A Framework for Integrating Requirements-Based Design and Diagnosis

Gregory Provan
Department of Computer Science, University College Cork, Cork, Ireland g.provan@cs.ucc.ie

ABSTRACT

Two key impediments for the commercial success of model-based diagnosis (MBD) include (a) a failure to integrate diagnostics modeling within the requirements and design phase, and (b) a high degree of diagnostic ambiguity during run-time. This article addresses both of these impediments by providing a formal framework that integrates requirements-based design with MBD modeling. The proposed framework extends the consistency-based theory of MBD with a requirements-based design theory based on contracts.

1 INTRODUCTION

While the field of Model-Based Diagnosis (MBD) has made significant progress in formalizing the process of diagnosis and developing algorithms to diagnose a range of systems, these technologies have not been readily adopted in many real-world applications. Two key impediments are a failure to integrate diagnostics modeling within the requirements and design phase, and issues with diagnostic ambiguity during run-time.

One impediment, the failure to integrate the development of embedded MBD code within the requirements and design process, is often due to a lack of (design-phase) component libraries for cheaply and efficiently building MBD models. Leveraging design information in the development of MBD component libraries can reduce the modeling burden. A second impediment, diagnostic ambiguity, arises when a diagnosis algorithm given an observation α and a diagnosis model Ψ_D, returns a disjunction of possible faults, e.g., component X, or component Y, or both, are faulty. It may turn out that only component X is faulty, as deduced given an additional observation.

This article addresses both of these impediments by providing a formal framework that integrates component-based design with MBD modeling. We present a formal framework that extends the consistency-based theory of MBD (Reiter, 1987) with a component-based requirements/design theory based on contracts (Martin and Lamport, 1993). This assume/guarantee theory defines a system Φ in terms of an inter-connected collection of "rich" components (Benveniste *et al.*, 2008), each of which must fulfill a contract (e.g., based on design requirements) given assumptions in which the component operates. Given a contract-based specification for Φ, one can prove properties about fulfillment of the design requirements. Contracts have been used for hardware design optimization (Sun *et al.*, 2009), and also for software analysis during run-time (Meyer *et al.*, 2009). Moreover, based on observations and the possibility of stochastic (or non-deterministic faults), one can then diagnose the reasons for the contracts violated during operation of Φ (Slâtten, 2010; Zulkernine and Seviora, 2005).

This approach offers a formal methodology not only to integrate requirements specification within diagnostics models, but also to significantly reduce the incidence of two challenging classes of ambiguous or "spurious" fault, commonly known as No-Fault-Found (NFF) and cascaded fault-report. During run-time, many ambiguous diagnoses can arise due to inability to define models that adequately distinguish "local" faults from exogenous influences. For example, the No-Fault-Found is a common diagnosis that causes problems in many domains, particularly avionics: it is a fault that is isolated during device operation, but when the "faulty" component is replaced, the fault cannot be duplicated during testing of the component. In many cases, this fault occurs when the component is operated outside of its design intent. For example, fighter jets have many actuator faults that occur when the jets are operated outside of design specifications.

Cascaded faults are another difficult situation that arise in typical FDD situations: in avionics, for example, an upstream module will compute some faulty data, and then all downstream modules that process this faulty data will issue (erroneous) fault reports, when in fact downstream modules do not have hardware faults, but issue fault reports due to the incoming corrupted data. In this case, the failure to identify exogenous anomalies properly leads to many ambiguous diagnoses.

Assume-guarantee reasoning considers components

not in isolation, but in conjunction with assumptions about their context. Hence, the assume/guarantee (A/G) approach focuses on reasoning about a component in terms of the assumptions about its environment, and by proving that these assumptions are satisfied by the environment, establishing a set of *system obligations*, the contract. This approach has been use for (a) validating the requirements of a design (and thereby reducing the design-space that must be searched during design optimization (Sun *et al.*, 2009)), and (b) during run-time for system-level verification (Giese *et al.*, 2010).

In our approach, we specify two main fault classes: hard and soft. Hard faults denote failures that occur when components are operated inside design specifications; soft faults denote failures that occur when components are operated outside design specifications. This classification has two important outcomes. First, it will probably significantly reduce NFF incidence, since most of these will be classified as soft faults (and hence treated differently than hard faults). Second, it will close the feedback loop between design and operation of a device; if many soft faults occur, this indicates that the actual operation of a device differs from the design, and hence the design must be changed, or the fault isolation code changed to align to the actual usage of the device.

The contributions of the article are as follows:

- We generalize the consistency-based theory of MBD to a contract-based theory that enables design models, with their environment-based requirements, to be integrated with an MBD model.
- We show how we can use the existing MBD inference to compute not only faults, but also operating-condition violations, and thereby rule out faults based on incorrect component inputs.

2 RUNNING EXAMPLE: TO/GA SYSTEM

2.1 Example

This section introduces a simple example that we use throughout the article. The Take-off/Go Around (TO/GA) system is an autopilot sub-system that activates take-off or go-around thrust. During take-off, pressing the TO/GA switch causes the engines to increase their RPM to their computed take off power, as computed from parameters such as runway length, wind speed, temperature, and the weight of the aircraft. The go-around mode is engaged on approach to land, and switches the plane from autopilot approach mode by engaging the thrust levers until they reach the position go-around thrust.

