Using Part-Whole Statecharts for the Safe Modeling of Clinical Guidelines

Luca Pazzi
Department of Information Engineering
University of Modena and Reggio Emilia
Via Vignolese 905, I-41100 Modena, Italy
Email: luca.pazzi@unimore.it

Marco Pradelli
Department of Information Engineering
University of Modena and Reggio Emilia
Via Vignolese 905, I-41100 Modena, Italy
Email: marco.pradelli@unimore.it

***Abstract*—Behavioral aspects of medical guidelines can be modeled and formalized in a straightforward way by flow diagrams. However, safety plays a critical role in both modeling and formalization given the intermingled involvement of human actors and medical devices which have to interact and coordinate according to precise rules and strict timelines. Although state based formalisms can be shown to be very apt in depicting complex situations in both an intuitive and formal manner, they do not provide modular constructs for defeating complexity and require model checking in order to be verified against safety requirements. The paper proposes to adopt a modular and hierarchical state based formalism for the sake of representing behavioral aspects in medical guidelines. Such a formalism can be shown to provide a natural arrangement of different fault management strategies at the different decomposition levels.**

I. Introduction

Medical guidelines have been introduced with the aim of providing physicians with decision and treatment support in providing appropriate health care, typically within the context of evidence-based medicine [1]. Although guidelines often originate in a textual, narrative form, the implicit nature of medical care, consisting in taking decisions among alternatives and issuing actions and treatments towards patients, leads to a very natural interpretation of guidelines as flow diagrams. The aim of issuing timely notifications to the different human actors involved as well as keeping information within patients records up to date led, in turn, to the attempt of making guidelines directly executable by a computer engine through a specific syntax [2].

It becomes then evident that the task of translating guidelines in a computer interpretable form bears a strong intersection with the more general problem of modeling and formalization of complex medical situations. Such situations involve the mixed interaction of human operators, patients and computer controlled medical devices. Such interacting asynchronous actors have to coordinate and interact according to strict timing deadlines. This in turn requires suitable tools and primitives in order to control complex tasks without leading to harm or injury to the human beings involved. Safety is therefore a challenging and inescapable issue in the modeling of clinical medical guidelines.

Many safety-related aspects are not evident at first glance and have to be carefully identified and taken into account.

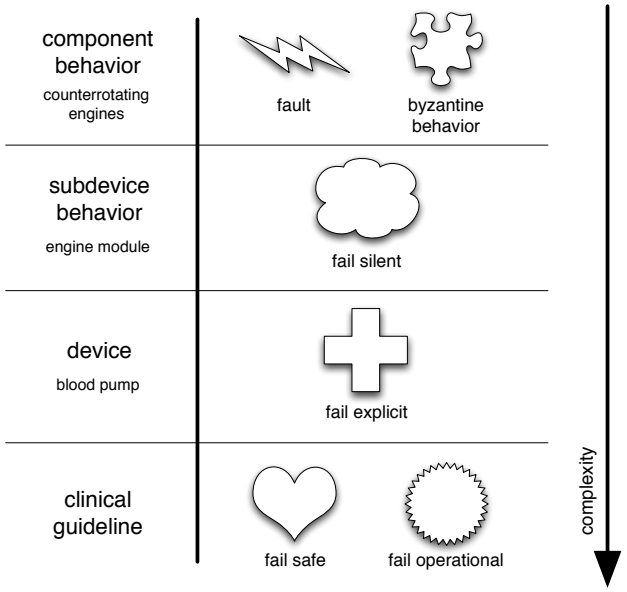

Fig. 1. Hierarchical part-whole decomposition of a complex medical domain allows to implement fault detection and recovery policies at different decomposition levels.

Safety depends in first place on the timely execution of actions towards patients, but also on the correct synchronization of the clinical agents involved in carrying out the different tasks required by the guideline. Additionally, late or incorrect interpretation of clinical data coming from monitoring devices may jeopardize patient's safety as well. Nonetheless, flowchart-like guideline models may lead to deadlocks or to non terminating loops: medical teams adopting them may therefore be found in critical situations in which contradictory, inconsistent or cyclical actions are issued by the flowchart algorithm to the team members and to the medical devices involved.

