Automatic Fault Behavior Detection and Modeling by a State-based Specification Method

Luca Pazzi
Department of Engineering Sciences
DII-UNIMORE
Modena, Italy
Email: luca.pazzi@unimore.it

Matteo Interlandi
Department of Engineering Sciences
DII-UNIMORE
Modena, Italy
m.interlandi@gmail.com

Marco Pradelli
Department of Engineering Sciences
DII-UNIMORE
Modena, Italy
marco.pradelli@unimore.it

***Abstract*—Safety assessment methods are typically based on the reliability of the single components making a system. A different notion of safety as an emergent property of the system taken as a whole is however emerging. The current state-based modeling paradigm tends at misrepresenting systemic behavior, thus contrasting the adoption and development of systemic compositional fault detection techniques. We propose a state-based formalism, highly committed towards the explicit representation of systemic behavior, by which it is possible to formally identify faulty behaviors once the regular one has been specified.**

I. Introduction and Motivation

By a widely accepted, albeit partial, point of view, complex systems fail due to some physical fault of some of their component parts. According to such a point of view, reliability and safety analysis methodologies help a safety designer in the safety assessment of the causal chain which led to the system failure. A different approach is however emerging, by which component reliability does not necessarily implies system safety. According to Leveson [1], *system safety*, the absence of accidents at the system level, is essentially an emerging property of the system taken as a whole. System accidents arise due to the *logical* complexity of the system, that by the set of potential interactions among the components. This kind of complexity, which may be referred to as *interactive*, has been further exacerbated by the introduction of computers, by which arbitrary interactions and mutual conditon testing may force a system to do what is not expected to do, or vice versa. As a consequence, a system made of reliable and working parts is not necessarily safe, i.e., it may undergo a major fault even if no components have failed.

It appears thus necessary to deal with a sound notion of system modeling in order to fully exploit the power of new systemic and compositional fault detection and avoidance techniques. Part-Whole Statecharts [2][3][4] allow to model *explicitly* the systemic behavior through a state diagram hosted within a specific construct in the formalism and allow, at the same time, to express *system state invariants*, that is logical propositions which hold of the system components for each state of such a system diagram. It is therefore possible to define formally, at design time, the semantics of the regular, nominal system behavior.

We expect, as a direct consequence, that once systemic behavior has been specified, faulty behaviors can be formally and automatically derived as part of the design process. Since, by an established result, the reactive behavior of the system is determined, jointly, by the state invariants and by the state machines which compose the system, we delineate research directions by which, with similar techniques faulty and boundary states can be identified and safety-critical components can be formally identified.

II. Part-Whole Statecharts

Most of current modeling methods, language and tools, tend at misrepresenting a system of interacting parts by a network of interrelated and communicating entities. The current modeling paradigm is mainly inspired by the fact that real world systems evolve through the effect of physical mutual relationships according to definite physical laws. The Statecharts [5] coordination and communication model, for example, *implicitly* assembles a global behavior by letting the state machine residing on each module to be both aware of the current status of the other machines and able, at the same time, to act on them by sending events which trigger further state transitions. Obtaining a systemic coordinated behavior through such a model has however severe limitations in terms of software quality factors, since the global behavior tends at being represented in a fragmented form, and as such it is difficult to understand, reuse and maintain. Moreover, its operational semantics is necessarily ambiguous since control events bounce from one component to the another depending on the current status of each one.

On the other side, Part-Whole Statecharts (shortened either as PW Statecharts or PWS) were conceived with a radical commitment towards the *explicit* modeling of the global behavior. A PWS consists of two main sections: one hosting a set of component state machines, referred to collectively as the *assembly*, and the other containing a single state machine representing the system behavior as a whole, called indeed the *whole*. Control as well as any mutual knowledge of current state and behavior is not allowed among component state machines; conversely, the whole is allowed to know, at each time, the current state of each component state machine as well as to send control commands to them. The whole is

finally notified of each state transition happening within the assembly of components. Such communication and knowledge restrictions ensure that behavioral component description are self-contained, since they are not allowed to refer to any of the peer components or to the system state machine and thus achieving modularity and full reusability: consequently the entire semantics of coordination and communication is explicitly transferred to the whole section. PWSs distinguish among two kinds of transitions: *non controllable transitions* which correspond to automatic or sensing behavior of a device and *controllable transitions* which correspond to an action that has to be accomplished.

