

From Specification Models to Explanation Models: An Extraction and Refinement Process for Timed Automata

Maike Schwammberger*

University of Oldenburg,
Oldenburg, Germany

`schwammberger@informatik.uni-oldenburg.de`

Verena Klös

TU Berlin,
Berlin, Germany

`verena.kloes@tu-berlin.de`

Autonomous systems control many tasks in our daily lives. To increase trust in those systems and safety of the interaction between humans and autonomous systems, the system behaviour and reasons for autonomous decision should be explained to users, experts and public authorities. One way to provide such explanations is to use behavioural models to generate context- and user-specific explanations at run-time. However, this comes at the cost of higher modelling effort as additional models need to be constructed. In this paper, we propose a high-level process to extract such *explanation models* from system models, and to subsequently refine these towards specific users, explanation purposes and situations. By this, we enable the reuse of specification models for integrating self-explanation capabilities into systems. We showcase our approach using a running example from the autonomous driving domain.

Keywords. Explanation model, self-explainability, formal models, timed automata, model extraction, model refinement, model reuse

1 Introduction

Nowadays, autonomous systems control many areas of our daily lives, e.g. tasks in transportation, medicine or industry. These tasks require context-awareness, and errors and failures might have severe consequences. To increase trust in and safety of those systems, the system behaviour and reasons for autonomous decision should be explained to users, experts and public authorities. Explainability has become a hot research topic in the area of artificial intelligence [11, 14], but also becomes more important for any autonomous system [5, 13, 26, 28].

One way to explain the behaviour of autonomous systems is to directly encode the generation of explanations into the system code (e.g., done in [2]). However, this approach only allows for generating explanations for behaviours that have been classified as relevant at design time. A more flexible approach, that also facilitates the adjustment of explanations to the explanation recipient and context, is the generation of explanations at run-time. While the first approach can profit from direct access to the encoded system decisions, the latter needs models of the system to identify the events and decisions that led to the current system behaviour. We call these models *explanation models*. These models are either manually constructed at design time (e.g., in [8]), or learned from data (e.g., in [20, 28]). In [7], we have proposed the MAB-EX framework that uses explanation models to generate explanations on demand, at run-time. There, we have defined explanation models as follows:

An explanation model is a behavioural model of the system that captures causal relationships between events and system reactions. It allows for identifying possible causes for the

*M. Schwammberger was supported by the German Research Council (DFG) in the PIRE Projects SD-SSCPS and ISCE-ACPS under grant no. FR 2715/4-1 and FR 2715/5-1.

behaviour that needs to be explained, e.g., traces of events that may lead to the behaviour. It may also allow for look-ahead simulation to enable answering questions like “What happens if ... ?” or “When will ... be possible again?”. [7, p. 544-545]

The flexibility of model-based explanations comes at the cost of higher modelling effort as explanation models are additional models that need to be constructed. These models have to capture causal relationships between events and system reactions. They should enable the identification of the current state from monitored data and back-tracing with monitored data to identify the chain of causes. An optional requirement is to allow for simulation to answer questions on alternatives and possible future behaviour. Thus to construct them, deep insights into the system behaviour in different situations and contexts is needed. Building them is a new challenge, that has not yet been researched thoroughly.

Contribution. In this paper, we propose to reduce the modelling effort by extracting explanation models from formal system models that were used to specify and verify the system behaviour at design time and to subsequently refine these for users, explanation purposes and situations. The notion of “refinement” that we use within this paper differs from the well-known refinement concept from formal methods. Instead, with the term refinement, we describe a process where a coarse explanation model is adapted to a more detailed structure. Thus with our meaning of refinement, the explanation-capability of an explanation model is increased. For now, we focus on an approach to extract initial models from timed automata [4] that is based on the vision paper [24]. Furthermore, we propose a high-level extraction and refinement process that describes which information should be hidden or added for which purpose. We also sketch continuous refinement at run-time to allow for user-specific adjustments and for integrating new information. We illustrate the steps of our high-level approach with a running example from the domain of autonomous driving. Note that the aim of this paper is to present the idea and general process of extracting explanation models from system models but that we do not yet give an implementation.

The main advantages of our high-level extraction and refinement process are:

- **Model reuse:** By extracting an explanation model from an existing system model, we enable the reuse of design-time models
- **Formal Foundation:** By using a formal, verifiably correct system model like a timed automaton, we avoid generating explanations from ambiguous natural language (e.g. directly from requirements).
- **Modularity:** With the help of the created explanation model and our MAB-EX framework, we can introduce self-explainability to existing systems and also facilitate system updates at run-time, as the explanation model can also be updated at run-time.
- **User-specificity:** The output of our framework is an explanation model that takes different types of explainees into account. This is reasonable as, for instance, an engineer might need differently detailed explanations than an end-user.

Outline. In the following, we briefly present the MAB-EX framework [7] for self-explaining systems that uses explanation models to construct explanations on demand. Afterwards, we present our explanation model extraction and refinement process in Sect.3 and introduce our running example in Sect. 4. We explain the extraction phase in detail in Sect. 5 and the refinement phase in Sect. 6. In Sect. 7, we discuss reasons for run-time adaptation of explanation models. We discuss our approach in Sect. 8 and conclude the paper in Sect. 9.

2 Preliminaries: MAB-EX Framework for Self-explaining Systems

To enable the design of self-explainable systems, we have proposed the MAB-EX Loop (Monitor, Analyse, Build, Explain) in previous work [7]. The main idea of this reference framework is to adopt the main principles of the well-known MAPE Loop [1] for self-adaptive systems to build self-explaining systems. We depict the MAB-EX Loop in Fig. 1.

The MAB-EX Loop monitors and analyses the behaviour of a system and decides whether the user (or another stakeholder) requires an explanation. This decision can depend on different factors: on the current context, on the user’s experience with similar situations, or on the characteristics of the situation (e.g., rareness). For more factors that influence the need for an explanation, we refer to [22]. If such an explanation need was detected, the next phases of the loop then build an explanation from explanation models and convey this explanation in a suitable way to the stakeholder. Building an explanation and actually presenting the explanation are separated in the MAB-EX Loop to allow for individual explanations for different stakeholders and contexts.

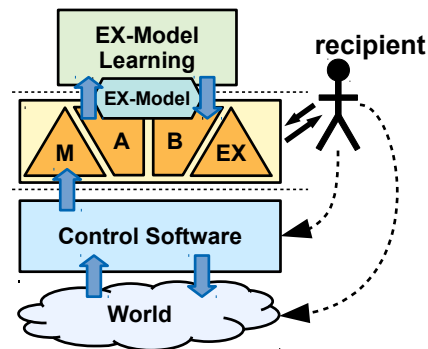


Figure 1: MAB-EX Loop from [7].

Possible implementations for an explanation model, that were discussed and illustrated in the paper, are (fault/decision) trees that connect observations to possible reasons and executable behaviour models, e.g., state machines. In the *Build*-phase, the observed behaviour is used to identify the current state in the explanation model and the events that have led to this state. These events form a trace, which we refer to as *explanation path* in the remainder of the paper. This path is an internal representation of the explanation that is further processed in the *EX*-phase of the loop, e.g. enriched with further information and transformed into a suitable presentation format that is given to the *explainee* (i.e. the recipient of the explanation).