Vice versa, wrong or exceptional actions and signals coming from both human and computer controlled agents have to be taken into account in order to undertake alternate recovery plans: in other words, exceptional events have to be exhaustively foreseen and considered by the flowchart algorithm in order to issue recovery actions accordingly.

Aim of this paper is to show that the specification and verification of safety aspects in complex medical guidelines may be carried out incrementally, starting from hardware components and ending into more complex coordination modules, each corresponding to some physical or logical entity in the domain. The paper shows, moreover, that such a decomposition is feasible and that each decomposition level may take into account, in a parallel way, safety and fault management aspects, ending in *fail safe* or *fail operational* [3] clinical guidelines involving both medical devices and human operators, as suggested by Figure 1. Finally, the adopted formalism may be used homogeneously from low level, intra-device specification, to high level, complex medical contexts.

The paper is structured as follows: in Section II open issues in the safe modeling of clinical guidelines are examined. In Section III a proposal for the modular and incremental modeling of behavior at different specification and safety levels is formulated and presented through a running example.

II. Open Issues

A. Safety issues

A novel view of safety as a systemic concept is recently emerging [4]; in first place, according to such a view, safety must not be confused with reliability. A reliable medical device may in fact harm once used incorrectly, for example once wrong working parameters are set by human operators or by having it to operate on the wrong patient at the wrong time.

In general a system is safe once (1) it is assembled from safe/reliable components and (2) its components interact in a safe manner. In the same way, a safe system, once assembled, can be used on its turn in order to assemble higher level safe systems. The safety of a single system depends therefore not only on the reliability of the single components making it, but also on the mutual interactions among them. In [5] it is shown, for example, that a naive assemblage of medical devices consisting of a blood pump, a valve and a patient's pressure monitor is harmful under specific circumstances.

B. Modularity issues

Safe modeling of complex medical domains requires suitable conceptual tools for controlling their inherent complexity, mainly through decompositon. Once a system of interacting entities is decomposed into less complex, manageable systems, the resulting behavior can be checked against propositional safety requirements.

The current model of control and communication, typical of Harel's Statecharts [6], requires the component behaviors to embed references to the other processes: by such an approach, the planned global behavior may be obtained by simply allowing any component entity behavior to read an modify the status of any others.

As a consequence, global behavior tends at being represented in a fragmented form, by tightly coupled modules. Figure 3 shows the mutual references, depicted by gray arrows, among different behavioral processes involved in the example presented in Section III, where each parallel process is hosted within a Statecharts' parallel section.

The main problem with such an approach to process aggregation is that the overall behavior of the system is only implicitly defined, that is it remains concealed to further analysis and use. A system behavior obtained through such a model is in fact difficult to understand, since it is difficult for the designer to have a complete view of the whole system behavior scattered into the different component sections. In addition, the system behavior may potentially either deadlock or not terminate, due to cross referencing mutual conditions as well as to infinite, circular successions of state transitions and command broadcasting. It may be finally observed that component processes are themselves barely reusable and analysable, since interprocess behavioral references make the parallel sections tightly bound one to the another.

Such a current way to achieve control and communication among processes can be referred to as the *implicit model* of process aggregation.

In order to verify that system components interact in a safe way, that is that a safe global behavior is obtained, model checking techniques [7] have to be employed in the implicit case defined above. In first place logical temporal safety formulae have to be specified in order to prescribe a correct, i.e. safe, logical behavior, then all the feasible interactions among system components have to be explored. Each local interaction gives rise to an implicit global state of the system which must, in turn, comply with such formulae. Specifying and checking formal properties through model checking techniques becomes however ineffective as the number of involved device states increases, since the size of the tree depicting all possible behaviors grows exponentially.

