Hazard identification for UAVs based on soft institutions

Flavio S. Correa da Silva[1], Paul W. H. Chung[2], Marcelo K. Zuffo[1], Petros Papapanagiotou[3], David S. Robertson[3], and Wamberto Vasconcelos[4] *

[1] University of Sao Paulo, Sao Paulo Brazil
[2] Loughborough University, Loughborough UK
[3] University of Edinburgh, Edinburgh UK
[4] University of Aberdeen, Aberdeen UK

Abstract. Hazard identification requires appropriate tools to generate and analyse states that can lead to hazards, in order to support the design of preventive and/or corrective measures. In the case of socio-technical systems, hazard identification can be difficult given that the behaviour of human centric components can at best be partially predictable. In the present article we focus on a specific class of socio-technical systems – namely air spaces containing pilot controlled as well as autonomous aircrafts – and introduce the notion of *relevant hazards*. We also introduce *soft institutions* as an appropriate platform for the identification of relevant hazards.

Keywords: Safety engineering, hazard identification, socio-technical systems, soft institutions.

1 Introduction

Hazard identification is used to assess all behaviours of a system so that safety engineers can intervene in the system design to ensure that each behaviour leads to planned, foreseen and safe states [1], providing information support to design preventive and/or corrective measures. In the case of socio-technical systems, hazard identification is a difficult task given that the behaviour of human centric components can at best be partially predictable[5].

Socio-technical systems can be characterised as open asynchronous concurrent systems in which some entities are humans and others are machines. Hence, interactions between heterogeneous entities are a central concept to design, implement and analyse socio-technical systems. In the present article we focus on safety and reliability and,

* This work has been partially supported by FAPESP-Brazil and by the EPSRC-UK. Many important comments and criticisms on early versions of this work have been generously provided by Dr. David Murray-Rust (Edinburgh, UK) and Dr. Amanda Whitbrook (Derby, UK).

[5] The concept of socio-technical systems was coined in the early 50s to analyse the impact of the introduction of novel technologies in coal mining, after the empirical observation that gains in productivity were not uniform in all studied work groups. Its roots can be traced back to the analysis of the introduction of mechanisation in jute milling in Scotland during the 30s [3, 11].

more specifically, on the construction of tools to support systems design based on hazard identification. Given that it can be impossible or too difficult to fully predict the behaviour of a socio-technical system as a whole, we introduce the notion of *relevant hazards* to be considered during the design of a system. In brief, we characterise relevant hazards as a well determined subset of the set of all potential hazards for a system and reason backwards to identify all initial states and chains of events that can lead to them. We then revise the system design in order to identify points in which design interventions can either prevent hazards or inject remedial procedures to be taken in case they occur.

We focus on a specific class of socio-technical systems for which hazard identification is particularly relevant – namely, bounded air spaces containing pilot controlled aircrafts as well as unmanned aerial vehicles (UAVs). We introduce a diagrammatic language to support the characterisation of relevant hazards, of sequences of events that can lead to them and of events to which can be associated actions to be kept in store for each relevant hazard. We introduce *soft institutions* as an appropriate platform for hazard identification based on relevant hazards, and illustrate how soft institutions can be used as a formal counterpart to diagrams employed to design a system for safe operations in bounded air spaces in which pilot controlled aircrafts share space with UAVs. In section 2 we detail a characterisation of socio-technical systems, highlighting as a relevant special case mission planning for coordinated UAVs with diversified levels of autonomy. In section 3 we briefly introduce the main concepts related to hazard identification and characterise in detail the notion of relevant hazards. We also introduce a diagrammatic language to represent socio-technical systems aiming specifically at the identification and analysis of failures. In section 4 we illustrate how the proposed diagrammatic language can be used to characterise complex agent interactions in such way that hazard identification is supported. As a concrete example, we illustrate how it can be used to support the design of missions in bounded air spaces in which pilot controlled aircrafts share space with UAVs. In section 5 we introduce the concept of soft institutions, a corresponding computational platform based on this concept and how it can be used as a platform to support hazard identification for the design of socio-technical systems. Finally, in section 6 we present a brief discussion, conclusions and proposed future work.

