

Virtual Testing for Smart Buildings

Julien Bruneau, Charles Consel, Marcia O'Malley, Walid Taha, Wail Masry Hannourah

▶ **To cite this version:**

Julien Bruneau, Charles Consel, Marcia O'Malley, Walid Taha, Wail Masry Hannourah. Virtual Testing for Smart Buildings. IE 2012 - 8th International Conference on Intelligent Environments, Jun 2012, Guanajuato, Mexico. hal-00715753

HAL Id: hal-00715753
https://inria.hal.science/hal-00715753

Submitted on 9 Jul 2012

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Virtual Testing for Smart Buildings

Julien Bruneau, Charles Consel
INRIA Bordeaux Sud-Ouest
Talence, France
{*julien.bruneau, charles.consel*}*@inria.fr*

Marcia O'Malley
Rice University
Houston, Texas, USA
omalleym@rice.edu

Walid Taha
Halmstad University
Sweden
Walid.Taha@hh.se

Wail Masry Hannourah
HVAC Consultant
Cairo, Egypt
wail_hannourah@yahoo.com

***Abstract*—Smart buildings promise to revolutionize the way we live. Applications ranging from climate control to fire management can have significant impact on the quality and cost of these services. However, smart buildings and any technology with direct effect on human safety and life must undergo extensive testing. Virtual testing by means of computer simulation can significantly reduce the cost of testing and, as a result, accelerate the development of novel applications. Unfortunately, building physically-accurate simulation codes can be labor intensive.**

To address this problem, we propose a framework for rapid, physically-accurate virtual testing. The proposed framework supports analytical modeling of both a discrete distributed system as well as the physical environment that hosts it. The discrete models supported are accurate enough to allow the automatic generation of a dedicated programming framework that will help the developer in the implementation of these systems. The physical environment models supported are equational specifications that are accurate enough to produce running simulation codes. Combined, these two frameworks enable simulating both active systems and physical environments. These simulations can be used to monitor the behavior and gather statistics about the performance of an application in the context of precise virtual experiments. To illustrate the approach, we present models of Heating, Ventilating and Air-Conditioning (HVAC) systems. Using these models, we construct virtual experiments that illustrate how the approach can be used to optimize energy and cost of climate control for a building.

***Keywords*-Virtual testing, Smart buildings, HVAC, Pervasive computing**

I. Introduction

Buildings are designed to achieve a wide range of goals. For example, ensuring occupant safety and health requires specialized appliances such as fire alarm systems [1] and proper ventilation [2]. These concerns are very real: Each year, hundreds of people die from carbon monoxide poisoning due to poor ventilation design. Similarly, ensuring occupant comfort (which is generally essential for ensuring their productivity [3]) also requires specialized appliances. The stringent uptime requirements placed on building appliances also give rise to concerns about energy [4], cost, and environmental impact [5]. Traditional technologies such as brick and mortar, standard heating and airconditioning units, have all been carefully scrutinized with respect to their impact on all of these goals, and have passed the test of extended timelines. Often, the design of such types of systems is intensionally kept simple and passive in order to meet some of these goals, such as safety and maintainability.

1) Smart Buildings.: Recently, there has been an increasing interest in smart building technologies. Such technologies bear a promise to revolutionize the way we live [6]. Applications ranging from climate control to fire alarm systems, to lighting management, to security systems can be found in smart buildings [7], [8], [9].

Intrinsic to the idea of smart buildings is the introduction of higher levels of active control into the traditional components of a building. Higher levels of active control are achieved by the use of more sophisticated control algorithms, more extensive sensing of the physical environment, more actuation of various physical subsystems, and more communication between different components of the system. However, because of the direct impact that buildings have on humans and on their resources, smart building technologies must be held to stringent standards for correctness, availability, and a wide range of other software quality considerations.