Most commercial aircraft use some form of hardware/software redundancy to ensure high reliability of autopilot systems. For example, this may be a dual-redundant or a triply-redundant approach, as in the Boeing 777 aircraft's TO/GA architecture. In this article we study a TO/GA System with a dual-dual redundant approach, as shown in Figure 1. In such systems, the TO/GA commands are replicated and sent to two autopilot flight director control (AFDC) computers, which compute thrust levels in each of the AFDC computers. The AFDC outputs are sent to the engines, and any anomalies are sent to fault-report monitors. Each TO/GA signal is tagged with a time stamp, to ensure that the signals being compared are closely-spaced temporally and thus represent the same computation done in different downstream components.

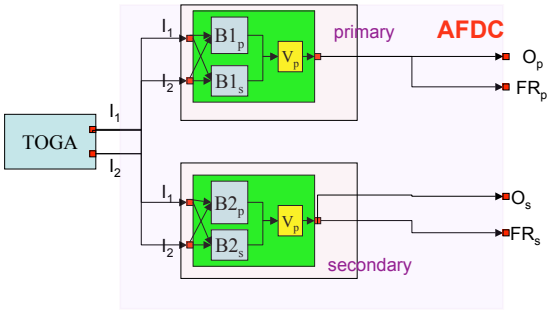

Figure 1: Dual-dual autopilot TOGA sub-system, with TO/GA signals I_1, I_2 sent to primary and secondary AFDC computers.

The AFDC has primary and secondary computers; the primary AFDC is engaged as long as no possible data corruption is detected. If a signal mis-compare occurs, the primary AFDC issues a fault report and the secondary AFDC is also engaged. If the secondary AFDC does not detect a mis-compare, it is now used as the primary unit. If the secondary AFDC also detects a mis-compare, it also issues a fault report and a pilot-warning, which notifies the cockpit of TO/GA problems, with a recommendation to switch to manual TO/GA procedures.

Environment-Based Requirements Specification

This section defines three TOGA system signal requirements as propositions (R_1, R_2, R_3), in order to fit in with the MBD language. The requirements for the AFDC are that it must test signal equality for two asynchronous signals (R_1), which must be generated within a time difference μ no greater than a fixed constant δ (R_2), such that the comparison test must be carried out with reliability > 0.9999 (R_3). In this article we focus on environmental requirements R_1 and R_2, which we encode as assumptions on the inputs for the AFDC. In future work we will extend this framework to cover stochastic properties of requirements within an assume/guarantee framework, e.g., R_3. We formalise the first two requirements as follows:

R_1 the time-difference between the AFDC input signals I_1 and I_2 must be such that $|\tau_{I_1} - \tau_{I_2}| < \delta$, when $\mu = t$; else $\mu = f$;

R_2 if the TOGA outputs are both t, set the input flag $I = t$; else $I = f$. This is given by $(I_1 \cap I_2) \Leftrightarrow I$.

Hence, for this sub-system, we can define the requirements specification as $\mathcal{R} = \mu \wedge I$. We assume that the requirements are consistent.

We can identify a few key states for this system, which we list in Table 1. A typical "run" of this system will consist of a sequence of states. For example, consider a state sequence $S = \{\sigma_1, \sigma_2, \sigma_3, \sigma_3, \sigma_3, \sigma_4\}$, as shown in Table 1. We can classify states as satisfying

	TO/GA output	μ	O_p	O_s
σ_0	(t,-)	-	-	-
σ_1	(t,f)	f	f	f
σ_2	(f,t)	f	f	f
σ_3	(t,t)	f	f	f
σ_4	(t,t)	t	t	t
σ_5	(t,t)	t	f	t
σ_6	(t,t)	t	f	f
σ_7	(f,f)	t	t	t

Table 1: Set of states for dual-dual comparator, with inputs, time-difference μ for inputs, and outputs O_p and O_s

the requirements or not. For example, if we examine the input-equality and timing requirements for S, we see that σ_4 through σ_7 satisfy these requirements, and the other states do not.

3 PRELIMINARIES AND NOTATION

This section introduces our notation for components, and for diagnosis and assume/guarantee models.

3.1 Component-Based Modeling

A component is a hierarchical entity that represents a unit of design. We measure the dynamic behaviours of components in terms of a trace, which is defined over states, or assignments to component variables, as follows.

Definition 1 *Given an initial state σ_0, a behavior (trace) γ of a component ϕ is a sequence of states $\{\sigma_1, \sigma_2, ..., \sigma_m\}$.*

$\Gamma \subseteq \Sigma^*$ denotes the set of all traces possible given a set Σ of events.

A component is formalized as follows:

Definition 2 *A component ϕ is defined over the tuple $(V_\phi, P_\phi, \Psi_\phi, \Gamma_\phi)$, where*

- *V_ϕ is the set of variables defined in ϕ, where V is comprised of inputs $\mathcal{U}$, outputs $\mathcal{Y}$, internal variables V_{int}, failure-mode variables $COMPS$ and assumption-violation variables $\mathcal{Z}$;*
- *P_ϕ is the set of ports, where $P = \mathcal{U} \cup \mathcal{Y}$;*
- *Ψ_ϕ is the behavioural model for ϕ, such that $y = \Psi_\phi(u, V_{int}, COMPS)$ for $y \in \mathcal{Y}$ and $u \in \mathcal{U}$; and*
- *Γ_ϕ is the set of traces over the input and output ports of ϕ.*

Components are connected together to form a system by connecting output ports of one component to input ports of another component, equating the values of the corresponding ports and variables.

3.2 Model-Based Diagnosis Notation

We represent a diagnosis model for an artifact as a propositional formula over a set V of variables (Reiter, 1987).

