SARSSi*: a Safety Requirements Specification Method based on STAMP/STPA and i* language

Jéssyka Vilela[1], Carla Silva[1], Jaelson Castro[1], Luiz Eduardo G. Martins[2], Tony Gorschek[3]
[1]Universidade Federal de Pernambuco (UFPE), Brazil, e-mail: {jffv, ctlls, jbc}@cin.ufpe.br
[2]Universidade Federal de São Paulo (UNIFESP), Brazil, email: legmartins@unifesp.br
[3]Blekinge Institute of Technology (BTH), Sweden, email: tony.gorschek@bth.se

ABSTRACT

Context: traditional hazard analysis techniques were not proposed to be used in the Requirements Engineering (RE) process. Objective: The aim of this work is to present and discuss a new method for early safety requirements specification called SARSSi* to be used at the beginning of the development of safety-critical systems. Method: this goal is achieved through the combination of two techniques: (1) STAMP/STPA; and (2) i* language. Results: this paper attempts to bridge the gap between two parallel trends in systematic safety approaches – the combination of requirements and safety engineering techniques. Our method consists of six steps and guidelines to perform a preliminary hazard analysis and facilitate the systematic identification of safety-critical functions and components. Conclusions: We demonstrate the utility of our method by applying it in a real industry case study. The initial results show preliminary suitability of our method and its contribution to improving the visualization of the information generated in the hazard analysis such as the hazards, their causes, environmental conditions, and safety requirements.

CCS CONCEPTS

Software and its engineering → Software creation and management → Designing software → Requirements analysis.

KEYWORDS

Hazard analysis, safety analysis, safety requirements, requirements engineering, safety engineering, i*, istar, STAMP, STPA, insulin infusion pump.

ACM Reference format:

Jéssyka Vilela, Carla Silva, Jaelson Castro, Luiz Eduardo G. Martins, Tony Gorschek. 2019. SARSSi*: a Safety Requirements Specification Method based on STAMP/STPA and i* language. In *Proceedings of I Brazilian Workshop on lArge-scale cRitical systEms BWare 2019, Salvador-BA, Brazil, 8 pages.* https://doi.org/10.1145/1234567890

1. Introduction

Safety-critical systems (SCS) should be carefully specified since failures could result in accidents that cause damage to the environment, financial losses, injury to people and even the loss of lives [3]. Besides, SCS are usually submitted to safety certification processes. Hence, in their development, Hazard Analysis (HA) is required to ensure that the system is safe and the hazards of the system were appropriately handled [17].

The increased complexity of sociotechnical systems has revealed the limited contributions of existing event-based accident analysis methods on sustainable safety improvements [1][2]. The most popular hazard analysis techniques such as Fault Tree Analysis (FTA), Event Tree Analysis (ETA), and Hazard and Operability Study (HAZOP) were developed many decades ago and have limitations in their applicability to today's more complex, software-intensive, sociotechnical systems [1][2]. This occurs since they assume that accidents are caused by component failures, which is mostly not true for software [17]. Furthermore, they do not consider software errors, human errors, and system design errors. Neither they usually include organizational and management flaws. The traditional HA techniques do not match the complexity of the systems being built today or the new emerging causes of accidents. Furthermore, these techniques require stable architectures making difficult their use in the beginning of the development process like in the RE phase as well as in a safety-guided design [17].

Systems-Theoretic Accident Model and Processes (STAMP) proposed by Levesson [1] is a causality model useful not only in analyzing accidents that have occurred but in developing new and potentially more effective system engineering methodologies to prevent accidents [1]. The STAMP causality model has been the theoretical background for new techniques used by safety engineering such as CAST (Causal Analysis based on STAMP) [1][7] – an accident analysis technique - and STPA (System-Theoretic Process Analysis) [1] for hazard analysis [2].

STAMP and STPA have been applied in different areas such as aerospace systems [2], railway transportation [5][6], water contamination accident [1], U.S. Army friendly fire shootings [1], biodefense [4], maritime accidents [7], and road tunnels [4]. However, to the best of our knowledge, no single study so far covers the use of STAMP and STPA in the insulin infusion pump industry.

In the safety analysis, there are many relationships among hazards, their causes, safety requirements and environmental conditions. Representing all such information through natural language has many difficulties. Among them, we can cite the dissatisfaction of practitioners [14] due to the tedious and error prone activity of managing large bodies of natural language requirements, and its ambiguity. In this context, the visualization and proper treatment of safety requirements is hampered. We observed these difficulties during an investigation of HA of an insulin infusion pump, but these problems are representative of those faced in other software-intensive safety-critical domains. Moreover, there is a tendency of using model-based hazard analysis [5].

The early consideration of safety concerns in RE and design processes is a challenging task reported by many authors, for instance [1][5][7]. In previous systematic literature reviews about requirements communication in the development of safety-critical systems [5] and integration between RE and safety engineering [6], we discuss many challenges and open issues on this topic.

