Interchange Formats and Automated Benchmark Model Generators for Model-Based Diagnostic Inference

Alexander Feldman[1] and **Gregory Provan**[2] and **Arjan van Gemund**[1]
[1]Delft University of Technology
Faculty of Electrical Engineering, Mathematics and Computer Science
Mekelweg 4, 2628 CD, Delft, The Netherlands
Tel.: +31 15 2781935, Fax: +31 15 2786632, e-mail: {a.b.feldman,a.j.c.vangemund}@tudelft.nl
[2]University College Cork, Department of Computer Science, College Road, Cork, Ireland
Tel: +353 21 4901816, Fax: +353 21 4274390, e-mail: g.provan@cs.ucc.ie

Abstract

This article proposes a Diagnosis Interchange Format (DIF), an XML-based interchange format for Model-Based Diagnosis (MBD). Its main purposes are to allow sharing of diagnostic models, observation data and fault hypotheses, and to facilitate empirical comparative study of the performance of existing and future MBD implementations. In this paper, we describe the syntax and the semantics of DIF as well as the principles underlying its design. Several examples are used to illustrate the use of DIF, with a particular focus on expressing structure, state and constraints for various domains. We also recommend several sources for creating a standardized MBD benchmark set and discuss possible extensions in subsequent versions of the format. We compare the proposed format to related approaches used in some modeling languages.

Introduction

The field of Model-Based Diagnosis (MBD) is in need of a repository of standardized models in order to test the efficiency of algorithms and the adequacy and efficiency of modeling representations. Algorithmic development in other AI disciplines (SAT, CSP, automated planning) has benefited from the existence of widely-accepted problem representations and benchmark sets. Since MBD covers a heterogeneous range of domains, ranging from discrete circuit through continuous value dynamical systems like ecosystems, *standardizing* an MBD representation is a challenging task. This paper defines an interchange format for MBD. We show how this model covers a wide range of model types, and compare this language with languages for related purposes.

From the algorithmic perspective, the task of finding a kernel diagnosis of minimal cardinality is $\Pi_2^{\mathbf{P}}$-complete (Eiter & Gottlob 1995), but it is not known whether it is computationally difficult for the "average" real-world system. There has been no systematic study of the complexity of diagnosing real-world problems, and few good benchmarks exist to aid in such a study. The pool of existing benchmarks can be supplemented by automatically-generating models (Provan & Wang 2007b) that have the properties of real-world models.

The ISCAS-85 set of combinatorial circuits (Brglez & Fujiwara 1985) has been used as the *de facto* benchmark in MBD. Unfortunately, the hierarchy in the original ISCAS-85 circuits has been flattened-out and they have been distributed in a simple Netlist format. Instead of the original flat representation, we have translated it to our suggested interchange format the reverse engineered ISCAS-85 circuits (Hansen, Yalcin, & Hayes 1999).

From the representational perspective, several different modeling representations have been developed, such as the Java-Based Model Programming Language (Williams & Nayak 1996), KOALA (Benazera, Travé-Massuyès, & Dague 2002), HYBRID CC (Carlson & Gupta 1998), LYDIA (Pietersma, Feldman, & van Gemund 2006), many of which are mutually incompatible. Hence, model-sharing under current conditions is virtually impossible. These issues can be overcome by developing a standard model format.

We do not expect diagnostic reasoners to support the full Diagnosis Interchange Format (DIF) specification; e.g., a solver capable of reasoning in propositional logic would not support hybrid constraints. Rather, a diagnostic solver should specify the domain and constraint types it supports, the DIF specification would provide a model classification framework.

The rest of this paper is organized as follows. The next section discusses related work. The third section describes the syntax and semantics of DIF. Finally, we propose some initial benchmark problems and discuss future work.

Related Work

We now summarize a selection of formats that have influenced the design of DIF, and some tools of practical consideration.

Interchange Formats

An interchange format provides a standardized, declarative semantics, i.e., the meaning of expressions in the representation can be understood without appeal to an interpreter for manipulating those expressions. Related formats include:

DTIF: The IEEE Digital Test Interchange Format specifies the information content and the data formats for the interchange of digital test program data between digital automated test program generators (DATPGs) and automatic test equipment (ATE) for board-level printed circuit assemblies. This information can be broadly grouped into data that defines the following: UUT Model, Stimulus and Response, Fault Dictionary, and Probe.

Although this is an IEEE standard, it is restricted to digital circuits and test-based diagnostic methodologies.

HSIF: The Hybrid Systems Interchange Format (Pinto *et al.* 2006) is probably the most advanced and most comprehensive format in existence today. It covers arguably the complete range of systems that one may want to diagnose. The main issue is extending this framework to make it more diagnosis-specific.

