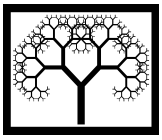
Paper 94



Proceedings of the Twelfth International Conference on
Civil, Structural and Environmental Engineering Computing,
B.H.V. Topping, L.F. Costa Neves and R.C. Barros, (Editors),
Civil-Comp Press, Stirlingshire, Scotland

Using Building Information Model Data for Generating and Updating Diagnostic Models

G. Provan[1], J. Ploennigs[2], M. Boubekeur[1], A. Mady[1] and A. Ahmed[2]
[1]Computer Science Department
[2]Department of Civil and Environmental Engineering
University College Cork, Ireland

Abstract

We show how to use a Building Information Model (BIM) to structure fault detection and diagnostics (FDD) models for building applications, and also how BIM data can be used for learning model parameters for updating the FDD parameters following building commissioning. We propose an approach for generating FDD rules using a generic meta-model together with the data defined in a BIM or Building Management System design database. Our meta-model is a detailed model that identifies a key set of properties of a system, e.g., connectivity and functionality of the devices that comprise the system. We then show how we can tune the parameters of such FDD rules using data from a building simulation model, or from actual building data collected in a data warehouse. We illustrate our approach using a lighting systems model within an intelligent building application.

Keywords: fault detection and diagnostics, building information model, parameter estimation.

1 Introduction

A modern Building Information Model (BIM) collects various data during the operational lifetime of a building. From design time, data for the building geometry, the HVAC design, and the building automation systems can, for example, be stored using a standard format like IFC [1]. The actual building performance measures, which are collected during runtime are stored in databases or data warehouses of the building management system (BMS). This broad variety of data cannot only be used for analysis tasks like energy simulation, building performance analysis, facility management, and diagnosis, but also allows new holistic design approaches of these tool chains.

For example, a Fault Detection and Diagnostic (FDD) [2] tool for a specific build-

ing is usually created from modules that implement common functions, e.g., threshold detection, anomaly detection, condition monitoring, control cycle diagnosis, etc. One key drawback to generating model-based fault detection and diagnostics for large systems is the cost of constructing appropriate system models. There are two sources to the difficulty of model construction. First, there is the need to create FDD modules for each element to monitor, and second, there is the need to tune the FDD parameters to the actual building application.

Creating the initial FDD models requires comprehensive information of the building, the HVAC system, the data points in the building automation system, the data bases or data warehouses of the building management system — just the kind of information available in a modern BIM like the one introduced in the paper.

Tuning the FDD parameters to the actual building application is a second difficult task. This is necessary to ensure that we minimize false alarms, yet still catch real faults. This task is more general, as the entire BMS needs to be set-up and customised for each specific project, and this laborious task is one of the main deficits of current BMSs [3, 4].

We assume that parameter estimation will occur in two phases: (1) initialization, and (2) tuning. During the initialization phase, we compute the initial values of the parameters using data simulated by a model. In our case, we use the lighting model described in Section 3. Using a simulation model speeds up the process of initialization, and it allows us to simulate faulty data (just by setting fault conditions in the model to *true*); in contrast, to obtain faulty data in a real building would require considerable work, and potentially would entail destructive testing. During the tuning phase, we take the initial rules and fine-tune the thresholds using data collected from the real building, as stored in the data warehouse.

The main contribution of this work is showing how to use the BIM to structure the diagnostics models, and also how BIM data can be used for learning model parameters for updating the parameters following building commissioning. We propose an approach for reducing the need to construct multiple models by using a meta-modeling and model-transformation approach. We generate FDD rules using a generic meta-model, which is a detailed model that identifies a key set of properties of a system. For example, we describe a meta-model that captures, among other things, connectivity and functionality of the devices that comprise a system. Given such a generic meta-model, we describe how how we can auto-generate an application-specific model, one defining FDD rules, through the use of building design data, such as the data defined in a BIM or Building Management System design database. We can then tune the parameters of such FDD rules using data from a building simulation model, or from actual building data collected in a data warehouse.

The article is structured as follows. Section 2 describes our FDD auto-generation software architecture. Section 3 introduces an example that we use throughout the article to illustrate our approach. Section 4 describes the notion of BIM that we adopt in the article. Section 5 outlines our FDD rule-generation approach. Section 6 describes our parameter estimation methodology, given FDD rules. Finally, we summarise our

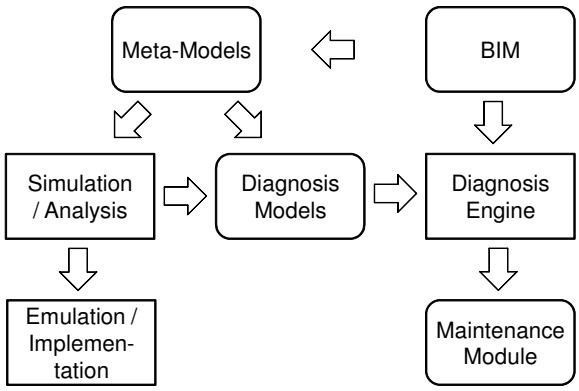


Figure 1: System Architecture and Design Flow

contributions in Section 7.

