Patterns in Safety Analysis

Tor Stålhane
Olawande Daramola
Vikash Katta
Department of Computer and Information Science
Norwegian University of Science and Technology
Trondheim, Norway
{stalhane, wande, vikash.katta}@idi.ntnu.no

***Abstract* – This work proposes the use of a pattern-based hazard descriptions and generic failure modes, in combination with domain ontology and Jackson's problem frames for automating the identification of hazards. This is an extension of our previous work in the CESAR project where we built a tool that enables a requirements engineer to write requirements in a semiformal notation based on domain knowledge described as ontologies plus a set of predefined requirement templates. Our approach will enable automatic generation of the complete FMEA table based on system's requirements, pattern based hazard descriptions and domain knowledge formalized as domain ontologies.**

Keywords – safety analysis; HazId; generic hazards; generic failure modes.

I. INTRODUCTION

The reported work is based on the work on requirement patterns defined by E. Hull et al. [1] and extended and improved in the CESAR project. In this project we combined requirements patterns with domain ontologies. This allows us to check the requirements for e.g., consistency and completeness. The ontologies also enabled the identification of system components such as sensors, actuators and control units. By including a set of generic failure modes for each component, we are able to build a partly filled in Hazard Identification (HazId) table based on Failure Mode and Effect Analysis (FMEA) for the system specified by the requirements. In order to make further progress we identified three needs: we needed (1) to describe existing hazards in the environment where the specified system should operate, (2) to formalize the failure mechanisms that operate in this environment and (3) an algorithm that could bridge that gap between the local, generic failure modes from the FMEA and the global, domain specific hazards.

The rest of this paper is organized as follows: in Section II we give a short description of the two concepts generic failure modes and generic hazards. In Section III we discuss how to describe hazards while Section IV discusses the use of textual templates – boilerplates – for hazard description. In Section V we discuss possible ways to bridge the gap between system FMEA and environment hazards. In the last section (Section VI) we discuss how we can combine theoretical work with industrial experiments to validate and improve the work.

II. GENERIC DESRIPTIONS IN SAFETY ANALYSIS

The starting point of this work was the use of textual patterns – boilerplates – for requirements, ontologies for describing equipment and generic failure modes for each part of the equipment to semi-automatically construct an FMEA table. Our work on the application of boilerplates for requirements engineering is based on the work of E. Hull et al. [1] and further developed by the partners in the CESAR project – see [2]. The reader should consult this article for further information on the definition and use of boilerplates.

To apply textual patterns in the HazId process, we have used two concepts:

- Equipment ontology. This is used to identify the components that are part of the equipment. This is used for two purposes (1) to control that there are requirements for all components of the equipment and (2) to keep an updated ontology with the generic failure modes for each component.
- Generic failure modes, which are already used in a wide area of application domains – e.g. offshore industry [3], nuclear industry [4], aerospace [5] and automotive industry [6].

A generic failure mode is a failure mode containing a group of more detailed, specific failure modes that all have the same high level manifestation – e.g., all failure modes that will lead to a motor stopping can be included in the generic failure mode "motor stops". After having studied a large set of published generic failure modes, we have settled for the following:

- Actuators – no action, wrong actions
- Sensors – no info, wrong info
- Control systems – omission, commission, incorrect, too late.

Below is a small part of the requirements written using boilerplates and the generated part of a HazId table for a steam boiler. For further discussion of the boilerplate requirements and the semi-automatic generation of a system HazId, see [7].

- <controller> shall <read> <water level> from <water level sensor>

 ISBN: 978-1-61208-221-9

- If <water level exceeds TBD>, <controller> shall <turn off> <feeding pump>
- If <water level below TBD>, <controller> shall <turn on> <feeding pump>

TABLE I. EXAMPLE OF A GENERATED PART OF A HAZID TABLE

Element	Failure mode	Risk from environment risk assessment
controller	Omission	
	Commission	
	Incorrect	
	Too late	
Water level sensor	No output	
	Wrong output	
Feeding pump	No action	
	Wrong action	

Generic hazards are widely used in industry e.g., the offshore industry's generic hazards for blow-out [8] and subsea drilling [9]. Important industrial areas like aviation [10], chemical plants [11] and the building industry [12] also have lists of relevant generic hazards.

III. HAZARD DESCRIPTIONS

There are several ways to describe hazards. We have chosen an approach based on Ericson [13], which is illustrated by the diagram in Figure 1.