This framework is focused more on modeling systems, and less on interface specifications for implementing embedded systems.

OSA-CBM: The Open-Systems Architecture for Condition-Based Maintenance interchange format[1] was developed specifically for diagnosis and condition-based maintenance. In addition, it provides code-generators that can be used for creating interfaces for distributed sensors, actuators, and other inference modules.

In comparison to HSIF, this framework is higher-level, as it does not define semantics for equation types (e.g., dynamical equations), or of the transformations among equation types. This framework is focused more on interface specifications for implementing systems, and not on the specifics of modeling.

KIF: The Knowledge Interchange Format is a computer-oriented language for the interchange of knowledge among disparate programs. It has declarative semantics; it is logically comprehensive (i.e., it provides for the expression of arbitrary sentences in the first-order predicate calculus); it provides for the representation of knowledge about the representation of knowledge; it provides for the representation of nonmonotonic reasoning rules; and it provides for the definition of objects, functions, and relations.

XMLBIF: The Bayesian Network XML Interchange Format represents directed acyclic graphs that can be associated to conditional probability measures for discrete variables, with the possibility that decision and utility variables be present in the graph.

PSL: The Process Specification Language defines a vendor- and representation-neutral formalism for manufacturing processes. This may be important for representing life-cycle analysis issues, and not just one-time model specifications. Process data is used throughout the life cycle of a product, from early indications of manufacturing process flagged during design, through process planning, validation, production scheduling and control. In addition, the notion of process also underlies the entire manufacturing cycle, coordinating the workflow within engineering and shop floor manufacturing.

In building our proposal, we have considered a number of specialized formats for representing data structures of interest to MBD. These include BDDs, Petri Nets (Billington & others 2003), Netlists and decomposable NNFs (Darwiche 2001). Furthermore, many digital circuits are expressed in VHDL and Verilog, and we envision tools for translating subsets of these two languages.

[1]Cf. http://osacbm.org/.

Diagnosis Model Auto-Generation

The literature does not contain any work, to our knowledge, that addresses diagnosis model generation for applications other than for circuit diagnosis. The most closely-related work in the literature is the work on diagnostic model generation for circuits (Vogels *et al.* 2004). This work addresses the detailed simulation of IC circuit defects (such as metal spot defects or defects in circuit geometry), which itself if a big task. This methodology is important in that very few other researchers have addressed the need to have libraries of components with detailed physics-based failure-mode definitions. This approach, however, has focused on very small circuits, such as a 4-bit ALU, and does not use algorithms for generating arbitrary circuit topologies. Further, the defect simulation cannot be generalised beyond circuits, as it adopts a number of circuit-specific heuristics and parameters.

Recently, a diagnosis generation methodology based on graphical model generators has been proposed (Provan 2006; Provan & Wang 2007a). This work is aimed at auto-generating models for arbitrary systems, given a library of model components. The proposed methodology first generates a graph representing the system topology, and then assigns system functionality using the component library, inserting a component for each node in the graph describing the system topology.

This approach is significant in that it can capture arbitrary systems. However, it is as accurate as (a) the topology generation mechanism and (b) the component library. Random graph generators can effectively capture the gross topology of complex systems, but much work remains to more precisely capture detailed structure of particular domains. For example, the actual structure of the WWW is known to differ from the predictions of random graph models (Donato *et al.* 2004). In contrast, the practical applications and validity of the circuit-synthesis methods are more heavily-researched than the applications and validity of the random-graph generation approach; as a consequence, the models that a circuit-synthesis method generates are provably closer to the real-world targets (circuits) than are the models generated by random-graph generators are to their real-world targets, such as the WWW (Donato *et al.* 2004). However, many aspects of the circuit-generation algorithms are so particular to the precise architectures of circuits that they are not generalisable to other domains.

Circuit Benchmark Auto-Generation

A second group of related work addresses automatic benchmark circuit generation for improving the design of programmable logic architectures. A considerable literature exists for auto-generating circuits, such as (Stroobandt, Verplaetse, & van Campenhout 1999; Pistorius, Legai, & Minoux 2000; Christie & Stroobandt 2000; Hutton, Rose, & Corneil 2002; Chang, Cong, & Xie 2003; Kundarewich & Rose 2003; Phillips & Hauck 2005; Chan, Congy, & Sze 2005; Holland & Hauck 2006).

Benchmark circuit auto-generation originally was based on applying a circuit generation rule, called Rent's rule (Landman & Russo 1971), but has since expanded to include other methods. Two recent surveys include (Chang, Cong, & Xie 2003; Adya *et al.* 2003).

We now describe the basic methodology of automatic discrete circuit generation, pointing out the similarities and differences to our approach for automating the generation of *diagnostic models* for circuits and other domains.