2 System Architecture

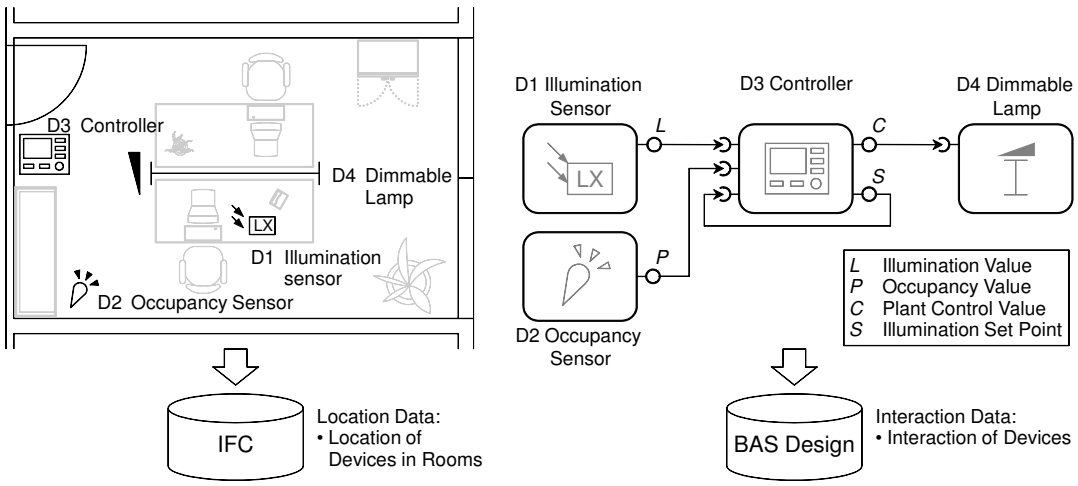
The proposed fault detection and diagnosis approach is described in Figure 1. As mentioned earlier, our approach is based on a multi-modelling platform that auto-generates various models dedicated to different use. Starting from the design information in the BIM like device lists of the HVAC and building automation system a meta-model is abstracted. From this meta-model further models for simulation and analysis are generated, using models such as hybrid systems models for simulation, and threshold detection approaches for diagnosis. The simulation models may then be translated into embedded code to be deployed in the wireless sensor/actuator network (WASN). The diagnosis models are linked with a diagnosis engine that detects faults in the actual measurements of the automation system which are collected in the data warehouse part of the BIM. The threshold that has been violated and the related zone are communicated to the diagnosis engine in order to isolate the faulty component using a model based diagnosis techniques. The maintenance component takes the output of the diagnosis engine to schedule maintenance activities.

We divide our diagnosis-generation approach into two parts: (1) generation of fault detection rules, which will indicate anomalous conditions; and (2) generation of diagnosis models, which isolate the faults given the anomalous observations from (1). In this article, we focus on the auto-generation of fault detection rules; the details of diagnosis model-generation are presented in [5]. To monitor the condition of a zone in a building, one needs to have some representation that would indicate when key parameters are exceeded.

Two key issues for such a representation are: (1) the structure of the fault detection rules, and (2) the values of the thresholds. We assume that we will adopt a standard template for the threshold detection, and use the BIM to auto-generate the rules parameters and the building-specific parameter thresholds.

3 Example: Lighting in a Building

We illustrate our approach using an example for a lighting control in Figure 2. We assume we have a room with an illumination sensor D1, which measures the light level E and a occupancy sensor D2 detecting the occupancy P. A controller D3 is attempting to maintain a setpoint light level of W in the room using the dimmable lamp D4 with the dim value D, if occupants are present; any internal lighting is off if no occupants are present.



(a) Room with the devices

(b) Device interaction design

Figure 2: Example scenario of a light control

The Fault Detection and Diagnosis has the aim to detect and analysis faults in the lighting system for this example. An illustration is given in Section 5. For example, this the fault should be detected, that occupants are present in the room, but the light is not on, even if it is to dark.

4 BIM

The term Building Information Model / Modeling (BIM) is defined broadly, so its meaning in this paper should be clarified. We see a *Building Information Model* as the digital collection of a building's data during its life cycle and *Building Information Modeling* as the process of creating this data. A Building Information Model usually comprises the geometrical model of a building, the quantities and properties of its components as well as related design information. For this kind of information standard exchange formats like Industry Foundation Classes (IFC) [1] or aecXML [6] exist that aim to enable tools to exchange and reuse this information over the building life cycle.