2 Socio-technical systems

A socio-technical system can be characterised as an open network of heterogeneous interacting entities which can exchange messages in order to coordinate their actions. Some of these entities are engineered and can be programmed to behave according to rules which are explicitly determined and fully understood, even in the cases when they are not fully deterministic; other entities are human centric and therefore their behaviour can, at best, be nudged towards desired patterns of behaviour. In [3] we find a framework that characterises six facets of socio-technical systems: (1) **People**; (2) **Technologies**; (3) **Processes/procedures**; (4) **Buildings/infrastructure**; (5) **Goals**; and (6) **Culture**. Depending on the combination and organisation of these facets, different design strategies for socio-technical systems are most appropriate. We identify five dimensions to characterise different design strategies for socio-technical systems: (1) **Openness** to admit or dismiss entities; (2) **Coordination** levels among entities; (3) **Heterogeneity** of entities; (4) **Statefulness**; and (5) **Context sensitiveness**.

Different combinations of values of these dimensions require different strategies for design, implementation and management of socio-technical systems. In the present work we are specifically interested in bounded air spaces in which pilot controlled aircrafts share space with UAVs. In this scenario, a system is typically (1) **Partially open**, as aircrafts are allowed in and out of the air space provided that well specified rules and norms are followed; (2) **Locally coordinated**, as entities communicate and coordinate their actions following strict protocols which induce a hierarchy of control; (3) **Heterogeneous**, as we are considering autonomous vehicles interacting with pilot controlled vehicles and control systems comprised by sensors and actuators as well as human operators; (4) **Fully stateful**, as the states of individual entities – especially engineered entities – must be stored and managed in order to manage the whole system, particularly with respect to hazard identification and engineering; (5) **Sensitive to system states** and changes resulting from external factors as well as from consequences of state updates of entities.

Our focus in the present article is on hazard identification during system design. We are interested in structuring the interactions among entities in this scenario in such way that all relevant hazards are taken into account and design decisions are made in order to avoid failures or to build readiness to fix them in case they occur.

3 Hazard identification based on relevant hazards

We assume that all participating entities have been admitted to the system by following the interaction protocols that characterise it. Entities which do not follow certified interaction protocols are considered as external entities which can influence but are not part of the system and, therefore, are not subject to design decisions related to it. We also assume that the behaviour of an entity can be completely described by the interactions in which it is prepared to participate. The internal functioning of any entity is not taken into account explicitly. This way, human centred entities can be considered uniformly together with complex engineered entities, and entities can be described using different levels of abstraction, according to the level of detail used to specify each interaction protocol under consideration.

Strategies for hazard identification during systems design are either preventive or corrective [6]. The design of complex systems that are resilient to failures combines these two strategies so that all relevant hazards are considered. We focus on a subset of the set of *all* hazards, which are considered to be the *relevant* ones, which are in fact the ones we are able to advance during synthesis and scrutiny of a system design. Our proposed strategy for hazard identification during the design of a socio-technical system is based on the following principles: (1) System entities are uniformly abstracted as components capable of reacting to incoming messages from other components. Their reactions correspond to (a) triggering internal, encapsulated behaviours which are influenced by environmental events, (b) updating internal states, and (c) interfacing with well specified interaction protocols which can generate outgoing messages to other components. (2) General system states and behaviour can be characterised by published states, messages and interaction protocols used by system entities. (3) Hazard identification can be performed based on general system states. (4) Hazard identification based on relevant hazards corresponds to the identification of a set of system states which are considered hazards, identification of events that can lead to these states, and identification of events that can result from hazards.

