Characterization of Failure Effects on AADL Models*

Bernhard Ern[1], Viet Yen Nguyen[2], and Thomas Noll[2]

[1] Next Level Integration GmbH, Cologne, Germany; bern@next-level-integration.com
[2] RWTH Aachen University, Germany; {nguyen,noll}@cs.rwth-aachen.de

Abstract. Prior works on model-based Failure Modes and Effects Analysis (FMEA) automatically generate a FMEA table given the system model, a set of failure modes, and a set of possible effects. The last requirement is critical as bias may occur: since the considered failure effects are restricted to the anticipated ones, unexpected effects - the most interesting ones - are disregarded in the FMEA.
In this paper, we propose and investigate formal concepts that aim to overcome this bias. They support the construction of FMEA tables solely based on the system model and the failure modes, i.e., without requiring the set of effects as input. More concretely, given a system specification in the Architecture Analysis and Design Language (AADL), we show how to derive relations that characterize the effects of failures based on the state transition system of that specification. We also demonstrate the benefits and limitations of these concepts on a satellite case study.

1 Introduction

Safety and dependability assessments are imperative for the engineering of safety-critical systems. In particular, a clear understanding of how failures emerge, how they propagate and how these are dealt with is key towards trustworthy operation of the system itself. Methods such as Failure Modes and Effects Analysis (FMEA) and Fault Tree Analysis (FTA) contribute to this. These two techniques relate faults to failures and failures to effects. In current engineering practice, these analyses are conducted manually. With the advent of model-based engineering, the trend is towards more automated techniques. In [3], an approach is used that generates an FMEA table from a system model expressed in the Architecture and Analysis Design Language (AADL). The concepts described in this paper improve upon it by reducing bias and allow for a more explorative analysis of failures and their effects.

This paper is organized as follows. Section 2 introduces our modeling language, a dialect of AADL. Then Section 3 describes and explains *effect relations* and *effect matrices*, which are our primary contributions. In Section 4, our prototypical tool is explained and our concepts are validated on a satellite case study based on [7]. Sections 5, 6 and 7 respectively deal with related work, future work, and conclusions.

* This work was partially supported by ESA/ESTEC (contract no. 4000100798) and Thales Alenia Space (contract no. 1520014509/01).

2 Preliminaries

The Architecture Analysis and Design Language (AADL) is an industry standard for modeling safety-critical system architectures, and is designed and governed by the Society of Automotive Engineers (SAE) [1]. The AADL dialect that we are using in this paper, just referred to as AADL in the following, has been developed within a project entitled Correctness, Modeling and Performance of Aerospace Systems (COMPASS) that was funded by the European Space Agency (ESA) [4]. It provides a cohesive and uniform approach to model heterogeneous systems, consisting of software and hardware components, and their interactions. Furthermore, it has been drafted and enhanced with the following essential features in mind:

- Modeling both the system's nominal and faulty behavior. To this aim, primitives are provided to describe software and hardware faults, error propagation (that is, turning fault occurrences into failure events), sporadic (transient) and persistent faults, and degraded modes of operation (by mapping failures from architectural to service level).
- Modeling (partial) observability and the associated observability requirements. These notions are essential to deal with diagnosability and Fault Detection, Isolation and Recovery (FDIR) analyses.
- Specifying timed and hybrid behavior. In particular, in order to analyze physical systems with non-discrete behavior, such as mechanics and hydraulics, the modeling language supports continuous real-valued variables with (linear) time-dependent dynamics.
- Modeling probabilistic aspects, such as random faults, repairs, and stochastic timing.

A complete AADL specification consists of three parts, namely a description of the nominal behavior, a description of the error behavior and a fault injection specification that describes how the error behavior influences the nominal behavior. These three parts are discussed in order below.

Nominal Behavior. The system model is hierarchically organized into components, distinguished into software (processes, threads, data), hardware (processors, memories, devices, buses), and composite components (called systems). Components are defined by their type and their implementations.

The component type specifies the ports through which the component communicates with its environment. There are two kinds of ports, namely event and data ports. Event ports enable components to synchronize their state upon each other whereas data ports are used to expose component variables to neighboring components.