2) Virtual Testing.: Testing is a crucial tool for eliminating poor designs, and developing a degree of confidence in promising designs. Testing is equally important for traditional physical design and design involving active control. But testing smart building technologies in physical buildings can be slow and prohibitively expensive. Addressing these two problems can accelerate the rate of innovation and deployment of successful smart building applications. In particular, using computer simulations to carry out some part of the testing virtually can help achieve this goal. The effectiveness of this approach, however, depends heavily on the accuracy with which we model both the control and the physical components of the system. Furthermore, building accurate simulation codes, especially for physical systems, can be labor intensive and can slow down the whole development process for smart building applications.

The goal of this paper is to address three technical challenges that must be overcome in order to enable effective virtual testing of smart buildings. The first is to accurately capture the distributed and networked nature of the active devices in the system. The second is to accurately capture the physical properties of the building. The third is to auto-

matically map such models directly to executable simulation codes.

3) State of the Art.: Existing approaches only cope at most with one of the three challenges raised by the virtual testing of smart buildings. For instance, several projects simulated building active devices using MATLAB/Simulink [10]. These projects focus on the fine-grained modeling of these devices. They provide libraries of digital components that can be used to model devices. However, they do not attempt to use analytically sound models of the physical environment surrounding such devices. COMSOL allows to accurately simulate the surrounding physical environment. For instance, it provides a heat transfer module and an acoustics one. However, these simulations are based on the Finite Element Method (FEM) and are prohibitively expensive for modeling the physical environment of a whole smart building. Other tools allow faster simulation of the physical properties of a building. Modelica is one of these tools. Modelica is an equation-oriented modeling language. The main draw back is that modeling systems in Modelica that combine discrete and continuous behaviors can be somewhat challenging.

4) Contributions.: In this paper, we present a new framework that allows an effective virtual testing of smart buildings where

- Physical aspects of smart homes are modeled explicitly in Acumen [11], a domain-specific language with specialized support for describing continuous systems (Section II).
- The active components of smart homes are modeled explicitly in DiaSpec [12], a domain-specific language with specialized support for modeling distributed, digital systems (Section III).
- The complete models, containing both Acumen and DiaSpec components, are mapped to executable codes (Section IV). This is achieved by combining Acumen's simulation capability and the DiaSim simulator [13].
- A virtual testing platform to monitor and gather statistics about smart building appliances can be built (Section V).
- Virtual experiments using different building models and control strategies can be analyzed (Section VI).

II. Modeling the Physical Environment

Virtual testing of smart building designs requires accurately capturing its physical properties. A building is a multi-physics system involving multiple interrelated physical characteristics. Physicists have already defined these phenomena with mathematical definitions (*e.g.,* with ordinary/partial differential equations). These analytical descriptions are an ideal material to reuse in order to capture the physical properties of a building. In this section, we first explain the approach that we adopt to modeling heat transfer and temperature change in a building. We then show how the differential equations that capture this model are expressed in Acumen. For reasons of space, we only present our temperature model. However, we have also modeled relative humidity and carbon dioxide density in Acumen.

A. Modeling Temperature and Heat Transfer in a Building

Human comfort and safety are highly sensitive to the temperature of the surrounding air. As a result, it is critically important to accurately model the factors that impact temperature, including appliances that can be used to control it, as well as the processes by which the temperature of the air in a given room is changed.

We present the models used in our study as a series of successive refinements of a basic model. All models are compartment models, in that they treat each room as one state variable. The basic model, as well as the refinements which include additional terms, are all differential equations.

1) Heat Transfer.: Heat transfer is the rate at which heat moves through a medium or from one medium to another, and is a topic studied extensively in thermodynamics (*e.g.,* [14]). Heat transfer between two media is linearly proportional to the difference in temperature between the two media. In addition, it is also affected by the thermal resistance of the boundary between the two media. For simplicity, we assign each room in a building one temperature value. Reasonable values for the thermal resistance of building walls, windows and doors can be determined using a reference book in the heating and cooling domain [14].

Let us assume that all rooms are numbered. We will use the subscripts i and k to refer to room numbers. Let $Neighbors(i)$ be the set of numbers representing the rooms neighboring room i. Let T_i denote the temperature of room i.

