Modeling Systemic Behavior by State-Based Holonic Modular Units

Luca Pazzi

University of Modena and Reggio Emilia
DIEF-UNIMORE
Via Vignolese 905, I-41125 Modena, Italy
`luca.pazzi@unimore.it`

Abstract. The paper explores a vision in modeling the behavior of complex systems by modular units hosting state machines arranged in part-whole hierarchies and communicating through event flows. Each modular unit plays at the same time the double role of part and whole, i.e. it is inspired by the philosophical idea of holon, providing both an interface and an implementation by which other component state machines may be controlled in order to achieve a global behavior. It is moreover observed that it is possible to assign a formal characterization to such state modules, due to their part-whole arrangement, since higher-level behaviors can derive formally their meaning from lower-level component behaviors. Such a way of arranging behavioral modules allows to establish directly correct-by-construction safety and liveness properties of state-based systems thus challenging the current approach by which state machines interact at the same level and have to be model-checked for ensuring correctness.

Keywords: state-based modeling, holons, component-based modeling, model checking, correctness by construction.

1 Introduction

Holons, in the terminology of Arthur Koestler in his 1967 book *The Ghost in the Machine* [1] 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 reductionist and the holistic view in systems analysis.

By the reductionistic view, which dates back to Descartes and is sometimes referred to as *divide and conquer* or more formally *analytic reduction*, a complex system can be analyzed by "reduction" into distinct parts so that they can be analysed separately. Such decomposition allows to deal effectively with systems complexity, by recursively confining it into less complex and distinct parts, namely *subsystems*. In order to be effective, analytic reduction implies the following assumptions: the division into parts will not distort the phenomenon being studied and the behavior of the components is the same when examined apart as when playing their part in the whole system. Additionally a third fundamental assumption is that it is possible to draw a clear boundary between the interactions among the subsystems and the behavior of the subsystems themselves [2][3].

While it is easy to discover and model standalone entities and systems, difficulty arises in assembling more complex systems using such entities as components, since there is no agreement on a composition model which allows full composability of abstractions. By full composability we mean that the *same exact* component should work in the whole without having to modify it in order to adapt it to any composition context (*off the shelf* approach), thus fully satisfying the second assumption of the reductionistic program reported above.

The current approach consists essentially of the composition model implicit in the object-oriented paradigm, by which systems modeled by objects interact and synchronize by invoking procedural methods on other objects, typically, albeit not only, by direct message exchange through object-valued attributes, called references. In other words, since there are no specific constructs for modeling the composition of objects as a whole, the object-oriented paradigm is inherently "component-oriented". As stated by Rumbaugh, it may be therefore observed that *"in the current object-oriented paradigm interactions are buried in the instance variables and methods of the classes, so that the overall structure of the system is not readily apparent"* [4]. A construct for modeling the overall structure of systems is missing in the object-oriented paradigm. Such a "missing" construct should be able to emphasize such a structure and to model its overall dynamics as a *whole*, thus correcting its tendency to be component-oriented.

Finally, such a construct should be moreover able to act as a standalone component into more complex wholes without further modifications. In other words, it is desirable, for elegance and simplicity, not only to have an additional modular construct for implementing wholes from components, but to have a single constructs playing both roles seaminglessy.

1.1 The "Missing Whole" Problem

The major difficulty in achieving effective object composition lies in missing the semantic distinction between *intra*- and *inter*-object behavior. While the former pertains naturally to the object itself, the latter acts as some sort of glue in the assembly of object components into more complex wholes.

Current object-oriented modeling techniques do not make any semantic distinction amongst the two kinds of behavior, since both are modeled by the same object construct, that is by mutual object references and remote method invocation through message passing. In this way, as observed, most of current modeling object oriented development methodologies and formalisms in analysis and implementation make the object construct semantically overloaded, since it hosts both its endogenous behavior as a component and, at the same time, the exogenous behavior of the system being assembled, resulting in the tight coupling of abstractions and implementations.

The "missing whole" problem has been moreover partially dealt with by means of different external mechanisms patched to the object-oriented paradigm in order to enhance it. for example, by specific object patterns and communication mechanisms, like the Mediator and the Observer design pattern [5] as well as the Model View Controller (MVC) mechanism [6]. The Mediator pattern decouples interacting classes by gathering their interacting portion of behavior within a single "mediator" class, thus improving the overall understandability and self-containment of the original classes. The role and the meaning of such "mediator" class has interesting interpretations beyond its immediate pragmatic one, which consists in laying a bridge among different behaviors. Such a bridging role is achieved by prescribing changes to other classes in reaction to other changes happening in the original classes. In other words such a bridging class hosts, as a matter of fact, a behavior on its own.

On the other hand, any systemic behavior can be seen abstractly as a reactive, coordinating behavior: it must in fact specify which actions have to be undertaken in reaction to specific changes in system components in order to prescribe additional changes to other system components. In other words a systemic behavior links different behaviors, and it can be modeled either by a specific modular construct of the language or it may be embedded within the original behaviors. The two approaches in modeling systemic behavior can be named respectively "explicit" versus "implicit" [7][8] depending on whether or not the system dynamics is hosted within a *single* modular unit of behavior. By the implicit approach an aggregate is modeled through a web of references by which the component objects refer one to another. This way the associative knowledge between the component objects is modeled directly (by object-valued attributes), hiding the structure and the behavior of the aggregate which is therefore not identified as a relevant object. By the explicit approach, an explicit additional object is inserted in the modeling instead, holding part-of relationships to the component objects. The two approaches bear consequences on software quality factors, for example implicit modeling tends to produce software artifacts which are tightly coupled and not self contained, thus producing software which is difficult to maintain, reuse, understand, and so on.