A component implementation (such as the one given in Listing 1) describes the internal structure of a component through the definition of its subcomponents (lines 2–3), their interaction through (event and data) port connections, the (physical) bindings at runtime and the behavior via modes (lines 4–6) and transitions (lines 7–10) between them. This behavior, which basically is a finite

state automaton, describes how the component evolves from mode to mode while being triggered by events, or by spontaneously triggering events at the ports. Upon a transition, data components (like integer, real and Boolean variables) may change values due to transition assignments. Modes can be further annotated with invariants (e.g. the expressions after **while** on lines 5–6) on the value of data components (continuous or clock variables), restricting for example residence time. They furthermore may contain trajectory equations, specifying how continuous variables evolve while residing in a mode. (Here `energy'` refers to the first derivative of the energy value.) This is akin to timed and hybrid automata. Mode transitions may give rise to modifications of a component's configuration: subcomponents can become (de-)activated and port connections can be (de-)established. This depends on the "in modes" clause, which can be declared along with port connections and subcomponents.

Listing 1. An example battery component implementation.

```
1  system implementation Battery.Imp
2    subcomponents
3      energy: data continuous default 1.0;
4    modes
5      charged: activation mode while energy' = -0.02 and energy >= 0.2;
6      depleted: mode while energy' = -0.03 and energy >= 0.0;
7    transitions
8      charged -[then voltage := 2.0 * energy + 4.0]-> charged;
9      charged -[empty when energy = 0.2]-> depleted;
10     depleted -[then voltage := 2.0 * energy + 4.0]-> depleted;
11 end Battery.Imp;
```

Error Behavior. Error models are an extension to the specification of nominal models and are used to conduct safety, dependability and performability analyses. For modularity, they are defined separately from nominal specifications. Akin to nominal models, an error model is defined by its type and its associated implementations.

An error model type defines an interface in terms of error states and (incoming and outgoing) error propagations. Error states are employed to represent the current configuration of the component with respect to the occurrence of errors. Error propagations are used to exchange error information between components.

An error model implementation (such as the one given in Listing 2) provides the structural details of the error model. It defines a (probabilistic) machine over the error states declared in the error model type. Transitions between states (lines 6–10) can be triggered by error events (lines 2–5), reset events, and error propagations. Error events are internal to the component; they reflect changes of the error state caused by local faults and repair operations, and they can be annotated with occurrence distributions to express probabilistic error behavior. Moreover, reset events can be sent from the nominal model to the error model of the same component, trying to repair a fault that has occurred. Whether or not

such a reset operation is successful has to be modeled in the error implementation by defining (or respectively omitting) corresponding state transitions. Outgoing error propagations report an error state to other components. If their error states are affected, the other components will have a corresponding incoming propagation.

Listing 2. An example battery error model implementation.

```
error model implementation BatteryFailure.Imp
  events
    die: error event occurrence poisson 0.001;
    works: error event occurrence poisson 0.2;
    fails: error event occurrence poisson 0.8;
  transitions
    ok -[die]-> dead;
    dead -[reset]-> resetting;
    resetting -[works]-> ok;
    resetting -[fails]-> dead;
end BatteryFailure.Imp;
```

Fault Injections. As error models bear no relation with nominal models, an error model does not influence the nominal model unless they are linked through fault injections. They describe the effect of the occurrence of an error on the nominal behavior of the system. More concretely, a fault injection specifies the value update that a data element of a component implementation undergoes when its associated error model enters a specific error state.

Multiple fault injections between error models and nominal models are possible. An automatic procedure, the so-called model extension, is employed to integrate both models and the given fault injections. It yields a combined specification that represents both the nominal and the faulty behavior of the system. Its semantics is formally defined by a transition system (cf. Definition 1) whose states are determined by the current modes and error states of all components, together with the current values of their data elements. Its transitions are derived from both the (nominal) mode and the (faulty) error state transitions, attaching a unique *transition label* to each. The latter can be used by the subsequent analyses, such as the failure effect analysis which is described in the present paper, to distinguish different types of transitions. More details on the specification language and its formal semantics can be found in [3].

3 Characterizing Effects

The two main contributions of this paper are described in this section, namely *effect relations* and *effect matrices*. A simple algorithm shall be sketched for computing these.

