Temporal Model-Based Diagnostics Generation for HVAC Control Systems

Marion Behrens and Gregory Provan

Complex Systems Lab, University College Cork, Ireland

Abstract. Optimizing energy usage in buildings requires global models that integrate multiple factors contributing to energy, such as lighting, “Heating, Ventilating, and Air Conditioning” (HVAC), security, etc. Model transformation methods can then use these global models to generate application-focused code, such as diagnostics or control code. In this paper we focus on using model transformation techniques to generate model-based diagnostics (MBD) models from “global” building systems models. This work describes the automated generation of models for MBD by considering control systems which are described through behavior that also relies on the state of the system.
Our approach contributes to model-driven development of complex systems by extending model consistency up to models for diagnostics. We transform hybrid-systems (HS) models into models based on propositional temporal logic with timing abstracted through sequentiality, and illustrate the transformation process through a simple example.

1 Introduction

One emerging area of research is developing software to improve the energy efficiency of buildings. To fully realize this goal, it is necessary to integrate the operation of all building systems, so that we can optimize the building’s operation on a global setting. In this case, a global building model is necessary. We can then use such a global model to generate application-specific code, e.g., lighting controls, diagnostics, etc. This use of a global model to drive code generation for multiple applications will significantly reduce the cost of software development, as opposed to generating code for each application from scratch.

In this article we focus on using model transformation techniques to generate MBD models from “global” building systems models. This transformation process is a very challenging task, given the significant differences between the global and MBD models. The global (or source) model is defined using a hybrid-systems (HS) [1] language, which describes the continuous-valued dynamics of a range of building parameters, such as temperature and light levels, movement of people throughout the zones of the building, control of HVAC, lighting, etc. The MBD (target) model focuses on discrete system parameters governing *abnormal* operation of building components, such as the HVAC/lighting controllers and the associated equipment, e.g., chillers, pumps, fans, etc. Consequently, the model transformation process must extract just the diagnostic-related information from the source model and further inject diagnostic-related information that is not contained in the source model, e.g. failure probabilities, from other sources.

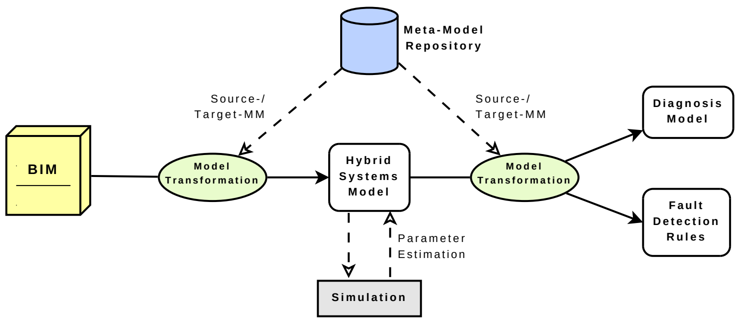

Fig. 1. Model-Transformation Context

Fig. 1 shows the global context of a framework for fault detection and diagnosis of building automation in which the transformation from HS models into models for diagnosis plays a key role. This framework has been roughly outlined in [2]. The figure shows how the source global model, which is provided by a building information modeling (BIM) tool, can be transformed. The BIM contains information about 3D objects, i.e. the building's geometry and quantities, properties and interconnection of building components, as well as information about the building control and a collection of data produced by components, e.g. sensors and actuators, during its life cycle. From the BIM we can extract an operational building model, represented as HS model. After simulating the system in order to assign parameters with estimated values, an MBD model can be generated from the refined HS model.

Our contributions are as follows:

1. We present a modeling methodology for HS models that extends existing approaches and enables the generation of MBD models.
2. We show how we can abstract the continuous-time dynamics of an HS model to create a discrete-event temporal model which can enable temporal diagnosis.
3. We illustrate our model-transformation approach through a detailed example of an on/off control system of a thermostat.

The article is organized as follows: In section 2 we describe the architecture of the model-transformation framework which we adopted. Section 3 describes the source models of the transformation (hybrid systems), and the target models of the transformation, temporal MBD models. Section 4 outlines the step-by-step transformation process from a HS model towards a model for temporal MBD.