C. Language issues

A natural cost effective solution for implementing clinical guidelines consists in adopting UML [8] as the modeling language. Many studies pointed in that directions [9][10].

Both UML state diagrams (adapted from Harel Statecharts [6]) and more general flow diagrams (called activity diagrams) have been incorporated within the UML with the aim of describing behavioral aspects of the domain being modelled. The distinction amongst the two concepts has not alway been very clear. Typically, Statecharts diagrams describe the behavior of single entities, while activity diagrams are used to model the overall coordination amongst interacting entities. Starting from the first release of UML, activity modeling has been separated from Statecharts modeling, thus creating a unnecessary representational dichotomy between intraobject and extraobject behavior, although an activity graph was primarily intended as a special form of state machine [8]. Such a distinction has been made even sharper in the latest revision of the UML language where state machines and activity diagrams are no longer correlated.

Adopting UML for the purpose of modeling behavioral aspects of clinical guidelines means losing expressive power in the modeling of safety critical domains mainly due to the

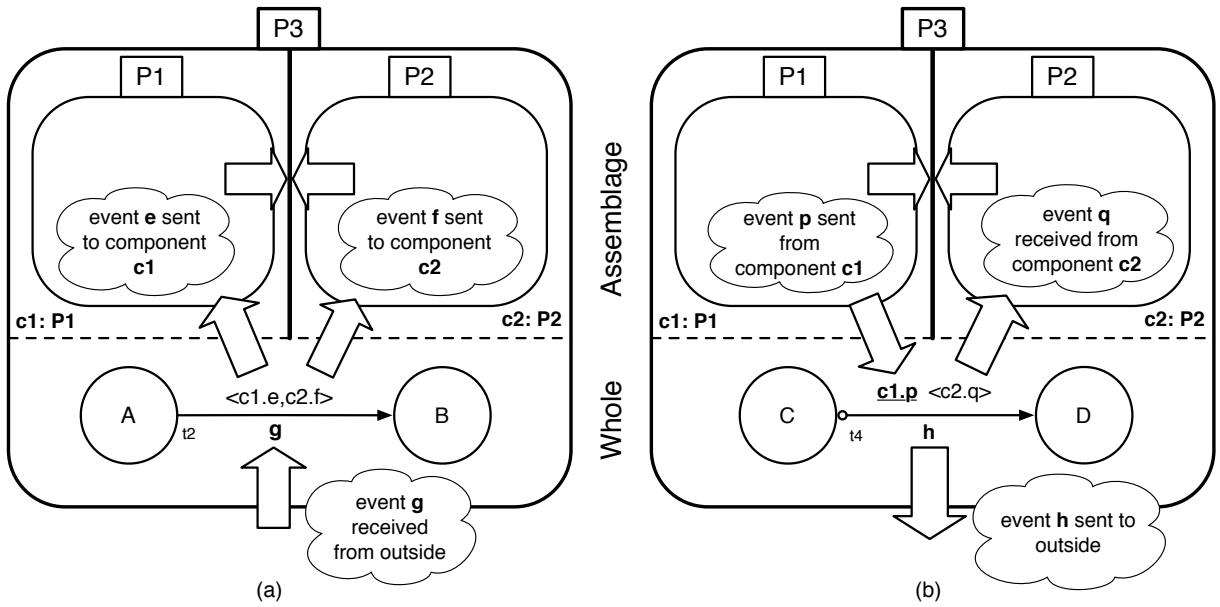

Fig. 2. Event flow within Part-Whole Statecharts. In (a) event **g** triggers transition t_2 in PWS P3 which forwards events **e** and **f** towards component PWSs P1 and P2. In (b) event **p** from component PWS P1 triggers transition t_4 in PWS P3 which forwards events **q** towards component PWS P2 and event **h** outside PWS P3. In both cases events are not allowed to travel amongst component PWSs.

above discussed missing capability of UML in providing a homogenous notion of intra- and interobject state and behavior.