III. State Invariants and Exit Zones

PWSs have been recently integrated with the innovative feature of allowing the definition of the semantics of each state during the design process, through state invariants [4]: by such propositions the designer specifies formally which states are allowed to be globally assumed by the system components when the system is within one of the states of the whole section which has such proposition as invariant. According to such a framework, faulty behaviors can be seen as violations of state invariants, by which the designer specified the correct nominal behavior. This is part of a more general mechanism by which the reactive behavior is formally determined by both logical propositions denoting state invariants and component state machines, whose behavior may invalidate the current state invariant by non controllable transitions, that is transitions which are not under the control of the whole section.

In this context, invariants can be used to investigate the *exit zones* of each state, i.e. combinations of logical propositions and non controllable transitions which may invalidate the current state invariant and then force the controller to move out to another state. The semantics of every state in the whole can then be indeed divided in two parts: *exit zones*, where, as mentioned above, given a proposition included in the state invariant, the happening of certain non controllable transition will bring the controller to jump to another state; and *internal zones* where, in the opposite way, the state invariant is always valid. In real time systems the individuation of exit zones is fundamental since it allows not only to formally characterize where the system needs to be *reactive*, but in addition it provides the possibility to be even *proactive*. Given in fact an exit zone, actions can be taken *proactively* before the state invariant is invalidated by the actual triggering of the related transition.

IV. Boundary States and Critical Components

The formal knowledge of the semantics of each state defines directly which state is part of the system nominal behavior and which one, instead, has to be considered faulty. Given this distinction, *boundary states* can be defined as the states where the systems is still working properly (although possibly with degraded performance) but non controllable transitions could cause the controller to move to a state whose semantics is considered faulty within the systemic behavior. In other words, once a distinction between nominal and faulty states has been made, it is considered boundary every state within the correct system behavior which is directly linked to a faulty state by a non controllable transition. Boundary states identification allows thus to know which combination of events may cause the system to switch to a faulty behavior.

In case fault detection mechanisms are not already implemented at either the hardware or software level, imaginary faults can be easily injected in the PWS model at the component level. Depending whether boundary states are identified at the system level, faults which might result in a systemic faulty behavior can be consequently detected. In this way, components that are critical for the system can be identified and countermeasures can then be adopted, typically by the introduction of redundant or spare parts. Such parts increase the probability that the system will operate properly, since new working states are added to the model proportionally to the number of redundant components which are added, thus increasing the distance of the nominal behavior from the boundary states and therefore from the potential fault behavior.

V. Conclusions

The paper illustrates the main concepts of a technique for detecting fault behaviors at the system level once faults are modeled at the component level in a state-based modeling formalism. The technique is an offspring of a broader research which assigns state semantics directly at design time through propositional state invariants. This allows not only to denote explicitly the global state of an assembly of components, but also to detect faulty behaviors as exceptions to the regular one. Within a proposition constituting a state invariant, exit zones can be identified making the system reactive – and potentially even proactive – with respect to a certain non controllable transition. Extending the same concept at the system behavior level, boundary states can be identified where the nominal systemic behavior could switch to a faulty behavior. Through fault injection or fault detection techniques at the component level, new boundary states can be also detected and critical components consequently recognized. In conclusion, emergent properties of the system behavior, such as safety and critical components individuation, are better handled by systemic-oriented formalisms, like PW Statecharts, which allow the explicit representation of both nominal and faulty behaviors.

References

[1] N. G. Leveson, *Safeware: system safety and computers*. New York, NY, USA: ACM, 1995.

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

[3] ——, "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.

[4] ——, "Modular model checking in part-whole statecharts," Tech. Rep. CRIS-2008-02-01, 2008, http://cris.unimore.it/cris/files/CRIS-2008-02-01.pdf.

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