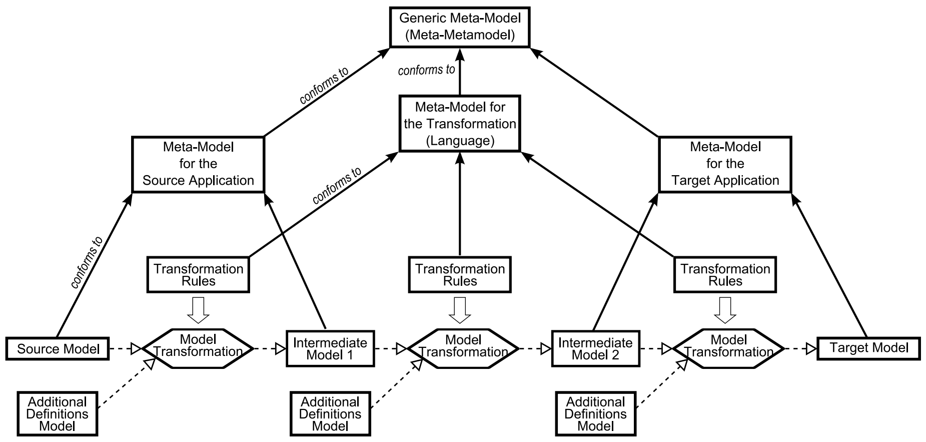
Generic Meta-Model (Meta-Metamodel)
conforms to
Meta-Model for the Transformation (Language)
Meta-Model for the Source Application
Meta-Model for the Target Application
conforms to
conforms to
conforms to
Transformation Rules
Transformation Rules
Transformation Rules
Source Model
Model Transformation
Intermediate Model 1
Model Transformation
Intermediate Model 2
Model Transformation
Target Model
Additional Definitions Model
Additional Definitions Model
Additional Definitions Model

Fig. 2. Architecture of the Model Transformation

2 System Architecture

Fig. 2 depicts the main features of the system architecture for our model transformation. This figure shows that we have two applications, a source and a target application, with a corresponding meta-model for each application.

To generate a target model from a source model, two intermediate forms are taken. Enrichment of the models is essential to inject information that is specific to the target application. We inject these information through additional models that contain general parameters as well as parameters referring to specific elements of the model at each transformation step.

We use the theory of model transformation [3] 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. According to Mens et al. [3], 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. The exogenous transformation step is encompassed by two endogenous transformation steps, in which the source and target model are expressed in the same language.

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

Definition 1. *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 Σ, 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 2. *An* instance *of mapping $\otimes$ is a pair $\langle \check{S}_1, \check{S}_2 \rangle$ such that $\check{S}_1$ is a model that is an instance of S_1, $\check{S}_2$ is a model that is an instance of S_2 and the pair $\langle \check{S}_1, \check{S}_2 \rangle$ satisfies all the constraints Σ. S_2 is a translation of S_1 if the pair $\langle S_1, S_2 \rangle$ satisfies definition 1. Hence, we must specify an appropriate mapping, or set of rules Σ, to ensure that this holds.*

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 $\mathcal{C}$. Further, we assume that for each component $C_i \in \mathcal{C}$, we have an associated meta-model. Given that we are mapping from component library $\mathcal{C}_1$ to $\mathcal{C}_2$, we have two component libraries, with corresponding meta-model libraries, $S_1 = \{s_{1,1}, \ldots, s_{1,n}\}$ and $S_2 = \{s_{2,1}, \ldots, s_{2,m}\}$.

Given a component-wise transformation, we must then assemble the resulting transformed model Φ^T from the transformed components. In other words, given input components C_i and C_j, we can create the original model as $\Phi = C_i \otimes C_j$. We can then compose the transformed components, $\gamma(C_i)$ and $\gamma(C_j)$, to create the transformed model, using the model composition operator $\oplus$ for the transformed system. In this case, we have $\Phi^T = \gamma(\Phi) = \gamma(C_i) \oplus (C_j)$.

In order to apply this approach to model transformation, we need to represent our models in terms of their corresponding meta-models. The Eclipse Modeling Framework Project (EMF)[1] provides a generic meta-model called "Ecore" for describing meta-models. To implement our model transformation process, we use a suite of tools based on the Ecore model, i.e., Xtend for Model Transformation, Xpand for Code Generation and a workflow language, which were formerly developed under the openArchitectureWare (oAW)[2] platform and have now been integrated into the EMF. Mapping rules are expressed in Xtend, which is a general purpose transformation language that enables to build transformation rules over meta-model elements.

3 Source and Target Model Formalisms

3.1 Hybrid-systems Model

A HS model can describe how a discrete controller can drive a system (plant) whose state evolves continuously. A HS describes the evolution of its state space over time. Each state can be either continuous, if it takes values in Euclidean space $\mathbb{R}^n$ for some $n \geq 1$, or discrete, if it takes values in a finite set $\{q1, q2, \ldots\}$.

Hybrid Systems Language A hybrid automaton H is a mathematical model for describing an HS. It is a finite state machine (FSM) augmented with differential equations for continuous updates. The state of the system is a pair $S \equiv (x, q)$ in which $x = (x_1, \ldots, x_n)$ with $x_i \in \mathbb{R}$ is the set of n continuous state variables, and $q \in Q$ with $Q = \{q_1, q_2 \ldots\}$ is the discrete state. The discrete state might have multiple dimensions $q \in Q_1 \times \ldots \times Q_m$ when automata are extended to enable modeling of hierarchical behavior, but through indices over the Cartesian product of the finite sets Q_i, the discrete state can always be reduced to a single dimension. In the hybrid automaton, the discrete state can

[1] `http://www.eclipse.org/modeling/`
[2] `http://www.openarchitectureware.org/`

change only during a transition into another mode $q \rightarrow q'$ while the continuous state can also be updated through differential equations within a mode q. Discrete changes $E = \{S_{t+1} = g_{q\rightarrow q'}(S_t)\}$ are called *jumps* and continuous updates $F = \{\dot{S} = f_q(S)\}$ are refered to as *flows* [1].