A complex behavioral process may in fact be seen, informally, as the *sum* of the single behaviors as well as of the single states of the entities participating in it.

State is indeed a common representation primitive in most of eleven guideline models surveyed in [2], meaning either the patients' clinical status or the ongoing status of the whole clinical process.

A unifying view of behavior under a state-based formalisms would therefore improve the overall modeling process of medical guidelines. States provide a well definite, although informal, snapshot of a situation; such a situation can be formally refined in later stages of software development. States allow homogeneity in representation by allowing to characterize both the status of activities inherent clinical processes — for example completed, ready, stopped, failed, and so on — as well as that of humans beings, such as patients, physicians and other technical operators — for example sick, healthy, ready, panicked. Finally the status of medical embedded system devices also can be described effectively by states such as on, off, pumping, refilling, et cetera.

Another benefit in adopting state diagrams is that they substitute flow diagrams in a natural way. States depicting situations act as decision points, since the world can evolve towards different situations as different events happens or different conditions are met. State transitons lead to such situations according to guard conditions, thus implementing naturally a control flow. Situations, once formalized into states, can be finally checked against safety formulae.

III. Proposal

The global behavior of interacting entities can be represented explicitly by Part-Whole Statecharts [11][12] (shortened either as PW Statecharts or PWS) created with a radical commitment towards state-based modularity. The idea behind their introduction was to have a formalism which could encapsulate a compound behavior, forcing the modeler to expose an interface representing the composed behavior as a whole. The formalism has been recently complemented by a constraint based specification method [13]. 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 to make *explicit* the representation of the interaction among the behavior of parallel state based processes, contrasting to the *implicit* way of achieving process aggregation discussed in Section II-B.

A Part-Whole Statechart consists of two main sections (Figure 2), one hosting a set of component state machines, referred to collectively as the *assemblage*, the another 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).

Part-Whole Statecharts forbid mutual knowledge among component modules in order to preserve self-containment: any communication and coordination happens therefore "vertically", from the components to the whole and vice versa, in a part-whole hierarchical fashion. As shown in Figure 2, control as well as any mutual knowledge of current state and behavior is not allowed among component state machines; the state machine in the whole section is instead allowed to exchange control commands with them.

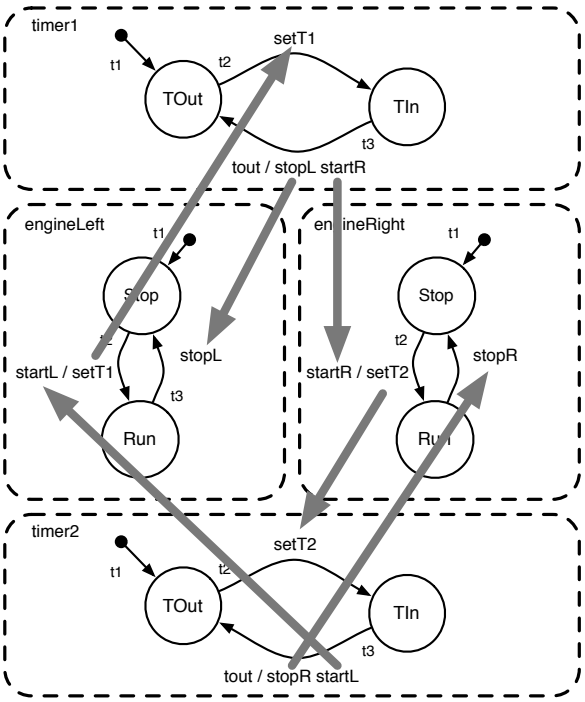

Fig. 3. Implicit modeling of the behavior of the two counterrotating engines by the ordinary Statecharts model of compositions. Gray arrows show mutual interrelationships among concurrent modules.

PW Statecharts can be employed, homogeneously at different description level. The behavior at each level is therefore *part-of* the more complex behaviors at the upper levels.

