SysML-based and Prolog-supported FMEA

Fabio Scippacercola*, Roberto Pietrantuono*, Stefano Russo*, Nuno Pedro Silva‡
*DIETI, Università degli Studi di Napoli Federico II
Via Claudio 21, 80125 Napoli, Italy
{fabio.scippacercola, roberto.pietrantuono, stefano.russo}@unina.it
‡Critical Software, SA
Parque Industrial de Taveiro, lote 49, 3045-504 Coimbra, Portugal
nsilva@criticalsoftware.com

***Abstract*—*Failure Mode and Effects Analysis* (FMEA) is a well-known technique for evaluating the effects of potential failure modes of components of a system. It is a crucial reliability and safety engineering activity for critical systems requiring systematic inductive reasoning from postulated component failures. We present an approach based on SysML and Prolog to support the tasks of an FMEA analyst. SysML block diagrams of the system under analysis are annotated with valid and error states of components and of their input flows, as well as with the logical conditions that may determine erroneous outputs. From the annotated model, a Prolog knowledge base is automatically built, transparently to the analyst. This can then be queried, e.g., to obtain the flows' and blocks' states that lead to system failures, or to trace the propagation of faults. The approach is suited for integration in modern model-driven system design processes. We describe a proof-of-concept implementation based on the Papyrus modeling tool under Eclipse, and show a demo example.**

***Index Terms*—FMEA, SysML, Prolog, Reliability assessment**

I. INTRODUCTION

Failure Mode and Effects Analysis (FMEA) is an engineering technique for evaluating the effects of potential *failure modes* of parts of a system[1]. In the critical systems domain, it is widely used to systematically identify the potential failures of components and analyze their effects on the system, which could adversely affect its overall reliability or safety.

The main artifact used in FMEA is a *worksheet*, providing guidance for conducting a structured analysis, for checking consistency, and for documentation. Different worksheet types may be adopted, the choice depending on the context, specific goals, the customer, the system safety working group, the safety manager, the reliability group, the analyst. All these influence the amount and type of information FMEA has to consider. A worksheet should report the following minimal information for every failure mode of a component:

- Failure mode;
- Component-level effects resulting from failure;
- System-level effects resulting from failure;
- Failure mode causal factors;
- Recommendations.

[1]Parts may be for instance subsystems, assemblies, components, functions. Here, we will generically refer to them as *components*.

FMEA is a time-consuming technique, generally performed manually. The support provided by tools is still limited to specific tasks, e.g., the analysis of faults propagation among components and the effectiveness of fault barriers on the system safety, isolated from the system development context. More complex tasks, such as the analysis of the effects of multiple failures, are often neglected by the analyst. This lack of support, along with the increasing complexity of systems, leads either to expensive and error-prone (manual) analyses or to approximate results.

We propose an approach to support FMEA by enabling formal knowledge representation – thus automated reasoning – within a SysML-based model-driven context. SysML is a design language increasingly used in critical domains, for embedded [1] as well as for large-scale systems [2][3]. Failure modes and propagation conditions are specified complementing SysML models (Block Definition Diagrams and Internal Block Diagrams – BDDs and IBDs), by means of annotations and stereotypes. This enables automatic transformation into a Prolog knowledge base, which can then be queried, e.g., to identify the flows' and blocks' states that lead to system failures, or to trace the propagation of faults. The approach eases the FMEA tasks in that it supports reasoning in the same conceptual framework of a model-driven design methodology, favoring communication among the designer and the analyst, early exploitation of design artifacts for FMEA, and automating inductive reasoning steps about fault propagation under single as well as multiple failures.

The paper is structured as follows. Section II details the motivations and discusses existing works on support for FMEA. Section III describes the proposed approach, and Section IV the architecture of an Eclipse-based support environment. Section V presents an example, based on a prototypal implementation of the environment. Section VI concludes the paper, highlighting future work.

II. MOTIVATIONS AND RELATED WORK

FMEA is a disciplined technique, mainly qualitative, for systematic reasoning about failures. The analysis considers the system decomposition down to basic components: each component is analyzed, identifying all its potential failure modes; for each mode, the propagation of the effects up to the system as a whole is studied. For quantitative analysis, e.g. for reliability assessment, FMEA includes estimates of quantitative parameters, such as expected failure rates. Failure modes and rates may derive from components' technical specifications, historical data, or further appropriate information, such as handbooks of reliability prediction models. Finally, an evaluation of severity and/or probability of failure modes provides a prioritized list for corrective actions and design improvements.

Several approaches to support FMEA have been proposed in the literature. They automate some tasks starting either from a *system model*, or from some form of *fault trees*.

Approaches falling in the first category augment an architectural system model with specifications of the failure behaviour of single components: these are either explicitly provided by the user adding information to the model, or they are implicitly defined in the domain (e.g., if the system includes an electric circuit, the failure modes of basic elements – capacitor, resistors and inductors – are known). Using the augmented model, the propagation of failures is analyzed, supporting the automated production of artifacts such as Fault Trees and FMEA worksheets. Typically, the analysis generates lists of failure modes that lead to the violation of safety requirements. The analysis is performed by traversal algorithms or by simulation in [4], [5] (approach categorized as *Failure Logic Modeling* in [6]). It is performed using model checkers or constraint solvers in [7], [8], [9] (*Model-Checking / Failure Injection*, according to [6]).