Based on this diagram, a hazard description must at least contain the three topmost components – hazardous element, initiating mechanism and target and threat.

- Control unit => ***initiating mechanism***, related to the control unit's failure modes. The control unit receives info from the equipment under control.
- Controlled equipment => ***hazardous element***, related to the equipment's failure modes. The equipment can move to a hazardous failure mode either due to a wrong control command or due to an internal failure
- Target and threat => equipment's ***environment***, e.g., building and personnel. These are represented as having generic hazards.

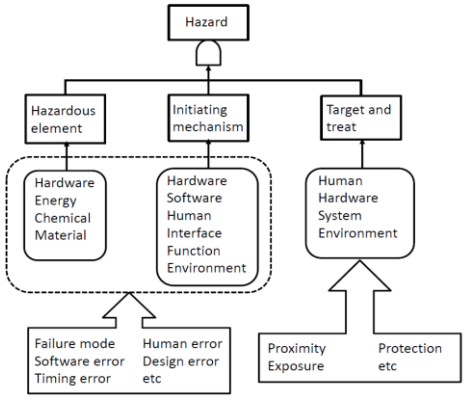

Figure 1: Hazard description pattern

We see that we need two descriptions in order to analyse an accident:

- How the equipment can harm the environment – cause an accident. Our starting point here is a list of generic hazards. The event sequence is as follows: (1) the equipment is brought into a hazardous state and (2) an event can then cause an accident – identified by one or more of the potential accidents contained in the list of generic hazards.
- How the equipment can reach a hazardous state. We need to consider how the equipment can do this alone, e.g., based on equipment characteristics or due to a faulty command from the controller.

Related to this, we need to consider (1) the controller's action and what causes it, e.g., an internal error or faulty information from equipment or from the equipment's environment via sensors and (2) the equipment entity that is affected, which will bring the equipment into a hazardous state – actuators such as pumps and valves.

The challenge is how much of the accident sequence we can describe in a generic fashion using one or more boilerplate patterns. The sequence of events that finally leads to an accident can also be described as a cause – effect chain.

In order to organize our hazard description, we have based our accident descriptions on Jackson's problem frames [14]. We need to map:

- Hazardous element, something that is in or can be brought into a hazardous state.
- Initiating mechanism, something that happens to the hazardous element or to something that might affect the hazardous element.
- Target and threat, which is a description of a potential accident.

The use of boilerplates to describe hazards will enable the representation of hazards in a semi-formal way and will thus be an improvement over the use of tables. We believe that the use of hazard boilerplates provides a useful mid-level in order to go from manual hazard analysis to machine understandable hazards for automated safety analysis. It is possible to translate hazards expressed as free text or tables into hazard boilerplates, and also to generate table-based hazards from a set of hazard boilerplates.

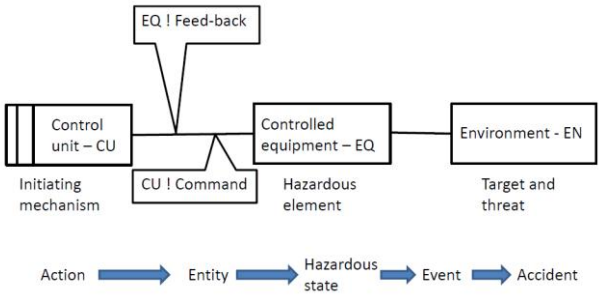

Figure 2: Pattern for equipment under control

The use of hazard boilerplates has the following advantages:

- It provides a unified structure and style of describing hazard and thereby reduces ambiguity. It also brings some consistency into the way similar hazards are represented.
- It will engender reuse of hazard descriptions since the semi-formal representation using boilerplates is more amenable to automated text processing. It creates a basis for pattern-based, structure-based, and semantic-based reuse in hazard analysis, which is useful in the context of product lines and variant systems in specific domains. Several automated safety analysis procedures for failure detection and prediction, hazard mitigation, and cause-effect analysis will benefit from reusable boilerplate hazard definitions.
- Increased completeness of hazard descriptions – ensuring that every hazard conforms to the requirements of the Hazard Classification Matrix used by Ericson [13] – see Figure 1.

