A practical guide to assuring the system resilience to operational errors

Overview of the guide

Update: March 29, 2015

This guide proposes guidelines for coping with failures of various sources (the operator, hardware, software, context …) by prevention, protection, escalation prevention, as well as by learning from incidents. This overview describes the most important guidelines.

An interactive version of this overview is available here.

1 Underlying concepts

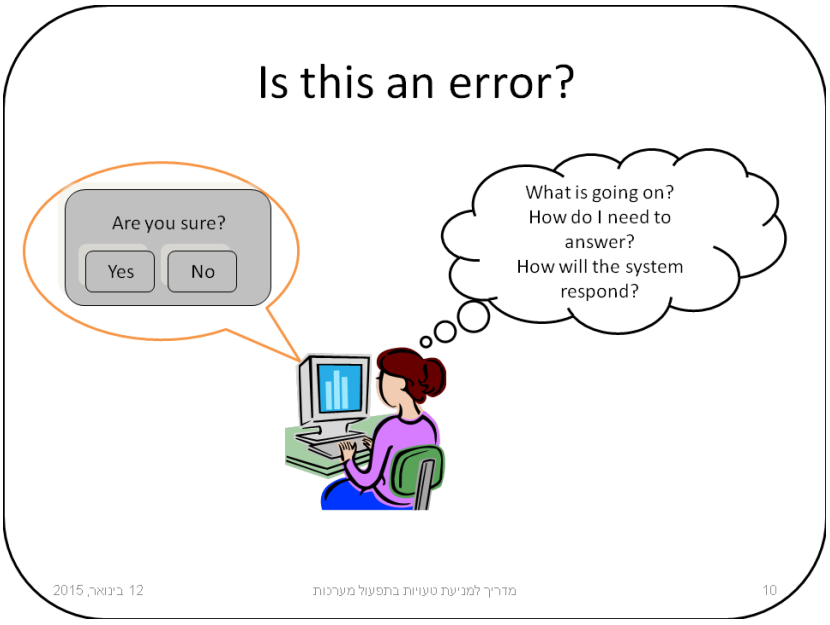
This section describes the following concepts used in the guide:

- Statement of the engineering role is resilience assurance
- Definition of the design scope, based on three categories of operational activities
- A model describing the generation of unexpected events
- A model describing the behavior of resilient systems
- The principle of self control, based on the STAMP paradigm
- Constructing defenses using the Swiss-cheese metaphor.

These concepts are hereby explained.

Operational errors

Often, the exceptional situations result from problems in the coordination between the machine and its human operators, as depicted in the following illustration.



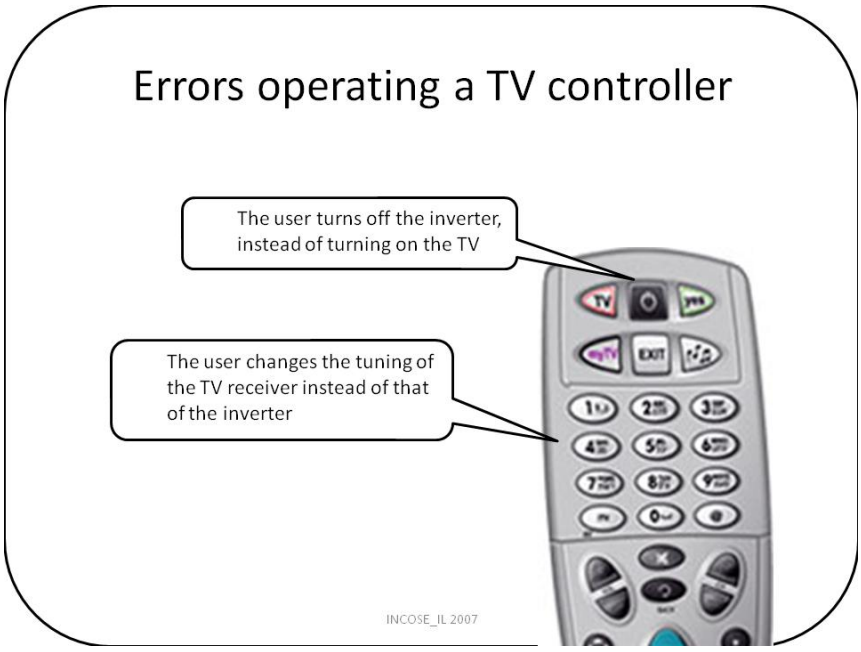
The coordination problem

A malfunction is regarded as an operational error if:

- The system operators fail to respond properly to an exceptional event,
- As a result, the system's situation becomes unusual, and
- The system operators fail to recover from the unusual situation.

Use errors and usability

Use errors and usability are two sides of the same coin. For example, many of the usability issues in operating consumer products, such as home TV, are due to use errors (Zonnenshain & Harel, 2009, download). The following figure illustrates problems typical of early generations of TV controllers.



Common mode errors

The engineering role

Common studies of resilience engineering focus on analysis of mishaps, stressing the management role and responsibility in creating safety culture (e.g., Hollnagel et al., 2006). In contrast, this guide focuses on aspects of system design. The guide assumes that it is the duty of the system engineers to prevent unexpected situations, and the guidelines are about achieving this goal.

Basic terms

Here are basic terms used in the guide:

The System. In the article, this term refers to the extended system, including the machine and the operators (Zonnenshain & Harel, 2009, download).

Human Factors. The discipline of Human Factors emerged during WW2 following a series of significant, critical human errors (Meister, 1999).

State. Used as shortcut for System State (should be distinguished from Unit State).

Scenario. Systems are designed to operate according to scenarios, implying that the design of the response of any unit to any event is based on assumptions about the operating scenario. In normal operation and in problem solving, all system units should assume the same scenario. The scenario is defined by the User function, in the operational context.

Operational procedure. A sequence of state transitions associated with a scenario.

Operational state. The active state, within an operational procedure.

Disturbance. The effect of a trigger, instigated by an operator, a hardware failure, a software bug, or an external threat.

Operational threats. Ideally, the system may recover from a disturbance easily and resume normal operation instantly, with minimal attention and effort by the operator. If a disturbance is not resolved instantly, it transforms into a threat.

Principles of resilience assurance

The guide proposes the following principles:

Prohibit mishaps. The design should not be tolerant to mishaps. Risky situations should be expected and the design should include means to protect from these risks.