Approaches falling in the second category support FMEA through *Synthesized Fault Trees* [10][11]. Algorithms are applied on top-level events of a fault tree, and the root causes is found by traversing the tree. A row of the FMEA table is generated for each path that connects the root to a leaf.

Our approach belongs to the first category. It includes three steps:

- *FMEA-oriented modeling*: we believe that the task of the FMEA analyst can benefit from the availability of a model of the system architecture, that (s)he complements with information on components' failure modes and propagation conditions; to this aim we adopt SysML, as it is increasingly being used for critical systems in industries;
- *Model transformation*: the extended (SysML) model is transformed into a knowledge base (KB) for subsequent analysis; to this aim, we adopt the logic programming language *Prolog*, as the KB can be algorithmically generated through a Model-to-Text transformation (M2T) of annotated SysML BDDs and IBD diagrams;
- *Model analysis*: FMEA queries, expressible in the form of Horn clauses, are performed on the KB; this supports typical inductive reasoning tasks of the FMEA analyst. Clearly, Prolog programming aspects are made transparent to the analyst. The KB is also meant to act as shared repository of knowledge about components' failure modes, so as to favour reuse across multiple projects in an organization.

Previous work has explored several formalisms for the modeling step, including UML/SysML [12][13], Architecture Analysis & Design Language (AADL) [9][14], Simulink models, block diagrams, and temporal logic notations [15]. For FMEA, a SysML model is augmented with information about the component failure modes; then it is translated into other formalisms suited for analysis, such as Altarica [16], or MAUDE [17]. The studies of David et *al.* [16][18] present a methodology, named MéDESIS, to enhance the development of safety critical systems. MéDESIS starts with an automatic computation of a preliminary FMEA, obtained by exploiting SysML diagrams with a dysfunctional behavior database, and links the functional design phase with reliability techniques (FMEA and dysfunctional models construction in AltaRica Data Flow) to compute reliability indicators. Likewise, we adopt a custom FMEA-oriented profile to transform the SysML model into a Prolog KB.

The idea of separating the KB from the modeled elements, promoting reuse, is present in some past studies. In [8], an external repository (defined in a non-standard language) is used to store components' failure characterization. The study [19] presents a domain meta-model that is exploited as a KB; this solution reuses the knowledge for the instances of the meta-model. A custom profile in a model-driven methodology has been proposed in [16], where FMEA worksheets are generated starting from transformations of sequence diagrams. In [20] the authors exploit an ontology to enable an automatic FMEA generation based on a functional system model. Authors in [21] propose an ontology defined in OWL for modeling the knowledge in an FMEA. Our external KB in Prolog aims at promoting the reuse and supporting FMEA by formal queries, without burdening engineers with formalisms, like ontologies, they may be less familiar with.

The use of Prolog for supporting FMEA is envisaged in very few works, but with no link to system design models. In [22], authors report an application of Prolog III for an FMEA case study. In [23] the author proposes, as future work, to exploit pattern analysis in Prolog to develop automated ways to apply model annotations for FMEA.

III. A MODEL-DRIVEN APPROACH TO FMEA

The proposed model-driven approach aims at supporting automated reasoning on propagation of (single or multiple) failures, on their interferences and their effects. The approach is outlined in Fig. 1.

It starts from a system model expressed in SysML. SysML design artifacts are augmented with FMEA-oriented information by means of annotations, that can exploit external knowledge derived by past projects or other sources (e.g. FMEA handbooks, technical datasheets). The additional information allows

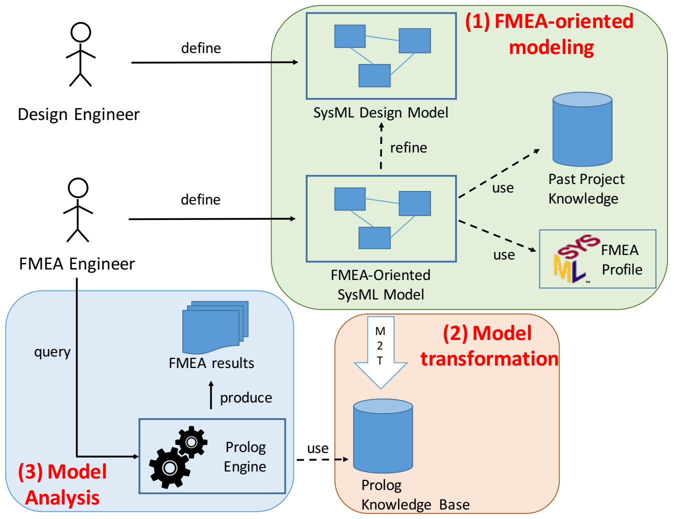


Fig. 1. Overview of the proposed approach.

model transformation to a Prolog knowledge base. The *Prolog engine* enables to query the model to derive FMEA results.

The *FMEA-oriented modeling*, *Model transformation* and *Model analysis* steps are detailed in the following.

A. FMEA-oriented SysML system modeling

SysML is an INCOSE[2] and OMG[3] standard for system modeling, supported by many commercial and open source tools. It is a graphical language, extensible through mechanisms like stereotypes and annotations. The system architecture is modeled using *structural diagrams* (namely *Block Definition Diagrams* and *Internal Block Diagrams*); information flow among components is modeled using *ports* and *flows*. We opt for SysML as the approach is meant to be integrated into standard-based model-driven development processes, where such models are already provided by design engineers. This fosters communication between the designer and the FMEA analysts, and allows to model components failure behaviors in an incremental way, as low-level system design proceeds.