Composition of Hybrid automata When developing large scale systems, the complexity of the system can be mastered by composing models of simple components, each described through a hybrid automaton. In such composition, hybrid automata $H_1, H_2, \ldots$ generally require an input $V = V_C \cup V_D$ and might provide an output $Y = Y_C \cup Y_D$ where C refers to continuous and D to discrete variables. Two components H_i and H_j can communicate through these interfaces. If an input variable v of H_j is renamed as an output variable y of H_i, that dimension of the state of H_i which is represented by y becomes visible to H_j.

Example: Thermostat Our running example is a thermostat which controls the temperature of a room as in fig. 3 by turning a radiator on and off. On this activation the radiator increases its temperature towards a maximum heat and decreases its temperature respectively. This control pattern causes the air temperature to bounce between the minimum and maximum border of the defined range, and even beyond these borders (fig. 4), depending on the response time and thermal inertia.

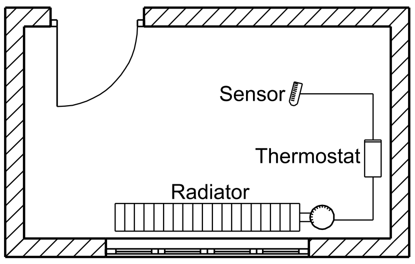

Fig. 3. Room with temperature sensor, thermostat and radiator

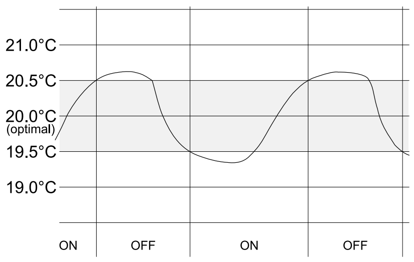

Fig. 4. Continuous flow of the air temperature in a range of $20.0 \pm 0.5°C$

A model for the thermostat is represented as a simple discrete automaton with a state $q \in \{\mathrm{ON}, \mathrm{OFF}\}$. The air temperature is defined to be optimal in a range of $\vartheta_{\mathrm{opt}} \pm \theta$. Therefore the jump conditions between the modes ON and OFF are defined as

$$\begin{aligned} &\sigma_{\mathrm{ON}\rightarrow\mathrm{OFF}} \Leftrightarrow x_1 \geq \vartheta_{\mathrm{opt}} + \theta \\ \text{and} \quad &\sigma_{\mathrm{OFF}\rightarrow\mathrm{ON}} \Leftrightarrow x_1 \leq \vartheta_{\mathrm{opt}} - \theta \ . \end{aligned}$$

The radiator has been modeled as a hybrid automaton as illustrated in fig. 5 with a discrete state $q \in \{\mathrm{IncrTemp}, \mathrm{HighTemp}, \mathrm{DecrTemp}, \mathrm{LowTemp}\}$ and a continuous state of dimension two: $x = (x_1, x_2)$, where x_1 denotes the air temperature in the room and x_2 refers to a local timer. When the radiator is in the state of HighTemp, the air temperature $x_1 \in \mathbb{R}$ in the room increases exponentially towards the hottest possible temperature T_H according to the differential equation $\dot{x}_1 = \text{rate}_H(T_H - x_1)$ for some $\text{rate}_H > 0$. When the radiator is in the state of LowTemp, the temperature of the room decreases exponentially towards the coldest possible temperature T_C according to the differential equation $\dot{x}_1 = \text{rate}_C(T_C - x_1)$ for some $\text{rate}_C > 0$. Between these steady modes, transitional modes DecrTemp and IncrTemp implement the switchover points by updating x_1 through some unspecified functions φ and ψ. The local timer $x_2 \in \mathbb{R}$ is reset to 0 when entering one of the transitional modes and updated through the differential equation $\dot{x}_2 = 1$. The jump conditions from the steady to the transitional modes refer to the state of the thermostat $v \in \{\text{ON}, \text{OFF}\}$, whereas the jump conditions from the transitional to the steady modes

$$\begin{aligned} &\sigma_{\text{IncrTemp}\rightarrow\text{HighTemp}} \Leftrightarrow x_2 \geq k_{\text{incr}} \\ \text{and} \quad &\sigma_{\text{DecrTemp}\rightarrow\text{LowTemp}} \Leftrightarrow x_2 \geq k_{\text{decr}} \end{aligned}$$

ensure that the transient modes are activated for just the appropriate time interval.

