A new safety and security risk analysis framework for industrial control systems

Siwar Kriaa[1,2,*], Marc Bouissou[1], Youssef Laarouchi[1]
[1] Electricité de France (EDF), 7 boulevard G. Monge, 91120 Palaiseau, France
[2] CentraleSupelec, Grande Voie des Vignes, 92290 Châtenay-Malabry, France
siwar.kriaa@gmail.com, {marc.bouissou, youssef.laarouchi}@edf.fr
*corresponding author

Abstract

The migration of modern Industrial Control Systems (ICS) towards information and communication technologies exposes them to cyber-attacks that can alter the way they function, thereby causing adverse consequences on the system and its environment. It has consequently become crucial to consider security risks in traditional safety risk analyses for industrial systems controlled by modern ICS.
We propose in this paper a new framework for safety and security joint risk analysis for industrial control systems. S-cube (for SCADA Safety and Security joint modeling) is a new model-based approach that enables, thanks to a knowledge base, formal modeling of the physical and functional architecture of cyber physical systems and automatic generation of a qualitative and quantitative analysis encompassing safety risks (accidental) and security risks (malicious). We first give the principle and rationale of S-cube then we illustrate its inputs and outputs on a case study.

Keywords

Industrial Control System, Safety, Security, Modeling, Risk Assessment, Cyber-physical system.

1. Introduction

Industrial systems like power plants, factories, airplanes or cars address the daily and vital needs of society. Their safety is usually given careful consideration as their failure or malfunction can engender adverse consequences on humans and the environment.
ICS offer the necessary means to control and supervise these critical systems and infrastructures. Traditional ICS were based solely on mechanical and electro-technical devices and proprietary standards which were well known. These systems have however become expensive to deploy, maintain and operate, and make it difficult to follow innovation trends in an industrial context. To address these challenges, new information and communication technologies are being increasingly integrated into modern control systems: radio-based services, commercial off-the-shelf products (e.g., Windows operating systems), TCP/IP based communications, etc. This migration towards standardized communication technologies and open protocols has facilitated the deployment of highly connected systems and enabled remote control and supervision of infrastructures. For instance, Supervisory Control and Data Acquisition (SCADA) systems are largely deployed in various industries. Although this has increased efficiency and reduced costs for industrial operators, the overall infrastructures have

become vulnerable to external malevolence. Indeed, with their increasing complexity and interconnection, modern industrial control systems are exposed to new security-related threats like cyber-attacks.

For a long time, much attention has been focused on safety concerns related to highly critical systems with large impacts on their environments. However, only accidental components failures or software errors were traditionally addressed in safety analyses. Today, in this context characterized by the migration of industrial infrastructures towards digital control systems, system safety can also be compromised by security breaches and electronic attacks. It is consequently no longer sufficient to address accidental threats of such systems, threats of intentional origin need to be covered as well. In line with the definitions in [1], we associate, in the remainder of this article, safety with accidental risks originating from the system that could result in unacceptable consequences on the system's environment. Security is related to malicious risks and we are mainly interested in cyber-security.

Although both safety and security communities deal with risks and share the same goal of protecting industrial infrastructures, they are still working separately. Yet, for industrial infrastructures having safety issues and being supervised and controlled by modern ICS such as SCADA, safety and security requirements and risks converge and can have mutual interactions [2]. Indeed, security related requirements and risks can influence the system safety and inversely safety related requirements and risks can influence the system security.

To address these challenges, a joint risk analysis framework considering both safety and security aspects has become essential. It enables exhaustive coverage of risks related to safety and security and identification of their potential interdependencies. As a result, it conditions a thorough and optimal risk management as well as cost and resource optimization.

We provided in [3] a survey of approaches, from industrial and scientific communities, that combine safety and security issues for industrial systems. These approaches have been evaluated in [4] according to the four following criteria which we believe essential for a good modeling approach:

1. enables formal modeling of the system architecture, and the related attack and failure modes;
2. yields both a qualitative and quantitative analysis;
3. automatically generates attack and failure scenarios that lead to a given undesirable event, from a description of the system architecture;
4. makes it possible to easily consider different hypotheses about the same system architecture and regenerate the new risk related scenarios.

Considering the limitations of existing approaches with respect to these criteria [4], we propose, a new model based approach, that we call S-cube, for SCADA Safety and Security joint modeling. The S-cube approach has already been introduced in [4] [5]. In this paper, we give the details on the rationale behind this approach and the different purposes for which it can be used. The remainder of this paper is organized as follows: Section 2 presents the related work on existing safety "only" and security "only" domain specific languages that inspired the S-cube approach. Section 3 describes the specificities of industrial control systems. Section 4 describes the S-cube principle. Section 5 explains the rationale and assumptions taken for the S-cube knowledge base (KB). Sections 6 and 7 address the qualitative respectively quantitative aspects in this knowledge base. Section 8 shows how the S-cube approach has been implemented and illustrates it on a case study. Section 9 concludes the paper and gives perspectives.

2. Related work on existing Security/Safety domain specific languages

Safety and security have been for a long time treated separately within distinct communities. The standards and tools related to each discipline have also been distinct and separate. In this section, we give an overview of existing domain specific languages (DSLs) for safety or security. DSLs aim at capitalizing knowledge on a specific domain, thus speeding up the construction of models in that domain.

2.1 Security domain DSLs

Our exploration focused on two main approaches that are based on security DSLs and enable automatic processing of system models: the Cyber Security Modeling Language (CySeMoL) [6] and the Multihost, multistage Vulnerability AnaLysis (MulVAL) [7]. These two approaches have inspired building the S-cube KB.

The CySeMoL [6] is an attack graph tool that can be used to assess the cyber security of enterprise architectures. It allows users to create models of their architectures and make calculations on the likelihood of different cyber-attacks being successful. The CySeMoL approach enables modelling of only IT components and the security-related risks. The possibility of extending CySeMoL in order to cover both safety and security had been considered but discarded for the following reasons:

- the CySeMoL metamodel is too comprehensive and enriching it with new elements requires rethinking all the dependencies and relationships between the metamodel's elements;
- the quantification process is a Monte Carlo based calculation of a Bayesian network. As a result, it does not allow dynamic aspects of the attack to be modelled and in particular its evolution over time (mean time until success). Instead, all parts of the metamodel assume that the attacker has one week to perform the attack;
- the calculation engine is not open source.

The MulVAL tool [7] is another attack graph tool used for security-related vulnerabilities assessment. It includes an engine that enables automatic generation of attack graphs, given the network configuration and security advisories given by a network scanner that identifies vulnerabilities on each host. This engine uses a set of reasoning rules that specify exploit rules, compromise propagation and multi-hop network access. MulVAL is essentially qualitative: its main output is a logical attack graph, i.e. a logical structure that can be enriched by metrics providing an assessment of the difficulty of various attack steps. Hence one can see that "logical attack graph", as used in [7], is just another name for an attack tree. Like CySeMoL, MulVAL does not consider safety issues. The possibility of extending MulVAL for safety and security joint modeling seems difficult to us for the following reasons:

- MulVAL adopts the Datalog language, which is a non-typed language; this makes it difficult to track the different modeling elements especially for complex systems and to associate them with their characteristics. We believe that an object-oriented language is more appropriate to this purpose;
- MulVAL uses non-user-friendly tools. All inputs have to be in textual form, and the "graphical" output is hardly usable for complex systems;

- The quantification of MulVAL models is too simplistic and could not be extended to safety related parts of a model.

We also had a look at DSLs designed for the formal definition and verification of protocols such as Proverif [8] but we could quickly discard them because they are limited to the specification and verification of security properties related to cryptographic protocols. Modeling higher level and abstract aspects of the system is not possible with such highly specific languages.

2.2 Safety domain DSLs

In the safety domain, there are two “levels” of domain specific languages: generic languages that enable knowledge bases to be built as libraries of classes and those libraries themselves that enable system models to be built.
Figaro [9], AltaRica [10] and AADL [11] are generic languages which have been designed for modeling systems and facilitating the safety analysis associated with them.
Figaro will be presented in Section 8.1. The Boolean logic Driven Markov Processes (BDMP) formalism [12] and the S-cube KB (cf. Section 5) are examples of libraries implemented using this language. Similar to Figaro, AltaRica is a high-level language that enables models of systems to be created. The main difference between the two languages [13], resides in the fact that Figaro enables the conception of generic modeling libraries that can be used by engineers through graphical user interfaces, without the need to manipulate the language itself; while AltaRica enables the reuse of some elements specified in libraries but always requires the system analyst to add some AltaRica code. Figaro is hence more user friendly.
To summarize, Figaro and AltaRica are generic DSLs, helping the definition of more specific DSLs in the form of libraries. There are several examples of Figaro libraries that have been in use at EDF[1] for many years, to carry out safety studies of systems, mainly from the thermo-hydraulic and electrical domains.

The Architecture Analysis & Design Language (AADL) [11] is quite different from both Figaro and AltaRica. It is dedicated to the description of electronic systems and explicitly refers to software and hardware, buses, buffers, processors etc. It was first published as SAE Standard AS-5506 in November 2004. Version 2.1 of this standard was published in Sept 2012. The AADL Error Model V2 (EMV2) is an error annex focused on safety analyses. It enables the extension of the architecture models with error types and error propagation rules that can be used to produce automatically Failure Mode and Effects Analysis (FMEA) and fault trees. By looking at the automatic translation of an AADL model into AltaRica, [14] shows that complicated and cumbersome constructions in AADL can be written in a more concise way in AltaRica (the same would apply for Figaro). But the essential difference between AADL and Figaro or AltaRica is the fact that nothing in EMV2 allows to model a dynamic behavior of the failures propagation. The same limitation applies to EAST-ADL, another language similar to AADL, but even more specialized, for the automotive industry.

The S-cube approach is based on a new knowledge base (KB) or DSL that gathers expertise on industrial information and control systems and the associated safety and security aspects. This knowledge base (KB) is the core of the S-cube approach. It incorporates some notions inspired from the existing DSLs previously mentioned but tries to overcome at the same time

[1] EDF: Electricité de France is the largest producer of electricity in France. It also covers a part of the European Union’s electricity demand.

some of their limitations. The S-cube approach aims at providing a common framework for dealing with the convergence of safety and security risks in modern control systems in order to capture their mutual interactions.
We provide in the next section an overview on Industrial Control Systems and their safety and security requirements which represent the basic notions modeled by the S-cube KB. We particularly underline the specificities related to industrial security compared to security of Information Technology (IT) systems.

3. Industrial Control Systems: specificities and requirements

3.1 ICS: definitions and Enterprise Architecture topology

We first define what we call ICS and then give an overview on the Enterprise Architecture topology encompassing these control systems.
The term ICS covers a large variety of control systems among which we find SCADA systems used generally for systems with a wide geographical range.
In the rest of this document, we use the term ICS to designate whatever control system is used to control an industrial system and the term SCADA particularly for digital systems used to supervise and control industrial infrastructures.
The Perdue Enterprise Reference Architecture (PERA) provides a reference model for Computer Integrated Manufacturing [15] that divides the enterprise architecture into different layers based on organizational hierarchy.

