WELDING & hloc=atWeldSpot) & (HRWp!=act);
```

[a] In a design variant discussed in Gleirscher and Calinescu [2020], we allow mitigations to synchronise with the robotArm and spotWelder on an event stop.

4.5 Modelling Performance

In analogy to the rewards for mitigation actions, in this stage (Figure 11), we quantify performance (e.g. effort, productivity) for all non-controller actions in the process. As a result, optimal policy synthesis from $\mathcal{M}$ is based on several reward structures quantifying the performance of both the safety controller and the process.

For controller performance, we distinguish mitigation and resumption options by manually estimated quantities such as *disruption* of the manufacturing process, *nuisance* of the controller to the operator, and resources (e.g. *effort*, *time*) consumed by the controller. We formalise these quantities as action rewards r^{q}_{action} with $q \in \{\text{disr,nuis,eff,time}\}$. As shown in Example 5, rewards can depend on parameters other than state variables.

Analogously, concerning process performance, we associate with each process action a *productivity* measure depending on the safety mode, using an action reward structure r^{prod}_{action}.

Example 5. *In a* YAP *model, mitigation and resumption options can be associated with action rewards:*

```
1 mode HCStOffVis ...
2   disruption=9 nuisance="alarmIntensity1 * 5"
3   effort=5.5; // wear/tear intensive, maintenance effort, energy
4 mode HCSrmstIdleVis ...
5   disruption=7.5 nuisance="alarmIntensity1 * 6" effort=6.5;
6 mode HCStOffAud ...
7   disruption=10 nuisance="alarmIntensity1 * 9" effort=5;
```

Note how nuisance depends on the parameter alarmIntensity1 *modelling the loudness or brightness of an alarm sound or lamp that can be varied in the search for an optimal controller. Rewards for process actions can be specified in a concise manner in a* YAP *model.*

```
1                          guard_prod      prod;
2 // actor: robotArm
3 r_moveToTable:           ""              "h";
4 r_grabLeftWorkpiece:     ""              "m";
5 r_placeWorkpieceRight:   ""              "h";
6 r_moveToWelder:          ""              "h";
```

5 VERIFIED SYNTHESIS OF SAFETY CONTROLLERS

In this stage, we integrate the risk and mitigation models with the process model, resulting in a risk-informed reward-enhanced stochastic model that includes the *controller design space* in the process decision space (Figure 12). The action sets for the cobot,

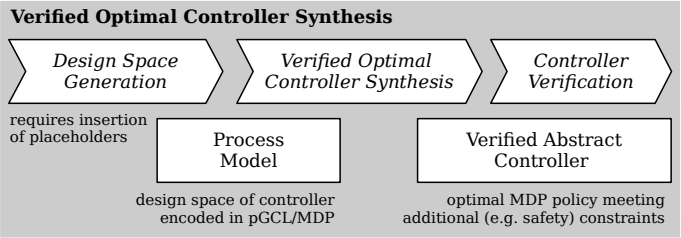


Figure 12: Overview of the work steps and artifacts of the controller synthesis stage

the operator, and the controller are now combined. With this integrated model, we perform a constrained policy synthesis to select an optimal yet abstract safety controller from the design space. We use constraints to encode the *safety requirements* and *optimisation queries* to facilitate this selection. For this to work, we express safety requirements as PCTL properties and verify them using a stochastic model checker. We accomplish controller synthesis in **two settings**.

The MDP Setting. We perform optimal policy synthesis from an MDP (using PRISM) where the design space is encoded as non-deterministic choice (e.g. among mitigation options). This approach has already been discussed in Gleirscher and Calinescu [2020].

The parametric DTMC (pDTMC) Setting. We perform an evolutionary search (using EVOCHECKER) of a set of DTMCs. This set defines the design space by fixing the parameters of a pDTMC. This pDTMC can be obtained from the original MDP by replacing non-deterministic choice with random choice and by introducing the corresponding parameters. In both cases, optimal policy synthesis produces a DTMC containing an abstract discrete-event safety controller. The pDTMC-based approach avoids the split into two verification stages as previously required in Gleirscher and Calinescu [2020].

The synthesis follows a two-staged search through the controller design space: The first stage focuses on the generation of the guarded commands according to Section 4.4. The gradient ∇r (Section 4.2.2) resolves the calculation of risk-minimal control updates for these commands. Reward structures are generated for risk and performance quantification (Section 4.5) in the second stage. Then, a stochastic model checker performs verified policy synthesis for $\mathcal{M}$. In Gleirscher and Calinescu [2020], we use $\mathcal{M}$ as an MDP for policy synthesis with PRISM [Kwiatkowska et al., 2011] and extract a controller from the resulting DTMC representing the policy.

Here, we enhance this approach. The flexible CSP-style concurrency of the pGCL modules is replaced by a more restrictive alternation (Section 4.1.1) of the process $\mathcal{P}$ and the safety controller, thereby resembling a closed-loop control scheme. The avoidance of interleaving and synchronous events for modelling real-time phenomena (e.g. sensor faults) results in an MDP that does not contain states where process actors and the controller compete in non-deterministic choices. The only choices left are actor-internal choices including the choice to idle and pass the token. As a consequence, we obtain *fairness* for the controller and a simplification of $\mathcal{M}$. Additionally, we interpret $\mathcal{M}$ as a pDTMC rather than an MDP. Choice in the safety controller among action options is explicitly controlled through decision parameters.

This scheme results in the removal of non-deterministic choice from the controller and a randomisation of residual choice in $\mathcal{P}$. We further improve the way how accidents can happen in $\mathcal{M}$. For fine-granular risk and performance quantification, we allow additional design space parameters (e.g. alarm intensity) to be used. We use EVOCHECKER [Gerasimou et al., 2018] for the search of optimal controllers in the space of DTMCs defined by these parameters. The modelling, analysis, and pre-processing required for the approach in Gleirscher and Calinescu [2020] and its enhancement described here are supported by the YAP tool [Gleirscher, 2020].

5.1 Design Space and Reward Structure Generation

The design space is created by instantiating the command patterns of the generic factor LTS in Figure 7. These patterns are used by Algorithm 1 and implemented in YAP. The function COMPCMD composes guard and update expressions and integrates these into controller commands compliant with the specifications in Section 4.4. The functions GRADUPDM and GRADUPDR implement Equation (1) respectively Equation (2) for controlling the updates of safety modes and activities. AS_{f} refers to the set of mitigation options for factor f. $\mathcal{T}_{S_a}$ refers to the set of activity tuples, that is, combinations of activities the actors can be involved at a particular point in time. The other command patterns in Section 4.4 are generated analogously. We omit the corresponding generation functions here. The listing in Example 6 shows a fragment of the design space.

Example 6. *pGCL fragment generated for the factor* HC*:*

```
1 // Endangerments (monitor, extension point: sensor and monitoring errors)
2 [si_HCact] OK_S & !(HCp=act | HCp=mit1 | HCp=mit2 | HCp=mis) & wact=idle & ract=exchWrkp & CE_HC –> (HCp'=act); ...
```

Algorithm 1 Controller design space generation

function GENSAFETYMODEMITIGATIONS(F, S_{sm}))
  for all $sm \in S_{sm}$, f $\in F$, $ms \in AS_{\mathsf{f}}$ **do**
    $m_{sm} \leftarrow$ COMPCMD($f \wedge sm$, GRADUPDM(sm,ms)) ▷ *create mit.*
    $A_{\mathcal{P}} \leftarrow A_{\mathcal{P}} \cup \{m_{sm}\}$ ▷ *add mitigation command*

function GENACTIVITYMITIGATIONS(F, $\mathcal{T}_{S_a}$)
  for all $a \in \mathcal{T}_{S_a}$, f $\in F$, $ms \in AS_{\mathsf{f}}$ **do**
    $m_a \leftarrow$ COMPCMD($f' \wedge a$, GRADUPDM(a,ms)) ▷ *create command*
    $A_{\mathcal{P}} \leftarrow A_{\mathcal{P}} \cup \{m_a\}$ ▷ *add mitigation command*
...

function GENMODERESUMPTIONS(F, S_{sm})
  for all $rs \in R(F)$, $sm \in S_{sm}$, f $\in am(rs)$, $ms \in AS_{\mathsf{f}}$ **do**
    $(\zeta, \kappa, sf) \leftarrow$ obtained from factor model for f
    $r_a \leftarrow$ COMPCMD($\bar{f} \wedge \neg\zeta \wedge \neg\kappa \wedge rs \wedge sm \wedge \neg sf$, GRADUPDR($rs$,f,$ms$)) ▷ *create safety mode resumption*
    $A_{\mathcal{P}} \leftarrow A_{\mathcal{P}} \cup \{r_a\}$ ▷ *add command*

```
// Change of safety modes
[si_HCSrmstIdleVissafmod] OK_S & HCp=act & dpHCmit=HCHCSrmstIdleVis & safmod=normal –> (safmod'=srmst)&(HCp'=mit1); ...
// Frame switches
[s_HChalt] OK_S & HCp=mit1 & dpHCmit=HCHCSrmstIdleVis & wact=welding & ract=exchWrkp –> (wact'=idle)&(HCp'=mit2); ...
// Execution of safety functions
[si_HCSrmstIdleVisfun] OK_S & HCp=mit2 & dpHCmit=HCHCSrmstIdleVis & hST_HOinSGA & !(notif=leaveArea)
–> (notif'=leaveArea)&(token'=mod(token+1,ag))&(turn'=token+1); ...
// For entering the mitigated phase
[si_HCmit] OK_S & HCp=mit2 & dpHCmit=HCHCSrmstIdleVis & !(hST_HOinSGA) –> (HCp'=mit); ...
// Switching off safety functions
[si_HCres2fun] OK_S & HCp=mit & dpHCres=HCHCres2 & !(hST_HOinSGA) & (notif=leaveArea | notif=leaveArea | notif=leaveArea) –> (notif'=ok); ...
// Meta–policy for resuming to a less restrictive safety mode
[si_HCres2safmod] OK_S & HCp=mit & dpHCres=HCHCres2 & !CE_HC & !hST_HOinSGA & (HSp=mit|HSp=res) & (HRWp=mit|HRWp=res) & safmod=normal & notif=ok
    –> (safmod'=pflim)&(HCp'=res); ...
// Resuming actor's activities
[s_HCresume2] OK_S & HCp=res & dpHCres=HCHCres2 & !CE_HC & !(hST_HOinSGA) & (HSp=mit|HSp=res) & (HRWp=mit|HRWp=res) & wact=welding & ract=
    exchWrkp –> (wact'=welding) & (ract'=welding) & (token'=mod(token+1,ag)) & (turn'=token+1) & (HCp'=inact); ...
```

Example 7 shows a fragment of the reward structures generated from the YAP model described in the Sections 4.2.1 and 4.5.

Example 7. *The listing below shows two reward structure fragments, one for risk from an active* HC *and one for nuisance.*

```
// Risk of HC–mishap when performing nominal action ...
rewards "risk_HC"
[rw_leaveWelder] !CYCLEEND & (RCE_HC | CE_HC) & safmod=pflim : 0.6 * 5 *
    9.0;
[h_exitPlant] !CYCLEEND & (RCE_HC | CE_HC) & safmod=hguid : 0.4 * 2 * 9.0;
[r_moveToTable] !CYCLEEND & (RCE_HC | CE_HC) & safmod=hguid : 0.4 * 5 *
    9.0;
...
endrewards

...
// Nuisance (e.g. to the human operator; per mitigation option)
rewards "nuisance"
[si_HCStOffVisfun] REWGUARD_HC : alarmIntensity1 * 5 / 1;
[si_HRWmit2fun] REWGUARD_HRW : alarmIntensity2 * 4 / 1;
[ si_pflimfun ] REWGUARD_HS : alarmIntensity1 * 8 / 1;
...
endrewards
```

5.2 *Verified Optimal Synthesis*

The pGCL action system consists of the process (i.e., cobot, welding machine, and operator) and the safety controller generated according to Section 5.1. The policy space $\Pi_{\mathcal{M}}$ includes the controller design space. This action system is expanded into an MDP by a probabilistic model checker such as PRISM or it is used as a pDTMC by EVOCHECKER relying on DTMC model checking. Choice in $\Pi_{\mathcal{M}}$ stems from commands (e.g. mitigations, resumptions) simultaneously enabled in a state $s \in S$, yielding multiple policies for s and from commands enabled in multiple states, giving rise to a policy for each ordering in which these commands can be chosen.