Most automatic circuit generation methods are based on one of two methods, which we call equivalence-class and Rent-based methods. The equivalence-class methods (Ghosh & Brglez 1999) are based on perturbing a *seed circuit* to generate a circuit with similar overall structure but different local connectivity. The Rent-based methods use a power-law methodology, called Rent's rule, to generate circuits (Christie & Stroobandt 2000). Both methods can generate combinational and sequential circuits, where we define a combinational circuit as one without any distinguished clock inputs (e.g., as provided by D-type Flip-Flop components), and a sequential circuit as a circuit with distinguished clock inputs. Most auto-generation algorithms first create the combinational circuits, and then use a hierarchical approach to generate the sequential circuits for each level of delay (Hutton, Rose, & Corneil 2002).

In the following, we examine the combinational circuit generation process, since this process has some properties that are potentially generalisable to any system model; sequential circuit generation addresses issues that are restricted to a *specific* class of temporal feedback systems with distinguished clock inputs, features that are not present in many other domains. Moreover, because of its greater generality, we focus in this article on the Rent-based combinational circuit methods.

Rent's rule (Landman & Russo 1971), was originally derived empirically, but has since been given mathematical underpinnings. Rent's rule describes the relationship between the number of external signal connections to a logic block (called the number of "pins") and the number of logic gates in the logic block.

Rent's rule is given by:

$$T = tn^{\xi},$$

where (a) T is the number of input/output pins,[2] (b) n is the number of gates, (c) and the (internal) Rent exponent $0 \leq \xi \leq 1$ represents the level of placement optimization within a statistically homogeneous circuit, which is characterized by an interconnection topology with an average node degree t (or in engineering terms, t terminals per gate). From an engineering perspective, $\xi = 1$ corresponds to no placement optimization, i.e., the circuit is interpreted as a random gate arrangement. In actual circuits, the parameter ξ is dependent on circuit-topology: microprocessors, gate-arrays, and high-speed computers are characterized by Rent exponents of $\xi = 0.45, 0.5$, and 0.63, respectively (Christie & Stroobandt 2000).

Several tools have been developed to generate benchmark circuits based on Rent's rule and other approaches. Examples of such tools are CIRC and GEN[3]. These tools can be integrated within the diagnostic model-generation framework described in this article.

Circuit generation algorithms have proven very useful for applications like FPGA design; however, they are restricted to a specific domain, and focus on *topology optimisation*, rather than on the issues of fault isolation that are relevant to diagnosis benchmarks. As a consequence, we have developed a more general approach to benchmark generation that has some commonality with circuit generation algorithms, but also some key differences.

The key commonality between our approach and these circuit generation algorithms is that we first generate the underlying system topology, using a graph generation algorithm. Specifically, we use a graph generation algorithm that generates a graph with a power-law topology. This approach is a generalisation of the Rent-based topology algorithm, in that Rent's rule uses a power-law method that is almost identical to the power-law approaches developed within the random-graph community.

Both the Rent-based and random-graph methods focus on defining a graphical structure $G(V, E)$ in which the nodes V correspond to components and the edges E correspond to wires between the components.

Key differences between these areas include (1) the extension of the system topology to incorporate functionality, and (2) the tuning of the topology and functionality. With regard to (1), our diagnostic benchmark generator extends the system topology to incorporate a functional description that describes both normal and anomalous system behaviours. With regard to (2), the diagnostic benchmark generator methodology has parameters that can be tuned to generate models to approximate particular domains, but assumes that these parameters are domain-dependent and need to be supplied by domain experts. In the absence of good domain parameters, the generated models will approximate real models with good accuracy, the quality of which can be improved with the use of precise parameters.

Diagnosis Interchange Format

The DIF supports representation of models, observation vectors and diagnoses. Modeling semantics is a topic of research and it is unlikely that our proposal would accommodate all the different approaches, especially when considering hybridization with continuous systems and time and state. In designing DIF, our main goals have been simplicity, translatability from existing implementation formats and extensibility.

Syntax and Semantics of DIF

As the standards described in this paper do not imply voluminous data according to current computing standards, and all the formats expose an ample amount of structure, our

[2] In graph-theoretic terms, if we represent component i using a node in a topology graph, then the degree k_i of component i corresponds to the set of terminals of component i in the circuit-generation domain.

[3] See http://www.eecg.toronto.edu/∼jayar/software/.