In a previous paper [25], we performed a comparative study of some goal-oriented requirements languages (i*, KAOS, GRL, and NFR-Framework) that showed that they currently lack important features, such as the modeling of safety-related aspects, for describing SCS. The results also indicated that i* appears to be a promising language to specify safety requirements.

The i* (istar) language is a requirements modelling language capable of representing the dependencies and relationships among actors in sociotechnical systems [10]. This language has been used in several domains [10], such as telecommunications, air traffic control, agriculture, e-government, healthcare and business process.

Considering i* ability to specify sociotechnical systems, we believe that such language is a good option to specify SCS and to represent the results of safety analysis. Hence, we advocate the use of this language in the Safety Requirements Specification method based on STAMP/STPA and i* (SARSSi*) proposed in this paper.

Our work was motivated by the difficulties of representing the results of safety analysis through natural language [14][15] and the fact that common requirements specification languages do not fully support the needs of specifying SCS. Our goal is to provide guidance for the requirements engineers during HA and to specify the results. We illustrate the application of our method in a real industry case study of a low-cost insulin infusion pump that is being developed in a partnership between Brazilian academy and industry.

This paper is organized as follows. The SARSSi* method combining STPA and i* is described in Section 2. An industrial case study of a low-cost insulin infusion pump is presented in Section 3. We discuss related works in Section 4; the paper concludes with a brief summary of the findings and recommendations for future work in Section 5.

2. Proposed method

The SARSSi* method is composed of six steps that can be conducted iteratively as illustrated in Figure 1. Dividing the process into discrete steps reduces the analytical burden on the requirements and safety engineers and provides a structured process for hazard analysis [1].

The main idea behind our method is to use the STPA hazard analysis procedure and model the results in i*. Hence, our method did not extend the language by adding new elements or relationships. The input is a preliminary system specification that can be represented in any requirements specification language. In the next sections, we detail each step of our method.

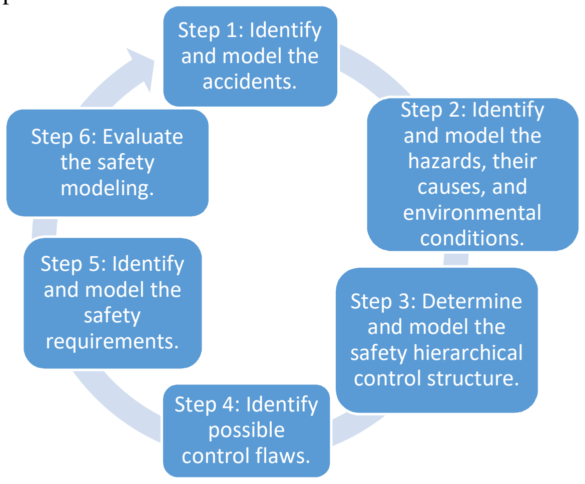
Step 1: Identify and model the accidents.
Step 2: Identify and model the hazards, their causes, and environmental conditions.
Step 3: Determine and model the safety hierarchical control structure.
Step 4: Identify possible control flaws.
Step 5: Identify and model the safety requirements.
Step 6: Evaluate the safety modeling.

Figure 1: SARSSi* method combining STPA and i* for hazard analysis.

2.1 Step 1: Identify and model the accidents

A safety-guided design must be adopted by the companies to avoid accidents and harms. When STPA is used in such type of design, only the system-level requirements and constraints may be available at the beginning of the process. Hence, the first step of the process is to identify the main *accidents* which may occur with the system including the participation of stakeholders with multiple competences and end-users.

An *Accident* is an undesired and unplanned (but not necessarily unexpected) event that results in (at least) a specified level of loss. Accidents may cause damage to the environment, financial losses, injury to people and even the loss of lives [25].

*1) Guidelines for describing accidents in i**

In i* language, the actors depend on each other to achieve their goals, to perform tasks or to obtain resources. In this step of SARSSi* method, we construct a Stategic Dependency (SD) model describing the dependency relationships among actors involved in the accidents adopting the following rules:

1. Represent the actor – *dependee* - that will suffer the consequences with the accident (actors can be *People, Property, Environment, Service, Hardware*);
2. Represent the actor – *depender* - that may cause the accident;
3. Represent the accidents through goals (the accident can be described with an "A" in the beginning of the sentence and in the negative form - ex. A.Avoid…);
4. Represent the dependency among the actors involved in the accident (*depender* and *dependee*);

5. Assign a code for each accident or harm identified for purpose of contributing for the traceability information.

After specifying the accidents of the system, the next step in our method is to identify the hazards, their causes and the environmental conditions that can lead to accidents.

2.2 Step 2: Identify and model the hazards, their causes, and environmental conditions

Hazards are system states that might, under certain environmental or operational conditions (context), lead to an accident or cause a harm [25].