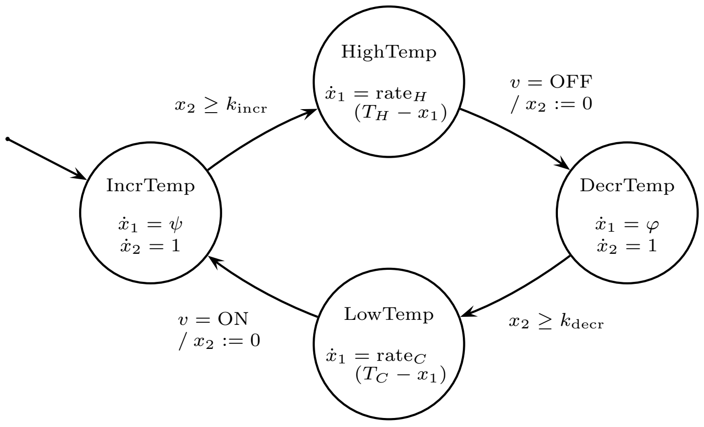
HighTemp
$\dot{x}_1 = \text{rate}_H (T_H - x_1)$
$v = \text{OFF}$ / $x_2 := 0$
$x_2 \geq k_{\text{incr}}$
IncrTemp
$\dot{x}_1 = \psi$
$\dot{x}_2 = 1$
DecrTemp
$\dot{x}_1 = \varphi$
$\dot{x}_2 = 1$
$v = \text{ON}$ / $x_2 := 0$
LowTemp
$\dot{x}_1 = \text{rate}_C (T_C - x_1)$
$x_2 \geq k_{\text{decr}}$

Fig. 5. Hybrid Automaton of the Radiator Component

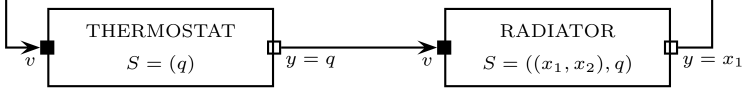

Fig. 6. Composition of two Components: Thermostat and Radiator

3.2 Model-Based Diagnosis

Model-Based Diagnosis Language. Fault diagnosis is the process of analyzing non-nominal system behavior in order to identify and localize the components that are the root cause of a failure. Diagnosis is generally involved with fault detection, known as FDD (Fault Detection and Diagnosis). Fault detection is the indication of anomalies through verifying observed values of a real system on the basis of rules. Diagnosis then isolates the underlying faults.

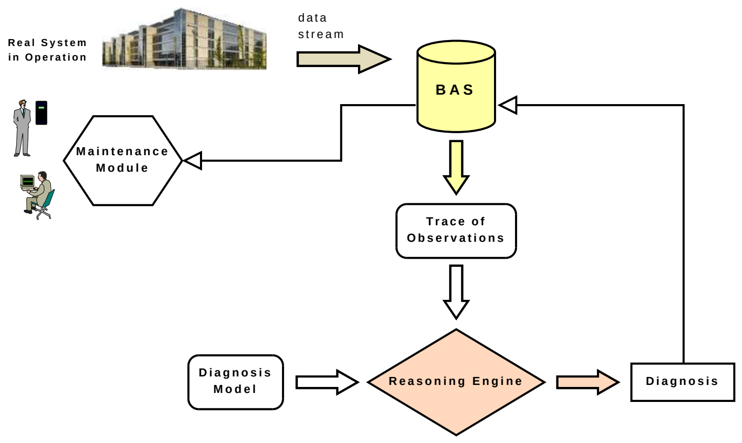

Fig. 7. Diagnosis Architecture

Fig. 7 illustrates how diagnosis is embedded in our framework for fault detection and diagnosis of building automation systems (BAS). Once a fault has been detected, the observed values are compared with the predicted state of the system using the diagnosis model. Through reasoning strategies implemented in MBD engines the malfunctioning components of the system can be located. Depending on the type of the diagnosis model the reasoning engine can even identify which type of failure occurred in each malfunctioning component. However, the generation of *strong* diagnosis models which describe not only the nominal but also the faulty behavior, and therefore allow the distinction between different failure types, is still work in progress. *Weak* models describe only the nominal behavior in relation to a boolean *health variable*, e.g. $h \Rightarrow (o \Leftrightarrow (i_1 \wedge i_2))$ for an and-gate with inputs i_1 and i_2, output o and the health variable h.

Most diagnoses in the intelligent building domain have a state- and time-dependent behavior. Taking a temporal dimension into account makes diagnosis significantly more complex than atemporal MBD with static models. Approaches for temporal MBD are distinguished between different temporal phenomena, categorized in [5]. We pursue an approach which abstracts timing by sequentialization. The diagnosis model describes the behavior of a system as a sequence of states. A diagnosis is no longer computed based on a single observation α_t over OBS made at a time t, but based on a sequence of observations.