Controller solutions selected from the design space are subjected to two kinds of requirements. The first are optimisation objectives, such as minimal energy consumption. The second are probabilistic and reward-based constraints for safety (i.e., unlikely reachability of bad states, e.g. accidents below probability threshold; accumulated risk below reward threshold) and response (e.g.

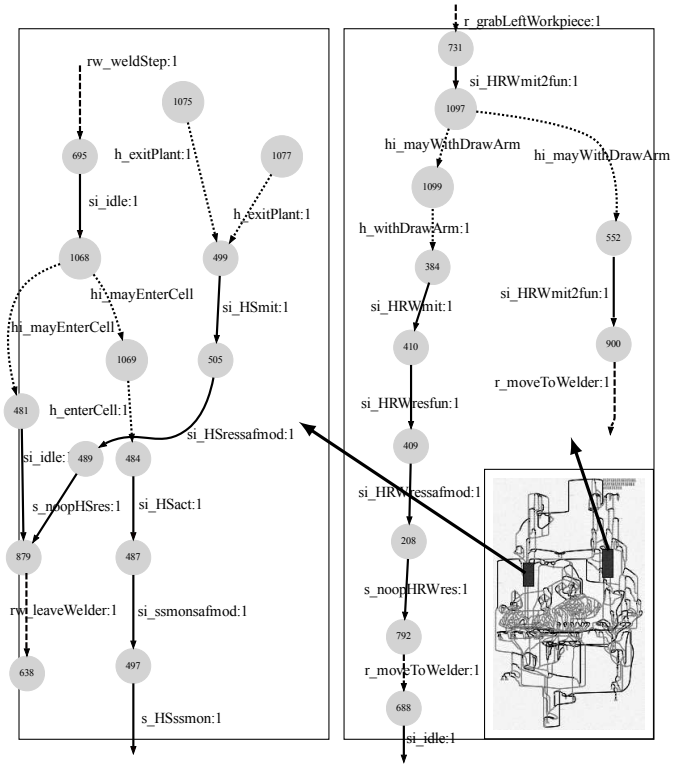


Figure 13: Visualisation of an optimal policy $\pi^\star$ (lower right box) refining the risk graph for our case study in Figure 8. The controller decisions (solid arcs), the spot welder and cobot actions (dashed arcs), probabilistic choices of the operator (dotted arcs). The two zoomed fragments highlight that mitigation of HS or HRW can deal with a variety of environments, for example, adversarial human decisions.

timely controller response exceeds probability threshold). Both kinds of requirements are expressed in reward-enhanced, quantitative PCTL (Section 3.2).

Optimisation Objectives. An optimal policy $\pi^\star \in \Pi_{\mathcal{M}}$, including the controller decisions, can be selected based on, for example, minimum nuisance or maximum productivity. For that, $\mathcal{M}$ includes action rewards to quantify controller and process performance (Section 4.5), risk reduction *pot*ential (in the **MDP setting**), and risk based on factors, modes, and activities as explained in Section 4.2.

Constraints. Constraints are of the form $\pi^\star \models \phi_{wf} \wedge \phi_c$. ϕ_{wf} captures well-formedness,[11] including properties for the verification of, for example, hazard occurrence and freedom from pre-*final* deadlocks, and properties for the falsification of, for example, that final states must not be initial states. Hazard occurrence fosters our focus on adversarial environments. ϕ_{wf} can help one to simplify model debugging, decrease model size, remove deadlocking states, and reduce vacuity of verification results.

ϕ_c specifies safety-carrying correctness and can include progress, safety, liveness, and reliability properties. For example, we want to verify *reach-avoid* properties of type $\mathbf{A}\,\mathbf{G}\,\mathbf{F}\,\gamma \wedge \mathbf{A}\,\mathbf{G}\,\neg\chi$; or that the probability of failure on demand of the controller or the probability of a mishap from any hazard is below a threshold. With $\mathbf{A}\,\mathbf{G}(f \rightarrow \mathbf{P}_{>p}[\mathbf{F}\,\bar{f}]) \wedge \mathbf{A}\,\mathbf{G}(\bar{f} \rightarrow \mathbf{P}_{>p}[\mathbf{F}\,\not{f}])$, we constrain the search for solutions in the design space to controllers whose mitigation paths from critical events are complete with at least probability p. In the **MDP setting** (as explained in Section 5), our model allows the evaluation of freedom from accidents in $\mathcal{M}$ with

$$\mathbf{P}_{\neg A} \equiv f_{s\in\Xi}\, \mathbf{P}^{s}_{\min=?}[\neg \underline{F}\ \mathbf{W}\ \not{F}] \tag{3}$$

where $f \in \{\min, \mathrm{mean}, \max\}$. For the non-accident F-unsafe region Ξ (Section 3.3), Equation (3) requires the controller to minimise the probability of mishaps until the F-safe region (i.e., $S\backslash(\Xi \cup \underline{F})$) is reached. $\mathbf{P}_{\neg A}$ aggregates min, the arithmetic mean μ, and max probabilities over Ξ. In the **pDTMC setting**, we use the plain quantification operator $\mathbf{P}$. Table 3 contains further examples of properties in ϕ_{wf} and ϕ_c to be verified or falsified of $\mathcal{M}$.

[11] Well-formedness refers to the class of properties (see, e.g. Table 3) to be checked to establish basic model validity prior to more interesting correctness properties related to the application under consideration.

Table 3: Objectives to be queried over $\Pi_{\mathcal{M}}$ and properties to be checked of every $\pi \in \Pi_{\mathcal{M}}$

Property[†]	**Description**
Optimisation objectives	
$\mathbf{R}^{\text{eff}}_{\max=?}[\mathbf{C}]$	Assuming an adversarial environment, select π that maximally utilises the safety controller.
$\mathbf{R}^{\text{nuis}}_{\min=?}[\mathbf{C}^{<T}]$	Select π that minimises nuisance up to time T.
Objectives with reward-based constraints	
$\mathbf{R}^{\text{prod}}_{\max=?}[\mathbf{C}] \wedge \mathbf{R}^{\text{sev}}_{\leqslant s}[\mathbf{C}] \wedge \mathbf{R}^{\text{risk}}_{\leqslant r}[\mathbf{C}]$	Select controller that maximises productivity constrained by risk level r and expected severity s.
$\mathbf{R}^{\text{prod}}_{\max=?}[\mathbf{C}] \wedge \mathbf{R}^{\text{sev}}_{\leqslant s}[\mathbf{C}]$	Select controller that maximises productivity constrained by exposure p to severe injuries.
Well-formedness constraints in ϕ_{wf}	
$\mathbf{E}\,\mathbf{F}\,\mathit{final}$	$\mathcal{P}$ can finish the production cycle.
$\mathbf{E}\,\mathbf{F}(f \wedge \neg \mathit{final})$	f can occur during a production cycle.
$\mathbf{E}\,\mathbf{F}(\mathit{deadlock} \wedge \neg \mathit{final})$	$\mathcal{P}$ can deadlock early. (f)
$\mathbf{A}\,\mathbf{F}\,f$	f is inevitable. (f)
$\forall s \in S : \neg \mathit{final} \vee \neg \mathit{init}$	Some initial states are also final states. (f)
Correctness constraints in ϕ_c	
$\mathbf{A}\,\mathbf{G}(\zeta \rightarrow \mathbf{A}\,\mathbf{F}^{<t} f)$	The controller detects χ for f within t steps.
$\mathbf{A}\,\mathbf{G}(\zeta \wedge \not\chi \rightarrow \mathbf{A}(\zeta\,\mathbf{U} f))$	The controller timely responds to the belief of χ.[‡]
$\mathbf{A}\,\mathbf{G}(f \rightarrow \mathbf{P}_{>p}[\mathbf{F}\,\bar{f}]) \wedge \mathbf{A}\,\mathbf{G}(\bar{f} \rightarrow \mathbf{P}_{>p}[\mathbf{F}\not\chi])$	The controller lively handles hazard f.
$\mathbf{A}\,\mathbf{G}(f \rightarrow \mathbf{P}_{>p}[\mathbf{F}\,\mathit{final}])$	The controller resumes $\mathcal{P}$ so it can finish its cycle after f has occurred.
$\mathbf{P}_{>p}[\mathbf{G}\,\neg \underline{F}]$	Mishap freedom is more likely than p.
$\mathbf{S}_{<p}\,\underline{F}$	The steady-state probability of any $\underline{f}$ is below p.

[†] *deadlock*: state with no commands enabled, *final*: end of manufacturing cycle, *init*: initial state of a manufacturing cycle, $\underline{F}$: mishap state, p: probability bound, f: to be falsified, prod: productivity, eff: controller effectiveness, sev: severity, risk: risk level. [‡]We adopt the universality pattern after Dwyer et al. [1999] using **U** instead of **W** because of the required response; note that we have to use the sensor predicate ζ rather than the ground truth predicate χ.

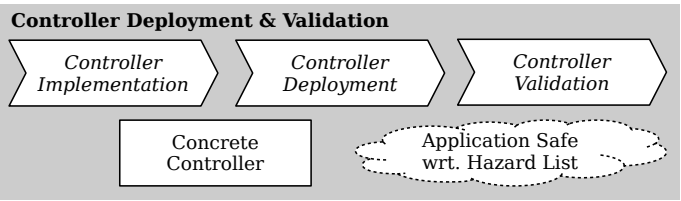


Figure 14: Overview of the work steps and artifacts of the controller deployment and validation stage

Verified Synthesis. Based on the action rewards from the Sections 4.2.1 to 4.2.2, the model checker investigates all choice resolutions and parameter valuations for $\mathcal{M}$ that fulfil well-formedness and further PCTL constraints. The checker then identifies policies that fulfil the given constraints and are (Pareto-)optimal with respect to the optimisation objectives. The existence of an optimal strategy depends on the existence of a strategy in $\Pi_{\mathcal{M}}$ that fulfils the PCTL constraints. Note that, by Definition 2, all policies considered for $\mathcal{M}$ are of the same size but may vary in their distribution of choice among the involved actors. The overall result of this step is a verified and optimal abstract controller extracted from the selected policy $\pi^\star$. An example of such a policy is visualised in Figure 13.

6 Controller Deployment and Validation

Given the execution semantics of a target platform (e.g. the digital twin framework), the selected controller can now be translated into a concrete executable form (Figure 14) to be deployed, validated, demonstrated, and eventually used on this platform.

6.1 Controller Implementation

The abstract controller, c, is part of the calculated policy $\pi^\star$, a DTMC with state space $S_c \subseteq S$. S_c is a result of combining the risk state space, generated by YAP from the factor set F, with the process state space. According to Definition 2, the transition relation of $\pi^\star$ is a list of $(\mathit{state}, \mathit{action}, \mathit{probability}, \mathit{state})$-tuples. The controller as a deterministic part of $\pi^\star$ is a set of transitions

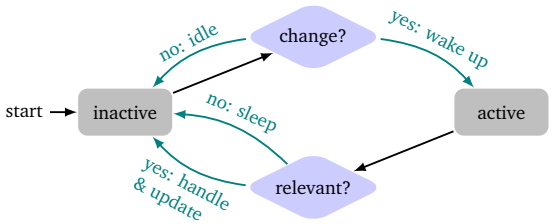


Figure 15: Execution of the controller in the digital twin framework

Algorithm 2 Execution of the safety controller

```
1: procedure SafetyController(in State m, out State c)
2:     (p, r, r′) ← init                    ▷ initialise controller and risk state
3:     while true do
4:         if (p, r) ≠ (m, r′) then          ▷ wake up on event/change
5:             (p, r) ← (m, r′)              ▷ capture monitored process state
6:             if p ∈ S_c then               ▷ is the event relevant?