Accidents can be the result of environmental conditions not considered combined with hazardous situations. Therefore, it is important to specify the causes of the hazards as well as the environmental conditions explicitly and in an analyzable form. In our method, we describe the hazards, their causes, and the environmental conditions in the Strategic Rationale (SR) model that allows expressing how the actors achieve their goals.

*2) Guidelines for modeling the hazards, their causes, and environmental conditions in i**

In the previous steps, we identified the relevant actors and their dependencies and modeled them in the SD model. Now, we can move on to the construction of the SR model that represents the intentional relationships within an actor.

We argue that the hazards are situations described as goals that we do not want to achieve. Goals in i* can be refined through the *AND-refinement* intentional relationship that allows the requirements engineers to relate a goal with the four types of intentional elements of i* (goals, tasks, softgoals and resources). We use such refinement to represent the causes of hazards, and environmental conditions as well as to visualize the dependencies among them.

In our method, we suggest the following instructions to model the hazards, their causes, and environmental conditions in i*:

1. Refine the *dependee actors* (those who can suffer an accident) in the SR model with goals they want to achieve;
2. Refine the *depender actors* (those who can contribute to a hazard) in the SR model adding new goals to represent the hazards (H), their causes (C), and the environmental conditions (E). These goals can be represented with letters in the beginning (H.; C.; E.) and in the negative format (ex. Do not have/Do not suffer/Should not [15]);
3. Relate the new hazards goals with the corresponding causes and environmental conditions using an AND-refinement relationship;
4. Add new actors, goals and dependencies if necessary;
5. Assign a code for each hazard, cause, and environmental condition identified for purpose of contributing for the traceability information.

From the SD and SR models, we can create the safety hierarchical control structure that is described in the next step of our method.

2.3 Step 3: Determine and model the safety hierarchical control structure

In STPA, by describing accidents in terms of a hierarchy of control based on adaptive feedback mechanisms, adaptation plays a central role in the understanding and prevention of accidents [1]. The safety hierarchical control structure will differ among companies and examples are spread among the book of Levesson [1] and many papers [4][5][6][7]. Therefore, there are several correct safety hierarchical control structures: what is practical and effective will depend greatly on cultural and other factors [1].

3) Guidelines for creating the safety hierarchical control structure from SD model

Actors in i* are active entities that depend on each other to achieve goals, perform tasks and provide resources. Since actors can achieve goals that would hardly be met alone, they are good candidates for components in the safety hierarchical control structure. To obtain a preliminary version of such structure, the guidelines below can be followed:

1. Represent the actors of the SD and SR models as components of the hierarchical control structure;
2. Decompose the components into its constituent sub-components until all sub-components that directly contributes to a hazard are identified;
3. Create the software module component and decompose it into sub-modules;
4. Represent the dependency relationships of the SD model as communication paths (interactions) among components;
5. Specify the interactions between the components or sub-components that directly communicate with each other;
6. Add new components, i.e. users, hardware/mechanical devices and hardware/software subsystems as well as new relationships if necessary.
7. Update the SD and SR models with the new components (actors) and interactions (dependency relationships) identified in this step.

After defining the safety hierarchical control structure, the next step is to identify and model possible control flaws.

2.4 Step 4: Identify possible control flaws

According to the STAMP causality model, the classification of accident causal factors starts by examining each of the basic components of the safety hierarchical control structure and determining how their improper operation may contribute to the general types of inadequate control [1].

At each level of the hierarchical structure, inadequate control may result from missing constraints (unassigned responsibility for safety), inadequate safety control commands, commands that were not executed correctly at a lower level, or inadequately communicated or processed feedback about constraint enforcement [1]. Therefore, we aim at this step to evaluate if there are control flaws not detected yet and specify them in i*.

4) Guidelines to identify possible control flaws

1. At each level of the safety hierarchical structure, verify if there are inadequate controls that can lead to hazardous situations that were not identified yet.
2. If there is, specify the hazards as proposed in step 2.
3. If necessary, update the safety control structure.
4. Evaluate if there are new environmental conditions and model them as described in step 2;
5. If new elements were identified, assign a code for each new element.

The hazard analysis requires the definition of system requirements and constraints necessary to eliminate or mitigate the hazards and therefore to increase the system safety. In the next section, we describe the step to represent this information.

2.5 Step 5: Identify and model the safety requirements

Our focus in this step of our method is to capture the safety requirements that should be able to eliminate or mitigate a hazard. We rely on the AND/OR refinements of i* to model these elements.

