Model-Driven Diagnostics Generation for Industrial Automation

M. Behrens, G. Provan, M. Boubekeur, A. Mady
CCSL, Computer Science Department,
University College Cork, Cork, Ireland.
{mb20, g.provan, m.boubekeur, mae1}@cs.ucc.ie

Abstract- We propose a methodology for overcoming the current approach of writing diagnostics code for industrial automation applications after the system is designed, which results in significant extra effort/cost, and potential discrepancies between design and diagnostics output. We show how we can automatically generate a diagnostics from a more complex simulation model. We show how a model-transformation tool, ATL, can transform a hybrid-systems simulation model into a propositional-logic diagnostics model with appropriate transformation rules. We discuss issues of correctness of model transformation.

I. INTRODUCTION

One key drawback to model-driven design of large systems is the cost of constructing appropriate system models. This issue is compounded when multiple models must be built, as is typically the case when one model is constructed for design and simulation, and a separate model constructed for diagnostics.

We propose an approach for reducing the need to construct multiple models by using a meta-modeling and model-transformation approach. We assume that we can specify a generic meta-model, which is a detailed model that identifies a key set of properties of a system. For example, this model may be a detailed simulation model that includes control and sensor configurations. Given generic meta-models for a source and a target-system, we show how we can auto-generate an application-specific model. In particular, we show how we can generate a discrete diagnostics model from a hybrid-systems model.

Our approach has several key advantages. First, it can automatically generate arbitrary instances of target (e.g., diagnostics) models from an instance of a generic meta-model. Second, it also ensures that inevitable changes to a system can then be reflected in the generic meta-model and then transferred to all derived models, thus ensuring consistency across all application models.

We assume that system models are constructed using a component-based framework [9,11]. Taking advantage of this framework, we propose a component-based model transformation process.

Our contributions are as follows:

1. We propose a model-generation process that creates simpler and/or abstracted instances of a target meta-model from an instance of a generic meta-model.
2. We adopt a component-based transformation process, such that we transform sub-models on a component-wise basis, and then compose the system-level model by sub-model composition.
3. We demonstrate our model transformation process using a hybrid systems generic model, with our target model being a propositional-logic diagnosis model.

II. COMPOSITIONAL MODELING PROCESS

A. Compositional Model Framework

Component-based modelling is a key part of modern engineering design, as it embodies the principles of modularity, regularity and hierarchy, which enable cost-effective and reliable design [10]. Further, the regularity of component-based methodology translates into reduced design, fabrication and operation costs.

In creating models from a component library within a Model-Based framework [9,11], we call a well-defined model fragment a component. We assume that each component can operate in a set of behaviour-modes, where a mode $M$ denotes the state in which the component is operating. For example, a pump component can take on modes nominal, high-output, blocked, and cavitating.

We define two classes of components: primitive and composite. A primitive component is the simplest model fragment to be defined. For such a component we specify the inputs $I$, outputs $O$, and functional transformation $\varphi$, such that we have $O = \varphi(I)$. In this article we will assume that we specify our systems as hybrid systems [12], in which case our primitive component transformation functions will have hybrid systems semantics. Fig. 1 shows two examples of components; for example, the controller component has one input ($C_{in}$) and one output ($C_{out}$), with $\psi(C_{in}) = C_{out}$.

A composite component consists of a collection of primitive components which are merged according to a set $\xi$ of composition rules [6]. In this article we assume standard composition rules; specifying the semantics of composition is beyond the scope of this article, and we refer the reader to [11] for details.
A set of (primitive/composite) components defines a component library. In this article we assume a component library consisting of sensors, actuators, human-agent models, and building components such as lights, windows, rooms, etc.

A model consists of the pair \((C, G)\), where \(C\) is a component set and \(G\) is a graph denoting the topology of component connections. We assume that our models are causal, so \(G\) has directed edges.

Each component has a set of properties that depends on the model class, e.g., discrete-event model, hybrid model, etc. We represent an abstract component using the tuple \(\{\text{model class}, \text{e.g., discrete-event model, hybrid model, etc.}\} \in \) each variable \(V\).