Design scope. The specifications should include a definition of its boundaries. Within these boundaries, the specification of system behaviour should be complete.

Prohibit errors. The design should not be tolerant to any human errors. The design should assume the **Human Factors variant of Murphy's law**:

> *If the system enables the operators to fail, eventually they will!*

Usability. The interaction will be designed based on models of the user's and operator's behavior, to ensure that they understand its procedures and behaviour.

Situation awareness. The design should be based on the premise that the operator cannot trace the machine state reliably. It is the designer's responsibility to ensure that the operators are always aware of the system situation.

Load reduction. The design should consider all the user's main tasks, including those that are not related to the system operation. To prevent distraction, the design should reduce the mental load required for the system operation.

Defend against mistakes. The probability of users' and operators' mistakes in exceptional states is high. The design should incorporate means to defend against them.

Training. The operators should be trained to operate in exceptional conditions, and the design should propose means for such training.

Testing with users. The system validation should be based on testing with real users and operators, in real operational conditions.

Management responsibility. The organization in charge of the system operation should provide the operators with guidelines and procedures to prevent operational errors.

Exceptional situations

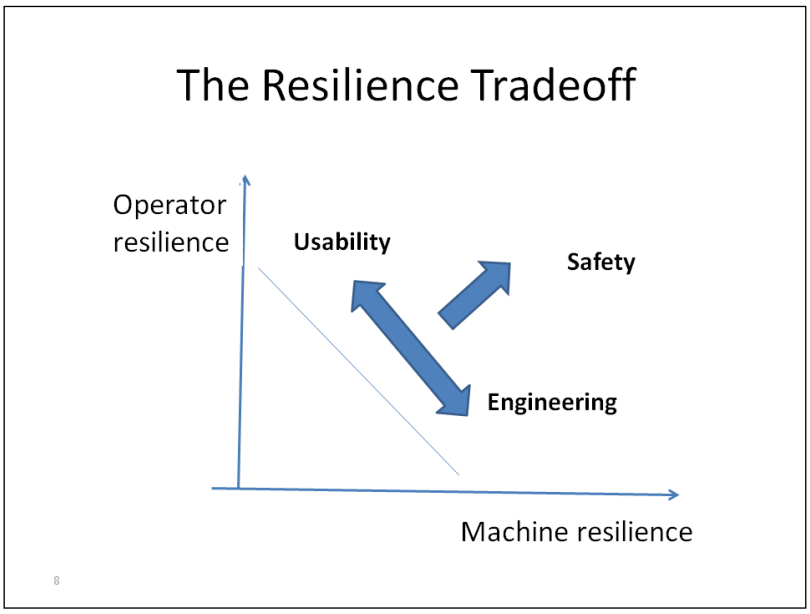
The term Exceptional Situations refers to the system situation resulting from exceptional events, such as component failure or inadvertent action. The problem with exceptional situations is that in regular system development, the support for these situations is typically lousy;

The complexity of the system behaviour in exceptional states is by orders of magnitude greater than the behaviour in normal states. Accordingly, the costs of designing and testing the procedures that handle exceptional states should be much higher the costs required to handle normal states.

Typically, the resources allocated to deal with exceptional situations are scarce. System design and testing is always constrained by budget and schedule. Considering these constrains, the development activities are typically prioritized, so that the initial focus is on the procedures that implement the primary functions, used in routine operation. Consequently, the specification, design and testing of the system behaviour in exceptional situations is lousy. In the testing, the operators do not have sufficient opportunities to practice the operation in exceptional situation. Under pressure for early delivery, the procedures that handle exceptional situations might hinder severe design mistakes. The design mistakes might be detected only after the system was deployed.

System behavior in exceptional situations

Because the behaviour of the human operators is unpredictable, system engineers do their best to automate as much as they can. However, the more reliable the automation, the less the human operator have opportunities to learn how to handle exceptions (Bainsbridge, 1983). For example, inadequate crew knowledge of automated aviation systems featured as a factor in more than 40 per cent of accidents between 2001 and 2009 (Learmount, 2011, download). The following chart from Zonnenshain & Harel (2013, download) depicts the human-machine trade-off dilemma:



The control-automation dilemma

Unexpected events

Because delivery time is always limited, the exceptional situations are typically error-prone, and the results of operating in exceptional situations are often unpredictable. Often, the trigger for the system malfunction is a normal event, arriving when the system is in an exceptional situation. The normal event may transit the system to an unexpected state (Perrow, 1984). When the operators encounter unfamiliar behaviour, they need extra time to understand the unusual situation, to explore possible reasons for the situation, to evaluate the available responses and to select the proper action. If the extra time required for responding properly is unavailable, and if the result is an accident or any significant damage, then the event instigating the problem is called an unexpected events (Harel & Weiss, 2011, download).

Unpredictable situations

Incidents are often associated with unpredictable situations. Eventually, operational resilience may be defined as the system persistence in unpredictable situations.

Unpredictable situations result from missing or wrong specifications, design mistakes, or implementation errors (mainly, software bugs). An example of common unpredictable situations is of operator-machine mode mismatch. In particular, situations when the human operator is not aware of a change in the machine situation are most common. Other types of unpredictable situations include:

- **Intra-system inconsistency**; exceptional machine state, irrelevant to a particular stage in a particular operational procedure. The inconsistency is between the machine's units, as was the case with the Therac 25 accidents (Casey, 1998),

- **Inter-system inconsistency**; exceptional context, not mentioned in the system requirement specification document. The inconsistency is between two sub systems, as was the case with the friendly-fire accident in Afghanistan, 2001 (Casey, 2006).

Inconsistent states are perceived as unpredictable, which means that the operators typically fail in identifying the source of the ambiguity.