In order to support this strategy, we introduce a diagrammatic language to abstract entities in a socio-technical system based on interaction protocols. The proposed language is as follows: (1) Every entity is represented as a box labelled by a unique *ID*. (2) Inside an entity we can have boxes representing the *contexts* into which the entity can enter. (3) Inside a context we can have boxes representing the *states* admitted for the entity in that context. (4) Inside a state we can have boxes representing the *interaction protocols* allowed for an entity in a given context and state. An interaction protocol can make an entity change context and/or state. In this case, the interaction protocol has a hand-off to a different context and/or state. (5) Inside an interaction protocol we have a directed graph of *actions*, in which nodes represent individual actions and edges characterise the order in which actions must occur in the interaction protocol. Every graph of actions has a root node which determines the first action to be performed, followed by its successor nodes in sequence. A branch represents a committed choice. A confluence represents a continuation that can be performed once at least one of the conflating branches succeeds. Hence, a graph of actions is a concise representation for a collection of alternative chains of actions that comprise an interaction protocol. An action can correspond to (a) querying the knowledge base of an entity, (b) performing a sensor-based operation in the environment, based on which the entity captures information from the environment, (c) receiving a message from another entity. Incoming messages must be sent by a specific entity in a given context and state, (d) updating a statement in the knowledge base of an entity, (e) performing an actuator-based operation in the environment, based on which the entity performs actions upon the environment, (f) sending a message to another entity. Outgoing messages must be addressed to a specific entity in a given context and state, or (g) changing context and/or state of the entity, in which case a hand-off of the interaction protocol in a different context and/or state is triggered.

Actions containing queries to the knowledge base, sensor-based operations and receipt of messages are called *in-actions* while actions corresponding to updates in the knowledge base, actuator-based operations, remittance of messages and change of context and/or state are called *out-actions*. Sequences of in-actions can work as preconditions for individual out-actions to occur.

Each element in the proposed language can be represented using standardised notation as presented in Figures 1a and 1b[6]. In Figure 1a we depict an entity which can participate in several contexts and assume several states within each of these contexts. For each state there are several interaction protocols which can be triggered by the entity. Some protocols have hand-offs in different contexts and/or states. Interaction protocols are portrayed as graphs inside white rectangles and hand-offs are represented as dashed arrows connecting graphs.

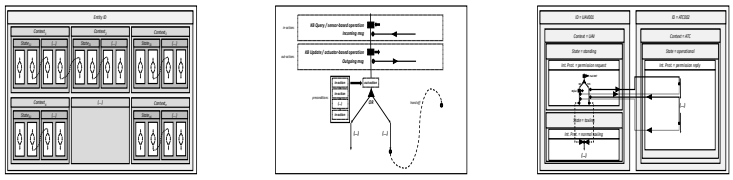
In Figure 1b we depict all possible types of actions that can belong to an interaction protocol.

As a brief example to illustrate the use of the diagrams, we feature in Figure 1c two entities – namely, a UAV and the Air Traffic Control (ATC) – during a simple interaction[7]. In this interaction, if necessary the UAV refuels and then it asks for permission to take-off. The ATC confirms the permission to take-off, and then the UAV changes state from *standing* to *taxiing*.

Hazard identification can raise the possibility that the message from the ATC never gets to the UAV. Backward reasoning could suggest that the exchange of messages

[6] Larger versions of figures can be requested to the authors.

[7] A detailed example is presented in section 4.



(a) Entity (b) Actions in interaction protocols (c) Language in action

Fig. 1: Notation for diagrammatic language

between the UAV and the ATC should contain additional steps, so that the UAV would acknowledge receipt of the message and the ATC would not stop sending copies of the permission to take-off until receiving an acknowledgement. Forward reasoning could suggest the inclusion of a time-out sensing operation as part of the interaction protocol for the UAV in *standing* state, to prevent the UAV from staying idle in case the message from the ATC never arrives. Both strategies could be combined in order to design a system that is resilient to failures.

Our purpose in building this diagrammatic language has been to support system designers activities with a pictorial language capable of exposing hazards in a system which can then be considered accordingly. In the next section we present a detailed example in which a UAV is followed from standing off-lane through flying to landing. We use this example to illustrate how the proposed diagrammatic language can be used to represent complex systems in operation and how it can be used to identify hazards and help in the refinement of system design to provide appropriate care to potential hazards.