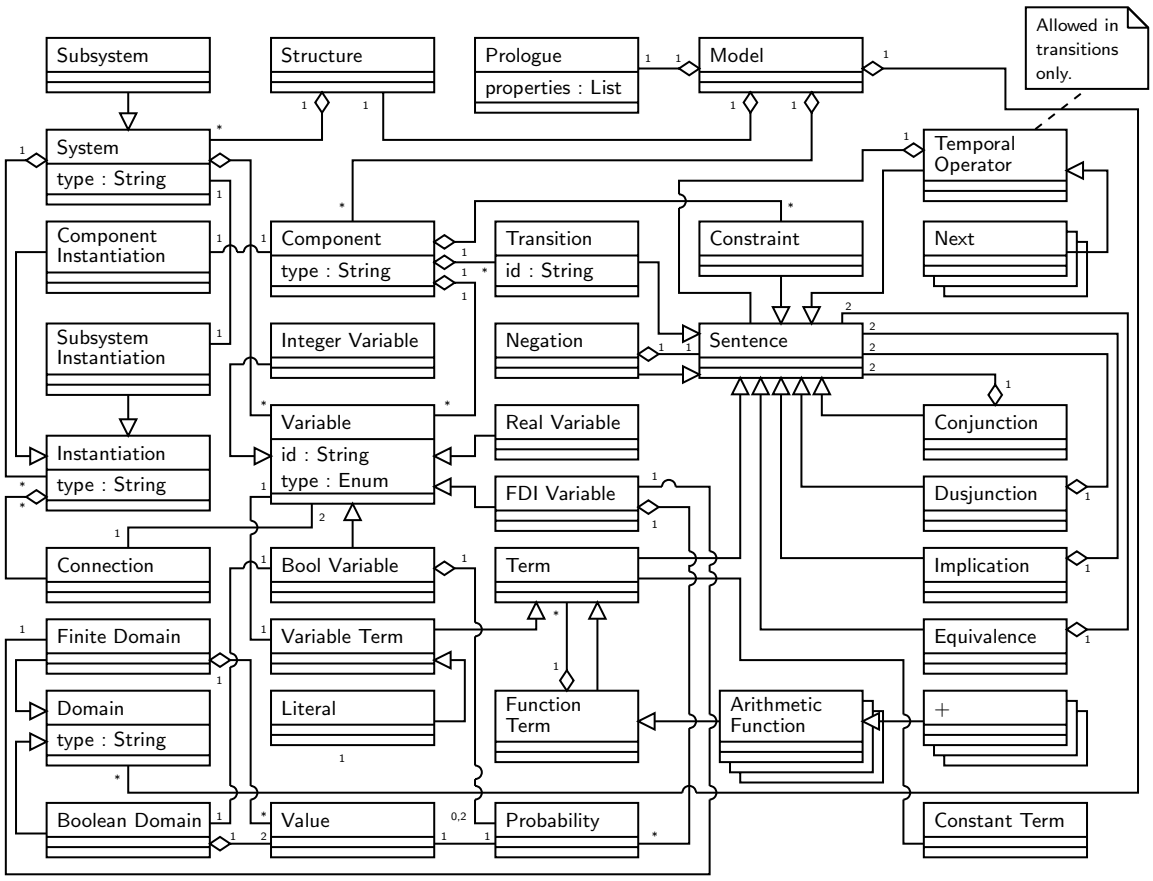

Figure 1: A visual representation of the DIF 1.0 model syntax (function classification, individual functions and some of the variable attributes are omitted).

benchmark is encoded using eXtensible Markup Language (XML) (Bray *et al*. 2006). The latter choice greatly simplifies the syntactical validation and representation, and allows the user to borrow from the vast amount of XML tooling. The DIF XML schema is visualized in Figure 1 and its full specification is available for download from (obscured for anonymity).

A DIF model has four sections: prologue, domains, structure, and components. The prologue describes the main characteristics of the model, i.e., the types of the constraints, fault-modeling, etc. The structure displays the model hierarchy that is essential for many of the MBD algorithms existing today. The domain description specifies symbolic values for all the Finite Domain Integer (FDI) variables (Booleans are treated as a special case of many-valued logic). Finally, for each component a set of constraints and transitions are specified. In particular, DIF supports constraints ranging from logic to differential equations.

DIF supports any First-Order Logic (FOL) sentence as a constraint, hence it provides for a large range of modeling techniques. The variables in a component or a subsystem, except Boolean or FDI can be in the real or (infinite) integer domains. For the latter two variable types, it is not necessary to specify domains. The framework is suitable for both abductive and consistency-based diagnosis.

Next, we discuss the DIF representation of some models.

A Combinatorial Circuit Example Expressing structure is important, both from the algorithmic and modeling perspective. A DIF model typically specifies a number of hierarchical systems and a set of components. A system, then, can *instantiate* an arbitrary number of subsystems and components (cyclic instantiations are forbidden). Each system also defines a set of variables and how these variables are mapped into the systems and components it references. The use of hierarchy is best illustrated with the circuit shown in Figure 2.