3.1 Effect Relations

We determine effect relations over a formal model of an AADL specification called a *transition system*, which captures all its possible behaviors. It is defined as follows:

Definition 1 (Transition System). *A* transition system *is a tuple* $\langle S, T, \rightarrow, I \rangle$ *where*

- S *is a set of states,*
- T *is a set of transition labels,*
- $\rightarrow \subseteq S \times T \times S$ *is the transition relation, and*
- $I \in S$ *is the initial state.*

Here the set T contains AADL transition labels. These are the transitions occurring *syntactically* in the AADL model. For example, line 8 in Listing 1 yields one transition label, as well as lines 9 and 10 in the same example. Also transitions in the error model are part of T, e.g. each of lines 7–10 in Listing 2.

Notationwise, we use several shorthands. A transition $\langle s, t, s' \rangle \in \rightarrow$ is also noted as $s \xrightarrow{t} s'$. It means that the AADL transition (labelled as $t \in T$) was applied to s to reach state s'. Also, we use $t(s)$ which, given $t \in T$ and $s \in S$, is defined as $t(s) = s'$ if $s \xrightarrow{t} s'$. Otherwise $t(s) = \bot$, i.e. undefined, where that $t(\bot) = \bot$. This shorthand assumes that the transition system is deterministic with respect to the set of labels T. This assumption holds for AADL, since non-deterministic behavior cannot be expressed in a *single* AADL transition. Furthermore, we describe the set of active transitions $A(s)$ as $\{t \in T \mid \exists s' : s \xrightarrow{t} s'\}$ (assuming that $A(\bot) = \emptyset$). Now we can define a binary effect relation over transitions in T:

Definition 2 (Effect Relation). *Given* $s \in S$ *and* $t_1, t_2 \in T$, t_1 *and* t_2 *can be in relation* Independent *(*$\|$*),* Dependent *(*$\nparallel$*),* Conflict *(*$\#$*) or* Enable *(*$\lhd$*). These relations are defined as*

- $t_1 \parallel t_2$ *if* $t_1(t_2(s)) = t_2(t_1(s))$ *and* $t_1(t_2(s)) \neq \bot$.
- $t_1 \nparallel t_2$ *if* $t_1(t_2(s)) \neq t_2(t_1(s))$ *and* $t_1(t_2(s)) \neq \bot$ *and* $t_2(t_1(s)) \neq \bot$.
- $t_1 \# t_2$ *if* $t_1, t_2 \in A(s)$ *and* $t_2 \notin A(t_1(s))$.
- $t_1 \lhd t_2$ *if* $t_2 \notin A(s)$ *and* $t_2 \in A(t_1(s))$.

Otherwise, the transitions are unrelated in s. *For* $\bowtie \in \{\|, \nparallel, \#, \lhd\}$, *the notation* $s \vDash t_1 \bowtie t_2$ *means that in state* s, *the transitions* t_1 *and* t_2 *are in relation* $\bowtie$.

The effect relations are depicted in Figure 1. Intuitively, the independency relation indicates that the transitions do not affect each other. Observe that this relation is symmetric, i.e. $t_1 \parallel t_2$ if and only if $t_2 \parallel t_1$. The dependency relation means that the related transitions affect each other, i.e. that changing the order in which they occur results in different states. Also this relation is symmetric. The conflict relation means that one transition disables another. Thus when $t_1 \# t_2$, transition t_2 is disabled and thus inactive after execution of transition t_1. The enable relation is of most interest to our application, namely characterizing failure

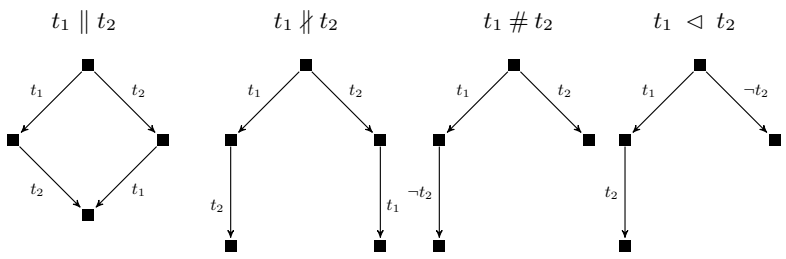


Fig. 1. The four effect relations $\parallel, \nparallel, \#$ and $\lhd$.

effects. This relation intuitively expresses that one transition can cause another to happen. Hence when $t_1 \lhd t_2$, it means that t_2 is caused by t_1. Both the conflict and the enable relations are generally non-symmetric.