To help introduce the reader to the notation and the equations that define heat transfer in a building, we begin by assuming that the only factor affecting temperature in a particular room is heat from neighboring rooms. To express even this simple process, we need some additional notation. In particular, we will also use the following convention:

- $\frac{dT_i}{dt}$ Rate of temperature change in room i ($^\circ$C.h^{-1}),
- C_i Thermal capacitance of room i (J.$^\circ$C^{-1}),
- R_{ik} Thermal resistance of the boundary between rooms i and k ($^\circ$C.h.J^{-1}). It takes into account the heterogenous elements of this boundary (*e.g.,* walls, windows, doors).

The equation constraining the rate of change for each and every room i is given by the equation:

$$\frac{dT_i}{dt} = \frac{1}{C_i} \sum_{k \in Neighbors(i)} \frac{T_k - T_i}{R_{ik}} \tag{1}$$

Because this equation is instantiated for each room, the whole building is modeled by a set of such equations.

2) Air-Conditioning Unit.: We now consider adding air-conditioning (AC) units to our model. An AC unit consists of a heater and a cooler. We need only to introduce four additional parameters:

$B_{h(i)}$ The heater in room i is active (1 if present and active, 0 o.w.),

P_i Heater power (W),

$B_{c(i)}$ The cooler in room i is active (1 if present and active, 0 o.w.),

Q_i Cooler power (W),

The equation above only needs to be extended as follows:

$$\frac{dT_i}{dt} = \frac{1}{C_i}\left(\sum_{k \in Neighbors(i)} \frac{T_k - T_i}{R_{ik}} + (B_{h(i)} * P_i - B_{c(i)} * Q_i)\right) \quad (2)$$

3) Occupants.: Occupants can be modeled as heat sources. The set of occupants of room i is denoted $Occupants(i)$. We only need one additional parameter to incorporate this aspect of building:

H_j Heat dissipation of occupant number j (W),

Thus, the final equation can be expressed as:

$$\begin{aligned}\frac{dT_i}{dt} =& \frac{1}{C_i} \sum_{k \in Neighbors(i)} \frac{T_k - T_i}{R_{ik}} \\ &+ \frac{1}{C_i} * (B_{h(i)} * P_i - B_{c(i)} * Q_i) \\ &+ \frac{1}{C_i} * \sum_{j \in Occupants(i)} H_j\end{aligned} \quad (3)$$

Other heat sources, such as equipment, appliances, and lights, were neglected for simplicity but will be included at a later stage. Now, we are ready to present the actual Acumen code used in the experiments we report on in the rest of the paper.

B. The Heat Model in Acumen

By design, the Acumen modeling language enables direct specification and simulation of continuous and discrete systems. Our virtual testing framework only uses its continuous system modeling and simulation capability, and not its support for modeling discrete behaviors.

Figure 1 presents the Acumen specification of the temperature defined in Equation 3. The `continuous` section in Figure 1 specifies the temperature rate of change for each room of the building. Equations in Acumen can refer to derivatives of variables. For example, `T'` refers to the first derivative of `T`. Finally, the boundary conditions subsection allows one to define the initial state of the physical environment. This initial state can be easily changed by setting new boundary conditions. As can be noticed, defining physical characteristics in Acumen is straightforward and has a direct correspondance to equational definitions. Thus, Acumen leverages standard mathematical notation used to define physical phenomena.

```
(* Building topology, Room 0 corresponds to the outside *)
building=((0),(0,2),(0,1,3),(0,2));

(* Temperature in each building room, T0 is the outside temperature *)
T=(T0,T1,T2,T3);

(* Other variable definitions *)
...
continuous
  foreach room in length(building) begin
    T'[room] = 1/C[room] *
      ((sum n < length(building[room]) in ((T[building[room][n]]−T[room]) /
            R_th[room][n]))
        + Bh[room]*P[room]−Bc[room]*Q[room]
        + (sum p < length(occupants[room]) in H[occupants[room][p]]));
  end

 boundary conditions
   T0 with T0(0) = 10;
 (* We define the boundary conditions for all continuous variables *)
   ...
```

Figure 1. Temperature specification in Acumen.