The circuit is a full adder, each gate of which is allowed to fail without specifying faulty behavior. It consists of two half adders and an OR gate. The XML element in Figure 3 represents the structure in DIF (the variable mappings are omitted from the example for brevity). The top level system instantiates two copies of the half adder subsystem and an OR logic gate. The half adder uses an XOR and an AND gate. Note, that in the actual model the top-level system has some internal variables (f, p, and q) representing the wire connections.

The structure of a model is, essentially, a rooted, edge-labeled multidigraph $G = \langle V, E \rangle$, where the set of nodes

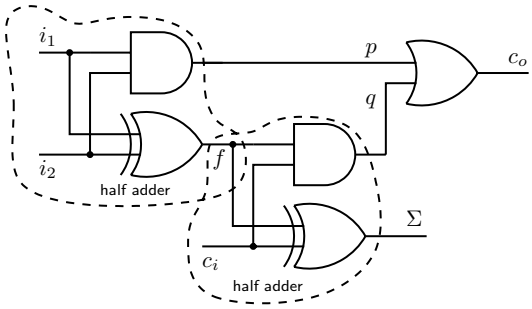

Figure 2: A full adder circuit.

```
<model>
    ⋮
  <structure>
    <system type="fullAdder">
      <subsysInst type="halfAdder" id="ha1" />
      <subsysInst type="halfAdder" id="ha2" />
      <compInstance type="orGate" id="orGate" />
    </system>
    <subsystem type="halfAdder">
      <compInst type="xorGate" id="xorGate" />
      <compInst type="andGate" id="andGate" />
    </subsystem>
  </structure>
```

Figure 3: Structure describing element of a full adder circuit.

V consists of all systems and components and there is an edge in E for each instantiation. One of the nodes is distinguished as a root (top-level) system. Constructing an algorithm which converts the hierarchical representation into a "flat" one is straightforward, by recursively merging the nodes of G.

A component model is given as a set of multi-valued propositional **Wff** over a set of variables V. A multi-valued variable $v_i \in V$ takes a value from a finite domain, which is a set of symbols (with 1-to-1 mapping to $\mathbb{Z}^+$) $D_i = \{s_1, s_2, \ldots, s_m\}$. All the domains of FDI variables have to be explicitly specified in the second section of a DIF model. In the combinatorial circuit used for illustration, all variables are in the Boolean domain and the DIF definition of the latter is shown in Figure 4.

```
<domains>
  <booleanDomain type="bool">
    <value default="true">true</value>
    <value>false</value>
  </booleanDomain>
</domains>
```

Figure 4: A DIF specification of the Boolean domain.

A positive multi-valued literal l_j^+ is a Boolean function $l_j^+ \equiv (v_i = d_k)$, where $v_i \in V$ and $d_k \in D_i$. A negative multi-valued literal l_j^- is defined in a similar fashion. A multi-valued propositional **Wff**, then, is a formula over the multi-valued literals $l_1, l_2, \ldots, l_n$, and the standard Boolean connectives: $\neg$ (negation), $\Leftrightarrow$ (equivalence), $\Rightarrow$ (implication), $\wedge$ (conjunction), and $\vee$ (disjunction).

```
<component type="xorGate">
  <var domain="bool" id="h" type="health"/>
  <var domain="bool" id="o"/>
  <var domain="bool" id="i1"/>
  <var domain="bool" id="i2"/>
  <constraint>
    <imply><lit id="h"/>
      <equiv><lit id="o"/>
        <equiv><lit id="i1"/>
          <not><lit id="i2"/></not>
        </equiv>
      </equiv>
    </imply>
  </constraint>
</component>
```

Figure 5: A weak-fault model of an XOR logic gate.

Figure 5 shows the DIF specification of an XOR gate which is a part of the sample full adder circuit, introduces above. The component defines four variables: h, o, i_1, and i_2, of which h is assumable. The single constraint specifies the propositional **Wff** $h \Rightarrow (o \Leftrightarrow (i_1 \Leftrightarrow \neg i_2))$, the interpretation of which stipulates that the component health variable h is true iff the output o is true only when the values of i_1 and i_2 are different. The model of the component is weak-fault, i.e., if all the n components in a model are constrained by expressions in the form $h_i \Rightarrow F_i, 1 \leq i \leq n$, where h_i is an assumable and does not appear in any of the propositional **Wff** $F_j, 1 \leq i \leq n$, then the Minimal Diagnosis Hypothesis (de Kleer, Mackworth, & Reiter 1992) holds.

Reasoning about time and state is central to MBR. Our discussion continues with some ways to represent dynamic characteristics of systems in DIF.