The generation of incidents

The guide assumes that incidents develop according to the following scheme, described by Harel & Weiss (2011). In this scheme, the unexpected events are generated by deviation from the rules describing normal operation, followed by escalation, according to the following sequence:

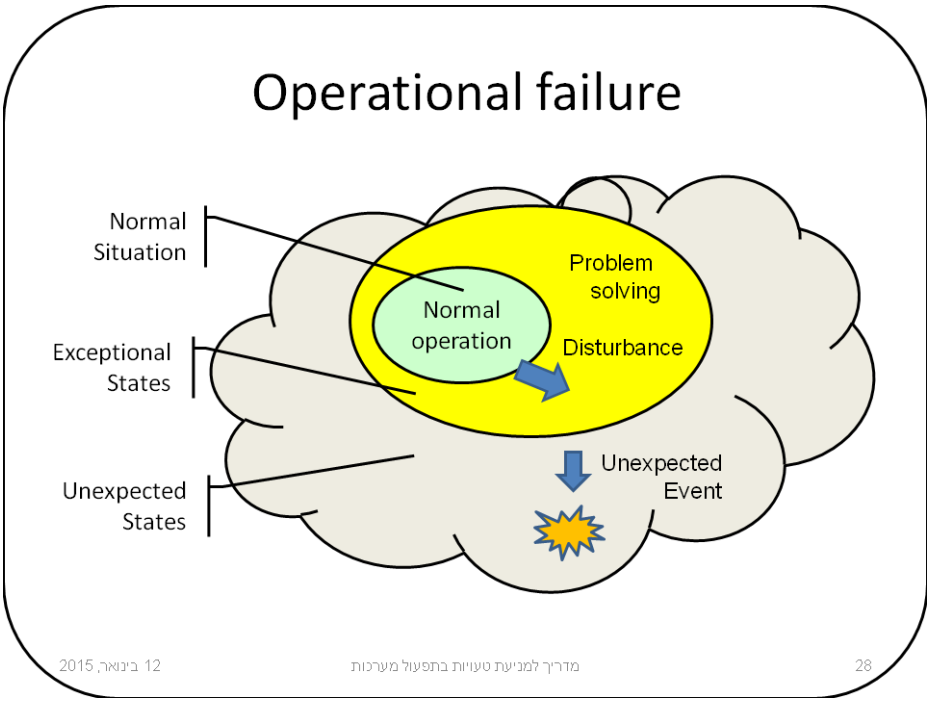
1. **Disruption**. A disturbance changes the operational state from normal to exceptional. Disturbances are expected when the system is in a normal state, and the system is prepared to handle them. Normally, the operators can identify the exceptional situation, and find a way to resume normal operation

2. **Disorientation**. Sometimes, the operators do not identify the exceptional situation (the case of latent condition), or fail to resume normal operation promptly (the case of lengthy troubleshooting and recovery). In both cases, the system remains in the exceptional operational state.

3. **Escalation**. Another disturbance arrives when the system is still in an exceptional state. The operational situation is unexpected. It is fuzzy, too complicated to handle.

4. **Incident**. When the operational situation is unexpected, the system response to any event is unpredictable. This is the point where the system is most likely to fail.

The various operational states and the generation of unexpected events are depicted in the following chart:



The generation of unexpected events

Three categories of operational states

The resilience model combines a model of common failures with common procedures for coping with the common failures. The model of common failures is described in terms of the operational states. The guide defines three categories of operational states, corresponding to three types of operational activities:

- **Normal states**, corresponding to normal operation
- **Exceptional states**, following a disturbance, resulting in a latent condition or in a lengthy recovery procedure
- **Unexpected states**, following another disturbance, while the system is in an exceptional state.

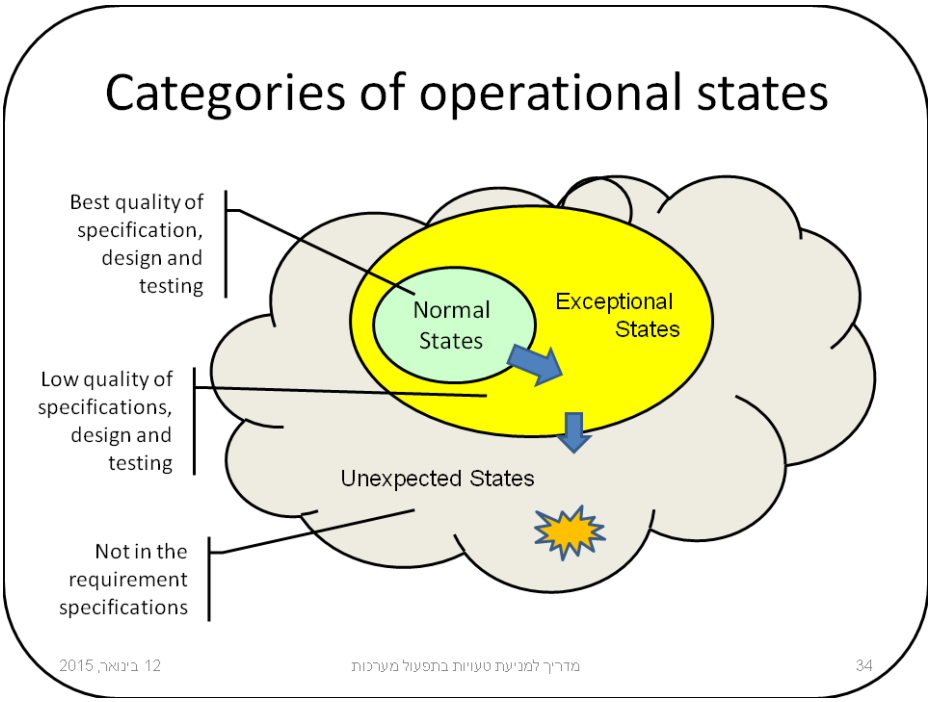
The quality of the interaction design

The quality of the interaction design is the result of investment in the specifications, design and testing, which depends on the category of the operational state:

- **Normal states**. Typical system development focuses on normal operation. Therefore most of the specification, design and testing resources are invested in developing these states.
- **Exceptional states**. Often, much of the activities involved in developing this part of the operational procedures are conducted informally, and the development is by trial and error. Consequently, the debugging is lengthy, and the results are of low quality.

- **Unexpected states**. Typically, these states are not mentioned in the specification at all. The design is concerned about preventing system crash, notifying the operators about the exceptional situation. Testing cannot be planned for the unexpected. The operators receive a message about something unfamiliar to them, and they can do nothing to cope with the situation.

The three categories are depicted in the following chart:



The quality of operational specifications

The resilience model

The resilience model is a framework for describing common operational failures. It is based on a model of human performance, in fuzzy context.

The resilience model described here is based on the preliminary model described by Zonnenshain & Harel (2011). The new model includes a definition of the design scope, referring to the state domain and activities associated with normal and exceptional situations.