The aspects of interest for FMEA are about modeling the *failure modes* and the *failure propagations*. We are defining an FMEA-oriented SysML profile to enable the analysts to add these aspects to the model. Domain-specific formalisms have been proposed in literature, such as the *Fault Propagation and Transformation Notation* [24] (Fig. 2), that we plan to further investigate as basis for defining the SysML profile.

The FMEA analysts enrich the SysML model with the following information:

1) The description of component's failure modes (and correct behavior), that are typed in order to be properly handled during the analysis.
2) The description of correct and incorrect states of the flows among components.
3) The set of conditions that constraint the component states with the flow states, and viceversa.
4) The severity of failure states of components and flows.
5) The description of fault tolerance mechanisms.

[2]International Council on Systems Engineering, www.incose.org.
[3]Object Management Group, www.omg.org.

The enriched model is then transformed in a Prolog knowledge base and used for FMEA tasks.

B. Model transformation

The *model-to-text transformation* (M2T, in the model-driven terminology) translates the annotated SysML model elements into facts and rules of a Prolog base. The failures of components and the conditions for their propagation are expressed in logical terms, by a set of *Horn clauses*[4]. For instance, let:

$$p = \text{"Electric outage"},$$
$$q = \text{"Low backup battery"},$$
$$t = \text{"Failure of system"},$$

then $\neg p \vee \neg q \vee t \iff (p \wedge q) \Rightarrow t$ is a Horn clause that asserts one failure mode of the system due to two concurrent faults, the *electric outage* and the *low backup battery*.

Any programming language with a Prolog-like syntax (e.g. ProbLog [25]) supports the M2T transformation, as well as the direct translation of queries, to be formulated by the FMEA analyst in terms of logic expressions (with Prolog syntactic details to be made transparent). It is also possible to associate a probability to the satisfaction of Horn clauses, enabling quantitative FMEA.

C. Model analysis

The Prolog engine supports the analyst in querying the modeled system. Indeed, an inference engine allows discovering information on the KB, especially knowledge that is hard

[4]A Horn clause is a *clause* (i.e., an expression formed by the disjunction of a finite collection of literals) with *at most* one positive (i.e., non-negated) literal. A Horn clause with no negative literals is sometimes called a *fact*. Horn clause are suited to be employed for both the analysis of single and multiple failures.

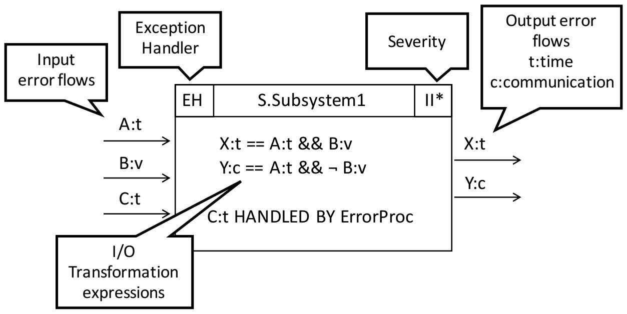


Fig. 2. The Fault Propagation and Transformation Notation [24].

to extract by a manual or a pure model-based analysis. For instance, the Prolog engine enables:

- to follow the propagation of failures inside the system;
- to identify root causes of a component's failure;
- to compute failures derived from multiple errors;
- to study the effectiveness of fault tolerance mechanisms.

The architecture includes a front-end that hides details of queries for the Prolog Engine, and offers an easy-to-use way to generate the most common FMEA results, such as worksheets or reports. When advanced analyses are needed, complex queries can be expressed directly in Prolog.

IV. AN ECLIPSE-BASED FMEA SUPPORT TOOL

The architecture of an environment supporting the approach is sketched in Fig. 3. It is based on the open source Eclipse platform [26], thus exploiting the full infrastructure of the Model-Driven Engineering tools provided by the Eclipse Foundation. The *FMEA plug-in* integrates the following components:

- **Papyrus**, an Eclipse plug-in [27] providing an integrated environment for editing models in SysML and UML 2. Papyrus offers advanced support for UML profiles, thus enabling the creation of editors for Domain Specific Languages (DSLs);
- **SWI-Prolog**, an efficient C implementation of the logic programming language Prolog [28], providing application interface libraries.

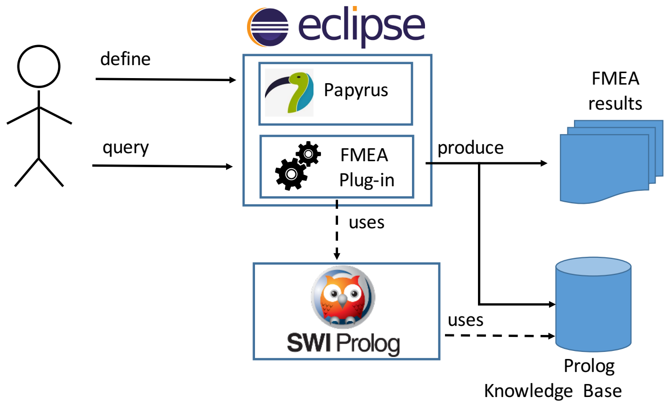


Fig. 3. An Eclipse-based architecture for the proposed approach.

An integrated environment can thus be provided for the Design Engineer to extend the system model using the FMEA Profile (Fig. 4), and to the FMEA Engineer for the analysis. The FMEA plug-in links the model with the knowledge base. The analyst can query the model through the plug-in; support for automated production of FMEA documents (worksheets or documentation) can be also provided.

