SafeConcert: a Metamodel for a Concerted Safety Modeling of Socio-Technical Systems

Leonardo Montecchi[1,2], Barbara Gallina[3]

[1]University of Florence – Firenze, Italy [2]University of Campinas – Campinas, Brazil
leonardo@ic.unicamp.br
[3]Mälardalen University – Västerås, Sweden
barbara.gallina@mdh.se

Abstract. Socio-technical systems are characterized by the interplay of heterogeneous entities i.e., humans, organizations, and technologies. Application domains such as petroleum, e-health, and many others rely on solutions based on safety-critical socio-technical systems. To ensure a safe operation of these interacting heterogeneous entities, multifaceted and integrated modeling and analysis capabilities are needed. Currently, such capabilities are not at disposal. To contribute to the provision of such capabilities, in this paper we propose SafeConcert, a metamodel that offers constructs to model socio-technical entities and their safety-related properties. SafeConcert also represents a unified and harmonized language that supports the integrated application of qualitative as well as quantitative safety analyses techniques. To support our claims we briefly report about the evaluation that was conducted and documented in the context of the EU CONCERTO project.

Keywords: Safety-critical; socio-technical systems; modeling; safety analysis.

1 Introduction

In application domains such as petroleum, e-health, etc., safety-critical socio-technical systems play a crucial role. Offshore installations as well as telenursing, for instance, are characterized by the interplay of heterogeneous entities: humans (e.g., workers, caregivers and patients), organizations (e.g., regulatory bodies) and technology (e.g., decision support systems, databases, etc.). To prevent accidents, the interplay between such heterogeneous entities must be acceptably safe. To perform safety analysis, both qualitative and quantitative compositional techniques are available. Currently, automated techniques are at disposal, addressing different aspects of the safety analysis of a complex safety-critical socio-technical system. However, the current gap resides in the integration: such kind of systems require the integrated application of different techniques, and currently no satisfying means is at disposal for multifaceted, integrated, and tool-supported safety modeling and analysis of socio-technical systems.

One of the objectives of the CONCERTO project [2], was to contribute to the provision of such means. In this paper, we propose a novel metamodel developed within the project, called SafeConcert, which permits architects to interpret human, organiza-

adfa, p. 1, 2011.

tional, and technological entities in terms of components, and model their behavior with respect to safety. SafeConcert supports, as in a concert, multiple voices: not only technological components but also human and organizational components; not only a single safety analysis technique but the interplay of safety analyses techniques; not only a single safety standard but a family of standards.

To summarize, the contribution of this paper is a novel metamodel that targets the design of safety-critical socio-technical systems by: i) offering constructs for modeling both socio (i.e., human and organizational), and technical entities in a common model, thus facilitating the unified analysis of their interdependencies; ii) offering constructs for modeling the nominal, erroneous, and fault-tolerant behavior of both socio and technical components; iii) offering constructs for classifying entities with respect to their criticality, supporting both cross-domain and domain-specific annotations; and iv) facilitating the interplay of qualitative quantitative safety analyses, by using a common harmonized conceptual model.

Two widely adopted techniques for safety analysis are those based on failure propagation logic, typically for qualitative analysis, and those based on stochastic Petri nets, when quantitative analysis is needed. After introducing the SafeConcert metamodel, we discuss its ability to support both techniques, and the benefits in such cross-fertilizing interplay. Finally, we briefly discuss about the implementation and the different ways in which the metamodel has been evaluated.

The rest of the paper is organized as follows. In Section 2, we provide essential background information. In Section 3, we present our metamodel, SafeConcert. In Section 4, we discuss how the metamodel supports safety analysis, and we summarize on its evaluation. In Section 5, we discuss related work. Finally, in Section 6, we present some concluding remarks and future work.

2 Context and Background

2.1 Safety-Critical Socio-Technical Systems

Safety is defined as the "absence of catastrophic consequences on the user(s) and the environment" [13], and it is an attribute of dependability. Safety-critical systems are those systems whose failure may have impact on the safety, i.e., can lead to catastrophic consequences on users and the environment. Many systems on which we rely every day fall in this category: transportations, medical devices, power plants.

Socio-technical refers to the interrelatedness of "social" (of people and society) and "technical" (of machines and technology) aspects [3,9]. Successful (or unsuccessful) system performance depends on this interrelatedness, which comprises linear 'cause and effect' relationships, and 'non-linear', complex, even unpredictable relationships. A wide range of safety-critical systems exhibit socio-technical aspects as well. Proper safety practices should try to investigate the cause-effect relations that play a role in reduced human performance, and put the blame on such relations rather than on humans mistakes alone [30].

Such systems must ensure harm-free operation, even in the event of components' failure or incorrect interactions between components. This is typically assessed through a structured development process that includes the execution of safety analysis, whose objective is to define safety-related requirements, and assess whether the system fulfills them. The metamodel in this paper aims to support such process.

Manufacturers as well as suppliers have to comply with safety standards e.g., ISO 26262 [16] in the automotive domain. These standards tend to be conservative, so new practices typically are not incorporated due to the impossibility to measure their confidence. In fact, in the industry, prevalently manual techniques like Fault Tree Analysis (FTA) and Failure Modes and Effects Analysis (FMEA) are still the main instrument to analyze the propagation of threats (faults/errors/failures [13]), and produce evidence that the system fulfills its safety requirements.