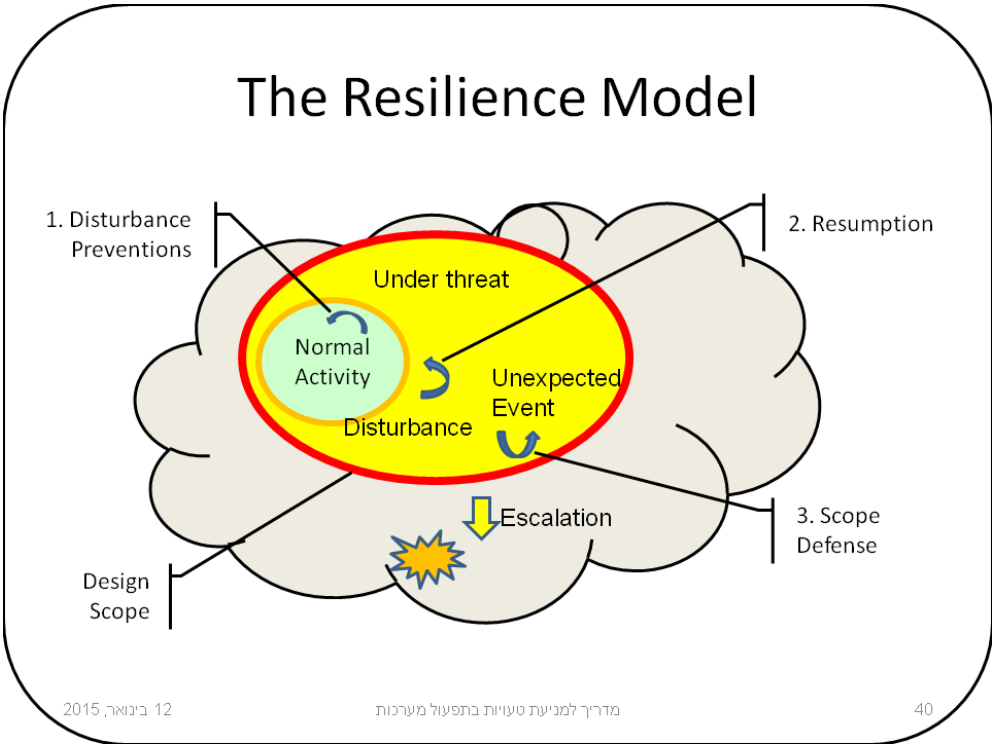
The system resilience is an emergent property, associated with three system properties: reliability, responsiveness and recovery capability. It relies on three common defences, typical of dealing with unexpected events:

- **Preventing disturbances**, in order to avoid the exceptional situations. This defence is applicable to preventing unexpected operator's actions and to coordinating the system units. However, it is not applicable to other sources of disturbances, such as component failure.

- **Recovery** from a disturbance: preventing latent conditions and ensuring prompt resumption from exceptional situations. This defence is applicable to component failures through effective troubleshooting procedures.

- **Preventing escalation**: enforcing staying in the design scope.

The resilience model with the three common defences is depicted in the following chart:



Common defenses

The STAMP Paradigm

According to this principle, safety issues are due to violations of (explicit or implicit) rules defining proper operation. To avoid safety issues, the operational rules should be stated explicitly, and the system should constrain its operation according to these rules.

The guidelines in this guide include instructions about designing a control unit in charge of handling the STAMP paradigm. Specifically, they include instructions for implementing the rules in the control unit, and for specifying the system response to deviations from the rules.

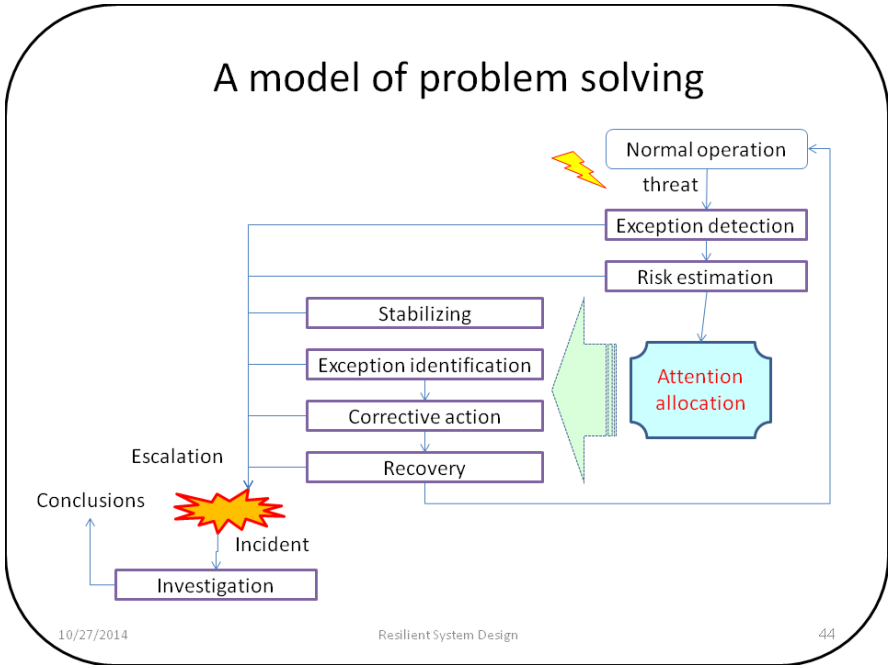
The design scope

The guide assumes that there is no practical way to specify the system behaviour in all possible situations and events. However, it can and should be specified completely for the states in the design scope, namely, the expected (both normal and exceptional) states.

In the figures above, the design scope is represented by the yellow area.