III. Modeling Active Components

Active systems deployed in smart buildings involve a wide range of devices and software components, communicate using a variety of protocols, and rely on intricate distributed systems technologies. Accurate virtual testing for smart buildings requires accurately capturing the distributed and networked nature of these active systems.

We rely on the DiaSpec language for modeling active systems. DiaSpec first consists of a taxonomy language dedicated to describing classes of devices that are relevant to the target active systems. These devices can be hardware or software. DiaSpec also provides an Architecture Description Language (ADL) layer. This ADL layer enables to model active controllers.

In this section, we explain how a taxonomy of devices is created in DiaSpec, present the ADL layer, and briefly summarize the benefits of using DiaSpec for modeling active systems in smart buildings.

A. Modeling Active Devices in DiaSpec

DiaSpec provides a dedicated language for specifying the taxonomy of devices relevant to a target application domain. As shown in Figure 2, a device consists of sensing capabilities, producing data, and actuating capabilities, providing actions. Following this structure, a device description declares a data source for each one of its sensing capabilities. For example, the `TemperatureSensor` class of devices in Figure 2 provides a `temperature` data source. An actuating capability corresponds to a set of method declarations. Thus, the `Heater` class of devices provides an `OnOff` action. This action allows to turn on and off a heater instance. A device declaration also includes attributes, characterizing properties of device instances (*e.g.,* the `LocatedDevice` instances are characterized by the room in which they are deployed). Finally, device declarations are organized hierarchically allowing device classes to

inherit attributes, sources and actions. In our example, the `Heater` and `TemperatureSensor` classes of devices inherit the `room` attribute.

```
device LocatedDevice {
  attribute room as Room;
}

device Heater extends
       LocatedDevice {
  action OnOff;
  action HeatControl;
}

action OnOff {on(); off();}
action HeatControl {heat(); cool();}

device TemperatureSensor extends
       LocatedDevice {
  source temperature as Float;
}

structure Room {
  name as String;
}
```

Figure 2. Device taxonomy in DiaSpec.

B. Modeling Active Controllers in DiaSpec

DiaSpec provides an ADL layer for specifying the architecture of active controllers. This layer is dedicated to an architectural pattern commonly used in the pervasive computing domain [15]. Active controllers are decomposed in two types of components: context and controller. Context components are fueled by sensing devices. These components filter, interpret and aggregate these data to make them amenable to the applications needs. Controller components receive application-level data from context components and determine the actions to be triggered on devices.

Figure 3 illustrates the definition of an active controller in DiaSpec with a temperature regulator. It is composed of a `TemperatureRegulation` context component and of a `HeaterController` controller component. First, we define a `TemperatureRegulation` context component. This context component receives low-level temperature data from temperature sensors. This context component is responsible for determining whether a temperature regulation is needed. The produced `Regulation` information is coupled to a `room` index so that the regulation is realized room per room. Finally, we define a `HeaterController` component. This controller component is responsible for applying the temperature regulation. It receives regulation information from the `TemperatureRegulation` context component. It acts on heaters to apply this regulation.

```
context TemperatureRegulation as Regulation indexed by room as Room {
  source temperature from TemperatureSensor;
}

controller HeaterController {
  context TemperatureRegulation;
  action OnOff, HeatControl from Heater;
}
```

Figure 3. Temperature regulator architecture in DiaSpec.

C. DiaSpec and Virtual Testing

Using DiaSpec brings multiple benefits to our virtual testing framework regarding development and simulation support, going beyond a contemplative approach. Not only is a DiaSpec description a high-level model of an active system, it is also processed by a compiler that generates a dedicated Java programming framework. This programming framework enables the developer to easily implement the active components that he modeled using DiaSpec. Indeed, the provided support is dedicated to what was described with DiaSpec. For instance, a remote procedure call from an active controller to a device is realized by simply calling a Java method provided in the generated programming framework. This framework abstracts over the distributed and heterogeneous nature of this type of system. Further information on the generated programming support can be found elsewhere [12].