Note that dynamically building explanations makes most sense for complex context-sensitive, maybe self-adaptive or learning systems where it is impossible to predict all situations that require an explanation beforehand. Furthermore, by adding an explanation layer that follows the MAB-EX framework, it is possible to add explanation capabilities to an existing system.

In this paper, we focus on constructing explanation models that can be used in the MAB-EX framework. Additionally, and in contrast to [7], we propose to already tailor the explanation models to different types of explainee. By this, the generated explanation path is reduced to events that are relevant for the explainee and the subsequent processing in the *EX*-phase can be simplified.

3 Explanation Model Extraction and Refinement Process

Our overall goal is to introduce a multi-level extraction and refinement process for making a variety of existing systems self-explainable. For this, we suggest that an *explanation model* is extracted from an existing *system model* and then further refined to cope with different explanation purposes and explainees. As motivated in Sect. 2, an explanation model is a causal structure that connects system actions with their reasons (i.e. events preceding the action). From such an explanation model, explanations may be generated on demand, at run-time. In the following, we motivate some key concepts of our approach, before we introduce the actual explanation model extraction and refinement process.

3.1 Type of system model.

In our terminology, a system model is some technical description of the system’s behaviour, goals and general functionality. We envision that our approach is not limited to a specific type of system: it summarises engineering steps needed to derive an explanation model from a system model. The extraction of an initial explanation model is done by exploiting the syntactical and semantical structure of the system model. In this paper, we use timed automata system models. Similar steps are also applicable for other types of system models. However, the notion of causality in different models might be very different, and influences the steps in our process.

3.2 Types of explanations.

Across domains and throughout research disciplines, different types of explanations are examined for their applicability and usefulness. In the context of semi-autonomous driving, the authors of [16] discovered that “why” (e.g., “Obstacle ahead”) explanations are preferred over “how” explanations (e.g., “The car is braking”) by drivers and led to a better driving performance. A combination of both “why” and “how” explanations led to the safest driving performance. Research results of [19] substantiate these findings as the authors describe that “why” explanations can improve a user’s trust in a system and are more easily understood by the user than “why not” explanations.

We follow these approaches and consider reasoning traces that combine “how” and “why” explanation types, e.g.: “The car did brake (“how”), because an obstacle is ahead (“why”). In our case, these reasoning traces relate to the explanation paths, that we introduced in Sect. 2.

3.3 Explainee types and explanation purposes.

A key feature of our approach is that we take different *types of explainees* into account. The term explainee comprises the recipient of an explanation. In [18], the authors also suggest, as a first requirement for explainability, to characterise traits of different types of explainees. Such different types of explainees can, for example, be deduced from the stakeholders that are identified in the requirements engineering phase of the system engineering process [9]. Here, we postulate that differently detailed explanation models are needed for different explainee types and for different *explanation purposes*. By explanation purpose, we mean the topic or circumstance that needs to be explained. Examples of explainee types and explanation purposes are:

- an end-user, who needs to cooperate with the system or who simply wishes to understand some system behaviour;
- an engineer, who needs to understand a system failure;

- a lawyer or the general public, who need to figure out whether a system is responsible for an accident; or
- another system that interacts with the considered system, e.g. two autonomous cars from different manufacturers that need to cooperate.

Note that mechanisms for exploring the needs of different explainee types are out of the scope of this paper, but we refer to [18] for this, where requirements for explainability in general are explored.

3.4 The multi-level extraction and refinement process – Overview.

We give an overview of our framework in Fig. 2. We distinguish three different phases (each having a separate box in Fig. 2) which lead from a formal system model SysModel to a user-specific explanation model $\text{EM5}(x_1)$, $\text{EM5}(x_2)$, ..., $\text{EM5}(x_n)$, for specific individual explainees x_1, x_2, \dots, x_n . We list and briefly describe these three phases in the following and give more details in Sects. 5 to 7, respectively.

Note that each of the steps that we propose is meant to optimise an initially extracted explanation model. This means that each of the intermediate explanation models is already functional on its own. Such intermediate models might be useful for an integration into other approaches and implementations of explainability. If necessary, for example to achieve a fully automatic extraction and refinement, it is possible to skip some of the steps, although then the resulting explanation model may not be optimised towards giving explanations.

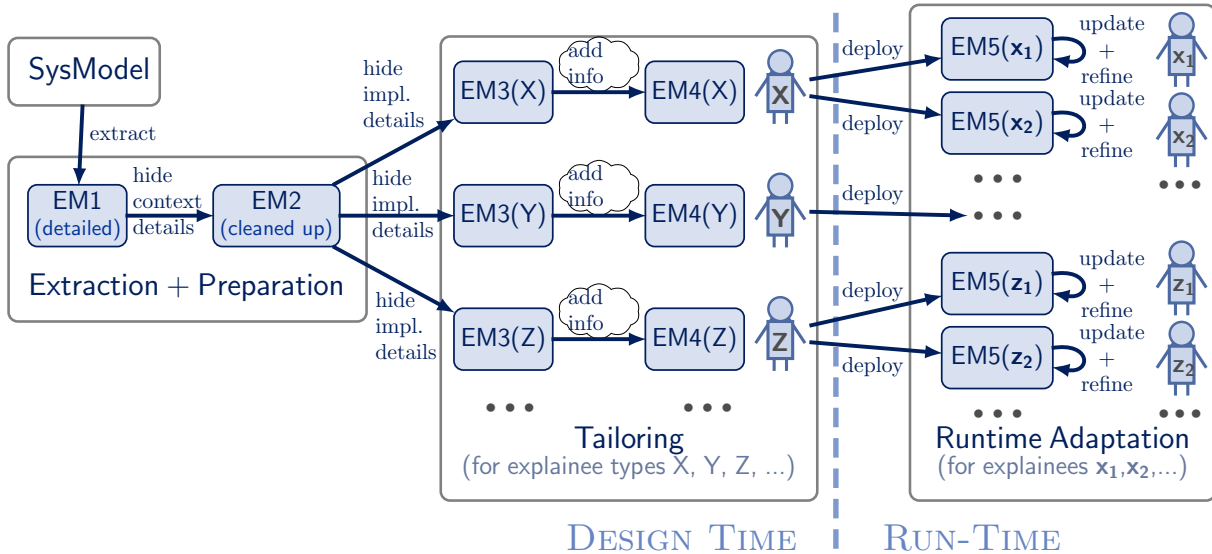


Figure 2: Overview of the explanation model creation process.

Phase 1: Extraction and Preparation (Sect. 5). This phase of the framework contains two different steps: Extraction of a first version of the explanation model (EM1) from the system model and a preparation of this extracted model for further phases.

Step 1: Extraction. In the case of a system model that is modelled as an automaton, this could, e.g., be done by connecting actions with their causes, as is done in the vision paper [24]. However, [24] does not give any methodology for retrieving the explanation model from the crossing controller and omits details in the model without giving reasons for that. The data contained in such an extracted explanation

model EM1 is not yet filtered and contains reasons for all details from the system model, possibly also details that are irrelevant for the explanation topic.