Model-based approaches (e.g., [14]), by improving the formalization and consistency of concepts, have the potential to support a better documentation of safety aspects, and semi-automated support to safety analysis. Nevertheless, the gap between industrial state of practice and academic state of the art is still evident, for different reasons, including the lack of supporting tools. Only recently, in the avionics domain, model-based development has been included as a DO-178C supplement, known as DO-331 [22]. Guidance is now available for applying model-based development.

2.2 Safety Analysis in the CHESS Framework

The "CHESS Framework" is a framework for the design, development, and analysis of safety-critical systems, with a strong focus on the specification and analysis of non-functional properties. The framework defines a UML-based modeling language, called CHESS ML [23], and includes a set of plugins to perform code generation, constraints checking, and different kinds of analyses. The framework was originally developed within the CHESS project [1], later extended within CONCERTO [2], and then released as open source under the Polarsys initiative [5]. Two safety analysis techniques are supported: CHESS-FLA and CHESS-SBA.

CHESS-FLA [31] allows users (system architects and safety engineers) to decorate component-based architectural models (specified using CHESS-ML) with dependability related information, execute Failure Logic Analysis (FLA) techniques, and get the results back-propagated onto the original model. Different FLA techniques are available in the literature [14], and can be used at the early stages of the design phase to achieve a robust architecture with respect to linear relationships. CHESS-FLA supports FPTC [4] and FI^{4}FA [26]; the reason behind this choice in relation to other existing techniques was discussed in [31]. FPTC is a compositional technique to qualitatively assess the dependability/safety of component-based systems, and partially combines and automatize traditional safety analysis techniques (i.e., FMEA and FTA). FPTC allows users to calculate the behavior at system-level, based on the specification of the behavior of individual components. The behavior of the individual components, established by studying the components in isolation, is expressed by a set of logical expressions (FPTC rules) that relate output failures (occurring on output ports) to combinations of input failures (occurring on input ports). FI^{4}FA extends

FPTC for reasoning about failures related to faulty design of concurrency control or fault tolerance within transaction-based systems.

The CHESS "State-based analysis" plugin (CHESS-SBA) [10] allows users to perform quantitative dependability analysis on models specified using CHESS-ML and enriched with quantitative (i.e., probabilistic/stochastic) dependability information, including failure and repair distribution of components, propagations delays and probabilities, and some fault-tolerance and maintenance concepts. Such information is added to the functional model in two main ways: i) with a set of stereotypes that allow simple attributes to be attached to software and hardware components [19], or ii) with an "error model", i.e., a particular kind of StateMachine diagram in which a more detailed failure behavior of a component can be specified.

Such enriched model is transformed to a stochastic state-based model, which is then analyzed to evaluate the degree of satisfaction of system-level dependability attributes, in the form of probabilistic metrics. As opposed to combinatorial models like Fault-Trees, state-based methods like Stochastic Petri Nets (SPNs) [29] are able to take into account complex dependencies between events.

One of the limitations that we faced in the previous versions of the framework was the lack of integration between the different analysis techniques. While they could be applied starting from the same system architecture specified in CHESS ML, those analyses were using different portions of the language, being de-facto impossible to use them in an integrated way. In general, in safety analysis some aspects are more conveniently analyzed with a certain technique with respect to another, and different levels of detail are required at different stages. Also, results obtained with a technique are useful inputs for subsequent analysis steps. Being able to apply both techniques on the same model is a contribution towards integrated model-based safety analysis.

3 SafeConcert

In CONCERTO we improve the integration of safety analyses by defining a common underlying metamodel, called SafeConcert, which solves inconsistencies and redundancies. Besides adding new modeling features, SafeConcert serves as a common abstract syntax for both FLA and SBA. From this common metamodel, different concrete syntaxes, suitable to the specificities of each technique, are then derived.

The harmonization is based on a set of principles, which are discussed in the following. The modeling philosophy follows the *component-based* approach, meaning that components, ports, and connectors are the main entities. Failure modes will be specified with respect to the services that a component provides, i.e., its output ports. We describe the failure behavior of system elements as *state machines*, which are flexible while being relatively simple to describe and understand. This also adheres to the classical literature on dependability, which defines key concepts like errors and failures as states and events, respectively [13].

The presentation of the SafeConcert metamodel is organized by topics: *structure*, *organizational components*, *human components*, *failure modes and criticality*, *behav-*

iors, *input and output events*, *internal events*, and *fault tolerance events*. A complete view of the metamodel is available at [21].

3.1 Structure

The central element of SafeConcert is the *SystemElement* abstract metaclass, which is the common supertype for all the entities that may have dependability information attached to them. A *SystemElement* can be either a *Component* or a *Connector*. We distinguish three kinds of components: *SoftwareComponents*, *HardwareComponents* (including mechanical ones), and *SocioComponents*. Software and hardware components can be described in a hierarchical fashion, i.e., they can contain sub-components of the same type. Software components can be allocated on hardware components (*isAllocatedOn* relation). Socio components are partitioned into two distinct kinds of components: *Organization* and *Human*.