Models with State DIF allows every component to define temporal constraints. The only assumption is that a diagnostic reasoner would maintain discrete time with limited horizon, and at every time step, it would copy all the variables and all the constraints for the current time instance.

A temporal constraint is a sentence in FOL that has variables from two instantiation of the system description in time. The only difference between a temporal constraint and a combinatorial constraint is that a temporal constraint allows the use of a "temporal operator".

An example of such a temporal operator is $\bigcirc$ (next) (the others include $\Box$ (globally), $\Diamond$ (eventually)) with semantics similar to the one in (Manna & Pnueli 1992). Note that $\bigcirc$ can only appear in front of a variable term, in which case $\neg \bigcirc x$ is equivalent to $\bigcirc \neg x$.

Our second example clarifies the DIF temporal semantics by discussing a model of a resettable pneumatic valve with

stateful behavior. The state transition diagram of this valve is shown in Figure 6.

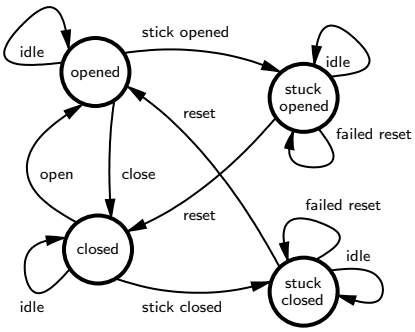

Figure 6: A state transition diagram of a resettable valve.

The XML element, shown in Figure 7, represents one of the possible transitions from Figure 6. It says that, given a positive assignment to the "reset" variable in the current instance, and if the valve is stuck open, it will change its state to closed when time progresses.

```
<transition id="reset">
  <constraint><lit id="c">reset</lit></constraint>
  <constraint>
    <imply><lit id="s">stuckOpened</lit>
      <next><lit id="s">closed</lit></next>
    </imply>
  </constraint>
  <constraint>
    <imply><lit id="s">stuckClosed</lit>
      <next><lit id="s">opened</lit></next>
    </imply>
  </constraint>
</transition>
```

Figure 7: Valve transition from state "stuck closed" to "open" upon reset.

It is not possible to show all transitions of the valve in DIF as the full model specifies a transition for every edge of the graph shown in Figure 6. These constraints, however, are very similar to the one we have discussed in this section.

Before we continue with an example having an ODE for a constraint, it worth to note that transitions are nothing more than set of constraints, having a name and applied by the reasoners progressively as the time expands.

Hybrid Systems Consider a numeric model of the primitive water clock, shown in Figure 8. The water level h (which has been used in ancient times for approximating the time of the day) in time t is the solution of the ODE specified next to the figure. It is possible to build a fault model which specifies that the component is healthy if the predicted value of h, $\hat{h}$, according to the numerical solution of the ODE, is within a certain threshold δ, or $|h - \hat{h}| > \delta$.
The rest of the parameters in Figure 8 are as follows. A_w and

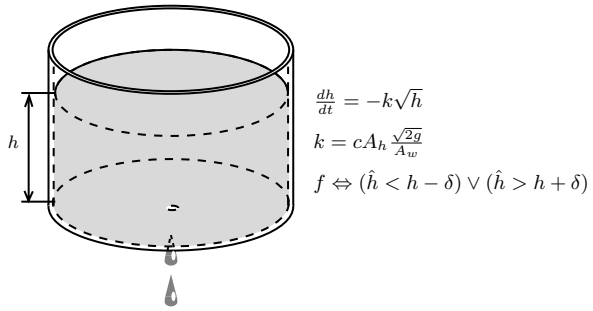

Figure 8: A set of hybrid constraints of a water clock fault model.

A_h are the cross-sectional areas of the water and the whole, respectively, and c is a friction constant. The gravity acceleration is denoted as g. The full model uses the Boolean fault variable f and an observable variable $\hat{h}$ in the real domain.

The DIF constraint, shown in Figure 9 specifies an ODE with t as the independent variable and both t and h in the continuous domain (both have type "real" in the variable declaration section of the full water clock model).

```
<equiv>
  <fder><var id="h" /><var id="t" /></fder>
  <uminus>
    <prod>
      <var id="k" /><sqrt><var id="h" /></sqrt>
    </prod>
  </uminus>
</equiv>
```

Figure 9: An ODE constraint for the water clock shown in Figure 8.

Having discussed DIF in specifying a wide range of models, we can continue with the remaining two data structure which are part of an MBD problem: observations and diagnoses.

Representation of Observations In addition to models, an MBD format should specify syntax and semantics for observation vectors (sensor data). The semantics of an observation vector in DIF is illustrated in Figure 10. An observation vector in DIF is always a *conjunction* of variable assignments.