In this article we consider transforming a generic model, \(\Phi\), into a hybrid systems model, \(\Phi_{HS}\), and building components such as lights, windows, rooms, etc.

In the following, we outline the models we use for specifying the hybrid systems and diagnosis frameworks.

Hybrid Systems Model

**Definition 1** An autonomous hybrid automaton \(HS\) is a tuple \((M, X, Init, Inv, t, f)\), where \(M\) is a finite set of mode variables interpreted over finite domains, \(M\) denotes the finite set of all valuations of the variables \(M\) over the respective finite domains, \(X\) is a finite set of variables interpreted over the reals \(\mathbb{R}\), \(X = \mathbb{R}^X\) is the set of all valuations of the variables \(X\), \(Init \subseteq M \times X\) is a set of initial states, \(Inv : M \rightarrow 2^X\) assigns to each discrete state \(M \in M\) an invariant set, \(t \subseteq M \times X \times M \times X\) is a set of (guarded) discrete transitions, \(f : M \rightarrow (X \rightarrow \mathbb{T} X)\) is a mapping from the discrete states to vector fields that specify the continuous flow in that discrete state.

Propositional Diagnosis Model

**Definition 2** A discrete diagnosis model is specified by a tuple \(\{\mathcal{V}, \mathcal{M}_D, \mathcal{O}, \Psi, \pi\}\), where
- \(\mathcal{V}\) is a set of discrete-valued variables;
- \(\mathcal{M}_D \subseteq \mathcal{V}\) is the set of failure mode variables;
- \(\mathcal{O} \subseteq \mathcal{V}\) is the set of observable variables;
- \(\Psi\) consists of propositional equations; and
- \(\pi\) is a discrete probability distribution over the equations and/or variables.

We can further specify the system variables as follows. For each variable \(V \in \mathcal{V}\) we have a domain for \(V\), denoted \(D_V\). A subset of variables, \(\mathcal{M} \subseteq \mathcal{V}\), is denoted as the set of mode variables, with domain \(D_M\).

In this section, we give examples of our modeling framework for two simple components, a sensor and a setpoint-based controller.

In the diagnostics framework, a sensor is a device with one input, one output, and 2 modes, \{OK, bad\}. If the discrete value of the variable being measured is \(V_{in}\) and sensor is OK, then the sensor reading must equal \(V_{in}\) and the output is \(V_{out}\). We adopt a weak-fault model framework for diagnostic models [13], i.e., we do not specify the faulty behavior, when the mode \(M_{s=bad}\) but just the normal behavior. This approach simplifies the modeling process, and has relatively simple associated diagnosis inference [13]; the drawback is that fault isolation may be less precise compared to models in which failure-mode behaviours are specified.

The generic sensor equations \(\psi_{S}\) for a weak fault model are thus:

\[
(M_s = ok) \Leftrightarrow (V_{in} = V_{out})
\]

A simple setpoint-based controller will always try to maintain a setpoint \(\mu\) for the signal, which in Fig. 1 is \(V_{in}\). The generic weak-fault controller equations for such a controller, \(\psi_{C}\), are given by:

\[
(M_{C} = ok) \Leftrightarrow [(V_{in} \geq \mu) \Rightarrow (C_{out} = off)]
\]

\[
(M_{C} = ok) \Leftrightarrow [(V_{in} < \mu) \Rightarrow (C_{out} = on)].
\]

In contrast to the diagnosis model for a sensor, in which the measured variable \(V_{in}\) takes on a finite set of discrete values, in a hybrid-systems framework \(V_{in}\) can take on continuous values governed by a differential equation. In addition, hybrid-systems simulation models typically do not restrict themselves to modes covering the health of a sensor, but will specify modes for the behavior of \(V_{in}\). For example, the equations may take the form:
Fig. 2. Component-based diagnosis model for lighting model. For each component with a failure mode, it is indicated using M_i.

\[
\begin{align*}
13 \geq t \geq 6 & \quad \partial V_u / \partial t = 50 + V_0; \\
18 \geq t \geq 13 & \quad \partial V_u / \partial t = 800 - V_u; \\
24 \geq t \geq 18 & \quad \partial V_u / \partial t = 0.
\end{align*}
\]