Until now, the effect relation is defined for unsynchronized transitions. In our AADL dialect, transitions can however synchronize. The above definitions can easily be lifted to support this by generalizing the definition of a transition relation to the type $\rightarrow \subseteq S \times T \times (T \cup \{\epsilon\}) \times S$. Here $s \xrightarrow{t,\bar{t}} s'$ means that $\bar{t}$ is the rendez-vous transition interacting with t. In case t is unsynchronized, $\bar{t} = \epsilon$. The definition for effect relations are then lifted similarly as well. Due to page limit constraints, we refer the reader to [6] for the lifted definition. In the remainder of this paper we concentrate on unsynchronized transitions, since this keeps the notations simpler. We will bear in mind that all the techniques described in the following can easily support synchronized transitions by lifting definitions.

Furthermore, it is important that no transitions are left unnoticed in the effect relation, meaning that in any state s, any pair of transitions $t_1 \in A(s)$ and $t_2 \in A(s) \cup A(t_1(s))$ are related by exactly one of the independency, dependency, conflict or enable relations. The proof of these results is given in [6], and the interested reader is referred to this publication due to lack of space.

3.2 Effect Matrix

Given the definition for effect relations, we now determine how one transition affects another. This can be done by traversing the transition system and, in each state, determining which effect relation applies to all pairs of active transitions. The results are then accumulated in a matrix which we call the effect matrix.

Definition 3 (Effect Matrix). *Given a transition system* $\langle S, T, \rightarrow, I \rangle$ *with* $T = \{t_1, \ldots, t_n\}$*, the* effect matrix $M^{\bowtie}$ *for relation* $\bowtie \in \{\parallel, \nparallel, \#, \lhd\}$ *is given by:*

$$M^{\bowtie} := (m_{ij}^{\bowtie})_{1 \leq i,j \leq n} \quad \textit{where} \quad m_{ij}^{\bowtie} = \{s \in S \mid t_i \neq t_j \wedge s \vDash t_i \bowtie t_j\}$$

A simple procedure to compute $M^{\bowtie}$ given a transition system is shown in Algorithm 1. The time complexity for constructing $M^{\bowtie}$ is polynomial in the number of states $|S|$ and transitions $|T|$. This (rather naive) algorithm traverses

all states in S (line 2) and then determines for all pairs of active transitions in two transition steps (lines 3 and 4) how they are related (line 5). This is the algorithm we implemented. In the worst case, all transitions in T are always active, resulting in a time complexity of $O(|S| \times |T|^2)$. However, in practice only a small fraction of the transitions T are active, making the size of the state space $|S|$ the most important factor. The set of active transitions $A(s)$ is pre-computed during state space generation, and hence does not add additional costs.

Algorithm 1 Compute effect matrix $M^{\bowtie}$.

Input: $\langle S, T, \rightarrow, I \rangle$
Output: $M^{\bowtie}$
1: $m_{ij} := \emptyset \quad \forall t_i, t_j \in T$
2: **for all** $s \in S$ **do**
3: **for all** $t_i \in A(s)$ **do**
4: **for all** $t_j \in A(s) \cup A(t_i(s))$ **do**
5: **if** $s \vDash t_i \bowtie t_j$ **then**
6: $m_{ij}^{\bowtie} := m_{ij}^{\bowtie} \cup \{s\}$

The effect matrix is key to estimating effects, and failure effects in particular. Especially the enable relation ($\lhd$) is of importance for that. If a pair of transitions *only* occurs as $t_1 \lhd t_2$ in the state space, then t_1 and t_2 can be considered as *causally* related. Typically, t_1 would be an error transition originating in the AADL error model and t_2 would be a change to the nominal data ports or subcomponents as consequence to the error. This is demonstrated in the next section, where we report on our case study.

4 Experimental Evaluation

In the following, we detail on our implementation of the concepts introduced in Section 3 and demonstrate their usefulness on an adaptation of the satellite case study described in [7].

4.1 Tool

We developed a prototypical tool in Java that takes an AADL model as input, and then computes the effect matrix (see Definition 3 and Algorithm 1) as result. Inside the tool, it uses our AADL-to-Promela translator [11] to have the SPIN model checker [10] generate the transitions system. This transition system, in explicit-state representation, is dumped to disk as a file. All computations are then performed on that state space.

