www.business-systems-review.org Business Systems Review ISSN: 2280-3866
Volume 1 – Issue 1

Controlling hazards and safety in complex systems: a multi-layered part-whole approach to system safety[i]

Luca Pazzi
Assistant Professor of Information Elaboration Systems. Department of Engineering "Enzo Ferrari". University of Modena and Reggio Emilia, Modena, Italy.
e-mail: luca.pazzi@unimore.it

Received September 30, 2012 / revised November 14 / accepted November 15 / published online November 16, 2012.
DOI: 10.7350/BSR.A17.2012 – URL: http://dx.medra.org/10.7350/BSR.A17.2012

ABSTRACT

The behavior of complex dependable systems poses severe safety issues due to hazards which may result from incorrect and unpredictable behavior. In order to prevent such hazards, system behavior has to be specified and checked incrementally, in order to defeat the overall system's complexity. Modularity in system design is however not trivial due to the intrinsic monolithic nature of the control loop, typical of such systems. An additional problem is given by the fact that the current modeling paradigm tends at introducing additional interactive complexity due to the direct communication and synchronization mechanism among decomposed modules. It can be shown, however, that modular decomposition is feasible by revising the current communication and interaction paradigm. Physical interactions in physical systems denote in fact less evident conceptual structures, which host the overall interaction and synchronization knowledge among the component parts. By introducing additional system entities with the aim of hosting such knowledge in a localized and compact manner, we obtain a part-whole hierarchy of systems, called holarchy. Such systems are, at the same time, both parts and wholes within a holarchy, thus giving a formal characterization to Koestler's holons.

Keywords: System-safety, Holonic frameworks, State-based modeling, Statecharts, PW-Statecharts.

1. INTRODUCTION

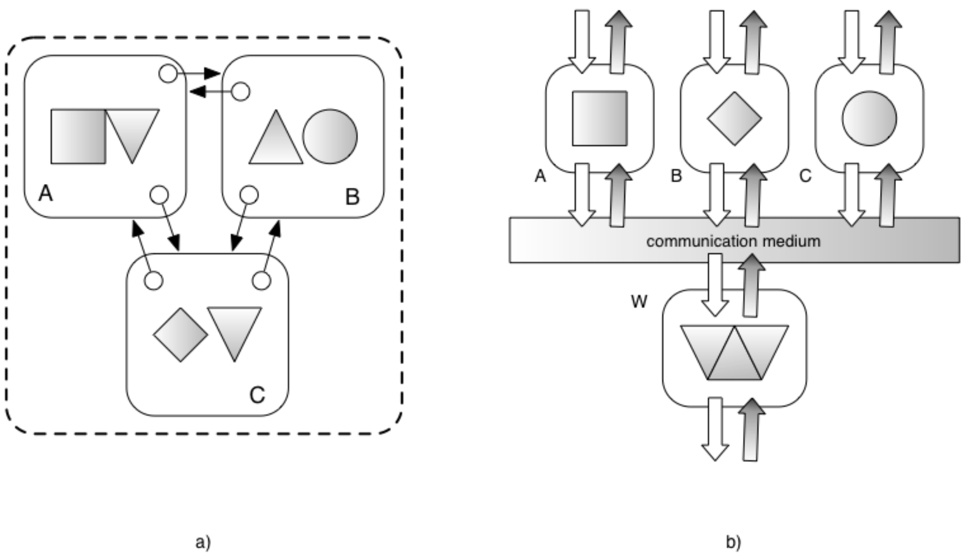
Three different kinds of systemic failures, which in turn may become system accidents, are possible: "faults", "failures" and "errors". A fault is a primary error within a system component. An error is a deviation from the expected behavior of the system, which may result in a global system failure.
Since the joint behavior of components make the system behavior, it is evident that any progress in system-based safety calls for a deep understanding of the way systems, whether natural or

artificial, are made up of components. Typically, system components interact in order to achieve a global task. It is therefore plausible to hypothesize that system structure play a crucial role in understanding how components' faults propagate emergent system behavior (Smith 1989) and may therefore result in a global system failure (Hammer 1980). The paper addresses such hypothesis and shows that well known fault management techniques may be framed into a hierarchical context. Such hierarchical arrangement, allows to bring new light into both safety related methodologies and, at the same time, into the very nature of controlled systems.
Holons, introduced by Arthur Koestler in his book "*The Ghost in the Machine*" (Koestler, 1976) are entities which are, at the same time, both parts and wholes. Accordingly, complex phenomena and entities can be decomposed into part/whole hierarchies, named holarchies, with holon nodes at each level. The main interest in the holonic approach lies in the fact that it reconciles both the holistic and the reductionist view in systems analysis. Aim of this paper is to show their usefulness in partitioning and reducing the complexity of dependable behavior in safety-critical systems employed in energy production systems.
By the reductionist view, which dates back to Descartes, a complex system can be analysed by "reduction" to its fundamental parts. Analytic reduction is the main weapon system scientists possess in order to control system complexity by dividing it into less complex distinct parts. Three important assumptions underlie the reductionist process, in order for it to be effective both in the analysis and synthesis phases (Leveson, 2011):

1. The division into parts will not distort the phenomenon being observed;
2. Components are the same when examined apart as when playing their part in the whole;
3. The interactions among the components are separate from the behavior of the components themselves.