Using DiaSpec for modeling the active system allows us to test active controllers, without code modification, in a 2D simulator, named DiaSim [13]. The simulation support provided by DiaSim is described next.

IV. Mapping the Models into Executable Simulation Codes

The final technical challenge to enabling effective virtual testing of smart building is to automatically map the models of both physical environment and active components to executable simulation codes. This section explains how this mapping is done, and explains how the simulations of the Acumen and DiaSpec models interact.

1) Execution of Acumen Models.: Acumen models are directly mapped to executable code. The process of mapping Acumen models to executable code is described in more details elsewhere [11]. The user only needs to specify the length of the simulation and the step size of each simulation iteration. Continuous variables in an Acumen model are discretized with respect to this step size. The simulation accuracy depends on the chosen step size. However, having a small step size requires larger computation resources. The user needs to properly set the step size value so that an acceptable level of accuracy is obtained in the simulation.

2) Execution of Active Controllers.: From the DiaSpec taxonomy and architecture, the DiaSpec compiler generates a dedicated Java programming framework. The developer uses this framework to implement both the active devices and the active controllers. Once the active controllers are developed, we can execute and test them with the DiaSim simulator. DiaSim allows active controllers to interact with simulated devices. The active controller implementation does not have to be modified in order to be simulated in DiaSim. This allows to test and assess the application logic of the active controllers. The simulated device localizations in the physical environment are graphically edited in the DiaSim editor. Once the physical environment is edited, we generate a simulation support that is used for executing the active system simulation. DiaSim also allows to test active controllers in hybrid environments, mixing virtual

and real active devices. This feature allows an incremental deployment of a smart building. DiaSim is more extensively presented elsewhere [13].

3) Smart Building Simulation.: The Acumen and DiaSim simulations interact in two ways. First, simulated sensors need to retrieve their sensed information from the Acumen model. Second, simulated actuators may modify the state of variables in the Acumen model. These two interactions are achieved using Acumen's socket-based Java API to interact with an Acumen simulation.[1] While editing the simulated active system in the DiaSim editor, the user specifies the Acumen variables of interest for each simulated sensor. Thus, the data sensed by a simulated sensor correspond to a variable in Acumen. Continuous variables in Acumen are discretized with respect to a given step size defined by the user so that the discrete components can use them. Finally, the user specifies which variable is modified by an actuator action and how it is modified.

Finally, we generate Java implementations for executing simulated devices. The necessary glue code to interface with Acumen is generated in the simulated device implementation with respect to what the user has specified during the edition.

V. Monitoring Virtual Smart Buildings

Virtual testing of smart building active components involves modeling and simulation of numerous interacting components. In addition, it is difficult for the user to determine if an active controller behaves correctly without observing the physical environment. To cope with these difficulties, we first provide support for monitoring the virtual testing of smart buildings. We also provide tools for analyzing the results after the completion of virtual testing.

4) 2D Graphical Rendering.: Simulations can be monitored using a 2D renderer provided by DiaSim. This renderer is based on the Siafu simulator [16]. The image rendered for an example virtual house is presented in Figure 4. This rendering facilitates monitoring the state of the active devices. Information is displayed when a sensor publishes an event and when an action is called on an actuator. Moreover, graphical rendering helps the user visualize the state of the physical environment. Indeed, it is possible to display a greyscale overlay that represents the values of the physical environment characteristics (*e.g.,* temperature and luminosity).