A proof-of-concept implementation of the framework has been developed in Java by means of the library provided by *tuProlog*[5], and is shown in operation with the following illustrative example.

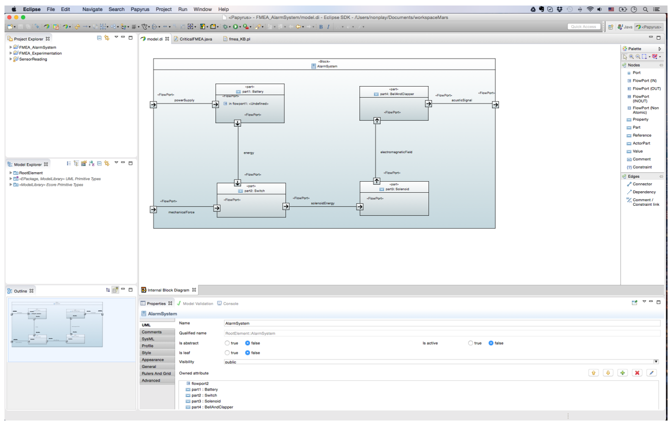


Fig. 4. SysML modeling with Papyrus in the Eclipse IDE.

V. ILLUSTRATIVE EXAMPLE

A simple alarm system is considered as example (Fig. 5). It consists of: a battery connected to the power supply, which tolerates short power outages; a push button that switches on the alarm; a solenoid, that generates an electromagnetic field; a clapper-bell device that produces the acoustic alarm sound.

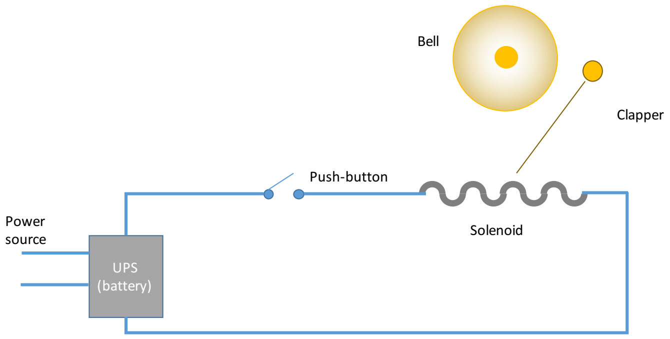


Fig. 5. The example alarm system.

Starting from the specification, the Design Engineer defines the SysML model of the alarm system, including the Internal Block Diagram shown in Fig. 6. The architecture of the system consists of four *blocks* (Battery, PushButton, Solenoid, Clapper), and of six *flows*, that connect the components through their *ports*. The flows are: the *powerSupply*; the *mechanicalForce*, that switches on the push-button; the *solenoidEnergy*, the energy from the PushButton to the Solenoid; the *electromagneticField*, generated by the solenoid when powered; and, finally, the *acousticSignal*, produced by the bell-clapper device under a proper electromagnetic field. We assume that if the Battery fails, then the energy cannot flow in the system.

[5]An open source Java-based light-weight Prolog system, available online at http://tuprolog.alice.unibo.it.

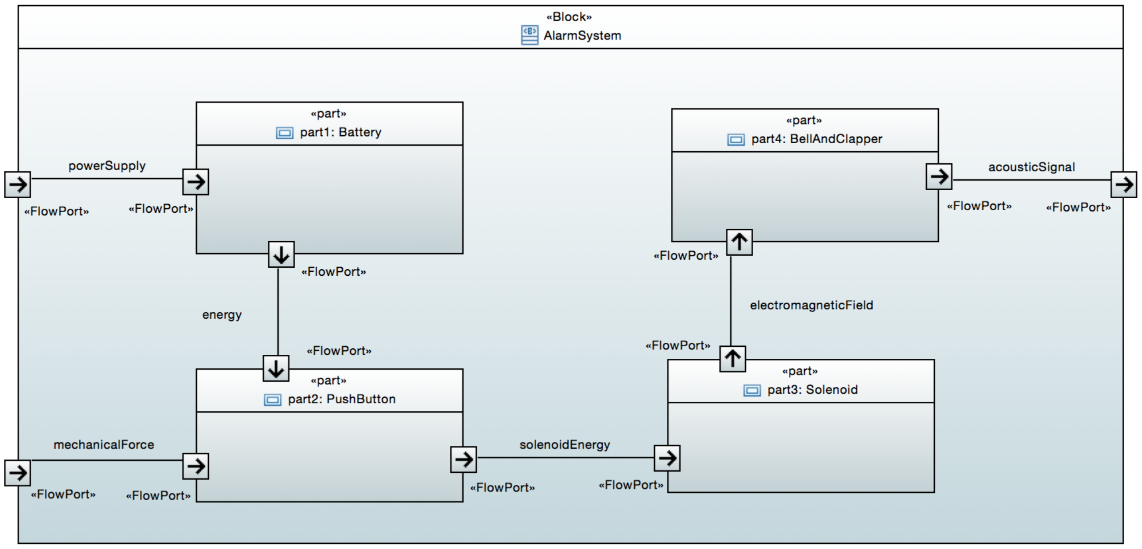


Fig. 6. SysML Internal Block Diagram of the alarm system.