7:                 c ← HandleEvent(p, r)     ▷ issue control command
8:                 r′ ← UpdateRiskState(p, r) ▷ stay active if changed
```

$\delta_c \subseteq S_c \times S_c$ that, at the concrete level, again comprises guarded commands of the form

$$\overbrace{[\text{controller action}]}^{\text{event}}\ \overbrace{\text{process state} \wedge \text{risk state}}^{\text{guard}} \rightarrow \overbrace{\text{mode \& activity switch, safety function}}^{\text{update}}\ .$$

We use the transition matrix obtained from the model checker (e.g. PRISM) to translate controller actions in $\pi^\star$ into concrete actions. Following the four-relation structure of controller models in Parnas and Madey [1995], this step involves (i) the translation of the abstract states into guard conditions based on a mapping of concrete process states into abstract states, and (ii) the translation of the abstract updates into low-level procedures generating control inputs to the process. Figure 15 shows the activation scheme of the concrete controller when deployed on an execution platform. Algorithm 2 describes the corresponding discrete-event SafetyController. According to Figure 5, through Line 4, the controller is aware of each atomic update of any of the monitored variables.

Example 8. *In our case study, the check of whether $p \in S_c$ in Line 6 of Algorithm 2 is implemented as a switch statement iterating over all relevant combinations of events known from $\mathcal{M}$. The function* HandleEvent *is responsible for issuing control inputs, such as switching into a power & force limitation mode and notifying the operator to leave the work cell if* HC *is activated. The function* UpdateRiskState *is responsible for managing and remembering the risk state internal to the controller (e.g. state in the mitigation of* HC*). In the supplemental material[a] for this work, we provide modelling and code examples[b] and a video[c] of the controller in action.*

[a] See https://github.com/douthwja01/CSI-artifacts/.
[b] See the `hrc2` example of the YAP package (https://github.com/ytzemih/yap).
[c] See https://youtu.be/cm-XkZ_aitQ.

Expected Overhead. The detection and handling overhead is the time elapsed in every cycle of the **while** loop in Algorithm 2. Let $d\colon \alpha_{\mathcal{P}} \rightarrow \mathbb{R}$ be the processing time required for an action, for example, the calculation of the detection of HC in e^{HC}. If implemented as part of a sequential cell controller, Line 6 requires a time slot of length $\Sigma_{\mathsf{f}\in F} d(e^{\mathsf{f}})$ in each control cycle. If Line 6 is monitored simultaneously in dedicated safety controller hardware, the slowest detection rate for F is $1/\max_{\mathsf{f}\in F} d(e^{\mathsf{f}})$. The overhead for handling f in Line 7 can be estimated from Figure 7 and may range from $d^{\mathsf{f}}_{\min} = d(m^{\mathsf{f}}_{sm}) + d(m^{\mathsf{f}}_{a}) + d(m^{\mathsf{f}}) + d(r^{\mathsf{f}}_{sm}) + d(r^{\mathsf{f}}_{a})$ to $d^{\mathsf{f}}_{\max} = d^{\mathsf{f}}_{\min} + x \cdot d(m^{\mathsf{f}}_{sf}) + d(r^{\mathsf{f}}_{sf})$ with a repetition factor $x \in \mathbb{N}$. The overhead of the implementation in Algorithm 2 can be obtained by recording timed event traces from an execution platform (e.g. the DTF). In order for the controller to interact with such a platform, the YAP model is extended by an interface specification in addition to the model fragments discussed in the previous sections. This interface is discussed in more detail in the following section.

6.2 *Controller Deployment*

Section 3.4 introduces the notion of actors within a collaborative manufacturing setting involving a cobot and an operator (see Figure 16). We are able to reconstruct this setting in the form of a digital twin using our digital twin framework. The DTF is a toolchain developed in C# and visualised in Unity3D that provides the kinematic, communication, and data infrastructure

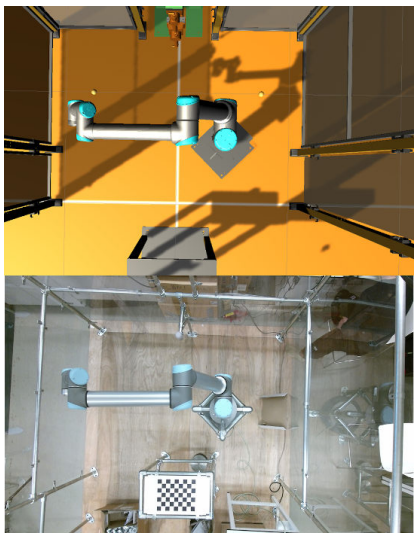


Figure 16: The DTF applied to our case study. Requests issued to the robot within the DTF are enacted by the physical twin in the real-world system. The robot, LIDAR, and light barrier provide feedback to the environment.

necessary to deploy digital twins on real world systems [12]. Using the DTF APIs for the MATLAB® robotics toolbox and the Robotic Operating System framework[13], the safety controller's actions are exchanged with the physical platform in real-time where a response is demonstrated.

We represent the scenario with as an aggregation of actions and decisions made by each actor. Actors may then be identified as (i) *digital twins*—actors with a distinct collection of models, state-machines and behaviours that emulate the capabilities of the physical system—and (ii) *abstract* actors without physical embodiment that may provide a service, communicate with or control other actors. The term *environment* will be used to describe this complete set of actors.

Process Representation. Let $\mathcal{A}$ be the set of all actors. Examining the process from a network perspective allows us to model an actor $n \in \mathcal{A}$ as a communication node, similar to the robotic operating system framework. Following terminology in Broy [2010], the actor n is able to communicate with other nodes through an interface $I_n = (S_n^j, P_n^k)$. Here, $S_n = [S_n^1, S_n^2, ..., S_n^j]$ and $P_n = [P_n^1, P_n^2, ..., P_n^k]$ denote the subscription (or input) and publication (or output) channels of actor n respectively. This decentralised structure allows us to represent a *process controller* $p \in \mathcal{A}$ as abstract actor with known feedback and command channels S_p and P_p respectively. The process controller is modelled as a state machine that responds to feedback from actor n and issues a requested action on P_p.

Each sensor present in the process is similarly introduced as a digital twin actor $s \in \mathcal{A}$. Here, data originating from the sensing capabilities of s are broadcast to assigned channels P_s in order to inform the network of changes to the physical environment. Manipulators and machinery are modelled as digital twins of the physical equipment, with behaviours informed by the actuation constraints of the physical system. In response to commands issued by p, the robot digital twin is able to interact with the work piece, operator or tendered machine. This communication is then forwarded and expressed by the physical twin as seen in Figure 16.

Safety Controller. A safety controller $c \in \mathcal{A}$ is introduced to the DTF as a singleton node declared as an abstract actor. c communicates on fixed channels $P_c = [P_c^1, P_c^2, ..., P_c^l]$ with a family of process actors $\{p_i\}_{i \in 1..k} \subset \mathcal{A}$. The nominal procedure of p is governed by a local hierarchical state machine responding to process updates on S_p. To allow the safety controller to intervene in this nominal procedure, the interpreter behaviour S_c is implemented as a parent state machine as seen in Figure 17. This secondary state machine allows then a safety controller to enact changes to n's safety mode(s) in response to requests made over channels in P_c.

Example 9. *As part of the case study presented in Section 2, an environment has been developed to evaluate the synthesised safety controller. This environment, shown in Figure 16, presents us with a distributed system composed of multiple digital twins, with each physical actor (e.g. robot, spot welder, operator) and component (e.g. sensors) in the real-world process assigned its own individual twin.*

[12] See https://github.com/douthwja01/CSI-artifacts/JSS/.
[13] See https://www.ros.org.

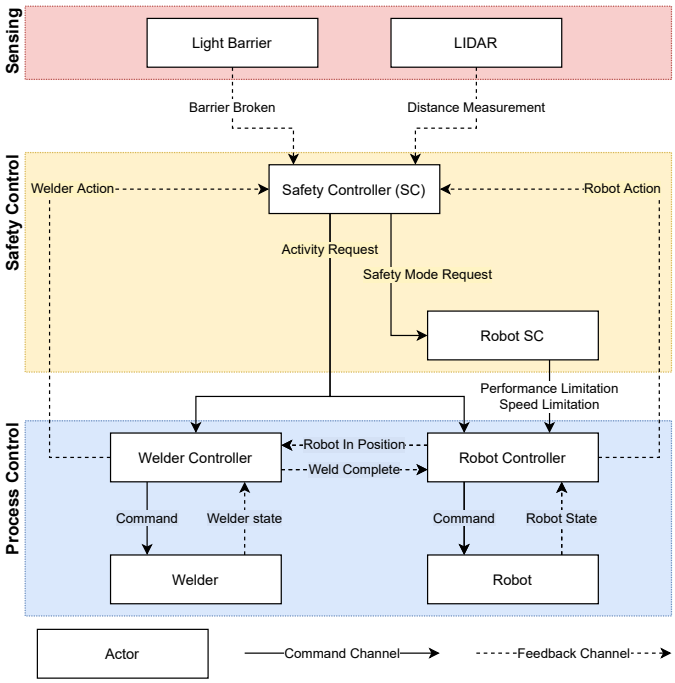


Figure 17: Communication structure in the DTF. In accordance with our case study, a safety mode is requested by the safety controller (middle block) to apply to the welder and robot digital twins (bottom block). In response, the nominal process for both systems is overridden. For example, in response to a detected hazard the controller may issue the safety action "safety stop", preventing the process continuing.

The process is a work-piece exchange and welding task, overseen by a process controller that issues and responds to tasks assigned to each actor. The safety controller observes feedback from the process controller, a workbench light barrier and a LIDAR positioned within the cell. Figure 17 then describes the communication of safety mode and activity change requests to all actors within the environment on occurrence of a critical event, following the event handling logic in Figure 15.

6.3 Controller Validation by Testing

The validation of a controller implementation c in the DTF can be done by use-case-based testing of scenarios generated from the process model $\mathcal{M}$. Given a use case U, a misuse case MU, an initial state $s_0 \in S$, a set $\mathcal{T}_{\mathcal{V}}(U)$ of *traces in vicinity* $\mathcal{V}$ *of* U, we say that c, as deployed in the DTF, *conforms* with $\mathcal{M}$ with respect to U, written $c \approx_U R'$, if and only if $\mathcal{M} \models R \wedge \forall t \in \mathcal{T}_{\mathcal{V}}(U) : t \models R'$. Moreover, we say that c complies with $\mathcal{M}$ with respect to MU if and only if $\mathcal{M} \not\models R \wedge \neg\exists t \in \mathcal{T}_{\mathcal{V}}(MU) : t \models R'$. Finally, we say that the controller implementation is $(U, MU, \mathcal{V}, R)$-*valid* if and only if $c \approx_U R'$ and the closest MU we can find, with $c \not\approx_{MU} R'$ and no overlap with U regarding $\mathcal{V}$, suggests implausible inputs to the DTF.

For controller testing, we translate requirements R given as PCTL properties (Table 3) into corresponding linear and metric temporal logic properties in R'. Particularly,

$$\mathbf{G}(\zeta \wedge \not f \rightarrow \zeta\, \mathbf{U}_{[0,d]} f) \tag{4}$$

requires c to detect the critical event, that is, to get active whenever the sensor predicate ζ holds true within d time units. Furthermore,

$$\big((\mathbf{F}\, final) \rightarrow \big(\mathbf{G}(f \rightarrow \mathbf{F}\bar{f}) \wedge \mathbf{G}(\bar{f} \rightarrow \mathbf{F}\not f)\big)\big) \wedge \mathbf{G}\, \neg \underline{f} \tag{5}$$

states for a completed process ($\mathbf{F}\, final$) that whenever the controller detects f (Table 1) it moves the DTF to a state where f is mitigated ($f \rightarrow \mathbf{F}\bar{f}$) and from there, given the environment (e.g. operator) eventually reacts, it returns the process to a state where f is inactive and operation is resumed as far as possible ($\bar{f} \rightarrow \mathbf{F}\not f$). Even if the environment does not react, an accident never occurs ($\mathbf{G}\, \neg \underline{f}$).

We describe U and MU as (generated) sequences of inputs to the DTF and the vicinity $\mathcal{V}$ as a set of (randomly generated) configurations of the DTF (e.g. considering the operator being far, near, and close to the spot welder, and entering the cell at different times). We statistically explore $\mathcal{T}_{\mathcal{V}}(U)$ by playing in a sample to the DTF, record the event trace t, and verify $t \models R'$ using the metric temporal logic run-time checker PYMTL [Vazquez-Chanlatte, 2019].