Also other data relevant in the building life cycle can be seen as part of the BIM using the above definition. For example, dedicated design information can be stored

and used in a *Building Automation Systems* (BAS). The building automation system implements the monitoring and control of the Heating, Ventilation, Air-Conditioning (HVAC) system, lighting, access control, etc. A BAS can contain several thousands of devices communicating over standard network technologies [7]. Due to their size, these systems are designed using specialized design tools. These design tools share the same databases to store their designs. Open-design databases have been developed, e. g., [8], to allow the different specialists working on an integration project to exchange design information; this is similar to the development of BIM in the general construction domain. Designs for building automation systems can also be exported to IFC [9] from such databases.

A basic example for the BAS design of the lighting control loop is given in Figure 2(b). The design shows the individual devices and their datapoints. In the design tools, these datapoints are logically connected to form the application system. For example, the connection of the illumination sensor (D1) to the controller (D3) means that the illumination level E is transmitted to the controller; the connection of the setpoint value S from the controller output to its own input indicates that the controller possesses a user interface that enables setting the illumination setpoint. The design database of the BAS contains all the information about the devices, their interaction, and the data exchanged.

Also, dynamic data is typically collected during the operation of a building. The data measured in the BAS from sensors, controllers and actuators is usually collected in a database in the *Building Management Systems* (BMS). This data is analysed for online Fault Detection and Diagnosis, among other uses.

Recent work has extended the simple DataBase concept to a *Data Warehouse* to support building performance analysis already on the data storage level [10]. Therefore, the data warehouse also contains information about the building structure (floors, rooms), the location of sensors in the building, etc. This allows semantic addressing of queries like: “Compute the energy consumption of the lights in room 123 for march.”. As parts of this information is contained in IFC, a parser has been implemented to allow the importation of information into the data warehouse. More information about the data warehouse can be found in [11].

5 Generating an FDD system from a BIM

This section describes how we create an FDD system from the BIM. The main steps of the generation are shown in Figure 3, and will be detailed in the course of this section. The approach is demonstrated using an example of *threshold detection*, which is the simplest approach for fault detection. We show how rules to generate thresholds for certain parameters. If the resulting rules are triggered by a threshold being exceeded, a fault is instantiated.

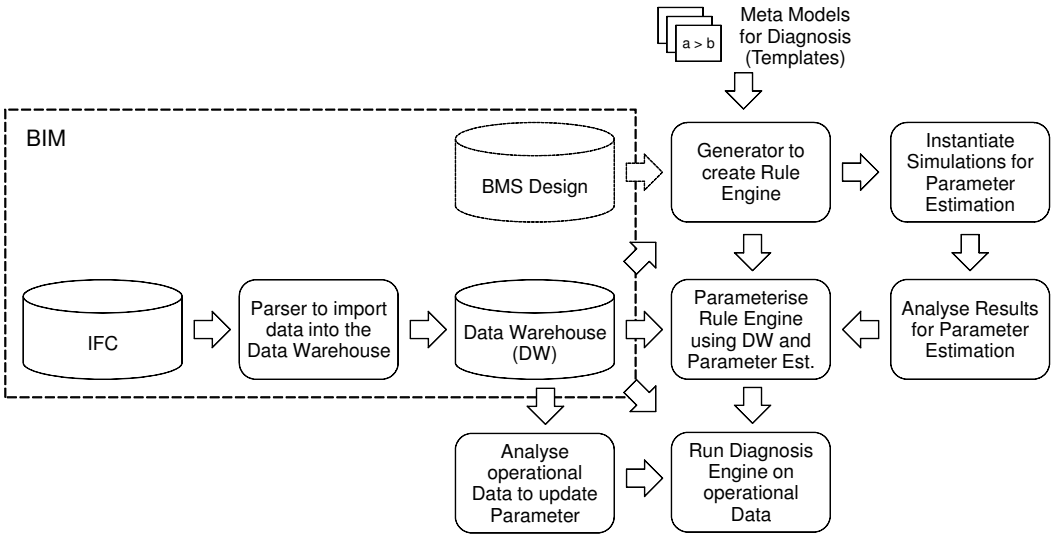


Figure 3: Model transformation process

5.1 Rule Syntax

We first describe the syntax we adopt for our rules, and then we outline how we use templates to instantiate rules according to this syntax.

Threshold detection rules have a structure such that a mode-value, e. g., $M = bad$, logically corresponds to sensor/actuator settings, i.e., $(M = bad) \Leftrightarrow (Act = off)$ denotes that an actuator *Act* is stuck off when its failure-mode status M is bad. Examples of such rules for the lighting domain include:

$$[(M_A = bad)] \Leftrightarrow [(P = f) \wedge (D > \delta_1)] \tag{1}$$

$$[(M_A = bad)] \Leftrightarrow [(P = t) \wedge ((W - E) > \delta_2)] \tag{2}$$

$$[(M_A = bad)] \Leftrightarrow [(P = t) \wedge ((E - W) > \delta_3)] \tag{3}$$

Here the first rule uses a threshold δ_1 to indicate that the internal lighting is incorrectly turned on when no occupants are present. The cause can be a erroneous controller or information about the device status. The second rule signals a fault that the light level is to low when a person is present. It may have its cause again in the controller, or incorrect information, but also in a broken actuator (lamp). The third rule detects the fault if the light level exceeds the set-point W more than some threshold δ_3. The causes are manifold again.