By adopting the explicit view, communication among systems has moreover to be revised accordingly, due to the presence of an additional centralized unit of behavior. In other words, components, which are no more tightly coupled one with another are now tightly coupled with the mediator class itself. The original tightly coupled modules have in fact to maintain static references to the class implementing the centralized behavior in order to notify them of changes in their internal status, allowing the mediator class to react appropriately through other static references. The Observer design pattern and the MVC mechanism may then be used in order to patch a reactive event-based communication framework to the object paradigm, with the aim of decoupling components classes from such specialized controller classes.

Object-oriented methodologies in the last two decades presented different approaches to the joint modeling of structural and behavioral aspects, essentially by encapsulating behavior around already discovered structures, with the result that not readily apparent systems and wholes were often missing in the final design. In other words, since components are self-evident while systems are often not, such methodologies provided very few support in discovering enclosing wholes, thus committing themselves towards the implicit modeling approach. A slightly different approach may be found instead in the Object Modeling Technique (OMT) by Rumbaugh [4] and in the Fusion method by Coleman et al. [9] which emphasized the role played by mutual relationships in discovering new encapsulating classes. Most of such work eventually merged into the UML [10] and the UML 2 [11] standards, which partly corrected the implicit tendency by a wealth of modeling constructs, for example by distinguishing between "weak" aggregation and "strong" composition relationships and introducing suitable "association" classes, albeit missing, in some sense, a unifying and comprehensive theoretical framework.

The aim of this paper is to show that a different paradigm may be pursued by going beyond the existing, partial, solutions towards a vision of the object paradigm which coherently combines components and wholes through a revised communication mechanism. The paper employs the Part-Whole Statecharts (PWS) formalism [12] in order to illustrate the more general idea of holonic modeling. Part-Whole Statecharts have already been used in pioneering the feasibility of holarchies of unmanned vehicles and of multiagent systems [13][14][15]. The same formalism has been endowed recently [16] with a formal syntax and semantics which allows, by construction, to build correct modules without using model checking techniques.

1.2 Structure of the Work

Section 2 discusses and compares general principles of behavioral composition, interaction and synchronization. Section 3 presents a modular construct which implements the general idea of holon and Section 4 discusses the feasibility of modeling real-world cases through Part-Whole Statecharts.

2 Behavioral-Driven Composition

It may be observed that entities which exhibit a peer to peer coordinated behavior act, globally, as a new aggregated entity. This is true both of entities communicating and coordinating through exchange of signals as well as of entities having a mechanical connection which trivially constrains them to behavioral coordination. Process algebras [17][18] furnish an interesting example in Hoare's CSP seminal work, where a customer CUST is modeled by a process which interact with a vending machine VM, the interaction becoming a new single process $\mathsf{P} = \mathsf{CUST}||\mathsf{VM}$, with $||$ being the concurrency operator in CSP. In the rest of the paper we will use, with no loss of generality with respect to process algebras, a state-based formalism which lends itself to a very readable, albeit formal, diagrammatic form through the concepts of states and state transitions.

Focusing directly on entities without considering their joint behavior, a practice inspired by common-sense real-world observation, may be misleading in finding higher level, more complex entities and systems. A system is in fact assembled from a set of physical components, which exercise physical control one upon another and exhibit individual state changes induced by mutual and direct physical interactions. The point is that local state changes can be seen, alternatively, as a global single state change at the system level, since an aggregate of coordinated entities moves from one global state to the another as its distinct components move from one state to the another.

The final step consists in hypothesizing that *since each entity hosts a behavior, each behavior implies the existence of a suitable entity which hosts it.* Looking at behavioral aggregation represents therefore a challenging clue in developing new modeling paradigms and constructs built according to the principle of explicit modeling of associative behavior.

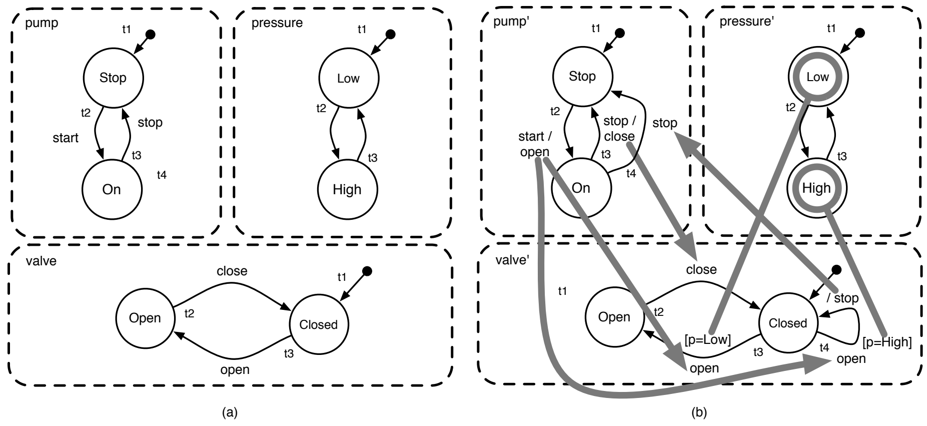

Fig. 1. The behavioral assemblage of three standalone Statecharts (a), into a complex system (b), obtained by modifying their behavior through direct event forwarding and mutual state condition testing. Grey arrows emphasizing mutual interactions and dependences visually depict the global systemic behavior.