Data Extraction. To allow communications between actors to be recorded and analysed, an additional abstract actor is introduced as a network *snooper*. This actor subscribes to updates from a family of actors $\{p_i\}_{i\in 1..n} \subset \mathcal{A}$ and exposes their communications externally as a series of discrete event streams. As the scenario is executed, the generated streams are marked with a reference time and origin of each data packet. Communications between the safety controller, process controller and other process members are then recovered for forensic analysis and evaluation of the safety controller.

The safety controller may be exported as a C# extension to our DTF and deployed within a digital twin of the case study in Section 2. At the point of analysis, each data stream is composed of a series of data packets with an assigned timestamp and entity identifier. Each event stream from time $t = 0 : n$ is exported from the database following the scenario execution. This allows the data to be presented as a ledger of all monitored channels, indicating where inter-actor communication occurred, actor states have changed and instances where the controller guards have been triggered.

7 EVALUATION

In this section, we pose three research questions about safety, utility, and scalability, describe the evaluation methodology for each of these questions, and discuss the results of our evaluation.

7.1 Research Questions and Methodology

Based on the questions raised in Section 1, we investigate key aspects of our approach by asking three research questions (RQs).

RQ1 (Safety) What is the likelihood of accident-free operation under the control of a synthesised safety controller?

RQ2 (Utility) Does the safety controller reduce the number of hard stops of the robot due to hazards, compared to a basic controller that switches off the system whenever the operator intrudes the work cell?

RQ3 (Scalability) How well can the proposed approach deal with multiple hazards and mitigation and resumption options?

Methodology for RQ1 (Safety). We answer **RQ1** in two stages, first based on our modelling approach (**RQ1a**) and, second, supported by our deployment and validation approach (**RQ1b**).

Methodology for RQ1a. We evaluate freedom from accidents according to Equation (3), leading to probability triples comprising min, the arithmetic mean μ, and max.

In the **MDP setting**, we use PRISM to synthesise policies from $\mathcal{M}$ according to the three optimisation queries

$$\mathbf{R}^{\text{pot}}_{\text{max}=?}[\mathbf{C}] \wedge \mathbf{P}_{\text{max}=?}[\mathbf{F}\, \mathit{final}_{\text{t}}], \tag{a}$$

$$\mathbf{R}^{\text{prod}}_{\text{max}=?}[\mathbf{C}] \wedge \mathbf{P}_{\text{max}=?}[\mathbf{F}\, \mathit{final}_{\text{t}}], \text{ and} \tag{b}$$

$$\mathbf{R}^{\text{eff}}_{\text{max}=?}[\mathbf{C}] \wedge \mathbf{R}^{\text{nuis}}_{\text{max}=?}[\mathbf{C}]. \tag{c}$$

where $\mathit{final}_{\text{t}} = \{s \in S \mid s \in \mathit{final} \wedge \text{all tasks finished} \wedge s \notin \underline{F}\}$. In the spirit of negative testing, Equation (a) aims at maximising the use of the safety controller (i.e., approximating worst-case behaviour of the operator and other actors) while maximising the probability of finishing two tasks, that is, finishing a workpiece and carrying through cell maintenance. This query does not take into account further opmitisation parameters defined for mitigations and resumptions. As opposed to that, Equation (b) fosters the maximisation of *prod*uctivity, any combination of decisions allowing the finalisation of tasks is preferred, hence, transitions leading to accidents or the use of the controller are equally neglected. While Equation (c) also forces the environment to trigger the controller, these policies represent the best controller usage in terms of *nuis*ance and *eff*ort. Because of restrictions in the use of $\mathbf{R}_{\text{min}}$ for MDPs, we maximise costs interpreting positive values as negative (e.g. the higher the nuisance the better). We then investigate the Pareto front of optimal policies synthesised from the Equations (a) to (c). For policies with less than 1000 states, we inspect the corresponding policy graphs (e.g. whether there is a path from *initial* to *final* or whether paths from unsafe states reachable from *initial* avoid deadlocks). Finally, we evaluate accident freedom according to Equation (3), except that we use $\mathbf{P}_{=?}$ for DTMCs instead of $\mathbf{P}_{\text{min}=?}$.[14]

In the **pDTMC setting**, we use EVOCHECKER to synthesise policies for $\mathcal{M}$ according to the objective

$$\mathbf{P}_{\leqslant 0}[\mathbf{F}(\mathit{deadlock} \wedge \neg \mathit{final})] \tag{6}$$

$$\wedge\, \mathbf{R}^{\text{prod}}_{\text{max}}[\mathbf{C}^{\leqslant T}] / \left(\mathbf{R}^{\text{disr}}_{\text{max}}[\mathbf{C}^{\leqslant T}] + \mathbf{R}^{\text{eff}}_{\text{max}}[\mathbf{C}^{\leqslant T}]\right) \tag{Productivity}$$

$$\wedge\, \mathbf{R}^{\text{nuis}}_{\text{min}}[\mathbf{C}^{\leqslant T}] \tag{Nuisance}$$

$$\wedge\, \Sigma_{\text{f}\in F}\left(s_{\text{f}} \cdot \mathbf{R}^{\text{risk}_{\text{f}}}_{\text{min}}[\mathbf{C}^{\leqslant T}]\right) \tag{Risk}$$

[14] To keep manual workload under control, if the model checker (here, PRISM) lists several adversaries, we apply the experiment only to the first listed.

over a time period $[0, T]$ and with factor-specific scaling factors s_f. This objective contains a probabilistic constraint ruling out early deadlocks, and three optimisation queries for maximising productivity, minimising nuisance, and minimising overall risk from a factor set F. We then assess the resulting Pareto front, extract a solution from this front, and use YAP to refine that solution into a concrete controller.

Methodology for RQ1b. In Section 5.2, we described the synthesis of a correct abstract controller and, in Section 6.1, its translation into a concrete controller c interfacing with the DTF explained in Section 6.2. Then, how do we assure the transfer of the results on freedom from accidents of the abstract controller (**RQ1a**) to the concrete one in the DTF? For this, we follow the framework in Section 6.3 and deploy and test c in the DTF for compliance with $\mathcal{M}$ in certain use cases and assess its behaviour in misuse cases.

The DTF is used in a simulation capacity to validate the proposed safety controller. The simulation provides for (i) automated testing and (ii) a safe environment for evaluation, whilst representing a faithful one-to-one construction of the work cell. Physical, digital, or mixed environments expose the same interface for integration with the DTF; the same controller can be used in either context without modification. As such, integration with the physical work cell is not evaluated, in part due to limited access to the lab replica.[15]

We follow one use case (U) and one misuse case (MU) to exercise the risk factors HC, HS, and HRW described in Table 1:

U: During operation, the operator reaches across the workbench and walks to the spot welder.

MU: During operation, an operator reaches across the workbench while another operator walks to the spot welder.

We perform tests for the use case as outlined in Figure 19g. The operator first heads to the workbench, breaking the light barrier (*HRW*), before heading inside the cell (*HS*) close to the spot welder (*HC*). The observed stream of events during each test can be split for the independent validation of each risk factor. Considering the stream as a whole provides for the validation of hazard mitigation and the absence of impact on further actions.

The operator waits at given positions while the cobot and spot welder proceed through their scheduled activities. Variations in the time spent by the operator in different states result in different interleavings of the operator and robot, and such variations in turn might be ground for the activation of hazards in the system. The configuration for each test is a vector of 4 values corresponding to 4 wait operations of the operator: entering the workbench, at the workbench, entering the work cell, and at the spot welder. To bound the time taken by each test, the values are picked such that the operator completes its actions in less than 20s. This is ample time for the robot to perform its own actions, and the operator is able to disrupt the different tasks in the process either by walking to the spot welder or reaching across the workbench. We rely on the Dirichlet-Rescale algorithm [Griffin et al., 2020] for generating vectors such that the values of the vector sum to a given total, and the distribution of vectors in the constrained space is uniform.

We rely on situation-based coverage criteria, proposed by Alexander et al. [2015], to assess the performance of our test campaign, and whether we have achieved a satisfactory level of testing. We define our situations in terms of either the actions of the spot welder, the actions of the robot, or the position of the arm, when the operator is reaching at the workbench or entering the cell in accordance with the use case. Full coverage is achieved when all such actions or positions have been observed under both interference factors. We also ensure all valid states for all risk factors have been encountered (as defined in Figure 7). This ensures causal factors have been encountered, mitigated, and the system could resume operation.

Methodology for RQ2 (Utility). We argue that the synthesised safety controller is better than a state-of-the-art controller that only has a stop mode and performs no automatic resumption. For this argument, we compare the whole range of controllers from the design space with those controllers that always perform a safety stop when detecting a critical event. Based on that, we informally assess the increase in productivity and fluency of collaboration, and the decrease of mean-time to finishing a process cycle. This argument underpins our contribution to the problem statement in Section 1.

Methodology for RQ3 (Scalability). We prepare and analyse multiple increments of the risk model, each adding one critical event, mitigation options, and constraints to the model. We record the resulting model sizes and analysis times.

7.2 Results

In the following, we present the results of our evaluation separately for each research question.

7.2.1 Results for RQ1a: Safety in the Model

We consider as inputs a risk model and a process model of the work cell, with a single initial state of these models where all actors are in the activity off and no critical event has occurred. The risk model is given in YAP's input language and the behavioural

[15] At the time of writing, COVID-19 prevents access to the Sheffield Robotics lab and physical components of the case study.

Table 4: Results of the experiment for **RQ1a** (accident-free operation) and **RQ3** (scalability) in the **MDP setting**

Risk Model†				**MDP**†			**(a) max-ASC**†			**(b) max-prod**			**(c) opt-ASC**		
F	mr/c	$\lvert R(F)\rvert$	t_Y [ms]	$\mathbf{P}_{\neg A}$ $\lfloor\mu\rfloor$	Ξ	sta/tra	$\mathbf{P}_{\neg A}$ $\lfloor\mu\rfloor$	Ξ	t_P [s]	$\mathbf{P}_{\neg A}$ $\lfloor\mu\rfloor$	Ξ	t_P [s]	$\mathbf{P}_{\neg A}$ $\lfloor\mu\rfloor$	Ξ	t_P [s]
HC	5/0	3	40	[.9,.9,.9]	14	322/1031	[1,1,1]	3	.02	[1,1,1]	1	.02	[1,1,1]	6	.15
+HS	9/2	5	52	[.92,.96,.98]	256	930/3483	[.07,.66,1]	11	.77	[0,.88,1]	8	.82	[.95,.98,1]	18	.9
+WS	11/3	8	44	[.93,.97,1]	288	1088/3865	[0,.29,1]	17	2.1	[0,.8,1]	5	2	[1,1,1]	24	1.5
+HRW	13/7	16	65	[.93,.97,1]	981	7675/33322	[1,1,1]	17	9.7	[1,1,1]	11	9.4	[1,1,1]	15	13.3
+HW	15/8	36	76	[.93,.97,1]	2296	21281/98694	[1,1,1]	15	42.9	[0,.71,1]	7	41.4	[1,1,1]	15	46.6
+RT	15/9	50	87	[.93,.97,1]	2864	21965/100133	[1,1,1]	13	48.2	[1,1,1]	9	46.4	[1,1,1]	15	53.8
+RC	15/15	122	162	[.93,.99,1]	12079	21670/102263	[0,.94,1]	35	38	[0,.72,1]	22	36.6	[1,1,1]	36	51.1

† F... critical event set; mr/c... number of mitigations+resumptions/constraints; $\lvert R(F)\rvert$... cardinality of the relevant subset of $R(F)$ defined in Section 3.3; t_Y... YAP's processing time; $\mathbf{P}_{\neg A}$... probability of conditional freedom from accidents; Ξ... set of F-unsafe states; sta/tra... number of states/transitions of the MDP (sta equals the size of the policies); Equations (a) to (c)... optimisation queries; t_P... PRISM's processing time

Table 5: Comparison of minimum, average (μ), and maximum freedom from accidents according to Equation (3) of a process with a controller and a process without one.