Although the assumptions above are reasonable for most observable systems, they still pose at least two open problems when such systems have to be modelled within a computer-based context.
The first problem consists in finding the exact system boundary, in order to have self-contained systems, which behave in the same way when taken apart and when assembled into more complex systems. The second problem deals instead with providing assembled system models by suitable modular constructs, in order to host separately the component subsystems and the interactions among them. The third assumption of the reductionist approach states indeed that mutual interactions amongst systems do not belong to the interacting systems: this is a reasonable requirement, otherwise the reduction into parts will necessarily produce subsystems which are not the same when taken alone or within a larger whole, thus contravening the first requirement.
The subtle point in solving both problems is essentially that, – as pointed out by the third reductionist requirement – "interactions" have to be modelled separately from the behavior of the component themselves. In other words, the joint behavior which pertains the aggregation of separate systems has to be modelled separately by a specific behavioral construct. This in turn requires that systems have to be free from the behavioral information pertaining their aggregation in a specific whole. In other words, the only behavioral information modelled within a system has to be the one pertaining such a system. In order to achieve such a result, structural guidelines easily follow. On one hand modelled systems have to expose an interface to outer composition contexts which allows external behavioral constructs to act on them; on the other hand, system models have to host not only component systems, but also a specific construct which specifies the behavioral semantics pertaining their composition.

Figure 1. Aggregation of 3 objects modelling



In figure 1, the aggregation of three objects can be modelled implicitly by three mutually referring objects (a). Behavior is depicted within objects by geometric shapes. Triangles represent the associative behavior spread through the objects. The same aggregation can be modelled also by a holarchy of four holons (b), where holon W gathers the global associative (triangle-shaped) behaviour.

Current modelling paradigms fail in achieving such a separation. In the rest of the paper it is observed indeed that either: (a) the behavioural information pertaining aggregation is spread into the aggregated parts or (b) it can be hosted within the system having the aggregated systems as parts. In the latter case the whole is said to be modelled explicitly, which in turns means that both the whole and the constituent parts are easy to understand, reuse, extend and so on. Figure 1 compares the two approaches.

1.1 Implicit versus explicit system modelling

A physical system is assembled from a set of physical components, which exercise physical control one upon another. For example, consider a simple heating appliance pilot light, which is a small flame used to start a furnace, which is controlled by a thermocouple. The ignition process starts by a button being pressed, which closes an electrical circuit an in turn charges a condenser while keeping, at the same time, an electrically-operated gas valve open. As soon as the condenser is charged a spark is emitted and the pilot flame is possibly lit. In case the flame is not lit, the process may be repeated different times by having the condenser recharge. When the pilot light is lit, the thermocouple produces a voltage, which keeps open the main supply valve that feeds gas to the heating appliance. So long as the pilot flame remains lit, the thermocouple remains hot, and the pilot gas valve is held open. In case the pilot light goes out, temperature in the thermocouple falls, causing the voltage to drop and the valve to close.

A set of mutually related devices may, globally, exhibit a system behavior by having direct physical, typically mechanical, interactions. The pilot flame example depicts physical entities, which interact through physical processes (mechanical, electrical, thermo-electrical). Such processes cause state changes in related components. Chains of component state changes are at the basis of the global system behavior. However, a different view is possible, since each chain of state transitions at the component level may be seen as a single state transition at the system level. In the same way, the state changes in the system before and after each casual chain of state changes can be seen as a single state change.

With the advent of electromagnetic devices, control has been exercised by voltage in electrical circuits, as in the case of the electromechanical valve of the pilot light example. The subsequent advent of digital controllers and field buses allowed actuating physical state changes by digital signals. Conversely, specific devices called sensors, which generate digital signals accordingly, can sense state changes in the environment.

The consequence of technological advance is therefore that a system of interacting components may be implemented by a set of devices that act one upon the by directly exchanging signals embodying both messages and state information. Direct communication among system components is however not the only way to implement a system behavior. In [] it is shown in fact that the same system behavior may be modelled either as above, by direct communication among system components, or through an explicit additional entity representing the system being modelled, which has the system components as parts and hosts the system behavior as a whole. The two approaches have been named implicit versus explicit system modelling.

It is possible to view systems interactions as being the primary structure of a system. By the former view, the system is simply a network of mutual interactions among the subsystems making it. Such a view emphasizes interactive complexity and tight coupling of system control modules, each residing within a single component of the system. Such an approach has been named implicit modelling in system design. Implicit modelling brings evident problems in software engineering terms. Each component typically hosts a single software controller. In order to achieve a useful system behavior the different controllers have to synchronize by mutually exchanging control signals, which encode event and state information.

In Pazzi (1999) it is shown that direct communication, albeit inspired by natural systems, is not the only way to implement a digitally controlled system. Although it is presumable that a system modelled by direct interactions can be equivalently modelled by an explicit structure hosting its behavior as well as its component parts, the explicit approach has notable advantages in terms of quality factors, which defeat its intrinsic complexity. Cognitive psychology established that the perception of complex structured entities as a single entity firmly depends on the observer's point of view and affect the overall performance in a problem solving context (Norman, 1993). Similarly Woods (1995) claims that there are no neutral representations, since different representations either increase or decrease the overall problem complexity.

Interactive complexity represents a threat to safety since the level of interactions reaches the point where failures cannot be planned, understood, anticipated, and guarded against (Leveson, 2011). Such systems can be expected to have many such unanticipated interactions, each potentially leading to a failure, and eventually making them vulnerable to accidents (Perrow, 1999).

On the other hand, systems which make use of little or no knowledge of the behavior of other separate components have a minor impact on each other in case of failure. If what happens in one part has little impact on another part, the system is said to be "loosely coupled", while it is said "tightly coupled" in the opposite case. This paves the way for a modular concept of safety.

