A Diagnosis Framework for Critical Systems Verification (Short Paper*)

Vincent Leildé[1], Vincent Ribaud[2], Ciprian Teodorov[1], and Philippe Dhaussy[1]

[1] Lab-STICC, team MOCS, ENSTA-Bretagne, rue François Verny, Brest, France
firstname.lastname@ensta-bretagne.fr,
[2] Lab-STICC, team MOCS, Université de Bretagne Occidentale, Avenue le Gorgeu, Brest, France Vincent.Ribaud@univ-brest.fr

Abstract. For critical systems design, the verification tasks play a crucial role. If abnormalities are detected, a diagnostic process must be started to find and understand the root causes before corrective actions are applied. Detection and diagnosis are notions that overlap in common speech. Detection basically means to identify something as unusual, diagnosis means to investigate its root cause. The meaning of diagnosis is also fuzzy, because diagnosis is either an activity - an investigation - or an output result - the nature or the type of a problem. This paper proposes an organizational framework for structuring diagnoses around three principles: that propositional data (including detection) are the inputs of the diagnostic system; that activities are made of methods and techniques; and that associations specialize that relationships between the two preceding categories.

Keywords: Diagnosis, Verification, Critical systems, Framework

1 Introduction

Critical systems are concerned by *dependability*, i.e. the ability of an entity to perform as and when required [3], that requires the *means* of improving the quality of systems design. This should be realized in three cyclical phases: verification, diagnosis and correction. Verification aims to demonstrate whether a system meets specification properties. This may be achieved using various techniques such as static analysis, simulation or model checking. Model checking is an automated technique that, given a finite-state model of a system and a formal property, systematically checks whether this property holds for that model [4]. If a property is violated, a counter-example is produced as a trace from the initial state to the state in which the error was detected. This triggers a diagnosis process (generally carried out through detection, localization and identification tasks) that aims to outline the violation root causes. Consequently, the system is corrected and the design cycle repeated.

* This is a short paper accepted in the new ideas and work-in-progress section of SEFM 2017.

There are many frameworks and approaches for performing a diagnosis [2, 5, 6, 11, 13], all of which face the following two issues. First, poor management and control [19] of the verification process produces a profusion of heterogeneous interrelated models that makes it more difficult to understand errors, for instance, localizing the relevant parts in a detailed source-level trace to identify why a verification run failed [10]. Second, diagnosis is also loosely formalized. As a result, models produced during the design and verification process are not well adapted to diagnosis tasks. Therewith, diagnosis is weakly integrated with other activities and interoperability between tools and processes is not easily achieved.

For the above reasons, understanding and formalizing the diagnosis is intended to foster the definition of diagnosis tools and methodologies, and reduce the set of diagnoses. If diagnosis is applicable in different fields (medicine, plant and process supervision), frameworks differ, and cannot be fully applied to trace-based diagnosis. We propose an organizational framework for diagnosis systems, based on three concepts: *activity*, *propositional object* and *association*.

2 Background

If model checking is often dedicated to faults detection, some frameworks also employ it for faults localization. For instance, slicing-based approaches [21] use dependency analysis to retrieve the set of elements which contains the fault. State space reduction [13] aims at reducing the state space size by exploiting the concurrent transitions commutativity. Ball et al. [5] introduced an approach to compare the counter-examples with successful traces and thus isolate faulty state transitions. In [6], the authors propose a Symbolic Model Checking framework for safety analysis diagnosis. These approaches focus on trace processing, and the identification task, i.e. identifying the specific nature of faults, is not considered. Consequently, a semantic gap between design models and traces still holds.

Some approaches allow for a complete diagnostic. For instance in [11], the authors define a framework that combines an abductive model-based diagnosis approach with a Labelled Transition System. This kind of method is also experienced by [2], who associated logic learning with trace-based diagnosis and error correction using positive and negative traces. These approaches are restricted to one diagnosis technique, model-based, that imposes the presence of either a fault or a well-functioning model, which is not always available.