Definition 3 (Diagnostic System) *A diagnostic system Φ is defined as the triple $\Phi = \langle \Psi, COMPS, OBS \rangle$, where Ψ is a propositional theory over a set of variables V, $COMPS \subseteq V$, $OBS \subseteq V$, $COMPS$ is the set of component failure-mode variables, and OBS is the set of observable variables.*

Throughout this paper we will assume that $\Psi \not\models \bot$. The traditional query in MBD computes terms of failure-mode variables which are explanations for the system description and an observation. We represent a diagnosis in terms of a health assignment.

Definition 4 (Health Assignment) *Given a diagnostic system $\Phi = \langle \Psi, COMPS, OBS \rangle$, an assignment $\mathcal{H}$ to all variables in $COMPS$ is defined as a health assignment.*

A health assignment $\mathcal{H}$ can be viewed logically as a conjunction of propositional literals. A diagnosis is defined as follows.

Definition 5 (Diagnosis) *Given a diagnostic system $\Phi = \langle \Psi, COMPS, OBS \rangle$, an observation α over some variables in OBS, and a health assignment ω, ω is a diagnosis iff $\Psi \wedge \alpha \wedge \omega \not\models \bot$.*

Models in this MBD framework have the general structure of $\mathcal{H} \Leftrightarrow \xi_i$; in other words, behavior models $\xi_i \in \xi$ are contingent on a health assignment. A Weak-Fault Model (de Kleer *et al.*, 1992) defines all failure-mode variables with the values {OK, bad}, i.e., it only specifies nominal behaviours. A Strong-Fault Model (Struss and Dressler, 1989) defines non-nominal failure-mode values, together with faulty behaviours.

Diagnostic inference is necessary if an anomalous observation is recorded, and diagnoses are isolated according to Definition 5. In the theory of diagnosis of (Reiter, 1987), systems and observations are assumed to be atemporal.

3.3 Contract-Based Design Notation

The A/G model is general enough to address a target system defined as a set of asynchronous finite state machines (FSMs) (Benveniste *et al.*, 2008) or hybrid automata (Giese *et al.*, 2010). Using this framework, the run-time model for a component is defined in terms of the behavior of that component, which in turn is represented by a FSM plus a set of incoming and outgoing ports. A FSM may contain variables storing the addresses (i.e., port IDs) of incoming and outgoing messages. This model is compatible with the Tagged-Signal model (Lee and Sangiovanni-Vincentelli, 1998), which can express behaviors for a wide range of models of computation.

The environment of each component specified in a FSM is assumed to be the other components and the system environment. The A/G approach of verification and/or diagnosis separates assumptions and guarantees in the component FSM specifications, and then discharges the assumptions based on the guarantees of the other components or of the system environment.

The notion of contracts is typically defined for systems with dynamics. Here, a run-time characterization of a component ϕ consists of a set of behaviors defined over the ports of ϕ.

Definition 6 *An implementation $\mathcal{M}$ is an instantiation of a component ϕ, and consists of a set P_ϕ of ports, V_ϕ of variables, and of a set $\mathcal{M}$ of behaviors, or runs, which assign a history of "values" to ports.*

A component may include both implementations and contracts. An assertion $E = (V, \Gamma_E)$ is a property that may or may not be satisfied by a behavior.

An assume/guarantee specification is derived from the given state information of the component instead of the predicates on the state of a component. Assertions are different from specification preconditions and post–conditions, as defined in areas like AI planning or program analysis. Pre-/post-conditions typically constrain the state space of a program at a particular point. Instead, assertions consist of properties of entire behaviors, and therefore include the dynamics of a component ϕ. We can define a relation between assertions and pre-/post-conditions by abstracting a finite trace executing a particular sequence of transitions using a pre-condition predicate p or q. In the MBD framework we adopt here, we need a static abstraction closer to a predicate. In contrast, the theory of contracts uses a dynamic notion of assertion satisfaction, as follows:

Definition 7 *An implementation $\mathcal{M}$ satisfies an assertion E whenever they are defined over the same set of ports and all the behaviors of $\mathcal{M}$ satisfy the assertion, i.e., when $\mathcal{M} \subseteq E$.*

The assume/guarantee paradigm separates the system specification into two parts, $G = (V, \Gamma_G)$ and $A = (V, \Gamma_A)$: G defines the guaranteed behavior of a component, and A defines the assumed behavior of the environment of the component or system. This implies conditional specification for a component ϕ_j, usually written as $\{A_j\}\phi_j\{G_j\}$. If A_j is satisfied by the environment then ϕ_j will satisfy G_j. By combining the set of assume/guarantee behavior pairs (A_j, G_j) in an appropriate manner for all ϕ_j, it is possible to prove the correctness of the whole system (Benveniste *et al.*, 2008).

We now formally define the notion of a contract for a component as a pair of assertions, which express its assumptions and guarantees.

Definition 8 *A contract C is denoted as a tuple (V, A, G), where V is the variable set, A is the assumption, and G is the guarantee, an assertion on the behaviors of a component subject to A.*

Definition 9 *An implementation of a component satisfies a contract whenever it satisfies its guarantee, subject to the assumption. Formally, $\mathcal{M} \cap A \subseteq G$, where $\mathcal{M}$ and C have the same ports. We write $\mathcal{M} \models C$ when $\mathcal{M}$ satisfies a contract C.*

We can check satisfaction of A/G models using the following equivalent formulas, where $\neg A$ denotes the set of all runs that are not runs of A:

$$\mathcal{M} \models C \;\Leftrightarrow\; \mathcal{M} \subseteq G \cup \neg A \;\Leftrightarrow\; \mathcal{M} \cap (A \cap \neg G) = \emptyset.$$

The transition or execution of a system can be represented by a sequence of global states and port contents, where each global state consists of the current states of all the components. For a system consists of N communicating components: $\{\phi_1, \phi_2, \cdots, \phi_N\}$, a (stable) global state (R) is an N-tuple $(\sigma_1; \sigma_2; ...; \sigma_N)$, and σ_j (symbolic state name) is a state of ϕ_j.