In the example that follows, PW Statecharts are used to model, for instance, the safe behavior of a medical device at different specification levels, namely the specification concerning a modular component of the device and its behavior (Section III-A), of the device itself (Section III-B) and of a portion of the medical guideline where the device is employed (Section III-C).

A. Subdevice behavior

The main behavior of a blood pump consists in alternating the work of two counterrotating engines.

By the current implicit modeling achieved by traditional Statecharts the behavior of the two counterrotating engines is represented by parallel sections which interact with a pair of timer devices (Figure 3). Once each engine is started, it sets a timer device to its time-in state. After a definite and fixed amount of time, the timer moves to a time-out state and sends a stop command to the engine and a start command to the other engine, and so on.

Checking the safety of the two engines module requires to verify that (1) *exactly one* of the two engines is active at each time and (2) that the two engines are always used one after the another. Both requirements can be expressed through suitable logical temporal formulae and verified through a model checking algorithm.

Figure 3 shows instead the part-whole modeling of the behavior of the two engines. The joint assemblage of the two devies is represented explicitly by an automaton in the lower section of the PWS. Each of the three explicit states denote a working state of the two engines taken as a whole. Both can be stopped (**Off** state) or exactly one engine is active at each time (**TLeft** and **TRight** state).

1) Syntax: The upper part of the diagram, named assemblage section, is separated from the lower part, named whole section, by a dashed line, meaning that communication is allowed among the two sections. The assemblage section hosts the PWSs involved in making the complex behavior, called components. Each component PWS within the assemblage is referred to through a unique identifier. Component PWSs are separated one from the another by a solid line, meaning that communication is not allowed among them. Each component PWS in the assemblage section exposes only its interface, that is it hides implementation and verification details. The interface of a PWS defines the behavior which can be observed and prescribed. State transitions in the whole section may be triggered by both internal (that is coming from the PWSs which compose the assemblage section) as well as external (that is coming from external contexts in which the PWS can be used in further compositions). Transitions are named *external* and *internal* accordingly, the latter being denoted by a small white dot near the starting state. For example, transitions t_2 and t_3 can be triggered by the external events **start** and **stop**, while transitions t_3 and t_4 are triggered by the internal event **tout**.

2) Semantics and Verification: Special state propositions, called state constraints, are used to to assign univocally a meaning to each state in the whole section of the PWS. Three state constraints are for instance used in the PWS of Figure 4; C_1 stands for both engines stopped, while C_2 and C_3 for exactly one engine working and the opposite stopped for exactly the time units taken by the timer in the time-in state:

1) C_1: eR = Stop $\wedge$ eL = Stop;
2) C_2: eR = Stop $\wedge$ eL = Run $\wedge$ t = TIn;
3) C_3: eR = Run $\wedge$ eR = Stop $\wedge$ t = TIn;

with the aim of assigning a meaning respectively to state **Off**, **TLeft** and **TRight**. State constraints are graphically assigned to the states in the diagram by a small black triangle on the top of the state.

Verification is performed interactively at run time, by comparing state constraints and the state diagram in the whole section. abd is centered around two main tasks:

a) Constraint validity: Each time a transition is drawn from state S to state T, a check is statically performed at design time on the compatibility of the transition with the constraint C of the arrival state S. For example, in Figure 4, both transitions t_2 and t_4 have state **TLeft** as their arrival state. Since the constraint C_2 assigned to **TLeft** requires that the left rotating engine must be running, both transitions have to comply with the requirement, by turning the engine on as part of the actions specified within the transitions.