Then, the FMEA analyst refines the SysML artifacts of the designer. He/she revisits the blocks and flows of interest for the analysis, augmenting the model with:

- the description of the (valid and invalid) logical states for all input flows (such as, "energy on", "energy outage");
- the description of the (valid and invalid) logical internal states of all blocks (e.g., "battery charged", "low battery"), related to their failure modes;
- the constraints on the blocks' and flows' logical states, that can depend on local or global conditions (e.g., "energy is low if the battery is low and there is an energy outage").
- additional information for quantitative analyses. In the example we add the probability that each component C_i be in a state s per demand, $P(C_i = s)$ where $s \in S_i$, the set of values (component's states) of C_i.

The last activity is mostly local to the blocks, and the modeler focuses on the conditions and effects of the failure modes. The results of the analysis are shown in Tab. I. At this stage, the model can be algorithmically translated into the knowledge base in Prolog, to perform the desired analyses.

The knowledge base contains, in general: *i)* the definition of rules and facts that are needed for the FMEA; *ii)* the knowledge that can be reused in multiple projects, part of the domain under analysis; *iii)* and by the knowledge specific for the instance of the system under analysis. The KB derived for the alarm system is listed in the following.

```
block(battery).
block(pushButton).
block(bell).
block(solenoid).
block(bell).
block(system).

flow(powerSupply).
flow(energy).
flow(solenoidEnergy).
flow(mechanicalForce).
flow(electromagneticField).
flow(acousticSignal).

isSolution(M, P) :- M = [[powerSupply, S1], [battery, S2],
  [energy, S3], [mechanicalForce, S4], [pushButton, S5],
  [solenoidEnergy, S6], [solenoid, S7],
  [electromagneticField, S8], [bell, S9],
  [acousticSignal, S10]], state(powerSupply, S1, M, P1),
  state(battery, S2, M, P2),  state(energy, S3, M, P3),
  state(mechanicalForce, S4, M, P4),
  state(pushButton, S5, M, P5),
  state(solenoidEnergy, S6, M, P6),
  state(solenoid, S7, M, P7),
  state(electromagneticField, S8, M, P8),
  state(bell, S9, M, P9),
  state(acousticSignal, S10, M, P10),
  P is P1*P2*P3*P4*P5*P6*P7*P8*P9*P10.

/* System - inputs */
state(powerSupply, on, _, 0.9999).
state(powerSupply, outage, _, 0.0001).

state(mechanicalForce, active, _, 0.001).
state(mechanicalForce, inactive, _, 0.999).

/* battery */
state(battery, lowBattery, _, 0.009).
state(battery, chargedBattery, _, 0.99).
state(battery, cell_malfunction, _, 0.001).

state(energy, on, M, 1) :-
  (member([powerSupply, on], M),
  not(member([battery, cell_malfunction], M))), !.
state(energy, on, M, 1) :-
  member([battery, chargedBattery], M).
state(energy, outage, M, 1) :-
  (member([powerSupply, outage], M),
    member([battery, lowBattery], M)), !.
state(energy, outage, M, 1) :-
  member([battery, cell_malfunction], M).

/* pushButton */
state(pushButton, working, _, 0.9998).
state(pushButton, stuckOpenCircuit, _, 0.0001).
state(pushButton, stuckClosedCircuit, _, 0.0001).

state(solenoidEnergy, on, M, 1) :-
  member([energy, on], M),
```

TABLE I
BLOCKS' AND FLOWS' LOGICAL STATES OF THE MODEL. PROBABILITY IS "-" WHEN IS DEPENDENT OF THE GLOBAL SYSTEM STATE.

Element	Type	State	Dependencies	Probability
powerSupply	Flow	on	-	0.9999
powerSupply	Flow	outage	-	0.0001
mechanicalForce	Flow	active	-	0.001
mechanicalForce	Flow	inactive	-	0.999
battery	Block	lowBattery	-	0.009
battery	Block	chargedBattery	-	0.99
battery	Block	cell_malfunction	-	0.001
energy	Flow	on	(powerSupply is on and not(battery is cell_malfunction)) or (battery is chargedBattery)	-
energy	Flow	outage	(powerSupply is off and battery is lowBattery) or (battery is cell_malfunction)	-
pushButton	Block	working	-	0.9998
pushButton	Block	stuckOpenCircuit	-	0.0001
pushButton	Block	stuckClosedCircuit	-	0.0001
solenoidEnergy	Flow	on	energy is on and ((switchButton is working and mechanicalForce is active) or (switchButton is stuckClosedCircuit))	-
solenoidEnergy	Flow	off	energy is outage or ((switchbutton is working and mechanicalforce is inactive) or switchButton is stuckOpenCircuit)	-
solenoid	Block	working	-	0.9999
solenoid	Block	brokenFerrite	-	0.0001
electromagneticField	Flow	active	(solenoidEnergy is on) and (solenoid is working)	-
electromagneticField	Flow	inactive	(solenoidEnergy is off) or (solenoid is brokenFerrite)	-
bell	Block	working	-	0.9999
bell	Block	damaged	-	0.0001
acousticSignal	Flow	active	(bell is working) and (electromagneticField is active)	-
acousticSignal	Flow	inactive	(bell is damaged) or (electromagneticField is inactive)	-

```
  (member([pushButton, working], M),
    member([mechanicalForce, active], M)), !.
state(solenoidEnergy, on, M, 1) :-
  member([energy, on], M),
  member([pushButton, stuckClosedCircuit], M).

state(solenoidEnergy, off, M, 1) :-
  member([energy, outage], M), !.
state(solenoidEnergy, off, M, 1) :-
  member([pushButton, working], M),
  member([mechanicalForce, inactive], M), !.
state(solenoidEnergy, off, M, 1) :-
  member([pushButton, stuckOpenCircuit], M).

/* solenoid */
state(solenoid, working, _, 0.9999).
state(solenoid, brokenFerrite, _, 0.0001).

state(electromagneticField, active, M, 1) :-
  member([solenoidEnergy, on], M),
  member([solenoid, working], M).
state(electromagneticField, inactive, M, 1) :-
  member([solenoidEnergy, off], M), !.
state(electromagneticField, inactive, M, 1) :-
  member([solenoid, brokenFerrite], M).

/* bell */
state(bell, working, _, 0.9999).
state(bell, damaged, _, 0.0001).

state(acousticSignal, active, M, 1) :-
  member([bell, working], M),
  member([electromagneticField, active], M).
state(acousticSignal, inactive, M, 1) :-
  member([bell, damaged], M), !.
state(acousticSignal, inactive, M, 1) :-
  member([electromagneticField, inactive], M).
```

```
/* System outputs */
state(system, system_ok, M, P) :- isSolution(M, P),
  ((member([acousticSignal, inactive], M),
  member([mechanicalForce, inactive], M));
  (member([acousticSignal, active], M),
  member([mechanicalForce, active], M))).
state(system, system_failed, M, P) :-
  isSolution(M, P),
  ((member([acousticSignal, active], M),
  member([mechanicalForce, inactive], M));
  (member([acousticSignal, inactive], M),
  member([mechanicalForce, active], M))).
```