2.1 The Explicit Modeling Conjecture

Statecharts, by David Harel [19], allow to model compound behaviors by a set of interacting parallel state machines, each state machine hosted within an AND-decomposed state, each single state of the machine being an XOR-decomposed state. Statecharts may then be used in a straightforward way to represent single behaviors. Such behaviors in turn may be composed into more complex ones through mutual coordination and synchronization, by forwarding command events from one machine to the another as well as by requiring state conditions to be satisfied before a transition is taken. As an example consider Figure 1 where three Statecharts automata taken in isolation in (a) form, globally, a system in (b) once automata are modified in order to mutually implement the following behavior:

> "The pump may be started and stopped, and the valve is opened and closed accordingly in order to allow the flow of liquid into tank. A sensor detects tank pressure and inhibits valve opening when the pressure is too high, in order to avoid reflux from the tank. When the opening of the valve is not permitted, a stop signal is sent back to the pump."

Behavioral interdependences which implement the behavior above are depicted in Figure 1 (b) by grey arrows: it may be easily observed that in such a form, the whole behavior is hardly understandable, modifiable, testable and not easily amenable to be checked for safety analysis. In other words, the Statecharts construct requires to modify the internal behavior of components, thus contravening Parnas' principle of information hiding [20]. It may be also observed that Statecharts' state-based modeling is subject to "the missing whole problem" of Section 1.1.

It may be conjectured, although a formal demonstration is outside of the scope of the paper, that it is always possible to obtain, through a single coordinating state machine W, called "whole", the same exact behavior that would be obtained by the direct interaction of a finite number of state machines. Let $\mathcal{A} = \{m_1, m_2, \ldots, m_N\}$ be a set of self contained state machine, and let $\mathcal{A}' = \{m'_1, m'_2, \ldots, m'_N\}$ be the corresponding set with m'_i being state machine $m_i \in \mathcal{A}$ extended in order to interact with other state machines in the same set by event forwarding and mutual condition testing. For example, Figure 1 depicts both the original self-contained machines (a) as well as the modified ones (b) (compare for example pump with pump′). Figure 2 shows a state machine W which implements the same behavior of Figure 1 (b) through the syntax and the semantics of the Part-Whole Statecharts formalism.

State machine W plays a coordination role towards the original interacting state machines, by labeling its state transitions with coordination commands directed to state machines belonging to the set $\mathcal{A}$ and reacting to state changes coming from the machines in the same set. Additionally, a definite semantics can be assigned to the states in W as shown in [16].

3 Holons

Holons are modular units which host a behavioral construct, in the case at hand a state machine, playing, at the same time, a twofold role:

1. the state machine coordinates the behavior of other component holons through their state machine interface;
2. the state machine provides an interface to other holons which coordinate the holon.

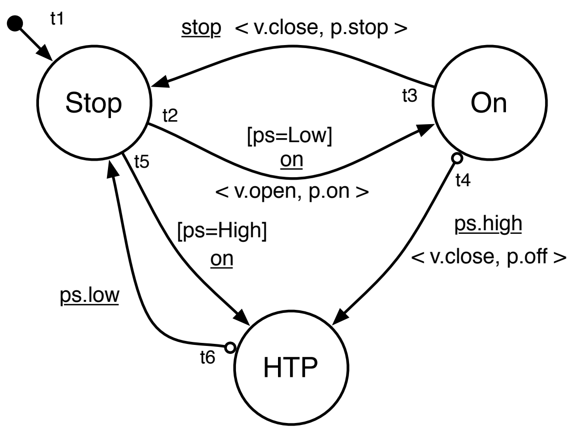

Fig. 2. The complex behavior of Figure 1 (b) modeled through a coordinating state machine, obtained as the "whole" section of an extend PW Statecharts machine (adapted from [21]). The automaton coordinates the behavior of the pump (p) and of the valve (v) depending on the pressure sensor (ps). State HTP denotes the exceptional case of "high tank-pressure". Transitions t_4 and t_6 are taken automatically as the tank pressure changes.

The two points match the general notion which stands behind the holonic paradigm (the so called "Janus", i.e., *double face*, paradigm). Holons are at the same time both whole (i.e., coordinating) and part (i.e., coordinated) entities. Holons are arranged in part-whole hierarchies called *holarchies* by the recursive pattern of composition depicted in Figure 3 (b). In such a pattern holon W coordinates the joint behavior of its component holons A, B, C.

The proposed holonic pattern of composition is *asymmetrical*, since wholes know their parts, but parts are forbidden to know the whole in order to maximize self containment and reusability.

Such an asymmetry is achieved by having two typologies of signals which travel from parts to whole and viceversa, as shown in Figure 3 (b):

1. the (whole section of the) holon has to *prescribe* coordinated behavior to each of its component holons: this is depicted by grey arrows in the picture;
2. the holon has to *react* to changes happening in its component parts: this is depicted by white arrows in the same picture.