A Connector represents anything via which two elements may interact including, but not only, hardware and software connections, which are modeled with the *HardwareConnector* and *SoftwareConnector* elements, respectively. Software connectors may be allocated on hardware connectors, by means of the *isAllocatedOn* relation. Other links that do not fit in that category (e.g., voice communication for human entities) are modeled as a generic *Connector* element.

A *SystemElement* may own a certain number of *Ports*, which allow it to interact with other system elements. Since also connectors can fail (e.g., a network cable), differently than in functional modeling, we adopt ports also for *Connector* elements. We assume here that component instances inherit the failure behavior of the component type, and we thus do not introduce the concept of component instance.

3.2 Organizational components

For social components, as initially explored in [3], we base our model on the SERA (Systematic Error and Risk Analysis)-related classification [18]. An organization can be decomposed in a certain number of different "units", modeling different aspects of how the organization gathers and processes information, and interacts with the other entities of the socio-technical system.

The abstract class *OrganizationUnit* is specialized to reflect different kinds of units, which are identified by the prefix "OU". Following [18], we include units devoted to mission management (*OUMissionManagement*), to the management of rules and regulations (*OURules&RegulationManagement*), climate (*OUClimateManagement*), oversight (*OUOversightManagement*), process (*OUProcessManagement*), and resources (*OUResourceManagement*).

3.3 Human components

Following what initially devised in [3], human components are represented as composite components; the motivation stems from the SERA-related classification. By inspecting thoroughly the twelve categories of human failures (e.g. attention failure), it

has been recognized that these failures were related to two types of human functionalities: internal functionalities responsible of sensing, perceiving, deciding, etc., and functionalities responsible of acting. Thus, a *Human* component consists of a set of *HumanSensorUnit* and a set of *HumanActuatorUnit* elements (Fig. 1).

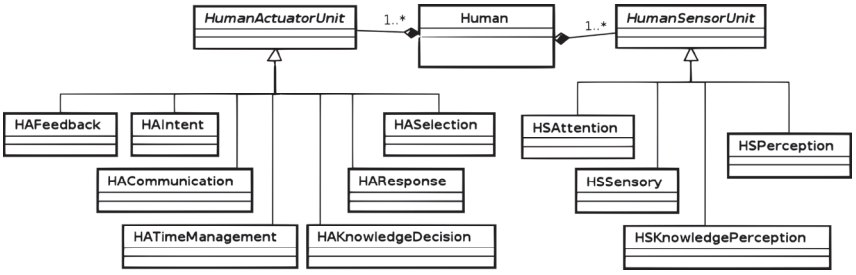

Fig. 1. SafeConcert model elements to model human components.

A sensor-like capability (*HumanSensorUnit*) can be one of the following kinds. *HSAttention*, responsible of attending to relevant information that is present or accessible; *HSPerception*, responsible of assessing a situation; *HSKnowledgePerception*, responsible of interpreting a situation, based on pre-existing baseline knowledge; and *HSSensory*, responsible of sensing the incoming visual, auditory, tactile and olfactory information.

An actuator-like capability (*HumanActuatorUnit*) can be one of the following kinds. *HAFeedback*, responsible of maintaining error-correcting feedbacks, aimed at adjusting the imprecision of the internal models; *HAIntent*, responsible of exercising a goal in compliance with rules and regulations; *HACommunication*, responsible of passing/receiving correct information; *HASelection*, responsible of formulating the right plan to achieve the goal; *HAResponse*, responsible of actuating the physical response required to perform the task; *HAKnowledgeDecision*, responsible of forming an appropriate or correct response to the situation; and *HATimeManagement*, responsible of using appropriate and effective time management strategies.

There may be different reasons for including human behavior in safety models. In the automotive domain, for example, it would help in characterizing the aspect of controllability [16] of the driver.

3.4 Failure modes and criticality

In our metamodel, a *Port* is the point where failures are propagated from/to other entities of the socio-technical system, i.e., its *service interface* as meant by [13]. In this perspective, also a *SocioComponent* can own ports.

As shown in Fig. 2, *Ports* are the model elements to which failure modes (*FailureMode* entity) are associated. More precisely, failure modes are grouped into *FailureModeGroups*, which are then associated to ports. The motivation of this choice is well explained in [11]: in this way failure modes can be organized by domain or context. For example, distinguishing failure modes related to mechanical failures from those related to electrical failures.

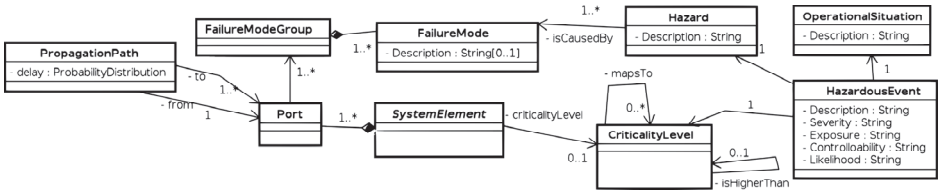

Fig. 2. SafeConcert model elements to represent failure modes and criticality.