Venkatasubramanian [20] has broadly classified fault diagnosis methods into quantitative model-based methods, qualitative model-based methods, and process history-based methods. This classification provides a large spectrum of methods and techniques, but focuses on industrial processes, and put aside important techniques for trace-based diagnosis like interaction-based techniques.

To the best of our knowledge, there are no frameworks for characterizing diagnosis systems, unrestricted to any diagnosis techniques, activities or application domains. Therefore, we focus on understanding diagnosis in order to identify a core set of concepts that can be applied for any diagnoses systems.

3 Conceptual Framework

We propose a framework for characterizing diagnosis systems, not restricted to a diagnosis technique or method. We start from a general definition of diagnosis given by Merriam Webster [1] : "*diagnosis is an investigation or analysis of the cause or nature of a condition, situation, or problem*". This framework is based on three concepts: - *Activity*, a set of mechanisms or tasks used to perform the diagnosis; - *Propositional object*, tangible or immaterial, produced or consumed by activities ; - *Association* between propositional objects and activities.

3.1 Activities

The foremost part of the diagnosis definition refers to an *activity*, whether *an investigation or an analysis*. An *activity* is a set of cohesive *tasks*. Activities and tasks use *mechanisms* as means to achieve their outcomes.

Diagnosis tasks. According to the literature, diagnosis systems support three main tasks, *fault detection*, *isolation*, and *causal analysis* [20].

Fault detection establishes that a system run raises so-called abnormal event. In the particular case of verification by model checking, detection is done by model checking itself.

When a diagnosis is required, the ensuing step consists in *isolating* the subset of elements, part of models, that needs to be corrected [9]. *Isolation* is performed through various techniques, such as slicing-based approaches [21], state space reduction techniques [13] or counter-example comparisons [5].

Once suspicious elements are localized, the *causal analysis* task, associates causes to the observed abnormalities. This is generally a reasoning process, either deductive, inductive or abductive. Deduction is concerned by deducting knowledge from already learned knowledge, induction identifies general rules from observations, and abductive reasoning discovers causes from facts by elaborating hypothesis. Each type of reasoning fits with a different situation, abduction produces ideas and concepts to be explained, then induction contributes to the construction of the abductive hypothesis by giving it consistency, finally deduction formulates a predictive explanation from this construction [8].

Mechanisms. Activities and tasks are supported by a set of mechanisms, including tools and methods, that can be organized in *model-based* or *process history-based* category. We complete the list with an *interaction-based* category, relevant for trace-based diagnosis.

Model-based mechanisms assume that a model of the system is available, representing its correct (consistency-based) [18] or abnormal (abductive-based)[22] behavior. In consistency-based, the reasoning consists in rejecting a set of assumptions using the correct behaviour, to restore consistency with (abnormal) observations [7]. In the opposite, abductive-based reasoning works with causes and effects models, for instance using Inductive Logic Programming to provide automated support for correcting the errors identified by model checking [2].

Process-history based mechanisms relies on the availability of large amount of historical process data. Mechanisms may use knowledge extraction techniques, like data mining or statistical analysis. Liu [15] uses statistical models to remove

false positive counterexamples. Probabilities can also be applied, using decision trees or Bayesian networks. In machine learning approach, neural networks and case-based reasoning try to reproduce the human way of reasoning. When a strong expertise is available, one can simply use expert systems gathering problems set, rules and an inference engine.

Interactions-based mechanisms allow for observing, controlling, understanding and altering the system execution. By storing the execution traces, omniscient debuggers enable back-in-time navigation features, postmortem query processing, trace-analysis and reduction facilities, and execution replay [17]. Besides, a large number of visualization tools exists [12], including diagram structures ranging from waveforms, finite state machines and business representations.

3.2 Propositional Objects

Activities handle different kinds of information [1], whether "situation or problem". As information may be tangible or immaterial, we define any information items as *propositional objects* that are, or represent sets of propositions about real or imaginary things.

Set of circumstances. Situation or problem are related to propositional objects. A *situation* is a way in which something is positioned with respect to its surroundings [1]. Regarding model checking, propositional objects comprises design models, properties, exploration graphs or model checker configurations. A *problem* is a difficulty that has to be resolved or dealt with [1]. Thus situation and problem are generalized in a concept called *set of circumstances*.