4 An illustrative example

In order to show how the proposed diagrammatic language can be used for hazard identification, we consider a slightly more sophisticated example in which a complete mission for a UAV is depicted and analysed. This mission corresponds to a complete flight – from standing off-lane through flying to landing – and requires interactions involving the UAV and an ATC. The number of states through which the UAV passes is seven: *Standing*, *Taxiing*, *Take-off*, *Initial climb*, *En route*, *Approach* and *Landing*.

The diagrams corresponding to each state are depicted in Figures 2a to 2g.

In Figure 2a the entity UAV001 is initially switched off and off-lane. It is assumed that it is listening to the appropriate channel for messages to receive a message requiring it to start the engine, which takes UAV001 to the context of UAV and standing state. The message triggers the interaction protocol depicted in Figure 2a. When it receives a message to start the engine, it updates the knowledge base and performs the action of starting the engine. It then queries the knowledge base to check whether the engine has started. If there is a failure, then it tries again to start the engine, otherwise it updates the knowledge base and checks fuel level and systems. If there is a problem, then it stops the engine and tries to start again, otherwise it updates the knowledge base and hands off control to an interaction protocol in Taxiing state.

The proposed strategies for hazard identification and prevention/recovery have resulted in the loops back to the engine start message, together with the action to stop

the engine in case fuel and system messages indicate that the UAV is not ready for flying.

In Figure 2b we have UAV001 and ATC001. UAV001 stays in the context of UAV but now moves to taxiing state. ATC001 assumes context ATC and state to authorise taxiing towards take-off.

The interaction protocol for UAV001 in context UAV and taxiing state is slightly more complex than the protocol for standing state. UAV001 sends a message to an entity that is available in the context of ATC. In our example, ATC001 receives this message and replies back with either *take-off OK* or *take-off denied*. If take-off is denied, then UAV001 loops back and re-sends the message, until take-off is OK. When take-off is OK, then UAV001 checks whether power back is required. In case it is, then it performs appropriate operations and checks again. When power back is not required, then it finally performs taxiing and hands off control to an interaction protocol in Take-off state.

In Figure 2c, UAV001 moves to take-off state and requests authorisation to take-off. If ATC001 authorises take-off, then UAV001 performs fuel and systems verification. If there is something wrong, then take-off is aborted and a new authorisation is requested; if verification succeeds then UAV001 proceeds to take-off. If ATC001 does not authorise take-off, then UAV001 checks its knowledge base to decide whether to hold take-off or to give up. If decision is to hold take-off, then a new authorisation is requested, otherwise mission is aborted.

In Figure 2d, UAV001 performs the transition from take-off to climb, which is itself a transition state towards en route state.

In Figure 2e, UAV001 moves to en route state and maintains communication with ATC001 anytime it requests change in cruise level, until it identifies it is time to start descent. When this situation arises, then UAV001 requests permission to start descent. When ATC001 grants permission for descent then UAV001 performs descent and state moves to approach.

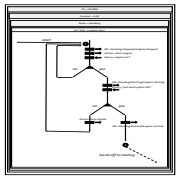
In Figure 2f, UAV001 moves to approach and maintains communication with ATC001 to request permission to start approach for landing. In case meteorological conditions are not adequate, permission is denied and, depending on what conditions are occurring, appropriate measures are taken before a second attempt to start approach for landing is started. In case meteorological conditions are fine, permission is granted and approach is started. In case some operation does not succeed during approach, UAV001 goes to circling and approach is restarted, otherwise approach is finalised and the entity moves to landing, which is the final state in this mission.

Finally, in Figure 2g, UAV moves to landing and attempts to perform landing. If it succeeds, then it goes to taxiing and switches off engines, otherwise it takes-off again.

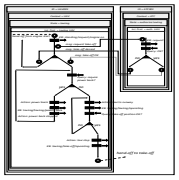
A design tool to support hazard identification in these terms must allow the representation of complex systems based on this vocabulary, and the exhaustive simulation of interactions involving entities in a system once an event (or set of events) is highlighted. In the next section we introduce *soft institutions* as an appropriate platform to build one such tool.