Step 2: Preparation. As an unnecessarily large explanation model is not desirable (e.g. because of too large computation times for time-critical explanations), some details are hidden in the preparation step of this phase, leading to EM2. Let us consider an example from a smart factory, where an autonomous robot fetches and delivers parcels between different work stations: We assume that the behaviour of the robot is modelled via two different system models that are working in parallel:

- System model 1: Identify, grab or lay down a parcel
- System model 2: Move, stop, turn left, turn right

If an engineer is interested in explanations for the acceleration or deceleration of the robot (i.e. reasons for the actions “move” and “stop”), it might be not necessary to keep the reasons for turning left and right within the explanation model EM1. Thus, these parts from EM1 are hidden upon the creation of EM2 in this example.

Phase 2: Tailoring (Sect. 6). The second phase is again parted into two steps: Firstly, explainee specific parts of EM2 are hidden, and secondly the resulting model EM3 is enriched by details that were missing in the system model. These details are added by human experts or could be extracted from requirements specifications or other (formalised) documents.

Step 1: Hiding explainee specific details. For different explainees, differently detailed information is needed for an explanation and hence also for the explanation model. For instance, while exact values of internal data variables might be of interest for an engineer explainee type X , e.g. for analysing the system, these values will have little to no meaning for an end-user explainee type Y ($X, Y \in \text{ExplaineeTypes}$). Thus, in this step, we start with EM2 and hide those details, that are irrelevant for a specific type of explainee X or Y . The result of this first step are explanation models EM3(X) and EM3(Y), for respective explainee types X, Y .

Step 2: Add expert information. Specific information, e.g., that an autonomous vehicle is breaking because of a traffic rule, might not be directly included in SysModel. Instead, it might be encoded that the car brakes because the value of a variable b is not allowed to exceed a constant value c . Thus, we enrich abstract internal system data with environmental information. Such information may be, e.g., gathered from the system requirements or a system engineer. Although this step requires some high-level insights into reasons for the system behaviour, it is still less complex than having to create the whole explanation model from scratch. The needed level of detail of this information is again dependant on the explainee, as, e.g., the value of a variable might have a sufficient meaning for an engineer or another system, but not for an end-user. The result of phase 2 is explanation model EM4(X). Note that, when involving a human expert, this step can only be semi-automated. However, there exist possibilities for automating this step: additional information may be extracted from formalised requirements (cf. [10]), formalised traffic rule books (cf. [15]) and similar sources.

Phase 3: Run-time Adaptation (Sect. 7). In our framework, the result of phase 2, the explanation model EM4(X), is meant to be deployed in a system to make it self-explainable for explainee type X . During run-time, further updates and adaptations can be necessary to consider evolving personal preferences of individual explainees or changes in the system behaviour due to self-adaptation, learning or software updates. The initially deployed explanation model EM5(x_1) for explainee $x_1 \in X$ in this phase is the same as EM4(X).

Note that, depending on the used system model SysModel, the explanation purpose and the considered type of explainee, it may be that some of the steps are omitted. For instance, if all of the details in EM2 are needed for an engineer explainee type, we may not hide details and EM2 is also used as EM3.

4 Set-Up for Running Example

We showcase the use and key features of our framework by using a specific running example with a specific system model SysModel in this paper: an extended timed automaton. Our example is from the domain of autonomous driving and considers a crossing protocol for autonomous turn manoeuvres at urban intersections. It is a simplified adaptation of a crossing protocol that was introduced in [6, 23]. We chose this example, as it comprises a concise, formal, definition of traffic manoeuvres and its soundness and the key features of the protocol have already been formally proven (e.g. safety). Furthermore, timed automata [4] are a popular mechanism for the specification of various system types. With this, our running example also conforms to other systems. We decided to simplify the protocol, as the introduction to the underlying formalism *Automotive Controlling Timed Automata (ACTA)* with its traffic logic *Urban Multi-lane Spatial Logic (UMLSL)* from [6, 23] would be beyond the scope of this paper, and the formal details are not necessary for the purpose of the running example: Showcasing and illustrating our extraction and refinement process from Sect. 3. We refer to [6, 23] for the formal details.

A first visionary approach for extracting an explanation model for the crossing controller from [6] was introduced in [24]. An example for a traffic situation that can be handled by the protocol is depicted in Fig. 3, where an *ego* car E approaches an intersection at which cars B and C are currently performing turn manoeuvres. With *ego* car, we refer to the car from whose viewpoint the traffic situation is considered. The *ego* car's goal is to turn left at the intersection.

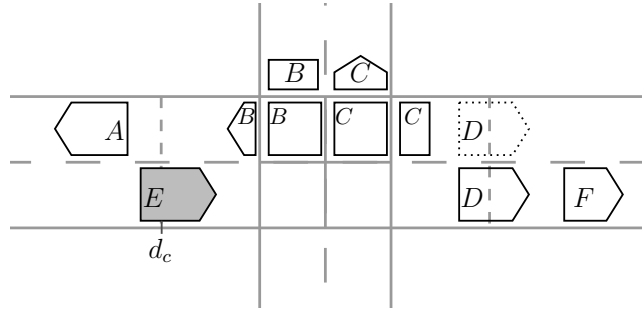


Figure 3: Example traffic situation where cars A to F meet at an urban intersection.

The simplified crossing controller that we present in Sect. 4.1 is an extended timed automaton. For extracting an explanation model from a timed automaton, we suggest to identify and connect *actions* and *reasons* in the automaton. For this, consider the schematic timed automaton transition that we depict in Fig. 4. Actions are elements that appear after $/$ and their reasons are the transition guards before $/$, as well as possible invariants in the starting location of the transition. Actions can be communicating events or operations on data and clock variables, which can also be specified in method code in some dialects of timed automata. Guards specify logical propositions on data and clock variables, which can also be encapsulated in method code, or specify communication events that have to be received. Invariants describe data and clock constraints that must not be violated while staying in the location.



Figure 4: Schematic timed automaton transition.

4.1 Crossing Controller Protocol

We depict our simplified crossing protocol in Fig. 5 and explain it in the following. The goal of this protocol is that an autonomous car, like the *ego* car E from Fig. 3, can safely turn at an intersection. The protocol also implements a fairness property, where only the car with the highest priority may enter an intersection. Each car C starts with a priority $p_c = 0$ and the longer a car waits, the more this priority increases.