We define a temporal diagnosis model using a standard temporal propositional logic [6]. A model for a diagnostic system, DS, consists of propositions P over a set of temporal variables V, a set of assumables (health variables) AS $\subseteq$ V and a set of observable variables OBS $\subseteq$ V. For modularity reasons we favor a modeling technique where each health variable h is encapsulated in a component $c \in$ COMPS with input and output parameters and instantiated by a superordinate component.

4 Transformation

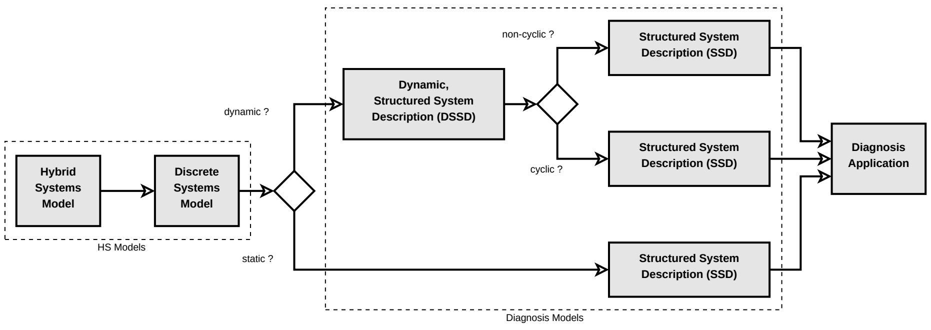
Hybrid Systems Model
Discrete Systems Model
HS Models
dynamic ?
static ?
Dynamic, Structured System Description (DSSD)
non-cyclic ?
cyclic ?
Structured System Description (SSD)
Structured System Description (SSD)
Structured System Description (SSD)
Diagnosis Models
Diagnosis Application

Fig. 8. Transformation Process

This section describes each step of the transformation from an HS model to a model for temporal MBD. Fig. 8 illustrates the steps of the model-transformation process. In this work we present the transformation into a model for MBD for the case of dynamic systems with feedback control through the following transformation steps:

1. **Discrete Abstraction** of a HS model into a discrete-systems model.
2. **Generation of Propositions** from automata contained in the components of the discrete-systems model results in a dynamic, structured system description (DSSD).
3. **Temporal Unfolding** creates a structured system description (SSD) from a DSSD for a finite sequence of time slices.
4. **Code Generation** to receive a model executable with an MBD tool.

The intermediate models are (1) a discrete-systems model that is a discrete abstraction of a HS model and therefore conforms to the HS meta-model, (2) a dynamic, structured system description (DSSD) [7] is a system model that describes component behavior through temporal propositional logic, permits directed cycles of component interconnectivity and which conforms to the diagnosis meta-model, (3) a structured system description (SSD) [7] that is a special form of a DSSD with only acyclic component interconnectivity and therefore conforms to the same diagnosis meta-model. The meta-models and transformation rules are also documented in a technical report [8].

4.1 Summary of Transformation Approach

Our model transformation converts a model described in a HS language into a model described in a diagnosis language, using meta-modeling as the methodology. We first outline the entities that are transformed, and then describe the transformations in detail.

As described earlier, we can summarize a HS model using the tuple $\langle \mathcal{S}_H, \Delta_H \rangle$, where $\mathcal{S}_H = \{Q, X\}$ is the set of state-variables, where each state is represented by its discrete and continuous parts, $\{q, x\}$ respectively, for $x \in X, q \in Q$. $\Delta_H = \{E, F\}$ is the set of flow equations, with E being the discrete and F the continuous equations. Analogously, we can summarize a temporal diagnosis model using the tuple $\langle \mathcal{S}_D, \Delta_D \rangle$, where $\mathcal{S}_D = \{Q_D\}$ is the set of discrete state-variables. Δ_D is the set of discrete-valued temporal propositional equations.

To generate a diagnosis model, we transform $\mathcal{S}_H$ to $\mathcal{S}_D$, and Δ_H to Δ_D. We must handle the continuous-discrete transformation differently than the discrete-discrete transformation as presented in [9], and we outline some details below.

4.2 Discrete Abstraction

The initial system models are HS models which consist of components containing hybrid automata. With reasoning engines however, only a finite number of system states can be diagnosed. Therefore it is necessary to transform continuous behavior with an infinite number of system states into discrete behavior.

Discrete abstractions involves in the first step the replacement of numeric variables by finite sets of intervals. Second, we can substitute differential equations through additional modes and components. Finally we must evaluate the model and eventually add primitive components that have not yet been considered.