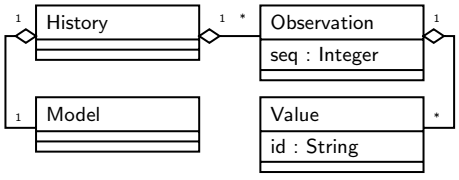

Figure 10: A visual representation of the DIF 1.0 observations syntax.

Figure 11 shows the DIF representation of a sample obser-

vation vector for the full-adder from Figure 2 (representing the propositional **Wff** $\mathrm{OBS}_2 = i_1 \wedge i_2 \wedge c_i \wedge \neg\Sigma \wedge c_o$). An observation refers to a specific instant in time. For the main MBD problem, a reasoner is supplied with a model in DIF and a sequence of observations in time, and it computes diagnoses, the format of which will be described later.

```
<obs seq="2">
  <lit id="i1"/><lit id="i2"/><lit id="ci"/>
  <not><lit id="sum"/></not><lit id="carry"/>
</obs>
```

Figure 11: An observation of the full adder from Figure 2.

As the problem of finding a kernel diagnosis is know to be Π_2^{P}-complete, most of the MBD implementations employ a *heuristic* based on the minimum number of failing components, or smallest probability failure mass (the DIF model language has a straightforward way for assigning probabilities to variable values). Hence the goal of an MBD benchmark is to provide such an observation which leads to a *kernel diagnosis of minimum cardinality having the maximum number of failing components*. The latter problem is a topic on its own with many applications in MBR.

Representation of Diagnoses The last part of our specification concerns the diagnoses as computed by MBD implementations. As both the observation and the diagnoses are sets of variable assignments, their formats are very similar. The class diagram for the DIF diagnosis representation is shown in Figure 12.

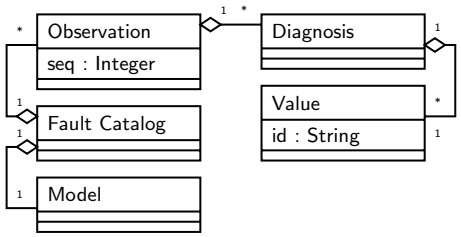

Figure 12: A visual representation of the DIF 1.0 diagnoses syntax.

A diagnosis file specifies zero or more diagnoses for some or all of the time instances at which observations have been performed. The observation of the full-adder (cf. Figure 2) from Figure 11 has been performed at time instance 2 according to its *seq* attribute. All kernel diagnoses of this observation are shown in Figure 13.
We use the dotted notation in the literal identifiers to specify the subsystems in which the component resides. The two possible kernel diagnoses shown in Figure 13 are each of the two XOR gates of the full-adder being faulty.

Conclusion

This paper addresses two major problems. First, it suggests a format for model exchange, observation vectors and diagnoses in MBD. The paper considers a wide spectrum of modeling techniques, and the implementation of its suggestion would allow exchange of component libraries, models, sensor data and facilitate cross-validation of diagnostic results.

```
<obs seq="2">
  <diagnosis>
    <not><lit id="ha1.xorGate.h"/></not>
  </diagnosis>
  <diagnosis>
    <not><lit id="ha2.xorGate.h"/></not>
  </diagnosis>
</obs>
```

Figure 13: An diagnosis of the full adder from Figure 2, given the observation from Figure 11.

The second problem we address, is finding a compact representation for existing models. By using inheritance and specialization, we show how a very broad modeling language like DIF can be specialized to represent explicit models such as digital circuits. This specialization decreases the modeling complexity and allows modelers to use DIF for fault-diagnosis of Boolean circuits, for example, while not being burdened with the language's expressiveness outside the domain of propositional logic.

While DIF may be compact for representing a wide range of systems, MBD employs a variety of representations allowing trade-offs in time, space, and off-line time (i.e., knowledge compilation approaches). A future extension of this standard would benefit from supporting compact compiled representations like NNF (Negation Normal Forms), OBDD (Ordered Binary Decision Diagrams) and others.

Acknowledgments

This work has been supported by STW grant DES.7015 and SFI grant 04/IN3/I524.

References

Adya, S. N.; Yildiz, M. C.; Markov, I. L.; Villarrubia, P. G.; Parakh, P. N.; and Madden, P. H. 2003. Benchmarking for large-scale placement and beyond. In *Proc. ISPD'03*, 95–103.

Benazera, E.; Travé-Massuyès, L.; and Dague, P. 2002. State tracking of uncertain hybrid concurrent systems. In *Proc. DX'02*, 106–114.

Billington, J., et al. 2003. The Petri net markup language: Concepts, technology, and tools.

Bray, T.; Paoli, J.; Sperberg-McQueen, C. M.; Maler, E.; and Yergeau, F. 2006. Extensible markup language (XML) 1.0. Technical Report REC-xml-20060816, W3C.