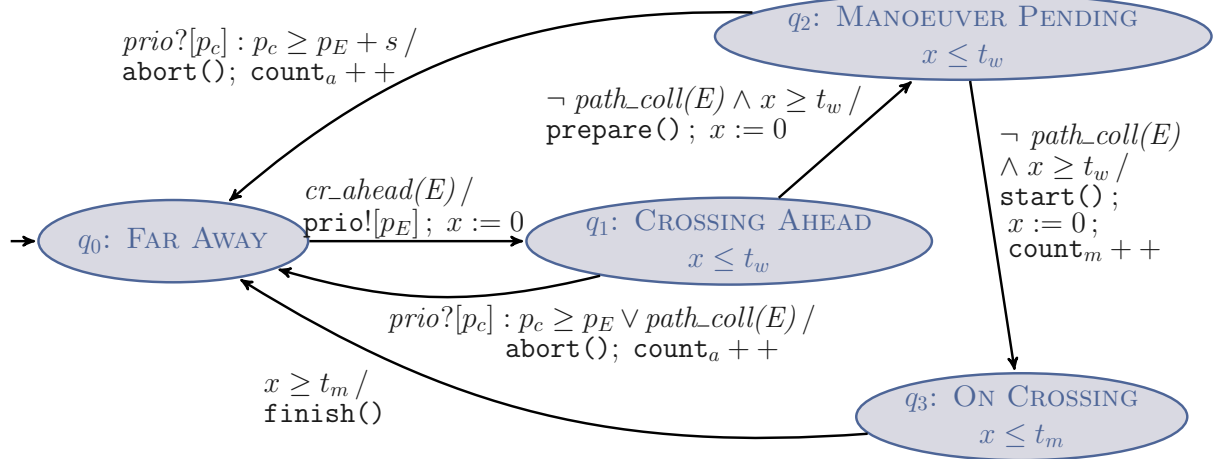


Figure 5: Simplified crossing controller protocol, inspired by [6].

The protocol from Fig. 5 summarises four phases, where each phase relates to one location q_0 to q_3 in the depicted timed automaton:

1. q_0 FAR AWAY: No crossing is in range.
2. q_1 CROSSING AHEAD: On approaching a crossing ($cr_ahead()$), the *ego* car E sends its own priority p_E for entering via broadcast ($prio![p_E]$) and compares it with the priorities of other traffic participants, and potential path collisions ($path_coll(E)$) are checked.
3. q_2 MANOEUVRE PENDING: The *ego* car E determines that its priority is the highest and that the desired path is free ($\neg path_coll(E)$) before entering this phase. If E 's priority p_E were smaller than the priority p_c of an arbitrary car C , the guarded input action $prio?[p_c] : p_c \geq p_E$ on the transition from location q_1 to q_0 would be valid and q_1 would be left for q_0 .
4. q_3 ON CROSSING: The crossing is only entered if E has the highest priority p_E and no potential path collisions have been detected ($\neg path_coll(E)$).

Note that in phase 3 of the protocol, q_2 MANOEUVRE PENDING, the *ego* car E is about to enter the intersection, as its priority is higher than any other cars' priorities and as no collisions with its planned path through the intersection have been identified. The only possibility for *ego* to abort its manoeuvre in this phase of the protocol is if a car with a significantly larger priority p_c with $p_c \geq p_E + s$ arrives, e.g. an emergency vehicle. Such an emergency vehicle would not start with a priority of 0 on arriving at the intersection, but with a much larger initial priority value.

In the timed automaton crossing protocol of Fig. 5, we observe three different types of *actions*:

- Communication actions: With $prio![p_E]$, E 's priority p_E is sent via broadcast to all other cars.

- Controller actions: Preparing (`prepare()`), aborting (`abort()`), starting (`start()`) and finishing (`finish()`) a crossing manoeuvre.
- Operations on data variables and clock resets: E.g., $x := 0$ to reset the value of a clock variable x or `countm` (resp. `counta`) to increase a counter for started (resp. aborted) manoeuvres.

The controller actions are a construct that has been defined for the special type of extended timed automata, Automotive-Controlling Timed Automata, in [23], but can be understood as abstract functions for our purposes in this paper. To model that actions like `start()` do not happen immediately, we add a time invariant, combined with a respective guard on the outgoing transitions, to most locations of the protocol. For instance consider the invariant $x \leq t_w$ in location q_1 , which specifies that the location must be left after at most t_w time units. Combined with the guard $x \geq t_w$ on the outgoing transition to q_2 , this transition can only be taken after exactly t_w time units. Location q_1 can only be left for location q_0 earlier than t_w time units, if either a path collision was detected with the guard `path_coll()` or if a higher priority p_c has been received and identified via the *guarded input action* `prio?[pc] : pc ≥ pE`. The communication semantics of timed automata specifies that, if a communication via the *output action* `prio![pC]` has been received from another car C 's timed automaton controller, and if the communication guard $p_c \geq p_E$ holds for a variable valuation $v(c) = C$ of the variable c , the ego controller *must* synchronise with this communication and thus change back to location q_0 . For more details on this type of guarded broadcast communication, we refer to [23].

5 Phase 1: Extraction and Preparation

The first phase of our framework in Fig. 2 contains two steps, which we describe separately: First (Sect. 5.1), the explanation model is extracted from the system model and then (Sect. 5.2) the extracted model is prepared for the next phase of the framework. In each section, we first describe the respective step in general, and after that we apply the step to our running example (cf. Sect. 4).

5.1 Extraction

In this step, we extract a first version EM1 of the explanation model from the system model SysModel. For this, we require the following assumptions to hold for our system model:

1. **Completeness:** Each possible system behaviour is modelled within the system model SysModel.
2. **Correctitude:** The system model is required to accurately and correctly represent the system's behaviour.

If the first assumption is not satisfied, we can still extract an explanation model but this then only allows for partially explaining the system behaviour. Equally, if the second assumption is not satisfied, we can also still extract an explanation model, but it most likely would not be correct. With such a faulty explanation model, we could then apply debugging methods to actually eliminate incorrect behaviour from the system model itself through faulty explanations. However, for our example, we assume that both assumptions hold to extract a correct explanation model.

For now, we focus on timed automata system models. For other types of system models, the extraction process may differ, as we discuss in Sect. 8.

For the extraction, we readopt the idea of [24] to connect system actions with their reasons. However, we broaden the idea of actions to *observables*. Such an observable may be any behaviour that can be observed from the outside of the system. We assume that a communication action or setting an internal

variable to a new value is also an observable, as, e.g., an engineer or another system might be interested to keep track of the valuation of certain system variables. Possible *reasons* basically are the preconditions for an observable. Note that the term “observable” does not only comprise those elements that are “visible” for a human. Instead, an observable in our sense is some observable system behaviour, which could, e.g., be identified via observer automata, as they are generally used for timed automata.

To retrieve the reasons for an observable, a backwards search through our system model can find all possible explanation paths that may lead to an observable. On assembling these explanation paths to one model, we get a causal tree structure that is structurally comparable to the causal diagram that has been described in [24]. The merits of such a tree structure are that on observing a certain phenomenon, the potential reasons and explanations can be directly extracted from this causal tree structure. However, we discuss potential limits of using a causal structure in Sect. 8.

Running Example: In our running example, we assume that our explanation purpose is to explain unexpected behaviour of an AV at an intersection, and that the explainee is a passenger of the AV.

To connect observables with their reasons, we exploit the extended timed automata semantics of our crossing protocol and connect *actions* in our timed automaton model with all their possible *reasons* (cf. Sect. 4). This idea of connecting actions with their reasons (resp. causes) stems from the “models of causality” approach [8]. A specific example for an action that we use in our protocol is the communication action $prio![p_E]$, with which the *ego* car communicates its priority p_E for turning at the intersection (cf. Fig. 5). A guard (i.e. reason) preceding this action is the function $cr_ahead()$. An example of an explanation path would connect the action $prio![p_{ego}]$ with the reason $cr_ahead()$.