D. Example: Lighting System

This section now extends the components to create a simple lighting system model. Fig. 2 shows a component-based model for a lighting system. Here, we aim to maintain a simple lighting system model. Fig. 2 shows a component-based diagnosis model for lighting model. For each component with a failure mode, it is indicated using M_i.

\[
\begin{align*}
13 \geq t \geq 6 & \quad \partial V_u / \partial t = 50 + V_0; \\
18 \geq t \geq 13 & \quad \partial V_u / \partial t = 800 - V_u; \\
24 \geq t \geq 18 & \quad \partial V_u / \partial t = 0.
\end{align*}
\]

In this model, a controller must take the value of Combined-Light, in addition to the output of Presence-Sensor, P_out, to determine whether to switch on the Interior-light. Hence we have three components for estimating light level (Combined-Light), complemented by the Presence-Sensor and the Control component that controls turning Interior-light on and off.

Note that Exterior-light and Presence-Sensor are examples of the sensor component described previously. The Control component is a setpoint-based controller that also incorporates the presence of a person in a room.

In a diagnosis model, the controller equations, \( \psi_C \), are given by:

\[
\begin{align*}
(M_C = ok) & \iff [(V_u \geq \mu) \land (P_{out} = t) \Rightarrow (S_{out} = off)] \\
(M_C = ok) & \iff [(V_u < \mu) \land (P_{out} = t) \Rightarrow (S_{out} = on)].
\end{align*}
\]

It is relatively straightforward to define transformations for the sensor and setpoint-based controller components. However, it is much more complicated to define other components, such as the Combined-light component. In these other cases, one needs to define specific transformation rules, on an instance-specific basis. To create a diagnosis model for the Combined-light component, for example, we must define a qualitative algebra to specify the Combined-Light equation. Given the possible qualitative values of Exterior-light and Combined-light are \{low, optimal, high\}, and of Interior-Light are \{on, off\}, a truth-table for this qualitative relation is given in Table 1.

<table>
<thead>
<tr>
<th>Exterior-light</th>
<th>Interior-Light</th>
<th>Combined-light</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low</td>
<td>Off</td>
<td>Low</td>
</tr>
<tr>
<td>Low</td>
<td>On</td>
<td>Optimal</td>
</tr>
<tr>
<td>Optimal</td>
<td>Off</td>
<td>Optimal</td>
</tr>
<tr>
<td>High</td>
<td>Off</td>
<td>High</td>
</tr>
<tr>
<td>high</td>
<td>on</td>
<td>high</td>
</tr>
</tbody>
</table>

Table 1. Qualitative truth table for determining value of Combined-Light from Exterior-Light and Interior-Light.

The hybrid-systems model for the Exterior-Light function is quite different from that of the diagnosis model. Here, we have an automaton with 3 modes, Sunrise, Day and Evening, where the behavior in each mode is governed by a differential equation based on a light factor \( \lambda \). Note that this model has a different set of modes to the diagnosis model for Exterior-Light, which is given by:

\[
(M_C = ok) \iff (Exterior-Light = V_{out}).
\]

In other words, the diagnosis model makes explicit only the distinction between the functional states of the sensor, and does not represent any nominal modes of the light level. A similar difference holds between all the hybrid-systems and diagnosis model components.

III. MODEL TRANSFORMATION

A. Model Transformation Underpinnings

We assume that we perform component-wise transformation, and then assemble to resulting transformed model \( \Phi^f \) from the transformed components. In other words, given input components \( C_i \) and \( C_j \), we can create the original model as \( \Phi = C_i \odot C_j \). We can then compose the transformed components, \( \gamma(C_i) \) and \( \gamma(C_j) \), to create the transformed model, using the model composition operator \( \odot \) for the transformed system. In this case, we have \( \Phi^f = \gamma(\Phi) = \gamma(C_i) \odot \gamma(C_j) \).