*5) Guidelines for modeling the safety requirements in i**

In the previous steps, we modeled the hazards, their causes, and environmental conditions as goals and related them with AND/OR refinements. With this modeling we represent all the situations we want to avoid in order to develop a safe system, similarly to the approach adopted by Lamsweerde [22] to model anti-goals related to security. In this step, we relate the hazards, their causes and environmental conditions with the safety requirements by using the AND/OR refinements of i* language considering the following guidelines:

1. Represent the safety requirement as a task element.
2. Evaluate if the hazard (goal) has an environmental condition (subgoal):
   - 2.1. If it does not have, associate the new task created directly with the hazard using an OR-refinement.
   - 2.2. If the hazard has environmental conditions:
     - 2.2.1. If the new task is a means to mitigate all environmental conditions, connect the new task with the hazard (parent goal) using AND-refinement;
     - 2.2.2. If the new task is a means to mitigate some (not all) environmental conditions, connect the new task with each environmental condition using OR-refinement;
     - 2.2.3. If the new task is not a means to mitigate any environmental condition, but it is a strategy to mitigate the hazard, connect the new task with the hazard (parent goal) using AND-refinement.
3. Add new dependency relationships to represent the safety requirements if necessary;
4. Assign a code for each safety requirement identified for purpose of contributing for the traceability information.

i* good practices, for example [27][30], can be adopted to model such information and to address situations not cited in this paper. After modeling the accidents, hazards, environmental conditions, control flaws and the safety requirements, the next step is to evaluate the safety modeling.

2.6 Step 6: Evaluate the safety modeling

The final step of our method is to analyze if we modeled the safety concepts needed at this early safety analysis. It is performed checks whether the safety concepts and relationships are presented in the specification. We present some guidelines to assist the requirements engineer in this task.

6) Guidelines for evaluating the generated models

- Verify if all accidents have at least one hazard;
- Verify if there is any cause of hazard or environmental condition missing;
- Verify if every hazard has at least one safety requirement;
- Verify if there is any component with an interaction missing in the safety control structure.

After such preliminary evaluation, a detailed and rigorous evaluation should be performed by safety engineers. In the next section, we will use our SARSSi* method to model and analyze a case to show the utility of our method.

3. SARSSI* applied in an Insulin Infusion Pump

The safety of insulin infusion pumps (IIP) has been one of the main concerns in health care domain since in the analysis of incidents involving these equipments, medical device regulators concluded that two of the major factors contributing to insulin infusion pump failures were software defects and user interface issues [12].

Modern insulin pumps depend on software for new features. Software is increasingly responsible for safety functions such as dosage control, interpreting user input and providing display output, and mitigating certain hazards through alarms and alerts [11]. We agree with Zhang et al. [11] that implementing safety requirements using model-based methods may reduce design/implementation flaws in insulin pump development and evolutionary processes, therefore improving overall safety of insulin pump software.

3.1 Sources of information

Unfortunately, no automatic tools exist for identifying hazards. It takes domain expertise and depends on subjective evaluation by those constructing the system. Accordingly, the HA provided in this paper regarding the IIP is based on the following sources of information:

- Domain knowledge from the authors;
- User manual of insulin infusion pumps available in the market;
- Insulin infusion pump requirements specification;
- Papers regarding a protocol previously developed [8] and lessons learned in previous analysis [9];
- Recommendations on insulin infusion pump design, and previous hazard analysis of insulin infusion pumps found in the literature [11][12][13].

The motivation for the Brazilian company to develop a low-cost IIP is the absence of companies in Brazil developing such device [8][9]. The system goal is to provide safe and effective treatment for people suffering from Diabetes Mellitus (DM1) and to enhance the long-term health of the patients. In the next sections, we outline the application of the proposed method in the IIP project.

3.2 Step 1: Identify and model the accidents

Some accidents than can occur due to the use of the insulin infusion pump are presented in Figure 2. It just shows a partial view. We assigned a code for all information with the purpose of contributing for the traceability information in future proposals. The actors identified at this step were the *Environment*, *User*, *Alarms*, *Motor Module*, *Battery*, and *Infusion Mechanism.*

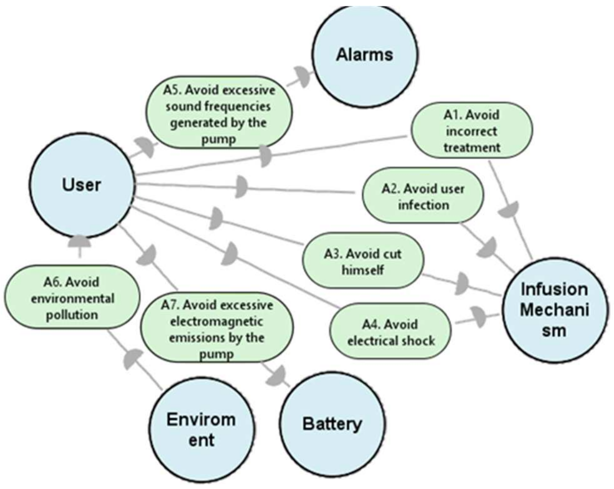
Alarms
A5. Avoid excessive sound frequencies generated by the pump
A1. Avoid incorrect treatment
User
A2. Avoid user infection
A3. Avoid cut himself
A6. Avoid environmental pollution
A7. Avoid excessive electromagnetic emissions by the pump
A4. Avoid electrical shock
Infusion Mechanism
Enviroment
Battery

Figure 2: Partial view of some accidents that can occur in using the IIP.