Process ...	min	μ	max
... with a safety controller	63%	97%	100%
... without a safety controller	63%	87%	100%

model is given in PRISM's flavour of pGCL, instrumented with YAP template placeholders. We consider the two settings explained in Section 5. In the **MDP setting**, we use YAP 0.5.1 for generating the design space (Section 5.1) embedded into a process model without alternating execution semantics and with controller synthesis directly from an MDP using PRISM 4.5. In the improved **pDTMC setting**, we use YAP 0.7.1 for generating the design space embedded into a process model with the alternating execution semantics described in Section 4.1.1, an improved accident model, and with controller synthesis from a pDTMC using EVOCHECKER and PRISM 4.5. For **RQ1a**, we used GNU/Linux 5.4.19 and 5.8.0 (x86, 64bit), and an Intel® Core i7-8665U CPU with up to 8 Threads of up to 4.8 GHz, and 16 GiB RAM.

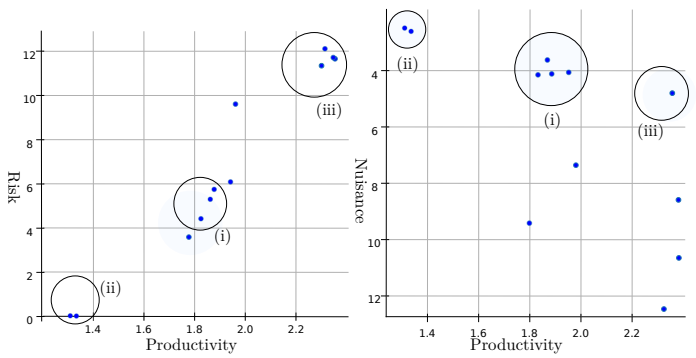
The results for the original **MDP setting** are displayed in Table 4, which shows the data collected based on seven increasingly more complex risk models. The $(\min, \text{mean}, \max)$ probability triples are denoted by $\lfloor\mu\rfloor$. The result $\lfloor\mu\rfloor = [1,1,1]$ for a policy denotes 100% conditional freedom from accidents. This desirable result is most often achieved with Equation (c) due to the fact that simultaneity of decisions of the environment and the safety controller in the same state is avoided by focusing on rewards only specified for controller actions. Such rewards model the fact that a controller is usually much faster than an operator. Equations (a) and (b) show poorer freedom from accidents because *prod*uctivity rewards given to the environment compete with rewards given to the safety controller to exploit its risk reduction *pot*ential.

In the improved **pDTMC setting**, we focus on a single risk model with the three factors HS, HC, and HRW. We calculate $\lfloor\mu\rfloor$ for Equation (3) for a process without a safety controller and one with a controller. Table 5 shows an increase in average freedom from accidents from 87% to around 97% when using a safety controller. Starting from any state in the process shows that there are very safe states with a probability of accidents of down to 0% (max column) and rather dangerous states with a probability of up to 37% (min column). Overall, going from 13% down 3% average accident probability (across the three risk factors) means that the controller in this particular process leads to a reduction of accidents by 77%.

The 3D Pareto front in Figure 18 indicates that the lower the risk of a controller (Conjunct Risk in Formula 6) the lower the productivity (Conjunct Productivity) due to the interventions of the controller (cf. Figure 18a). Figures 18a and 18b show (i) four controllers in the design space with low nuisance (Conjunct Nuisance) and medium productivity and risk, and (ii) two minimal-risk controllers with low nuisance but low productivity. (iii) Controllers that minimise nuisance and maximise productivity tend to do this at the cost of high risk. If the minimal-risk controllers under (ii) are not satisfactory, a more detailed trade-off regarding the controllers in (i) or a repetition of the analysis with different design parameters or an altered process and controller model will have to be made.

7.2.2 Results for RQ1b: Safety in the Digital Twin

We performed 100 tests for the use case shown in Figure 19g. This proved sufficient to achieve full coverage of (i) the mitigation states for all considered risk factors, and (ii) operator interference types across spot welder and cobot states. All recorded traces t verify $t \models R'$, that is, all traces satisfy the selected properties translated from Table 3. Risk factors were correctly detected (with $d = 0.25ms$ in Property 4) and mitigated by the synthesised safety controller in all observed situations in the DTF. All results for **RQ1b** were produced under the Windows 10 Home Edition operating system, build 19042.985, on an Intel® Core i5-8250U



(a) Projection: risk and productivity (b) Projection: nuisance and productivity

Figure 18: Two projections of a 3D Pareto front for the optimisation objective in Formula (6) for the time period up to $T = 30$

4-core CPU at 1.8 GHz, with 8 GiB RAM. The whole test suite, including validation of the traces, took less than two hours to complete. No time compression was used, the DTF ran simulations in real-time.

Figure 19 illustrates the output of the DTF for the use case (Figure 19g). Each image captures the state of the system as the operator begins a wait operation. The use case highlights the activation of all monitored risk factors (see Table 1) with the operator and cobot making concurrent use of the workbench, and the operator in the vicinity of the spot welder as an operation is about to start.

MU relies on two operators simultaneously following the behaviours outlined by U. The tested model assumes a single operator interacts with the system, such that an operator must leave the workbench before entering the work cell. As such, the safety controller correctly mitigates the risk due to both cobot and operator reaching for the workbench. However, as the second operator enters the cell the related risk factors are neither identified as active nor mitigated. Figure 20 highlights the situation, with the second operator in close proximity to an active spot welder.

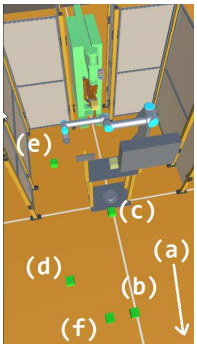
The work cell is safe considering a single operator following the use case, that is entering the work cell or the workbench during operation. However, the risk and process models would need to be revised to account for multiple operators interacting with the system (e.g. by removing factor dependencies between HRW and HS, see Section 4.2.1). The case with two operators would normally be picked up in an extended risk assessment following Table 1, but it was not relevant to this particular process. This case is also not within scope for the purposes of keeping this example simpler.

7.2.3 Results for RQ2: Utility

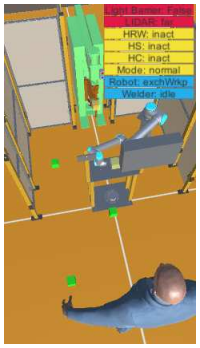
In the **MDP approach**, Table 6 highlights several aspects. First, the fact that none of the controller configurations achieves the productivity level (64.19) of the process without a controller. Such a process reaches a theoretical productivity maximum assuming no safety requirements, which is for obvious reasons not desirable. Perhaps unsurprisingly, safety comes at the cost of productivity. However, more importantly, among all configurations, the one switching to "stopped", whenever e^{HS}, e^{HC}, or e^{HRW} occurs, is the one performing the poorest (37.06, highlighted in gray). As expected, controllers implementing the pflim and ssmon safety modes as part of their policy are among the highest performing variants (51.42). With the best safety controller, the process achieves about 40% more productivity than with the worst controller and is only 25% less productive than the maximum productivity achievable in the process.

Moreover, the most risky configurations (68, highlighted in gray) are around 77% less risky than not using a safety controller. However, the most difficult trade-off is to be made between very low risk controllers ($\leqslant 9$) and moderate risk controllers ($\geqslant 49$) where the gain in productivity is comparatively little. Finally, with this model instance, the addition of the factor HC does neither influence risk nor productivity.

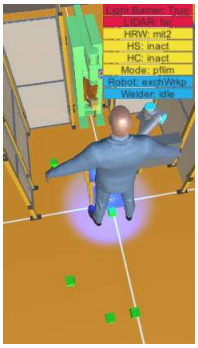
Regarding the expected overhead (Section 6.1), we used the C# profiling tool integrated in the development environment for the DTF. We measured the execution time of control cycles in the DTF across 1000 invocations of the controller. Our observations cover both nominal activity, and the occurrence of risk factors. The longest observed execution time $d_{\max}^{\mathsf{HS,HC,HRW}}$, its high watermark, is around $40ms$. The high watermark occurs upon activation of a risk factor, and accounts for mode switch, and the required mitigation actions. In the absence of a mitigation requirement, the average execution time $d^{\mathsf{HS,HC,HRW}}$ is less than $1ms$. Execution times are expected to be lower in the physical twin. Profiling was performed on the same environment as for **RQ1b**, that is a 1.8 GHz i5 CPU with 8 GiB of RAM running Windows 10.



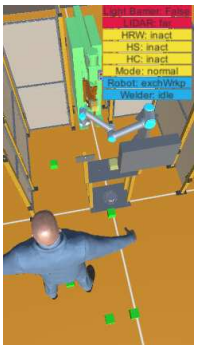
(a) System configuration for testing, with the operator's path highlighted by numbered waypoints.



(b) Initial state. The operator waits in front the workbench while the cobot moves to retrieve a work piece.

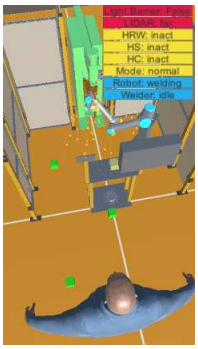


(c) The controller issues mitigations as the operator and cobot simultaneously reach for the workbench (*HRW*).

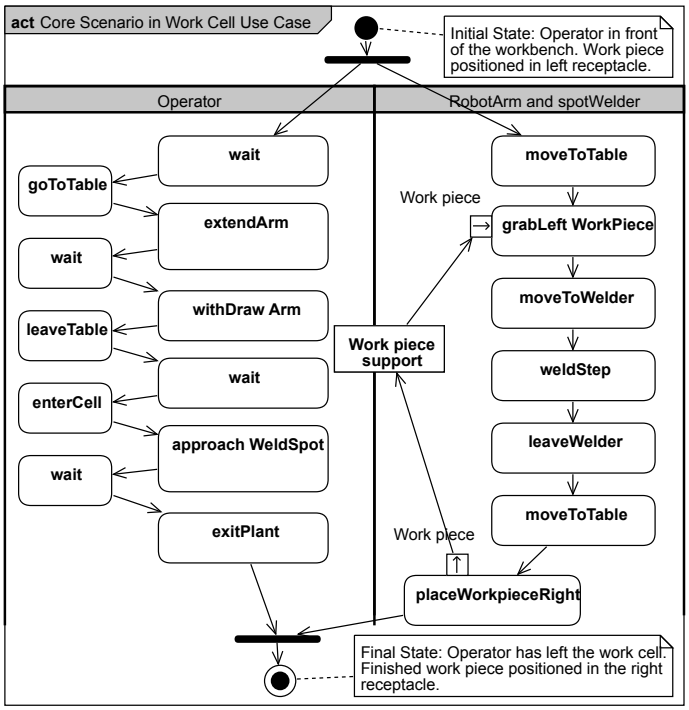


(d) The operator leaves the workbench while the cobot moves to the spot welder. The risk factor has been removed.

(e) The controller issues a stop in response to the operator approaching the spot welder during operation (*HS* and *HC*).



(f) Welding resumes after the operator leaves the work cell. The notification to leave the cell is deactivated.



(g) Use case describing operator, cobot and spot welder (inter)actions. Each image on the left further highlights, in the top right corner, the state of the different actors of the system as the use case progresses, that is, the sensors (light barrier and LIDAR), the safety controller (*HRW*, *HS*, *HC*, and the safety *mode*), and the activities (of the cobot and the spot welder). A visual notification instructing the operator to leave the handover table or welding area are issued in the form of blue and red lights respectively.

Figure 19: Illustration of different states (19a)-(19f) of the use case described in (19g).

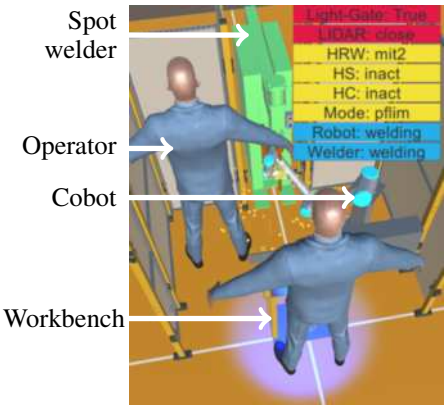


Figure 20: Result for the misuse case (*MU*): the second operator is not notified to leave the work cell, and the spot welder remains active while he is in close proximity.