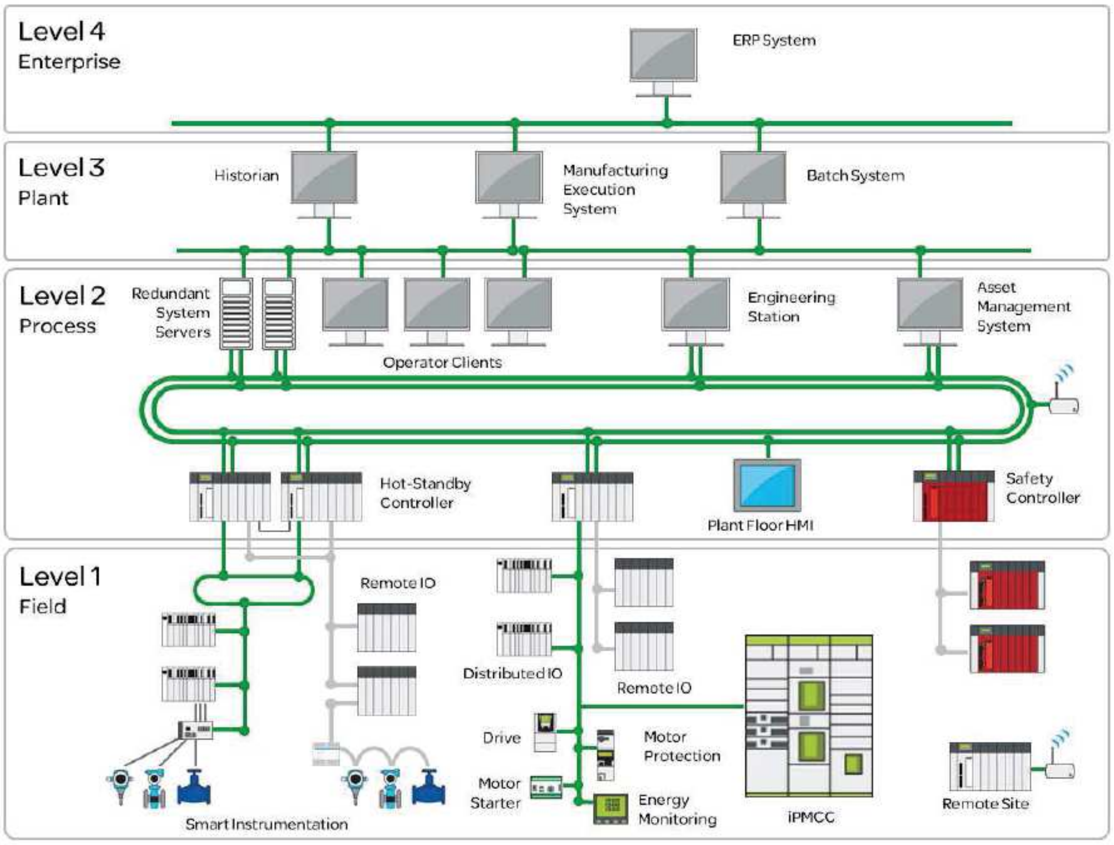

Figure 1: Typical components of modern control architectures [16]

Inspired from PERA, IEC 62264-1 [17] provides a model for the enterprise that splits the architecture into five key levels[2], based on functional hierarchy:

- Level 0: the physical process;
- Level 1: functions involved in sensing and manipulating the physical process;
- Level 2: functions involved in monitoring and controlling the physical process;
- Level 3: functions involved in managing the work flows to produce the end-products;
- Level 4: functions involved in business-related activities needed to manage the manufacturing organization.

This functional decomposition can be mapped to the real enterprise architecture where components ensuring these functionalities are placed in the corresponding level. Devices that are directly involved in the industrial control process are particularly located at levels 1 and 2. [16] depicts the typical system components of modern control architectures as illustrated in Figure 1 (Level 0 is not presented as it depends on the nature of the industrial system).
Given the evolution of the industrial control infrastructures and the integration of advanced technologies, we can find intelligent sensors that ensure both the functionalities of measurement and process control; and can therefore be considered to belong to level 1 and 2. Particularly, in smart grids, communicating smart sensors can be used to control the process of electricity distribution. Contrary to the classical hierarchical architecture, we then have a "flat" architecture where control is distributed between the different components.
In the following section, we give an overview on ICS specificities and underline the differences between securing traditional IT systems and securing ICS.

3.2 ICS specificities and security challenges

As ICS integrate new Information and Communication Technologies traditionally used in Management Information Systems, they consequently inherit their vulnerabilities. However ICS have their own specificities: they are time-critical as they monitor generally real-time processes, they require high availability and need to be fault-tolerant.
These ICS characteristics that differ from traditional IT systems imply different security risks and priorities. Security requirements for traditional IT systems, ordered with decreasing priority are respectively Confidentiality Integrity and Availability (CIA). This priority order is inversed when it comes to control systems (AIC):

- Availability: control and process data should always be available to guarantee the good execution of the industrial process;
- Integrity: control and process data (generated, transmitted, displayed and stored) should be genuine and intact;
- Confidentiality: data confidentiality is desirable in the industrial context but not of paramount importance. Unlike availability and integrity problems, data disclosure should not induce safety related issues, but may impact the enterprise image.

Unlike traditional IT systems, ICS are on the first frontier facing human lives and ecological environment. Security properties and requirements applied for IT systems are consequently not completely adapted to control systems and need to be adjusted taking into consideration ICS specificities.

[2]This standardized decomposition can, however, differ between different industries and communities.

The S-cube approach's main goal is to provide a DSL that catches ICS specificities and the related safety and security risks into a common framework. We present in the next section the main principle of this approach

4. S-cube principle

S-cube enables a joint safety and security risk analysis for systems having safety challenges and integrating new information technologies and particularly SCADA-based ICS.
Seen as a black box, the S-cube approach, depicted in Figure 2, takes as input the system architecture and gives as output the attack and failure scenarios that are likely to happen on it and that may lead to a given undesirable event. Undesirable events associated with a given system can be identified in advance by some safety systematic techniques such as FMEA. These events represent risks with intolerable consequences that can happen to the system and lead to safety issues (human losses, environmental and ecological impact, or high economic losses).
The S-cube approach relies on a knowledge base, called S-cube KB (cf. Section 5), that gathers expertise on ICS and particularly SCADA systems and their associated safety and security aspects. The S-cube KB can be seen as a DSL or a library that enables the typical components of digital industrial infrastructures to be described, including corporate enterprise network, industrial control network, field and instrumentation networks; and the related security mechanisms (authentication, access control, etc.) and safety mechanisms (redundancy, voters, etc.). Each component is associated with the attacks and failure modes that can happen on it (cf. Section 6). The effects of these failures and attack steps are described as well as the way they can propagate within the overall system architecture.

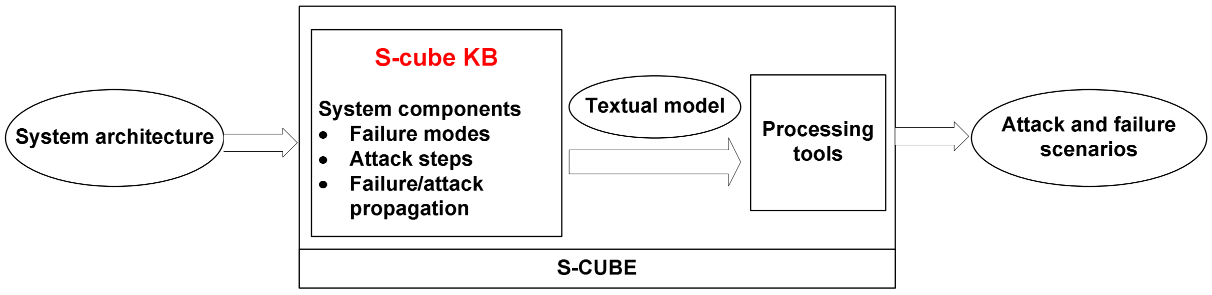

Figure 2: The S-cube approach principle

The generic models of the S-cube KB are instantiated on the input system architecture. This instantiation results in a textual model which can be processed by calculation engines that automatically generate attack and failure scenarios, with an estimation of their probabilities.

5. The S-cube knowledge base

We explain in this section the rationale used in building the S-cube KB, and the main notions modeled.

5.1 Rationale

We have chosen to model in the S-cube KB the following aspects related to industrial architectures:

5.1.1 Modeling the enterprise levels

The five enterprise architecture levels of the PERA decomposition (c.f. Section 3.1) are modeled in the S-cube KB as follows:

- **The physical process** (level 0 in the PERA decomposition) is not modeled as the S-cube KB aims at modeling ICS regardless of the kind of physical system they control. It rather includes modeling risks having impacts on the physical process (safety issues). It can be later coupled with other knowledge bases describing a specific industrial domain, for a better visibility on the physical impacts of attack and failure scenarios.
- **The field level**: (level 1 in the PERA decomposition) comprises devices that are close to the industrial process. These devices can be either sensors or actuators. Sensors are devices used to measure physical quantities like pressure, speed, temperature, etc. Actuators are devices that act directly on the physical process e.g., valves, pumps, circuit breakers;

Level 2 in the PERA methodology is split into the process and the supervision levels.

- **The process level**: comprises automation devices that enable the monitoring and control of the industrial process. In this level, we find typically Programmable Logic Controllers (PLC) and Remote Telemetry Units (RTU);
- **The supervision level**: comprises SCADA servers and remote supervision devices that enable a global view and control the process level;
- **The IT level**: comprises machines integrating information technologies. For modern control systems, information technologies are also integrated in control devices (e.g. SCADA servers, process controllers).

This decomposition is based on the functional specificities of each level with respect to control. Each level can consist of one or many networks.

5.1.2 Modeling the network zones

A network zone, in the S-cube KB, models a set of machines that are allowed to exchange information between one another using either a wired or a wireless communication technology. Examples of network zones could be the field network that connects sensors and actuators to process controllers and carries their data exchange.

5.1.3 Modeling the hardware/software system components

The functional architecture, described in § 5.1.1, does not necessarily map to the physical architecture of the system. For this purpose we have chosen to differentiate in the S-cube KB between software and hardware components. Hardware (also called physical) components correspond to the physical architecture of the system while software components make explicit a functional viewpoint of the system.

We model, with the S-cube KB, the different physical machines (hardware) connected to each network zone. We then associate the physical machines with the services (software) running on them. This distinction between software and hardware allows an appropriate level of detail for which failures and local attacks like physical access are associated with the physical machines and remote cyber-attacks exploiting vulnerabilities are associated with the software component which houses a specific vulnerability.

5.1.4 Modeling control data flows

Finally, we include in the S-cube KB, the modeling of data flows between the software components in the system architecture. Acquisition, control and supervision being the

fundamental functionalities of a SCADA system, we address more specifically control data flows.
After defining the different levels of the system architecture in § 5.1.1 we started modeling the typical components of each level and their contribution to the control dataflow:

- At the field level, sensors measure physical quantities of the industrial process and send measurements to the process controller. Actuators receive instructions from the latter and act accordingly on the industrial process. To keep the S-cube KB as generic as possible and representative of control systems in any industrial domain, we have chosen not to model a specific industrial process. The KB can however be coupled with other DSLs that describe the dynamics and behavior of a specific industrial system.
- At the process level, the process controller (e.g., PLC, RTU) receives measurements from the sensors, processes data and sends instructions to actuators, if necessary. On the other hand, it sends feedback on the process status to the supervision station and potentially receives instructions from it. In some architectures, process controllers can exchange orders/feedback with one another;
- At the supervision level, the operator station receives feedback from different process controllers, providing a centralized view of the physical process, and sends back instructions;
- The IT level initially models systems that enable the optimization and management of the business process. Such systems are not supposed to have a direct impact on the control process. Yet, as modern controllers and SCADA servers integrate information and communication technologies (e.g., ftp servers, TCP/IP based communications), we make both the supervision and process levels inherit the IT characteristics.

We also distinguish between two types of control data[3]:

- Instructions: this data is sent by system components having a control functionality (such as process controller, supervisor, etc.) to field devices in order to execute an order;
- Feedback: this data is sent by field devices to acquisition, control and supervision components in order to report a status of the system; it can be either a measurement or an alarm.

In addition to this level of detail, the S-cube KB models the data flow direction with respect to a given component; whether it is an input flow (data received) or an output flow (data sent).

We discussed in Section 3.2 the specificities and challenges of ICS in terms of time criticality and stressed the importance of data availability and integrity of control data flows. Indeed, control data flows should be available and unaltered in order to ensure the normal operation of the industrial process. Otherwise, data alteration or unavailability can result in safety-related issues. In the S-cube KB, we study and propagate the effects of attacks and failures on the data flow's integrity and availability. For example, a jamming attack on a wireless network would lead to unavailability of all data flows carried by this network.

We show in the next section how the different aspects described above have been aggregated into the S-cube KB by explaining the metamodel used to build this KB.

[3] This distinction was added in the KB for more accuracy on attacks impacting data flows in the process and field levels.