Propagation between two *Ports* is established by the *PropagationPath* entity, which has essentially the role of the classical "connector", i.e., it connects two *Port* entities. The *delay* attribute may specify a propagation delay between the two ports.

SafeConcert also includes a set of constructs especially devoted to safety aspects. As defined in safety standards (e.g., [16]), a hazardous event occurs as a combination of a hazard and an "operational situation", e.g., parking vs. driving for the automotive domain, takeoff vs. landing for avionics. As such, the *HazardousEvent* element relates itself to an *OperationalSituation* and a *Hazard*. A *Hazard* is in relation with one or more *FailureModes* that are its causes. Besides having a textual description, a *HazardousEvent* has also additional attributes that characterize the event. Controllability, severity and exposure, for instance, are attributes used to establish the criticality level of hazardous events in the automotive domain. Such attributes allow analysts to derive a criticality level to be assigned to the hazardous event.

Due to the different existing classification of criticality levels in different domains (e.g., SILs, ASILs, DALs [15]), the metamodel allows hierarchies of criticality levels to be specified and put in relation with each other. A *CriticalityLevel* may then be related to a lower-level one through the *isHigherThan* relation; for example, "SIL-4" would be in such relation with "SIL-3". Criticality levels from different domains can be put in relation through the *mapsTo* relation; for example, "ASIL-D" could be in such relation with "SIL-3" [15]. Using this approach permits to: i) reuse (at least partially) safety models from different domains, and ii) have both domain-specific and cross-domain criticality levels defined in the same model.

A criticality level can be associated to both *SystemElement* and to *HazardousEvent*. In principle, the criticality level assigned to a system element should be equal or higher than the highest criticality level associated with hazards caused by its failure modes. Ensuring such consistency is beyond the scope of this paper: actually such consistency should be reached as a result of the safety analysis process.

3.5 Behaviors

The failure behavior of system elements is defined as a state machine, i.e., states and transitions between them. Note that we want to model the behavior from the perspective of safety analysis only. Accordingly, each *SystemElement* owns one or more *States* (Fig. 3), either *NormalStates*, states that belong to the nominal behavior of the component; *DegradedStates*, states in which the service provided by the component is degraded but still following the specifications; or *ErroneousStates*, states which deviate from the correct behavior of the component.

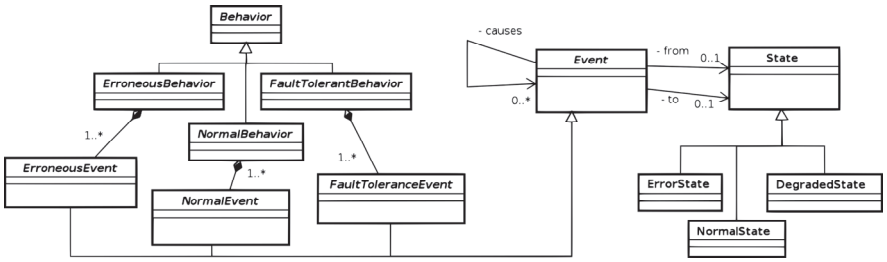

Fig. 3. Modeling of behaviors, events, and states in SafeConcert.

An *Event* is something that occurs either internally or externally to the component, and that is deemed atomic with respect to the adopted level of detail. An event may (but not necessarily), cause the component to transition from one state to another. We classify events according to two dimensions: i) location, i.e., whether they are *InputEvents, InternalEvents*, or *OutputEvents*, and ii) type, i.e., whether they are *NormalEvents*, *ErroneousEvents,* or *FaultToleranceEvents*.

A *SystemElement* may own a set of *Behaviors*, i.e., collections of events related to that component, organized according to different views. We separate different aspects of a component's behavior into three views: *NormalBehavior*, which contains transitions related to the nominal behavior of the component (*NormalEvents*), *ErroneousBehavior*, which contains transitions related to the erronoeus behavior of the component (*ErroneousEvents*), and *FaultTolerantBehavior*, which contains fault-tolerant behavior, in terms of *FaultToleranceEvents*. This approach adheres to the concept of "idealized fault-tolerant component" [17], in which different aspects of a component's behavior are kept separated.

3.6 Input and Output events

External events occur on the ports of a system element, and are either *InputEvents* or *OutputEvents.* In both cases, they are in relation with a *Port* element (Fig. 4). Two kinds of events may occur as an input event: a *NormalInput* event, or an *ExternalFault*, meaning that the component has received an input that deviates from the specification, i.e., the service it receives from another entity is not correct [13].

Output events can be *NormalOutput*, i.e., the component provides a correct service, or *Failure*, meaning that the service provided by the component on the involved port has become incorrect [13]. A *Failure* event has a relation to a *FailureMode* entity, which specifies the kind of failure that has occurred.

It is often difficult to precisely know the failure behavior of a component. In SafeConcert such uncertainty can be expressed according to two dimensions: i) in terms of

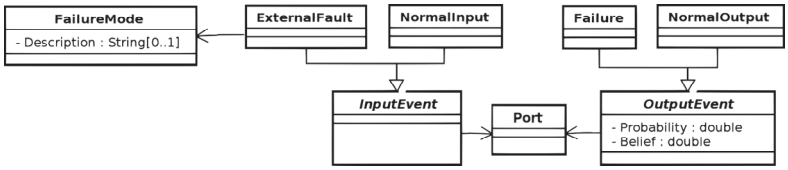

Fig. 4. Input events and output events in SafeConcert.