Table 6: Comparison of a process without a safety controller with processes including controllers using different safety mode and task switching policies. Safety functions such as warnings are not taken into account in this comparison.

Process	Configuration†			Productivity‡	Risk‡
	HC	HRW	HS	[units]	[units]
Using controller	smrst / idle	stop	stop	37.06	0
	stop / off	stop	stop	37.06	0
	hguid / –	stop	stop	37.06	0
	smrst / idle	pflim	stop	44.75	9
	stop / off	pflim	stop	44.75	9
	hguid / –	pflim	stop	44.75	9
	smrst / idle	stop	pflim	46.26	49
	stop / off	stop	pflim	46.26	49
	hguid / –	stop	pflim	46.26	49
	smrst / idle	stop	ssmon	48.15	59
	stop / off	stop	ssmon	48.15	59
	hguid / –	stop	ssmon	48.15	59
	smrst / idle	pflim	pflim	49.90	58
	stop / off	pflim	pflim	49.90	58
	hguid / –	pflim	pflim	49.90	58
	smrst / idle	pflim	ssmon	51.42	68
	stop / off	pflim	ssmon	51.42	68
	hguid / –	pflim	ssmon	51.42	68
No controller	–	–	–	64.19	291

†Configuration of mitigation options (safety mode and activity switches), *smrst*: safety-rated monitored stop, *stop*: power shutdown with user-initiated work cell reset, *hguid*: hand-guided operation, *pflim*: power & force limitation, *ssmon*: speed & separation monitoring, *idle*: currently not performing work steps, *off*: work cell not in operation.

‡According to $\mathbf{R}^{prod}_{\max=?}[\mathbf{C}^{\leqslant T}]$ and Property (Risk) with $T = 50$.

7.2.4 *Results for RQ3: Scalability*

We answer this question using six increments applied to a basic risk model (Table 4). We only consider the **MDP setting** because applying model increments in the **pDTMC setting** will not provide additional insights into the scalability aspect of our method.

For demonstration of YAP's capabilities, the incident RT and the accident RC are included in the risk model without handler commands. However, these factors add further constraints on $R(F)$ to be dealt with by the safety controller. Hence, *mr* stays at 15 actions and *c* rises to 15 constraints. In model 7 (last line of Table 4), $R(F)$ (122 risk states) and the Ξ-region of S (12079 states) differ by two orders of magnitude. Risk states offer a higher level of abstraction for risk assessment. The derivation of properties that focus on relevant regions of the MDP state space from a risk structure can ease the state explosion problem in explicit model checking. For example, the constraints HRW prevents HC and HC prevents HRW express that the combined occurrence of HC and HRW is considered infeasible or irrelevant by the safety engineer. Hence, checking properties of the corresponding region of the MDP state space can be abandoned.

7.3 *Threats to Validity*

Internal Validity. *Too strong environmental assumptions reduce the scope of the safety guarantee.* To limit the need for game-theoretic reasoning about $\mathcal{M}$, we reduce non-deterministic choice for the environment (i.e., operator, cobot, spot welder). The more deterministic such choice, the closer is the gap between the policy space $\Pi_{\mathcal{M}}$ and the controller design space. Any decisions left to the environment will make a verified policy $\pi^{\star}$ safe relative to $\pi^{\star}$'s environmental decisions. These decisions form the assumption of the controller's safety guarantee.

Occupational health and safety assumes trained operators not to act maliciously, suggesting "friendly environments" with realistic human errors. To increase the priority of the controller in the **MDP-based approach**, we express realistic worst-case assumptions, for example, by minimising factor- and activity-based risk and maximising the risk reduction potential. The **pDTMC-based approach** weakens such assumptions by the fact that all outcomes of environmental decisions remain in the policy in form of probabilistic choices. Hence, the controller extracted from $\pi^{\star}$ will contain an optimal choice for all states reachable from environmental decisions. Obtaining appropriate trade-offs between productivity and safety is challenging. Hence, in the future

we may switch to game-theoretic approaches which may allow for a more nuanced view of the environment and hence allow for even greater productivity.

Incorrect application of tools. evoChecker relaxes constraints if the constrained solution space is empty, leading to solution under relaxed constraints. Hence, the design space needs to be redefined in order for evoChecker to select from a non-empty set of solutions obeying safety-related hard constraints.

Insufficient validation. Confidence in the testing-based validation depends on the range and complexity of (mis)use cases and the parameters used for randomisation. The use case for **RQ1b** is generic enough to cover a wide range of work cell scenarios and the misuse case clarifies the environmental assumptions of the use case.

External Validity. *Reality gap through inadequate quantification.* The lab replica of the work cell matches closely to the one in the company. This allowed us to develop an accurate digital twin and an MDP compliant with the actual work cell that incorporates all relevant parameters (particularly, probabilities). We crafted the MDP such that assessment and optimisation based on probabilities and rewards rely more on qualitative relations between the parameters rather than accurate numbers. It is more important and easier to conservatively assess whether an activity or safety mode is more or less dangerous than another and it is less important and more difficult to provide accurate numbers. The resulting gap between optimality in the model and optimality in the digital twin should hence not alter the demonstrated confidence in the controller. More accurate numbers can be obtained from observations, and after a model update, a repeated synthesis can improve productivity of the safety controller.

Incorrect sensing assumptions. In our case study, the safety controller relies on the detection of an operator (e.g. extremities, body) and a robot (e.g. arm, effector) entering a location, the cell state (e.g. grabber occupied, workbench support filled), and the work piece location (e.g. in grabber, in support). For $\mathcal{M}$, we assume the tracking system (i.e., range finder and light barrier in the industrial setting, MS Kinect[16] in the lab replica) to map the location of the operator and robot to the areas "in front of the workbench", "on the workbench", "in the workcell", and "close to the spot welder" rather than to a fine-grained occupancy grid. In Figure 2b, the range finder signals "at welding spot" if the closest detected object is nearer than the close range, and "in cell" if the closest object is nearer than the wide range. Tracking extensions, not discussed here, could include object silhouettes and minimum distances, operator intent, or joint velocities and forces.

Incorrect real-time behaviour. pGCL, as we used it, requires care with the modelling of real-time behaviour, particularly, when actions from several concurrent modules are enabled. To model real-time controller behaviour, we synchronise operator actions with sensor events and, in the original **MDP setting**, force the priority of controller reactions in $\pi^\star$ by maximising the risk reduction potential (cf. *pot* in Table 3). While synchronisation restricts global variable use, increasing $\mathcal{M}$'s state space, we found it to be the best solution in the multi-module **MDP setting**. The alternating execution scheme used in **pDTMC setting**, however, avoids the use of rewards to implement priorities and prevent actors from competing in an unnatural way.

7.4 Discussion

Tool Restrictions and Model Debugging. State rewards allow a natural modelling of, e.g. risk exposure. However, in Prism 4.5, one needs to use action rewards for multi-objective queries of MDPs. Risk gradients help to overcome a minor restriction in Prism's definition of action rewards.[17] Alternatively, we could have introduced extra states at the cost of increasing $\mathcal{M}$'s state space, undesirable for synthesis. Rewards require the elimination of non-zero end components (i.e., deadlocks or components with cycles that allow infinite paths and, hence, infinite reward accumulation). Prism provides useful facilities to identify such components, however, their elimination is non-trivial and laborious in larger models and can require intricate model revisions. We strongly discretise model parameters such as location to further reduce the state space and keep the model small. We use probabilistic choice in synchronous updates only in one of the participating commands to simplify debugging. We avoid global variables to support synchronisation with complex updates

Misuse Cases for Controller Testing. Misuse cases help in exploiting deficiencies of a control concept through counterexamples. The latter can be generated from the process model and exercised as negative test cases to aid in debugging the controller. We explored this idea in Gleirscher [2011], applying a Prolog-based Golog interpreter that constructs counterexamples from backward depth- and breadth-first search from a given accident state. Prism uses forward breadth-first search from an initial state and stops when reaching an accident state. In both cases, explicit state exploration is used with the usual problem of state space explosion. In any case, an environment model is needed to express accident states. In this work, we prefer a model checker because the encoding of the stochastic process in pGCL appeared to be easier than in a stochastic situation calculus and because of more flexibility of expressing properties to be verified in PCTL.

[16] See https://en.wikipedia.org/wiki/Kinect.
[17] Currently, rewards cannot be associated with particular updates, that is, with incoming transitions rather than only states.

8 RELATED WORK

Our work builds upon a set of well known theoretic principles on which research on controller design and synthesis [Kress-Gazit et al., 2018] for collaborative robots has been carried out. In the following, we compare our results with other results.

Risk-informed Controller Design and Verification. Askarpour et al. [2016] discuss controller assurance of a cobot work cell based on a discrete-event formalisation in the linear-time temporal language TRIO. Actions are specified as *pre/inv/post*-triples for contract-based reasoning with the SAT solver Zot. Violations of the safety invariant *inv* lead to pausing the cell. Fine-grained severity quantification [Vicentini et al., 2020] allows a controller to trigger safety measures on exceeding of certain risk thresholds. Additionally, Askarpour et al. [2019] present a model of erroneous or non-deterministic operator behaviour to enable designers to refine controller models until erroneous behaviours are mitigated. This approach aims at risk-informed prototyping and exhaustive exploitation of all possible executions to identify constraint violations and remove hazardous situations. Risk thresholds correspond to a refined variant of Property (4). Whilst not the focus of this work, our approach also allows for the quantification of severity in detector predicates (ζ). Their severity model [Askarpour et al., 2019, Vicentini et al., 2020] inspires future risk factor models. Instead of a priority parameter, which reduces state variables, we use guards to implement action orderings. Our approach is more flexible because it can deal with multiple mitigation options offering more variety in safety responses. Beyond model consistency checks and the search of counterexamples for model repair, our approach yields an executable policy. Our use of pGCL and PCTL [Kwiatkowska et al., 2007] results in a separation of action modelling and property specification. Although this separation is a non-essential difference, it syntactically supports a more independent working on two typical abstraction levels, required process properties and process implementation.

Verified Controller Synthesis for Cobots. A number of authors have suggested synthesis approaches for controllers in human-robot collaboration. Key features of these approaches are verified optimal synthesis for collaborative plan execution, quantitative verification of plans, code generation for deployment on robot platforms.

For generic robots, Orlandini et al. [2013] and Bersani et al. [2020] employ synthesis by game solving (i.e., finding winning strategies) over timed game automata (TIGA) supported by the model checker UPPAAL-TIGA. Correctness properties can be formulated as reach-avoid problems (i.e., reach the goal state and avoid unsafe states) specified as $\mathbf{A}[\mathit{safe}\,\mathbf{U}\,\mathit{goal}]$ in timed computation tree logic. From a timeline-based plan description [Cesta and Fratini, 2008], Orlandini et al. [2013] generate a TIGA with clock constraints encoding temporal degrees of freedom for performing control actions. From the TIGA, a winning strategy (i.e., a robust plan execution minimising violations of clock constraints) is then synthesised. The TIGA-based stage is continuously performed during operation. The distinction of controllable (i.e., duration known) from uncontrollable actions (i.e., duration unknown) allows one to react to temporally uncertain environmental events by obtaining strategies that can schedule robot actions in the presence of worst-case timing of the environment. In Cesta et al. [2016], an extension of this approach is applied to controller synthesis for safe human-robot collaboration. For task coordination between humans and robots, Cesta et al. [2016] and other works utilise the distinction of uncontrollable and controllable actions. Our reactive execution scheme (Figure 5) accommodates this basic feature required to separate the capabilities of the cobot from the capabilities of its physical environment.

Kshirsagar et al. [2019] demonstrate on-line controller synthesis for human-robot handovers obeying timing constraints specified in signal temporal logic. Cobot kinetics need to be given in terms of ordinary differential equations. Being suitable for low-level synthesis in homogeneous action systems, this approach could be integrated into our framework, for example, for controlling a speed and separation monitoring mode switched on during a handover.

