Algebraic Modeling of Diagnostic Problems in Hw-Sw Codesign *

A. Pataricza
Technical University of Budapest
Department of Measurement and Instrument Engineering
H–1521 Budapest
E-mail: pataric@mmt.bme.hu

The widespread use of embedded controllers in dependability-relevant products requires a radical increase both in the productivity and in the quality of the overall design process, thus necessitating the automation and the integration of the functional and diagnostics design processes. Recently, the most effective system design methodology is hardware-software codesign, supporting a unified modeling of hardware and software at the highest level of abstraction, hierarchical step-wise model refinement, and finally, automatic silicon and software synthesis.

Data-flow network (DFN) based modeling is the most widely used paradigm in the initial phases of the hardware-software codesign process, as it provides a user-friendly graphic interface for the practitioner engineer with an exact mathematic formalism in the background. The DFN model of the system is composed of *nodes* (functional elements) interconnected by unidirectional *channels*. The presence and flow of data items across the system are represented by *tokens*, moving along the channels.

A node starts its computation, when a sufficient number of tokens is present in its input channels as prescribed by some of its *firing rules*. If multiple rules of the same node are simultaneously in a fireable state, then the actually fired one can be selected either by a predefined priority scheme or by a random choice, similarly to the Petri-nets. Tokens are produced on the output channels and the node enters a new state after firing. [1]

Uninterpreted DFNs are used at the highest level of abstraction to model the dynamics of the flow of data between the individual functional elements of the system under design. Only the presence of data is modeled at this level, in the form of so-called simple or *un-coloured* tokens. Data dependencies are either neglected, or represented only by a non-deterministic behavior of the corresponding element. (Note, that in the firing rules of the individual DFN nodes, non-unique mappings formulated as *relation*s can be used instead of the more restrictive class of *function*s.)

Interpreted DFNs use different types of tokens in a similar way, as coloured Petri-nets do it. Interpreted modeling is typically used during the functional design process for the fine granular modeling of the target system, including the description of data dependencies. Naturally, this high level of model realism results in a huge computational complexity in the evaluation phase. However, the main intention of approach was to provide a support to the designer during the architecture design phase in such typical decisions, as which components have to be replicated in order to assure a system level fault tolerance etc. Accordingly, the domains of the token types are kept very limited. For instance, tokens can be coloured either as *correct* or as *erroneous* according to the fault state of the informa-

*This work was carried out during a stay at the FAU Erlangen-Nuremberg IMMD3 sponsored by DFG SFB 182, with an additional grant of the Hungarian NSF Project OTKA-T015728

[1] Channels behave like ideal (capacity unbounded and latency free) FIFO queues. The notion of deterministic and/or stochastic execution times can be associated with firings in timed DFNs in a similar form, as in Petri-nets. In the current paper only untimed DFNs with immediate firing rules will be used.

tion delivered by them. Even these, highly simplified *qualitative DFN-error models* are capable to represent both the effects of *faults in the components* and *error propagation* in the network, as well.

A complete model of the effects of the component faults on the system behavior can be composed, if the set of relations describing the functional input-state-output correlation of the corresponding component is extended by:

- data transfer relations for each individual fault in the component (describing the local effect of the fault) in a form similar to the firing rules in the fault-free case, and
- the propagation rules defining the reactions to the different erroneous input values.

Coloured tokens play a similar role in this formalism, as multiple valued logic $(0, 1, D, \overline{D})$ in traditional gate level testing. For instance, a firing rule defining the behavior of the component in the presence of a specific fault is the counterpart of a primitive cube of a fault (pcf). Due to this analogy, ATPG can be performed by processing this model in a very similar way, as in the case of gate level testing [**?**].

Note, that the flexibility of this approach exceeds essentially that of the traditional system level diagnostic modeling. For instance, neither the domain of the fault states, nor the range of the error effects are restricted to be binary, thus within the limits of a reasonable computing complexity, a fine granular diagnostic resolution is supported e.g. by allowing the introduction a separate color to each failure mode of a component. Moreover, a Russell-Kime type model of test invalidation can be simply represented by a non-deterministic behavior in the faulty case.