Discrete Abstraction of Domains. We follow the approach of Sokolsky and Hong [10] who define finite sets of *landmarks* to replace numeric variables with enumerative variables. After screening the HS for all occurrences of numeric variables, a finite set $L = \{l_1, \ldots, l_n\}$ is proposed as landmarks for each variable. The condition that landmarks for a variable v are completely ordered as claimed in [10], is fulfilled by the majority of control systems in intelligent buildings, and taken for granted throughout this paper. [3] Therefore we can define a finite set of

[3] If this condition is not fulfilled, e.g. if for a numeric variable two landmarks were found, one is an expression and the other is a constant value in the range of that

disjoint intervals $\{(-\infty, l_1), [l_1, l_2), \ldots, [l_n, \infty)\}$ where $-\infty$ and ∞ represent the lower and upper bound of the domain. [4] After appropriate intervals have been specified for each numeric variable, all occurrences must be rewritten following the rules of interval arithmetics as described by Lunde [11].

Discrete Abstraction of Equations. For variables x_k that represent a dimension of the continuous state of the superordinate component of a HS architecture, discrete behavior needs to be added in order to represent x_k as part of the discrete state of the overall system. That means that for each interval of the discrete domain of x_k, as defined in the first step of the discrete abstraction, modes $q_1^k, \ldots, q_n^k$ must be added. For x_k that is updated in a hybrid automaton H through a set of equations $\dot{x}_k = F_q(x_k)$ an additional automaton H' is created to represent x_k as its discrete state. In the new automaton H' the discrete state of H is queried by the jump conditions and therefore required as input to H'.

Local variables x_l that represent a dimension of the continuous state of a hybrid automaton without being connected to other components are substituted by splitting modes q of the hybrid automaton H into modes $q_1^l, \ldots, q_n^l$ only if x_l is observable in the real system. Other local variables and equations for their continuous updating are removed from the hybrid automata. The HS model, after completing the discrete abstraction, has at this point been transformed into a model of a purely discrete system with a finite number of possible states. The automata that model the behavior of each component are now plain FSM without the extension of differential equations.

Completion of Components. System models that are created for different purposes may be decomposed differently. For example, a model that has been created to simulate the behavior of a system is unlikely to contain components that "do nothing" from the simulation point of view. But especially fault-prone subsystems must be detached from others and partitioned as a separate component in order to generate a reliable diagnostics model. This might seem redundant in many cases, but necessary in order to separate, for example, a broken sensor from other malfunctioning components.

Example: Thermostat (cont.) In the following we illustrate the discrete abstraction through the example of a thermostat described as a HS model in 3.1.

In the source model for the single-zone heating system we identify the landmarks $l_1 = \vartheta_{\text{opt}} - \theta$ and $l_2 = \vartheta_{\text{opt}} + \theta$ for the variable x_1 which represents the air temperature. We define the discrete domain for x_1 through the intervals

expression, total order is not assured. In such case, a structure type can be created to represent the discrete domain of the variable.

[4] The usage of inclusion or exclusion at the interval endpoints depends on the comparisons ($>$, $\geq$, $<$, $\leq$, $=$, $\neq$) used with the landmarks.

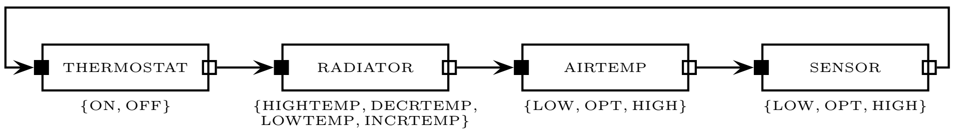

Fig. 9. Composition after Discrete Abstraction

$I_1 = (-\inf, \vartheta_{\text{opt}} - \theta]$, $I_2 = (\vartheta_{\text{opt}} - \theta, \vartheta_{\text{opt}} + \theta)$ and $I_3 = [\vartheta_{\text{opt}} + \theta, \inf)$. Representing the intervals $\{I_1, I_2, I_3\}$ with an enumeration $\{\text{LOW}, \text{OPT}, \text{HIGH}\}$, we must rewrite the occurrences of the variable in the model. For example, $x_1 \geq \vartheta_{\text{opt}} + \theta$ becomes $x_1 = \text{HIGH}$.

The composition of components in fig. 9 shows that we substitute x_1 globally through a new component AIRTEMP. Continuous updates of the numeric variable x_1 are substituted with an additional automaton in which each mode refers to an interval of the discrete domain of x_1. Transitions to activate these modes are controlled through a variable representing the discrete state of the RADIATOR component.

In our example the sensor that measures the air temperature that must be included additionally. Its behavior is modeled with the same modes LOW, OPT and HIGH as used already for the air temperature itself. For every state we add transitions in order to reach this state from every other state at the same condition.

4.3 Generation of Propositional Logic

In this transformation step we transform the FSM of each component into temporal logic propositions. The FSM consists of a set of modes M and a set of transitions $\varDelta$. Each transition $\delta \in \varDelta$ is guarded by a condition c_δ.