4.2 Case Study

We computed effect matrices for adaptations of a satellite case study as described in [7]. The adaptation reduces the original case to only cover the Attitude &

Orbit Control System (AOCS) and its fault management system, making its size (16008 states) more amenable to validating our approach. The AOCS is a control system responsible for maintaining the satellite's orientation. It is equipped with sensors (e.g. gyroscopes, Earth trackers, Sun trackers, magnetometers) to determine its current orientation. It uses actuators (e.g. reaction wheels, magnetic torquers, thrusters) to modify its orientation if required. These components may however degrade and/or fail due to space hazards, in particular radiation. So during the design of the AOCS, fault-tolerant aspects need to be incorporated. For example, components are redundantly equipped and failure compensation techniques are integrated. In our case study, we are in particular interested in the failure detection, isolation and recovery (FDIR) procedures. These procedures continuously monitor the system for non-nominal phenomena. If one is detected, FDIR attempts to determine its cause and subsequently initiates a corresponding (recovery) procedure. A minimal downtime is desired while doing so.

For demonstrating and validating our approach, we focus on one of the recovery procedures, namely the one that handles Earth tracking sensor failures. These sensors may for example fail to provide signals to the remainder of the AOCS. If this situation is left unhandled, it may cause the system to attain an incorrect orientation. The involved AOCS subsystems are

- Primary Earth Sensor (ES_A)
- Secondary Earth Sensor (ES_B)
- Control and Data Unit (CDU), containing
  - Processor Module (PM)
  - On-Board Data Handling software (OBDH), containing
    - Earth Sensors Control Software (ES_CTRL)
    - Earth Sensors Failure Detection, Isolation and Recovery Software (ES_FDIR)

There are many more components present, but we omit them here. Furthermore, the satellite may reside in particular modes of operation. In normal conditions, it is in the Nominal Mode. Upon failures, it is expected to switch to the Degraded Mode. If the recovery is successful, the satellite is expected to return to Nominal Mode. If not, it should switch to Safe Mode, in which ground control takes further action. There is also an Orbit Control Mode during which the satellite performs trajectory corrections. While switching to different modes, the topology of the system is reconfigured by disabling/enabling components and rerouting data and command streams from disabled to enabled components.

A particular recovery procedure consists of the following sequence of transitions (indicated by t_i), starting with error transition t_1 that causes the failure:

t_1**:** Injection of signal loss error in primary Earth sensor.
t_2**:** ES_FDIR detects signal loss.
t_3**:** ES_FDIR changes AOCS mode from Nominal to Degraded.
t_4**:** ES_FDIR isolates the failure and initiates switch-over of Earth sensors.
t_5**:** ES_CTRL disables primary Earth sensor.
t_6**:** ES_A sets its status flag to off.

t_7: ES_CTRL enables secondary Earth sensor.

t_8: ES_B checks its power and sets its status flag to on.

t_8: ES_FDIR changes AOCS mode from Degraded to Nominal.

This recovery procedure triggers transitions in manifold components (e.g. ES_FDIR, ES_CTRL, ES_A, ES_B), which are part of the AOCS, CDU, PM and OBDH subsystems. The satellite model captures all these interrelations. The Earth sensor FDIR component furthermore includes many more behaviors (and thus more transitions) to cover other scenarios, like the transition to Safe Mode after a signal loss of the secondary Earth sensor while being in Orbit Control Mode. Another example is the switch-over that is initiated when the primary Earth sensor has failed while being in that mode. This makes the FDIR component tightly coupled with a major part of the overall system. It is therefore imperative to understand which effects the FDIR component has on the system, and under which (isolated) conditions these effects apply. This aims to avoid undesired behaviors.

When we computed the effect matrix for a scenario in which the primary Earth sensor fails, we were able to determine the exact recovery procedure chain, namely that $t_1 \lhd t_2 \lhd \ldots \lhd t_8$. The effect matrix also showed that this order is strict, i.e., there are no cases where those transitions were executed in another order (e.g. $t_7 \lhd t_5$). This proves that the recovery procedure as implemented in the model is indeed restricted to the occurrence of the failure, namely transition t_1. The effect matrix can thus be distilled to a FMEA table entry where upon the occurrence of a primary Earth sensor signal loss, the effect is the chain of transitions from t_2 up to t_8. More entries can be distilled by computing effect matrices for the models with different fault configurations, like a secondary Earth sensor failure.