Note that for this example we do not include clock reset actions $x := 0$ for a clock variable x into our notion of observables. This is because we assume that resetting of clock variables is an internal concept of timed automata. The explanation model that we extract by connecting actions with their reasons is depicted in Fig. 6. Note that this first version EM1 contains reasons for all observables. Also note that the superstates which we use in Fig. 6 are only used to simplify the figure: Their meaning is that, e.g., both the observables $count_a$ and $start()$ have the same reasons.

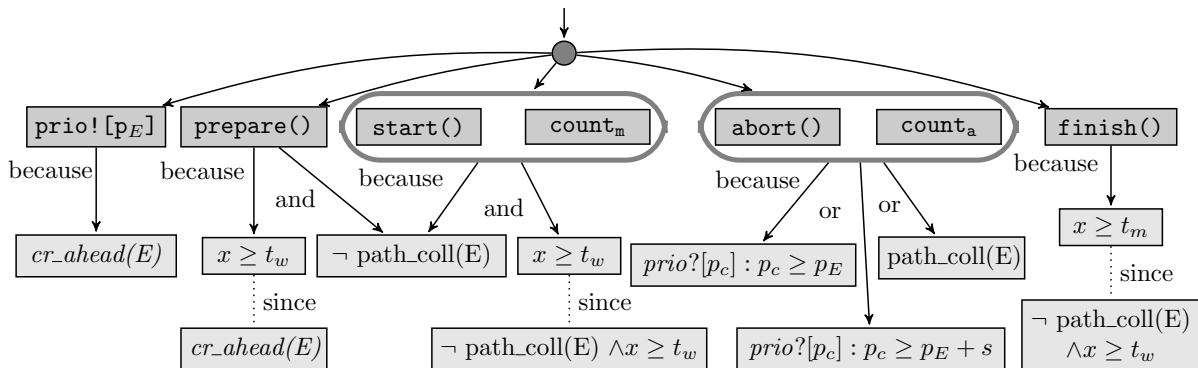


Figure 6: Explanation model EM1 connects all possible actions of the system model SysModel with their possible reasons.

5.2 Preparation

In our second step of phase 1, we hide context details that are irrelevant for our explanation purpose. Thus, we reduce the content of our explanation model EM1 to behaviour that actually fits the explanation purpose within EM2.

In our meaning, *relevant information* is such information that helps the explainee to better understand system behaviour and functionality, w.r.t. an explanation purpose. To formally capture this notion of *understanding explanations*, related approaches [3, 12] compare the actual world model W with a locally believed model M_i of an explainee i . A relevant explanation is one that helps in transforming a believed model M_i closer to the actual world model W . As an example for irrelevant information, we refer back to our factory robot from Sect. 3, p. 23. Note that we do not yet hide information that are irrelevant for specific explainee types in this step. For this we refer to Sect. 6.1.

To reduce the content of explanation model EM1, we hide entire branches: Those that contain actions that are not connected to our explanation purpose. However, we do not add or remove, nor alter in any way, information within any of the other branches. With this, the suggested procedure is a variation of program slicing, which was introduced in [25], where the explanation purpose is used as slicing criterion. For future work, we want to investigate more sophisticated slicing methods, were we do not necessarily hide entire branches in the explanation model.

Running Example: Consider again our running example, where we stated in Sect. 5.1 that our explainee is a passenger (cf. end-user) of an AV. We assume that internal system variables are not of interest for explaining visible behaviour of the AV to a passenger. Thus, we hide those actions that assign new values to data variables. I.e., we hide all branches that relate to the actions count_a and count_m . This means that the superstates containing multiple actions in EM1 are obsolete in EM2. The resulting explanation model EM2 is depicted in Fig.7.

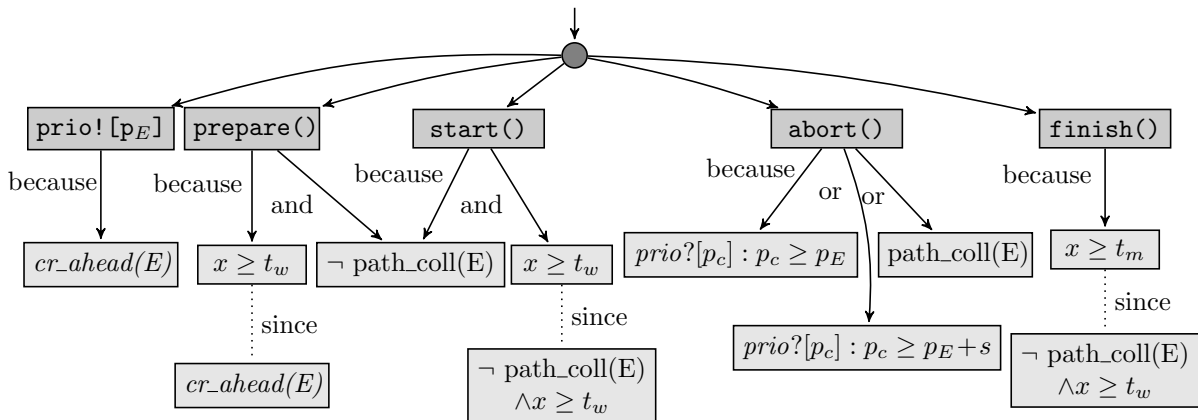


Figure 7: In explanation model EM2, branches that are related to actions out of the scope of the explanation purpose are removed from EM1 using a variation of slicing.

6 Phase 2: Tailoring and Refinement

This second phase of our framework from Fig. 2 contains again two steps, which we again describe separately: First (Sect. 6.1, the explanation model is tailored towards specific explainees and then (Sect. 6.2) context information is added. After this phase, the explanation model may be deployed. Note that we again explain our concepts by using our running example.

6.1 Tailoring Towards Explainee

Recall that before, in Sect. 5.2, we suggested to hide information that are irrelevant *w.r.t. an explanation purpose*. In this step, we use another notion of relevance: Relevance *w.r.t. a specific explainee type*.

Thus, the goal of this step is to tailor the explanation model EM2 towards a specific type of explainee, e.g. to an end-user, an engineer or another system. For this, explainee-specific details of the model may be eliminated. The intuition is that an end-user may not be interested in internal data variables, while an engineer might very well need explanations for values of such variables. We again suggest to use slicing methods to reduce the explanation model EM2 to a user-type-specific explanation model EM3. We again hide entire branches that contain reasons for actions out of the scope for a specific explainee.

Running Example: We continue our case-study, where we consider that our explainee is an end-user and that the explanation purpose is to explain the behaviour of the crossing controller from Fig. 5. We assume that an end-user will only be interested in explanations for visible behaviour of the AV, not perceiving the invisible behaviour. Thus, we omit the explanation branch for the action $\text{prio!}[p_E]$, where the AV communicates its priority to other traffic participants. Further on, we assume that both the actions $\text{prepare}()$ and $\text{finish}()$ are not of interest for an end-user: the manoeuvre preparation comprises an internal system state, which is not visible to an end-user and while finishing the manoeuvre is visible (“leaving the intersection”), from the perspective of an end-user, this might not be perceived as an action that needs to be explained.