1.2 Towards a modular concept of safety

Ensuring safety means "reducing accidents throughout the life cycle of a system" (Storey, 1996; Leveson, 2011), which in turn means preventing, eliminating and controlling hazards (Becker et al., 2006), i.e., situations that have the potential to pose threat to life, health, property, or environment. Safety deals with the limitation of hazards and the management of their effects, and depends on system failures, that are failures due to the mutual arrangement and interactions of the components of the system. Such components may, possibly but not necessarily, have failed on their turn.

It is customary in the literature to distinguish between failures of the whole system and failure due to single components, called faults. Faults are underlying defects, imperfections, or flaws that have the potential to cause system failures. A fault is something within a system component, which may cause the system not to meet one of the functions for which it has been designed, resulting in a deviation from the expected system behavior and possibly in a system failure. The failure of a system can be seen as a fault in the context of the larger system of which the system is component.

Safety should not be confused with “reliability”. Reliability is the property of a system to perform a specific function over a definite period of time, such function being part of the specification of the systemic correct behaviour (for example: “the lamp must turn on once the switch closes a circuit”). Apparently, the more reliable a component is, the more reliable the system having that component as part should be. It can be observed, instead, that even a system assembled from reliable (working) components may undergo a systemic failure, due to unforeseen interactions among its components. A safe device may, therefore, be harmful once it interacts incorrectly with other reliable devices.

Two switches arranged in series within a circuit give a trivial example of systemic failure with no fault components. Pupils learn that, in the series case, sometimes one of the two switches does not affect the light bulb, even if the two switches and the other components of the circuit work correctly. A more complex example is given in (Pazzi & Pradelli 2008) where an assemblage of medical devices, consisting of a blood pump, a valve and a patient’s pressure monitor, fails under specific circumstances, even if its components work perfectly.

In other words, a system is safe if, recursively:

1. It is assembled from safe components;
2. Its components interact in a safe manner.

Safety is both a hierarchical and a behavioral property of systems. Safety is a hierarchical property since safety of subsystems impacts on the safety of super systems, along chains of part-of hierarchical arrangements. Safety is, additionally, a behavioral notion since, as observed, not only a system may be unsafe due to the interaction of different safe/reliable subsystems, but, on the opposite, unsafe/unreliable subsystems may be arranged within a system in such a way to be safe, typically by adding redundant components. Understanding dynamical relationships among systems arranged in part-of hierarchies is, therefore, a basic step in understanding how safety management policies can be effectively implemented. Moreover, as first observed in (Pazzi & Pradelli, 2010) different safety management policies (Carter et al., 1987) fit naturally at different hierarchical levels.

1.3 Structure of the paper

Aim of this paper is to investigate whether a sound notion of system modularity drastically reduces the complexity of safety management policies.

Section 2 discusses and compares two general principles of entity composition, interaction and synchronization. Such a distinction calls for the modelling of explicit entities in place of the observed mutual behavioral relationships. Part- Whole Statecharts can be shown to implement the concept of Holon, thus satisfying such a requirement. Such formalism is used, through the case study in dependable energy production systems of Section 3, in order to allow a clean and effective modelling of hierarchical fault management strategies at different hierarchical level.

2. FROM SYSTEMS TO HOLONS

Arthur Koestler coined the term holon for a part-whole construct that can be seen, at the same time, as a component of one or more higher level systems or as whole which has other lower level holons as parts. Holons have been used as an effective paradigm for modelling flexible (Dominici, 2012; Dominici & Palumbo, 2012) and multi-agent production systems (Dominici et al. 2010).

In this Section it is shown that holonic modularization can be introduced effectively in state-based control paradigm, by revising the communication mechanism amongst interacting entities along two main directions.

On one hand it is suggested to forbid any mutual communication amongst peer modules. Such a requirement springs from basic software engineering considerations, requiring modules to be self-contained and represents the fundamental notion over which the explicit approach in the modelling of wholes is based (Pazzi, 1999).

On the other hand, the previous requirement implies to introduce additional "bridging" systems in order to allow communication amongst basic systems, which are forbidden to communicate directly. Such systems become the right place in order to host the semantics of interaction. In other words, the need to express the semantics of interaction brings to new entities, often not readily apparent. Section 2.1 describes such a modularization process in state-based control. Such entities are nothing but holons playing the role of wholes, linking the original systems, which become holons playing the role of parts.