5) Virtual Testing Analysis.: During a simulation, we log the values of the continuous and discrete variables. This allows to save the physical environmnent state and the active device states at each simulation iteration. At the end of a simulation, all logged variables are automatically plotted. This plot can serve as a basis for analyzing events that occured during the simulation. If the user is interested in a more specific analysis (*e.g.,* energy consumption of the active devices), it is easy to create dedicated plots with tools such as GNUplot. For example, we created GNUplot scripts for reading the logged variables and displaying the energy use and comfort level achieved in Section VI.

[1]Thanks to Cherif Salama for providing this API.

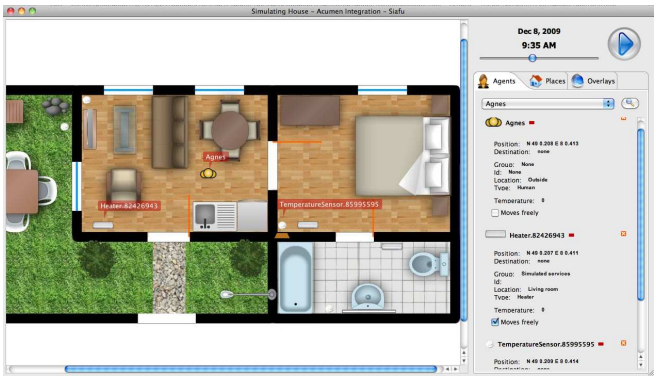


Figure 4. 2D graphical rendering of a virtual house.

VI. A Virtual Experiment

This section presents the active system analyses that enable our virtual testing framework. Active systems can first be evaluated using *algorithmic variations*. Indeed, multiple algorithms for the same active controller can be compared. Virtually testing these algorithms can help us find the most efficient one with respect to the functionalities of the active controllers. A second variation allowed by virtual testing is *structural variation*. Multiple configurations of sensors and actuators can be tried to choose the most efficient one. These variations are facilitated by the DiaSpec approach, confining changes to the application logic.

To illustrate both types of variation, we evaluate a Heating, Ventilating and Air-Conditioning (HVAC) system deployed in a house. The plan of the house is illustrated in Figure 4. We apply algorithmic and structural variations to this system for illustrating the usefulness of our virtual testing framework.

A. Heating, Ventilating and Air-Conditioning Systems

HVAC systems are responsible for providing thermal comfort and acceptable indoor air quality to the building occupants. These systems are a standard part of mechanical engineering curricula, see for example [17], [18]. HVAC systems regulate multiple physical properties of a building. For example, temperature and humidity must be regulated so that the occupants feel comfortable. As well, carbon dioxide density needs to be regulated for keeping an acceptable indoor air quality. Finally, the amount of airflow introduced to an air-conditioned zone must be controlled to remain pleasant for the occupants. To regulate these multiple physical characteristics, HVAC systems interact with numerous active devices. It retrieves information from sensors such as temperature, humidity and carbon dioxide sensors. It

also controls heaters, humidifiers and ventilators to provide comfort and acceptable indoor air quality to occupants. Developing an HVAC system that regulates these multiple physical characteristics is complex. Virtual testing is useful for testing such system.

B. Global vs. Local Temperature Management

Multiple Regulating and Standards organizations define comfortable ranges of temperature depending on the indoor humidity and the outside temperature. For instance, ASHRAE defines the comfortable humidity and temperature ranges in its "Thermal Environmental Conditions for Human Occupancy" standard. We use these values for evaluating our HVAC control algorithms.

Originally, to keep the temperature in the comfort zone, HVAC systems had a single thermostat and regulated the temperature everywhere using this single thermostat. We call this strategy *global temperature management*. Global temperature management is obviously not optimal, because the temperature can be different in each room. To achieve finer grained, and more efficient regulation, EnergyStar recommends managing the temperature locally [19]. In this case, the building is divided in areas, each area with its own thermostat. We call this strategy *local temperature management*. To illustrate virtual testing, we set up an experiment to evaluate the effect of applying this recommendation to the HVAC system of a 3-room house. We then observe the comfort and the energy use differences between these two temperature management policies.