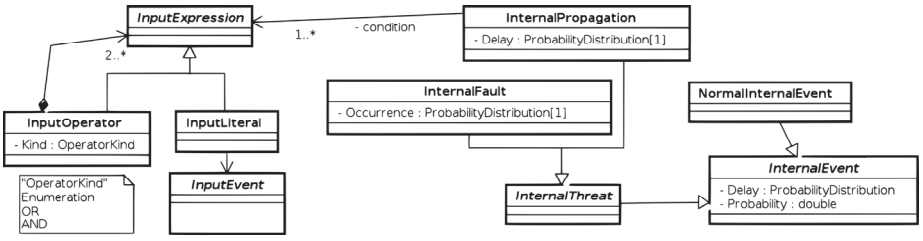

Fig. 5. Internal events in SafeConcert.

the *probability* of occurrence of one failure mode with respect to others, or ii) in terms of the *belief* on which failure mode will occur. In the first dimension, it is assumed that different failure modes can be clearly distinguished, and they occur with known probabilities. The second dimension reflects the condition in which the actual nature of failure modes is known only with a certain *belief*, as intended in the fuzzy logic domain. Belief supports the integration of F^2I^4FA [33].

3.7 Internal events

We distinguish between two macro-categories of internal events: *NormalInternalEvent,* and *InternalThreat* (Fig. 5). An internal event may have a *delay*, expressed by a probability distribution, and a *probability* of occurrence.

A *NormalInternalEvent* is an event foreseen by the specification, e.g., a switch to a power saving mode due to a low battery level. For the purpose of safety analysis it is important to take into account such aspects of a component's behavior: the occurrence of failures, as well as their effects and criticality, may depend, for example, on the current system operational mode (e.g., take off, cruise, and landing for an aircraft).

An *InternalThreat* is essentially an *InternalFault*, or an *InternalPropagation*. An *InternalFault* represents a fault that occurs spontaneously within the component (e.g., an electrical fault) or that is pre-existing and dormant [13] (e.g., a software fault). The *occurrence* attribute can be used to specify a probabilistic delay, after which the fault manifests itself in the state of the component, i.e., it gets activated [13].

The *InternalPropagation* concept defines how input events, or combination thereof, affect the internal state of the component. The condition that triggers the propagation is specified by the *condition* attribute, as a Boolean expression over *InputEvent* elements. The semantics of such expressions is as follows. A predicate *p.E*, where *E* is an *InputEvent* element occurring on port *p*, is *true* iff the most recent event occurred on *p* is *E*, and *false* otherwise. The effects of an *InternalPropagation* are specified by: i) specifying a state-transition for it, or ii) specifying a set of events that are triggered by it (e.g., *Failure* events), through the *causes* relation.

3.8 Fault-tolerance events

The classical taxonomy of dependable computing [13] classifies fault tolerance techniques in three main groups: error detection, error handling, and fault handling. In SafeConcert we follow such classification.

The *ErrorDetection* event represents the detection of an error in the state of the component, to which different actions may follow. Those actions can be defined by associating a state transition with the event (e.g., to a safe state), or by specifying that additional events are triggered by the error detection event (e.g., reconfiguration events). The *ErrorHandling* event represents the elimination of an existing error in the state of the component, thus bringing the component to an error-free state (e.g., rollback or rollforward [13]). The *FaultHandling* event represents the application of a technique that prevents existing faults from being reactivated, e.g., fault isolation (faulty components are excluded from the service delivery) or reconfiguration.

We note that maintenance concepts are not explicitly addressed in our metamodel. Events like physical repairs, replacement of failed components, restart of software are represented as *FaultToleranceEvents* elements. In fact, maintenance differs from fault tolerance only in requiring the participation of an external agent [13].

4 Safety Analysis with SafeConcert

4.1 Failure Logic Analysis with SafeConcert

As briefly introduced in Section 2.2, in FPTC and in failure propagation analysis in general, a specification for a component contains a set of expressions that relate the failures occurring on its input ports with failures occurring on its output ports.

In SafeConcert, the behavior corresponding to an FPTC specification is described by a collection of *Events*, grouped in one or more *Behavior* elements. Each failure received as input, and thus referenced in the left part of a FPTC rule in the specification, is described by an *ExternalFault* event, referencing the involved *FailureMode* and *Port* elements. Similarly, each failure that can be produced by the component, i.e., those referenced in the right part of a FPTC rule, is described by a *Failure* event, with the appropriate *FailureMode* and *Port* elements as references. Such events are grouped into the *ErroneousBehavior* of the involved component.

A set of *NormalInput* events, one for each of the input ports of the component, expresses the occurrence of a "normal" input on the ports of the component (i.e., the "nofailure" of [3]); similarly, a set of *NormalOutput* events, one for each output port of the component, express the "normal" output emitted on those ports. Those are collectively grouped into the *NormalBehavior* of the involved component.