Thus, the resulting explanation model EM3 that we depict in Fig. 8 only comprises the explanation branches for the actions $\text{start}()$ and $\text{abort}()$. Especially the latter one is of interest, as it is a version of a “why not” explanation (“Why do we not start the manoeuvre?”). Such explanations have been found to be of interest in [19].

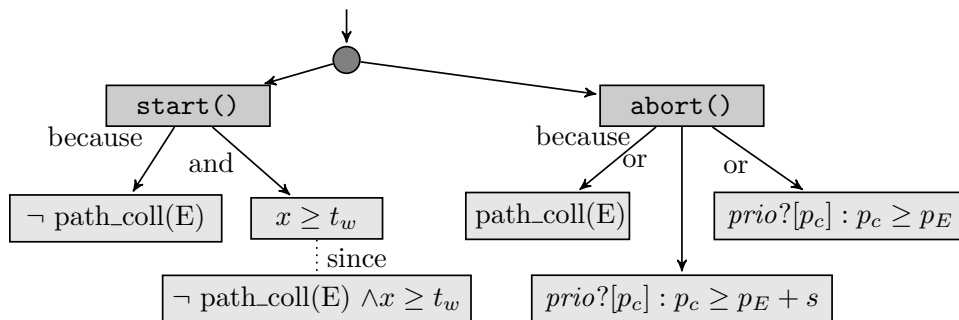


Figure 8: For tailoring explanation model EM2 towards an end-user as explainee, invisible, internal system actions are omitted, resulting in a smaller explanation model EM3.

6.2 Refinement: Add environmental information

In this step, the explanation model EM3 is extended by explainee-specific environmental information. The intuition is that some of the information that is needed for a good explanation might not be contained within the system model SysModel. Such information could be rules that the system abides by that are only encoded by abstract data variables within SysModel. These information can be obtained by different sources: for example, from an engineer or from system requirements. If the addressee is an end-user or non-expert, we also suggest that abstract system data is annotated with natural language text snippets in this step. Note that these text snippets are not yet fragments of a formulated explanation that is provided to an addressee but should rather be understood as modules from which an explanation may be retrieved later on. We further clarify this progress from EM3(X) to EM4(X) with our running example in the following.

Running Example: In the case of our end-user, we suggest to annotate EM3 with text snippets that can later be used to formulate explanations. For instance, the abstract clock variable x and it exceeding some constant t_w has no meaning to an end-user. However, if the right-hand branch of the action `start()` is summarised by “manoeuvre time exceeded”, this text snippet might be integrated into an explanation later on. Further on, consider the reasons for `abort()`: A special case is the reason where the priority p_c of another AV exceeds the priority p_E of our AV E under consideration *significantly* (by a constant value “s”). While this value “s” is abstract and without any meaning for an end-user, its meaning reflects the traffic rule where an emergency vehicle has the right of way at an intersection. This again is a user-understandable, and moreover, user-acceptable information.

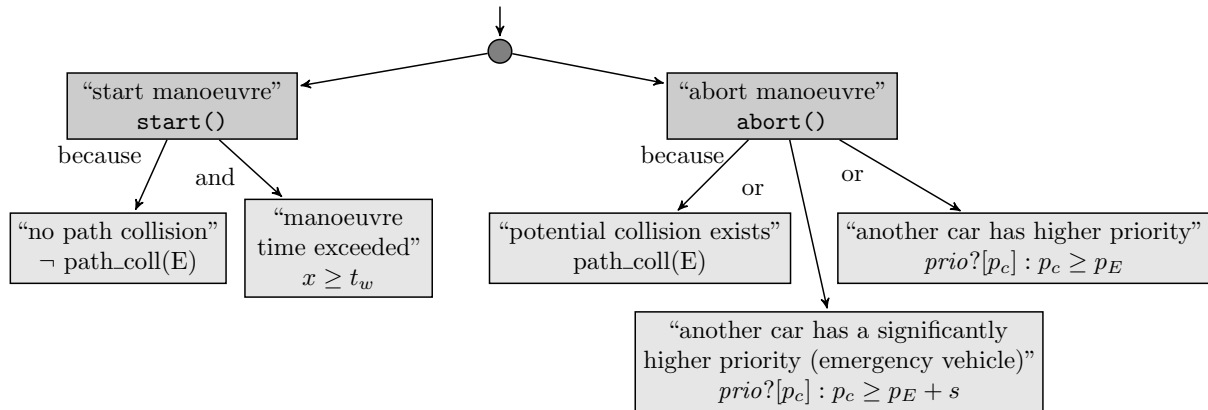


Figure 9: Explanation model EM4 is enriched by natural language annotations to simplify the generation of explanations from this model later on. Further on, information about traffic rules, for instance that emergency vehicles have a right of way (“having the highest priority”) is reflected within EM4.

7 Phase 3: Run-time Adaptation

We consider the explanation model EM4(X) from the previous phase to be ready for deployment for the explainee type X . However, we postulate that an explanation model should be individualised towards a specific explainee $x \in X$. Thus, on deploying the explanation model, we consider an initial explanation

model $EM5(x_1)$ with $EM5(x_1) \sim EM4(X)$ for an individual explainee $x_1 \in X$. For different explainees x_1, x_2 , a further individualisation or personalisation of the initial models $EM5(x_1)$, $EM5(x_2)$ is necessary as different explainees have different explanation preferences, prior system knowledge and attention levels that may influence their cognitive workload. Such different attention levels have, e.g., been identified for air-plane pilots within the *Saliency, Effort, Expectancy and Value (SEEV)* model in [27]. Also, users gain experience with the system during its lifetime and might want to hide specific (types of) explanations on demand. Equally, a run-time adaptation of $EM5(x_1)$ will be necessary, whenever the system model itself changes, e.g. due to self-adaptation, learning or software updates. We identify possibilities to detect the need for model updates in the following and leave the techniques to update the explanation model for future work.

System Update-triggered Adaptations.

The need for model updates of $EM5(x_1)$ due to an updated system model can be automatically identified if a corresponding update process for the system model exists. With this updated model, we can start the extraction and refinement process again. To do this automatically at run-time, all steps need to be fully automated. However, to reduce the computation overhead, we plan to investigate an incremental approach that adds or removes parts of the explanation models in future work. This requires a meta model for the explanation model that defines operations with which the explanation model can be adjusted. This could, for example, be realised by graph transformation rules.

During run-time, the explanation layer could also detect the need for updating the system model when an explanation cannot be extracted from the existing explanation model $EM5(X)$, e.g. due to a novel situation that the system encounters.

User-centered Adaptations.

We identify several possibilities to detect the need for those updates that are necessary for the further individualisation towards the explainee x_1 :

- The explainee x_1 requests a more detailed explanation that cannot be extracted from the existing explanation model $EM5(X)$
- The explainee x_1 requests to hide a specific explanation
- Through observation of the user's behaviour, the system detects that they have gained experience with the system and do not need detailed explanations for known situations.