C. HVAC System Evaluation

Both global and local regulations use the same regulation logic. The only difference is the scope of this regulation. We name T_{c-min} and T_{c-max}, respectively the lowest comfortable temperature and the highest comfortable temperature. Our regulator needs to keep the temperature in a range $[T_{min}, T_{max}]$. If the temperature is below T_{min}, the HVAC system ventilates hot air. If the temperature is above T_{max}, the hot air stops being ventilated. Obviously, the logic of our regulator is very simple and could be refined. However, it is enough for illustrating the variations we apply for testing this regulator.

In this virtual experiment, we use the temperature model presented in Section II. In this model, our virtually tested HVAC system has a heating power of 1500 Watts and a cooling power of 600 Watts. The boundaries between areas of our virtual physical environment are illustrated in Figure 4. The calculated thermal resistance coefficients depend on the windows, doors and walls that compose these boundaries.

1) Algorithmic Variation.: In this section, we test several algorithms for our temperature regulator and choose the most efficient one with respect to its energy use. T_{min} and T_{max} are defined in Equation 4.

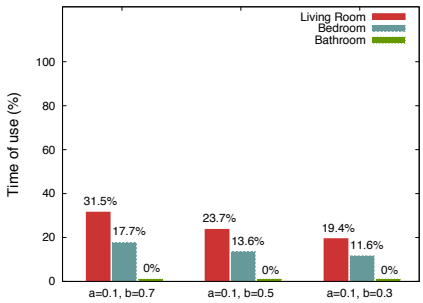


Figure 5. Comparison of different algorithms for regulating the temperature in terms of energy use.

$$\begin{aligned} T_{min} &= T_{c-min} + a * (T_{c-max} - T_{c-min}) \\ T_{max} &= T_{c-min} + b * (T_{c-max} - T_{c-min}) \\ & a, b \in [0, 1] \text{ and } a < b \end{aligned} \tag{4}$$

We test our HVAC system with different values of T_{min} and T_{max} over a period of one month. We decrease these two values as long as comfort is provided 100% of the time. We choose January because heating is critical during this month. The chosen outside temperatures correspond to Bordeaux average temperatures in January. We test three different algorithms. The percentage of time when the hot air is ventilated is presented in Figure 5. We see that these algorithmic variations bring differences in terms of heater time of use. A low range of temperature results in less heating time. Since the heaters have the same power, the differences in the time of use of these heaters allow to evaluate the energy consumption gain. The third algorithm in Figure 5 allows a 38% gain of energy use compared to the first one.

2) Structural Variation.: We also test two different configurations of active components for comparing global and local temperature regulation. The global configuration consists of one temperature sensor in the living-room and one heater in each room. The local configuration consists of one temperature sensor and one heater in each room. We use the most efficient algorithm from the previous section. We test our HVAC system over the month of January. The comfort provided by these two configurations is presented in Figure 6. Local temperature regulation provides perfect comfort to the three rooms of the house. In comparison, global temperature regulation is not as comfortable for the occupants. The bathroom temperature is comfortable for only 86% of the time. The percentage of time when hot air is ventilated for the two types of regulation is presented in Figure 7. We can see that global temperature management ventilates more hot air than local temperature management. Our virtual testing framework allowed us to see that following the EnergyStar recommandation enables to get a better comfort with less energy use from the HVAC

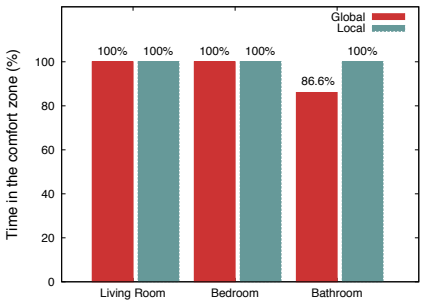


Figure 6. Comparison of the comfort provided by global and local temperature management in the house.