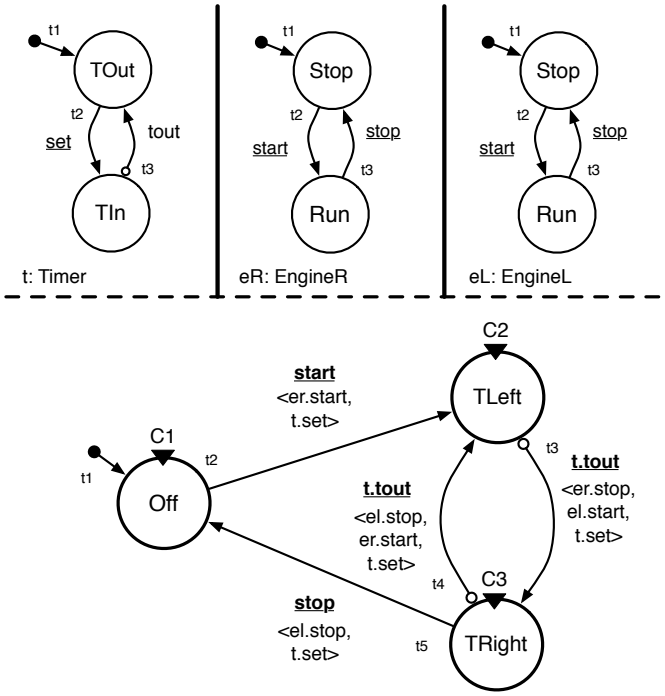

Fig. 4. Explicit modeling of the behavior of the two counterrotating engines of Figure 3 by the PW Statecharts model of compositions. The global behavior is explicitly represented by a three states automaton hosted in the "whole" section of the PWS.

The compatibility of constraints with incoming transitions is performed by computations on a state proposition algebra.

b) Exception handling computation: Each time a state constraint C assigned to a state S may be invalidated by an autonomous, non controllable transition of one of the assemblage components, a suitable internal transition t having state S as starting state has to be added with the aim of moving control out of S once its constraint it no more valid. For example, in Figure 4, constraint C_2 requires the timer to be in the time-in (**TIn**) state when the control in state **TLeft**. When the timer moves to the time-out (**TOut**) state and consequently the constraint in **TLeft** is no longer valid, the *ad hoc* inserted transition t_3 moves the control to state state **TRight**. an algorithm is used to detect whether all feasible uncontrollable events which may invalidate a state constraint are covered by suitable state transitions.

3) Hierarchical Fault Management Mechanism: The main advantage, among the others, in decomposing an overall complex behavior by a single, unifying formalism consists in having events which are allowed to travel uniformly from lower to higher level context, for example from subdevice behavior to alarm devices (and therefore to human operators) in a operating room. Conversely, commands are allowed to travel the other way around, from very complex context to simpler ones.

By assembling each level behavior on top of simpler ones, it is possible implement fault detection and recovery policies at different decomposition levels, as shown in Figure 1. Moreover, different fault management typologies can be adapted to the different hierarchical levels according to the different technologies employed.

At the subdevice level, systems often interact by the synchronous hardware paradigm (also known as *time triggered* paradigm [14]), for example the counterrotating engines discussed above may be controlled by a single logical board. It also reasonable to assume that such synchronous modules are composed asynchronously, for example operate jointly through a message passing communication medium (also known as *event triggered paradigm* [14]). Since hardware devices undergo *random* faults [3] whose happening is not predictable except than in statistical terms, synchronous subsystems assembled from hardware components result in a behavior that can be assumed to be *fail silent*, meaning that they either work or stop responding. Any other causal or malicious behavior which can be subsumed under the general category of *byzantine* failures, may also be reduced to fail-silent behaviors by well know techniques.

B. Device behavior

Fail silent behaviors can, on their turn, be detected and transformed into fail explicit behaviors as part of the specification of the composition layer which embodies them as components. It is shown show, as part of the example, that the fail silent behavior of Figure 4 can become part of the more complex fail explicit behavior, achieved by explicit *fail states*. Such a behavior pertains to the modeling of the entire device, as shown in Figure 5, which then becomes fail explicit.