The next state (output) of an FSM is a function of the condition (input) and of the current state of the FSM. Since the FSM is deterministic [5], it can be reduced to a canonical form. For each component we introduce a state variable x and define the following rule for generating propositions:

$$\bigwedge_{\delta \in \varDelta} \left([x = \text{start}_\delta]_{t-1} \wedge [c_\delta]_t \Rightarrow [x = \text{end}_\delta]_t \right) \wedge$$
$$\bigwedge_{m \in \text{FSM}} \left([x = m]_{t-1} \wedge \neg \left(\bigvee_{\delta \in \text{out}_m} [c_\delta]_t \right) \Rightarrow [x = m]_t \right)$$

where $\text{start}_\delta / \text{end}_\delta$ is the mode from which the transition δ starts/ends, respectively. For a mode m, out_m denotes the set of all transitions that start from m.

[5] We assume that the extended automata of the HS was deterministic, therefore its discrete abstraction is deterministic as well.

Logical propositions generated with the above rule describe the nominal behavior of a component and are therefore logically equivalent to the system health $[h]_t$ at time t. The resulting model after this transformation step is a dynamic, structured system description (DSSD) as defined by Darwiche et al in [7]. We further assign values for the probability that the boolean health variable turns out to be true to instances of components.

Example: Thermostat (cont.) Applying the transformation rule for generation of temporal logic to the example, the following proposition is automatically generated for the thermostat component:

$$\begin{aligned}[h_{\text{THERMOSTAT}}]_t \Leftrightarrow\ & (([x = \text{ON}]_{t-1} \ \wedge\ [v = \text{LOW}]_t) \Rightarrow [x = \text{OFF}]_t) \wedge \\ & (([x = \text{OFF}]_{t-1} \ \wedge\ [v = \text{HIGH}]_t) \Rightarrow [x = \text{ON}]_t) \wedge \\ & (([x = \text{ON}]_{t-1} \ \wedge\ \neg[v = \text{LOW}]_t) \Rightarrow [x = \text{ON}]_t) \wedge \\ & (([x = \text{OFF}]_{t-1} \ \wedge\ \neg[v = \text{HIGH}]_t) \Rightarrow [x = \text{OFF}]_t)\end{aligned}$$

Some components are described through a behavior that is independent of the previous state. These behaviors can be reduced to propositions that do not consider the previous value of the state variable x as shown for the sensor component of the example:

$$\begin{aligned}[h_{\text{SENSOR}}]_t \Leftrightarrow\ & ([v = \text{LOW}]_t \Rightarrow [x = \text{LOW}]_t) \wedge \\ & ([v = \text{OPT}]_t \Rightarrow [x = \text{OPT}]_t) \wedge \\ & ([v = \text{HIGH}]_t \Rightarrow [x = \text{HIGH}]_t)\end{aligned}$$

4.4 Temporal Unfolding of Cycles

In this step we transform the architecture of the DSSD which is a directed graph depicting component interconnectivity into an acyclic architecture. We adopt the approach of temporal unfolding described in Darwiche et al [7] for diagnosis of discrete event systems. We clone the whole cyclic system in each time slice and then disconnect each cycle between the same two components. For the removed connections we reassign the output port of the component from which the connection started to the input port of the target component in the next time slice.

It conveys a certain meaning which connection between components is cut in each clone and reassigned to the next time slice. Therefore, for each cycle at least one component must be flagged as *entry-point* before the transformation can run automatically. Models of HVAC control systems, as a general rule, contain components that represent embedded systems, sensors with very short response time and components that represent unobservable physical phenomena, e.g. the temperature of a medium, with a rather long response time. We advise that the component of the cycle to be flagged as entry-point should be the component of the system that takes most time to react. In most cases this is a component describing a slowly changing physical phenomenon. This results from the assumption that events from the real system are taken during the operation of the component with the longest response time.

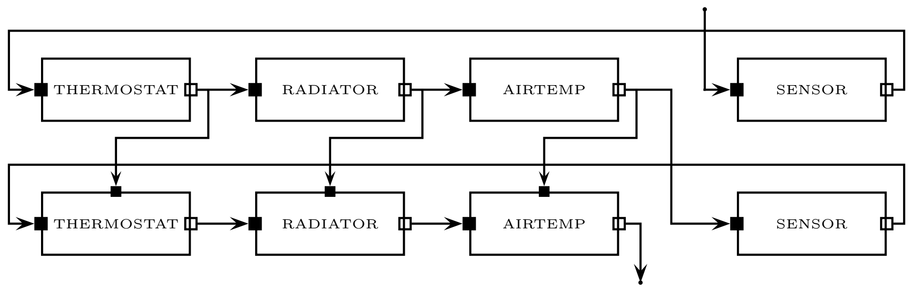

Fig. 10. Unfolded Composition

Example: Thermostat (cont.) We assume that observations of the system are provided by the thermostat that submits the sensor reading and the corresponding actuation command periodically or event-driven. Therefore we can choose either the air temperature or the sensor as entry point for the unfolding of components. Choosing a temporal unfolding of two time slices with the sensor as entry-point, the architecture of the unfolded model as shows in fig. 10 contains two instances of each component. Except the sensor component whose behavior could be reduced to an atemporal form as shown in 4.3, all components of the second time slice require the state variable of the instance in the first time slice as additional input.