At this stage, the analyst can pose queries and generate reports. Suppose we want to detect if there are system failures when the alarm is active. We transform the query in Prolog, and submit it to the inference engine: `state(system, system_failed, M, P),member([acousticSignal, active], M).`

The engine identifies three system's failure states where the acoustic alarm is on:

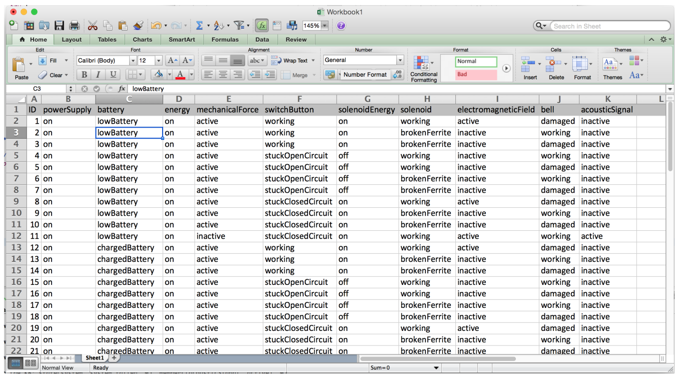
```
M = [[powerSupply, on], [battery, lowBattery],
  [energy, on], [mechanicalForce, inactive],
  [pushButton, stuckClosedCircuit],
  [solenoidEnergy, on], [solenoid, working],
  [electromagneticField, active], [bell, working],
  [acousticSignal, active]],
P = 8.988302969721009e-7

M = [[powerSupply, on], [battery, chargedBattery],
  [energy, on], [mechanicalForce, inactive],
  [pushButton, stuckClosedCircuit],
  [solenoidEnergy, on], [solenoid, working],
  [electromagneticField, active], [bell, working],
  [acousticSignal, active]],
P = 9.887133266693112e-5

M = [[powerSupply, outage], [battery, chargedBattery],
  [energy, on], [mechanicalForce, inactive],
  [pushButton, stuckClosedCircuit],
  [solenoidEnergy, on], [solenoid, working],
  [electromagneticField, active], [bell, working],
  [acousticSignal, active]],
P = 9.888122078901003e-9
```

The inference engine found three solutions: the alarm can ring when the *pushButton* is not pressed, but the button fails with failure mode "*stuckClosedCircuit*" and the bell circuit is powered: the bell circuit is powered when the battery is "charged" and powerSupply is "on" (case 1), in "outage" state (case 2), or when the battery is low but the powerSupply is "on" (case 3).

In addition, we have computed the overall probability of *failure per demand* for the system (1.01E-4), using the query $\sum_P$ `state(system, system_failed, _, P).`). A report is finally generated (Fig. 7) listing all the failure modes of the alarm system. In total, 69 failure modes are identified.