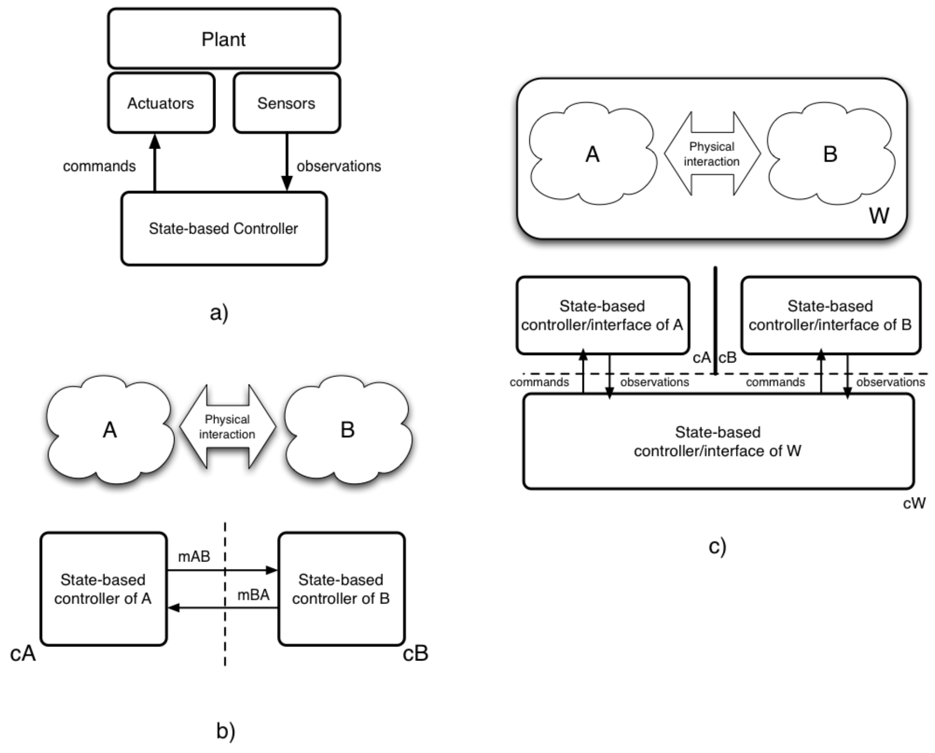
For example in Figure 3-(a) different subsystems interact by exchanging command events: such an interaction models the global behavior of the system, namely the alternate timed behavior of two counter-rotating engines. As shown in Section 2.2, such a behavior can itself be described by a state machine having two states and two state transitions linking them. In general, the emergent behavior involving different interacting systems can be hosted in holons playing the role of wholes, while the original systems become holons playing the role of parts.

Finally, parts have to be logically independent from emergent wholes. This requires holons to implement interfaces in order to be controllable when playing the role of parts. On the other hand, parts do not have to host details concerning the whole, in order to allow full composition-ability in different, unforeseeable, contexts. This introduces an asymmetry in the vertical, whole to parts, composition process, since parts do not know the whole, but the whole must be able to know the parts in order to control them.

2.1 Modularization of the control loop

State-based control is achieved through a closed loop, such as the one depicted in Figure 2-(a), where a plant, that is a physical system, is sensed and acted through sensor and actuator devices, through the exchange of events and the knowledge of the current state of each device (Kopetz, 2011). Each device is associated to a state machine, acting as an interface to the underlying hardware. Such interfaces show the current state of the device to which they are associated, together with the available actions that can be acted on it and the resulting state of such actions, if successful.

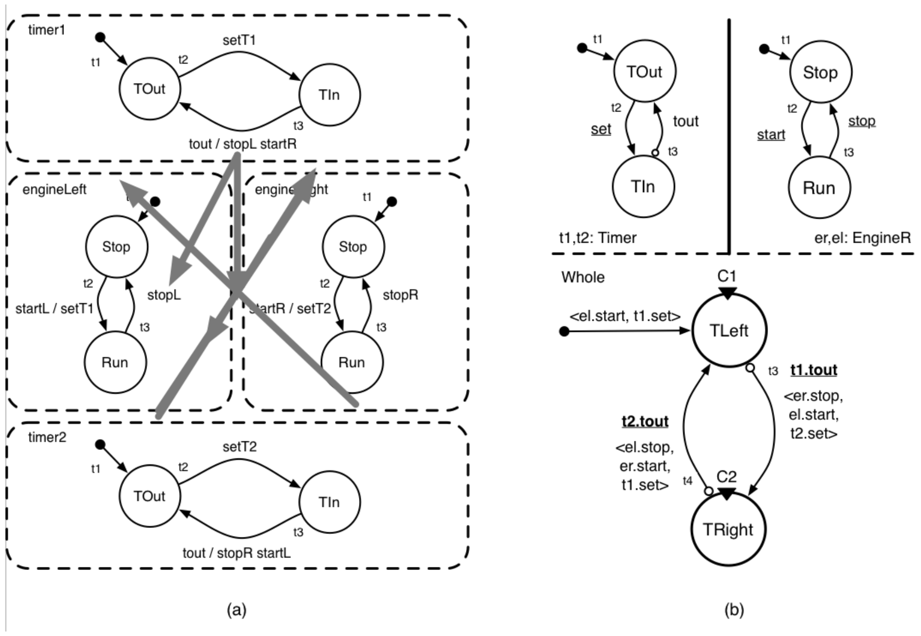
Figure 2: Modularization of closed loop control


Plant
Actuators
Sensors
commands
observations
State-based Controller
a)
A
Physical interaction
B
State-based controller of A
State-based controller of B
mAB
mBA
cA
cB
b)
W
State-based controller/interface of A
State-based controller/interface of B
cA
cB
commands
observations
commands
observations
State-based controller/interface of W
cW
c)

In figure 2: modularization of closed loop control (a) requires to keep into account not only the existing subsystems in the physical plant (b), but also to introduce specific modules in order to host mutual interactions among them, such as *cW* in (c).
In other words the interface of a device is a state diagram, and the device behavior can be abstracted as the one belonging to a state machine. Any of such state diagrams should be also able to signal to the controller not only its current state, but also to signal other events that may be related to the behavior. For example, a “smart” lamp should be able to signal not only its faults, but also maintenance or substitution requests.
The state machine hosted within the state-based controller changes its state according to the signal emitted by the system through the sensors; vice versa the controller issues signals towards the system, which changes its physical state through actuators.
Control is therefore achieved through state machine interactions: the controller state machine sends command events towards interface state machines, which, on their turn, send feedback events to the controller. It is possible to view the current state of the sensors and the actuators as representing the global state of the plant: by such a view the controller takes control decisions depending on the current global state of the plant.
The problem with the approach of Figure 2-(a) is that it is monolithic, that is, it is not clear how to divide the controller into modules in order to partition its complexity.
The customary approach towards modularization consists in distinguishing subsystems in the physical plant and having a corresponding control module in the state-based controller. This seems reasonable, since most systems have parts that are naturally distinguishable, for the reason