Such a feature heavily relies on a suitable communication mechanism which implements both event delivery from the whole to a recipient (grey arrows) as well as a notification mechanism of any change happening within a component towards the whole (white arrows in the picture). Aim of the mechanism is, beyond carrying events, to decouple component holons from their coordinating counterparts, since the holon interface does not contain any reference to them. If, on one hand, such a holon implementation is tightly coupled to its component

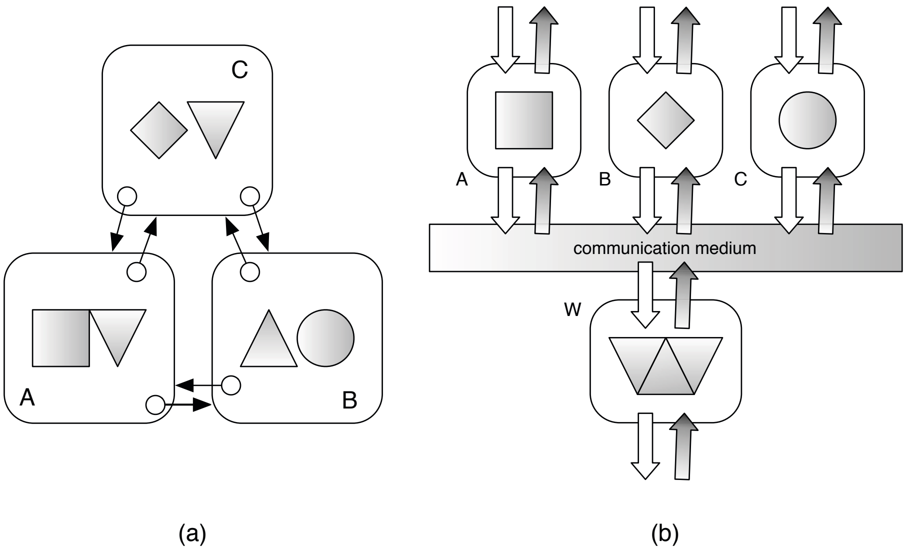

Fig. 3. The implicit modeling of three interacting entities through the reference construct (a) and the correspondent explicit modeling through four holons (b). The picture suggests that triangle-shaped associative behavior may be gathered within holon W thus freeing holons A, B and C from unnecessary details.

parts, holon interface does not contain any reference to any other holon in order to achieve loose coupling among them. Parts do not know the whole, since they have to be composed in many different, not foreseeable, contexts. The whole does know its parts, instead, in order to achieve a useful composition.

For example, state transitions in the Part-Whole Statecharts formalism (chosen for illustrating holons' features in the paper) contain event commands of the form $\langle \mathsf{c.e} \rangle$: once the transition is triggered, event e is delivered to component c whose interface contains a transition which has e as trigger (Figure 4 (a)). Vice versa, any change in a component holon, say d, is "notified" to the holon which has d as component, where a transition may have $\mathsf{d.f}$ as trigger, meaning that holon W has to react to event f from holon d (Figure 4 (b)).

4 Formal Specification and Semantics

It is possible to annotate state-based holons at design time in such a way that the behavior implemented at each level of composition can be formally specified and verified. Part-Whole Statecharts, as observed, already provide the formal instruments for performing formal specification of the semantics of state-based holons. This marks an evolutionary advantage with respect to traditional object-oriented and state-based modeling, where interacting and mutually referring modules have no semantics, as observed in the Statecharts case [22].

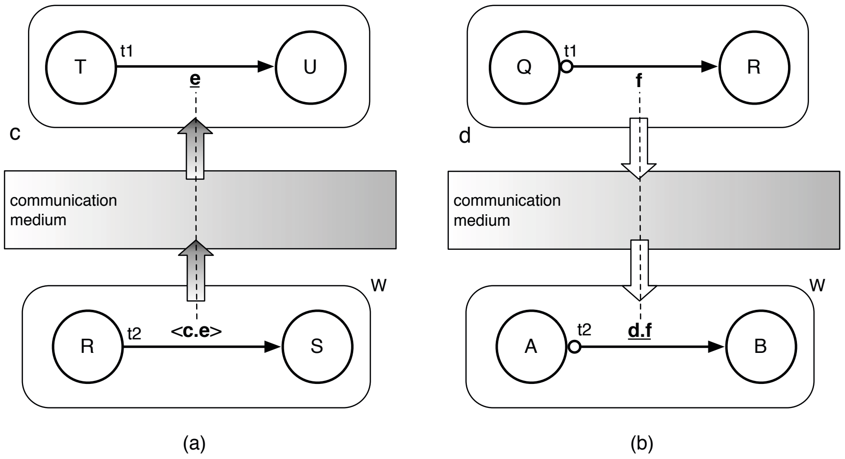

Fig. 4. The double role played by a communication medium among holons. (a) The "delivery" role, by which a command c.e is dispatched, by the occurrence of state transition t_2, from the whole automaton of holon W to component holon c, activating, through its interface, state transition t_1 triggered by e. (b) The reverse "notification" role, by which the occurrence of state transition t_1, labelled by f in the interface of component holon d, is notified to the whole section of holon W activating state transition t_2 triggered by d.f (Figure 4 (b)).

It is presumable that part-whole arrangement of modules in combination with hierarchical rules of control are at the basis of this important property of holons. Coordinating state-based holons, each referring to a finite and fixed number of component holons, allows in fact to map each of its states at a given composition level to a well defined configuration of states belonging to the next composition level. Such configurations can be equivalently expressed by a propositional formula in a suitable boolean algebra [16].

Correctness, and consequently safety and liveness, may thus be achieved by construction, by computing such propositions state by state as already shown in Part-Whole Statecharts. It can be hypothesized that checking correctness of each holon by construction may be also achieved by employing other formalisms for expressing holons dynamics, for example by procedural languages, since any state automaton may be deterministically translated into plain code. It may be finally observed that state based specifications and models may be directly executed.