Additionally to ATPG, practically the entire spectrum of diagnostic problems can be handled based on this modeling approach. The subsequent part of the paper presents a model transformation into an algebraic formulation, for which operation research offers effective solution methodologies even for large-scale complex systems.

The tokens in a test sequence enter the system through some controllable primary input(s) in an error-free state and leave the system at least partially re-coloured to erroneous ones through some observable outputs. In an other form, the effect of the test sequence along the affected paths is a firing sequence, in which:

- Each individual firing is consistent with some of the firing rules of the corresponding component;
- The entire token flow associated to the test sequence traverses the DFN without altering the internal token distribution, thus the total balance between the token counts at each node before and after test execution equals 0.

The basic idea of the new algorithm outlined here originates in the analogy between these token balance and T-invariants (firing sequences not altering the token count) in Petri-nets (PN). [2] (A DFN itself can be unfolded to a PN).

Accordingly, the correlation between input test sequences, fault states and output test responses in the form of the token balance described above can be defined by a homogeneous linear Diophantine equation system, (referred further on as the *system constraint*). The variables in this equations correspond to *input activities* (e.g. which inputs were activated, when the failure occurred or which inputs have to be triggered for detect a specific fault), *output firings* (where are the failures observable), and *fault states* of the individual components, respectively. Additional inequalities express for instance the permanency of faults (at most a single value can be associated with a particular fault state).

Several diagnostic problems can be formulated and solved by means of this equation system depending on the variables of a known value. For instance:

[2] In an earlier work [**?**] a limited diagnostic model was presented describing the error propagation effects by deterministic, negation-free Horn-clauses and solving the problem of diagnostics by deducing it to a T-invariant problem.

- If the input activities and fault states are known, *fault simulation* can be carried out by estimating the corresponding output values of the system;
- The estimation of the set of fault states compatible with the known input values and output sequences delivers a complete *diagnostic image*. As a special case, when estimating the faulty states consistent with some input test sequence and the reference (fault free) test response generated by it, *faults non-covered* by this candidate test can be estimated in a similar way, as effect-cause analysis does it at the logic gate level.

Note, that either a *single* arbitrary solution or *all* solutions can be estimated by solving the problems described above. However, frequently a single, but in some terms *best* solution of the diagnostic problem is searched for. The system constraints have to be extended by an additional *objective function* in order to find an optimal solution of a diagnostic problem. Integer programming and/or constraint satisfaction algorithms provide algorithms [**?**] to find a solution of an extremal (minimal or maximal) objective function value for a variety of objective function classes (e.g. linear or quadratic) even for large scale systems. This way optimized diagnostics can be performed, among others like:

- *minimal diagnostics* in the terms of *number of faulty elements*, as objective function to be minimized;
- *maximum-likelihood diagnosis* is carried out by using the occurrence probability of a fault state vector as objective to be maximized;

.

Moreover, even *quantitative dependability features* can be formally proven by the introduction of a proper objective function. For instance, the designated t-error detection of a system can be directly checked by estimating the minimal number of faults resulting in a fault-free response without any error indication. In this case, the objective function consists simply the the number of faults, and the entire solution space is to be checked, without applying any restrictions on the input sequence.

References

[1] P. Barth. *Logic Based* $0-1$ *Constraint Programming.* Kluwer Academic Publishers, Boston, London, Dordrecht, 1996.

[2] Gy. Csertán, A. Pataricza, and E. Selényi. Dependability Analysis in HW-SW codesign. In *Proceedings of the IEEE International Computer Performance and Dependability Symposium, IPDS'95*, pages 316–325, April Erlangen, Germany, 1995.

[3] L. Portinale. Exploiting T-invariant analysis in diagnostic reasoning on a Petri net model. In M. A. Marsan, editor, *Application and theory of Petri nets 1993*, volume 691 of *Lecture notes in computer science*, pages 339–356, Berlin..., 1993. Springer.