IV. HAZARD BOILERPLATE DESCRIPTIONS

An element that is in a state where it has the potential to cause an accident is said to be in a hazardous state. Such an element is called a hazardous element. The element is brought to the hazardous state by an action – e.g., an equipment failure. This gives the following boilerplate formulation:

<action> **to** <entity> **in** <state> **can cause** <hazardous state>

A hazardous state does not necessarily lead to an accident. Instead it might just be the first step out of several that eventually leads to an accident. Thus, we might need several boilerplate statements in sequence to describe the full accident sequence.

If we stick to Ericson's model as shown in Figure 1, we see that an initiating mechanism applied to a hazardous element will create a threat to a target – an accident. We will use the notation {...} to indicate an alternative and since <action> **to** <entity> = <event>, we can write:

- {<action> **to**} <entity> **in** <state> **can cause** <hazardous state>
- <event> {**in** <hazardous state>} **can cause** <accident>

It is possible to have different events leading to the same hazardous state and to have several hazardous states leading to or enabling the same accident. When we have a chain of events finally leading to an accident, it is also possible to identify the event in the chain where it is most easy to stop the process and thus prevent the accident from happening.

Using the following steps, we can build the complete cause - consequence chain for any controlled equipment in any environment.

- ***Environment***: identify all generic hazards that are relevant for the environment under consideration, e.g., explosion, flooding and fire.
- ***Controlled equipment:*** use the two boilerplates "<action> **to** <entity> in <state> **can cause** <hazardous state>" and "<event> **in** <hazardous state> **can cause** <accident>" to describe how the equipment can cause an accident in the relevant environment. We start with the generic hazard and can write "<too high pressure> **to** <vessel> **can cause** <explosion>". When we consider the equipment ontology we see that <too much heat> **can cause** <too high pressure>.
- ***Control unit:*** The control unit can cause an accident by sending the wrong command to the equipment. The reason for the wrong command is either wrong input, e.g., from a sensor, or a wrong understanding of the current state of the equipment, which again is based on wrong info from the equipment.

The last step is to identify how the events leading up to the accident can be initiated. This involves understanding of how the controller works, i.e. the mapping from input sensor signals to the output actuator signals. E.g., <wrong value> **to** <temperature> **can cause** <wrong command> **to** <heater>

The last step can be used to map instruments – e.g., sensors – to an initiating event. The necessary knowledge can be taken from an equipment ontology where we find that temperature is measured using a temperature sensor. E.g., <sensor error> **can cause** <wrong value> **to** <temperature>.

With the examples above in mind we have the following event chain: EQ ! wrong temperature value => CU ! wrong command to heater => too high pressure => explosion. The event EQ ! wrong pressure value will have the same consequences.

V. BRIDGING THE GAP

We now have a semi-formal description of how the system can fail (Section II) and a semi-formal description of the environmental hazards (Section IV). We can thus bridge the gap between system failures and environment accidents – consequences.

The control unit's components are identified by using information from the control unit's requirements and from the equipment's ontology, see fig. 3. The equipment ontology will also contain failure modes – generic or specialized – for each component.

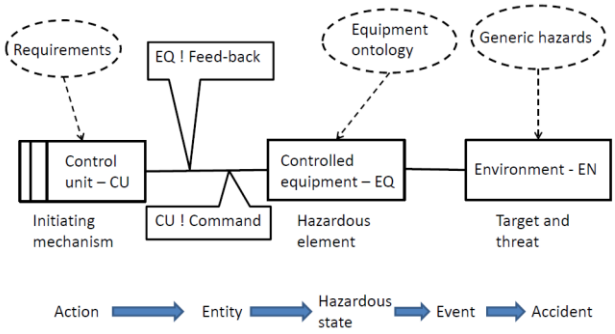

Figure 3: System-under-control pattern with necessary information

If we consider the examples at the end of the previous section, we see that the initiating event is an equipment sensor failure. A sensor registered in the tool's ontology base will have two generic failure modes: wrong output and no output and the first of these failure modes can cause an explosion.

Given that the control unit is correctly implemented, the control unit can move the equipment into a hazardous state in three ways:

- By acting correctly on a wrong signal from the equipment – e.g., a faulty sensor.
- By getting to a wrong state due to a wrong signal and then acting wrongly on a correct signal.
- By acting too late on a signal from the equipment.

Thus, the main activity for bridging the gap between hazard descriptions and the equipment and control unit's failures is to map generic failure modes onto the first action in the cause – consequence chain used in the hazard descriptions.