A model of Problem Solving

Threats are controlled by the human operators, assisted by the machine. Threat recovery is an interactive activity, in which the machine informs the operators about the situation, the operators integrate this information with their own information and knowledge, and act to overcome the threat. The following figure depicts the mental activities of the operators during problem solving:



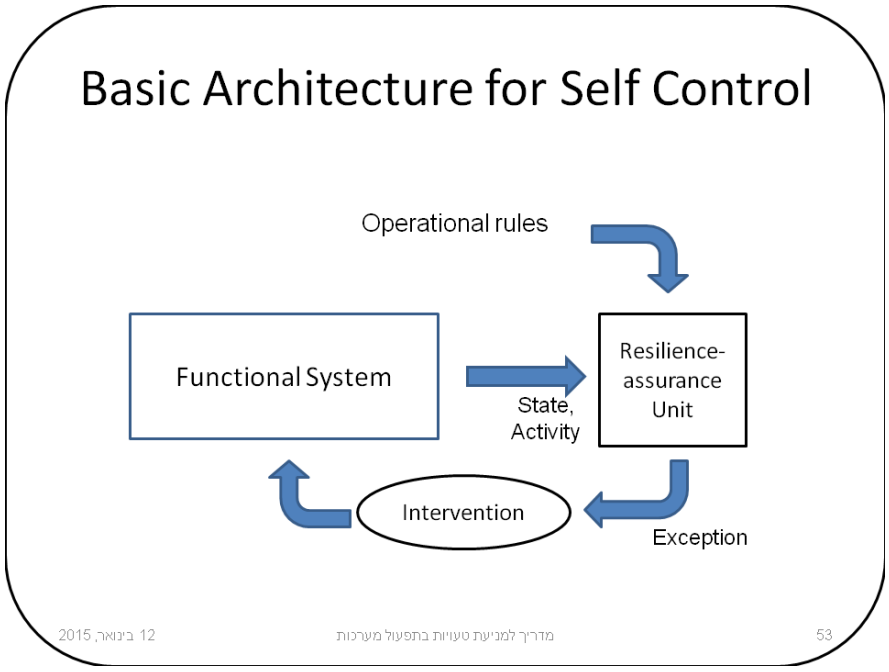
The operator's tasks in responding to alarms

The activities involved in handling these interactions consume attention and require intervention by the human operator. Because the operators' attention is required also to other problem solving activities, these mental activities are error-prone. Also, if any of the mental activities described above fail, a new event might enter the system, and the system might enter an ambiguous or inconsistent state.

Self Control of the system activity

The guide relies on the STAMP paradigm, namely, the system should constrain its operation according to operational rules, prohibiting problematic states. This implies that the requirement specifications documents should include detailed specification of recommended operational procedure, and the design should incorporate means to detect deviations from these procedures. Any deviation from these rules is considered a trigger of a potential incident, and the design should include means to identify these triggers and to respond properly.

The implementation is by a model of self control, as illustrated in the following figure:



Implementing the STAMP paradigm

Types of operational rules

The guide defines the following categories of rules:

- Event driven:
    - State transitions: the mapping: (Current State, Event) => New State
    - Event impact: the mapping: Event => Expected change in parameters
    - Process initiation: the mapping: (State, Time) => Expected change in parameters
- Ongoing:
    - Behaviour: the mapping: State => Expected change in parameters. Parameter values that should stay in a pre-defined range.
    - Compatibility: the mapping: Unit State => State

Swiss cheese metaphor

The resilience model offers two modes of employing the layers of the Swiss cheese metaphor:

- **Cumulative defences**: the traditional representation (commonly attributed to Reason; see Eurocontrol, 2006, download), describing the propagation from hazard to incident through means to avoid the incident, is used here to model the defenses described by the resilience model

- **Cumulative activity**: another representation, describing the propagation from threat to recovery through barriers preventing operation as intended, is used here to model the operator's mental activities during the recovery from an exceptional situation.

Firewalls

The system resilience may be obtained by designing three primary firewalls, (aka, slices of the Swiss-Cheese cumulative defences) comprising of defences against known failures.

- **Fault prevention**. The first firewall is about preventing certain faults by design. It is applicable to certain types of operational errors, such as activating the wrong function. The guidelines are borrowed from the discipline of cognitive engineering.

- **Escalation prevention**. The second firewall defends from faults that cannot be prevented by design, such as component failures, power failures and communication disruptions.

- **Learning from incidents**. The third firewall is about capturing and identifying the incident, and concluding about steps and means required to avoid similar incidents

2 Key Topics

This section describes key topics in resilience assurance

Behavior specification

The behaviour specification of the design scope may include a list of the expected states and a list of event-responses for each of these states.

Preventing operator's errors

This defence focuses on preventing operator's actions that divert the operational situation from normal to exceptional.

Free control execution, such as that used in learning by trial-and-error, is error-prone. The risk is of activating a function which not adequate to particular stage of the operational procedure in the active scenario. For example, a user of a home TV might unintentionally turn off the digital converter, instead of turning on the TV screen, as demonstrated by Zonnenshain & Harel (2009, download).

To prevent such errors, except for special cases, the commands available to the operator should be restricted to suit the stage in the operational procedure, which implements the task in the active scenario. This implies that the operator's control over the machine should be limited.

An important topic in error prevention is that of defining the level of automation. Automation can prevent certain operator errors, but on the other hand might prevent the operator's control

in exceptional situation, when human intervention is required. Therefore, the system should identify exceptional situations, notify the operators when they occur, and enable the operator to switch to manual operation.

Preventing mode errors

Mode errors are typical of controls overloaded with several functions, such that the particular function activated is dependent of the operational mode. For example, if the On-Off button of a home TV serves for turning on and off both the TV screen and the digital converter, then the unit activated is the one that matches the operational mode. In case of mode confusion, the users might unintentionally turn on or off the wrong unit.

Mode error can be prevented if (within the scope of the active scenario); each control invokes a single command, independent of the system state. For example, unintentional turning off of the digital converter may be prevented by assigning the On-Off switch the single command of controlling the TV screen (and not the digital converter).

Enforcing inter-unit consistency

Inconsistency may appear in various shapes. One common shape is when the operational scenario is not shared by all the units. When this happens, normal operation might result in the system moving to an exceptional state.

The guide proposes that all the system units should share the active scenario, and recommends that a special unit is nominated for enforcing it. The guidelines for scenario-based design are:

1. Formalize the operational scenarios
2. Specify the operational rules for each of the operational scenarios
3. Implement the operational scenario using a software variable, accessible by all the system units
4. Authorize the units about modifying the active scenario
5. Specify how the units should modify the active scenario

Preventing latent threats

A key concept emerging from the analysis of celebrated accidents is that of latent threats. The problem of latent threats was demonstrated in many accidents, such as that of the TMI. In that accident several valves did not function properly, due to operator's negligence or to mechanical failure.

One of the most challenging goals of resilience assurance is to prevent latent threats. Latent threats can be the result of component failure, operational error, software bug, misinformation

about inconsistent state transition or about communication problems, etc. If the operators are not aware of the threat, they might become aware of it only after it is too late, when the incident materializes.

To prevent latent threats, we need to specify rules about proper system operation, and we need to program the control unit to trace the operational state, to verify compliance with the context (operational scenario) and to alert in case of violation.