5 Soft institutions

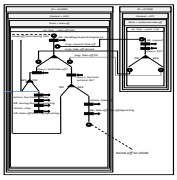
We introduce soft institutions as a tool to design and implement socio-technical systems which is particularly useful for hazard identification, given that a translation from the diagrammatic language presented in the previous sections to interactions protocols in



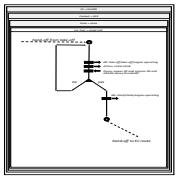
(a) UAV (standing)



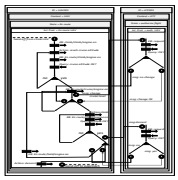
(b) UAV (taxiing), ATC (Auth. taxiing)



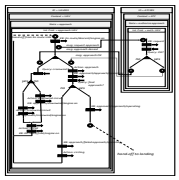
(c) UAV (take-off), ATC (Auth. take-off)



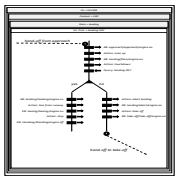
(d) UAV (climb)



(e) UAV (en route), ATC (auth. change level), ATC(auth. descent)



(f) UAV (approach), ATC (auth. approach)



(g) UAV (landing)

Fig. 2: Interaction protocols

a soft institution is immediate. Soft institutions generalise the concept of electronic institutions [4, 5, 10] to provide means to model complex systems comprised by human as well as engineered peers [7]. They have been proposed as an appropriate platform to design and implement socio-technical systems [2]. Electronic institutions are a powerful framework to build systems comprised by multiple entities based on the principle that the global behaviour of a complex system can be managed by the establishment of norms, rewards for entities that abide by these norms and sanctions for those who challenge them. In order for an entity to participate in an electronic institution, it must be prepared to respond to norms, rewards and sanctions, as well as interact with other participating entities.

Norms, rewards and sanctions in an electronic institution form a *normative system* which should be flexible in order to adjust to the observed behaviour of participating entities in an institution. The normative system dictates the way entities should behave in order to be allowed into an electronic institution and an entity (or organisation comprised by entities) must comply with the normative system in order to be able to request participation in an electronic institution. Technological entities can be designed and built to comply with normative systems and, therefore, participate in electronic institutions. Human entities, however, may feel uncomfortable to need to learn and then to be submissive to third party rules as a prerequisite to join into a network of peers.

Soft institutions, in contrast, allow entities to act freely and adjust their behaviour in a minimalist way to be able to join into local interaction protocols. Instead of having a centralised control around the normative system (as is the case with electronic institutions), soft institutions have a decentralised, possibly asynchronous control, centred on entities which choose to interact according to available protocols. Soft institutions

also consider as a basic principle that a full account of all states of the systems being modelled is not feasible, hence hazard identification can only – and at best – be based on relevant hazards as characterised in section 3.

We assume a language $\mathcal{L}$ used to describe facts and computational expressions. The language consists of three constructs: (1) **Terms**: constant or atomic expressions; (2) **Variables**: uniquely identified strings; (3) **Functions**: collections of mappings from tuples of terms to terms. Value assignments to variables are expressed as substitutions σ of the form $\{x_i \mapsto c_i\}_1^n$, which denote that the construct c_i is assigned to the variable x_i. A substitution application function $\hat{\sigma}$ is applied to whole constructs, producing a new construct in which every variable in a construct c that is present in a substitution σ is replaced by the corresponding construct. A substitution application $\hat{\sigma}$ unifies two constructs c_1 and c_2 if the application of $\hat{\sigma}$ to both constructs yields the same result, i.e. $c_1 \stackrel{\hat{\sigma}}{=} c_2$ *iff* $\hat{\sigma}c_1 = \hat{\sigma}c_2$.