There are two entities that we must transform give a model \( \Phi^f = (C, G) \), the components \( C \) and the component connections, defined by \( G \).

It is beyond the scope of this article to prove such properties, and we refer the reader to [3] for details of such formal properties. In this article, we assume that we have an abstraction transformation, in that the diagnosis model is an abstraction of the generic model. We adopt the following
Definition 3 Let $\Phi = (M, X, \text{Init}, \text{Inv}, t, f)$ be a hybrid automata and $\Phi^T = (M', \text{Init}', t')$ be a discrete transition system. We say $\Phi^T$ is an abstraction for $\Phi$ if there exists a mapping $\gamma: M \times X \rightarrow M'$ such that

(a) If $(M, x) \in \text{Init}$, then $\gamma(M, x) \in \text{Init}'$, and
(b) If $((M, x), (M', x')) \in t$, then $\gamma$ is a transition in the discrete transition system $\Phi^T$ corresponding to $\Phi$ with respect to $\gamma$, then there exists a transition $(\gamma(M, x), \gamma(M', x')) \in t'$ in $DS$.

Hence, as noted in [3], the abstraction $\Phi^T$ of a hybrid automaton $\Phi$ is a discrete transition system that simulates the discrete system $\Phi^T$ associated with $\Phi$, where $\gamma$ defines the corresponding simulation relation [7]. Consequently, if a temporal logic formula (e.g., in ACTL$^\omega$) is true in the model $\Phi^T$, then it is also true in $\Phi^T$ [6].

B. Transformation Process

We use the theory of model transformation [5] to formalize our transformation process in terms of a rewrite procedure. Model transformations that translate a source model into an output model can be expressed in the form of rewriting rules. Such rules can be classified according to a number of categories [1]. According to [1], the transformation we adopt is an exogenous transformation, in that the source and target model are expressed in different languages, i.e., hybrid-systems and propositional logic languages.

In order to use this approach to model transformation, we need to represent our models in terms of their corresponding meta-models. A meta-model is an abstraction of the model that highlights the properties of the model itself. We adopt the OMG standard for the meta-model, called Meta-Object Facility, or MOF. We use the MOF-based tool, ATL [2], to implement our model transformation process. For the generic model, we use the Charon language [15], and for the diagnostics models we use the Lydia language [16].

Figures 4 and 5 depict the meta-models for the hybrid-systems and diagnosis models, respectively. Fig. 6 depicts the meta-model transformation process.

We adopt the definitions of [4] for meta-model mapping and instance.

Definition 4: A meta-model mapping is a triple $\Omega = (S_1, S_2, \Sigma)$ where $S_1$ is the source meta-model, $S_2$ is the target meta-model and $\Sigma$, called the mapping expression, is a set of constraints over $S_1$ and $S_2$ that define how to map from $S_1$ to $S_2$.

Definition 5: An instance of mapping $\Omega$ is a pair $(\mathcal{S}_1, \mathcal{S}_2)$ such that $\mathcal{S}_1$ is a model that is an instance of $S_1$, $\mathcal{S}_2$ is a model that is an instance of $S_2$ and the pair $(\mathcal{S}_1, \mathcal{S}_2)$ satisfies all the constraints $\Sigma$.

For example, given our general meta-model, a sensor component must satisfy the property of being an instance meta-model.
$S_2$ is a translation of $S_1$ if the pair $\langle S_1, S_2 \rangle$ satisfies Definition 3 [3]. Hence, we must specify an appropriate mapping, or set of rules $\Sigma$, to ensure that this holds. In the following, we will describe component-based rules.

C. Component-Based Meta-model Transformations

In this paper, we assume a component-based framework for meta-model mapping, in which we map component by component. In other words, we assume that we can represent a model in terms of a connected set of components, where we call our set of components a component library $C$. Further, we assume that for each component $C_i \in C$, we have an associated meta-model. Given that we are mapping from component library $C_i$ to $C_j$, we have two component libraries, with corresponding meta-model libraries, $S_1 = \{s_{11}, \ldots, s_{1n}\}$ and $S_2 = \{s_{21}, \ldots, s_{2m}\}$. In an analogous fashion to the system-level requirements for the model transformation, at the component level the following must hold: $s_{2i} \in S_2$ is a translation of $s_{1i} \in S_1$ if the pair $\langle s_{1i}, s_{2i} \rangle$ satisfies Definition 3. This means that there is a relation $\rho \subseteq S_1 \times S_2$ that maps meta-model components of $S_1$ to equivalent components in $S_2$.