We define a component contract using a tuple of the following form $(\sigma_{j(i-1)}, A, G, \sigma_{ji})$, where A and G are the assumptions and guarantees of the component ϕ_j based on its interaction with the other components or the system environment between $\sigma_{j(i-1)}$ and σ_{ji}, where $\sigma_{j(i-1)}$ and σ_{ji} represent two successive states, the i^{th} and $(i+1)^{th}$, of ϕ_j, respectively.

3.4 Example (Continued)

We now show how the dual-dual comparator example will be formulated as an MBD and A/G model.

Model-Based Diagnosis Model

A typical MBD model focuses on defining a model of the system and seeing if the input/output observations are consistent with a nominal system state. The full set of variables is V_{MBD}, for which we will assign observable variables as $\{I_1, I_2, O_p, O_s\}$, unobservable variables for the outputs of the two buffers in each primary and secondary comparators, and failure–mode variables for each buffer and voter. The system description, SD, then consists of a set of equations specifying the relations over V_{MBD}. Because observations are assumed to be atemporal, i.e., the observable set is $\{(I_1, I_2), (O_s, O_p)\}$, we must map the state sequence $\{\sigma_1, \sigma_2, \sigma_3, \sigma_3, \sigma_3, \sigma_4\}$ into a sequence of observations such as that shown in Table 1.

In the following, we define two different models that correspond to two different types of fault reporting found in practice. The first model (which is implemented in a real aircraft), Ψ_1, assumes that presence of inputs $(I = t)$ means that the output must be t. Hence, it will issue a fault report whenever a signal mismatch is identified, as long as non-null inputs are available. We model this comparator (either primary or secondary) as follows:

$$(M_i = OK) \Leftrightarrow [(I_1 \vee I_2) \Rightarrow O_i], \;\; i = p,\; s.$$

The second model, Ψ_2, captures more detail, in that it specifies the input values, returning an output of t only if the inputs match:

$$(M_i = OK) \Leftrightarrow [(I_1 = I_2) \Leftrightarrow (O_i)], \;\; i = p,\; s.$$

We can compute diagnoses for these observations, which we list in Table 2. We denote the diagnoses for model 1 using the pair (ω_p^1, ω_s^1), which corresponds to a diagnosis based on the primary and secondary comparators. Similarly, the diagnoses for model 2 use the pair (ω_p^2, ω_s^2). In each case, we denote the presence of a diagnosis using a check-mark, ✓. In all cases, the diagnosis indicates an ambiguity group involving the relevant processors and voter. Hence, for model 1, the primary comparator diagnosis is $B1_p \vee B2_p \vee V_p$, using the component notation from Figure 1. Table 2 shows that Ψ_1 declares diagnoses for inputs corresponding to all states except σ_4; of these, the diagnoses for $\sigma 1, \sigma_2$ and σ_3 are not really due to component failures in this module, but rather due to the model identifying diagnoses given abnormal inputs. This corresponds to an example of the "cascaded faults" issue discussed earlier.

Assume/Guarantee Model

The A/G model needs to define the assumed and guaranteed behaviors in terms of observations. This model

	(I_1, I_2)	μ	O_p	O_s	(ω_p^1, ω_s^1)	(ω_p^2, ω_s^2)
σ_1	(t,f)	f	f	f	(✓, ✓)	(-,-)
σ_2	(f,t)	f	f	f	(✓, ✓)	(-,-)
σ_3	(t,t)	f	f	f	(✓, ✓)	(✓,✓)
σ_4	(t,t)	t	t	t	(-,-)	(-,-)
σ_5	(t,t)	t	f	t	(✓, −)	(✓,-)
σ_6	(t,t)	t	f	f	(✓, ✓)	(✓,✓)
σ_7	(f,f)	t	t	t	(✓, ✓)	(-,-)

Table 2: Set of states for TO/GA system, with corresponding diagnoses ω^1 and ω^2 for primary and secondary AFDCs, using MBD approach.

can specify a wide range of temporal modelling frameworks, but our example only entails time-tag comparison: the assumed behavior *requires* that the time-tag requirements $\mu \wedge I$ hold; if not then the assumption requirements fail and no guarantees can be ensured. In this case the inference process is referred to upstream components, to identify the reasons for the time-tag comparison failing.

We need to define the implementation $\mathcal{M}$, the assumption A, and the guarantee G, all of which need to be specified as temporal observations. We record each observed variable as a (value,time) pair, but for simplicity here we represent only the value. The set of observed variables is $\{I_1, I_2, \mu, O_p, O_s\}$.

If the assumption A holds, then the guarantee is that both O_p, O_s are t (in our first model Φ_1), or have the same value (in our second model Φ_1). Table 3 shows the A and G values for both models. Consider

	A	G
Φ_1	$\{\sigma_4, \sigma_5, \sigma_6\}$	$\{\sigma_4\}$
Φ_2	$\{\sigma_4, \sigma_5, \sigma_6, \sigma_7\}$	$\{\sigma_4, \sigma_7\}$

Table 3: Assumptions and guarantees for dual-comparator example

three examples of state-sequences: $\Gamma_1 = \{\sigma_0, \sigma_1\}$; $\Gamma_2 = \{\sigma_0, \sigma_4\}$; $\Gamma_3 = \{\sigma_0, \sigma_5\}$; $\Gamma_4 = \{\sigma_0, \sigma_7\}$. In the contract-based framework, the assumption is satisfied if $\mathcal{M} \cap A \neq \emptyset$, and the contract is satisfied if $\mathcal{M} \cap A \subseteq G$.