MDP policy synthesis has also been used to generate optimal motion plans for mobile robots [Lahijanian et al., 2012]. The MDPs used in this solution model the physical layout of the space within which a mobile robot can navigate. As such, the problem addressed in this work is complementary to our use of probabilistic models and MDP policy synthesis, as our approach tackles the generation of optimal hazard mitigation actions.

Modelling of Controllers for Cobots. *Domain-specific languages* for controller design can support control engineers in encoding their control schemes. Cesta et al. [2016] employ domain and problem definition languages (DDL & PDL) for encoding the state spaces and action domains a controller can select from. Bersani et al. [2020] provide a lean language for robotic controller design. Models in that language typically include a description of the uncontrollable environment. Safety properties can be difficult to formulate, as shown in Bersani et al. [2020] and, particularly, when dealing with multiple hazards or risk factors. We provide YAP's input language for safety controller design informed by risk models, streamlining the specification of safety properties from multiple hazards.

Approaches such as Lahijanian et al. [2012] and Kshirsagar et al. [2019] focus on *homogeneous action systems*, that is, systems with few and similar types of actions manipulating type-wise simple state spaces (e.g. movement in a Euclidean plane or in a 2D grid). Such systems make it easier to provide complete and tractable synthesis algorithms for realistic models. In contrast, and as suggested by Kress-Gazit et al. [2018], our approach focuses on *heterogeneous action systems*, such as human-robot collaboration, which can be characterised by a wide variety of actions over a heterogeneously typed state space. Such actions can differ in their complexity and discrete or continuous nature (e.g. grabbing a work piece, moving a robot arm, performing a welding action, switching a mode, turning off an alarm sound).

Robust controllers are able to *deal with unstructured environments*, to react to a wide variety of uncontrollable adverse events (e.g. human error). Overall, they guarantee correct behaviour under weak assumptions. Game-based approaches, such as policy synthesis for TIGAs [Jessen et al., 2007, Orlandini et al., 2013, Bersani et al., 2020], inherently support such environments. Whilst our MDP-based approach benefits from more efficient algorithms and provides fine-grained methodological guidance, an extension to utilise the greater flexibility of game-based approaches should be part of our next steps.

A distinctive feature of safety controllers is their ability to *control the resumption of normal operation* (i.e., the recovery from a conservative or degraded safe state) in addition to mitigation. While resumption is implicit to many solutions for homogeneous-action reach-avoid problems [Orlandini et al., 2013, Bersani et al., 2020], for heterogeneous action systems, resumptions often need to be modelled separately (e.g. switching off a safety mode or function, resuming/restarting a suspended/cancelled task). Risk factors at the core of our approach accommodate primitives for specifying resumptions.

Overall, an advantage of timeline-based approaches [Cesta et al., 2016] is their ability to encode physical domains qualitatively. In pGCL, corresponding domain constraints need to be distributed over action guards, which can be cumbersome. Moreover, an advantage of TIGAs over MDPs is that time is continuous and managed via clocks, leading to more concise models. However, in summary, our use of pGCL enables the concise encoding of the action domain of human-robot collaboration with multiple actors operating over a discrete state space. MDPs represent a natural model of interaction of agents with an uncontrollable and uncertain environment. Reward-enhanced PCTL provides a flexible and expressive language for specifying multiple objectives (e.g. minimum risk, maximum performance) and reward constraints (e.g. accepted risk thresholds). We could enhance our approach to probabilistic timed automata in order to further increase model fidelity.

Controller Deployment. An important step in many synthesis approaches is the deployment of the resulting controllers in an execution environment. Although for another domain (i.e., climate control in a pig stable), Jessen et al. [2007] show how controllers synthesised using UPPAAL-TIGA can be embedded as components of a Simulink model to perform validation by simulation. As part of their evaluation, Bersani et al. [2020] demonstrate the deployment of their controllers on the low-cost platform Turtlebot used as a cobot. Orlandini et al. [2013] deploy their approach as a module on a $G^{en}{}_{o}M$-based robotic software platform with a mapping into the real-time framework BIP. Their flexible approach could be a potential route for future extensions of our approach. Currently, YAP provides a generator for C# modules for the DTF (Sections 3.4 and 6.2).

Overall, to the best of our knowledge, our method is the first end-to-end approach to synthesising and deploying safety controllers for handling multiple risks from collaborative robots in manufacturing processes. The works discussed above either address parts of the synthesis challenge or implement alternative solutions for cobot (safety) controllers.

9 CONCLUSION

We introduced a tool-supported software engineering approach for the verified synthesis of optimal safety controllers from Markov decision processes, focusing on human-robot collaboration. These software controllers implement regulatory safety goals for such applications. We describe steps for streamlined application modelling and risk-informed controller design and demonstrate our method using a tool chain consisting of YAP [Gleirscher, 2021] for structured risk modelling and pGCL program generation, EVOCHECKER [Gerasimou et al., 2018] for search-based policy synthesis from pDTMCs, and PRISM [Kwiatkowska et al., 2011] for probabilistic model checking and MDP policy synthesis. We show that our approach can be used to incrementally build up multi-hazard models including alternative mitigation and resumption strategies. We also discuss how our approach can simplify explicit model checking when dealing with large state spaces. Our approach improves the state of the art of controller synthesis for collaborative robots, particularly when dealing with multiple risks, mitigation options, activities and safety modes. That way, we contribute to the alignment of lower-level requirements posed by cobot safety standards (e.g. ISO 15066) with higher-level cobot hazard analysis and risk assessment [Chemweno et al., 2020] through a structured risk-informed design approach. Furthermore, we translate the verified controllers into executable code and deploy them on the digital twin framework and, thus, accomplish a smooth transition between formal controller verification and testing-based controller validation in a realistic environment. Using the DTF, we demonstrate the controller's correct and timely response in a representative use case. In summary, the verification and validation results generated by our approach can contribute evidence to a controller assurance case [Gleirscher et al., 2019, Foster et al., 2020, Calinescu et al., 2018].

Future Work. For optimal synthesis, the proposed method uses parameters such as upper risk and severity bounds as constraints. We plan to introduce parameters for probabilities, such as sensor failure and human error, into the MDP and to use parametric risk gradients by extending YAP.

It is important to model all relevant behaviours of the environment, or more generally the uncontrollable actions, the safety controller needs to respond to in order to increase freedom from accidents. Hence, in future work, we plan to use stochastic games to more accurately (and less conservatively) model the behaviour of the environment in which this controller operates. We also plan to explore online policy synthesis [Calinescu et al., 2017] to allow more variety in environmental decisions (e.g. mitigating hazards due to malicious operators). This corresponds to weakening the assumptions under which the controller can guarantee safety.

Accident data for the considered industrial application was generally not available. We were also unable[18] to collect data from the lab replica. Thus, we had to make best guesses of probabilities (cf. Section 4.3). However, the frequency of undesired intrusion of operators into the safeguarded area and accident likelihood can be transferred into our case study.

The DTF is a full-fledged digital twin, from which the interaction between the safety controller and the real system can be demonstrated. Limited access to the physical cell has required validation in the digital twin to take precedence. Further validation integrating the physical work cell will therefore be the focus of future work.

The case study can be extended by randomised control decisions with fixed probabilities (e.g. workload), by adding uncertain action outcomes (e.g. welding errors), and by time-dependent randomised choice of mitigation options. To use time in guarded commands, we want to explore clock-based models rather than only using reward structures, as far as the synthesis capabilities allow this.

For motion planning over finite partitions of geometric spaces, Lahijanian et al. [2012] describe algorithms for the synthesis of MDP policies that maximise the probability of a given arbitrary PCTL formula. By assuming that state estimation is precise, the authors avoid the use of partially observable MDPs in their application. To improve our approach regarding the synthesis over homogeneous action systems, their algorithms could be integrated into our MDP synthesis tool chain, for example, in addition to PRISM or underpinning the search-based synthesis in EVOCHECKER.

REFERENCES

Arash Ajoudani, Andrea Maria Zanchettin, Serena Ivaldi, Alin Albu-Schäffer, Kazuhiro Kosuge, and Oussama Khatib. Progress and prospects of the human-robot collaboration. *Autonomous Robots*, 42(5):957–975, 2017. doi: 10.1007/s10514-017-9677-2.

R. Alami, A. Albu-Schaeffer, A. Bicchi, R. Bischoff, R. Chatila, et al. Safe and dependable physical human-robot interaction in anthropic domains: State of the art and challenges. In *IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, pages 1–16, 2006. doi: 10.1109/iros.2006.6936985.

Rob Alexander, Heather Rebecca Hawkins, and Andrew John Rae. *Situation coverage – a coverage criterion for testing autonomous robots*, volume Report number YCS-2015-496. Department of Computer Science, University of York, February 2015.

Gary Anderson. The economic impact of technology infrastructure for advanced robotics. Economic analysis briefs, NIST, 2016. URL https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=921956.

Mehrnoosh Askarpour, Dino Mandrioli, Matteo Rossi, and Federico Vicentini. SAFER-HRC: Safety analysis through formal vERification in human-robot collaboration. In *LNCS*, pages 283–295. Springer, 2016. doi: 10.1007/978-3-319-45477-1_22.

Mehrnoosh Askarpour, Dino Mandrioli, Matteo Rossi, and Federico Vicentini. Formal model of human erroneous behavior for safety analysis in collaborative robotics. *Robotics and Computer-Integrated Manufacturing*, 57:465–476, June 2019. ISSN 07365845. doi: 10.1016/j.rcim.2019.01.001. URL https://linkinghub.elsevier.com/retrieve/pii/S0736584518303247.

A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *Dependable and Secure Computing, IEEE Transactions on*, 1(1):11–33, 2004. ISSN 1545-5971. doi: 10.1109/TDSC.2004.2.

Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. MIT P., 2008. ISBN 026202649X.

David Basin, Felix Klaedtke, Samuel Müller, and Eugen Zălinescu. Monitoring metric first-order temporal properties. *Journal of the ACM*, 62(2):1–45, 2015. doi: 10.1145/2699444.

Marcello M. Bersani, Matteo Soldo, Claudio Menghi, Patrizio Pelliccione, and Matteo Rossi. PuRSUE -from specification of robotic environments to synthesis of controllers. *Formal Aspects of Computing*, 32(2-3):187–227, July 2020. ISSN 0934-5043, 1433-299X. doi: 10.1007/s00165-020-00509-0. URL http://link.springer.com/10.1007/s00165-020-00509-0.

A. Bolton, L. Butler, I. Dabson, M. Enzer, M. Evans, T. Fenemore, and F. Harradence. The Gemini Principles. Technical report, Centre for Digital Built Britain, University of Cambridge, Cambridge, UK, 2018. URL https://www.cdbb.cam.ac.uk/system/files/documents/TheGeminiPrinciples.pdf.

Manfred Broy. A logical basis for component-oriented software and systems engineering. *The Computer Journal*, 53(10): 1758–82, 2010. doi: 10.1093/comjnl/bxq005.

Radu Calinescu, Marco Autili, Javier Cámara, Antinisca Di Marco, Simos Gerasimou, Paola Inverardi, Alexander Perucci, Nils Jansen, Joost-Pieter Katoen, Marta Kwiatkowska, et al. Synthesis and verification of self-aware computing systems. In *Self-Aware Computing Systems*, pages 337–373. Springer, 2017. doi: 10.1007/978-3-319-47474-8_11.

Radu Calinescu, Danny Weyns, Simos Gerasimou, Muhammad Usman Iftikhar, Ibrahim Habli, and Tim Kelly. Engineering trustworthy self-adaptive software with dynamic assurance cases. *IEEE Transactions on Software Engineering*, 44(11):1039–1069, November 2018. ISSN 0098-5589. doi: 10.1109/TSE.2017.2738640.

[18] Due to restricted lab access during the COVID-19 pandemic.

Amedeo Cesta and Simone Fratini. The timeline representation framework as a planning and scheduling software development environment. In *Proc. of 27th Workshop of the UK Planning and Scheduling SIG*, pages 1–8, 2008. URL https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.432.1860&rep=rep1&type=pdf.