In addition to these abstract requirements, there are two variable-types that must be defined to map to a diagnosis model: the failure-mode and observable variables, $M_D$ and $O$ respectively. In most model-based diagnosis applications, it is assumed that (1) every component has an associated failure-mode, and that (2) the observables $O$ are restricted to sensor- and actuator-outputs. Using our component-based approach, property (1) can easily be ensured for weak-fault models.

D. Verification Properties

One important property of the model transformation is that it enables you to verify that the appropriate properties of the generated model hold.

The correctness of a model transformation depends on several factors, covering issues such as computational properties (e.g., whether the transformation terminates), syntactic properties (e.g., whether the output model complies with the syntactical rules of the output language), and correctness properties (e.g., whether output model achieved the intended result of mapping the semantics of the input model into that of the output model). In this article, the output diagnosis model is "simulation-correct" if it reproduces the behaviour of the original hybrid-systems model where model behaviours correspond. The mapping of Definition 3 ensures simulation-correctness [3].

A second correctness property could be termed "diagnosis-correct". In this article we have specified a simple form of the diagnosis model, called a weak fault model, in which we specify only normal behaviour using a binary-valued failure mode [13]. This approach guarantees a diagnosis-correct output model.

Theorem 1: A meta-model mapping $\Omega = (S_1, S_2, \Sigma)$, where $S_1$ is a hybrid-systems meta-model, $S_2$ is a weak-fault propositional diagnostics meta-model and $\Sigma$ is a set of simulation-correct mapping constraints over $S_1$ and $S_2$, is diagnosis correct.

If we want to specify detailed failure mode behaviours, as in strong fault models [14], then additional mapping constraints are necessary to provide guarantees of diagnosis-correctness.

A final form of correctness is topology-correctness. In this article we have assumed that the hybrid-systems and diagnosis models will have the same topology, and all that is transformed is the component specifications. In general, this assumption may not hold. When it does hold, work has been done to guarantee that the transformation matches certain graph structures in the input graph and creates certain structures in the output graph [8].

IV. EXAMPLE

In this section we illustrate through a simple lighting system the transformation steps to generate Lydia programs from Charon models. As explained earlier the system consists of a simple lighting controller that tracks the presence and defines the internal light level according to the external light intensity.

The Charon model is composed by two agents: agent Presence that deals with the presence, it implements the mode that switches the light according to presence and internal light sensor, agent Light defines the mode that controls the combined light.

The transformation engine parses the Charon model following the hierarchical structure in accordance to the transformation rules. The code shown in Fig. 7 illustrates the rules that apply to transform agent hierarchy into the corresponding system/sub-systems Lydia model.

At the bottom level, the transformation engine translates the Charon transitions into Lydia if/else structure following the rules described in Fig 8. The basic idea is to track to condition for the observable variables. This can be done by looking at the action and its related conditions. Transformation rules had to be written to consider the case where same actions are executed under different conditions.
We have described a framework for automatically transforming a generic model into an abstracted model. We have described how we can define a generic model using the hybrid-systems language, and then transform this to a discrete propositional diagnosis language. Further, we have shown how compositional model representation can be adopted for such model transformation. We have provided an example of a lighting system for building automation to demonstrate this procedure.

This approach can make significant contributions to building automation systems. Instead of needing to create multiple models, and maintain consistency among multiple models, this transformation approach provides the methodology for creating a single generic model, and then creating component-based meta-models and transformation rules to automate the generation of the additional models needed for building automation applications.

ACKNOWLEDGMENT

This work was funded by SFI grant 06-SRC-11091.

REFERENCES

[16] http://www.st.tewi.tudelft.nl/~gemund/Lydia/