Table 4 shows the outcomes of examining the four state sequences within the A/G framework for both models. We see that:

Γ_i	$\Gamma_i \cap \lambda_1$	Φ_1-out	$\Gamma_i \cap \lambda_2$	Φ_2-out
$\{\sigma_0, \sigma_1\}$	$\emptyset$	λ_1	$\emptyset$	λ_2
$\{\sigma_0, \sigma_4\}$	σ_4	-	σ_4	-
$\{\sigma_0, \sigma_5\}$	σ_5	$\times$	σ_5	$\times$
$\{\sigma_0, \sigma_7\}$	$\emptyset$	λ_1	σ_7	-

Table 4: Outcomes of four state sequences within the A/G framework for two models, Φ_1 and Φ_2. Φ_i-out denotes the outcome: λ_i means that assumption i is violated, - denotes contract satisfied, $\times$ denotes contract is not satisfied (a fault occurred).

- Γ_1 violates the assumption for both models;
- Γ_2 satisfies the assumption and contract for both models;
- Γ_3 satisfies the assumption and violates the contract G for both models;
- Γ_4 violates the assumption for Φ_1, but satisfies the assumption and contract for Φ_2.

Differences Between Approaches

In contrast to the A/G approach, which requires observations for the implementation $\mathcal{M}$, the assumption A, and the guarantee G, MBD uses a single observation $\mathcal{O}$ that instantiates the observable variables in the model. This model is focused on isolating faults given $\mathcal{O}$. Typical MBD models are indifferent to notions of *required* inputs or outputs but specify the outputs generated by the inputs; for this model there is a requirement on both the inputs and the outputs, but the model will just compute the consistency of the outputs given the inputs.

The A/G approach requires additional inputs, but it can also identify more information than can the MBD model. This extra information includes:

- satisfaction of requirements on the module, such as input and output requirements,
- analysis of temporal interactions, and
- verification of model properties, such as forbidden states, etc.

4 ASSUME-GUARANTEE DIAGNOSTICS FRAMEWORK

This section extends the consistency-based diagnosis framework to incorporate Assume-Guarantee notions. We show that we can specify requirements $\mathcal{R}$ such that they can be used to strengthen a diagnosis model Ψ_D, i.e., to create $\Psi_{AG} = \mathcal{R} \cup \Psi_D$.

Many MBD models implicitly model the influence of the environment (inputs) on a component; here, we explicitly distinguish exogenous influences from component-specific behaviors, defining Ψ conditional on "correct" environmental conditions. The extension covers specifications for nominal operating-condition specifications (the assumption A), and component-function given A (the guarantee G). For A, we could define a set of constraints, or as a simpler abstraction, we could specify a set of variables (predicates) denoting the satisfaction of those constraints. In this article we choose the latter approach.'

We can use multiple approaches to integrate the A/G approach into an MBD model, of which we describe two approaches below. The first approach, which we call an Extended-Behaviour approach, defines a strong-fault diagnostic model that adds an assumption value representing violated input-assumption requirements, together with appropriate behaviours for such violations. The second approach appends each component behaviour specification, Ψ_ϕ with a constraint set $\mathcal{R}_\phi$ specifying valid inputs and assumption-violation mode-variables, i.e., we obtain $\Psi_{AG} = \Psi_\phi \cup \mathcal{R}_\phi$.

4.1 Extended-Behaviour Approach

We assume that we start either with a Weak-Fault Model or a Strong-Fault Model. In either case, we define a violation of an input-assumption to be a failure-

mode variable with the value *A*. and obtain a Strong-Fault Model in which failure-mode variables include the value λ.

We use the strong fault model created by introducing A/G behavior equations to classify two fault types: (a) *soft faults*, which correspond to violations of input assumptions and occur when a health assignment takes on an λ-value, and (b) *hard, or local component faults*, which correspond to faults explaining anomalous outputs in terms of faults in local components, as opposed to anomalous outputs caused by anomalous input values. Clearly, we want to be able to distinguish these two fault types, and effectively exclude the soft faults, which do not correspond to true faults in the local component.

For example, in the first model, Ψ_1, each comparator assumes that the presence of input data entails t as the output. Using this approach, we will now introduce violations of input assumptions to this model, for comparators labeled $i = 1, 2$.

$$(M_i = \lambda_i) \quad \Leftrightarrow \quad [(I_1 \neq I_2) \Rightarrow (O_i = f)]$$
$$(M_i = \lambda_i) \quad \Leftrightarrow \quad [(I_1 = f) \wedge (I_2 = f) \Rightarrow (O_i = t)]$$

Table 5 shows the A/G diagnoses for the comparator system for model Ψ_1. The table shows that the observations corresponding to $\sigma_0, \sigma_1, \sigma_2, \sigma_3$, and σ_7 denote violations of the assumed input conditions, and hence only soft faults ((λ_p, λ_s)) are generated, corresponding to primary and secondary outputs. σ_1 violates the input timing condition, and σ_2 and σ_3 violate the inputs having the same value. Notice that for these observations we do not generate local component faults, as in the standard models 1 and 2. Failure to receive $\{t, t\}$ or $\{f, f\}$ is due to upstream "problems". When dealing with larger models this approach enables us to prune the fault space and eliminate cascaded faults. Observations σ_4 through σ_6 satisfy the assumed input conditions, and hence we compute local component faults for the two cases where hard faults are actually indicated, i.e., σ_5 and σ_6.