Observation. Problems are revealed by *symptoms*, a special case of *observations*, which are effects or visible consequences of the passage of the system into an abnormal state. Regarding model checking it includes counterexamples. As stated by [18], "real world diagnostic settings involve observations, and without observations, have no way determining whether something is wrong and hence whether a diagnosis is called for".

3.3 Associations

"Cause or nature" are both *diagnoses*, i.e. statements or conclusions from diagnosis analysis [1]. A diagnosis specializes an *association* between activities and propositional objects. Following a systemic triangulation, we organize *diagnoses* in three viewpoints, *causality*, concerned with functional aspects, *nature*, concerned with structural aspects, and *evolution*, concerned with historical aspects.

Causality is defined by [16] as a sequence of linked events. Consider for instance a car with flat tires that suddenly slips on a water poll, resulting to an accident. The accident is a succession of related events. Closed to our concerns, a *Fault*, an *Error* and a *Failure* are considered for [3] as causal events, a *fault* may produce an *error*, which may lead to a *failure*.

Nature consists in determining the type, the characteristics or the essence of something, "what the object is". By taking up the example of a car, the owner inspects each tires and finds that some are more damaged than others, and classifies one tire in the category "too flat". The *nature* association itself can be refined in more specific relations, like generalization or specialization.

Evolution represents the historical, that is linked to the evolutionary nature of the system, "what the system was or is becoming". For instance, a man is driving when an impact happen closed to the car wheels. He remembers he found one flat tire during his last car inspection, and supposes the tire is scratched.

4 Framework by example

We present different kind of diagnosis systems using our framework, each pursuing a different objective. We refer to the classical example of a one bit adder, see in Fig. 1 an illustration from wikimedia commons *Full-adder.svg*. A full adder is composed of two AND gates, two XOR gates, and one OR gate.

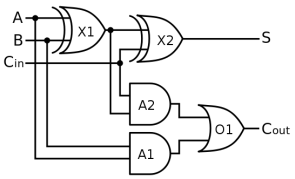

Fig. 1. Full adder

(**Analysis** ∨ *Investigation*) of the (*Cause* ∨ **Nature** ∨ *Evolution*) of a (**Situation** ∨ *Problem*) ⇒ **Pedagogical objective**	(*Analysis* ∨ **Investigation**) of the (**Cause** ∨ *Nature* ∨ *Evolution*) of a (*Situation* ∨ **Problem**) ⇒ **Curative objective**	(**Analysis** ∨ *Investigation*) of the (*Cause* ∨ *Nature* ∨ **Evolution**) of a (**Situation** ∨ *Problem*) ⇒ **Prognosis objective**

Table 1. Diagnosis Systems Examples

An analysis of the nature of a situation pursues a *pedagogical* objective. If we are not aware of the purpose of a digital circuit, we might build the truth table which sets out the output values for each combination of input values. The truth table is a diagnosis that helps to understand how the circuit works (assuming the circuit behavior is normal). The analysis associates outputs (observations) to inputs (facts) and tries to figure out the nature of the circuit. Regarding verification, simulation activity helps to understand the way the system behaves, or ensure it behaves correctly.

An investigation of the cause of a problem pursues a *curative* objective. Consider we expect from the circuit a full adder behavior, and thus one expected property is *P1 : "for the set of entries A=1, B=1 and C=1, the result is S=1 and Cout=1"*. Assume that the XOR gate X1 was inadvertently replaced by an OR gate. Then the output of the circuit conflicts with the property P1, i.e. *S=0 and Cout= 1"*, and we must investigate the cause of the failure. Regarding model checking, if a violation of functional specifications is discovered by a model checker. One has to correct the design or model accordingly.

An analysis of the evolution of a situation pursues a *prognosis* objective. Given a set of properties (probably non-exhaustive), running the model checker over the set without any errors yields an indication that the circuit, as far we know, behaves correctly. The underlying diagnosis is used as a prognosis of circuit major dysfunctions. In software, design patterns, like security patterns, are prevention mechanisms. Regarding model checking of system design, if we consider a set of historical state spaces, one could apply design prognosis by using statistical and probability analysis.