5.2 Metamodel

The S-cube metamodel, depicted in Figure 3, gives an overview on the hierarchy of classes modeled in the S-cube KB. It models the typical components of digital industrial architectures. Each class is represented with a “box” and models a system element template. The S-cube KB adopts the Figaro modeling language (cf. Section 8.1). Being object oriented, Figaro allows knowledge to be structured using the inheritance mechanism and the metamodel to be built progressively. The latter can also be extended in order to refine details about the system.

Each class of the metamodel is associated with its attributes as well as the attacks and failure modes likely to happen on it. Attributes are represented in Figure 3 by small horizontal rectangles. The type of these attributes is put in brackets or braces for enumerated types. The attributes for which the type is not mentioned are Boolean (can have either the value True or False). The gear wheels icons model the dynamic behavior of the attack steps and failure modes associated with each class.

We explain in the following paragraphs the main steps followed for building the knowledge base. We stress the key modeling elements of the S-cube metamodel and the assumptions made in order to have the appropriate level of detail. This level of detail has been chosen in a way that ensures a compromise between the generality of the knowledge base and the relevance of the results obtained. A too coarse level of detail, which is the case of generic approaches identified in [3], does not provide sufficient knowledge about the attack and failure scenarios. On the other hand, a too fine level of detail would make it difficult to update models and result in combinatorial explosion in their processing. We have chosen in the S-cube KB to cover accidental and malicious risks with a level of detail able to provide an idea of the risk scenarios. More detailed models can be used for analyzing in depth security issues like access control [18] or cryptographic protocols [8].

In the explanations below, classes modeled in the knowledge base and the associated attributes are put in *Italic* font.

We first model, with the generic class *component*, a system component which can fail accidentally or be compromised by an attacker. Accidental failures can be repaired by maintenance actions (cf. Section 6.1). The classes *network_zone* and *physical_cpt* inherit the characteristics of the mother class *component*, and model respectively a network zone (cf. § 5.1.2) and a physical component (cf. § 5.1.3). The physical component models a machine (hardware) connected to a network zone and that hosts one or many software components (*software_cpt*). Identical physical components that can fail simultaneously due to a common cause are associated with the same *CCF_group* (cf. § 6.1.2).

Following the system decomposition presented in § 5.1.1, we make the distinction in the S-cube KB between field system components, process system components, supervision system components and finally IT system components. They model generically the physical machines of each system level. Actuators and Sensors are field system components, while *process_controller* (e.g., Programmable Logical Controller) is one of the process system components. An IT system component (*IT_sys_cpt*) models a physical machine integrating advanced IT typically running an operating system (*OS*) and hosting IT software components. As discussed in § 5.1.1, the *process_controller* inherits from the *IT_sys_cpt* class.

Furthermore, we model the following software components:

- a sensor software component (sensor_soft_cpt)models the software capturing and reporting the physical measurements;
- an actuator software component (actuator_soft_cpt) models the software receiving and executing the process controller instructions;

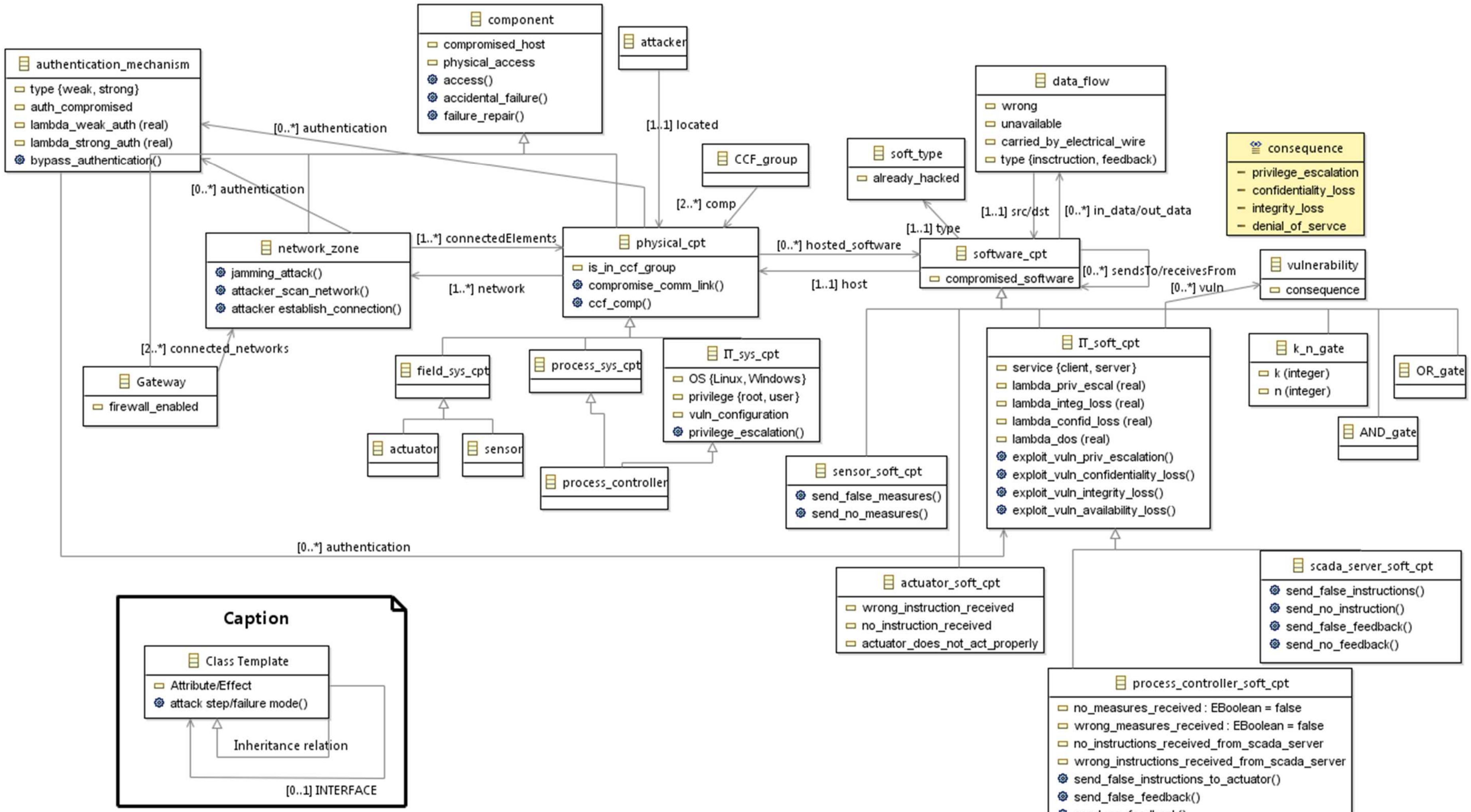

Figure 3: The S-cube metamodel

- a process controller software component (*process_controller_soft_cpt*) models the software receiving and processing sensors measurements, and sending orders to actuators;
- a scada server software component (*scada_server_soft_cpt*) models the software supervising the process controllers, through receiving feedback and sending instructions;
- an IT software component (*IT_soft_cpt*) models a software component from the IT domain and not directly used for control purposes (e.g., ftp client, http server).

Software components exchange data flows (*data flow)* (cf. § 5.1.4). With S-cube, the user models graphically only the legitimate data flows allowed by firewalls. The firewalling functionality is enabled or disabled by the *gateway* binding two or many networks.
An *IT_soft_cpt* can host one or many vulnerabilities. Each *vulnerability* has one or many *consequences* among the following: privilege escalation, confidentiality loss, integrity loss or denial of service.
A vulnerability can be associated with a software component or with a physical machine. In the latter case, the vulnerability models a bad machine configuration (e.g., system files not write-protected) which is assumed to allow privilege escalation when exploited by an attacker.

We assume, in the S-cube KB, that an IT machine is said to be compromised if an attacker manages to have root privileges on it. If that is the case, he/she can compromise all software components running on this machine. This assumption is credible as compromising one software component on a machine does not allow the attacker to compromise all other services unless he/she succeeds in obtaining root privileges.

For IT level networks such as the corporate network, we are interested in the attack propagation between different IT level machines until some component having a control action on the process is reached. When reaching the control network, we are more interested in data integrity/availability as the modification or unavailability is generally the main reason leading to undesirable events.
Access Control is modeled by associating an *authentication_mechanism* with a machine (e.g., a login/password is required in order to log into the OS), a network (e.g., WEP/WPA2 authentication) or an application (e.g., ftp server needs to authenticate with the ftp server in order to read/write files).
In the following section, we give the taxonomy of the different attack vectors embodied in the S-cube KB.

5.3 Taxonomy of attacks

The S-cube KB has been built upon a taxonomy of attack vectors that allows reasoning about attacks at a higher level rather than a simple list of vulnerabilities, which guarantees the coverage of all types of attacks. The hierarchical thinking methodology was adopted when building the knowledge base; it consists in starting at first from a high level of abstraction and progressively refining the levels of detail. The fundamental mechanism of abstraction allows us to deal with the complexity of systems and the myriad of vulnerabilities they are subject to.
Our experience with BDMP applied on the industrial use cases like the pipeline example described in [19] revealed some patterns of typical attacks on the control and field levels. The attack taxonomy in the S-cube KB was inspired from these patterns, but also from other

existing DSLs like CySeMoL, MulVAL (cf. Section 2.1) and the Ethical Hacking and countermeasures Courseware (CEH) [20].
The CEH courseware [20] states the five following steps required for a successful attack:

1. Foot-printing and reconnaissance;
2. Scanning;
3. Gaining access;
4. Maintaining access;
5. Clearing track.

We map these steps with what has been modeled in the S-cube KB. The foot-printing and reconnaissance phase can be modeled in two ways: either by the attack step "preparing for the attack" associated with the *attacker* template, or by the attack step "*access*" associated with a physical component in the system. For this second case, the rate of this attack step combines the frequency of attack occurrence and the time needed for the attacker to collect information about the target network. The scanning step (Step 2) is included in the different paths used by the attacker to access the system. The different ways for the attacker to gain access to the system (Step 3) are given hereafter. Steps 4 & 5 are not relevant in our context, as we are interested in successful attacks leading to safety issues; these steps are consequently not modeled by S-cube KB.
The taxonomy of attack vectors used in the S-cube KB addresses the different entry points used by the attacker in order to access the system. We believe that the attacker should initially have some sort of physical access, whether local or remote, to a machine or a network zone related to the system architecture in order to try some attack scenario. We considered the following entry points (EP) from which an attacker can gain access to the system:

- EP1: physical access to the network. If the network is wired (e.g., Ethernet-based), and the attacker has physical access, he/she can plug into the network and manage to connect to the switch/hub via the wired link (e.g., Ethernet cable). If the network is wireless, the attacker should be able to capture the network traffic; which may require that he physically moves next to the access point. If the network employs an authentication mechanism, the attacker has to additionally bypass authentication in order to reach other machines connected to the same network. The network is said to be compromised;
- EP2: physical access to a machine connected to the network. If the attacker has physical access to a machine connected to the network and he manages to bypass authentication, if it is employed (by the machine OS and/or by the network), the machine is said to be compromised. If a network hosts a compromised machine it is also said to be compromised, which means that the attacker can reach (i.e. send packets to) any other machine connected to the same network;
- EP3: the network is remotely reachable via another compromised network. If the network is connected to a gateway which does not enable firewalling and which is connected to another compromised network, the attacker can then scan the network in order to identify live systems and open ports (e.g., ICMP scanning). He can then either:
    - EP 3.1: access via a vulnerable service. If there is vulnerable software which is reachable via compromised software communicating with it or which is located on a compromised machine or a compromised network zone, then the attacker can exploit this vulnerability in order to penetrate the system;
    - EP 3.2: access via a vulnerable machine. If a machine is not correctly configured or is running a vulnerable OS and it is reachable by the attacker (it is located on a compromised network or running a compromised software or the attacker can

physically access the machine) then the attacker can exploit this vulnerability in order to gain root privileges.

The network is said to be compromised if the attacker can reach any machine connected to the network zone. We assume that if the attacker can access the network (through the vectors listed above) than he can reach any machine connected to the same network.