For each rule, the causation link between the input events (left part of the rule), and output events (right part of the rule) is represented through an *InternalPropagation* model element for which i) the *condition* attribute is set based on the left hand side of the FPTC rule, referring the corresponding *NormalInput* and/or *ExternalFault* events; and ii) the *causes* attribute is instead set based on the right hand part of the rule, adding all the corresponding *NormalOutput* and/or *Failure* events.

Concerning states, the adopted level of detail implies a single *State* for each component, which is not modified by the occurrence of events. This is consistent with the interpretation of failure propagation specifications: cause-effect relations between

input events (external faults) and output events (failures), that are abstraction of the internal behavior of the component, and do not change over time.

4.2 State-Based Analysis with SafeConcert

We first consider the "template" stereotypes that are used in CHESS-SBA to compactly describe the failure and repair behavior of component [19]. For stateless components («StatelessSoftware» or «StatelessHardware»), and components annotated with «SimpleStochasticBehavior», it is assumed that they immediately become failed as result of fault activation. In SafeConcert, this is represented by defining two states for the component, "healthy" and "failed", and an *InternalFault* event causing a transition between them. The event causes (through the *cause* relation) one or more *Failure* events, one for each output port of the component.

For stateful components instead («StatefulSoftware» and «StatefulHardware»), time elapses between fault activation and the subsequent component failure. In SafeConcert this is modeled with an additional "erroneous" state of the component. The *InternalFault* event makes the component transition from "healthy" to "erroneous"; an additional *InternalPropagation* event, representing error latency, triggers the transition from "erroneous" to "failed", and causes the *Failure* events for the component. Concerning repairs, in both cases, they are modeled with a *FaultHandling* event, which makes the component transition from the "failed" state back to the "healthy" one. The *FaultHandling* event also causes one or more *NormalOutput* events (one for each output port of the component), meaning that normal service is restored.

Mapping elements of the CHESS ML "ErrorModel" is even simpler, since it is defined using a UML StateMachine diagram. The initial healthy state, as well as «Error» and «FailureMode» states defined in the CHESS ML error model are mapped to states of the SafeConcert metamodel. Similarly, «InternalFault» elements and «InternalPropagation» elements have an almost direct correspondence with elements in SafeConcert. The main difference resides in how failures are described. In the CHESS ML ErrorModel the ports that are affected by a failure are specified on «FailureMode» states. In SafeConcert they are specified using the *causes* attribute of *InternalPropagation* events. This way provides greater flexibility and it allows users to model a wider range of failure behaviors with respect to what it was possible in CHESS.

How the modeled information will actually be translated into a Stochastic Petri Nets model is beyond the scope of this paper. The CHESS-SBA implementation is based on an intermediate model [5,10] that bridges the distance between the CHESS ML profile and the Stochastic Petri Nets formalism. The plugin architecture and implementation is described in [10].

4.3 Approach Evaluation

Latest releases of the CHESS Framework [5] incorporate SafeConcert, as well as the new plugins for performing safety analyses based on it [25]. The evaluation has adopted an incremental and iterative approach. An initial evaluation is reported in [3], where concepts for socio-technical systems are applied to a case study in the petrole-

um domain. A further evaluation has been performed by industrial partners of the CONCERTO project and it is reported in [24]. Representatives of the petroleum domain have applied the implemented profile to a case study of a gas detection system, providing useful feedback for the refinement of concepts. The evaluation has then been extended to the telecare domain in [25], in which a use case is modeled with the profile derived from SafeConcert, and availability of the system analyzed.

At the end of the project, industrial partners have provided further feedback, judging their satisfaction in using the features of the CONCERTO framework. Fig. 6 reports the evaluation of petroleum and telecare partners; "component modeling" and "fault propagation and analysis", which are features related to SafeConcert, received a positive judgment. Afterwards, this positive experience has led to the definition of a systematic way to integrate CHESS-SBA in the safety procedures within the petroleum domain [27].

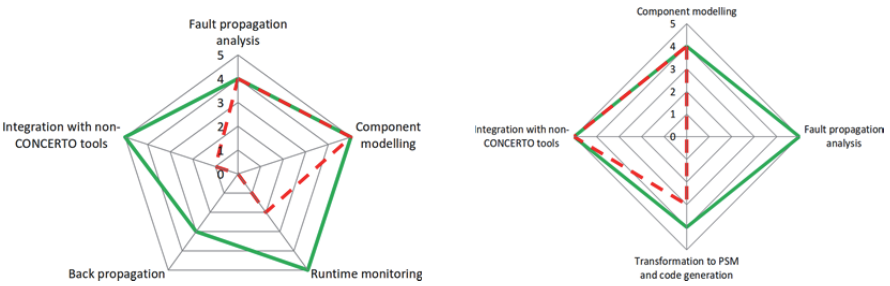

Fig. 6. Evaluation of CONCERTO features by petroleum (left) and telecare (right) users [24].

Finally, as part of the project's self-assessment, a questionnaire about CONCERTO features has been prepared and submitted to 9 external experts in the field of safety assessment. Concerning dependability modeling, 80% of the responses are positive [24]. Concerning analyses, while still no negative answers were received, we noticed that experts were more cautious (almost half of the answers were blank or "neutral"). This might reflect the lack of confidence with a specific analysis technique, and the difficulty in judging it without extensive first-hand usage.