These generic rules will have to be instantiated with real sensors and actuators defined in the BIM. This means that for the three rules above the queries to access the values for the occupancy P, illumination E and set-point W in the data warehouse need to be defined. Also, the parameters $(\delta_1, \delta_2, \delta_3) \geq 0$ require values. These values can be fixed if the rule if the value does not need to be changed; the values may be exported from other design information (e. g., value ranges for rule (4) are usually

defined in the BMS design); or alternatively, we will need to use some parameter-estimation techniques for actual data.

5.2 BIM to Rule Template Transformation

The first step of setting up an FDD system is to partition the system into small elements that can be handled separately, in order to reduce the complexity. This requires understanding the elements of the system, and their interaction. Consider the example of the light control loop of Figure 2, which consists of the illumination sensor, the occupancy sensor, the controller and the actuator/lamp. The combination of different values from different devices in the rules (1) to (3) already demonstrate that the fault detection requires information about this interaction.

However, in a building with several rooms equipped with such a light control, defining the rules for each room is a laborious task. It requires: (1) checking in the BAS design which devices are related to each other, especially if there are multiple sensors or actuators in a room; (2) identifying each device and datapoint in the BMS database or data warehouse, and their association to the rules; and (3) defining the parameters of the rules.

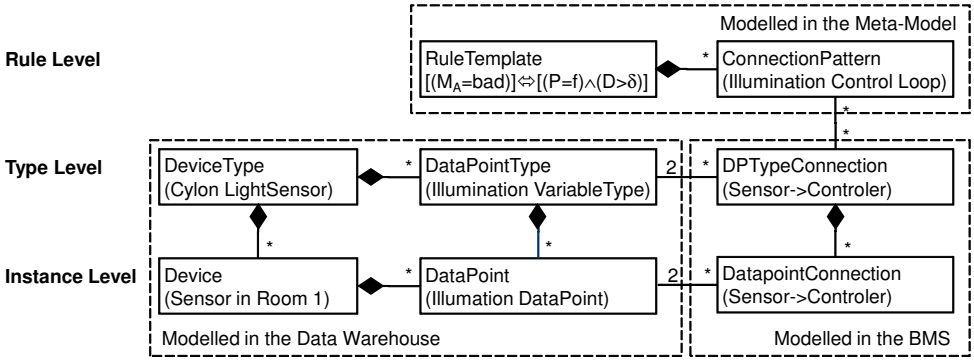


Figure 4: UML class diagram of the data model used for the generation of rules

Figure 4 shows the data model used to generate such rules automatically. A `RuleTemplate` applies to a `ConnectionPattern`, which defines a group of interacting devices via their datapoints, e. g., for the "light control loop". To allow the definition of such patterns, not only for specific devices but in a generic way, the pattern is defined by a set of `DPTypeConnection` of abstract `DataPointTypes`, like the connection of any illumination datapoint output to any light controller input. The abstract `DataPointTypes` belong to abstract `DeviceTypes`. If a specific illumination sensor and controller are identified as belonging to such a generalized type, then the pattern can be identified and applied.

The generator therefore uses the input from different sources of the BIM. The devices and their abstract device types are modeled in the data warehouse. The general types are identified during the import from IFC into the data warehouse, using either

the name of the manufacturer and device classes, or as modeled later by hand in the data warehouse.

If devices are located in the same room match a pattern, then we can deduce that they also communicate with each other. As this assumption is not unambiguous, especially if the room has multiple sensors, we can export the interaction directly from a BMS design, or use approaches like [9] to export them via IFC.

5.3 Rule Templates

We describe some templates that are relevant to this domain, to illustrate our approach. In the lighting model, we focus on the sensors and actuators used to control the lights and blinding. We examine several classes of rules: (1) sensor rules, (2) actuation rules, (3) delay rules, and (4) sensor/actuator rules.

5.3.1 Sensor Rules

Sensor out of range This rule covers the case when a sensor ($S \in \{E, P\}$) is registering a value outside of its normal operating range. So if the minimum and maximum values of the nominal range are S_{min} and S_{max} respectively, we have a rule such as

$$[M_S = OK] \Leftrightarrow [S_{\text{min}} \leq S \leq S_{\text{max}}]. \tag{4}$$

No sensor output This rule covers the case when a sensor outputs no data. In this case we have the rule that the timestamp $T(S)$ of the last sensor value is older than τ_S, which is a multiple of the common sampling interval:

$$[M_S = OK] \Leftrightarrow [T(S) \leq \tau_S]. \tag{5}$$

Sensor mismatch This rule covers the case when two or more sensors, which should correspond on their outputs, register inconsistent outputs. In this case we assume that the sensors will agree on their outputs within a tolerance δ_t:

$$[(M_{S_1} = OK) \wedge (M_{S_2} = OK)] \Leftrightarrow [|S_1 - S_2| \leq \delta_t]. \tag{6}$$

5.3.2 Actuation Rules

We must create rules for detecting whether the correct controls have been initiated, like rule (1).