3.3 Step 2: Identify and model the hazards, their causes, and environmental conditions

We used our sources of information to determine the hazards, their causes as well as the environmental conditions that contribute to accidents. For example, the *User* actor depends on the *Infusion Mechanism* actor for *A1. Avoid incorrect treatment* (accident). To achieve this goal, *Infusion Mechanism* should mitigate the hazard *H1. Avoid an overdose*. This hazard will not occur if the cause *TC1.Do not have free flow* is satisfied. This goal is satisfied if all subgoals that represent the environmental conditions associated with this cause of hazard is satisfied.

Examples of conditions associated with the free flow are:

- *Valves in the delivery path are broken (TC.1.C1);*
- *Air pressure within the pump is much lower/higher than ambient air pressure (TC.1.C2);*
- *Pump is positioned much higher than the infusion site, causing unintentional drug flow (TC.1.C3);*
- *Delivery path is damaged, creating a vent on the path that allows unintentional gravity flow (TC.1.C4); and*
- *Large temperature changes causing a mismatch between drug reservoir volume change and insulin density change (TC.1.C5).*

The modeling of the hazard *overdose*, *free flow* as a cause of this hazard and the associated environmental conditions are illustrated in Figure 3. Finally, the SD and SR models were updated with the new actors and dependencies identified in this step.

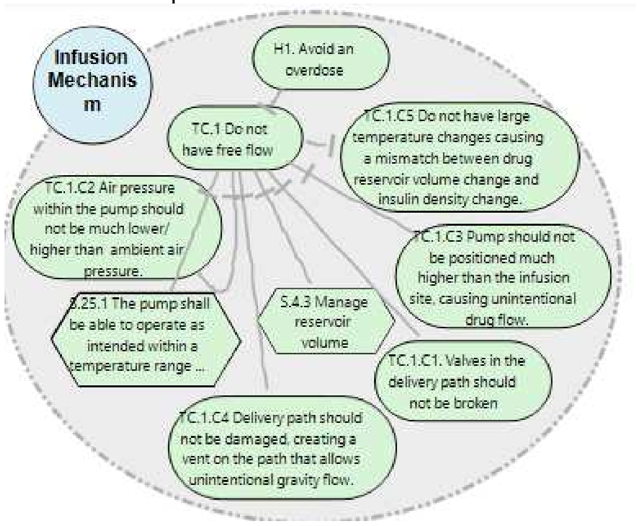
Infusion Mechanism
H1. Avoid an overdose
TC.1 Do not have free flow
TC.1.C5 Do not have large temperature changes causing a mismatch between drug reservoir volume change and insulin density change.
TC.1.C2 Air pressure within the pump should not be much lower/ higher than ambient air pressure.
TC.1.C3 Pump should not be positioned much higher than the infusion site, causing unintentional drug flow.
S.25.1 The pump shall be able to operate as intended within a temperature range ...
S.4.3 Manage reservoir volume
TC.1.C1. Valves in the delivery path should not be broken
TC.1.C4 Delivery path should not be damaged, creating a vent on the path that allows unintentional gravity flow.

Figure 3: Partial view of the hazard overdose, causes, environmental conditions and safety requirements.

3.4 Step 3: Determine and model the safety hierarchical control structure

From the SD and SR models, we obtained the safety hierarchical control structure of the IIP, following the guidelines of section 2.3, and it is illustrated in Figure 5. We represented the actors of the SD model as components of the control structure and we decomposed the system into its constituent sub-components: *syringe plunger*, *catheter (tube)*, *reservoir*, *display (interface)*, *device data recorder (flash memory), microcontroller*. We also decomposed the software module component into sub-modules: *Motor Module*, *Business*, *Display*, *Main*, *State-view*, and *Timer*. The information about the decompositions of the components were extracted from the sources of information presented in Section 3.

The dependency relationships of the SD model were represented as interactions among components and we also specified the interactions between the system and its sub-components that directly communicate with it.

The control structure diagram in STPA may have four components: controller (automated/human), actuator, controlled process, and sensors. In this case study, we did not include any sensor component such as pressure sensors (up or down) which can detect when the syringe is empty or when the patient's vein is blocked. Since these sensors are defined in the design phase and we are specifying the system at requirements level, it is not the scope of the method to describe sensors at this stage of analysis.