We gave in this section the different entry points that can be used by the attacker to access the system. The attack step “access” associated with a physical component of the system (which can be a machine of a network zone) is the initiating vector in attack scenarios generated by the quantitative analysis. The remaining steps describe how the attack propagates given the system configuration.

We give more details in the next two sections on the qualitative and quantitative aspects included in the S-cube KB.

6. Qualitative aspects in the S-cube KB

The S-cube approach has advantages both for building system models and processing them. First the system architecture is modeled using the templates corresponding to classes defined in the S-cube KB (cf. metamodel in § 5.2). This model can be either graphical or textual. Next the KB is instantiated on the system model and the resulting instantiation is processed with quantification tools which yields a qualitative and quantitative analysis.

The qualitative part of the analysis consists in generating the attack and failure scenarios likely to happen on the system model and that can lead to an undesirable event initially set by the user. The quantitative part, depicted in Section 7, allows these scenarios to be sorted according to their decreasing probabilities and gives an estimation of the undesirable event probability. This undesirable event represents the ultimate risk leading to safety issues and that we want to avoid for the system. We therefore exclude from the scope of our analysis stealthy attacks that only decrease the performance of the system or have marginal consequences on the system.

Ideally, the assessment of a cyber physical system should take into account the failures and attacks of the control system via their effects on the physical process (since safety issues can arise only because of an unwanted behavior of the physical part). This would require dedicated models for each use case. For example in [21], the effects of attacks on power grids are examined. Since we wanted to focus on the control part, for which we could define generic models, we only distinguish two categories of effects on the physical system: the fact that actuators receive incorrect (i.e. falsified in the case of an attack) instructions or no instructions at all. It is then a pessimistic approximation to consider that one (or both) of these malfunctions inevitably lead to an undesirable event, from a safety perspective.

In the remainder of this section, we give the types of failures and attack steps modeled in the S-cube KB. The risk scenarios output by S-cube are built from these generic failure modes and attack steps.

6.1 Failure modes and repair

We address in the S-cube KB accidental failures which can be either independent or dependent.

6.1.1 Failure in operation (independent events)

The S-cube KB models for each system component the accidental failure in operation which may occur randomly and independently from other components failures. This failure is associated with network zones and different physical machines (i.e. hardware failures). We assume in the current version of the knowledge base that software always functions in a deterministic and dependable way if not altered by third-parties (which excludes software bugs and crashes from our scope).

6.1.2 Common cause failures (dependent events)

A Common Cause Failure (CCF) is the failure of multiple components that result from a single cause, like for example a fire, a flood, an earthquake, etc. This cause is shared by a given set of components and can be related for instance to the design, the software, the environment, etc.

In safety-critical systems, redundancy is often introduced to improve reliability. However, the intended effect may be reduced when components are subject to common cause failures. According to expert judgment, CCFs account for 1 to 10% of a component's failure rate [22]. It is consequently important to consider this kind of failure in the safety analysis in order not to overestimate the system reliability.

We model in the S-cube KB CCFs which we associate with physical components, and which represent dependent failures that may occur at the same time or within a short time interval, due to a shared cause.

6.1.3 Repair actions

The S-cube KB includes the modeling of maintenance actions aiming at repairing the accidental failures of the physical components. Once repaired, these resume their normal operation.

We present in the next sub-section the attack steps modeled in the S-cube KB. An attack scenario generated by S-cube will consist of one or several attack steps among the following.

6.2 Attack steps

In addition to failure modes previously described, we summarize in Table 1 the attack steps associated with each class as described in the metamodel in Figure 3. Classes describing physical components are in blue and classes corresponding to software components are in green.

Class	Attack steps / Failure modes
component (generic class)	Accidental failure: models an accidental failure, in operation, of a given system component (cf. § 6.1.1); Failure repair: models the repair of the accidental failure of a system component; Access: models physical access of the attacker to the component.
network_ zone (inherits from component)	Jamming attack: models a jamming attack on a wireless network; Scan network: models the attacker scanning the network in order to identify live systems and open ports; Establish connection: models the attacker establishing illegitimate connection with an open port; Bypass authentication: models the attacker bypassing authentication to the network. We distinguish between weak and strong authentication.
physical_cpt (inherits from component)	Common Cause Failure: models the failure of the physical component due to a common cause (cf. § 6.1.2); Compromise communication link (Man In The Middle attack): models the attacker compromising the communication link between two machines;

	Bypass authentication: models the attacker bypassing authentication to the OS of a physical machine. We distinguish between weak and strong authentication.
IT_sys_cpt (inherits from physical_cpt)	Privilege escalation: models the attacker exploiting a bad configuration or a vulnerability related to the OS in order to escalate privileges.
process _controller (inherits from IT_sys_cpt)	No attack or failure specific to this class.
sensor (inherits from physical_cpt)	No attack or failure specific to this class.
actuator (inherits from physical_cpt)	No attack or failure specific to this class.
software_cpt (generic class)	No attack or failure specific to this class.
IT_soft_cpt (inherits from software_cpt)	Bypass authentication: models the attacker bypassing authentication required by an IT software component (e.g., ftp server). We distinguish between weak and strong authentication; Exploit vuln priv escalation: models the attacker exploiting a vulnerability that results in privilege escalation; Exploit vuln integrity loss: models the attacker exploiting a vulnerability that results in integrity loss; Exploit vuln denial of service: models the attacker exploiting a vulnerability that results in denial of service; Exploit vuln confidentiality loss: models the attacker exploiting a vulnerability that results in confidentiality loss.
scada_server_soft _cpt (inherits from IT_soft_cpt)	Send false instructions: attacker falsifies instructions sent from SCADA server software;
	Send no instructions: attacker removes instructions sent from SCADA server software;
	Send false feedback: attacker falsifies feedback sent from SCADA server software;
	Send no feedback: attacker removes feedback sent from SCADA server software.
process_controller _soft_cpt (inherits from IT_soft_cpt)	Send false instructions to actuator: attacker falsifies instructions sent from process controller software; Send no instructions to actuator: attacker removes instructions sent from process controller software; Send false feedback: attacker falsifies feedback sent from process controller; Send no feedback: attacker removes feedback sent from process controller software.
sensor_soft_cpt (inherits from software_cpt)	Send false measurements: attacker falsifies measurements sent from sensor; Send no measurements: attacker removes measures sent from sensor.
actuator_soft_cpt	No attack or failure specific to this class.

(inherits from software_cpt)	

Table 1: Failure modes and attack steps modeled in the S-cube KB

The attack steps modeled, so far, in the S-cube KB have been discussed with security engineers. They are consistent with the level of detail at which we have decided to stop. This list is not, in fact, exhaustive; the knowledge base can be further extended with other “categories” of attacks and existing attack steps can be decomposed into more detailed attack steps.
We explain in the next section how accidental and malicious scenarios are generated from the system architecture and the S-cube KB.

6.3 Attack and failure scenario generation

After the system architecture is described, the S-cube KB is instantiated on it. This instantiation generates a textual model, which constitutes a virtual definition of the state space of the system (all the states in which the system can be). This state space is defined locally, by the list of possible transitions from any state and the states they lead to.

The textual model can be explored in two ways:

- Using a path-based exploration algorithm, the state space is explored step by step. Starting from the initial state, we explore the tree of all possible paths in a depth left first manner. The exploration of one path is terminated if one of the following cases is reached: the targeted state, an absorbing state or one of the truncating criteria. The principle of this algorithm is illustrated in Figure 4. If the explored state graph is in fact a Continuous Time Markov Chain (CTMC) then probabilities calculated analytically can be associated with sequences; this gives on one hand a relevant criterion to eliminate most sequences, which makes the exploration tractable, and on the other hand an estimation of the probability of reaching a target state before a given time.
- Using the Monte Carlo method, which allows processing of any problem having a probabilistic interpretation. Based on the law of large numbers, this method simulates many histories of the system using repeated random sampling. These histories yield independent and identically distributed realizations of a numerical variable of interest. By calculating the average of these realizations, one obtains an estimator of the mean of this variable’s distribution. If the variable is Boolean, its mean value is equal to the probability that the variable takes the value 1. This is how the probability of the system being in a target state can be estimated.

The qualitative analysis exhaustively generates all the scenarios leading to the undesirable event specified. We can distinguish three kinds of possible scenarios:

- Purely accidental scenarios: which consist only of accidental component failures;
- Purely malicious scenarios: which consist only of attack steps;
- Hybrid scenarios: which consist of a mixture of accidental failures and attack steps.

The number of scenarios can easily be huge and unmanageable especially for large systems. In order to be exploitable, the qualitative results should be associated with some quantitative parameters that enable sorting and prioritization of the most probable scenarios. In the next section, we give the hypotheses taken for the safety and security metrics associated with failure modes resp. attacks modeled in the S-cube KB and that are the basis of the quantitative results obtained.

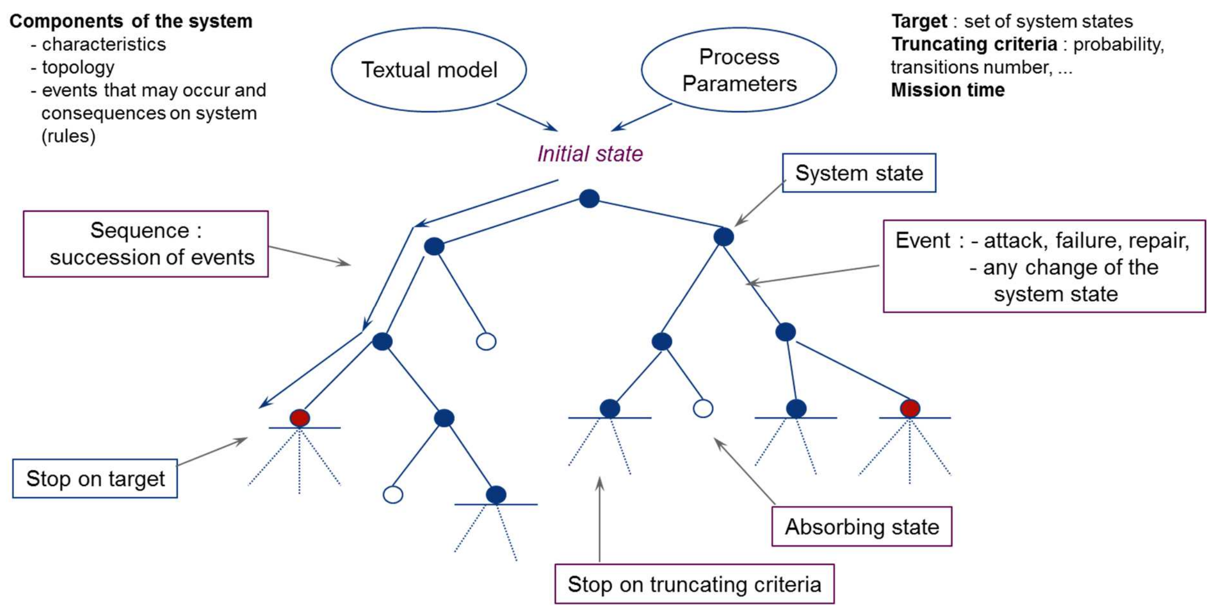
Components of the system
- characteristics
- topology
- events that may occur and consequences on system (rules)
Textual model
Process Parameters
Target : set of system states
Truncating criteria : probability, transitions number, ...
Mission time
Initial state
System state
Sequence : succession of events
Event : - attack, failure, repair, - any change of the system state
Stop on target
Absorbing state
Stop on truncating criteria

Figure 4: Sequence exploration principle

7. Quantitative aspects in the S-cube KB

We have already shown in [19] the advantages of building a common probabilistic model for safety and security. In a similar vein, S-cube offers a quantitative framework, based on probabilities, for assessing accidental and malicious risks. Each attack step and failure mode defined in the S-cube KB is associated with a security respectively safety metric as detailed later in this section.

7.1 Safety metrics

We explain the safety metrics associated with the failure modes described previously in Section 6.1.

7.1.1 Independent accidental failures

In dependability analysis, the system (or component) reliability corresponds to its ability to perform a required function, under given environmental and operational conditions and for a stated period of time.