5.3.3 Delay Rules

Often the delay in systems need to be considered in fault detection as well. For example, a person changes the set-point of the control from 500 lx to 800 lx. But, the light controller changes the dim level of the lamps only slowly, which is a “comfort”

(effect) feature by some controllers, and the light needs several seconds to adjust to the new set-point. To avoid to detect such a normal change as fault, the parameter δ_2 in rule (2) for example should be at least larger then the 300 lx set-point difference. This is quite large, especially if in stable state (after the light adjusted) a light variance of maximal 100 lx should be ensured. For such cases, rule (2) is extended by a delay parameter τ that tolerates small variances within τ, but creates a fault if the difference stays longer than δ_2

$$[(M_A = bad)] \Leftrightarrow [(P = t) \wedge ((W - E) > \delta_2) \text{ for } (t > \tau)]. \tag{7}$$

5.3.4 Sensor/actuator rules

The typical scenario that we address is when the desired actuation setting disagrees with the sensor output. For example, this might arise if the light actuator D is on $(D > 0)$, but the sensor E registers a very low light level.

$$[(M_S = OK) \wedge (M_A = OK)] \Leftrightarrow [(E \leq \delta) \wedge (D > 0)]. \tag{8}$$

For this rule, violation implies that the light actuator and sensor cannot both be nominal.

The next rule covers the case where no people are detected, and the actuator must be off (in nominal conditions):

$$[(M_S = OK) \wedge (M_A = OK)] \Leftrightarrow [(P = f) \wedge (D = 0)]. \tag{9}$$

6 Threshold Parameter Estimation

This section describes how we estimate the threshold parameters for our rules.

6.1 Parameter Estimation Process

In the following, we will characterise our estimation of parameters for FDD rules as a task in system identification [12]. Assume that our system can be defined using $\boldsymbol{y} = \phi(\boldsymbol{u}, \boldsymbol{\theta})$, where $\boldsymbol{y}$ is the vector of output signals, $\boldsymbol{\theta}$ is the vector of parameters, and $\boldsymbol{u}$ is the input vector. Further, assume that we model our system using $\hat{\boldsymbol{y}} = \hat{\phi}(\boldsymbol{u}, \boldsymbol{\theta})$.

If we assume that the actual system has output $\boldsymbol{y}$, and the model has output $\hat{\boldsymbol{y}}$, our task is to determine suitable values of the parameters $\hat{\boldsymbol{\theta}}$ such that $\hat{\phi}(\boldsymbol{u}, \boldsymbol{\theta}) = \phi(\boldsymbol{u}, \hat{\boldsymbol{\theta}})$ [12]. To accomplish this, we assume that a *residual signal* is obtained as a difference between the outputs of the system and the model, i. e., $\boldsymbol{r} = \boldsymbol{y} - \hat{\boldsymbol{y}}$. The simplest model-based residual generation procedure reduces to system identification, and entails checking the norm of the residual signal $\| \boldsymbol{r} \|$.

In the first instance, we assume that we are generating rules within a building commissioning scheme. In this case, since there is no building from which to collect data,

we use a simulation model to act as the "real system". The "model" corresponds to the specification of the set of rules. We aim to estimate the parameter values for the rules, and we do this by using the simulation model to generate a set I instances of data, and then conducting data mining on I to estimate the parameters $\hat{\boldsymbol{\theta}}$.

Once the commissioning process has estimated an initial set of FDD rules, we can then move into a parameter-tuning phase, in which we assume that we obtain data from the actual building sensors/actuators (termed the system $\boldsymbol{y}$), and the model consists of the set of rules. In this phase we update the parameters estimated during the inference for the "commissioning process".

We now summarise the approach we use for parameter estimation. We have two competing objectives: (1) minimise the norm of the residual signal $\| \boldsymbol{r} \|$ for each rule, and (2) distinguish a *normal operating region* using bounds on the parameters, in order to avoid rules generating false alarms by having point-estimates for key parameters. For example, if we want to control the light level to a setpoint α, we want to include some range around α, i. e., $[\alpha - \epsilon_1, \alpha + \epsilon_2]$, which defines normal operating conditions (given noise in sensing, etc.). In this case, we want a general mechanism to estimate the values of $\alpha,\ \epsilon_1$, and ϵ_2.

During the first phase of parameter estimation, corresponding to "building commissioning", we take cases relating to the normal operation involving the setpoint α, and estimate its mean μ and standard deviation σ. Our decision then involves the interval around α which defines normal operation. In implementing such a setpoint-based rule, we assign to ϵ_1 and ϵ_2 either a 3σ limit, which defines about 99.7 % of the samples as normal, or a 4σ limit, which defines about 99.99 % in terms of normality.