	TO/GA output	μ	O_p	O_s	ω_{AG}
σ_0	(t,-)	-	-	-	$(-,-)$
σ_1	(t,f)	f	f	f	(λ_p, λ_s)
σ_2	(f,t)	f	f	f	(λ_p, λ_s)
σ_3	(t,t)	f	f	f	(λ_p, λ_s)
σ_4	(t,t)	t	t	t	$(-,-)$
σ_5	(t,t)	t	f	t	$(\checkmark,-)$
σ_6	(t,t)	t	f	f	$(\checkmark,\checkmark)$
σ_7	(f,f)	f	t	t	(λ_p, λ_s)

Table 5: Faults computed for TO/GA system

4.2 Extended-Constraint Approach

In this approach, we define the guarantee to be the traditional system description: $G \equiv [\mathcal{H} \Leftrightarrow \xi_i]$. We then introduce a set of assumption-violation mode-variables, λ_i: if the assumed operation conditions for component i are satisfied, then $\lambda_i = t$, and $\lambda_i i = f$ otherwise. Given this set of variables, we can now have a component operation-condition assignment, which is analogous to a health assignment: $\Lambda = \{\lambda_1, , \lambda_m\}$.

An *Assume-Guarantee Diagnostics Model* extends the traditional diagnosis model Ψ_D with equations of the form $\lambda_i \Leftrightarrow \mathcal{R}_i$ for each component i in the model. We now can define hard and soft faults for component i:

Definition 10 (Hard Fault) *ω_i is a hard fault for component i given observation α if $\Psi \cup \alpha \cup \lambda \cup \omega \not\models \perp$, and if ω is a health assignment with at least one negative literal, such that for every negative assignment $\omega_i \in \omega$ the corresponding operation-condition assignment $\lambda_i i \in \lambda$ is negative.*

A hard fault indicates an isolated fault when the system is operating within assumed (designed) conditions.

Definition 11 (Soft Fault) *ω_i is a soft fault for component i given observation α if $\Psi \cup \alpha \cup \lambda \cup \omega \not\models \perp$, and both ω_i and $\lambda_i i$ are negative.*

A soft fault indicates an isolated "fault" in component i when the system is operating with assumed (designed) conditions violated in that component.

We can identify the A/G diagnoses for the comparator system, which we list in Table 5, in the case of hard and soft faults. Note that these faults are identical to those of the extended-behaviour approach. The table shows that the observations corresponding to σ_0 through σ_3 denote violations of the assumed input conditions, and hence only soft faults ((λ_p, λ_s)) are generated. σ_1 violates the input timing condition, and σ_2 and σ_3 violate the inputs having the same value. Notice that for these observations we do not generate hard faults, as in the standard models 1 and 2. Observations σ_4 through σ_6 satisfy the assumed input conditions, and hence we compute hard faults for the two cases where hard faults are actually indicated, i.e., σ_5 and σ_6.

5 PROPERTIES OF EXTENDED MODEL

5.1 Diagnostic Soundness/Completeness

We can show that this strong fault model Ψ_{AG} can preserve all local component faults while excluding faults that the weak fault model Ψ_M identifies.

Theorem 1 *Given a weak fault model Ψ_M, a corresponding A/G model Ψ_{AG}, and an observation α, Ψ_{AG} is sound and complete with respect to the local component faults of Ψ_M.*

Proof:

Sound: Since we are using a monotonic propositional logic, adding extra clauses to any formula F will reduce the number of logical models (diagnoses) of F. If $\Omega(\Psi_M, \alpha)$ and $\Omega(\Psi_{AG}, \alpha)$ denote the set of diagnoses given the weak-fault and strong-fault models, respectively, then $\Omega(\Psi_M, \alpha) \supseteq \Omega(\Psi_{AG}, \alpha)$. Hence for a local component diagnosis ω, there is no $\omega \in \Omega(\Psi_{AG}, \alpha)$ such that $\omega \notin \Omega(\Psi_M, \alpha)$.

Complete: Let $\tilde{\Omega}(\Psi_{AG}, \alpha)$ as the local component diagnoses, i.e., diagnoses with value *bad*. If $\exists \omega \in \tilde{\Omega}(\Psi_{AG}, \alpha)$ such that $\omega \notin \tilde{\Omega}(\Psi_M, \alpha)$, then we must have $\Omega(\Psi_M, \alpha) \not\supseteq \Omega(\Psi_{AG}, \alpha)$, which is a contradiction. Hence we must have completeness of the local component faults of Ψ_{AG} with respect to Ψ_M. $\square$

We can prove an analogous theorem for our second modeling approach:

Theorem 2 *Given a diagnosis model Ψ_M, a corresponding A/G model Ψ_{AG}, and an observation α, Ψ_{AG} is sound and complete with respect to the hard faults of Ψ_M.*

5.2 Ambiguity Reduction

We now show that, by using Ψ_{AG}, we can reduce the number of ambiguous faults that arise during the fault isolation process without losing any true faults.

A complete test vector $\vec{\alpha} = \{\alpha_1, \cdots \alpha_m\}$ for a fault ω and model Ψ_D is a sequence of observations such that $\Psi_D \cup \vec{\alpha} \cup \omega \not\models \bot$ and there is no other fault $\omega' \neq \omega$ such that $\Psi_D \cup \vec{\alpha} \cup \omega' \not\models \bot$.