Amedeo Cesta, Andrea Orlandini, Giulio Bernardi, and Alessandro Umbrico. Towards a planning-based framework for symbiotic human-robot collaboration. In *Emerging Technologies and Factory Automation (ETFA), 21st Int. Conf.*, pages 1–8. IEEE, sep 2016. doi: 10.1109/etfa.2016.7733585.

Peter Chemweno, Liliane Pintelon, and Wilm Decre. Orienting safety assurance with outcomes of hazard analysis and risk assessment: A review of the ISO 15066 standard for collaborative robot systems. *Safety Science*, 129:104832, 2020. doi: 10.1016/j.ssci.2020.104832.

Matthew B. Dwyer, G. S. Avrunin, and J. C. Corbett. Patterns in property specifications for finite-state verification. In *ICSE*, pages 411–20, 1999. doi: 10.1145/302405.302672.

Simon Foster, Mario Gleirscher, and Radu Calinescu. Towards deductive verification of control algorithms for autonomous marine vehicles. In *Engineering of Complex Computer Systems (ICECCS), 25th Int. Conf., Singapore*, pages 113–118, 2020. ISBN 978-1-7281-8558-3. doi: 10.1109/ICECCS51672.2020.00020.

Simos Gerasimou, Radu Calinescu, and Giordano Tamburrelli. Synthesis of probabilistic models for quality-of-service software engineering. *Automated Software Engineering*, 25(4):785–831, 2018. doi: 10.1007/s10515-018-0235-8.

R. B. Gillespie, J. E. Colgate, and M. A. Peshkin. A general framework for cobot control. *IEEE Transactions on Robotics and Automation*, 17(4):391–401, 2001. doi: 10.1109/70.954752.

Mario Gleirscher. Hazard-based selection of test cases. In *Automation of Software Test (AST), 6th ICSE Workshop*, pages 64–70, 2011. doi: 10.1145/1982595.1982609.

Mario Gleirscher. Run-time risk mitigation in automated vehicles: A model for studying preparatory steps. In *1st iFM Workshop Formal Verification of Autonomous Vehicles (FVAV)*, number 257.8 in EPTCS, pages 75–90, 2017. doi: 10.4204/eptcs.257.8.

Mario Gleirscher. YAP: Tool support for deriving safety controllers from hazard analysis and risk assessments. In Matt Luckuck and Marie Farrell, editors, *Formal Methods for Autonomous Systems (FMAS), 2nd Workshop*, volume 329 of *EPTCS*, pages 31–47. Open Publishing Association, 2020. doi: 10.4204/EPTCS.329.4.

Mario Gleirscher. YAP *Against Perils: Application Guide and User's Manual*. University of York and Technical University of Munich, 2021. URL https://yap.gleirscher.de/dl/yap-0.7-manual.pdf.

Mario Gleirscher and Radu Calinescu. Safety controller synthesis for collaborative robots. In *Engineering of Complex Computer Systems (ICECCS), 25th Int. Conf., Singapore*, pages 83–92, 2020. ISBN 978-1-7281-8558-3. doi: 10.1109/ICECCS51672.2020.00017.

Mario Gleirscher, Simon Foster, and Yakoub Nemouchi. Evolution of formal model-based assurance cases for autonomous robots. In *17th Int. Conf. SEFM*, volume 11724 of *LNCS*, pages 87–104. Springer, 2019. doi: 10.1007/978-3-030-30446-1_5.

Mario Gleirscher, Radu Calinescu, and Jim Woodcock. Risk structures: A design algebra for risk-aware machines. *Formal Aspects of Computing*, 2021.

David Griffin, Iain Bate, and Robert I. Davis. Generating utilization vectors for the systematic evaluation of schedulability tests. In *IEEE Real-Time Systems Symposium, RTSS 2020, Houston, Texas, USA*, pages 76–88. IEEE, 2020. doi: 10.1109/RTSS49844.2020.00018. URL https://www-users.cs.york.ac.uk/~robdavis/papers/DRSRTSS2020.pdf.

Sami Haddadin, Alin Albu-Schäffer, and Gerd Hirzinger. Requirements for safe robots: Measurements, analysis and new insights. *The Int. Journal of Robotics Research*, 28(11-12):1507–1527, 2009. doi: 10.1177/0278364909343970.

Bradley Hayes and Brian Scassellati. Challenges in shared-environment human-robot collaboration. In *Collab. Manipulation Workshop at HRI*, pages 1–6, 2013.

Jochen Heinzmann and Alexander Zelinsky. Quantitative safety guarantees for physical human-robot interaction. *The Int. Journal of Robotics Research*, 22(7-8):479–504, 2003. doi: 10.1177/02783649030227004.

E. Helms, R. D. Schraft, and M. Hagele. rob@work: Robot assistant in industrial environments. In *11th IEEE Int. Workshop on Robot and Human Interactive Communication*, pages 399–404, 2002. doi: 10.1109/roman.2002.1045655.

ISO 10218. Robots and robotic devices – safety requirements for industrial robots. Standard, Robotic Industries Association (RIA), 2011. URL https://www.iso.org/standard/51330.html.

ISO/TS 15066. Robots and robotic devices – collaborative robots. Standard, Robotic Industries Association (RIA), 2016. URL https://www.iso.org/standard/62996.html.

Jan Jakoh Jessen, Jacob Illum Rasmussen, Kim G. Larsen, and Alexandre David. Guided controller synthesis for climate controller using UPPAAL tiga. In Jean-François Raskin and P. S. Thiagarajan, editors, *Formal Modeling and Analysis of Timed Systems*, volume 4763, pages 227–240. Springer, Berlin, Heidelberg, 2007. ISBN 978-3-540-75453-4. doi: 10.1007/978-3-540-75454-1_17.

Richard Hugh Jones. *A Study of Safety and Production Problems and Safety Strategies Associated with Industrial Robot Systems*. PhD thesis, Imperial College, 1986.

Lukas Kaiser, Andreas Schlotzhauer, and Mathias Brandstötter. Safety-related risks and opportunities of key design-aspects for industrial human-robot collaboration. In *LNCS*, pages 95–104. Springer, 2018. doi: 10.1007/978-3-319-99582-3_11.

Hadas Kress-Gazit, Morteza Lahijanian, and Vasumathi Raman. Synthesis for Robots: Guarantees and Feedback for Robot Behavior. *Annual Review of Control, Robotics, and Autonomous Systems*, 1(1):211–236, May 2018. ISSN 2573-5144. doi: 10.1146/annurev-control-060117-104838. URL https://www.annualreviews.org/doi/10.1146/annurev-control-060117-104838.

Werner Kritzinger, Matthias Karner, Georg Traar, Jan Henjes, and Wilfried Sihn. Digital Twin in manufacturing: A categorical literature review and classification. *IFAC*, 51(11):1016–1022, 2018. ISSN 24058963. doi: 10.1016/j.ifacol.2018.08.474.

Alap Kshirsagar, Hadas Kress-Gazit, and Guy Hoffman. Specifying and Synthesizing Human-Robot Handovers. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5930–5936, Macau, China, November 2019. IEEE. ISBN 978-1-72814-004-9. doi: 10.1109/IROS40897.2019.8967709. URL https://ieeexplore.ieee.org/document/8967709/.

Marta Kwiatkowska, Gethin Norman, and David Parker. Stochastic model checking. In M. Bernardo and J. Hillston, editors, *Formal Methods for the Design of Comp., Comm. and Soft. Sys.: Performance Evaluation (SFM)*, volume 4486 of *LNCS*, pages 220–70. Springer, 2007. doi: 10.1007/978-3-540-72522-0_6.

Marta Kwiatkowska, Gethin Norman, and David Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *23rd CAV*, volume 6806 of *LNCS*, pages 585–591. Springer, 2011. doi: 10.1007/978-3-642-22110-1_47.

Morteza Lahijanian, Sean B. Andersson, and Calin Belta. Temporal Logic Motion Planning and Control With Probabilistic Satisfaction Guarantees. *IEEE Transactions on Robotics*, 28(2):396–409, April 2012. ISSN 1941-0468. doi: 10.1109/TRO.2011.2172150. Conference Name: IEEE Transactions on Robotics.

Nancy Gail Leveson. *Safeware: System Safety and Computers*. Addison-Wesley, 1995. ISBN 9780201119725.

Nancy Gail Leveson. *Engineering a Safer World: Systems Thinking Applied to Safety*. Engineering Systems. MIT P., 2012. ISBN 9780262016629. doi: 10.7551/mitpress/8179.001.0001.

Philip Long, Christine Chevallereau, Damien Chablat, and Alexis Girin. An industrial security system for human-robot coexistence. *Industrial Robot: An Int. Journal*, 45(2):220–226, 2018. doi: 10.1108/ir-09-2017-0165.

Jeremy A. Marvel, Joe Falco, and Ilari Marstio. Characterizing task-based human-robot collaboration safety in manufacturing. *IEEE Tran. on Systems, Man, and Cybernetics: Systems*, 45(2):260–275, 2015. doi: 10.1109/tsmc.2014.2337275.

Bjoern Matthias, Soenke Kock, Henrik Jerregard, Mats Kallman, and Ivan Lundberg. Safety of collaborative industrial robots: Certification possibilities for a collaborative assembly robot concept. In *IEEE Int. Symposium on Assembly and Manufacturing (ISAM)*, pages 1–6, 2011. doi: 10.1109/isam.2011.5942307.

Elisa Negri, Luca Fumagalli, and Marco Macchi. A Review of the Roles of Digital Twin in CPS-based Production Systems. *Procedia Manufacturing*, 11(June):939–948, 2017. ISSN 23519789. doi: 10.1016/j.promfg.2017.07.198.

Peter Nicolaisen. Occupational safety and industrial robots. In Bonney and Yong, editors, *Robot Safety*, pages 33–48. IFS, 1985. doi: 10.1007/978-3-662-02440-9_9.

Andrea Orlandini, Marco Suriano, Amedeo Cesta, and Alberto Finzi. Controller synthesis for safety critical planning. In *Tools with Artificial Intelligence (ICTAI), 25th Int. Conf.*, pages 1–8. IEEE, 2013. doi: 10.1109/ictai.2013.54.

David Parnas and Jan Madey. Functional documentation for computer systems. *Science of Computer Programming*, 25:41–61, 1995. doi: 10.1016/0167-6423(95)96871-J.

Agostino De Santis, Bruno Siciliano, Alessandro De Luca, and Antonio Bicchi. An atlas of physical human–robot interaction. *Mechanism and Machine Theory*, 43(3):253–270, 2008. doi: 10.1016/j.mechmachtheory.2007.03.003.

Neville A. Stanton. Hierarchical task analysis: Developments, applications, and extensions. *Applied Ergonomics*, 37(1):55–79, 1 2006. doi: 10.1016/j.apergo.2005.06.003.

N. Sugimoto. Safety engineering on industrial robots and their draft standards for safety requirements. In *7th Int. Symposium on Industrial Robots*, pages 461–470, 1977.

Fei Tao, Jiangfeng Cheng, Qinglin Qi, Meng Zhang, He Zhang, and Fangyuan Sui. Digital twin-driven product design, manufacturing and service with big data. *The International Journal of Advanced Manufacturing Technology*, 94(9):3563–3576, 2018.

Marcell Vazquez-Chanlatte. mvcisback/py-metric-temporal-logic: v0.1.1, 1 2019. URL https://doi.org/10.5281/zenodo.2548862.

F. Vicentini, M. Askarpour, M. G. Rossi, and D. Mandrioli. Safety Assessment of Collaborative Robotics Through Automated Formal Verification. *IEEE Transactions on Robotics*, 36(1):42–61, February 2020. ISSN 1941-0468. doi: 10.1109/TRO.2019.2937471. Conference Name: IEEE Transactions on Robotics.

Valeria Villani, Fabio Pini, Francesco Leali, and Cristian Secchi. Survey on human-robot collaboration in industrial settings: Safety, intuitive interfaces and applications. *Mechatronics*, 55:248–266, 2018. doi: 10.1016/j.mechatronics.2018.02.009.

Xi Vincent Wang, Zsolt Kemény, József Váncza, and Lihui Wang. Human-robot collaborative assembly in cyber-physical production: Classification framework and implementation. *CIRP Annals*, 66(1):5–8, 2017. doi: 10.1016/j.cirp.2017.04.101.