During the second phase of parameter estimation, corresponding to "parameter tuning", we take cases of real data relating to the normal operation involving the setpoint α, and repeat the process described for phase 1. This parameter tuning can take place periodically during operation of the real system, using data stored in the data warehouse.

6.2 Data Generation using a Simulation Model

Based on the lighting control shown in Figure 2, we have implemented a simulation model using a hybrid systems representation [13]. Hybrid systems representations describe both the discrete and continuous aspects of a simulation, which are critical to building systems. In our example, the light diffusion equations follow a continuous process, whereas there are discrete outputs from sensors about presence of people in a zone.

The tool that we have adopted for this modeling is Charon [14]. Charon is a hybrid modeling language that supports the construction of hierarchical, component-based hybrid systems models. Each primitive system component is called an agent; concurrency of agents means that agents can communicate with other agents asynchronously. Charon describes behavior in an agent using the formalism of a mode, which is a hierarchical hybrid state machine that can have submodes and transitions connecting

them.

Charon runs simulations by alternating a series of discrete and a continuous steps for each agent. A discrete step consists of a series of transitions from the active atomic mode to another atomic mode. This flow of control is determined through the assignment of mode variables, which are determined by mode invariants, transition guards, or transition actions. We assume that the continuous variables evolve according to algebraic and differential constraints governed by the active modes, and this continuous evolution may also trigger discrete mode changes.

Given a set of initial conditions, we can run a simulation in Charon, from which we can sample data at pre-defined time-points to generate data. By appropriately setting the modes in Charon, we can simulate either normal or faulty behaviours, thus generating data for learning both classes of operating condition.

6.3 Example

In this section we illustrate through our simple lighting system the data mining analysis when considering the simulation results obtained using a simulation tool called Charon. Charon is a high-level language for modular specification of multiple, interacting hybrid systems, and was developed at the University of Pennsylvania [14]. The toolkit distributed with Charon is entirely written in JAVA, providing many features: a Graphical User Interface (GUI), a visual input language, an embedded type-checker, and a complete simulator.

In the following, we first present the Charon model for the lighting system and then the data mining analysis on the simulation results where erroneous behaviour is injected.

6.3.1 Charon Model for the Lighting System

The example in Section 3 has been modelled and simulated using Charon tool [14]. The model computes lighting control levels, based on sensor data for occupancy and ambient light.

The basic Charon model for our lighting system consists of a system agent with two concurrent control agents: a simple controller for the interior light (`SwitchControl`) and another simple controller for the blinding (`BlindingControl`). Each control agent contains one top-level mode to describe its behaviour. Figure 5 shows the architectural and behavioural hierarchy of the simple lighting system.

The top-level mode of the `SwitchControl` agent contains transitions between two submodes: `on` and `off`. Both modes are empty, and for our simple example the behaviour could be modelled with only one submode (in which case transitions start and end in the same submode).

In the Charon model the light level in the room is influenced by two parameters: the external light (regulated by the blinding controller) and the artificial internal light.

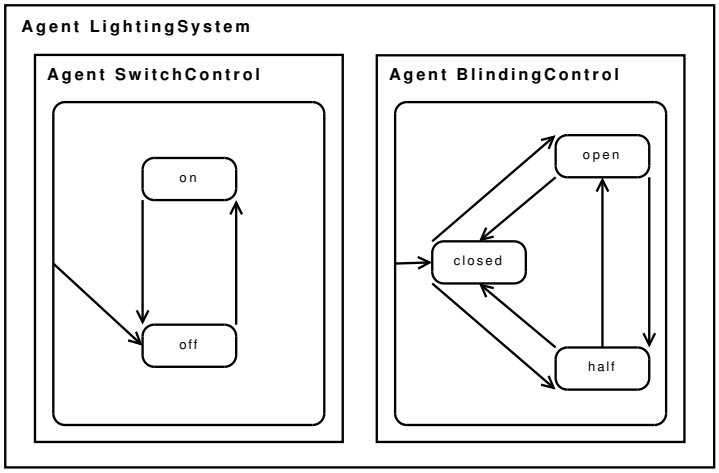


Figure 5: Charon model of a simple lighting system

The external light intensity is additionally influenced by several disturbances. When considering a correct behaviour for the overall system, we obtain the result depicted in Figure 6. This shows shows how the controller changes the internal light of the lamps, depending on the external light intensity, in order to maintain a set-point of 500 lx, which is denoted by 0.5 on the y-axis of Figure 6.