In the first runs on this case study, we loaded up models in which the primary Earth sensor failed while the system was in Nominal Mode. What we then expected to see in the effect matrix were the effects $t_1 \lhd \ldots \lhd t_8$. The effect matrix however also exhibited $t_4 \lhd t_{ocm}$ and $t_4 \lhd t_{eam}$, which involve the Telemetry Tracking & Control (TT&C) component that is used for communication with the satellite:

t_{ocm}: TT&C changes AOCS mode from Orbit Control to Nominal.

t_{eam}: TT&C changes AOCS mode from Earth Acquisition to Nominal.

The effects $t_4 \lhd t_{ocm}$ and $t_4 \lhd t_{eam}$ were unexpected to us because t_4 should only be occurring while being in Degraded Mode (due to transition t_3). Somehow, there was an interleaving in between that caused a transition to the Orbit Control Mode or to the Earth Acquisition Mode. The effect matrices furthermore showed that other transitions in the TT&C were enabled by other unexpected transitions. This indicated that our model did not handle the mutual exclusion of the TT&C with other components correctly. This was corrected in our final model.

4.3 Scalability

To obtain an impression of the scalability of our prototypical tool, we made a few adaptions of the satellite case study entailing varying sizes of the state space. The results are as follows:

Number of states	Generate state space [min]	Compute $M^{\bowtie}$ [sec]
10792	46	6
21584	86	14
31044	206	30

The column "Generate state space" gives the time needed for generating the state space by SPIN. The column "Compute $M^{\bowtie}$" lists the time needed for computing the effect matrix once the state space is generated. As we expected, the majority of time is necessary for generating the state space itself (which happens within an order of hours). Once this is accomplished, the state space is loaded into memory and is traversed in linear time. As shown in the table, this proceeds in an order of seconds. We also experimented with models beyond 31044 states, but for those our machine (2.1 GHz processor, 192 GB of RAM) runs out of memory during state space generation.

5 Related Work

Our concepts relate closely to approaches towards existing FMEA table generation techniques [3, 9]. With respect to [3], our approach overcomes the need to specify a set of *expected* failure effects as an input, and instead the effect matrix can be used to *discover* failure effects. This was the motivation of this work. With respect to [9], our approach uses the variant of AADL described in [3], which contrary to [9] incorporates behavioral aspects beyond the original AADL and its Error Model Annex. Furthermore, their approach appears (as details are scarce) to extract FMEA table entries from state space traces, and post-processes those entries using user-defined filters. Our effect matrix considers effect relations across multiple traces that globally hold in the state space, and hence is more fine-grained. Our concepts are not alternative to [3, 9], but rather complementary because of their ability to study finer effect relations.

Our approach borrows and incorporates concepts from existing works. The (in-)dependency relations are inspired by Mazurkiewicz trace equivalences [5], which capture both aspects. The conflict relation is inspired by the theory of event structures [12]. The enable relation is inspired by dynamic partial order reduction [8]. These concepts on their own are thus not novel, but their combination is, in particular when used for characterizing failure effects using an effect matrix.

6 Future Work: Entangled Effects

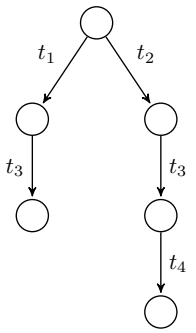
During our experimental evaluation, we observed failure effects whose faults are intertwined with other (not necessarily failure) effects. We call them *entangled effects*. We developed experimental approaches whose refinement and applicability require further investigation.

We observed that in the effect matrix, two transitions are not strictly in one relation throughout the state space. For example, it could occur that there exists a state s_k where $s_k \vDash t_i \lhd t_j$ while in another state s_l it holds that $s_l \vDash t_i \# t_j$. Thus, in one state one transition enables the other while in another state they conflict. With regard to the effect matrix, this means that both $m_{ij}^{\lhd}$ and $m_{ij}^{\#}$ are non-empty. To explain these deviating effects, we need to understand how the states s_k and s_l are related. We believe that constructing two paths, one leading to state s_l and the other to s_k, and then comparing them helps to understand the deviating effects. The two paths can be found by a backwards synchronized breadth-first search from s_k and s_l that proceeds until a common ancestor state is identified. Both paths, say $\pi_k = s_0, \ldots, s_k$ and $\pi_l = s_0, \ldots, s_l$, can then be sliced to emphasize the differences between the visited states. This may help for long paths or for states that are determined by many variables. The differences are computed by checking the valuations of variables between states.