In the S-cube KB, we adopt the exponential approximation for the Time To Failure (TTF) of a system component in operation. The safety metrics used in this case correspond to failure rates (λ) of components. The Mean Time To Failure (MTTF) of a component is equal, in this instance, to the inverse of the failure rate λ.

Data related to the failure rates can be obtained from the experience feedback on the system components' failures or from the manufacturer's documentation. Experimental data may take a long time to be made available (the range of MTTF generally is from years to decades), especially for systems with a long service life.

7.1.2 Common Cause Failures

The Common Cause Failures (CCFs) have been addressed in the probabilistic risk analysis with different models. These models use feedback of experience data in order to quantify the probabilities of events causing the failure of a specific group of identical of components. The transition between these models and the CCF model used in static (i.e. essentially made with fault trees) analyses and called the Basic Parametric Model [23] is then possible. The latter is used to evaluate the probability of the different combinations of components failures within the same CCF group.

In the S-cube KB, we adopted the dynamic generalization of the Basic Parametric Model as described in [24]. The number of combinations of components common cause failures, can be quickly significant with large groups of components. For reasons of simplification, we have chosen to model in the S-cube KB only groups of two or three components.

In the S-cube KB, the numeric values of safety metrics were assigned based on estimations of our safety experts or based on existing studies from the literature [24][22].

In the next section, we address the quantitative aspects related to security modeling in the S-cube KB.

7.2 Security metrics

We make the assumption that Times To Success (TTS) are, also, exponentially distributed. Hence, security metrics used in the S-cube KB are the success rates (defined similarly to the failure rates) of the attack steps described in Table 1. However, it is easier for experts to reason in terms of Mean Time To Success (MTTS) that are simply the inverse of success rates.

Unlike for failures, feedback on attacks is not easy to obtain. Industrial companies often refuse to communicate about their experience with attacks as this can infringe their image. Quantitative security data are consequently less available and can be subject to large uncertainties. Security metrics are, in addition, intimately linked to the attacker's profile and behavior which is difficult to predict.

Primarily, there are mainly two approaches for security quantitative assessment: approaches based on scoring like for instance the Common Vulnerability Scoring System (CVSS) [25] or the nSHIELD framework [26], and approaches based on probabilistic assessment [27][28] [29].

We give in the following a quick overview on the CVSS framework and the McQueen's quantitative model and explain how we have exploited both of them to assess the security metrics related to the attack steps modeled in the S-cube KB.

7.2.1 The Common Vulnerability Scoring System

The NIST [25] introduced the Common Vulnerability Scoring System (CVSS), an open framework for scoring IT vulnerabilities. The CVSS score ranges from 0 to 10; the higher it is, the more critical the vulnerability.

The CVSS score is calculated according to an equation as a function of the following base metrics:

- Access Vector (AV): reflects how the vulnerability is exploited: locally, with adjacent network access or remotely. The more remote an attacker can be to the target of the attack, the greater the vulnerability score;
- Access Complexity (AC): measures the complexity of the attack required to exploit the vulnerability once an attacker has gained access to the target system. The access

complexity can be high, medium or low. The lower the required complexity, the higher the vulnerability score;

- Authentication (Au): measures the number of times an attacker must authenticate to access a target in order to exploit a vulnerability. This metric can take the following values: multiple, single or none. The fewer authentication instances are required, the higher the vulnerability score.
- Confidentiality Impact (C): measures the impact on confidentiality of a successfully exploited vulnerability. It can be complete, partial or none;
- Integrity Impact (I): measures the impact to integrity of a successfully exploited vulnerability. It can be complete, partial or none;
- Availability Impact (A): measures the impact to availability of a successfully exploited vulnerability. It can be complete, partial or none;

7.2.2 The McQueen's model

McQueen [29] proposed a Markovian model for estimating the Time To Compromise (TTC) a computer system through exploiting a given vulnerability. The authors model the TTC, which they define as the measure of the effort expended by an attacker for a successful attack. They also assume that effort is expended uniformly, as a random process composed of three attacker sub-processes:

- Process 1: is when at least one vulnerability is known and the attacker has at least one exploit readily available. The probability that the attacker is in process 1 is given by equation (4):

$$P_1 = 1 - e^{-Vm/k} \qquad (4)$$

Where V is the number of vulnerabilities on the component of interest, m is the number of exploits readily available to the attacker, and k is the total number of vulnerabilities.
Assuming that the attacker is familiar with at least one of the available vulnerabilities and has experience with at least one exploit associated with the known vulnerabilities, the authors estimate the time required for Process 1 with t_1=8 hours.

- Process 2: is when at least one vulnerability is known but the attacker does not have an exploit readily available. Since Process 1 and Process 2 are mutually exclusive, the probability that the attacker is in process 2 is given by equation (5):

$$P_2 = 1 - P_1 = e^{-Vm/k} \qquad (5)$$

  The mean time needed to complete Process 2 is modeled as the expected value of the number of tries times 5.8 days: $t_2 = 5.8 * ET$; where ET is the expected number of tries;

- Process 3: is when the attacker identifies new vulnerabilities and exploits (zero-days). The time estimated for this process is given by equation (6):

$$t_3 = \left(\frac{V}{AM} - 0.5\right) * 30.42 + 5.8 \qquad (6)$$

  Where AM is the average number of the vulnerabilities for which an exploit can be found or created by the attacker;

Assuming that the three processes are mutually exclusive (Process 3 only applies if Processes 1 and 2 do not apply or are unsuccessful), the overall TTC is given by equation (7):

$$T = t_1 * P_1 + t_2 * (1 - P_1) * (1 - u) + t_3 * u * (1 - P_1) \qquad (7)$$

Where $u = (1 - (\frac{AM}{V}))^V$: the probability that Process 2 is unsuccessful (*u=1* if V=*0*).

We explain in § 7.2.3, how we used McQueen's [29] model in order to assess the MTTS for exploiting a vulnerability in a software system component.

7.2.3 Assumptions for security metrics

The S-cube KB includes modeling the CVSS base metrics as described below:

- (AV): the different access vectors are modeled as described in Section 5.3;
- (AC): the access complexity is included in the MTTS assessment;
- (Au): authentication is modeled by associating an authentication mechanism with a service, a network or a host. We also model whether the authentication is weak or strong. The MTTS of the attack step "bypassing authentication" is higher when a strong authentication is in place;
- (C, I, A): the impact on confidentiality, integrity or availability is associated with each vulnerability, by security experts at the system description phase. As explained in Section 3.2, confidentiality is not too important in the industrial context and could not lead to safety issues. Vulnerabilities having as consequences: "privilege escalation", "integrity loss" or "denial of service" are the most relevant when it comes to safety-related risks and their impact on data flows integrity and availability are propagated throughout the system model.

Other metrics that impact the MTTS are the attacker's profile (expert, intermediate, and beginner) and the resources (money) he/she is ready to invest in the attack. In order to meet safety requirements, we make, in the S-cube KB, the pessimistic assumption that the attacker is an expert and holds unlimited resources to achieve the attack.

In S-cube KB, we considered two different random variables corresponding to TTS:

- TTS associated with the attack step *access* (*TTS_access*); which corresponds to the time until the system is accessed by an attacker. As discussed in Section 5.3, the attacker needs first to have some sort of access to the system in order to try some attack scenario. *MTTS_access* is the mean time required for an attacker to access the system;
- TTS associated with other attack steps given in Table 1; which corresponds to the time required for the attacker to achieve a given attack step;

We have chosen to use, in the S-cube KB, the exponential distribution to model the TTS for all the attack steps. We defend this choice below.

For the time after which the system is accessed by an attacker (*TTS_access*), the empirical results obtained by Holm in [28] (cf. § 7.2.2) show that the exponential distribution is relevant if *TTS_access* <600 days. Furthermore, the Grigelionis theorem given in [30] proves the relevance of the exponential distribution if we make the following assumptions:

- Each attack scenario can be approximated by a point renewal process $T^{n,i} = (T_k^{n,i})_{k\geq 0}$ $(1 \leq i \leq n)$; with an initial delay time $T_0^{n,i}$;
- All delay times $T_0^{n,i}$ are independent from one another; which means that attacks that may target the system are independent from one another;
- On the time scale, these attacks can superpose and each of the n processes (n is large because it corresponds to the number of potential attackers) has a small contribution;

Given these assumptions, the theorem stipulates that the superposition of the n independent renewal processes converge towards a Poisson point process. We can consequently infer that the random variable corresponding to the minimum of the delay times $T_0^{n,i}$, corresponding in our context to the *TTS_access*, follows an exponential distribution.

For the other attack steps, the use of the exponential distribution is not always approved for security assessment. However, we have chosen this assumption because it makes it possible to use Figseq (cf. Section 8.1) for the qualitative analysis (Figseq works only with Markovian models).

7.2.4 Practical security metrics values

As previously mentioned, the McQueen's [29] model (cf. § 7.2.2) has been used in order to estimate the MTTS associated with attack steps modeling the attacker exploiting a vulnerability of a software component in the S-cube KB. In order to use the formula (7) of this model, we extracted statistical data[4] on vulnerabilities from the Common Vulnerabilities and Exposures (CVE) dictionary [31]. The *k* parameter in McQueen's model corresponds to the total number of vulnerabilities and the *m* parameter corresponds to the number of exploits publicly available.

As previously discussed, vulnerabilities are categorized in the S-cube KB, into three main categories, according to their consequences and impact on confidentiality (C), integrity (I) and availability (A):

- Vulnerabilities resulting in privilege escalation (have an impact on C, I and A);
- Vulnerabilities resulting in integrity loss (have an impact on I);
- Vulnerabilities resulting in denial of service (have an impact on A).

We give, in Table 2, the data obtained from the CVE dictionary[5] [31] corresponding to *k* and *m* parameters for each type of vulnerability. We explain below how we proceeded to get this data.

Vulnerability type	Number of total vulnerabilities (k)	Number of total exploits publicly available (m)
Privilege escalation	3388	184
Integrity loss	2222	60
Denial of service	14791	654

Table 2: Statistical data on vulnerabilities sorted by type

For vulnerabilities resulting in privilege escalation (cf. first line of Table 2), data extracted correspond to the type "Gain privilege" in [31].

For vulnerabilities resulting in integrity loss (the second line of Table 2), we consider from [31] data of type "Gain information" and having a complete or partial impact on integrity. We consider particularly vulnerabilities having a CVSS score >5; given that for vulnerabilities with a CVSS <5 the impact on integrity is "None".

For vulnerabilities resulting in denial of service (the third line of Table 2), we took the data corresponding to "DoS" in [31].

We also made the assumptions given in Table 3 when using the McQueen's [29] model.

Assumption	Rationale
V = 1	In each "exploit vulnerability" attack step, the attacker exploits just one vulnerability
AM/V = 1	We assume the attacker's skill level is "expert"
t_1= 1	The time needed for an expert to exploit a known vulnerability with an exploit readily available is one working day
T=1	The expected number of tries is equal to one; i.e. the attacker tries to exploit a vulnerability just once and abandons in case of an unsuccessful attempt

Table 3: Assumptions taken for MTTS evaluation using McQueen's model

[4]We took this data from the CVE dictionary in August 2015

[5]Data has been collected since 1999

Given these assumptions, the MTTS obtained using the equation (7) and data in Table 2 is approximately five days for all types of vulnerabilities (1/MTTS ~ 0.01 h^{-1}).
For other kinds of attack steps modeled in the KB, the MTTS was estimated by our security experts.

In the next section, we show the possible quantitative analyses that S-cube enables to generate.

7.3 Possible quantitative analyses

The results obtained are based on the qualitative and quantitative aspects in the S-cube KB previously described. Below, these results are also called qualitative and quantitative analysis. The MTTS associated with the initiating "access" attack step (*MTTS_access*) can be parameterized in two different ways for the two following purposes, which can be complementary:

1) To assess, from scratch, a given architecture in order to pinpoint the different access paths privileged by the attacker and identify the most vulnerable components. In this case, *MTTS_access* is set with the inverse of the frequency of the attacks that target the specific kind of system architecture under study (such information can be obtained in-house from security feedback e.g., log-files). With this kind of study, the qualitative results are the most interesting to analyze.
2) To quantify more precisely the probability of a successful attack scenario given that the attacker has started at t=0 to target the system. In this case, *MTTS_access* is set with the mean time needed for the attacker to have some physical or remote access to the system, assuming that the reconnaissance phase was already performed. With this kind of study, more focus is given to the quantitative analysis. The qualitative and quantitative results provide the attack scenarios with a more accurate estimation of the time needed to complete each scenario.