According to such a behavior, the device tries to reach a working state **Work** by moving the engine module from **Start** to **TLeft**, by transition t_2 which sends the start command to the engine module. For each action sent, there are two possible failure outcomes: either the commanded component do not move to the desired state in a given time or it fails silently, which again means that it takes an indefinitely long time to complete. On the other hand, since both the components and the controller operate and communicate asynchronously, it becomes necessary to achieve, at the programming level, some form of nonblocking synchronization amongst them. The regular behavior consists therefore, by negation, in letting some limited time pass in order for the components to complete the requested operations, before moving the controller to the final state of the transition.

Special intermediate wait states are therefore introduced, like the state W of Figure 5. Let C_2 be the original constraint proposition of the ending state of transition t_2. An intermediate derived proposition C_2' can be associated to state W, meaning "constraint C_2 has not been achieved (yet) *and* less than T time units have been elapsed", where T is the maximum allowed time required to start the engine module.

C. Clinical guideline behavior

Finally, fail explicit devices can be easily intermingled with human behavior in order to obtain fail safe or fail operational behavior. By explicitly stating the overall behavior pertinent

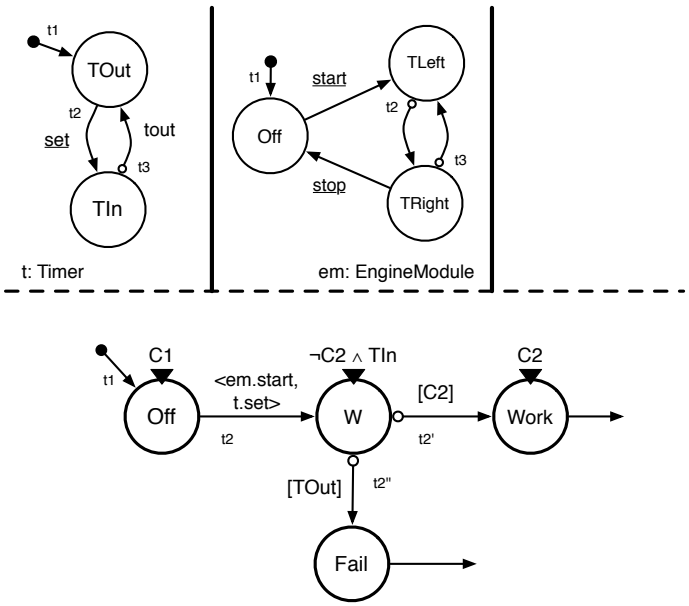

Fig. 5. The fail silent behavior of the counterrotating engine module of Figure 4 can be turned into a fail explicit behavior of the blood pump device by composing it with a timer device. State **Fail** denotes explicitly that the engine module failed to start within a given time.

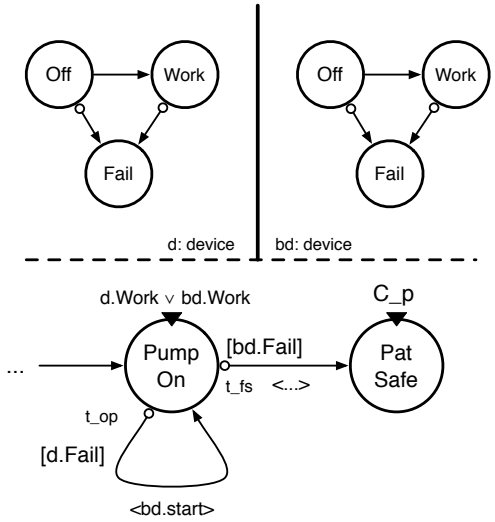

Fig. 6. Two instances of the blood pump behavior (simplified) of Figure 5 can be used in order to obtain a fail operational clinical guideline behavior by the state **PumpOn**. An additional fail safe state **PatSafe** can be added to the behavior once suitable actions are feasible in order to put the patient in a safe state.

a clinical operational scenario, critical global states in which devices are required not to fail can be easily identified. In case of explicit failure, alternate fail operational course of actions can be easily identified, provided backup devices are made available as part of the protocol. Fail safe operations can be also foreseen, provided there exists a safe reachable state (Figure 6).