Each entity maintains a personal knowledge base. It is assumed, as a design principle, that entities do not have access to each others' personal knowledge bases. It is also assumed, however, that each entity participating in a soft institution maintains a part of its knowledge base stored as a collection of $\mathcal{L}$ constructs, which we here name *institutional knowledge base*, and which are updated and consulted using two operators: (1) $\mathbb{A}(c)$ updates a fact c (*KB Update* in Figure 1b). Depending on specific institutions being designed, an update may correspond to inserting, actual updating or deleting information from the institutional knowledge base; (2) $\mathbb{K}(c, \hat{\sigma})$ consults the institutional knowledge base (*KB Query* in Figure 1b). Similar to the $\mathbb{A}$ operator, variations on the semantics of the $\mathbb{K}$ operator can be used for different soft institutions. Essentially, $\mathbb{K}(c, \hat{\sigma})$ checks whether the construct c belongs to the institutional knowledge base of an entity; if it does, then it is retrieved from the knowledge base, and the substitution $\hat{\sigma}$ is used to build the construct $\hat{\sigma}(c)$. The institutional knowledge base contains a set of ground terms $\mathcal{R} = \{R_1, \cdots, R_m\}$ which represent a set of contexts available to the entity. Contexts are parameterised by states, so that e.g. R_{i/s_j} refers to state s_j in context R_i. It also contains a set of constructs $\mathcal{PROT}$ using the syntax specified in the following paragraphs, which characterise interaction protocols available to the entity given a context and a state.

Given an implementation of a platform for soft institutions, contexts and states are the means for an entity to enter a soft institution: an entity can pick a context and then a state from $\mathcal{R}$, which become the institutional context and state of the entity and grant the entity the right to engage into interactions using an appropriate protocol available in $\mathcal{PROT}$. Contexts and states can be retrieved and updated using the $\mathbb{A}$ and $\mathbb{K}$ operators.

Messages are passed from entity to entity. To each entity is assigned a unique ID, and messages depend upon contexts and states to be properly treated. A message M is assumed to have the format $M = \langle R_{send}, gT, R_{rec}, ID_{other} \rangle$, where (1) R_{send} is the context/state that the sending entity must necessarily hold when the message is sent; (2) gT is a ground term which corresponds to the content of the message; (3) R_{rec} is the context/state that the receiving entity must hold in order for the message to be received; (4) ID_{other} is the ID of the "other" entity: it is the ID of the receiver when a message is being sent and the ID of the sender when a message is being received.

The institutional knowledge base also contains two constructs that represent the state of the entity with respect to the soft institution: (1) *Comm* stores the status of communications. It contains the entity ID and two message queues containing incoming and outgoing messages respectively. (2) *Coord* stores the status of coordination. It

contains the list of contexts and states already held by the entity including the current context/state as head of the list, the protocol being followed, the stage of execution of the current protocol and the set of variable assignments/substitutions.

Protocols are defined as a variation and extension of the *Lightweight Coordination Calculus (LCC)* [9] specified as follows: (1) A *protocol* is a list of *clauses*. A *clause* defines a script to be followed in order for an interaction to take place. Clauses have the format $cl(R, [c_1, \cdots, c_r]) ::= Def$ where $R \in \mathcal{R}$ is a context parameterised by a state, $c_1, \cdots, c_r$ are optional parameters and Def is the body of the clause:

$$\begin{aligned} Def &:= \text{Closed} \mid \text{Out} \mid \text{Out} \leftarrow [\text{In}_1, \cdots, \text{In}_s] \mid Def \textbf{ then } Def \mid Def \textbf{ or } Def \\ \text{In}_i &:= \text{rec(Msg)} \mid \text{cond}(c) \\ \text{Out} &:= \text{Null} \mid \text{snd(Msg)} \mid \text{chR}(R', [c'_1, \cdots, c'_{r'}]) \mid \mathbb{A}(c) \end{aligned}$$