We now define the notion of fault ambiguity. Given a complete test $\vec{\alpha} = \{\alpha_1, \cdots \alpha_m\}$, a "true" fault ω^* is such that $\Psi_D \cup \vec{\alpha} \cup \omega^* \not\models \bot$. An ambiguous fault is some $\omega \neq \omega^*$ such that ω is entailed by some observation $\alpha \in \vec{\alpha}$, i.e., $\Psi_D \cup \alpha \cup \omega \not\models \bot$, but not for a superset test of α, i.e., $\exists \vec{\alpha'} \supset \alpha$ such that $\Psi_D \cup \vec{\alpha'} \cup \omega \models \bot$.

Given these definitions, we now prove that the strengthened model does not exclude any true faults.

Lemma 1 *Given an MBD model Ψ_D and any observation vector $\vec{\alpha}$, $\nexists \omega_R$ such that $(\mathcal{R} \cup \Psi_D) \cup \vec{\alpha} \cup \omega_R \not\models \bot$ unless $\Psi_D \cup \vec{\alpha} \cup \omega_R \not\models \bot$.*

Proof: We perform a proof by contradiction. Assume that there exists some ω_R such that $(\mathcal{R} \cup \Psi_D) \cup \vec{\alpha} \cup \omega_R \models \bot$ and $\Psi_D \cup \vec{\alpha} \cup \omega_R \not\models \bot$. In this case it must be that $\mathcal{R} \models \bot$, i.e., the requirements are inconsistent, which is a contradiction.□

5.3 System Composition

Given a set of components (V_i, P_i, Ψ_i), for $i = 1, ..., m$, the composite system is given by $(\bigcup_i V_i, \bigcup_i P_i, \bigcap_i \Psi_i)$.

The key issue is what happens to the system-level diagnosis given a sequence of observations $\alpha = \{\alpha_1, \alpha_2, ..., \alpha_l\}$: the resulting diagnosis is $\omega_l(\Psi, \alpha) = \bigcap_i \omega(\Psi, \alpha_i)$.

Since we are dealing with atemporal diagnosis models, this principle of composition works when we have MBD models extended with A/G constructs, and corresponds to the notion of contract conjunction (Benveniste *et al.*, 2008). However, for temporal models, this approach will not work, and we will need to adopt parallel composition techniques for system composition (Benveniste *et al.*, 2008). In this case, it must follow the definition below:

Definition 12 *Let $C_i = (V_i, \lambda_i, G_i)$ with $i = 1, 2$ be two contracts. We define the parallel composition between C_1 and C_2, denoted $C_1 \parallel C_2$, to be the contract $(V_1 \cup V_2, A_1 \cap A_2 \cup \neg(G_1 \cap G_2), G_1 \cap G_2)$.*

6 RELATED WORK

Assume/guarantee (or contract-based) analysis has been formalised for a long time in computer science, e.g., see (Hoare, 1969; Martin and Lamport, 1993; Abadi and Lamport, 1989). Contract-based modeling has been applied to many domains, most notably object-oriented software engineering, but also to hardware design (Sun *et al.*, 2009), component-based design (Bozzano *et al.*, 2009) and hybrid systems (Giese *et al.*, 2010). As an example of its use in object-oriented programming, contract-based modeling enables the analysis of the services provided by a class as a contract between the class and its caller. A contract is specified using two aspects: requirements made by the class upon its caller, and promises made by the class to its caller.

The area of contract-based modeling closest to this work is its use in software debugging and in safety analysis. While most work on contracts in software has focused on the verification of properties, some work on diagnosing run-time faults has been published, e.g., (Zulkernine and Seviora, 2005). A second application of contract-based modeling in software analysis is runtime verification, (Leucker and Schallhart, 2009), a software verification technique that can be viewed as a lightweight version of verification techniques such as model checking and testing. Runtime verification deals only with observed executions generated by the real system, and uses a black box (model-free) approach, as opposed to a system model. In contrast, MBD uses a full system model, and corresponds to a "heavy" verification approach like model checking. Runtime verification has examined notions of "environmental" versus "internal" properties (Schaefer and Poetzsch-Heffter, 2009), in a manner similar to that proposed here, but focusing on precise definitions of environments in which code executes using lightweight monitors.

Several model-based approaches have been proposed for safety analysis. We can classify these approaches into two main frameworks: (a) adaptations of formal methods and verification techniques for support safety analysis (Bozzano *et al.*, 2009; Joshi and Heimdahl, 2007; Grunske *et al.*, 2007); and (b) semi-formal (or *ad hoc* compositional safety analysis techniques (Joshi and Heimdahl, 2007; Zulkernine and Seviora, 2005). In contrast to the verification methods, MBD has a long history of developing diagnosis-specific algorithms, whereas the verification approach just applies existing tools, the performance of which does not scale well. In contrast to the semi-formal approaches, our extended MBD model has a clear semantics and formal basis.

On of the closest approaches, HiP-HOPS (Wolforth *et al.*, 2010), performs model-based synthesis and analysis of fault trees and FMEA. It introduces a generalized failure logic (GFL) for specifying the failure behavior of components using a mixture of Boolean logic and references to component ports and failure classes. Component behavior models consist of logical expressions of the form: Output Deviation = Internal Failures AND/OR Input Deviations. This approach lacks the theoretical underpinning of MBD, however. (Schneider and Trapp, 2010) introduces a similar notion of conditional safety certificates, where component safety is specified in terms of the component's environment, and the safety certificate is bound to these preconditions.