   In the second kind of study, using a joint model for assessing safety and security risks leads to results that generally promote attack scenarios. Indeed, the MTTFs used for failure modes are very large compared to security metrics. On the contrary, in the first kind of study, it is possible to have the same order of magnitude for both kinds of risk, thanks to the fact that the frequency of targeting a given industrial architecture is indeed low (fortunately); unfortunately, this metric is even more difficult to predict than other security metrics and highly subjective as it is related to human intention.

The quantitative results are aimed at providing operators of the control systems with a measure of the risk associated with potential attacks in order to effectively manage their resources. The results obtained should not be considered as definitive and accurate values. They are based on the assumptions taken for the different attacks modeled in the S-cube KB and related to the level of detail modeled.
In the next section, we explain how the main notions of the S-cube KB above described have been implemented and give an illustration of the approach on a use case.

8. Tool chain and use case

We first present the tool chain associated with the S-cube approach. We next illustrate it on a use case.

8.1 Tool chain

In order to explain how the principles of the S-cube approach have been implemented, we reproduce the Figure 2 given in Section 4, to which we add how each aspect has been implemented in blue italic text. The result is depicted in Figure 5.

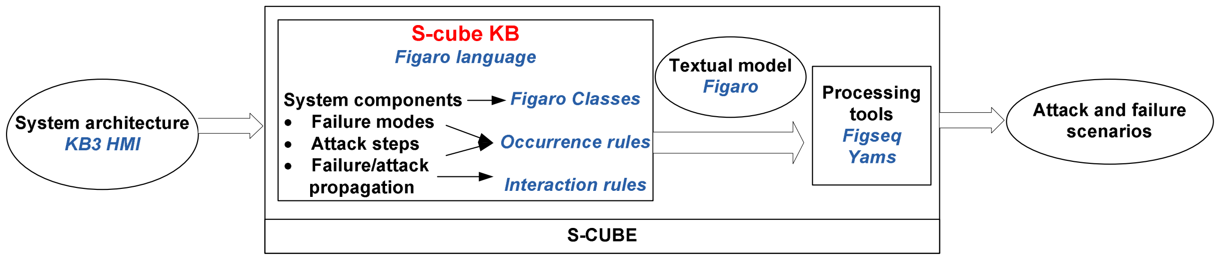

Figure 5: The S-cube tool chain

The S-cube KB had been implemented using the Figaro modeling language [9]. Initially developed for reliability analysis, Figaro is an object oriented language and implements additionally some artificial intelligence notions. Although specific to dependability, Figaro is general enough to be adaptable for other domains and especially for security.
Figaro provides an appropriate formalism for developing knowledge bases with generic descriptions of components. It enables, thanks to the inheritance mechanism, the knowledge to be structured and avoids information redundancy. Each generic system component is described with a class. A class can be compared to a mold which, when filled, gives an object having the shape of the mold and all its characteristics.
A class consists of two parts [32]:

- A purely static and declarative part: where one finds the name of the class, the class(es) from which it inherits characteristics, the other classes with which it interacts, the constant characteristics and the state variables with their domains and initial values;
- A dynamic part: where the behavior of the class is described thanks to two kinds of rules: the occurrence rules and the interaction rules.
  - The occurrence rules: describe elementary events with the conditions governing how they are triggered and the associated probability distributions. If the conditions of an occurrence rule are satisfied, an event can occur:
    - Instantaneously: this is used in order to describe the choice between different instantaneous transitions; each transition is associated with a probability, and the sum of transition probabilities appearing in a given rule must be equal to 1;
    - After a time that follows a given probability distribution: in this case the type of the distribution and its parameters are associated with the transition;
  - The interaction rules: aim to propagate the effects that are the immediate and certain consequences of an event (i.e., the firing of a transition of an occurrence rule) in the system.

During the S-cube KB development, the Figaro classes have been used to describe the generic components included in industrial information architectures and their main characteristics as already introduced in Section 5.2. For each class, occurrence rules are used to model security (attacks) and safety (failures) events that may happen to each system component (cf. Section 6). These rules contain also the probability distribution of the time after which the event will happen (cf. Section 7). For each class, the interaction rules model the propagation of the instantaneous effects within the whole system architecture, for instance how the

compromission or the failure of one component impacts other system elements (e.g., data no longer available). We give in annex an excerpt of the class that models a network zone as described with Figaro in the S-cube KB.
The choice of the Figaro modeling language led us to use the KB3 workbench [32] that enables Figaro-based models to be built and processed. The KB3 tool enables graphical elements to be associated with the different classes defined in the S-cube KB. These elements are used to build the system architecture using the KB3 Human Machine Interface. Automatic verification of the graphical model is performed and the correctness and coherence of the model input is checked. For instance, if some graphical element cannot accept a certain type of link the user is notified with an error message. The system model can also be described textually but this requires a basic knowledge of the Figaro language and the S-cube KB.
The S-cube KB is instantiated on the graphical model of the system architecture, which generates a textual model. This model implicitly defines a Continuous Time Markov Chain (CTMC), since all timed transitions of the model are associated with exponential distributions. Because of the combinatorial explosion of the states, this CTMC cannot be exhaustively built, but it can be explored in two ways by the following quantification tools, in order to yield qualitative and quantitative results:

- The Figseq tool: explores, step by step (cf. § 6.3), the sequences leading to the undesirable event. Given the mission time and truncation criteria, Figseq computes an estimated value of the undesirable event probability taking into account the contribution of the explored sequences that led to the undesirable event, and gives also a pessimistic value taking additionally into account the truncated sequences. The truncation criteria are specified in the Figseq tool and can be for instance the minimum probability of the sequence or the maximum number of the sequence branches.
  Figseq can be used only in case of a purely Markovian model. The use of exponential distributions only in the S-cube KB allows us to benefit from the mathematical properties of the tool, and in particular the qualitative analysis yielding the attack and failure scenarios and the use of the Harrison [33] technique to compute their probabilities;
- The Yams tool: uses the “analog”[6] Monte Carlo simulation [34] on the system model to compute an estimated value of the undesirable event probability. Any kind of probability distribution can be associated with transitions for this tool. Yams is also able to output a selection of simulated scenarios, but the obtained results are much more “noisy” than those obtained with Figseq; in particular, there is no guarantee that all scenarios with a probability greater than a given threshold can be obtained.

Before processing the Figaro textual model with quantification tools, the user defines an undesirable event (target state). The model processing generates attack and failure scenarios leading to the undesirable event defined, with an estimation of their probabilities.
The attack and failure scenarios are listed in a table (cf. Table 4 for example) and sorted by decreasing contribution to the undesirable event, whose probability is also calculated. These scenarios are composed of the attack steps and failure modes associated with each system component (cf. Section 5.2) described in the S-cube KB.
In the next section, we illustrate the S-cube approach on a case study of a hypothetical industrial system with its control and corporate networks in order to show its applicability and its ability to generate a joint safety and security risk analysis.

[6]Analog means here: without an acceleration technique. The use of such techniques is not easy in the general case where various kinds of probability distributions are used [34].

8.2 Illustration on a case study

In this section, we give the example of an industrial system with the associated control and corporate architecture, where new information and communication technologies are used. We first describe the architecture of the case study then we give the associated risk analysis using the S-cube approach. In this case study, we particularly show how the S-cube approach can be used during the operational phase of the system lifecycle in order to assess the emergent risks and vulnerabilities.

8.2.1 Description of the case study

We consider the system architecture, depicted in Figure 6, which consists of four network zones: the corporate network, a demilitarized zone (DMZ), the process control network and the field network. The corporate network, the DMZ and the process control network are separated by firewalls. The field network comprises the sensors and actuators used to sense and manipulate the industrial process, as well as the Process Controller. The latter communicates with an Acquisition Server via the process control network. The Acquisition Server is used for both collecting the process data and supervising the industrial process. The process data are stored in an ftp server (*http_ftp_server* in Figure 6) placed in the demilitarized zone. An operator workstation, connected to the corporate network, hosts an http client application which uses the data stored in the *http_ftp_server* for statistical and optimization purposes. This system can be considered as a simple example containing all levels of the PERA (cf. Figure 1).

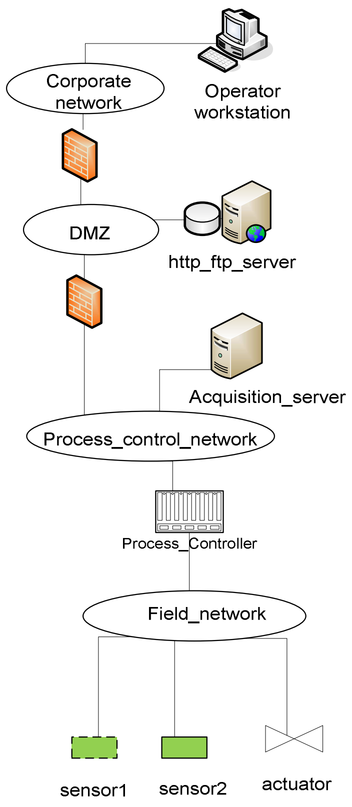

Figure 6: The system architecture under study

We describe the system architecture, as illustrated in Figure 7 using the modeling elements provided by the S-cube KB. The words in *italic* refer to classes used in the S-cube KB (cf. metamodel in Section 5.2) or to the modeling elements used in Figure 7.

As previously discussed in § 5.1.3, S-cube models both the functional and logical architectures. The functional architecture is described by the different machines, the networks they are connected to, and the software components they are hosting (modeled with circles). The logical architecture is addressed by modeling the data flows between the different software components. The graphical representation of Figure 7 is based on some choices that we made in order to have a representation with the main elements but not cluttered by too many graphical links. Some relations between objects are not visible, but of course the user interface allows the user to declare and inspect them in the KB3 tool.

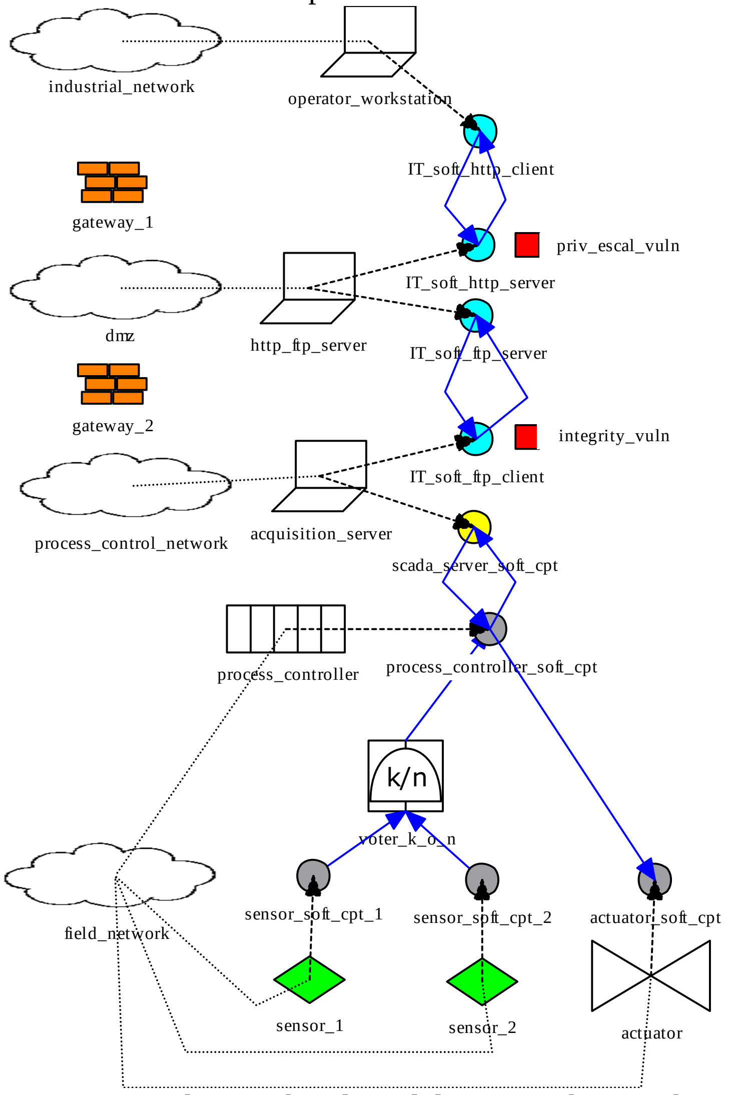

Figure 7: The graphical model as input by S-cube

The connection of a physical machine to a network zone is modeled with a dotted black link (*link_machine_network*). The association between the physical components and the software running on them is modeled with dashed black arrows links (*link_machine_soft*). The solid blue arrows model the allowed data flows between the different software components. The firewall models the filtering policy between the two network zones it separates which implies that only the modeled data flows can be exchanged and no other undefined data flow can be initiated.