ID	powerSupply	battery	energy	mechanicalForce	switchButton	solenoidEnergy	solenoid	electromagneticField	bell	acousticSignal
1	on	lowBattery	on	active	working	on	working	active	damaged	inactive
2	on	lowBattery	on	active	working	on	brokenFerrite	inactive	working	inactive
3	on	lowBattery	on	active	working	on	brokenFerrite	inactive	damaged	inactive
4	on	lowBattery	on	active	stuckOpenCircuit	off	working	inactive	working	inactive
5	on	lowBattery	on	active	stuckOpenCircuit	off	working	inactive	damaged	inactive
6	on	lowBattery	on	active	stuckOpenCircuit	off	brokenFerrite	inactive	working	inactive
7	on	lowBattery	on	active	stuckOpenCircuit	off	brokenFerrite	inactive	damaged	inactive
8	on	lowBattery	on	active	stuckClosedCircuit	on	working	active	damaged	inactive
9	on	lowBattery	on	active	stuckClosedCircuit	on	brokenFerrite	inactive	working	inactive
10	on	lowBattery	on	active	stuckClosedCircuit	on	brokenFerrite	inactive	damaged	inactive
11	on	lowBattery	on	inactive	stuckClosedCircuit	on	working	active	working	active
12	on	chargedBattery	on	active	working	on	working	active	damaged	inactive
13	on	chargedBattery	on	active	working	on	brokenFerrite	inactive	working	inactive
14	on	chargedBattery	on	active	working	on	brokenFerrite	inactive	damaged	inactive
15	on	chargedBattery	on	active	stuckOpenCircuit	off	working	inactive	working	inactive
16	on	chargedBattery	on	active	stuckOpenCircuit	off	working	inactive	damaged	inactive
17	on	chargedBattery	on	active	stuckOpenCircuit	off	brokenFerrite	inactive	working	inactive
18	on	chargedBattery	on	active	stuckOpenCircuit	off	brokenFerrite	inactive	damaged	inactive
19	on	chargedBattery	on	active	stuckClosedCircuit	on	working	active	damaged	inactive
20	on	chargedBattery	on	active	stuckClosedCircuit	on	brokenFerrite	inactive	working	inactive
21	on	chargedBattery	on	active	stuckClosedCircuit	on	brokenFerrite	inactive	damaged	inactive

Fig. 7. The spreadsheet report generated by the tool.

VI. Conclusions and Future Work

We have proposed a model-driven approach to support FMEA. It exploits annotated system models in SysML to generate a knowledge base in Prolog, to provide automated support for the FMEA inductive reasoning tasks.

The approach is meant to work with modern model-driven methodologies. As SysML is being adopted by industries even in critical domains [2], the availability of a SysML model of the system architecture - the starting point of the approach - appears to be an opportunity rather than a limitation. BDD and IBD diagrams are intuitive models for systems engineers, which - properly extended - may serve also for conducting and documenting FMEA. We envisage the integration of the proposed approach into the model-driven engineering process based on the V-model, that we have defined and experimented in industrial collaborations [1][2].

The logic programming language Prolog is suited for representing the knowledge base, as this can be algorithmically generated by a model-to-text transformation of BDDs and IBDs. Typical FMEA queries expressible in the form of Horn clauses can then be performed on the knowledge base by means of a Logic Reasoner tool. Reuse of knowledge of components failure modes over different FMEA projects in an organization is also supported in the envisaged approach.

A tool has been set up and shown in operation with a simple example. The research is however still on-going. We foresee the following next steps. We plan to define a SysML profile and to provide guidelines for the analyst to integrate, into the system model, the information needed for FMEA. This will enable the formal definition of the model-to-text transformation into the knowledge base. The effectiveness of the approach requires also the Prolog knowledge base to be made as transparent to the analyst as possible; this demands for the definition of a proper tool supporting queries and relating results back to the system model. Indeed, the search with Prolog can become unfeasible when dealing with complex systems, like other techniques for the automatic FMEA/FTA generation. Therefore sophisticated techniques must be conceived in order to perform the analysis, such as solutions to manage the state space explosion and to parallelize the computation [30].

The investigation of such techniques and the application to real-world FMEA case studies will assess the applicability and scalability of the approach, as well as the performance of the envisaged support tools.

Acknowledgement

This work has been partly supported by the SVEVIA (Innovative methods and techniques for Software VErification and ValIdAtion of near-realtime complex systems) Research Project funded by the Italian Ministry of Education and Research (Grant no.: PON02 00485 3487758), and by EU under Marie Curie Actions of the FP7 People Programme, REA grant n.324334 CECRIS (CErtification of CRItical Systems, www.cecris-project.eu). The work of F. Scippacercola has been partially done during secondment periods at Critical Software S.A., Coimbra (PT).

References

[1] F. Scippacercola, R. Pietrantuono, S. Russo, and A. Zentai, "Model-Driven Engineering of a Railway Interlocking System," in *Proceedings 3rd Int. Conference on Model-Driven Engineering and Software Development (MODELSWARD)*, pp. 509-519, 2015.
[2] G. Carrozza, M. Faella, F. Fucci, R. Pietrantuono, and S. Russo, "Engineering Air Traffic Control Systems with Model-driven Approach," *IEEE Software*, vol. 30, pp. 42-48, 2013.
[3] G. Carrozza, M. Faella, F. Fucci, R. Pietrantuono, and S. Russo, "Integrating MDT in an Industrial Process in the Air Traffic Control Domain," in *Software Reliability Engineering Workshops (ISSREW), 2012 IEEE 23rd International Symposium on*, pp. 225–230, 2012.
[4] N. Snooke, P. Price, C. Downes, and C. Aspey, "Automated failure effect analysis for PHM of UAVs," in *Proceedings International System Safety and Reliability Conference (ISSRC)*, 2008.

[5] P. Hawkins and D. Woollons, "Failure modes and effects analysis of complex engineering systems using functional models," *Artificial Intelligence in Engineering*, vol. 12, no. 4, pp. 375–397, 1998.

[6] O. Lisagor, J. McDermid, and D. Pumfrey, "Towards a practicable process for automated safety analysis," in *Proceedings 24th International System Safety Conference*, vol. 596, p. 607, 2006.