7 SUMMARY AND CONCLUSIONS

We have proposed a technique for extending MBD models with assume/guarantee contracts. This more general framework enables requirements-based design

properties to be integrated with an MBD model. The notion of assumptions provides extra constraints on an MBD model, thereby enabling us to distinguish operating-condition violations from internal component faults, and to rule out fault reports based on incorrect component inputs. Identified operating-condition violations can then be used to provide feedback on the system design.

In future work we plan to extend this framework further, and also to show how assume/guarantee contracts result in more precise diagnostics on real-world systems. In addition, we plan to use the extended MBD models to provide design feedback. A diagnosability analysis can identify assumption violations, which imply that particular components are operating outside of design conditions. In this case, one can then target the re-design process as indicated by the violated assumptions. Unlike standard diagnosability, which only reveals faults, this analysis reveals both faults and operating-condition violations.

ACKNOWLEDGMENTS

This work was funded by SFI contract 06-SRC-I1091.

REFERENCES

(Abadi and Lamport, 1989) M. Abadi and L. Lamport. Composing specifications. In *Stepwise Refinement of Distributed Systems Models, Formalisms, Correctness*, pages 1–41. Springer, 1989.

(Benveniste *et al.*, 2008) A. Benveniste, B. Caillaud, A. Ferrari, L. Mangeruca, R. Passerone, and C. Sofronis. Multiple viewpoint contract-based specification and design. In *Intl. Symp on Formal Methods for Components and Objects*, pages 200–225, 2008.

(Bozzano *et al.*, 2009) M. Bozzano, A. Cimatti, M. Roveri, J.P. Katoen, V.Y. Nguyen, and T. Noll. Codesign of dependable systems: a component-based modeling language. In *Proceedings of the 7th IEEE/ACM international conference on Formal Methods and Models for Codesign*, pages 121–130. IEEE Press, 2009.

(de Kleer *et al.*, 1992) Johan de Kleer, Alan Mackworth, and Raymond Reiter. Characterizing diagnoses and systems. *Artificial Intelligence*, 56(2-3):197–222, 1992.

(Giese *et al.*, 2010) H. Giese, S. Henkler, and M. Hirsch. A multi-paradigm approach supporting the modular execution of reconfigurable hybrid systems. In *Transactions of the Society for Modeling and Simulation International*, 2010.

(Grunske *et al.*, 2007) L. Grunske, R. Colvin, and K. Winter. Probabilistic model-checking support for FMEA. In *Quantitative Evaluation of Systems (QEST 2007)*, pages 119–128, 2007.

(Hoare, 1969) CAR Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–580, 1969.

(Joshi and Heimdahl, 2007) A. Joshi and M.P.E. Heimdahl. Behavioral fault modeling for model-based safety analysis. In *10th IEEE High Assurance Systems Engineering Symp.,*, pages 199–208, 2007.

(Lee and Sangiovanni-Vincentelli, 1998) E.A. Lee and A. Sangiovanni-Vincentelli. A framework for comparing models of computation. *IEEE Transactions on computer-aided design of integrated circuits and systems*, 17(12):1217–1229, 1998.

(Leucker and Schallhart, 2009) M. Leucker and C. Schallhart. A brief account of runtime verification. *Journal of Logic and Algebraic Programming*, 78(5):293–303, 2009.

(Martin and Lamport, 1993) A. Martin and L. Lamport. Composing specifications. *ACM Trans. Program. Lang. Syst*, 15:73–132, 1993.

(Meyer *et al.*, 2009) B. Meyer, A. Fiva, I. Ciupa, A. Leitner, Y. Wei, and E. Stapf. Programs that test themselves. *Computer*, 42(9):46–55, 2009.

(Reiter, 1987) Raymond Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32(1):57–95, 1987.

(Schaefer and Poetzsch-Heffter, 2009) I. Schaefer and A. Poetzsch-Heffter. Model-based Verification of Adaptive Embedded Systems under Environment Constraints. In *2nd Workshop on Adaptive and Reconfigurable Embedded Systems*, 2009.

(Schneider and Trapp, 2010) D. Schneider and M. Trapp. Conditional safety certificates in open systems. In *Proceedings of the 1st Workshop on Critical Automotive applications: Robustness & Safety*, pages 57–60. ACM, 2010.

(Slâtten, 2010) V. Slâtten. Model-Driven Engineering of Dependable Systems. In *2010 Third International Conference on Software Testing, Verification and Validation*, pages 359–362. IEEE, 2010.

(Struss and Dressler, 1989) Peter Struss and Oskar Dressler. Physical negation: Integrating fault models into the general diagnosis engine. In *Proc. IJCAI'89*, pages 1318–1323, 1989.

(Sun *et al.*, 2009) X. Sun, P. Nuzzo, C.C. Wu, and A. Sangiovanni-Vincentelli. Contract-based system-level composition of analog circuits. In *Proceedings of the 46th Annual Design Automation Conference*, pages 605–610. ACM, 2009.

(Wolforth *et al.*, 2010) I. Wolforth, M. Walker, L. Grunske, and Y. Papadopoulos. Generalizable safety annotations for specification of failure patterns. *Software: Practice and Experience*, to appear, 2010.

(Zulkernine and Seviora, 2005) Mohammad Zulkernine and Rudolph Seviora. Towards automatic monitoring of component-based software systems. *J. Syst. Softw.*, 74(1):15–24, 2005.