The field network comprises sensors: *sensor1* and *sensor2* and the *actuator*. The sensors' measurements are sent from the sensors software components (*sensor_soft_cpt_1* and *sensor_soft_cpt_2*), to a voter (*k/n gate*), which sends the measurement to the *process_controller_soft_cpt* (the process control software running on the

Process_Controller). The latter sends back instructions to the *actuator_soft_cpt* which executes the action on the process. The field network uses a wireless communication to exchange data between the process controller, sensors and actuators.
The *process_controller_soft_cpt* communicates feedback about the process to the SCADA server software (*scada_server_soft_cpt*), running on the acquisition server, and receives back instructions from the operator. The acquisition server also hosts an *ftp_client* that communicates with the *ftp_server* running on the *http_ftp_server* placed in the demilitarized zone. The *http_client* application running on the operator workstation communicates with the http server hosted by the *http_ftp_server*.

We make the following assumptions regarding the architecture under study:

- physical access to the operator workstation is possible;
- the *http_ftp_server* is running with user privileges;
- the *acquisition server* is running with user privileges;

We assume that the following vulnerabilities exist on the architecture and have not been patched:

- a vulnerability exists on the *http_server* and results in privilege escalation;
- a vulnerability exists on the *ftp_client* and results in integrity loss;
- a configuration vulnerability exists on the acquisition server and results in root privilege acquisition.

In order to analyze this architecture with the S-cube approach, we first input the graphical model using KB3. The graphical elements corresponding to the different classes defined in the S-cube KB are used to reproduce the system architecture, depicted in Figure 7. Then before any processing can take place, the model composed by the S-cube classes and the objects input graphically is instantiated in a textual model that implicitly defines a Markov chain. A state of this Markov chain corresponds to a given value for each attribute. The occurrence rules give the possible transitions going out of any state, and the interaction rules propagate the effects of a transition on all attributes, thus defining the state reached after this transition.
In the next section, we process the textual model and analyze the results generated.

8.2.2 Qualitative and quantitative risk analysis

To evaluate the described architecture, we consider that the undesirable event is reached as soon as the attribute *actuator_does_not_act_properly* of the object *actuator* takes the value TRUE. For instance, we can imagine that this architecture is used to control and supervise a chemical plant and that the process controller is supposed to send an instruction to stop heating but the instruction is not sent, which can lead to exceeding temperature limits and result in safety consequences (explosion, human injuries).
We focus in this example on the qualitative behavior described in the various components of the S-cube KB and make simple hypotheses for the occurrence rates of the events that can affect the security or safety of the system. The quantitative analysis produces the following results: after one year of functioning without maintenance (the components are supposed non-repairable in this example) and without considering detection and prevention measures, the probability of the actuator not acting properly reaches 0.57. Of course, this seems very high, but we made the pessimistic assumption that the undesirable event occurs whenever one actuator in the field network receives a wrong instruction or no instruction from the process controller. The malfunction of the heater may not be sufficient to create a safety-related risk and must be combined with malfunctions of other components such as hard-shutdown mechanisms.

The attack and failure scenarios that can lead to this undesirable event are automatically generated using the FigSeq quantification tool. They are generated based on the rules in the knowledge base describing attacks, failures and the rates (inverse of mean time to compromission resp. mean time to failure) of the exponential distribution associated with each rule (cf. Section 7). These scenarios are sorted in a table according to their deceasing probabilities of occurrence and hence their contributions to the undesirable event. We grouped the generated scenarios into minimal cutsets given in Table 4.

Scen. n°	Transition name	Rate	Pr.
1	access(operator_workstation)	1e-4	0.19
	exploit_server_vuln_priv_escal (IT_soft_http_server)	1e-2	
	exploit_server_vuln_integrity_loss (IT_soft_client_ftp)	1e-2	
	privilege_escal.(acquisition_server)	1e-2	
	send_false_instruct_to_process_controller(scada_server)	0.8	
2	access(operator_workstation)	1e-4	0.19
	exploit_server_vuln_priv_escal (http_server)	1e-2	
	exploit_server_vuln_integrity_loss (client_ftp)	1e-2	
	privilege_escal.(acquisition_server)	1e-2	
	send_no_instruct_to_process_controller(scada_server_soft_cpt)	0.7	
3	access(field_network)	1e-4	3.5e-2
	compromise_communication_link(process_controller)	1e-5	
	Send_false_instructions_to_actuator(process_controller)	0.7	
4	access(field_network)	1e-4	3.5e-2
	compromise_communication_link(process_controller)	1e-5	
	Send_no_instruction_to_actuator(process_controller)	0.7	
5	accidental_fail(acquisition_server)	1e-5	3.4e-2
6	accidental_fail(field_network)	1e-5	3.4e-2
7	accidental_fail(process_controller)	1e-5	3.4e-2
8	jamming_attack(field_network)	1e-6	2.9e-3

Table 4: The most probable attack scenarios

Attack scenarios: Scen. n°1 to 4 and 8 are purely malicious as they are composed of only attack steps.

We can see for example that the first scenario (Scen. n°1 in Table 4), which is the most probable one, consists of five attack steps: in the first step the attacker succeeds in having access to the operator workstation (here because the attribute *physical access* of this machine was set to true but the attacker can also have remote access). In the second step, the attacker exploits remotely the existing vulnerability in the *http server* which results in privilege escalation. The *http server* is consequently compromised and the attacker has root privileges on the *http ftp server* machine which enables him to compromise also the *ftp server* running on it. As the *ftp server* communicates with an *ftp client* running on the *acquisition server* (cf. Figure 6), the attacker tries in the third step to remotely exploit the vulnerability in the *ftp client*. The *ftp client* is then compromised. Given that the vulnerability leads only to integrity loss the attacker will also need to make a *privilege escalation attack*, in the fourth step, exploiting the configuration vulnerability related to the acquisition server in order to be able to compromise the *scada server* software. If the attacker succeeds in compromising the latter then he can, finally, send false instructions to the *process controller* which will itself send false instructions to the *actuator*. The latter will consequently not act properly when required which leads finally to safety related consequences.

The second attack scenario (Scen. n°2) is the same as the first one except for the last step. For the latter, instead of falsifying data, the attacker will deny service so that no instructions will be sent to the process controller which will itself send no instructions to the actuator when needed.

Scenarios n°3 and 4 consist of three steps: first the attacker accesses the field network, which is a wireless network with no authentication. Second, he/she compromises the communication link between the process controller and the actuator (man in the middle attack). Third, and finally, he/she falsifies or denies the instructions sent by the process controller to the actuator.

The eighth attack scenario (Scen. n°8) is a jamming attack on the wireless field network. This attack remains with a low probability of occurrence as it requires the attacker to have special equipment and to be at the vicinity of the network access point.

Accidental scenarios: Scen. n°5 to 7 are purely accidental as they are composed of only component failures. These scenarios contain just one failure event (called single point of failure): the failure of the acquisition server, the field network or the process controller will cause instructions not to be sent to the actuator when needed.

Hybrid scenarios: The structure of this system is too simple to give rise to hybrid scenarios, where the *combination* of accidental failures and attacks leads to the undesirable event. This is due to the absence of redundancy for the PLC or acquisition server. If there were such redundancies, we could see scenarios where one of these components is lost accidentally and the other one because of an attack.

We conclude from the results obtained that for the case study architecture given in Figure 6, the acquisition server and the process controller are the most critical components and their failure or compromise can lead to safety related consequences. Mitigation measures in this context would be to deploy redundant components with different technologies in order to provide the main functionalities to control the process in case of unavailability. This would also make attack scenarios more difficult to achieve as the attacker would need to find other vulnerabilities and succeed in exploiting them in order to falsify instructions sent to actuators.

8.2.3 Using S-cube to improve the system architecture

We have modeled in this example attacks that target the corporate networks and then try, by multi-stage multi-hopping, to reach the industrial network in order to interfere with the normal operation of the industrial process.

Security enhancement measures recommended by our security experts would be to inhibit any incoming dataflow towards the process control network. This can be achieved by the introduction of data diodes that allow data to travel only in one direction. Classic firewalls can decide about who initiates the connection; but once the communication channel is established, the data can be exchanged in both directions.

We modified the system architecture in Figure 7 by removing the data flow from the *ftp_server* to the *ftp_client* (unidirectional communication from the *ftp_client* to the *ftp_server*). When the modified architecture with S-cube is processed, the first two attack scenarios (Scen. n°1 and 2 in Table 4) leading to the undesirable event are no longer possible. Scenarios n°3 and 4 are still feasible and are the most probable. In order to mitigate these scenarios, the field network security can be reinforced by deploying a strong authentication

mechanism between the different components communicating through this network. This measure can however be antagonistic with some safety requirements related to time criticality of the instructions sent to the actuators. Another mitigating measure can be to install wired connections in field network. This solution is not only more secure but also reinforces the availability of the data flows carried by the field network. It can however be costly for installation over a wide geographic range.
We have demonstrated in this case study how the S-cube approach can be used to assess the risks related to operational system architectures. In particular, the impact of the new vulnerabilities to which the system may be subject during its exploitation phase can be assessed. The S-cube can also be used in the design phase to compare the safety and security of industrial architectures controlled by modern ICS.

9. Conclusions and perspectives

We presented in this paper the main principles of the S-cube approach, related to modeling notions and the associated qualitative and quantitative aspects, and how they have been implemented.
S-cube provides a risk analysis framework (tool-based approach) to assess the information and control architecture of industrial systems. Thanks to a taxonomy and hierarchical reasoning, we identified the attacks and failure modes these systems are subject to and associated them with quantitative metrics.
The S-cube approach has been implemented with the help of the Figaro modeling language and its associated tools. The system architecture is first modeled graphically by the user, and then processed with the quantification tools. The qualitative analysis provides the scenarios composed of attack steps and failures that lead to a given undesirable event. The quantitative analysis allows these scenarios to be sorted by their probabilities, which makes it easier to exploit the results, and gives an estimation of the undesirable event probability.
By illustrating S-cube on a use case, we showed its ability to model industrial system architectures and yield the associated risk analysis encompassing safety and security issues. An illustration of the S-cube approach on a real and complex use case is given in [5].
Future work will focus on refining the quantitative metrics with feedback of experience on attacks and failures. Detection and reaction to attacks can also be included later in the S-cube knowledge base.

Acknowledgements

This work was supported by Electricité de France and Centrale Supelec who funded this research.

References

[1] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, “Basic concepts and taxonomy of dependable and secure computing,” *IEEE Trans. Dependable Secure Comput.*, vol. 1, no. 1, pp. 11–33, 2004.
[2] S. Sadvandi, N. Chapon, and L. Pietre-Cambacédes, “Safety and security interdependencies in complex systems and SoS: Challenges and perspectives,” in *Complex Systems Design & Management*, Springer Berlin Heidelberg, 2012, pp. 229–241.
[3] S. Kriaa, L. Pietre-Cambacedes, M. Bouissou, and Y. Halgand, “A survey of approaches combining safety and security for industrial control systems,” *Reliab. Eng. Syst. Saf.*, vol. 139, pp. 156–178, Jul. 2015.