[7] L. Grunske, K. Winter, N. Yatapanage, S. Zafar, and P. A. Lindsay, "Experience with fault injection experiments for FMEA," *Software: Practice and Experience*, vol. 41, no. 11, pp. 1233–1258, 2011.

[8] C. Picardi, L. Console, F. Berger, J. Breeman, T. Kanakis, J. Moelands, S. Collas, E. Arbaretier, N. De Domenico, E. Girardelli, O. Dressler, P. Struss, and B. Zilbermann, "Autas: a tool for supporting FMECA generation in aeronautic systems," in *Proceedings 16 th European Conference on Artificial Intelligence (ECAI04)*, pp. 750–754, 2004.

[9] M. Bozzano, A. Cimatti, J.-P. Katoen, V. Y. Nguyen, T. Noll, and M. Roveri, "Safety, dependability and performance analysis of extended AADL models," *The Computer Journal*, 2010.

[10] Y. Papadopoulos, D. Parker, and C. Grante, "A Method and Tool Support for Model-based Semi-automated Failure Modes and Effects Analysis of Engineering Designs," in *Proceedings 9th Australian Workshop on Safety Critical Systems and Software (SCS '04)*, vol. 47, pp. 89–95, 2004.

[11] M. Walker, Y. Papadopoulos, D. Parker, H. Lönn, M. Törngren, D. Chen, R. Johannson, and A. Sandberg, "Semi-Automatic FMEA supporting complex systems with combinations and sequences of failures," *SAE International Journal of Passenger Cars-Mechanical Systems*, vol. 2, no. 2009-01-0738, pp. 791–802, 2009.

[12] M. Hecht, E. Dimpfl, and J. Pinchak, "Automated Generation of Failure Modes and Effects Analysis from SysML Models," in *Proceedings IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, pp. 62–65, 2014.

[13] F. Mhenni, N. Nguyen, and J.-Y. Choley, "Automatic fault tree generation from SysML system models," in *Proceedings IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM),* pp. 715–720, 2014.

[14] A. Joshi, S. Vestal, and P. Binns, "Automatic generation of static fault trees from AADL models," in *Proceedings Workshop on Architecting Dependable Systems of The 37th Annual IEEE/IFIP Int. Conference on Dependable Systems and Networks*, 2007.

[15] M. Bozzano and A. Villafiorita, "Improving System Reliability via Model Checking: The FSAP/NuSMV-SA Safety Analysis Platform," in *Computer Safety, Reliability, and Security*, ser. Lecture Notes in Computer Science, vol. 2788, pp. 49–62, 2003.

[16] P. David, V. Idasiak, and F. Kratz, "Reliability study of complex physical systems using SysML," *Reliability Engineering & System Safety*, vol. 95, no. 4, pp. 431–450, 2010.

[17] J. Xiang, K. Yanoo, Y. Maeno, and K. Tadano, "Automatic synthesis of static fault trees from system models," in *Proceedings 5th IEEE International Conference on Secure Software Integration and Reliability Improvement (SSIRI),* pp. 127–136, 2011.

[18] P. David, V. Idasiak, and F. Kratz, "Towards a better interaction between design and dependability analysis: FMEA derived from UML/SysML models," in *Proceedings ESREL 2008 and 17th SRA-EUROPE annual conference,* p. 8, 2008.

[19] M. Molhanec, "Model based FMEA method for solar modules," in *Proceedings 36th International Spring Seminar on Electronics Technology (ISSE),* IEEE, pp. 183–188, 2013.

[20] Y. Kitamura, N. Washio, Y. Koji, M. Sasajima, S. Takafuji, and R. Mizoguchi, "An ontology-based annotation framework for representing the functionality of engineering devices," in *Proceedings ASME International Design Engineering Technical Conferences and Computers and Information in Engineering Conference,* pp. 125–134, 2006.

[21] V. Ebrahimipour, K. Rezaie, and S. Shokravi, "An ontology approach to support FMEA studies," *Expert Systems with Applications*, vol. 37, no. 1, pp. 671–677, 2010.

[22] P.-J. Gailly, W. Krautter, C. Bisière, and S. Bescos, "The Prince project and its applications," in *Logic Programming in Action*, ser. Lecture Notes in Computer Science, vol. 636, pp. 54–63, 1992.

[23] P. Schmidt, "An ontology-based annotation framework for representing the functionality of engineering devices," in *Proceedings Workshop on Spacecraft Flight Software (FSW-12)*, 2012.

[24] P. Fenelon and J. A. McDermid, "An integrated tool set for software safety analysis," *Journal of Systems and Software*, vol. 21, no. 3, pp. 279–290, 1993, applying Specification, Verification, and Validation Techniques to Industrial Software Systems.

[25] KU Leuven, "Probabilistic Logic Programming (ProbLog)," https://dtai.cs.kuleuven.be/problog, 2015, [Online].

[26] Eclipse Foundation, "Eclipse," http://www.eclipse.org, [Online].

[27] Commissariat à l'Énergie Atomique, Atos, Cedric Dumoulin, "Papyrus," http://www.eclipse.org/papyrus, [Online].

[28] J. Wielemaker et al., "SWI-Prolog," http://www.swi-prolog.org, [Online].

[29] C.A. Ericson, "Hazard analysis techniques for system safety," John Wiley & Sons, 2005.

[30] G. Gupta, E. Pontelli, K. A. Ali, M. Carlsson, and M. V. Hermenegildo, "Parallel execution of Prolog programs: a survey," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 23, no. 4, pp. 472–602, 2001.