5 Related Work

To offer dependability modeling capabilities, many works (including ours) have proposed UML extensions that are then implemented as UML profiles. EAST-ADL2 [8], for instance, is a modeling language for electronics system engineering within the automotive domain, which extends UML and SysML. While safety properties and error propagation can be specified to some extent, EAST-ADL2 is very tied to the automotive domain. For example, integrity levels can be defined only by means of ASILs (*Automotive* Safety Integrity Levels).

The SAE "Architecture Analysis and Design Language" standard defines the AADL language. Its Error Model Annex [6] is of particular relevance, since it allows users to add information on erroneous behavior to functional models. AADL models extended with the Error Model Annex have been shown to be analyzable using

Generalized Stochastic Petri Nets [12]. While using a similar approach for modeling the failure behavior, SafeConcert also includes elements for socio-technical systems, and for modeling of the criticality associated with components and failure modes.

The work in [7] defined the Dependability Analysis Modeling (DAM) profile, a MARTE-based UML profile for dependability modeling. The same work includes an an interesting survey on UML profiles for dependability analysis. While DAM is an important step forward in the introduction of dependability attributes at UML level, it was not suitable for ours and project's objectives. The DAM profile offers too much freedom to the modeler, and it is very coupled with MARTE: this is against the "correctness-by-construction" and "separation of concerns" pillars behind CONCERTO.

The authors of [32] define SafeML, a SysML profile for modeling safety-related information. Their focus is on information presentation and traceability, and they assume that all the safety analyses have been already performed [32]. Thus, SafeML does not support the application of safety analyses, but only the documentation of results. To a certain extent, SafeML is complementary to our work, which has instead a stronger focus on the analysis dimension.

As the above works do, we also aim at supporting safety annotations via an UML profile. However, in this paper we do not focus on the UML representation of our language (i.e., its concrete syntax, available at [23]), but only on the underlying metamodel, i.e., we define which are the concepts and the relationships between them. Furthermore, our work goes beyond the objectives of the previous mentioned works: i) it provides support for modeling both "socio" and "technical" entities, as well as their interactions; and ii) it provides support for both failure logic analysis and stochastic state-based analysis in a common metamodel.

Other works in the literature focused on the conceptual level, introducing metamodels that provide the foundation for more concrete and user-oriented languages. Lisagor [11] defined a metamodel to support the application of different existing failure propagation analysis techniques. This is perhaps the work which is most closely related to the one presented in this paper. With respect to Lisagor's work, our metamodel covers a wider range of aspects: i) quantitative analysis, ii) the modeling of socio-technical systems, and iii) criticality levels.

6 Conclusion and Future Work

In this paper we introduced a novel metamodel, SafeConcert, which aims at supporting architects in the iterative and incremental modeling and analysis of safety-critical socio-technical systems. Safe concertation of human, organizational and technological entities is enabled thanks to the promising interplay of qualitative and quantitative safety analyses.

Currently, work is devoted at further evaluating SafeConcert within industrial settings beyond the research project's border. Several directions are open for long-term activities. Based on what done by Sljivo et al. [20], we aim to enable the generation of fragments of safety case arguments from analysis results. Based on the extensive taxonomy work in [28], we aim to extend the metamodel to integrate concepts from the

Systems-of-Systems approach, thus further expanding the range of systems addressed by SafeConcert.

Acknowledgement

This work has been partially supported by the EU ARTEMIS project CONCERTO [2], and by the ECSEL Joint Undertaking project AMASS (No 692474).

References

1. ARTEMIS-JU-100022 CHESS – "Composition with guarantees for High-integrity Embedded Software components aSsembly". http://www.chess-project.org.
2. ARTEMIS-JU-333053 CONCERTO – "Guaranteed Component Assembly with Round Trip Analysis for Energy Efficient High-integrity Multicore Systems". http://www.concerto-project.org/.
3. B. Gallina, E. Sefer, A. Refsdal, "Towards Safety Risk Assessment of Socio-Technical Systems via Failure Logic Analysis," in Software Reliability Engineering Workshops (ISSREW), 2014 IEEE International Symposium on, pp.287-292, 3-6 Nov. 2014.
4. M. Wallace. "Modular architectural representation and analysis of fault propagation and transformation" Electronic Notes in Theoretical Computer Science, volume 141 n.3, pp. 53-71, December, 2005.
5. PolarSys CHESS, https://www.polarsys.org/chess/ (last accessed 01/06/2017).
6. Society of Automotive Engineers. SAE Standards: AS5506/1, Architecture Analysis & Design Language (AADL) Annex Volume 1, June 2006.
7. S. Bernardi, J. Merseguer, D.C. Petriu. "A dependability profile within MARTE", Software and Systems Modeling, vol. 10, no. 3, pp. 313-336, July 2011.
8. ATESST consortium. "EAST-ADL2 UML2 Profile Specification", January 2008.
9. G. Walker, N. Stanton, P. Salmon and D. Jenkins. "A Review of Sociotechnical Systems Theory: A Classic Concept for New Command and Control Paradigms", Human Factors Integration Defence Technology Centre, U.K. Ministry of Defence Scientific Research Programme, HFIDTC/2/WP1.1.1/2, 2007.
10. L. Montecchi, P. Lollini, A. Bondavalli, "A reusable modular toolchain for automated dependability evaluation.", VALUETOOLS 2013, Torino, Italy, Dec 2013, pp. 298-303.
11. O. Lisagor, "Failure Logic Modelling: A Pragmatic Approach", PhD thesis, University of York, Department of Computer Science, March 2010.
12. A.-E. Rugina, K. Kanoun, M. Kaâniche, "A System Dependability Modeling Framework Using AADL and GSPNs", Architecting Dependable Systems IV, LNCS 4615, pp. 14–38, 2007.
13. A. Avižienis, J.-C. Laprie, B. Randell, C. Landwehr, "Basic Concepts and Taxonomy of Dependable and Secure Computing" IEEE Transactions on Dependable and Secure Computing, 1, 11-33, 2004.
14. L. Grunske, J. Han, "A Comparative Study into Architecture-Based Safety Evaluation Methodologies using AADL's Error Annex and Failure Propagation Models", 11th IEEE High Assurance Systems Engineering Symposium, pp. 283–292, Nanjing, China, 3-5 Dec., 2008.
15. E. Verhulst, J.L. de la Vara, B.H. Sputh, V. de Florio, "ARRL: A Criterion for Composable Safety and Systems Engineering", SAFECOMP 2013 Workshops - SASSUR'13, 2013.