The typical event described in the previous section is EQ ! wrong temperature value => CU ! wrong command to heater => too high pressure => explosion. The local effect is related to the CU – send wrong command, while the global effect is the generic hazard – explosion. If we use the standard elements in an FMEA or HazId table we get the table shown in Table II below.

It is straight forward to include a detection strategy in the equipment ontology description for each component, thus extending the FMEA to a Failure Mode, Effect and Diagnostics Analysis (FMEDA). The detection strategy can be linked to each component or to each failure mode depending on the granularity of the information available. For our sensor example this could be a built-in self-test or sensor duplication.

Building on an already existing CESAR tool, we can add a new tool, which will enable the definition of hazards described as boilerplates and production of a HazId table using FMEA or FMEDA.

TABLE II. EXAMPLE OF COMPLTE HAZID TABLE

Component	Failure mode	Local effect	Global effect	p	C
Temperature sensor	No signal				
	Wrong signal	Wrong command to heater	Explosion		

VI. CONCLUSIONS AND FURTHER WORK

Our present work is based on our work in the CESAR project where we built a tool that enables a requirements engineer to write requirements in a semi-formal notation based on domain knowledge and stored as boilerplates. The equipment ontology can contain a set of generic failure modes, which allows a tool to automatically generate the first part of an FMEA table.

In the present work we have shown that it is also possible to describe hazards that stems from defined equipment failures using boilerplates and an equipment ontology. This enables us to complete the FMEA table. The new tool will also allow the engineers to add new knowledge and experience, thus making the tool an important part of the company's memory for safety analysis.

Our next steps will be to build a tool prototype and, in cooperation with an industrial partner, to enter a set of hazard definitions written as boilerplates. The tool prototype will be used in an experiment to identify strong and weak points plus identifying new functionality that need to be added in order to satisfy industrial users' needs.

REFERENCES

[1] E. Hull, K. Jackson, and K. Dick, (2004): "Requirements Engineering", Springer.

[2] O. Daramola, T. Stålhane, T. Moser, and S. Biffl (2011): "A Conceptual Framework for Semantic Case-based Safety Analysis", 16th IEEE Intl. Conf. on Emerging Technologies and Factory Automation, Toulouse France, IEEE Press

[3] SINTEF: "OREDA Offshore Reliability Data", 5th Edition

[4] J.D. Lawrence: "Software Safety Hazard Analysis", NUREG/CR-6430, February, 1996.

[5] C. Seguin, "Formal Notation Suitable to Express Safety Properties", ESACS technical report, September 17, 2001

[6] P. Johannesen, F. Tørner, and J. Torin: "Actuator Based Hazard Analysis for Safety Critical Systems", Proceedings of the 23th International Conference on Computer safety, Reliability and security, Potsdam, Germany September 2004.

[7] T. Stålhane, S. Farfeleder, and O. Daramola: "Safety analysis based on requirements", Extended Halden Reactor Project Meeting, Sandefjord, Norway, 2011

[8] H. Brant et al.: "Environmental Risk Assessment of Exploration Drilling in Nordland IV", DnV no. 2010-04-20

[9] J.L. Melendez: "Risk Assessmnet of Surface vs. Subsea Blowout prevneters (BOPs) on Mobil Offshore Drilling Units focusing on Riser Failure and the use of Subsea Shear Rams" Texas A&M University, May 2006

[10] N. Alvares and H. Lambert: "Realistic Probability Estimates For Destructive Overpressure Events In Heated Center Wing Tanks Of Commercial Jet Aircraft". 5th International Seminar on Fire and Explosion Hazards". Edinburgh, United Kingdom April 23, 2007 through April 27, 2007

[11] S. Rathnayakaa, F. Khana,, and P. Amyotte: "SHIPP methodology: Predictive accident modeling approach. Part I" Methodology and model description". Process Safety and Environmental Protection 8 9 (2011) 151–164

[12] B.E. Biringer, R.V. Matalucci, and S.L. O'Connor: "Security Risk Assessment and Management: A Professional Practice Guide for Protecting Buildings and Infrastructures". John Wiley & Sons, 2007

[13] C.A. Ericson II: "Hazard Analysis Techniques for System Safety". John Wiley & Sons, Inc., New Jersey, 2005

[14] M. A. Jackson: "Problem frames: analysing and structuring software development problems". Addison-Wesley 2001

 ISBN: 978-1-61208-221-9