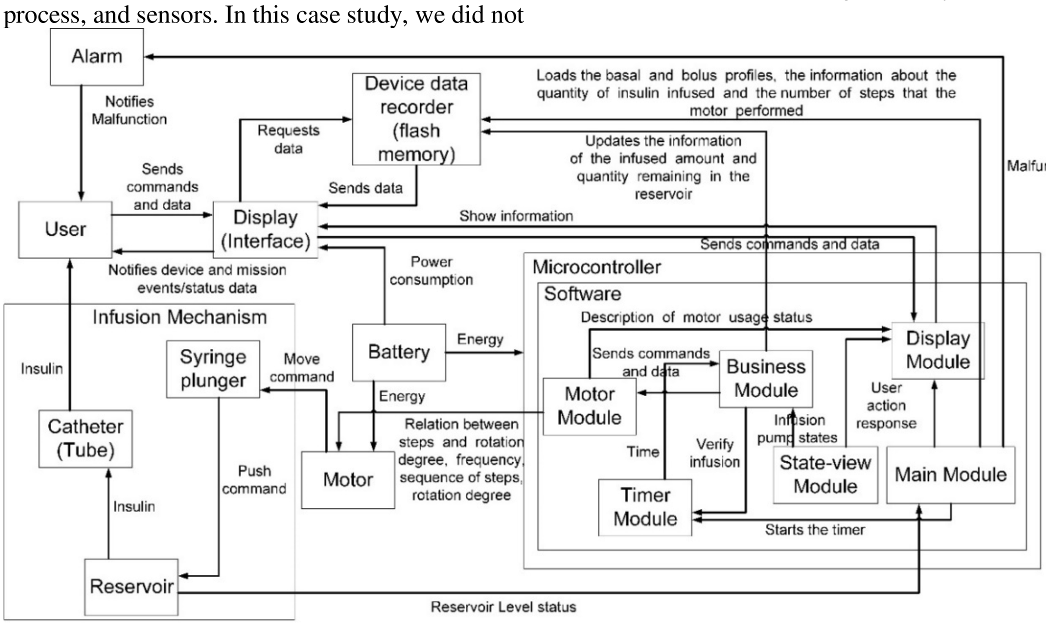

Figure 4: The safety hierarchical control strcuture of infusion pump.

3.5 Step 4: Identify possible control flaws

In this step, we evaluated if control flaws were not detected by analyzing the causes of the hazards specified as well as in the safety hierarchical control structure. An example of a serious control flaw in the insulin infusion pump would be not controlling the timer. This flaw could originate incorrect treatment for the user (patient) since it is used to control the amount of insulin he/she will receive. If control flaw are detected, the engineer should represent them as hazards and return to step 2.

3.6 Step 5: Identify and model the safety requirements

From the hazards, their causes, and environmental conditions, a set of safety requirements were derived to prevent them. For example, considering the *H.1 Avoid an overdose* hazard and one of its causes that is the *Free flow (TC.1)* illustrated previously. Some safety requirements to mitigate them can be the operation of the pump within a temperature range of x°C to y°C, and Manage reservoir volume (see 00). They should be described in i*, following the guidelines of Step 5.

3.7 Step 6: Evaluate the system safety

We performed the evaluation of the generated i* models. We checked the SD and SR models for syntax and modeling errors and we ensure that the accidents have at least one hazard; every hazard have at least one safety requirement; and we believe that the safety relevant interactions among the components of the safety hierarchical control necessary were specified. As a problem inherent of software development, problems may be detected in the later stages of system development. However, it is important to note that this is an iterative method; hence, if some information is missing, the steps can be performed as much as necessary until the models are complete.

We conducted a comparative analysis among other specification methods of safety requirements: SafeUML [23], an UML Profile [24], and SafeML [21]. We analyzed the methods using 20 criteria classified in four groups: *Modeling*, *Guidelines*, *Analysis*, and *Tool*. The results are presented in the next section.

4. Related work and Comparative analysis

The *modeling* group represents the features related to specification of important safety concepts addressed by SARSSi*, such as hazards, environmental conditions, safety functional requirements, system goals, that should be addressed by RE.

The *Guidelines* group describes if the method has some mechanism to guide engineers during safety analysis. The third group supports different types of *analysis*: safety analysis (backward or forward), trade-off analysis and communication among teams. Finally, the last group gathers features that *tools* used in these methods should support such as Generation of documentation and reports from the system model, Language consistency checker, ability to relate language elements with external resources, and maintenance of consistency among the multiple models and views. We assigned Y (yes) if the method satisfies the criterion and N (No) if it does not satisfy. The results are listed in Table 1.

Table 1: Results of the comparative analysis among the proposed method, safeuml, umlprofile, and safeml.