that either they are assembled from standalone devices or they perform some function. It seems therefore realistic to structure system control around physical or functional parts.
The problem consists in the fact that subsystems are not isolated, that is they exhibit necessarily, as part of the system, some sort of coordinated behavior. Consider for example figure 2-(b), where two systems A and B interact at the physical level (consider for example a cogwheel A acting on a cogwheel B). The two corresponding state-based interfaces associated to the control of A and B have therefore to take into account not only the control of the corresponding physical entities, but have also to synchronize their internal behavior in order to mimic the physical interaction among A and B. As shown in figure 2-(b), the two controllers have therefore to exchange synchronization messages mAB and mBA.
If, on one hand, either a mutual interaction is observed among two systems, A and B, having two different state machines or such an interaction has to be enforced among them, having two corresponding modules on the control side of the loop, and enforcing control through them, may incur to the problems discussed above. The problems observed may be solved by adopting an explicit modelling approach (Pazzi, 1999). As shown in figure 2-(c), we forbid the two controllers *cA* and *cB* to exchange synchronization messages (a solid line is placed between them, in place of the dashed one). By providing instead each controller state machine with a suitable interface, we decouple controller modules one from another by adding an additional controller module *cW*. Such a module is again a state machine, which provides control to the subsystems A and B, and can be controlled on its turn through its own interface. Since controller *cW* models the physical interactions amongst A and B, it seems evident that an additional systems W, embodying the former interacting two systems A and B, has some level of objective existence, although it may not be readily visible.

Figure 3. The modelling of the behavior of two alternate counterotating engines by traditional Statecharts (a) and by Part-Whole Statecharts (b).



2.2 Holons implemented by Part-Whole Statecharts

Part-Whole Statecharts (Pazzi, 1997; Pazzi, 2000) (shortened either as PW Statecharts or PWS) were created with a radical commitment towards state-based modularity. The idea behind their introduction was to have a formalism which encapsulates a compound behavior, forcing the modeller to expose an interface representing the composed behavior as a whole. Such a behavior is annotated by state propositions which are verified directly in the specification phase (Pazzi, 2008). In the paper a brief account of PWS syntax and semantics is given through the running example in the next Section. Part-Whole Statecharts require making explicit the interaction among the behavior of parallel state based processes, i.e. to introduce an explicit module containing the state diagram depicting such a behavior. A Part-Whole Statechart diagram consists of two main sections, as shown in figure 3-(b), the upper one hosting a set of component state machines, the lower one a single state machine representing the global system behavior, named "whole" (referred to in the rest of the paper as either the whole or the whole section of the PWS). By such a "connecting behavior", the description of the behavior of the system as a whole is made explicit. For example in Figure 3-(a) different subsystems interact by exchanging command events: such an interaction models the global behavior of the system, namely the alternate timed behavior of two rotating engines. Timing specifications are modelled through timer state machines, such as *t1* and *t2* in the picture, which are initially set to a time-in (*Tin*) state and move autonomously to a time-out (*Tout*) a state once a fixed and definite time interval has expired.

The same behavior can be modelled explicitly, as observed, by the state machine in the lower part of the Part-Whole state diagram of Figure 3-(b). It consists of two states, *TLeft* and *TRight,* each one describing a global state of the system through state propositions *C1* and *C2* associated, respectively, to states *TLeft* and *TRight*:

$$\mathbf{C1 = t1.TIn \land el.Stop \land er.Run} \quad (1)$$

$$\mathbf{C2 = t2.TIn \land er.Stop \land el.Run} \quad (2)$$

Since the system is at any time either in state *TLeft* or *TRight*, but not in both, joint logical and timing properties of the components can be easily inferred, for example "the two engines are not active at the same time" and "each engine is active only when its associated timer is in state time in". It is also possible to verify that state transitions linking the two states maintain the validity of the associated state propositions, as described in (Pazzi, 2012).

The state machine within a PWS plays the double role of being both an interface to the complex system which has the current system as part and to implement a complex synchronized behavior in the component machines. The two roles played by the state behavior thus define the so called "*Janus*", i.e. double face, aspect of the holonic paradigm.

3. THE STEAM BOILER SPECIFICATION PROBLEM

The steam boiler specification problem has been used for years as a test for comparing specification formalisms. It deals with the most safety-critical part of a nuclear energy production plant, the steam boiler which produces steam subtracting heat from the nuclear reaction. The steam boiler must be continuously supplied with water through some pumps: the quantity of water present when the steam-boiler is working has to be neither too low nor too

high; otherwise the steam-boiler or the turbine operated by the steam flow might be seriously affected.
Although the problem seems relatively simple, safety constraints raise its complexity. A strong notion of modularized control is therefore needed. We show that the holonic framework, through the PWS approach, is well suited in structuring control specifications in such a way that safety policies may be enforced at different hierarchical levels, each corresponding to some physical or logical entity in the domain.