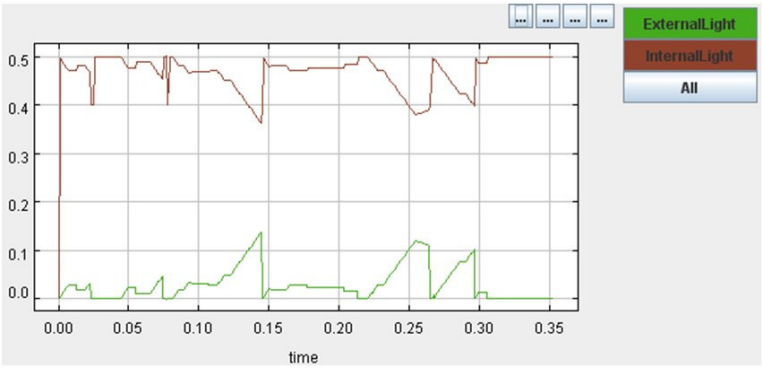


Figure 6: Internal Light vs. External Light (in 1000 lx)

6.3.2 Parameter Estimation from the Simulation

The parameter estimation is demonstrated for the δ_2 parameter of rule (2). The parameterisation must trade off two objectives, by ensuring: (1) a high detection rate of faults (small δ_2), and (2) a low rate of false alarms (large δ_2). To find a good set of parameters within this trade-off, it is usually necessary to analyse the signal behavior.

Figure 7 shows an example of the simulation results of a faulty light control. The set-point W is constantly at 500 lx, as the office is occupied. The controller tries to hold this value, given various influences, such as a noisy light level (e. g., see time 240–280 s). However, the illumination E drops to 400 lx from time to time, due to a

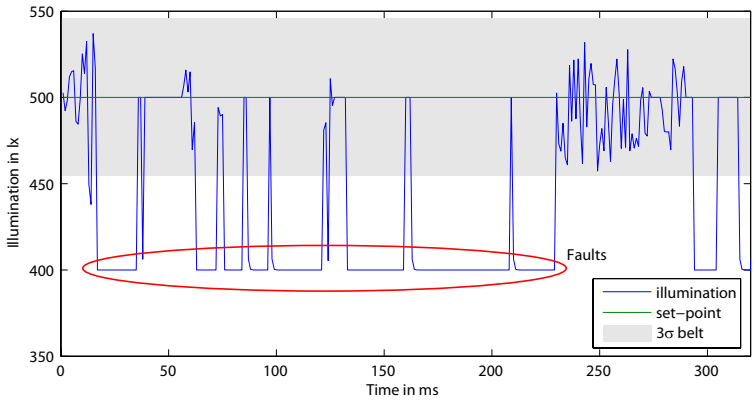


Figure 7: Simulation results for a faulty light control

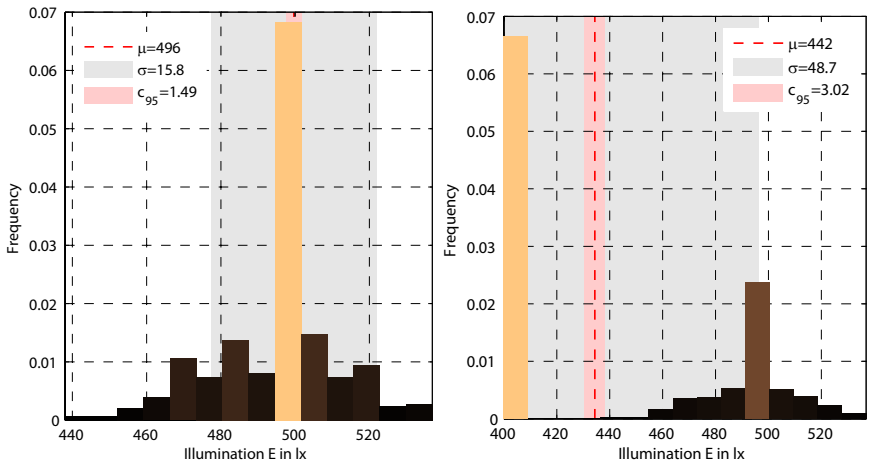
loose contact at a lamp, an error at the controller, or at the sensor. These faults should now be detected with rule (2), but the noise should be ignored. Analysing the signal visually, it is clear that δ_2 should be around 75 lx for this example. However, our objective is to have a general mechanism to automatically detect the parameter δ_2, in order to simplify the set-up process during design time and to optimize the parameters for all individual rooms of a building, given their specific influences and dynamics during run time.

Therefore, a fault-free sample from the simulation is used to analyse the parameters of the noise. Figure 8(a) shows the histogram of the fault-free behavior. In most cases the controller is able to hold the set-point at 500 lx, which results in the peak at 500 lx. The noise is normally distributed, and the signal has a standard deviation of σ =15.8 lx. This standard deviation is used to parameterise δ_2 as introduced in Section 6.1. δ_2 is set to a 4σ limit (63.2 lx), as that would tolerate 99.99 % of an ideal normally-distributed signal as "nominal" i. e. not faulty.

Figure 8(b) shows in contrast the histogram of the faulty behavior in Figure 7. The peak at 400 lx is caused by the frequent faults and results also in a changed mean value μ and a increased standard deviation σ. These changed properties of the distribution are also the reason why for the rule parameterisation of a fault-free simulation run is used to avoid the adaption of the parameters to the faulty behaviour.