#	Group	Criterion	SARSS i*	SafeU ML [23]	UML Profile [24]	SafeML [21]
1	Modeling	Modeling of causes of hazards	Y	Y	Y	Y
2		Modeling of environmental conditions	Y	Y	Y	Y
3		Modeling of Safety Functional Requirements	Y	Y	Y	Y
4		Representation of system goals	Y	N	Y	Y
5		Allow to represent the relationships among hazards, their causes, the environmental conditions and the safety requirements in a graphical form	Y	Y	Y	Y
6		Support of different types of hazards	Y	N	N	N
7		Different levels of abstraction	Y	Y	Y	Y
8		Extend the language	N	Y	Y	Y
9		Specification Language	i*	UML	UML	SysML
10	Guidelines	Guidelines to derive safety requirements	Y	N	N	N
11		Guidelines to support hazard analysis	Y	N	N	N
12		Structured process for documenting the hazard analysis	Y	N	N	N
13	Analysis	Backward (B) and Forward (F) analysis	F/B	F/B	B	B
14		Support trade-off analyses	Y	N	N	N
15		Contribute to communication amongst RE and safety teams	Y	Y	Y	Y
16	Tool	Tool support	Y	Y	Y	Y
17		Generation of documentation and reports from the system model	N	Y	Y	Y
18		Language consistency checker	Y	Y	Y	Y
19		Elements can be related to the external resources	N	Y	Y	Y
20		Maintain consistency among the multiple models and views	Y	Y	Y	Y

All methods have tool support to model the safety concepts, which is fundamental for their adoption. Our method is the only one that does not extend the language and therefore all tools that support i* [16] can be used, for instance OpenOME, OME, and Pistar[1]. Moreover, SARSSi* is the only method that provides a structured process for documenting the hazard analysis and guidelines to derive safety requirements (Section 2.5).

SafeUML aims to model design elements (out of scope of SARSSi*) such as implementation, behavior, and interface elements. Therefore, the concepts necessary in early safety analysis are not addressed. An alternative is use SARSSi* and SafeUML together in different stages of development.

SafeML is a profile for SysML language that has more safety concepts to be specified at the RE phase compared to SafeUML and our method. However, its broader scope makes it more difficult to use in early safety analysis by unexperienced stakeholders. Finally, an UML Profile focuses in structuring arguments rather the specification of hazards and safety-related information, neither consider the requirements of safety standards.

The method was applied in a real case study. All sources of information regarding insulin infusion pumps used in this paper are available in the literature and were properly cited (see section 3). Therefore, we provided enough information to researchers that support the full or partial independent verification or replication of the claimed contributions.

In the low-cost IIP, using i* we came up with a very different view of the hazards, their causes and safety requirements. Since this language was proposed to be used early in the RE process, we argue that it is adequate to model the results of a preliminary HA. We found this language to be a good fit for capturing the accidents, hazards, their causes, environmental conditions as well as safety requirements and model them in the SD and SR models. We did not encounter challenges that would indicate an inadequacy in the expressive power of i* language for hazard analysis, nor did we find that i* made the design more complex than necessary.

Furthermore, it is possible to generate behavioral models for adaptive [18] and context-sensitive systems [19] from goal-oriented models such as i*. Since our method relies on i*, it

[1] https://www.cin.ufpe.br/~jhcp/pistar/tool/#

supports trade-off analysis [20] that can be conducted to evaluate the impact of the different hazards and their causes in the satisfaction of safety requirements. Moreover, such analysis contributes to risk analysis.

5. Conclusions and further research

Traditional HA techniques are limited by a focus on failure events and the role of component failures in accidents; they do not account for component interaction accidents, the complex roles that software and humans are assuming in high-tech systems, the organizational factors in accidents, as well as the indirect relationships among events and actions required to understand why accidents occur [1].

In this context, the early consideration of safety concerns in the RE and design processes is a challenging task in industry. We aim to reduce this gap with the novel SARSSi* safety requirements specification method based on STAMP/STPA and i* language proposed in this paper. It enables to analyze system safety in the RE phase based on the use of STPA and i* language. The combination of STPA and i* language is novel and we advocate that is useful in practice, and can represent a contribution in the safety-critical software industry.

In our method, we modeled *accidents*, *hazards*, their *causes*, and *environmental conditions* as *goals* and related them with AND/OR *refinements*. With this approach of modeling in early analysis, we represent all situations we want to avoid in order to develop a safe system, similarly to the approach adopted by Lamsweerde [22] to model anti-goals related to security.

As future works, we envision, besides applying the method in other real cases, the proposal of traceability relationships as well as a tool to fully support the application of our method. We aim to develop a plugin for the pistar tool to support our method. One useful feature of this tool would be the generation of documentation and reports from i* models. Moreover, a controlled experiment to compare costs, understanding and usability of the specification of safety requirements in i* as proposed in our method with the related works may be performed.

Acknowledgment

We would like to acknowledge that this work was supported by the KKS foundation through the S.E.R.T. Research Profile project at the Software Engineering Research Lab, Blekinge Institute of Technology.

References

[1] N. Leveson, Engineering a safer world: Systems thinking applied to safety. Mit Press, 2011.

[2] N. Leveson, “A new accident model for engineering safer systems”, Saf. Sci. 42 (4), 2004, pp. 237–270.