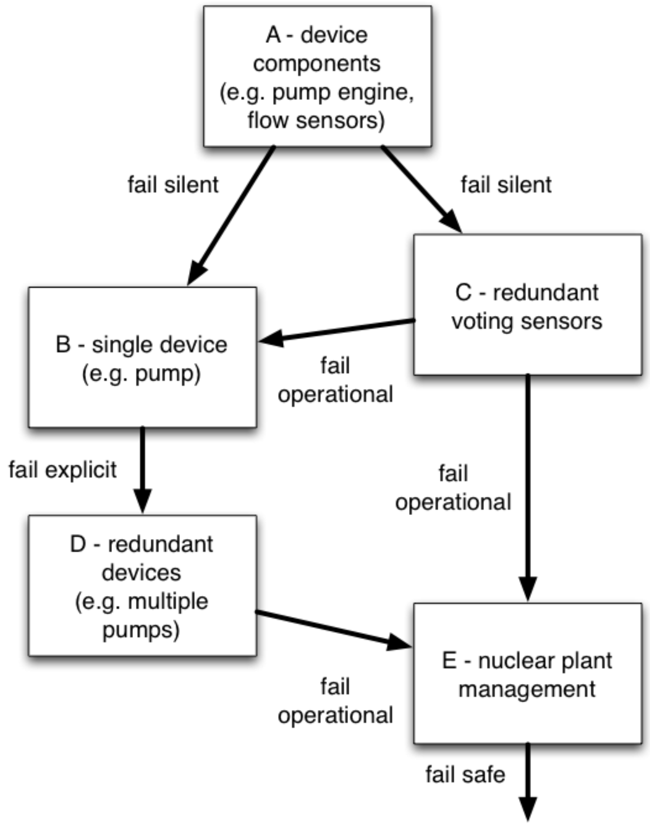
3.1 Overview

The system must maintain the water level in the steam boiler within a given interval. Failing to do so may result in the overheating of the nuclear plant. The basic control task is therefore given by feeding water into the boiler in such a way to maintain the proper level. In order to do that, pumps have to be switched on and off. When a pump fails starting, an alternate, backup, pump is used. A pump fails when the water does not circulate within five seconds after the start of the engine. the water level is read through specific sensors. Since one or more may fail, redundant sensors are employed, and the correct value is given by comparing the different instruments read. Operating the steam boiler plant means therefore having to take into account not only control strategies, but also fault management policies. It is clear that a single, monolithic, control program has to face unusual complexity issues, given by the concurrent modelling of both aspects, including the different operational modes which may result from the different failures that may occur. Adopting the right level of modularization helps therefore in partitioning the different tasks, operating modes and failure management policies. Figure 4 shows the mutual relationships among the different levels of complexity at which the different devices may be grouped; each level correspond to a specific holonic module and may be used as part of the more complex module linked to it by the arrow, labelled by the fault management behavior provided by each level. For example, the topmost one (labelled by A) gathers basic components. Such components fail silently and are used both in assembling simple devices (C) and redundant voting sensors (B). Redundant voting sensors and pools of redundant devices (D) exhibit finally a fail operational behavior towards the main control module (E).

3.2 Fail-Silent Sub-devices

This group gathers simple mechanical or hardware components, whose structure is not further specified and perform their function when they are turned on or off. Such components undergo random faults, which are faults which are not further analyzable and whose happening is predictable only in statistical terms. It includes synchronous subsystems assembled from hardware components, which can be seen as single units on their own. The elements of this group exhibit a behaviour that can be assumed to be fail silent, meaning that they either work or, in case of failure, stop responding for an unspecified period of time. Any other causal or malicious behaviour which can be subsumed under the general category of byzantine failures, may also be reduced to fail silent behaviours by well know techniques, such as the Voltan approach.

Figure 4. Hierarchical arrangement and mutual dependence of control levels and corresponding failure modes.

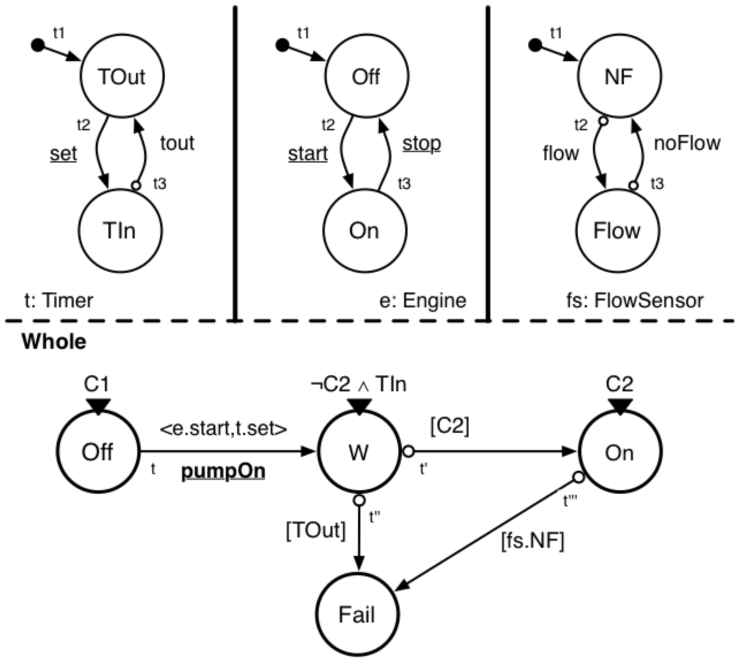

A - device components (e.g. pump engine, flow sensors)
fail silent
fail silent
C - redundant voting sensors
B - single device (e.g. pump)
fail operational
fail explicit
fail operational
D - redundant devices (e.g. multiple pumps)
E - nuclear plant management
fail operational
fail safe