The guide recommends that a special control unit is allocated for detecting deviations from the design rules, such as those due to component failure. The control unit can check compliance of the operational activity with the operational procedures, according to the STAMP paradigm, and inform the alarm unit in cases of deviations from the constraints.

Detecting operator's errors

Not all the operator's errors may be prevented. Sometimes the operators are expected to deviate from the rules. For example, when the operators experience exceptional behavior, or when they encounter a risk due to an external event.

An operator may be authorized to activate special functions, for example, in emergency, which deviate from the operational procedures. However, the operators might also activate these special functions inadvertently, or by mistake. The guide recommends that the control unit should detect deviations from the operational procedures due to operator's errors.

Indicating component failure

All system components are liable to fail. In the design of warning systems, we need to consider all possible circumstances, such as a failure in a sensor or in the alarm system, and the failure of the operator to perceive the alarm (Weiler & Harel, 2011, download).

System designers are always under pressure to reduce the system complexity, in order to reduce its price, and to facilitate procedures for construction, installation, and maintenance. Typically, the stakeholders are not willing to give up functionality, and they incline to give up components used for resilience assurance: sensors, displays, indications, and sound alarms.

A first question that a designer should consider regarding adding a failure indicator is:

> What are the risks that a critical component fails, and nobody knows about it?

If those instances might issue a problem, then the second question is:

> Can we rely on the component MTBF?

MTBF is a statistics. What if the component happens to live in the tail of the distribution? Will the stakeholders tolerate black swans due to such failures?

If the stakeholders may tolerate a latent failure associated with the component, then it might be the case that the component is not essential for the system operation. It may be redundant, and it is a good idea to simplify the system by removing the component, and the associated functions, from the system design. Otherwise, if the stakeholders might be furious about a latent threat, the design should provide means to detect it and to report about it.

MTBI – The MTBF of Incidents

Let us assume now that the stakeholders are willing to consider taking the risk of occasional latent threats, for the sake of reducing the system price. It is a good idea to provide them with an estimate about the risk. Because the incidents are unpredictable, and because they are very sensitive to the circumstances (Hollnagel et al., 2006), we cannot provide reliable estimates of the costs of incidents. What we can do is calculate an estimate of the Mean Time Between Incidents (MTBI).

$$\text{MTBI} = 1 / \sum_{Component} \text{FR(Component)}$$

Where FR (Fault Rate) is

$$\text{FR(Component)} = 1 / \text{MTBF (Component)}$$

To get a rough idea about probable MTBIs, consider a system with 120 non-critical components, each component with an MTBF of ten years. Because these components are not regarded as critical, they are not equipped with safety indicators. Using the calculation as above, the estimated MTBI is 1 month. This estimate should be presented to the stakeholders, and they need to decide if they are willing to take the risk.

The implied guideline is that in the system there should be only two types of components: those which are essential, and those which are redundant. The essential components should be assigned with indication of failure, and the redundant components should be eliminated, to reduce complexity.

Detecting faults in the safety indicators

The additional components required to alert about component failure (sensors, algorithms, displays, sound alarms) are not only costly, but also risky, because they are liable to fail, providing opportunities for new kinds of incidents (as was the case with the PORV failure in the TMI accident).

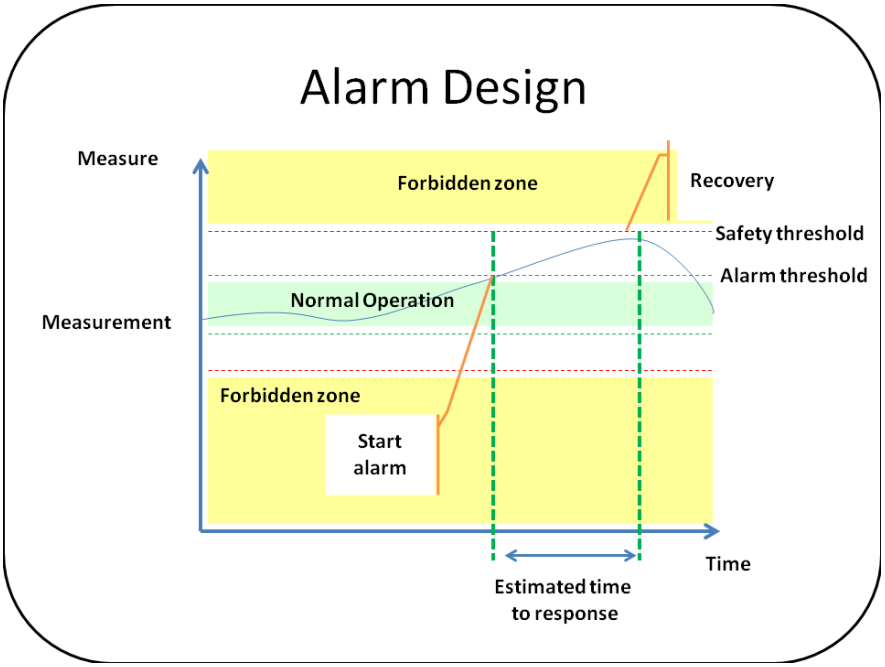
Safety indicators are liable to fail, and it is important to notify the operators when this happens. It is important to distinguish between the case of component failure and that of indicator failure. The guide recommends adding secondary indication of failure for each of the primary indicators, and provides tips for adding the secondary indicators without adding to the system complexity, by coding. Typically, when the indicator fails, it does not send signals to the control unit. Therefore, a most effective way to detect faults in the indicators is

by designing the indicators such that they always send signals about the component state, whether it functions properly or not.

Indirect fault detection

State sensors enable seamless identification of the component that failed, but not in all kinds of failure. Certain faults, such as leakage from a valve or a container, cannot be detected directly. Yet, such instances can be detected, because the fault can affect certain operational parameters. For example, a leakage from a container may affect the pressure in that container, resulting in deviation from the range of probable measurements. If a measurement is out of this range, then the exception control unit can detect it, and inform the alarm unit about it.

A simple example of employing this method is of the Statistical Process Control (SPC) used in production control. This is illustrated in the following figure:



Control by statistics

To enable fault detection, the operational specification should include descriptions of the system behaviour in cases of such faults. Fortunately, such descriptions may be available by employing FMEA techniques.