16. ISO26262. “Road vehicles – Functional safety”. International Standard, November 2011.
17. P.A. de C. Guerra, C.M.F. Rubira, A. Romanovsky, A., R. Lemos, "A Fault-tolerant Software Architecture for COTS-based Software Systems” Proc. of the 9th European Software Engineering Conference, ACM, 2003, 375-378.
18. K. C. Hendy. “A tool for Human Factors Accident Investigation, Classification and Risk Management.” Defence R&D Canada, Toronto, DRDC Toronto TR 2002-057, March 2003.
19. L. Montecchi, P. Lollini, and A. Bondavalli. “Towards a MDE Transformation Workflow for Dependability Analysis”. In: IEEE International Conference on Engineering of Complex Computer Systems. Las Vegas, USA, 2011, pp. 157–166.
20. I. Sljivo, B. Gallina, J.Carlson, H. Hansson, S.Puri. “A Method to Generate Reusable Safety Case Argument-Fragments from Compositional Safety Analysis”, The Journal of Systems & Software: Special Issue on Software Reuse, July 2016 ,
21. L. Montecchi, B. Gallina, Complete diagram of the SafeConcert metamodel, http://rcl.dsi.unifi.it/~leonardo/safeconcert.png (last accessed 01/06/2017)
22. DO-331 “Model-Based Development and Verification Supplement to DO-178C and DO-278A”. RTCA, December 2011.
23. CONCERTO Deliverable D2.7 “Analysis and back-propagation of properties for multicore systems – Final Version”, November 2015.
24. CONCERTO Deliverable D5.6 “Use Case Evaluations – Final Version”, April 2016.
25. CONCERTO Deliverable D3.3 “Design and implementation of analysis methods for non-functional properties – Final version”, November 2015.
26. B. Gallina, S. Punnekkat, “FI4FA: A Formalism for Incompletion, Inconsistency, Interference and Impermanence Failures' Analysis," 37th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA’11), pp.493-500, Aug.30-Sept 2, 2011.
27. L. Montecchi, A. Refsdal, P. Lollini, and A. Bondavalli. “A Model-Based Approach to Support Safety-Related Decisions in the Petroleum Domain”. In: 46th IEEE/IFIP International Conference on Dependable Systems and Networks (DSN’16). Toulouse, France, June 28–July 1, 2016, pp. 275–286.
28. A. Bondavalli, S. Bouchenak, H. Kopetz (Eds.), “Cyber-Physical Systems of Systems – Foundations – A Conceptual Model and Some Derivations: The AMADEOS Legacy”, Lecture Notes in Computer Science, Volume 10099, 2016.
29. G. Ciardo, R. German, C. Lindemann, “A characterization of the stochastic process underlying a stochastic Petri net” IEEE Trans. on Software Engineering, 1994, 20, 506-515.
30. R.J. Holden, “People or systems? To blame is human. The fix is to engineer.” Professional safety, 54(12), pp. 34-41, 2009.
31. B. Gallina, M. Atif Javed, F. Ul Muram and S. Punnekkat. “Model-driven Dependability Analysis Method for Component-based Architectures”. In proceedings of the Euromicro-SEAA Conference, Cesme, Izmir, Turkey, September, 2012.
32. G. Biggs, T. Sakamoto, T. Kotoku, “A profile and tool for modelling safety information with design information in SysML”, Software & Systems Modeling, Volume 15, Issue 1, pp 147–178, February 2016.
33. B. Gallina, A. Dimov, S. Punnekkat. “Fuzzy-enabled Failure Behaviour Analysis for Dependability Assessment of Networked Systems”. IEEE International Workshop on Measurement and Networking (M&N), p 6, Anacapri, Italy, August, 2011.