3.3 Fail-Explicit Devices

By the term device we mean an object exposing an internal structure constituted by simpler parts which exhibit a fail silent behavior. By comparing the behavior of different components belonging to the previous level, it is possible to infer explicitly a fault of the assembly. In the steam boiler specification, pumps play a critical role. Each pump may be turned on or off. In moving from off to on the pump requires five seconds in order to balance the pressure of the steam boiler to which it injects fresh water: after this time the water should circulate from the pump to the steam boiler. According to such a real-life specification, a more general pattern of behavior may be inferred, constituted by a timer, by an actuator and by a sensor. In the pump, for example, the actuator is the engine of the pump and the sensor is the water-flow sensor. A pump capable of detecting its own failure is therefore, at the behavioral level, the assemblage of a timer, an engine and a flow sensor, as shown in the upper part of the PWS in Figure 5.
States *Off* and *On* are specified, respectively, by the state propositions $C1 = e.Off \wedge fs.NF$ and $C2 = e.On \wedge fs.Flow$, that is the behavior linking the two states is considered successful if engine e starts (i.e. it moves from the *Off* to the *On* state) and flow detector fs moves from the *NF* (no flow detected) to the *Flow* state. The overall state transition linking the two states employs a given amount of time in order to possibly achieve the final result; the system has therefore to wait for such a definite time interval. After such a time interval has expired without reaching

condition *C2* the pump reaches an explicit failure state. The PWS diagram in Figure 5 depicts a fragment of the whole joint behavior, namely the starting phase.
Starting from state *Off,* transition t, once triggered by event *pumpOn*, moves the system to a special waiting state *W* after having sent a start event to engine e and having started the timer t. The timer is designed for resting in the time-in state for a fixed amount of time, namely the five seconds required by the specification. The system rests in the *W* state until exactly one the two conditions occur: either state proposition *C2* becomes true or timer t moves to the timeout state Tout. In the former case the system starts successfully within the specified timing requirements and the system takes state transition t′ towards the success state *On*. In the latter case an explicit failure is detected and the system moves to the explicit failure state Fail through transition t″,. Such a state may be reached, at any time, through transition t‴, when the flow sensor denotes a lack of pressure moving to state *NF*.

Figure 5. The assembly of two low-level fail-silent components and of a timer allows to detect a failure either in the starting or in the working phase of a pump, through an explicit Fail state.

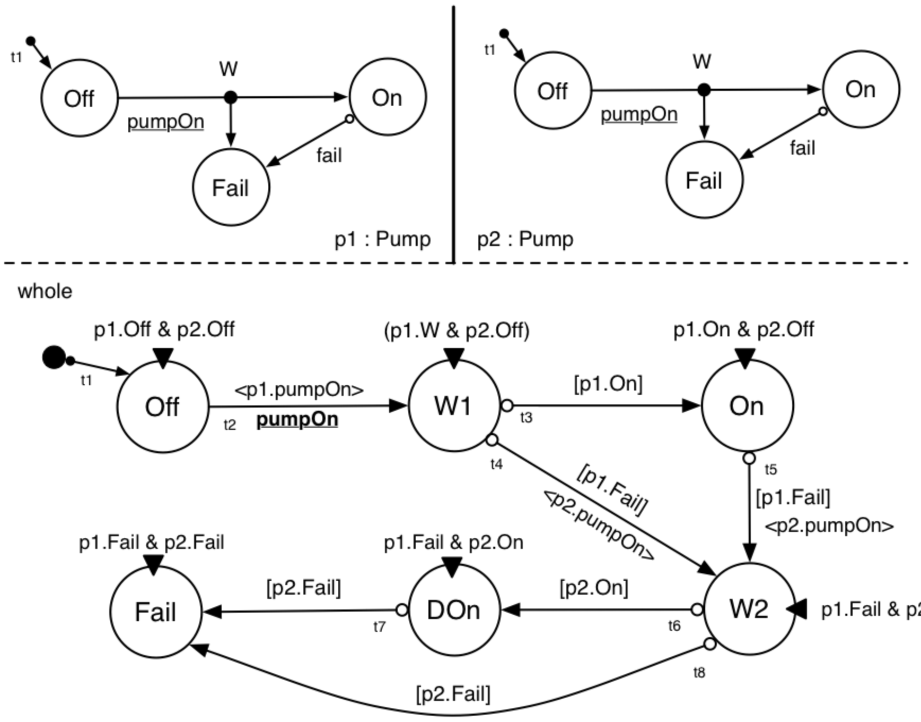


3.4 Redundant devices

Pumps are used to inject water into the steam boiler. Due to their critical function, there are four pumps in parallel which may be turned on in order to always have at least one working, replacing a faulty device by a backup one and implementing one of the most effective techniques in fault management. Unfortunately, dealing with different devices makes the approach very difficult to be managed effectively. In other words, the use of redundancy often makes problem complexity worse, since redundancy management has to be interleaved with the overall process control.
The proposed approach suggests instead dealing with redundant devices as components of a single system which guarantees the functionality of a single "virtual" pump. In other words, the implementation activates a single pump at the time, and in case of failure, switches to a backup one. At the logical level, a system of N redundant pumps may therefore be programmed and seen

as a single pump. The interface of the PWS therefore hides the swap amongst defective and working units, thus cutting the complexity of the control algorithm which deals with the pumps. Similar techniques may be also effective for the water level sensors, where a voting algorithm may be used instead. In both cases, the system has to signal that it is working in a degraded mode that is in a mode where one or more failures happened and we have to take the entire system towards a fail safe state.

Figure 6. Redundant fail explicit devices represented as a single device at the aggregate behavioural level.



In figure 6, redundant fail explicit devices can be represented as a single device at the aggregate behavioral level (due to space reasons we depict the case of only two replicated pumps). The interface distinguishes amongst a regular and a degraded working state (respectively *On* and *DOn*).