5 Related Work

This section summarizes related work in three areas, hybrid systems transformations, auto-generation of diagnostics, and model transformation.

Due to the difficulty of analyzing hybrid systems models, a variety of transformations have been applied to these models, e.g., [12, 13]. We focus on abstraction into a discrete model, which is similar to [10], which describe a qualitative version of the Charon modeling language and the transformation from a hybrid-systems model to its discrete abstraction. Our work is different in that the output model is a diagnosis model.

This work is related to work on the auto-generation of diagnostics. Our approach is different from most existing approaches, which generate models from diagnosis component models rather than abstract a diagnosis model from a more complex hybrid systems model. Among this class of approaches, we examine two particular papers. [14] describes a methodology of diagnostics generation for car subsystems, based on the assumption that models of car systems can be automatically composed from model libraries. [15] presents an approach to build models for temporal MBD for VHDL designs, in which they model the temporal behavior of VHDL by temporal unfolding of process execution, which is then converted into a component-connection model which can be used by a MBD reasoner.

Our work applies the theory of model transformation in a novel way. We propose an exogenous translation, for which the source and target are from different meta-models, and our target meta-model has the semantics of a temporal diagnosis model. This target model is unique in the model transformation literature, and it also extends prior work in which the target model was a static diagnosis model [9].

6 Conclusions and Future Work

This article has described an implemented framework for transforming hybrid systems models to temporal propositional logic diagnosis models. Our approach contributes to research in model-driven development of complex systems by extending model consistency up to models for diagnostics. We have illustrated the transformation process using the domain of HVAC, with an on/off thermostat control system.

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.

We hope to extend this work in a variety of ways. First, we plan to extend the class of diagnosis models from the current approach, which describes only the correct behavior of a system, to models that define explicit fault behaviours. Second, we want to extend the current transformation rules, which only apply to systems with observable or unobservable components, to rules that can deal with observable and unobservable transitions in each state machine.

Acknowledgment

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

References

1. Henzinger, T.A.: The theory of hybrid automata. In: LICS '96: Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science, Washington, DC, USA, IEEE Computer Society (1996) 278
2. Provan, G., Ploennigs, J., Boubekeur, M., Mady, A.: Using building information model data for generating and updating diagnostic models. Conf. on Civil Engineering and Computing (August 2009)
3. Mens, T., Czarnecki, K., Gorp, P.V.: A taxonomy of model transformations. In Bezivin, J., Heckel, R., eds.: Language Engineering for Model-Driven Software Development. Number 04101 in Dagstuhl Seminar Proceedings (2005)
4. Fagin, R., Kolaitis, P.G., Popa, L., Tan, W.C.: Composing schema mappings: Second-order dependencies to the rescue. ACM Trans. Database Syst. **30**(4) (2005) 994–1055

5. Brusoni, V., Console, L., Terenziani, P., Dupr, D.T.: A spectrum of definitions for temporal model-based diagnosis. Artificial Intelligence **102** (1998) 39 – 79
6. Emerson, E.: Temporal and modal logic. Handbook of theoretical computer science **8** (1990) 995–1072
7. Darwiche, A., Provan, G.: Exploiting system structure in model-based diagnosis of discrete-event systems (1996)
8. Behrens, M.: Model transformation: Hybrid systems to temporal diagnosis models. Technical report (2010) `http://www.cs.ucc.ie/~mb20/`.
9. Behrens, M., Provan, G., Boubekeur, M., Mady, A.: Model-driven diagnostics generation for industrial automation. In: Proc. 7th IEEE International Conference on Industrial Informatics. (June 2009)
10. Sokolsky, O., Hong, H.S.: Qualitative modeling of hybrid systems (2001) Presented at the Monterey Workshop on Engineering Automation for Computer Based Systems. `http://repository.upenn.edu/cispapers/87`.
11. Lunde, K.: Object-oriented modeling in model-based diagnosis. In: Proceedings of the First Modelica workshop, Lund, Sweden (2000)
12. Antoulas, A., Sorensen, D., Gugercin, S.: A survey of model reduction methods for large-scale systems. Contemporary Mathematics **280** (2001) 193–219
13. Mazzi, E., Vincentelli, A., Balluchi, A., Bicchi, A.: Hybrid system reduction. In: 47th IEEE Conference on Decision and Control, 2008. CDC 2008. (2008) 227–232
14. Dressler, O., Struss, P.: Generating instead of programming diagnostics. In: 25 Jahre Elektronik-Systeme im Kraftfahrzeug, Haus der Technik Fachbuch 50. (2005) 159–169
15. Köb, D., Peischl, B., Wotawa, F.: Debugging vhdl designs using temporal process instances. In: IEA/AIE'2003: Proceedings of the 16th international conference on Developments in applied artificial intelligence, Springer Springer Verlag Inc (2003) 402–415