4.1 Example

An automated car has to be controlled in order to start and stop depending on a traffic light on a track. The same car has a system of automated doors which can be operated either by a controller or manually. The car can be therefore seen as a holon having the automated doors and the engine as component holons

(Figure 5); at the same time the car holon can be composed into a higher level holon which has the car and the traffic light as components (Figure 6). Moreover safety and liveness constraints have to be met by the whole system and will be be enforced, compositionally, at different levels of the holarchy:

1. The doors have to be closed while the car is moving;
2. The car has to be stopped when the traffic light is red, restarted when green.

Doors holon module: The doors may be opened and closed either by a signal from the car controller or manually. In both cases the module moves from the Open to the Closed state and vice versa. In the former case, the doors system works as an actuator, by receiving event signals open and close which trigger state transitions t_3 and t_4. In case doors are opened and closed manually, the same holon may be seen as a sensor, since it takes autonomously state transitions t_2 and t_5 and emits event signals open and close towards the holon which has the doors as components, namely Car in the example. Autonomous state transitions such as t_2 and t_5 are denoted by a small white circle near the starting state.

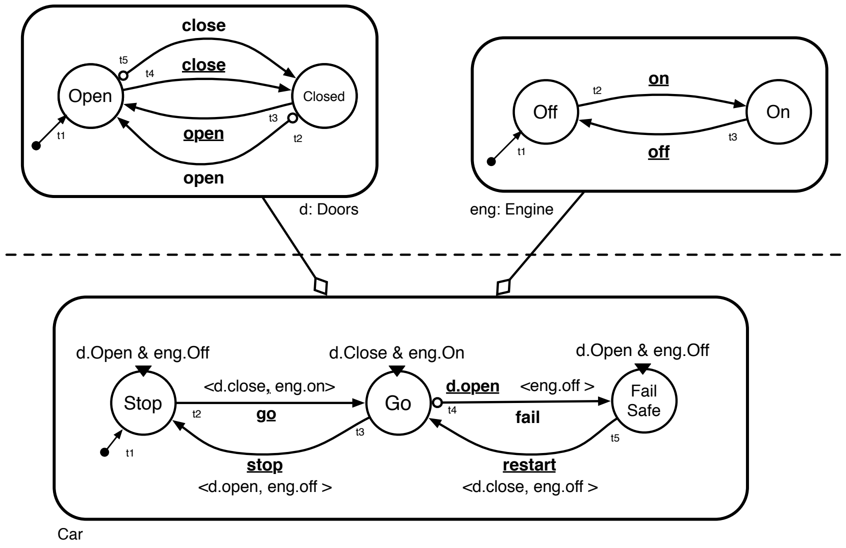

Fig. 5. A holarchy controlling a safe automated car on a track

Engine holon module: The car engine may be simply turned on and off by receiving event signals on and off which trigger state transitions t_2 and t_3 and move the holon into the corresponding states On and Off.

Car holon module: this holon has two regular working states, Stop and Go. An additional FailSafe state takes into account the exceptional behavior resulting from the manual opening of the doors while the car is moving. As in PW Statecharts, states are annotated by state propositions which are guaranteed to hold when the system is within that state. For example, d.Open & eng.Off associated to state Stop means that when in such a state the doors have to be opened and the engine must be turned off. State proposition d.Close & eng.On associated to state Go means conversely that when the car is moving the doors have to be closed and the engine turned on. Transitions t_2 and t_3 are externally triggerable by events go and stop which will be part of the interface of the holon (see holon Car in the context of the holarchy in Figure 6). Transitions t_2 and t_3, once triggered, propagate respectively command events $\langle \mathsf{d.close}, \mathsf{eng.on} \rangle$ and $\langle \mathsf{d.open}, \mathsf{eng.off} \rangle$ towards component holons Doors and Engine. It can be easily verified that both transitions agree with the state propositions of the starting and arrival states. Finally, when in the Go state, the manual opening of the doors causes event open belonging to transition t_2 of holon Doors to be sent towards holon Car. This in turn triggers transition t_4, which sends an off command event $\langle \mathsf{eng.off} \rangle$ to the Engine holon which then moves to state FailSafe. Since transition t_4 is triggered by an event coming from a component, the resulting transition t_4 will be seen by external contexts as autonomously triggered (see the interface of Car in Figure 6). The system may be restarted by sending a restart event from external composition contexts (such as holon GlobalTrackMonitor of Figure 6) which triggers transition t_5 which in turns closes the doors and restarts the engine by commands $\langle \mathsf{d.close}, \mathsf{eng.on} \rangle$.

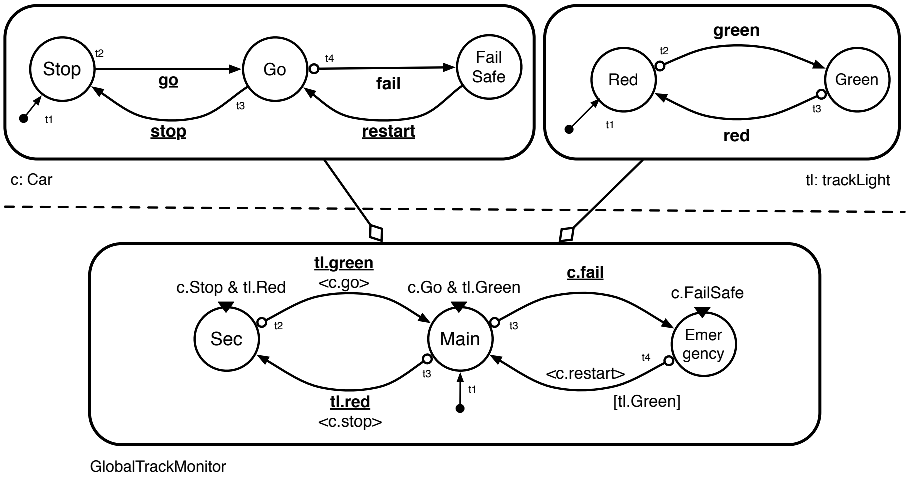

Fig. 6. The holarchy coordinating the automated car of Figure 5 and a traffic light which enables either a main (Main) or a crossing secondary (Sec) road. Holon Car is now employed in the model through its interface, obtained by simply hiding its components' holons as well as any reference to them from the implementation of Figure 5.