(2) Closed concludes an interaction. (3) Out is an output action: (a) Null is an empty action that does nothing. (b) snd(Msg) sends message Msg to another entity. (c) $\text{chR}(R', [c'_1, \cdots, c'_{r'}])$ either changes the context of the entity during the execution of a clause or changes the state of the entity within the same context. (d) $\mathbb{A}(c)$ updates the construct c into the institutional knowledge base. (4) Out $\leftarrow [\text{In}_1, \cdots, \text{In}_s]$ performs a list of input actions and then performs an output action. An input action In_i is one of the following alternatives: (a) rec(Msg) receives a message Msg from another entity. (b) cond(c) checks whether there is a construct c' in the institutional knowledge base and a substitution $\hat{\sigma}$ such that $\mathbb{K}(c', \hat{\sigma}) = c$. The construct c is a *condition* which can be satisfied if the answer is positive. (5) **then** is a connective that represents sequential and, i.e. it joins two computational steps in sequence. (6) **or** is a connective that represents non-deterministic choice between two computational steps.

Carefully crafted sets of protocols embedded into appropriate states and contexts can implement sophisticated patterns of interaction, servicing large and complex socio-technical systems. Interaction protocols work as support services for entities to engage into well regulated and carefully designed interactions, but they are not mandatory and they do not necessarily cover all aspects of all interactions that connect entities participating in the same socio-technical system. System modeling based on soft institutions can be used to highlight facets of a system that are considered most relevant. For hazard identification, relevant hazards can be characterised in detail and simulations can be performed, so that forward and backward reasoning can be performed and the design of a system can be refined and improved towards resilience with respect to failures.

6 Conclusion and future work

In this work we have considered hazard identification during the design of systems for flight control of autonomous UAVs, based on a diagrammatic language that can be translated to protocols in *soft institutions*.

Implementations of platforms for soft institutions have already been presented elsewhere [7], and frameworks for formal verification of interaction protocols with respect to desired properties have also been developed [8]. In future work, we plan to employ these systems as a platform to support the activities of safety engineers during the design of complex systems, by providing them with tools to identify potential relevant hazards.

References

1. F. Belmonte, W. Schon, L. Heurley, and R. Capel. Interdisciplinary safety analysis of complex socio-technological systems based on the functional resonance accident model: An application to railway traffic supervision. *Reliability Engineering and System Safety*, 96:237–249, 2011.
2. F. S. Correa da Silva, P. Papapanagiotou, D. Murray-Rust, and D. Robertson. Soft institutions – a platform to design and implement sociotechnical systems (submitted). In *20th International Conference on Knowledge Engineering and Knowledge Management*, Italy, 2016.
3. M. C. Davis, R. Challenger, D. N. W. Jayewardene, and C. W. Clegg. Advancing socio-technical systems thinking: a call for bravery. *Applied Ergonomics*, 45:171–180, 2014.
4. M. Esteva, J. A. Rodriguez-Aguilar, C. Sierra, P. Garcia, and J. L. Arcos. On the formal specification of electronic institutions. In *Agent mediated electronic commerce*, pages 126–147. Springer, 2001.
5. M. Esteva and C. Sierra. *Electronic Institutions: from specification to development*. Consell Superior d'Investigacions Científiques, Institut d'Investigació en Intel· ligència Artificial, 2003.
6. E. Hollnagel. A tale of two safeties. *Nuclear Safety and Simulation*, 2013.
7. D. Murray-Rust, P. Papapanagiotou, and D. Robertson. Softening electronic institutions to support natural interaction. *Human Computation*, 2(2), 2015.
8. P. Papapanagiotou, D. Murray-Rust, and D. Robertson. Evolution of the lightweight coordination calculus using formal analysis. *Personal communication*, 2016.
9. D. Robertson. *Multi-agent coordination as distributed logic programming*, pages 416–430. Proceedings 20th International Conference on Logic Programming – Springer LNCS 3132. 2004.
10. C. Sierra, J. A. Rodriguez-Aguilar, P. Noriega, M. Esteva, and J. L. Arcos. Engineering multi-agent systems as electronic institutions. *European Journal for the Informatics Professional*, 4(4):33–39, 2004.
11. E. Trist. The evolution of socio-technical systems. *Occasional paper*, 2:1981, 1981.