[3] J. Vilela, J. Castro, L. E. G. Martins, T. Gorschek.. Safety Practices in Requirements Engineering: The Uni-REPM Safety Module. IEEE Transactions on Software Engineering, 2018.

[4] K. Kazaras, K. Kirytopoulos, A. Rentizelas, “Introducing the STAMP method in road tunnel safety assessment”, Safety science, 50(9), 2012, pp. 1806-1817.

[5] J. Vilela, J. Castro, L. E. G. Martins, T. Gorschek, Requirements communication in safety-critical systems. In: Workshop on Requirements Engineering (WER), 2019.

[6] J. Vilela, J. Castro, L. E. G. Martins, T. Gorschek, Integration between Requirements Engineering and Safety Analysis: A systematic literature review. In: Journal of Systems and Software, vol. 125, 2017, pp. 68–92.

[7] Tae-eun Kim, S. Nazir, K. I. Øvergård, "A STAMP-based causal analysis of the Korean Sewol ferry accident." Safety science 83, 2016, pp. 93-101.

[8] L. E. G Martins, T. de Oliveira, “A case study using a protocol to derive safety functional requirements from Fault Tree Analysis”, International Requirements Engineering Conference (RE), 2014, pp. 412-419.

[9] L. E. G. Martins, H. de Faria, L. Vecchete, T. Cunha, T. de Oliveira, D. E. Casarini, J. A. Colucci, “Development of a Low-Cost Insulin Infusion Pump: Lessons Learned from an Industry Case”, International Symposium on Computer-Based Medical Systems, 2015, pp. 338-343.

[10] E. Yu, “Social modeling and i*”, Conceptual Modeling: Foundations and Applications, Lecture Notes in Computer Science, vol. 5600, 2009, pp. 99–121.

[11] Y. Zhang,P. L. Jones, R. Jetley, “A hazard analysis for a generic insulin infusion pump”, Journal of diabetes science and technology, 4(2), 2010, pp. 263-283.

[12] P. Masci, Y. Zhang, P. Jones, H. Thimbleby, P. Curzon, “A generic user interface architecture for analyzing use hazards in infusion pump software”, OASIcs-OpenAccess Series in Informatics, vol. 36. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2014.

[13] Y. Zhang, R. Jetley, P. L. Jones, P. L., A. Ray, “Generic safety requirements for developing safe insulin pump software”, Journal of diabetes science and technology, 5(6), 2011, pp. 1403-1419.

[14] E. Sikora, B. Tenbergen, and K. Pohl, “Industry needs and research directions in requirements engineering for embedded systems,” Requirements Engineering, vol. 17, no. 1, 2012, pp. 57–78.

[15] J. Whitehead, “Collaboration in software engineering: A roadmap.” FOSE, vol. 7, 2007, pp. 214–225.

[16] I* Wiki. Available at:http://istar.rwth-aachen.de/tiki-index.php?page=i*+Guide.Acessed in: January, 31st , 2017.

[17] Y. Wang, S. Wagner, “Towards applying a safety analysis and verification method based on STPA to agile software development”, inProceedings of the International Workshop on Continuous Software Evolution and Delivery, 2016, pp. 5-11.

[18] J. Pimentel, J. Castro, “Designing adaptive systems”, in: Proceedings of the Eighth International i* Workshop (istar), 2015, pp. 91–96.

[19] J. Vilela, J. Castro, J. Pimentel, “A systematic process for obtaining the behavior of context-sensitive systems”, in Journal of Software Engineering Research and Development, 4(1), 1, 2016.

[20] J. Horkoff, E. Yu, “Interactive goal model analysis for early requirements engineering”, inRequirements Engineering, 21(1), 2016, pp. 29-61.

[21] G. Biggs, T. Sakamoto, T. Kotoku, “A profile and tool for modelling safety information with design information in SysML”, in Software & Systems Modeling, 15(1), 2016, pp. 147-178.

[22] A. V. Lamsweerde, “Elaborating security requirements by construction of intentional anti-models”, in: International Conference on Software Engineering (ICSE), 2004, pp. 148–157.

[23] G. Zoughbi, L. Briand, Y. Labiche, “Modeling safety and airworthiness (RTCA DO-178B) information: conceptual model and UML profile”, Software & Systems Modeling, v. 10, n. 3, 2011, pp. 337-367.

[24] J. F. Briones, M. A de Miguel, J. P. Silva, A. Alonso, “Application of safety analyses in model driven development”, IFIP International Workshop on Software Technolgies for Embedded and Ubiquitous Systems. Springer Berlin Heidelberg, 2007, pp. 93-104.

[25] J. Vilela, J. Castro, L. E. G. Martins, T. Gorschek. Specifying Safety Requirements with GORE languages. In: Proceedings of the 31st Brazilian Symposium on Software Engineering. ACM, 2017. p. 154-163.