Once holon Car has been designed and verified against the first of the two safety constraints listed above, its interface may be employed in higher level composition contexts without any concern for safety. In other words, any of the go-, stop- and restart-triggered state transitions may asked to be taken in holon Car without having doors opened while the car is moving. We then model a global monitor for the track system, which has in charge both the automated car and a traffic light in order to stop it when a secondary road is enabled (Figure 6). Since it employs the holon Car which has been already verified for safety, it now suffices to employ it for implementing the final behavior checking only for the second safety constraint.

Traffic Light holon module: the state machine is the interface of a sensor which detects the current color of the lights from the main road, by changing state (Red and Green) by taking the two autonomous transitions t_2 and t_3. The traffic light device enables and prevents access to the secondary road accordingly.

Global Track Monitor holon module: as red and green events are emitted by the Traffic Light, autonomous transitions t_2 and t_3 are triggered, which in turn send actions go and stop to the car ($\langle$c.go$\rangle$ and $\langle$c.stop$\rangle$). It may be easily verified that state propositions c.Stop & tl.Red c.Go & tl.Green, associated respectively to states Sec and Main are trivially verified by such state transitions. When in state Main the event unsafe signaling that a a door has been opened while the car is moving, brings the system to state Emergency where additional actions can be taken (not shown in the example): the car is restarted by transition t_4 through action $\langle$c.restart$\rangle$ as soon as such additional actions are completed and the traffic light is green.

4.2 Application to Incremental Modeling of Safety Constraints

The holonic approach allows to partition safety tasks and to model them into hierarchically arranged modules, which can be checked incrementally by visiting a single finite state diagram in constant time instead of having to unfold all feasible behaviors of a set of many interacting machines, as in current model checking techniques, which leads to exponential visiting time. Once designed and checked for safety, the module can be used "as is" in further composition contexts. In general a *safe* holon module can be arranged from already-designed *safe* modules by specifying that their interaction will occur in a *safe* manner, that is, as observed, by checking a single state machine in constant time. In case of physical systems which inherently fail, a suitable fault management strategy can be hosted at each level of decomposition, provided a sound decomposition has been carried out in the entire design phase. It becomes thus possible to defeat the overall complexity given by the concurrent modeling of operating modes and failure management policies. For example, fail silently sub-devices may be used as components for assembling a device behavior, which is able, at the higher level to *reduce* the fail silent behavior to a more tractable fail explicit behavior. The latter, in turn, may be used, at the next composition level, to obtain a fail safe or fail operational behavior. Examples of such hierarchical arrangement of failure modes and related devices are given in [23][24][25] and are summarized in Figure 7.

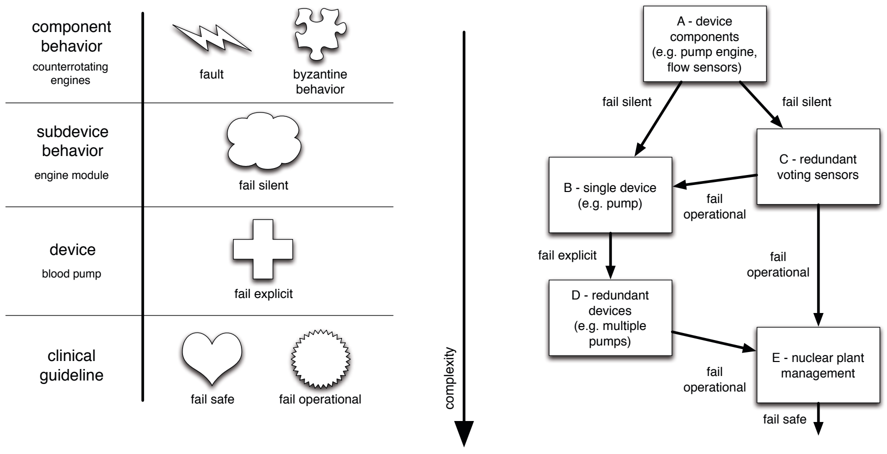

Fig. 7. Fault management strategies and devices tend to be placed at different levels of complexity in hierarchically decomposed behavior(adapted from [24])

5 Conclusions and Further Research

The paper main thesis is that an unifying paradigm may be founded upon different empirical and theoretical evidences, among which already existing improvements to the current OO paradigm.

Interacting modules synchronize system behavior by message exchange. Such messages, however, denote different kinds of information. Typically, systems communicate either by "peer to peer" or "part to whole" message exchange, the latter case pertaining to systems composed of other systems. The problem consists, at the ontological level, in determining whether two systems stand in the former or in the latter relationship. The object oriented paradigm, for example, do not distinguish amongst the two cases, thus giving rise to the "semantic overloading" of the reference mechanism.