[4] S. Kriaa, M. Bouissou, and Y. Laarouchi, “A Model Based Approach for SCADA Safety and Security joint Modeling: S-cube,” in *IET System Safety and Cyber Security*, Bristol, 2015.

[5] S. Kriaa, “Joint safety and security modeling for risk assessment in cyber physical systems,” phdthesis, Université Paris-Saclay, 2016.

[6] H. Holm, T. Sommestad, M. Ekstedt, and L. Nordstörm, “CySeMoL: A tool for cyber security analysis of enterprises,” in *Electricity Distribution (CIRED 2013), 22nd International Conference and Exhibition on*, 2013, pp. 1–4.

[7] X. Ou, S. Govindavajhala, and A. W. Appel, “MulVAL: A Logic-based Network Security Analyzer.,” in *USENIX security*, 2005.

[8] B. Blanchet, “Automatic Verification of Correspondences for Security Protocols,” *J. Od Comput. Secur.*, vol. 17, no. 4, pp. 363–434, 2009.

[9] M. Bouissou, H. Bouhadana, M. Bannelier, and N. Vilatte, “Knowledge modeling and reliability processing: presentation of the FIGARO language and of associated tools,” in *proceedings of SAFECOMP 91*, Trondheim, Norway, 1991.

[10] T. Prosvirnova, “AltaRica 3.0: a Model-Based approach for Safety Analyses,” phdthesis, Ecole Polytechnique, 2014.

[11] P. H. Feiler, D. P. Gluch, and J. J. Hudak, “The Architecture Analysis & Design Language (AADL): An Introduction,” CMU/SEI-2006-TN-011, Feb. 2006.

[12] M. Bouissou and J.-L. Bon, “A new formalism that combines advantages of fault-trees and Markov models: Boolean logic driven Markov processes,” *Reliab. Eng. Syst. Saf.*, vol. 82, no. 2, pp. 149–163, Nov. 2003.

[13] M. Bouissou and C. Seguin, “Comparison of the modeling languages AltaRica and Figaro,” in *In proceedings of the 14th congress on reliability and maintenability (IMDR)*, Lille, France, 2006.

[14] J. Brunel *et al.*, “Performing Safety Analyses with AADL and AltaRica,” in *International Symposium on Model-Based Safety and Assessment*, 2017, pp. 67–81.

[15] J. T. Williams, “A Reference Model For Computer Integrated Manufacturing (CIM) A Description from the Viewpoint of Industrial Automation,” 1989.

[16] J.-M. Brun, L. Platel, and F. Tea, “Cyber Security of Industrial Control System Why ICS specificity lead to Cyber Security Challenge?,” in *C&ESAR*, 2013.

[17] International Electrotechnical Commission, *Enterprise Control System Integration -- Part 1: Models and terminology*. 2013.

[18] T. Lodderstedt, D. Basin, and J. Doser, “SecureUML: A UML-Based Modeling Language for Model-Driven Security,” in *≪UML≫ 2002 — The Unified Modeling Language*, J.-M. Jézéquel, H. Hussmann, and S. Cook, Eds. Springer Berlin Heidelberg, 2002, pp. 426–441.

[19] S. Kriaa, M. Bouissou, F. Colin, Y. Halgand, and L. Pietre-Cambacedes, “Safety and Security Interactions Modeling Using the BDMP Formalism: Case Study of a Pipeline,” in *Computer Safety, Reliability, and Security*, vol. 8666, A. Bondavalli and F. Di Giandomenico, Eds. Cham: Springer International Publishing, 2014, pp. 326–341.

[20] *Ethical Hacking and Countermeasures (CEH v7.1)*, EC-Council. .

[21] B. Genge, I. Kiss, and P. Haller, “A system dynamics approach for assessing the impact of cyber attacks on critical infrastructures,” *Int. J. Crit. Infrastruct. Prot.*, vol. 10, pp. 3–17, Sep. 2015.

[22] S. Hauge, P. Hokstad, S. Håbrekke, and M. A. Lundteigen, “Common cause failures in safety-instrumented systems: Using field experience from the petroleum industry,” *Reliab. Eng. Syst. Saf.*, vol. 151, pp. 34–45, Jul. 2016.

[23] A. Mosleh, K. N. Fleming, G. W. Parry, H. M. Paula, D. H. Worledge, and D. M. Rasmuson, “Procedures for Treating Common Cause Failures in Safety and Reliability Studies,” Electric Power Research Inst., Palo Alto, CA (USA); Pickard, Lowe and Garrick, Inc., Newport Beach, CA (USA), EPRI-NP-5613-Vol.2, Dec. 1988.

[24] R. Donat and M. Bouissou, “Common Cause Failures in Discrete Dynamic Models: Theory and Applications in the Figaro Modelling Language,” in *In proceedings of the 25th European Safety and Reliability Conference (ESREL)*, Zürich, 2015.

[25] P. Mell, K. Scarfone, and S. Romanosky, “A complete guide to the common vulnerability scoring system version 2.0,” in *Published by FIRST-Forum of Incident Response and Security Teams*, 2007, pp. 1–23.

[26] G. Hatzivasilis, I. Papaefstathiou, and C. Manifavas, “Software Security, Privacy, and Dependability: Metrics and Measurement,” *IEEE Softw.*, vol. 33, no. 4, pp. 46–54, Jul. 2016.

[27] E. Jonsson and T. Olovsson, “A quantitative model of the security intrusion process based on attacker behavior,” *Softw. Eng. IEEE Trans. On*, vol. 23, no. 4, pp. 235–245, 1997.

[28] H. Holm, “A Large-Scale Study of the Time Required to Compromise a Computer System,” *IEEE Trans. Dependable Secure Comput.*, vol. 11, no. 1, pp. 2–15, Jan. 2014.

[29] M. A. McQueen, W. F. Boyer, M. A. Flynn, and G. A. Beitel, “Time-to-compromise model for cyber risk reduction estimation,” in *Quality of Protection*, Springer, 2006, pp. 49–64.

[30] C. Cocozza-Thivent, *Processus stochastiques et fiabilité des systèmes*. Springer Science & Business Media, 1997.

[31] “CVE details (The ultimate security vulnerability datasource).” [Online]. Available: www.cvedetails.com.

[32] M. Bouissou, “Automated dependability analysis of complex systems with the KB3 workbench: the experience of EDF R&D,” in *Proceedings of the International Conference on ENERGY and ENVIRONMENT, CIEM 2005*, 2005.

[33] P. G. Harrison, “Laplace Transform Inversion and Passage-Time Distributions in Markov Processes,” *J. Appl. Probab.*, vol. 27, no. 1, pp. 74–87, 1990.

[34] M. Bouissou, “A simple yet efficient acceleration technique for Monte Carlo simulation,” in *The 22nd annual European Safety and Reliability Conference ESREL*, Amsterdam, 2013.

[35] M. Bouissou and J.-C. Houdebine, “Inconsistency detection in KB3 tools,” in *ESREL 2002*.

Annex: The Figaro language

We give in this annex an excerpt of the Figaro description of the class that models a network zone, and show some of the key words used. We also explain the mechanisms behind the processing of the system model.

```
CLASS network_zone KIND_OF component; (* declaration of a class
modeling a network zone *)
CONSTANT (* declaration of the constants related to the class *)
      wireless  (* this Boolean is set to true in an object modeling
a wireless network zone *)
      DOMAIN BOOLEAN
      DEFAULT FALSE;
ATTRIBUTE (* declaration of the attributes related to the class *)
      lambda_auth (* rate of the attack “bypass authentication” *)
      DOMAIN REAL
      DEFAULT 0.01;
EFFECT (* effects are Boolean variables reset to FALSE after the
occurrence of an event and recalculated by interaction rules *)
      network_access
      LABEL "attacker has access on network %OBJECT";
INTERFACE (* declaration of relations with other classes *)
      connectedElements (* this name will be used in the rules to
designate a set of physical_cpt *)
      KIND  physical_cpt
      CARDINAL 1 TO INFINITY
      LABEL "set of components connected to the network_zone";
FAILURE (* declaration of the failure modes and attack steps related
to the class *)
     jamming_attack
      LABEL "network jamming attack";

OCCURRENCE (* description of the dynamic behavior of the failure
modes and attack steps *)
     wireless_network_jamming_attack (* name of the rule – used only
for traceability in debug tools *)
      IF wireless
      MAY_OCCUR
      FAULT jamming_attack
      DIST EXP(0.000001);

INTERACTION (* propagation of the instantaneous effects of the
failure modes and attack steps *)
     network_unavailable (* effects of a jamming attack: the output
data of all software executed on all machines connected to the
network become unavailable. *)
      IF failure OR jamming_attack OR denial_of_service_attack
      THEN FOR_ALL x A connectedElements
      DO (
            FOR_ALL y A hosted_software(x)
              DO (
                   FOR_ALL z A out_data(y) DO unavailable(z) ) );
```

In Table 5, we detail the meaning of a few keywords used in the Figaro language. The complete documentation on the language is free and available in two manuals: one is dedicated to the language syntax and the other to its semantics and use.

Keyword	Signification
CLASS	Declares a Class
KIND_OF	Inheritance relationship with other classes
CONSTANT	Declares the constants related to the class
ATTRIBUTE	Declares the attributes related to the class. Contrary to constants, attributes can change value by execution of the occurrence/interaction rules
EFFECT	Declares the effects related to the class. An effect is a Boolean that is used to propagate the effects of the attack steps and failure modes. The value of this Boolean is reset to FALSE after each application of a transition, then updated via the execution of the interaction rules
INTERFACE	Declares the interfaces related to the class. An interface describes a relationship between the class and other classes
FAILURE	Declares the failure modes and attack steps related to the class
OCCURRENCE	Occurrence rules describe the transitions that may affect the state of a component of this class, by : their guard (conditions required for a transition to be applicable), the state changes that they induce and the associated probability distribution
INTERACTION	Interaction rules propagate the deterministic and instantaneous effects of a transition, among which the effects resulting from the realization of the attack steps and failure modes

Table 5: Meaning of some keywords of the Figaro language

There are two levels of the Figaro language: order 0 and order 1. The order 1 Figaro, represented so far, is used to write knowledge bases. Using a variety of keywords including quantifiers (e.g., IT_EXISTS, FOR_ALL), it is a highly expressive and natural language. The order 0 Figaro is the language in which the textual model, resulting from the instantiation of the KB on the graphical model, is generated. This language includes few keywords, which makes it simple and efficiently executable by machines for further processing.
The formal definition of the Figaro language is given in [35] and allows inconsistencies to be detected or the consistency of knowledge bases to be ensured as they are built. The S-cube KB respects a set of rules given in [35] that ensure the consistency of this KB. This implies that all the models built with the S-cube KB are coherent from their very construction, and can embed no inconsistencies or undesirable properties. In particular, for any model built using the S-cube KB the following properties are satisfied:

- The space of states is finite as all the variables defined have a finite domain;
- The model is not totally repairable, as detection and reaction measures have not so far been modeled. This implies that the space of states includes some states from which the initial state can no longer be reached;
- Monotonous inference: the EFFECTs are only set to true in the interaction rules. Given that all effects are initialized to false each time the interaction rules are executed, this guarantees that whatever the execution order of the interaction rules, the inference converges towards the same state.

The S-cube KB is instantiated on the graphical model of the system architecture, which generates a textual model in order 0 Figaro. This model implicitly defines a Continuous Time Markov Chain (CTMC), since all timed transitions of the model are associated with exponential distributions. Because of the combinatorial explosion of the states, this CTMC cannot be exhaustively built (in most cases), but it can be explored by the quantification tools, in order to yield qualitative and quantitative results. Initially, the state of the system is given by the values of the attributes and constants associated with each object. The interaction rules are first executed in order to initialize the values of EFFECTs, before the occurrence rules are executed. If the conditions of an occurrence rule are fulfilled, the risk event (FAILURE) can occur (instantaneously or after an exponentially distributed time). After the simulation of an event, which locally changes a few attributes of a given object, the interaction rules are executed again in order to refresh the effects in the entire model.
In order to yield qualitative and quantitative results, the order 0 Figaro model can be processed using the Figseq or Yams tools (see § 8.1).