IV. Conclusion

The paper showed how safety can be enforced starting from the subdevice level to higher level contexts, by a state based hierarchical, highly modular, formalism. Such a formalism has the advantage of making explicit, that is representing by an global state diagram, mutual interactions among the entities present at each level. By such an explicit approach, it is possible to assign state propositions to the global states and to verify them at design time, instead of model checking the design at a later development stage [15][16]. It can be observed finally that well-known fault management strategies fit inherently at the different decomposition levels, thus providing both an additional, empirical, confirmation of the validity of the approach and a stimulus for further research.

References

[1] H. Weatherly, M. Drummond, and D. Smith, "Use of economic evidence in the design of health improvement programmes (himps)," Centre for Health Economics, University of York, Working Papers 042cheop, Apr. 2002. [Online]. Available: http://ideas.repec.org/p/chy/respap/42cheop.html

[2] M. Phil, M. Peleg, S. W. Tu, A. A. Boxwala, R. A. Greenes, V. L. Patel, E. H. Shortliffe and D. Wang, "Representation primitives, process models and patient data in computer-interpretable clinical practice guidelines: A literature review of guideline representation models," *International Journal of Medical Informatics*, vol. 68, no. 68, pp. 1–3, 2002.

[3] W. R. Dunn, "Designing safety-critical computer systems," *Computer*, vol. 36, no. 11, pp. 40–46, 2003.

[4] N. G. Leveson, "Software safety: In embedded computer systems," *Commun. ACM*, vol. 34, no. 2, pp. 34–46, 1991.

[5] L. Pazzi and M. Pradelli, "A state-based systemic view of behavior for safe medical computer applications," *Computer-Based Medical Systems, IEEE Symposium on*, vol. 0, pp. 108–113, 2008.

[6] D. Harel, "Statecharts: A visual formalism for complex systems," *Science of Computer Programming*, vol. 8, pp. 231–274, 1987.

[7] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang, "Symbolic Model Checking: 10^{20} States and Beyond," in *Proc. IEEE Symposium on Logic in Computer Science*. Washington, D.C.: IEEE Computer Society Press, 1990. [Online]. Available: citeseer.ist.psu.edu/burch90symbolic.html

[8] G. Booch, J. Rumbaugh, and I. Jacobson, *The Unified Modeling Language User Guide*. Addison-Wesley, 1998.

[9] T. Knape, L. Hederman, V. P. Wade, M. Gargan, C. Harris, and Y. Rahman, "A uml approach to process modelling of clinical practice guidelines for enactment," 2003.

[10] I. Porres, E. Domínguez, B. Pérez, A. Rodríguez, and M. A. Zapata, "A model driven approach to automate the implementation of clinical guidelines in decision support systems," in *ECBS '08: Proceedings of the 15th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 210–218.

[11] L. Pazzi, "Extending statecharts for representing parts and wholes," in *Proceedings of the EuroMicro-97 Conference, Budapest, Hungary*, 1997.

[12] ——, "Part-whole statecharts for the explicit representation of compound behaviors," in *Proceedings of the UML 2000 Conference, York (UK)*, ser. LNCS, vol. 1939. Springer, 2000, pp. 541–555.

[13] ——, "A method for ensuring safety and liveness rules in a state-based design, http://cris.unimore.it/cris/files/2008-02-01.pdf," Tech. Rep. CRIS-2008-02-01, 2008, patent pending PCT/EP2008/051300.

[14] H. Kopetz, "The time-triggered model of computation," in *IEEE Real-Time Systems Symposium*, 1998, pp. 168–177.

[15] B. Perez and I. Porres, "Verification of clinical guidelines by model checking," *Computer-Based Medical Systems, IEEE Symposium on*, vol. 0, pp. 114–119, 2008.

[16] P. Martini, K. Kaiser, and S. Miksch, "Easing the formalization of clinical guidelines with a user-tailored, extensible agile model driven development (amdd)," *Computer-Based Medical Systems, IEEE Symposium on*, vol. 0, pp. 120–125, 2008.