Brglez, F., and Fujiwara, H. 1985. A neutral netlist of 10 combinational benchmark circuits and a target translator in fortran. In *Proc. ISCAS'85*, 695–698.

Carlson, B., and Gupta, V. 1998. Hybrid cc with interval constraints. In *Proc. HSCC'98*, 80–95.

Chan, T.; Congy, J.; and Sze, K. 2005. Multilevel generalized force-directed method for circuit placement. In *Proc. ISPD'05*, 185–192.

Chang, C.-C.; Cong, J.; and Xie, M. 2003. Optimality and scalability study of existing placement algorithms. In *Proc. DAC'03*, 621–627.

Christie, P., and Stroobandt, D. 2000. The interpretation and application of Rent's rule. *IEEE Trans. Very Large Scale Integr. Syst.* 8(6):639–648.

Darwiche, A. 2001. Decomposable negation normal form. *Journal of the ACM* 48(4):608–647.

de Kleer, J.; Mackworth, A.; and Reiter, R. 1992. Characterizing diagnoses and systems. *Artificial Intelligence* 56(2-3):197–222.

Donato, D.; Laura, L.; Leonardi, S.; and Millozzi, S. 2004. Simulating the webgraph: A comparative analysis of models. *Computing in Science and Engineering* 6(6):84–89.

Eiter, T., and Gottlob, G. 1995. The complexity of logic-based abduction. *Journal of the ACM* 42(1):3–42.

Ghosh, D., and Brglez, F. 1999. Equivalence classes of circuit mutants for experimental design. In *Proc. ISCAS'99*, 432–435.

Hansen, M.; Yalcin, H.; and Hayes, J. 1999. Unveiling the ISCAS-85 benchmarks: A case study in reverse engineering. *IEEE Design & Test* 16(3):72–80.

Holland, M., and Hauck, S. 2006. Improving performance and robustness of domain-specific CPLDs. In *Proc. FPGA'06*, 50–59.

Hutton, M. D.; Rose, J.; and Corneil, D. G. 2002. Automatic generation of synthetic sequential benchmark circuits. *IEEE Trans. on CAD of Integrated Circuits and Systems* 21(8):928–940.

Kundarewich, P. D., and Rose, J. 2003. Synthetic circuit generation using clustering and iteration. In *Proc. FPGA'03*, 245–245.

Landman, B. S., and Russo, R. L. 1971. On pin versus block relationship for partitions of logic circuits. *IEEE Trans. Computers* 20(6):1469–1479.

Manna, Z., and Pnueli, A. 1992. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer-Verlag.

Phillips, S., and Hauck, S. 2005. Automating the layout of reconfigurable subsystems using circuit generators. In *Proc. FCCM'05*, 203–212.

Pietersma, J.; Feldman, A.; and van Gemund, A. 2006. Modeling and compilation aspects of fault diagnosis complexity. In *Proceedings of IEEE AUTOTESTCON-06*.

Pinto, A.; Carloni, L. P.; Passerone, R.; and Sangiovanni-Vincentelli, A. 2006. Interchange formats for hybrid systems: Abstract semantics. In *Proc. Hybrid Systems: Computation and Control*, 491–506.

Pistorius, J.; Legai, E.; and Minoux, M. 2000. Partgen: a generator of very large circuits to benchmark thepartitioning of FPGAs. *IEEE Trans. on CAD of Integrated Circuits and Systems* 19(11):1314–1321.

Provan, G., and Wang, J. 2007a. Evaluating the adequacy of automated benchmark model generators for model-based diagnostic inference. In *Proc. of IJCAI'07*, 99–104.

Provan, G., and Wang, J. 2007b. Automated benchmark model generators for model-based diagnostic inference. In *Proc. IJCAI'07*, 513–518.

Provan, G. 2006. Automated benchmark model generators for model-based diagnostic inference. In *Proc. DX'06*, 99–104.

Stroobandt, D.; Verplaetse, P.; and van Campenhout, J. 1999. Towards synthetic benchmark circuits for evaluating timing-driven cad tools. In *Proc. ISPD'99*, 60–66.

Vogels, T.; Zanon, T.; Desineni, R.; Blanton, R.; Maly, W. Brown, J.; Nelson, J.; Fei, Y.; Huang, X.; Gopalakrishnan, P.; Mishra, M.; Rovner, V.; and Tiwary, S. 2004. Benchmarking diagnosis algorithms with a diverse set of ic deformations. In *Proc. ITC'04*, 508–517.

Williams, B., and Nayak, P. P. 1996. A model-based approach to reactive self-configuring systems. In *Proc. AAAI'96*, 971–978.