Figure 6 depicts the behavior of pump with a single backup pump: the same mechanisms may be extended to more replicated pumps. Observe that the interface taken from the behavior of the pump of Figure 5 is used twice in the upper, component part of the diagram. The system starts in the *Off* state, whose semantics is denoted by the state propositions which asserts that both pumps are off (*p1.Off ∧ p2.off*). When a *pumpOn* start event is received by the PWS, transition t2 is taken and waiting state *W1* is reached, in which pump p1 completes its waiting phase. After that, p1 moves either to state *On* or to state *Fail*. In the former case, the pump units complete the startup phase moving to state *On* through transition t3. In the latter case, pump p1 fails starting, and the system moves to the waiting state W2, where alternate pump p2 is started. State *W2* may be reached also from the working state state On after a failure in pump p1: in that case recovery pump p2 is also started by transition t5. Control moves out from state *W2* when either pump p2 starts successfully or when it fails starting. In the former case the control moves to state *DOn*, meaning that the system is working in a degraded state since one pump has already failed and a second failure would not be be mended by a third pump (in this example). In case pump p2 fails, state *Fail* is reached through transition t7. Observe that state *Fail* may be also reached through transition t8 in case pump p2 fails while working.

4. CONCLUSIONS

Since systems are formed by subsystems, that is by systems being part of more complex systems, it is customary to model, in the static case, such a containment relationship by a special modelling construct, called part-of relationship. Little or no attention has been paid, however, to the dynamical characterization of containment among systems. Understanding how the behavior of the parts influences the behavior of the whole, and vice versa, is in our opinion, of paramount importance, since most software systems are assembled from other software systems, each embodying its own behavior. More complex behavior host emergent properties, which do not pertain, however, to the component of the system exhibiting such a behavior. The paper shows that the holistic framework, as implemented by Part-Whole Statecharts, is a viable idea in order to encapsulate emergent properties of systems by a modular approach which allows to control complexity of artefacts, as well as to place increasingly complex behavior at different composition levels.

REFERENCES

Becker, S., Hasselbring, W., Boskovic, M., Dhama, A., Giesecke, S., Happe, J., Koziolek, H., Lipskoch, H., Meyer, R., Muhle, M., Paul, A., Ploski, J., Rohr, M., Swaminathan, M., Warns, T., Winteler D. (2006). Trustworthy software systems: a discussion of basic concepts and terminology. *SIGSOFT Software Engineering Notes,* 31(6), 1-18.

Carter, W. C., Avizienis, A, Laprie, J. C. (eds.). (1987). *The Evolution of fault-tolerant computing*. Wien, New York: Springer-Verlag.

Dominici, G., Palumbo, F. (2012). Decoding the Japanese Lean Production System according to a Viable Systems Perspective. *Systemic Practice and Action Research*, online first: doi: 10.1007/s11213-012-9242-z.

Dominici, G. (2012). The holonic approach for flexible production: a theoretical framework. *Elixir Journal*, 42, 6106-6110.

Dominici, G. Cuccia, L., Argoneto P., Renna, P. (2010). The Holonic Production System: a Multi Agent Approach. *iBusiness*, 2(3): 201-209. doi: 10.4236/ib.2010.23025.

Hammer, W. (1980). *Product Safety Management and Engineering*. Prentice-Hall.

Koestler, A. (1976). *The ghost in the machine*. London: Hutchinson.

Kopetz, H. (2011). *Real-time systems*. New York: Springer.

Leveson, N. G. (2011). *Engineering a Safer World: Systems Thinking Applied to Safety*. Cambridge, Ma: MIT Press.

Norman. D. A. (1993). *Things that Make us Smart*. Addison-Wesley.

Pazzi, L. (1997). Extending Statecharts for representing Parts and Wholes. *Proceedings of the EuroMicro-97 Conference*, Budapest, Hungary.

Pazzi, L. (1999). Implicit versus explicit characterization of complex entities and events. *Data & Knowledge Engineering* ,31, 115-134.

Pazzi, L. (2000). Part-Whole Statecharts for the Explicit Representation of Compound Behaviors. *Proceedings of the UML 2000 Conference*, York (UK): Springer. 1939, 541-555.

Pazzi, L., Pradelli, M. (2008). A State-Based Systemic View of Behavior for Safe Medical Computer Applications. *CBMS '08. 21st IEEE International Symposium on Computer-Based Medical Systems*, 2008, 108 -113.

Pazzi, L., Pradelli, M. (2010). Using Part-Whole Statecharts for the safe modeling of clinical guidelines. *Proceedings of the 2010 IEEE Workshop on Health Care Management* (WHCM), 1-6. doi: 10.1109/WHCM.2010.5441269.

Pazzi, L. (2012). Modularity and part-whole compositionality for computing the state semantics of statecharts. *Proceedings - International Conference on Application of Concurrency to System Design*, ACSD2012, 193-203.

Perrow, C. (1999). *Normal accidents : living with high-risk technologies*. Princeton, N.J.: Princeton University Press.

Smith, G. F. (1989). Representational effects on the solving of an unstructured decision problem. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-19, 1083-1090.

Storey, N. (1996). *Safety-critical computer systems*. Addison-Wesley.

Woods, D. D. (1995). Toward a theoretical base for representation design in the computer medium: Ecological perception and aiding human cognition. Flach, J. M., Hancock, P. A., Caird, K., Vicente K. J. (eds.). *An Ecological Approach to Human Machine Systems I: A Global Perspective*. N.J.: Erlbaum, Hillsdale.

[i] This work is a revised and extended version of the paper presented at The 2nd WCSA Conference, *Complexity Systemic Science and Global Energy Agenda*, Palermo, Italy, September 26 and 27, 2011.