5 Conclusion

In this paper we presented core concepts of a framework for understanding diagnosis. We believe that this set of concepts will enable the exploration of the possible and constrained compositions of diagnostic systems, reducing the minimal set of diagnoses. This work paves the way for the construction of an organizing system, an ongoing work [14], for storing system data (propositional objects), interpreting them (association), and diagnosing the critical systems (activities).

References

1. Dictionary | Merriam-Webster, `https://www.merriam-webster.com`
2. Alrajeh, D., Kramer, J., Russo, A., Uchitel, S.: Automated support for diagnosis and repair. Communications of the ACM 58(2), 65–72 (2015)
3. Aviienis, A., Laprie, J.C., Randell, B., Landwehr, C.: Basic concepts and taxonomy of dependable and secure computing. Dependable and Secure Computing, IEEE Transactions on 1(1), 11–33 (2004)
4. Baier, C., Katoen, J.P.: Principles of model checking. The MIT Press, Cambridge, Mass (2008)
5. Ball, T., Naik, M., Rajamani, S.K.: From symptom to cause: localizing errors in counterexample traces. In: ACM SIGPLAN. vol. 38. ACM (2003)
6. Bertoli, P., Bozzano, M., Cimatti, A.: A symbolic model checking framework for safety analysis, diagnosis, and synthesis. In: MoChArt'16. pp. 1–18. Springer (2006)
7. Bourahla, M.: Model-Based Diagnostic Using Model Checking. IEEE (2009)
8. Buccafurri, F., Eiter, T., Gottlob, G., Leone, N.: Enhancing model checking in verification by AI techniques. Artificial Intelligence 112(1), 57–104 (1999)
9. Cleve, H., Zeller, A.: Locating causes of program failures. p. 342. ACM Press (2005)
10. Groce, A., Visser, W.: What went wrong: Explaining counterexamples. In: Model Checking Software, pp. 121–136. Springer (2003)
11. Gromov, M., Willemse, T.A.: Testing and model-checking techniques for diagnosis. In: Testing of Software and Communicating Systems, pp. 138–154. Springer (2007)
12. Hamou-Lhadj, A., Lethbridge, T.C.: A survey of trace exploration tools and techniques. In: CASCON '04. pp. 42–55. IBM Press (2004)
13. Holzmann, G.J.: The Theory and Practice of A Formal Method: NewCoRe. In: IFIP Congress (1). pp. 35–44 (1994)
14. Leilde, V., Ribaud, V., Dhaussy, P.: An Organizing System to Perform and Enable Verification and Diagnosis Activities. In: IDEAL. pp. 576–587. Springer (2016)
15. Liu, Y., Xu, C., Cheung, S.: AFChecker: Effective model checking for context-aware adaptive applications. Journal of Systems and Software 86(3), 854–867 (Mar 2013)
16. Mackie, J.L.: The cement of the universe: a study of causation. Clarendon library of logic and philosophy, Clarendon Press, Oxford, 5. dr. edn. (1990), oCLC: 258760915
17. Pothier, G., Tanter, ., Piquer, J.: Scalable omniscient debugging. ACM SIGPLAN Notices 42(10), 535–552 (2007)
18. Reiter, R.: A theory of diagnosis from first principles. Artificial intelligence (1987)
19. Ruys, T.C., Brinksma, E.: Managing the verification trajectory. International Journal on Software Tools for Technology Transfer (STTT) 4(2), 246–259 (Feb 2003)
20. Venkatasubramanian, V., Rengaswamy, R., Kavuri, S.N.: A review of process fault detection and diagnosis. Computers & Chemical Engineering 27(3) (Mar 2003)
21. Visser, W., Havelund, K., Brat, G., Park, S., Lerda, F.: Model checking programs. Automated Software Engineering 10(2), 203–232 (2003)
22. Wotawa, F., Rodriguez-Roda, I., Comas, J.: Abductive Reasoning in Environmental Decision Support Systems. In: AIAI workshops. pp. 270–279. Citeseer (2009)