Operating under threat

The operator's mental activities involved in the operation under threat is described in the Model of Problem Solving above (go there). The operator's tasks are:

1. **Exception detection**: the specifications should include rules defining normal behavior in each scenario. The design should include means (such as the Threat Detector described in the Implementation section) to detect instances of violation of these rules.

2. **Threat recognition**: the specifications should include details about the risk levels of all known expected exceptions, and the alarms provided with the various risk levels. The design should ensure that an alarm signal is generated, and that the operators can recognize the risk level immediately, so that they can prioritize the new threat in their ongoing tasks.

3. **Troubleshooting**: the specifications should include procedures enabling the operator to identify the hazards, and understand better their risks. The design should include means for the operators to associate the procedures to the threats.

4. **Recovery**: the specifications should include procedures for fixing the problem, and the design should include means for the operators to find the proper procedure.

5. **Safe-mode operation**: the specification should include details about the conditions for safe-mode operation, the subset of operations essential in safe mode, and those that should be avoided.

6. **Resumption**: Normally, resumption should be allowed as soon as the system situation resumes compliance with the operational rules. Exceptions to this guideline might better be transformed to changes in the constraints.

Troubleshooting faults of critical components

After a fault was detected, the operators need to identify the source, namely, the particular component that failed. The problem is that sometimes the operators cannot identify the failed component based on the attributes of the alarm, because they were not trained to recognize components by associated alarms, or because they did not experience the alarm often enough to remember the particular system behavior.

Another problem is when the same behaviour may be due to various sources. In the example of pressure decrease in a container, the decrease may be due to leakage from the container or from one of the valves, unexpected state of a valve, and more. This makes the troubleshooting procedure very tedious and too slow (as was the case in the PORV of the TMI accident).

To facilitate the recovery procedure, the system behavior should represent the failure source uniquely. The mapping from failure source to system behavior should be isomorphic. The operational rules should be specified in such details to enable modelling the system behavior in case of failure. This kind of models may enable identify the source for the alarm, by measurements of changes in the system parameters (such as pressure and temperature in the container). Such detailed descriptions may be obtained by applying FMEA methods, and using simulation.

Escalation prevention

According to the Resilience model, escalation is the result of getting out of the design scope. To enforce staying in the design scope, the design should restrict and control the acceptable events when in exceptional situations to the bare operational necessities. In case of delay in resuming normal operation, the system operation should change to safe-mode operation.

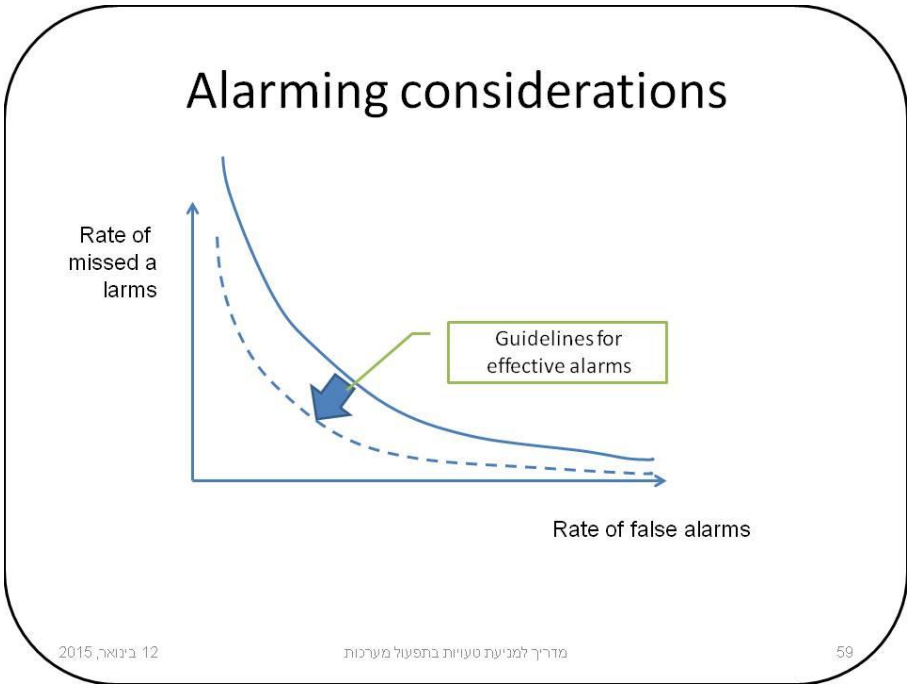
The guidelines are based on explicit definition of normal system operation, formal rules for constraining the system behavior, and a resilience-oriented architecture supporting graceful reaction to deviations from the constrains. The only events that may be received when in exceptional situations are those required to resume normal operation and to keep the system alive. In case of emergency operation, the design should check that the system remains in the design scope. Otherwise, the system should use a safety net, which implies operating in emergency mode. (Zonnenshain & Harel, 2013, download).

Comparing design options

Designers often need to trade between design options. For example, in setting the alarm threshold, we need to balance the probability of missed alarms with that of false alarms.

Sometimes, a shift in the design goal may result in better results. In the example of alarm design, the shift may be by specifying a procedure for notification and confirmation, according to the practices of responding to the alarm, which reduces or even prevent nuisance, without compromising low rate of missed alarms. An example of such a shift is demonstrated by the standard ANSI/ISA 18.2, about managing alarms in the process industry.

The guide presents guidelines for better engineering, as demonstrated in the following chart:



Improving the conditions for design tradeoff

Resilience verification

Resilience verification assumes that the operational rules are well defined, and the verification goal is to ensure that operators can tackle all exceptional situations. The design should include special means to simulate various exceptional conditions, and the testing procedures should include usability testing in exceptional situations.

Emergency operation

If we knew at the time of the requirement specification all about all the exceptional situations, and on the preferred reaction, then we could fully automate the system behavior. Instead of letting the operators make their mistakes, we could let the machine decide how to act.

Because not all the operational details are known in design time, it is a common practice to enable the operators to override a predefined procedure "just in case" that the operator needs a sequence that the designer did not anticipate. This is especially applicable to emergency operation. The problem is that in emergency, when under stress, operators are not creative. They cannot find the proper solution. Instead they take the action that they are used to take, the one they used during normal operation (Bainbridge, 1983)

The guide recommends that overriding the predefined procedures is restricted to special safety-critical functions. The guide recommends providing warning to the operators about the risks of activating the emergency functions. The guide recommends about features of these warnings, for ensuring that the operators consider them even when under stress. Special rules should be specified and designed to avoid false alarms. The effect on the system behaviour should be examined carefully.