Using an approach similar to this building commissioning approach, we plan to create a module that periodically analyses the operational data, in order to tune the rules to the properties of the real data. To avoid learning incorrect behavior, the analysis process should only if the sampled period creates a set of error-free rules. Then the data form that period can be analysed, and the parameters can be automatically updated if they differ, but nor significantly, from the last value; with significant differences,s the user need to be informed about the change of the system behavior and the resulting rule parameters.

This given example for the parameter estimation of the light control is quite easy, as it is a very fast process and delays can be neglected. For slower processes with a significant delay, e.g., a temperature control, the paramterisation is more complex,



(a) Fault-free behavior (b) Faulty behavior

Figure 8: Histogram of the illumination values E from the simulation

as the steady state and change state need to be considered, as discussed in rule (7). Approaches like [15] allow the identification of these states, and the specific parameters, respectively. The time delay τ in rule (7) can then also set in relation to the step-response time of the system.

7 Summary and Conclusions

We have described a methodology for automatically generating FDD rules from previously defined building information model, such as the design information contained in IFC, a BMS and/or the runtime data stored in data warehouse, thereby reducing the need to construct FDD models by hand. Our proposed approach consists of two phases: first we generate the FDD rules based on rule templates, and second we tune the parameters in the rules using data from a building simulation model, or from actual building data collected in a data warehouse. In the first phase we use building-system meta-models, which denote device functionality and connectivity, in conjunction with BIM/BMS specifications for an actual building, to generate the rules. In the second phase, we tune the rule parameters, first using data generated from a building simulation model; we can then update the parameters' values on a continuous basis using data collected from the building, e. g., as stored in a data warehouse.

Acknowledgement

This work was funded by SFI grant 06-SRC-I1091. Joern Ploennigs further wants to thank the Humboldt-Foundation and the German BMBF for their support.

References

[1] International Alliance for Interoperability, *Industry Foundation Classes - IFC2x Edition 4 alpha version*, 4 α edition, June 2008.

[2] S. Katipamula, M.R. Brambley, "Methods for Fault Detection, Diagnostics, and Prognostics for Building Systems - A Review, Part I", *HVAC&R Research*, 11 (1): 3–25, Jan. 2005.

[3] S. Wang, Z. Xu, J. Cao, J. Zhang, "A middleware for web service-enabled integration and interoperation of intelligent building systems", *Automation in Construction*, 16(1): 112 – 121, 2007, ISSN 0926-5805, CAAD Futures, 2005.

[4] A. Malatras, A. Asgari, T. Baugé, M. Irons, "A service-oriented architecture for building services integration", *Journal of Facilities Management*, 6(2): 132–151, 2008.

[5] M. Behrens, G. Provan, M. Boubekeur, A. Mady, "Model-Driven Diagnostics Generation for Industrial Automation", in *Proc. INDIN 2009*, July 2009.

[6] Bentley Systems, *aecXML Preliminary Specification*, Working draft 0.81 edition, Aug. 1999.

[7] W. Kastner, G. Neugschwandtner, S. Soucek, H.M. Newman, "Communication Systems for Building Automation and Control", *Proceedings of the IEEE*, 93 (6): 1178–1203, 2005, ISSN 0018-9219.

[8] J. Ploennigs, M. Neugebauer, K. Kabitzsch, "Automated model generation for performance engineering of building automation networks", *Int. Journal on Software Tools Technology Transfer (STTT)*, 8(6): 607–620, Nov. 2006, ISSN 1433-2779.

[9] A. Karavan, M. Neugebauer, K. Kabitzsch, "Merging Building Automation Network Design and IFC 2x Construction Projects", in *22nd Conf. on Information Technology in Construction*, pages 575–581. Dresden, Germany, 19.–21. July 2005.

[10] M. Keller, J. O'Donnell, K. Menzel, M. Keane, U. Gokce, "Integrating the specification, acquisition and processing of building performance information", *Tsinghua Science and Technology*, 13(1): 1–6, 2008.

[11] A. Ahmed, K. Menzel, J. Ploennigs, B. Cahil, "Aspects of multi-dimensional Building Performance Data Management", in *16th Workshop of the European Group for Intelligent Computing in Engineering*, July 2009, submitted.

[12] L. Ljung, "Issues in system identification", *IEEE Control Systems Magazine*, 11 (1): 25–29, 1991.

[13] G. Labinaz, M. Bayoumi, K. Rudie, "A survey of modeling and control of hybrid systems", *Annual Reviews in Control*, 21: 79–92, 1997.

[14] R. Alur, R. Grosu, Y. Hur, V. Kumar, I. Lee, "Modular specification of hybrid systems in Charon", in *3rd Int. Workshop on Hybrid Systems: Computation and Control*, 2000.

[15] V. Vasyutynskyy, J. Ploennigs, K. Kabitzsch, "Passive Monitoring of Control Loops in Building Automation", in *Fet - 6th IFAC Int. Conf. on Fieldbus Systems and their Applications*, pages 263–269. Puebla, Mexico, 14.–15. Nov. 2005.