The first case is connected to findings that “explaining” is not a static process, where an explanation is given only once. Instead, related work [17] suggests that not only one isolated explanation, but instead an *explanation process* should be considered. In such a process, the system may start with a brief explanation and refine this explanation later, if the user is not initially satisfied with it. For this, a differently detailed explanation model might be needed, for which the different phases of our approach are ideal. We will consider such dynamic explanation processes in future work. To enable adjustments for specific explainees at run-time, we assume that the explainee can give feedback for each explanation, e.g., by rating whether the explanation was helpful and whether the level of detail was adequate.

8 Discussion and Future Work

In this paper, we introduce an extraction and refinement process for explanation models that allows making existing systems self-explainable by using an additional explanation layer like it is done in our MAB-EX framework. We showcase the usability of our approach with a timed automata running example. Nonetheless, our process is only the first stepping stone towards an automatic, generalised process of deriving explanations directly from system models. Here, we discuss and assess the main strengths of our approach, but also name its limits and potential topics for future work.

Running Example and Generality.

We have chosen a timed automaton specification (SysModel) for our running example, as timed automata allow to formalise a good variety of system types. Also, for our specific example, a formal semantics and proofs for system properties like safety, already exist in [6, 23]. With our running example, we enrich [6] by an explainability module. In future work, we plan to examine whether, for timed automata, some state history should also be integrated into the explanation model EM1.

Note that our extraction and refinement process from Fig.2 itself is not limited to timed automata. Indeed, our general framework is capable of retrieving explanation models from arbitrary types of system models. To use our approach for other types of systems, we currently identify an important additional requirement for the system model SysModel (apart from those that were introduced in the beginning of Sect. 5.1): It must be possible to identify observables (“actions”) and reasons within the system. We intend to examine whether this additional requirement indeed is enough with further case-studies. Also, a generalised definition for the explanation model extraction process for causality-based system models should be provided.

Automation.

Most steps of our approach do not require human intervention and could be done automatically. In future work, we thus aim to formally define the requirements for all steps of our process and to implement them to automatically extract an explanation model from a system model. To this end, we will also investigate elaborate slicing techniques to tailor the extracted explanation model to different stakeholders. Only for the tailoring of EM3(X) to EM4(X), we suggest to involve an expert, e.g. to connect internal system data to rules. For future work, we intend to examine whether such information could also be automatically retrieved from other sources. While currently, we only use the system model SysModel for constructing our explanation model, one could also exploit other available sources from the entire system development process: For instance, a requirements document or system code and its code documentation might be of help to annotate abstract observables and reasons with text fragments (cf. as in Sect. 6.2).

Integration into MAB-EX.

Following the MAB-EX framework, the explanation model that we present here is not intended to generate actual explanations that would be presented to an end-user. Instead, we retrieve internal explanation paths from our explanation model in the *Build* phase. The next step would be to translate these internal explanation paths into actual explanations in the last MAB-EX phase. For this, e.g., [21] discusses translation problems between language and logical representations and a variety of approaches exists that discuss how to best present explanations.

Formal Models for Explainability.

Our explanation model is a formal model for explanations, as it is retrieved from a formal system model. With that, we have internal, formalised explanation paths that allow for a translation for different types of explainees, e.g. experts or other systems. Also, our explanation paths are concise, formal representations of explanations. With this, information loss due to natural language translation is minimised. Further on, we envision that we can actually formally analyse and prove certain properties of explanations in future work. For this, our notion of explanation paths would need to be integrated into a formal definition of explanations.

9 Conclusion

We present a high-level process for extracting and refining explanation models from formal system models, thus allowing to make an existing system self-explainable. From our explanation model, explanation paths may be automatically retrieved at run-time and then be translated into explainee-understandable explanations. A strength of our approach is that we take different types of explainees into account: e.g. an end-user, an engineer, or another system. Further on, our formal approach to explanations ensures that future formal analyses are possible. Also, not only the final explanation model $EM5(x_1)$ is of use for the process of explaining system behaviour. The different intermediate explanation models from each of our phases themselves show a different degree of detail which could be used to integrate these models into other approaches. We showcase our approach using a case-study from the automotive domain, where we extract and refine an explanation model from an extended timed automaton controller for turn manoeuvres at intersections. While our reference process is a first step towards a system capability of automatic self-explanation of system behaviour, we identify and sketch further tasks in our future work section.

References

- [1] (2005): *An Architectural Blueprint for Autonomic Computing*. White Paper, IBM.
- [2] Ankit Agrawal & Jane Cleland-Huang (2021): *Explaining Autonomous Decisions in Swarms of Human-on-the-Loop Small Unmanned Aerial Systems*. In: *Proceedings of the AAAI Conference on Human Computation and Crowdsourcing*, 9, pp. 15–26. Available at <https://ojs.aaai.org/index.php/HCOMP/article/view/18936>.
- [3] Astrid Rakow Akhila Bairy, Willem Hagemann & Maïke Schwammbberger (2022): *Towards formal concepts for explanation timing and justifications*. In: *2022 IEEE 30th International Requirements Engineering Conference Workshops (REW)*.
- [4] Rajeev Alur & David L. Dill (1994): *A Theory of Timed Automata*. *Theoretical Computer Science* 126(2), pp. 183–235, doi:10.1016/0304-3975(94)90010-8.
- [5] Sule Anjomshoae, Amro Najjar, Davide Calvaresi & Kary Främling (2019): *Explainable agents and robots: Results from a systematic literature review*. In: *18th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2019), Montreal, Canada, May 13–17, 2019*, International Foundation for Autonomous Agents and Multiagent Systems, pp. 1078–1088, doi:10.5555/3306127.3331806.
- [6] Christopher Bishopink & Maïke Schwammbberger (2019): *Verification of Fair Controllers for Urban Traffic Manoeuvres at Intersections*. In Emil Sekerinski, Nelma Moreira, José N. Oliveira, Daniel Ratiu, Riccardo Guidotti, Marie Farrell, Matt Luckcuck, Diego Marmosler, José Campos, Troy Astarte, Laure Gonnord, Antonio Cerone, Luis Couto, Brijesh Dongol, Martin Kutrib, Pedro Monteiro & David Delmas, editors: *Formal Methods. FM 2019 International Workshops - Porto, Portugal, October 7-11, 2019, Revised Selected*