Resilience validation

The purpose of resilience validation is to identify faults in the operational rules. Two types of faults should be targeted:

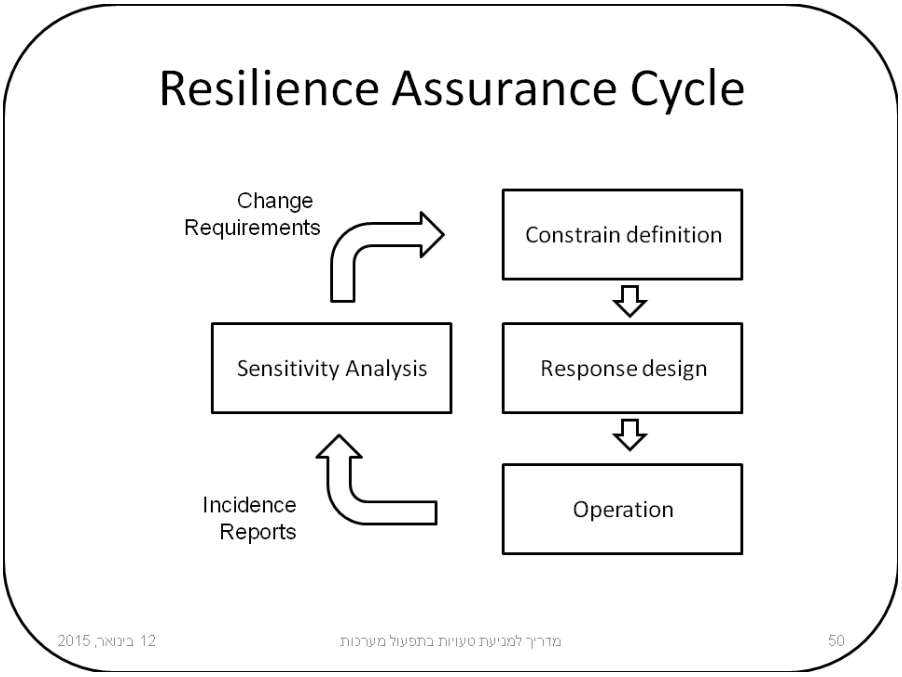
- Rules which are too restrictive, disabling execution of essential tasks
- Rules which are too permissive, enabling risky activities.

The design should provide means for the operators to report on risks that they experience during the operation. The reports should include a description of the operator's intention and the history of the recent system behavior. Also, the design should include means for examination of the development of the system behavior, such as trend presentation.

Iterative Resilience Assurance

In resilience-oriented development, the operational specification documents include definitions of the operational rules, which are based on the knowledge of experts, and obtained by procedures of system analysis. Mistake in these definitions are detected during the operation, in the form of incidents: Strict rules will result in false incidents. Missing rules will result in latent failures.

It should be expected that during the operation it becomes evident that certain rules need to be changed. These changes should be managed in a controlled procedure. The system resilience develops gradually, through cycles triggered by incidents, and followed by failure identification, concluding and implementing the changes, as demonstrated by the following diagram:



Tuning the constraints

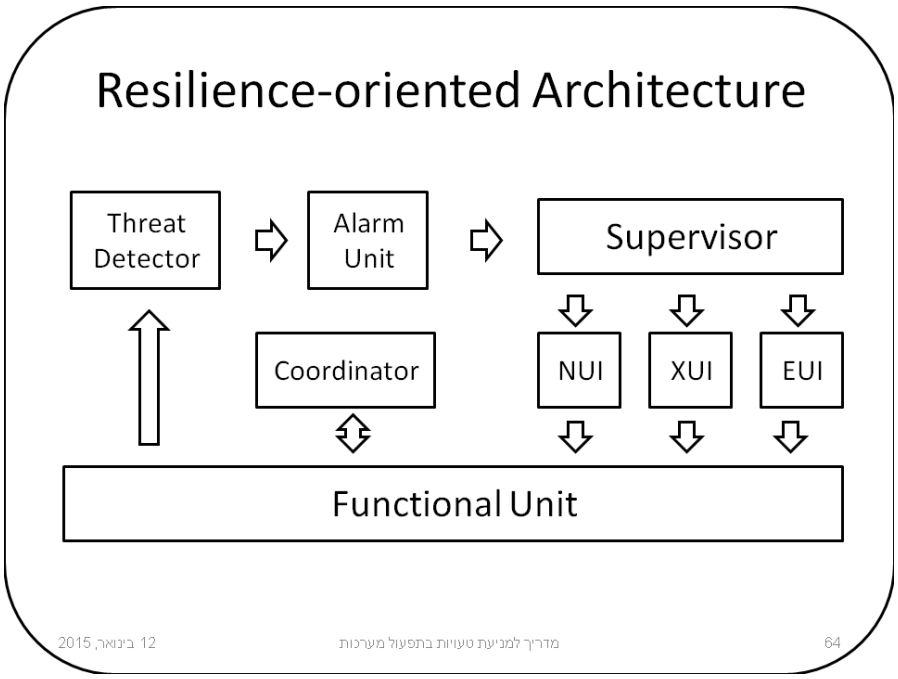
Special tools for incident reporting and information sharing may facilitate the shift from blame-and-punish culture to safety culture. The guide recommends on developing probes, enabling capturing of incidents or accidents. The guidelines for probe design are about capturing instances of constrain violation, namely, deviation from the rules. Two levels of probes are defined:

- High resolution incidents are about instances of changing from normal to exceptional situations
- Low resolution incidents are about instances of changing to unexpected situations.

The output of each cycle is a change requirement document.

Implementation

The guide recommends on an architecture suited for resilience assurance, demonstrated by the following block diagram:



Implementation example

This architecture enables modular development of main resilience features, as follows:

- A coordination unit, in charge of keeping track of the active scenario and of informing the other units about it, which is required for consistency assurance
- A threat detector, in charge of testing the system compliance with the rules and notifying on violations
- An alarm unit, in charge of informing the operators about the situation and guiding them in the problem solving
- Interaction managers: for managing the interaction in normal (NUI), exceptional (XUI) and emergency (EUI) operation.
- A supervisor unit defines the operational situation (normal, exceptional, emergency) and activates the proper interaction unit accordingly.