As observed, vertical, part-whole, system composition is asymmetrical in nature and preserves model reusability. On the other hand, horizontal, peer to peer message exchange hinders model reusability, since it forces system modelers to introduce exogenous details within systems being modeled, bringing severe limitations to the overall software quality of the modeled systems. Physical interactions among physical systems denote in fact conceptual structures, not evident at first sight, which are well suited in order to host the overall interaction and synchronization knowledge among the component parts. By introducing additional system entities with the aim of lodging such a 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 the notion of holon [26][27] which may give, in turn, further impulse to use of holarchies in distributed agent-based manufacturing systems[28][29][30][31].

The paper presents an explicit construct for the recursive modeling of systems. The approach forces the modeler to express the behavior of composition by a single unit of behavior. Such a behavioral unit plays the double role of being both a specification of the behavior of the system of interacting parts, as well as an interface for further composition of the entire assembled system as whole. This double side, "Janus"-like feature makes such kind of construct suitable for modeling, as observed, the behavior of holons.

The presented approach may be finally used in order to partition safety tasks into hierarchically arranged modules, each checked incrementally. Real-time critical systems, for example, may benefit from the approach since it allows to decompose a single, monolithic, control program into smaller, safe, reusable and composable systems, each hosting a different safety policy.

5.1 Further Research

Peer-to-peer (P2P) direct modeling is of paramount importance in expressing mutual interaction among entities and systems. Peer-to-peer interacting entities may be seen in many cases as playing specific roles within an implicit whole/holon. For example, "husband" and "wife" are both entities playing the respective roles in a P2P cooperation. Peer-to-peer modeling however does not allow to compute an exact state semantics, while part-to-whole (P2W) modeling instead does, as suggested by the paper. It seems evident that any peer-to-peer modeling corresponds to a specific part-to-whole modeling. For example, wife and husband are both part of a "family", which may be represented by an explicit entity/holon having two "humans" as components playing the "husband" and "wife" roles as components of the family holon.

As observed in the paper, it may be conjectured that it is always possible to obtain through the holonic P2W approach the same exact behavior that would be obtained by the direct P2P interaction of a finite number of state machines. It would then be interesting to investigate further whether an equivalence theorem between P2P and P2W modeling could be show to hold. Any P2P cooperation could then be checked by transforming it in a P2W holonic model for the sake of verification, and back for the sake of readability.

Another point which is worth further investigating and is not covered in the paper deals with inheritance. It is worth noting that the more we model state constraints within single modules the more we restrict the resulting global cartesian automaton [32]. A starting point towards a novel notion of holonic inheritance should therefore take into account, among possible other aspects, adding or removing behavioral restrictions to state machines in moving along inheritance hierarchies.

Finally, more research is needed in order to move towards more complex composition structures albeit retaining the part-whole hierarchical arrangement. For example, what if the same component is part of two different holons? Figure 8-(a) shows two different cross road controllers (CrossRoad1 and CrossRoad2) which share traffic light main1:TLight1. In order to avoid race conditions, the idea is to restrict control of main1 to a single holon (CrossRoad1 in the picture) through triggerable transitions t_2 and t_3 by command events go and stop (b). The rule, to be further investigated, is that transitions controlled by a given holon become simply observable by other holons in the composition graph. In this way, holon CrossRoad2 would acquire the same traffic light with transitions t_2 and t_3 seen as autonomous and non controllable (c), i.e. by simply "sensing" its state changes and taking decisions accordingly. Holarchies, by such a perspective, may be thus thought as acyclic direct graphs instead of simple partonomic trees, thus gaining more flexibility in modeling complex scenarios.

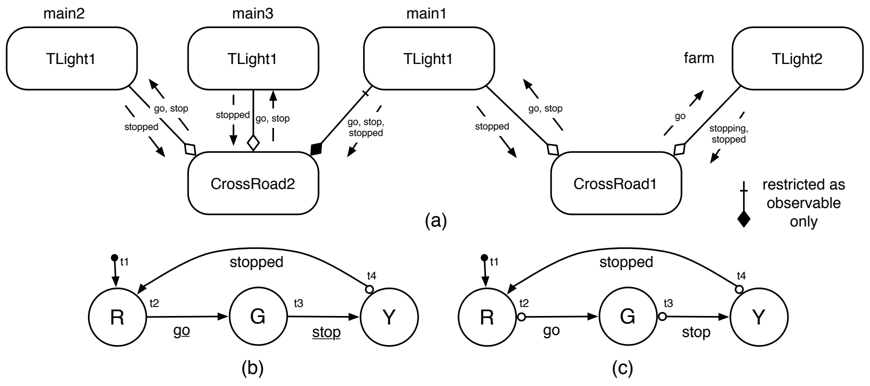

Fig. 8. (a) The same traffic light holon main1:TLight1 may be shared by two crossroad controller holons, with the restriction that at most one is allowed to trigger its transitions in order to avoid race conditions. (b) The interface of main1 as seen by CrossRoad1 with transitions t_2 and t_3 triggerable by events go and stop. (c) The interface of main1 as seen by CrossRoad2 with autonomous transitions t_2 and t_3 emitting events go and stop.