- Papers, Part I, Lecture Notes in Computer Science* 12232, Springer, pp. 249–264, doi:10.1007/978-3-030-54994-7_18.
- [7] Mathias Blumreiter, Joel Greenyer, Francisco Javier Chiyah Garcia, Verena Klös, Maike Schwammberger, Christoph Sommer, Andreas Vogelsang & Andreas Wortmann (2019): *Towards Self-Explainable Cyber-Physical Systems*. In: *22nd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion*, pp. 543–548, doi:10.1109/MODELS-C.2019.00084.
- [8] Francisco Javier Chiyah Garcia, David A. Robb, Xingkun Liu, Atanas Laskov, Pedro Patron & Helen Hastie (2018): *Explainable Autonomy: A Study of Explanation Styles for Building Clear Mental Models*. In: *Proceedings of the 11th International Conference on Natural Language Generation*, Association for Computational Linguistics, Tilburg University, The Netherlands, pp. 99–108, doi:10.18653/v1/W18-6511. Available at <https://www.aclweb.org/anthology/W18-6511>.
- [9] Jeremy Dick, M. Elizabeth C. Hull & Ken Jackson (2017): *Requirements Engineering, 4th Edition*. Springer, doi:10.1007/978-3-319-61073-3.
- [10] Marie Farrell, Matt Luckcuck, Oisín Sheridan & Rosemary Monahan (2022): *FRETting About Requirements: Formalised Requirements for an Aircraft Engine Controller*. In Vincenzo Gervasi & Andreas Vogelsang, editors: *Requirements Engineering: Foundation for Software Quality - 28th International Working Conference, REFSQ 2022, Birmingham, UK, March 21-24, 2022, Proceedings, Lecture Notes in Computer Science* 13216, Springer, pp. 96–111, doi:10.1007/978-3-030-98464-9_9.
- [11] Randy Goebel, Ajay Chander, Katharina Holzinger, Freddy Lecue, Zeynep Akata, Simone Stumpf, Peter Kieseberg & Andreas Holzinger (2018): *Explainable AI: the new 42?* In: *International cross-domain conference for machine learning and knowledge extraction*, Springer, pp. 295–303, doi:10.1007/978-3-319-99740-7_21.
- [12] Martin Fränze Goerschwin Fey & Rolf Drechsler (2022): *Self-Explanation in Systems of Systems*. In: *2022 IEEE 30th International Requirements Engineering Conference Workshops (REW)*.
- [13] Joel Greenyer, Malte Lochau & Thomas Vogel (2019): *Explainable software for cyber-physical systems (es4cps): Report from the gi dagstuhl seminar 19023, january 06-11 2019, schloss dagstuhl*. arXiv:1904.11851, doi:10.48550/arXiv.1904.11851.
- [14] Andreas Holzinger, Anna Saranti, Christoph Molnar, Przemyslaw Biecek & Wojciech Samek (2020): *Explainable AI Methods - A Brief Overview*. In Andreas Holzinger, Randy Goebel, Ruth Fong, Taesup Moon, Klaus-Robert Müller & Wojciech Samek, editors: *xxAI - Beyond Explainable AI - International Workshop, Held in Conjunction with ICML 2020, July 18, 2020, Vienna, Austria, Revised and Extended Papers, Lecture Notes in Computer Science* 13200, Springer, pp. 13–38, doi:10.1007/978-3-031-04083-2_2.
- [15] Louise Dennis Joe Collenette & Michael Fisher (2022): *Advising Autonomous Cars about the Rules of the Road*. In: *Proceedings of the Fourth Workshop on Formal Methods for Autonomous Systems, FMAS@SEFM'22, 26th-27th of September 2022, EPTCS*.
- [16] Jeamin Koo, Jungsuk Kwac, Wendy Ju, Martin Steinert, Larry Leifer & Clifford Nass (2015): *Why did my car just do that? Explaining semi-autonomous driving actions to improve driver understanding, trust, and performance*. *International Journal on Interactive Design and Manufacturing (IJIDeM)* 9(4), pp. 269–275, doi:10.1007/s12008-014-0227-2.
- [17] Douglas S. Krull & Craig A. Anderson (1997): *The Process of Explanation*. *Current Directions in Psychological Science* 6(1), pp. 1–5, doi:10.1111/1467-8721.ep11512447.
- [18] Maximilian A. Köhl, Kevin Baum, Markus Langer, Daniel Oster, Timo Speith & Dimitri Bohlender (2019): *Explainability as a Non-Functional Requirement*. In: *2019 IEEE 27th International Requirements Engineering Conference (RE)*, pp. 363–368, doi:10.1109/RE.2019.00046.
- [19] Brian Y. Lim, Anind K. Dey & Daniel Avrahami (2009): *Why and Why Not Explanations Improve the Intelligibility of Context-Aware Intelligent Systems*. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI '09, Association for Computing Machinery, New York, NY, USA*, p. 2119–2128, doi:10.1145/1518701.1519023.

- [20] Swantje Plambeck, Görschwin Fey, Jakob Schyga, Johannes Hinckeldeyn & Jochen Kreutzfeldt (2022): *Explaining Cyber-Physical Systems Using Decision Trees*. In: *2nd International Workshop on Computation-Aware Algorithmic Design for Cyber-Physical Systems (CAADCPS)*, doi:10.1109/CAADCPS56132.2022.00006. Available at <https://conferences.computer.org/psi/psiot/pdfs/CAADCPS2022-5TICzWIXsIbzy5ctgEfHPL/820100a003/820100a003.pdf>.
- [21] Aarne Ranta (2011): *Translating between Language and Logic: What Is Easy and What Is Difficult*. In Nikolaj Björner & Viorica Sofronie-Stokkermans, editors: *Automated Deduction - CADE-23 - 23rd International Conference on Automated Deduction, Wrocław, Poland, July 31 - August 5, 2011. Proceedings, Lecture Notes in Computer Science 6803*, Springer, pp. 5–25, doi:10.1007/978-3-642-22438-6_3.
- [22] Mersedeh Sadeghi, Verena Klös & Andreas Vogelsang (2021): *Cases for Explainable Software Systems: Characteristics and Examples*. In: *IEEE 29th International Requirements Engineering Conference Workshops (REW)*, pp. 181–187, doi:10.1109/REW53955.2021.00033.
- [23] Maïke Schwammbberger (2018): *An abstract model for proving safety of autonomous urban traffic*. *Theoretical Computer Science* 744, pp. 143–169, doi:10.1016/j.tcs.2018.05.028.
- [24] Maïke Schwammbberger (2021): *A Quest of Self-Explainability: When Causal Diagrams meet Autonomous Urban Traffic Manoeuvres*. In: *2021 IEEE 29th International Requirements Engineering Conference Workshops (REW)*, pp. 195–199, doi:10.1109/REW53955.2021.00035.
- [25] Mark Weiser (1981): *Program Slicing*. In: *Proceedings of the 5th International Conference on Software Engineering, ICSE '81*, IEEE Press, p. 439–449, doi:10.1109/TSE.1984.5010248.
- [26] Danny Weyns, Jesper Andersson, Mauro Caporuscio, Francesco Flammini, Andreas Kerren & Welf Löwe (2021): *A Research Agenda for Smarter Cyber-Physical Systems*. *J. Integr. Des. Process Sci.* 25(2), p. 27–47, doi:10.3233/JID210010.
- [27] Christopher D. Wickens, John Helleberg, Juliana Goh, Xidong Xu & William J. Horrey (2001): *Pilot Task Management : Testing an Attentional Expected Value Model of Visual Scanning*. Available at <http://apps.usd.edu/coglab/schieber/psyc792/workload/Wickens-etal-2001.pdf>.
- [28] Florian Ziesche, Verena Klös & Sabine Glesner (2021): *Anomaly Detection and Classification to enable Self-Explainability of Autonomous Systems*. In: *Design, Automation Test in Europe Conference Exhibition (DATE)*, pp. 1304–1309, doi:10.23919/DATE51398.2021.9474232.