Entanglement can also originate from effects caused by a *combination of faults*. Such a notion is useful for enhancing single-fault FMEA with multiple-failure configurations. In case of independent multiple failures, our concepts captures those as independent transitions which all enable a common effect.

A similar conceptualisation for multiple dependent failures is more involved. For this, we need to relate transitions by *transitivity*. In this paper, our definition of effect relations only considers pairs of transitions that directly follow each other in the state space. This led us to distill the effect relation of $t_1 \lhd \ldots \lhd t_8$ in our case study. However, this logically means by transitivity that $t_1 \lhd t_8$. The latter relation is however not detectable by our current definition.

We have been experimenting with transitive effect relation definitions in [6] to overcome this. However, we have not yet developed a suitable definition in which effect relations do not become *over-approximated*. For example, a transitive rule stating that $t_1 \lhd t_3 \wedge t_3 \lhd t_4$ implies that $t_1 \lhd t_4$ does not *generally* hold. The figure on the right exemplifies this situation. It shows that transitions t_1 and t_4 are in conflict rather than enabling each other. Hence we regard this topic as future work.



7 Conclusion

In this paper, we propose an approach for characterizing failure effects on AADL models by automatically detecting effect relations over the state space underlying the AADL model. This is of particular value to the automated generation of precise

FMEA tables, although not being limited to this application. Our approach is also not limited to AADL models, but is of a more general nature. An effect matrix can be computed for any model equipped with a formal semantics that describes its behavior by means of a labeled transition system. Furthermore, the effect matrix is not limited to characterizing failure effects. It also captures nominal effects, yet we believe it is of most applicable value to the safety and dependability domain. This is also demonstrated by our evaluation using a satellite case study.

Our work also revealed directions for future work. In particular, entangled effects through transitivity are of interest. These may for example occur due to failure propagations. There is also much room for improvement in the performance of our prototypical tool. Symbolic state representations like BDDs or SAT encodings [2] could be possible. The overall outcome so far demonstrates and validates our concepts that aid in constructing precise fault-failure effect relations (e.g. FMEA tables) as rigorously and automatically as possible.

References

[1] *Architecture, Analysis and Design Language*. AS5506. SAE, 2004.

[2] A. Biere, A. Cimatti, E.M. Clarke, and Y. Zhu. "Symbolic Model Checking without BDDs". In: *Proc. 5th Int. Conf. on Tools and Algorithms for Construction and Analysis of Systems (TACAS)*. Vol. 1579. 1999, 193–207.

[3] M. Bozzano, A. Cimatti, J.-P. Katoen, V.Y. Nguyen, T. Noll, and M. Roveri. "Safety, Dependability and Performance Analysis of Extended AADL Models". In: *Computer Journal* 54.5 (2011), pp. 754–775.

[4] *COMPASS Project*. `http://compass.informatik.rwth-aachen.de`.

[5] V. Diekert and G. Rozenberg, eds. *The Book of Traces*. World Scientific, Singapore, 1995.

[6] B. Ern. "Model-Based Criticality Analysis by Impact Isolation". `http://www-i2.informatik.rwth-aachen.de/dl/noll/theses/ern.pdf`. Master's thesis. RWTH Aachen University, 2012.

[7] M.-A. Esteve, J.-P. Katoen, V.Y. Nguyen, B. Postma, and Y. Yushtein. "Formal Correctness, Safety, Dependability and Performance Analysis of a Satellite". In: *Proc. 34th International Conference on Software Engineering (ICSE 2012)*. 2012, pp. 1022–1031.

[8] C. Flanagan and P. Godefroid. "Dynamic partial-order reduction for model checking software". In: *Proc. 32nd Symposium on Principles of Programming Languages (POPL 2005)*. 2005, pp. 110–121.

[9] M. Hecht, A. Lam, C. Vogl, and C. Dimpfl. *A Tool Set for Generation of Failure Modes and Effects Analyses from AADL Models*. Presentation at Systems and Software Technology Conference 2012. 2012.

[10] G.J. Holzmann. *The SPIN Model Checker: Primer and Reference Manual*. Addison-Wesley Professional, 2003.

[11] M.R. Odenbrett. "Explicit-State Model Checking of an Architectural Design Language using SPIN". Diplomarbeit. RWTH Aachen University, 2010.

[12] G. Winskel. "Events, Causality and Symmetry". In: *Computer Journal* 54.1 (2011), pp. 42–57.