References

1. Koestler, A.: The ghost in the machine. Hutchinson, London (1976)
2. Leveson, N.: Engineering a Safer World: Systems Thinking Applied to Safety. Engineering Systems. MIT Press (2011)
3. Weinberg, G.M.: An introduction to general systems thinking. Behavioral Science 21(4), 289–290 (1976)
4. Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorensen, W.: Object-Oriented Modeling and Design. Prentice Hall (1991)
5. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design patterns: elements of reusable object-oriented software. Addison-Wesley Longman Publishing Co., Inc., Boston (1995)
6. Krasner, G.E., Pope, S.T.: A cookbook for using the model-view controller user interface paradigm in smalltalk-80. J. Object Oriented Program. 1(3), 26–49 (1988)

7. Pazzi, L.: Implicit versus explicit characterization of complex entities and events. Data & Knowledge Engineering 31, 115–134 (1999)
8. Wand, Y., Storey, V.C., Weber, R.: An ontological analysis of the relationship construct in conceptual modeling. ACM Trans. Database Syst. 24, 494–528 (1999)
9. Coleman, D., Arnold, P., Bodoff, S., Dollin, C., Gilchrist, H., Hayes, F., Jeremaes, P.: Object-Oriented Development: The Fusion Method. Prentice-Hall, Inc., Upper Saddle River (1994)
10. Rumbaugh, J., Jacobson, I., Booch, G.: The Unified Modeling Language Reference Manual. Addison-Wesley (1998)
11. Rumbaugh, J., Jacobson, I., Booch, G.: Unified Modeling Language Reference Manual, The 2nd edn. Pearson Higher Education (2004)
12. Pazzi, L.: Extending statecharts for representing parts and wholes. In: Proceedings of the EuroMicro 1997 Conference, Budapest, Hungary (1997)
13. Bou-Saba, C., Esterline, A., Homaifar, A., Rodgers, D.: Formal, holarchical representation of tactical behaviors. In: 2005 IEEE International Conference on Systems, Man and Cybernetics, vol. 4, pp. 3828–3834 (October 2005)
14. Bou-Saba, C., Esterline, A., Homaifar, A., Rodgers, D.: Learning coordinated behavior: Xcss and statecharts. In: 2005 IEEE International Conference on Systems, Man and Cybernetics, vol. 1, pp. 436–442 (October 2005)
15. Esterline, A.C., BouSaba, C., Pioro, B., Homaifar, A.: Hierarchies, holons, and agent coordination. In: Hinchey, M.G., Rago, P., Rash, J.L., Rouff, C.A., Sterritt, R., Truszkowski, W. (eds.) WRAC 2005. LNCS (LNAI), vol. 3825, pp. 210–221. Springer, Heidelberg (2006)
16. Pazzi, L., Pradelli, M.: Modularity and part-whole compositionality for computing the state semantics of statecharts. In: 2012 12th International Conference on Application of Concurrency to System Design (ACSD), pp. 193–203 (June 2012)
17. Milner, R.: A Calculus of Communicating Systems. LNCS, vol. 92. Springer (1979)
18. Hoare, C.: Communicating Sequential Processes. Prentice-Hall (1985)
19. Harel, D.: Statecharts: A visual formalism for complex systems. Science of Computer Programming 8, 231–274 (1987)
20. Parnas, D., Clements, P., Weiss, D.: The modular structure of complex systems. IEEE Transactions on Software Engineering SE-11(3), 259–266 (1985)
21. Pazzi, L., Pradelli, M.: A state-based systemic view of behavior for safe medical computer applications. In: 21st IEEE International Symposium on Computer-Based Medical Systems, CBMS 2008, pp. 108–113 (2008)
22. Von der Beek, M.: A comparison of statecharts variant. In: Langmaack, H., de Roever, W.-P., Vytopil, J. (eds.) FTRTFT 1994 and ProCoS 1994. LNCS, vol. 863, pp. 128–148. Springer, Heidelberg (1994)
23. Pazzi, L., Pradelli, M.: Part-whole hierarchical modularization of fault-tolerant and goal-based autonomic systems. In: 2nd IFAC Symposium on Dependable Control of Discrete Systems, DCDS 2009, pp. 175–180 (2009)
24. Pazzi, L., Pradelli, M.: Using part-whole statecharts for the safe modeling of clinical guidelines. In: 2010 IEEE Workshop on Health Care Management, WHCM (2010)
25. Pazzi, L., Interlandi, M., Pradelli, M.: Automatic fault behavior detection and modeling by a state-based specification method. In: 2010 IEEE 12th International Symposium on High-Assurance Systems Engineering (HASE), pp. 166–167 (November 2010)
26. Hilaire, V., Koukam, A., Rodriguez, S.: An adaptative agent architecture for holonic multi-agent systems. ACM Trans. Auton. Adapt. Syst. 3(1), 2:1–2:24 (2008)
27. Mella, P.: The Holonic Revolution. Holons, Holarchies and Holonic Networks. The Ghost in the Production Machine. Pavia University Press (2009)

28. van Leeuwen, E., Norrie, D.: Holons and holarchies (intelligent manufacturing systems). Manufacturing Engineer 76(2), 86–88 (1997)
29. Sabaz, D., Gruver, W., Smith, M.: Distributed systems with agents and holons. In: 2004 IEEE International Conference on Systems, Man and Cybernetics, vol. 2, pp. 1958–1963 (October 2004)
30. Giret, A., Botti, V.: Identifying and specifying holons in manufacturing systems. In: 7th World Congress on Intelligent Control and Automation, WCICA 2008, pp. 404–409 (June 2008)
31. Dominici, G., Palumbo, F.: Decoding the japanese lean production system according to a viable systems perspective. Systemic Practice and Action Research 26(2), 153–171 (2013)
32. Arnold, A.: Finite Transition Systems: Semantics of Communicating Systems. Prentice-Hall International Series in Computer Science. Masson/Prentice Hall (1994)