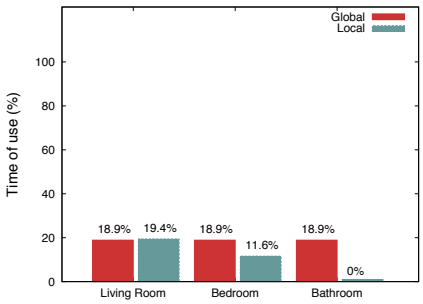


Figure 7. Comparison of the time of heating required by global and local temperature management in the house.

system.

3) Virtual Testing Accuracy.: The user needs to carefully choose the accuracy of his virtual testing. The accuracy of virtual testing depends on two parameters: physical environment model accuracy and size of the time discretization step. The temperature model we present in this paper considers that each room has a single state. A smaller discretization of the space would allow the simulation of the physical environment to be more precise. However, a smaller space discretization requires more computation.

Likewise, the size of the time discretization step impacts the virtual testing accuracy. In our evaluation, we choose a discretization step of one minute. A one minute accuracy is enough for evaluating an HVAC system over a month. However, this criteria needs to be carefully chosen depending on the tested smart building application.

VII. Related Work

Existing tools provide partial means to cope with the three challenges stated in the introduction of this paper.

Several projects focus on modeling and simulating building active devices using MATLAB/Simulink [10]. These projects allow a fine-grained modeling of these devices. They provide libraries of digital components that the tester can use to model his devices. However, they do not allow an analytically sound simulation of the physical environment that impacts such devices. Ptolemy goes one step further than MATLAB/Simulink. Ptolemy provides a library of computation models that the user can compose for modeling and simulating embedded systems. Its main contribution is to allow the simulation of systems that contain heterogeneous computation models. Using Ptolemy would be complementary with our approach. Ptolemy could be used for defining the computation models of active devices, whereas DiaSpec describes the outside interface of these active devices. However, the support provided by Ptolemy for modeling continuous systems is too restricted for modeling the physical environment of a building.

A few simulators exist in the pervasive computing domain [20], [21], [22], [23]. Ubiwise [20] and Tatus [21] are built upon 3D game graphics engines, respectively Quake III Arena and Half-Life. However the game graphics engine becomes a burden when it comes to define new scenarios; users can neither add their own actuators and sensors, nor simulate arbitrary physical characteristics. Lancaster [22] and PiCSE [23] allow to add new types of sensors and actuators for testing active systems. However, there is no support for modeling and simulating the physical environment that surrounds active systems.

Other projects focus on modeling the physical environment. COMSOL [24] allows to accurately simulate the surrounding physical environment. For instance, it provides a heat transfer module and an acoustics one. However, these simulations are based on the Finite Element Method (FEM) and are too slow for modeling the physical environment of a whole building. Other tools allow a faster simulation of the physical properties of a building. Modelica [25] is one of these tools, and is in fact a closely related language to Acumen. The differences between the two are primarily in Acumen's support for binding time separation and partial derivatives [11], but these are in fact orthogonal to the models used here. For more sophisticated models of heat transfer, however, partial derivatives are needed.

Our study of the related works showed that testing virtual smart building applications with existing tools is a very complex task. Our approach is a step towards decreasing this complexity by using DSLs to model and execute the physical and digital aspects of smart buildings. Indeed, these DSLs provide the user with a declarative and high-level support that simplifies the virtual testing of these applications.

VIII. Conclusion

In this paper, we have presented a virtual testing framework for smart buildings. This framework first provides an analytically sound modeling of the physical environment of smart buildings using Acumen. Our framework also enables the modeling of active components of a building using DiaSpec. Finally, our virtual testing framework map directly both the physical environment model and the active component model into executable simulation codes.

This work can be expanded in several directions. First, we plan on testing other smart building applications in our

virtual testing framework. One of these applications is a fire alarm system. Another future work is to couple our virtual testing framework for smart buildings to a human behavior model. Such coupling would allow to see how people behave depending on the active components present in a building, and depending on the physical environment. An example would be to observe people behavior in a building in case of fire.

REFERENCES

[1] National Fire Protection Association website. http://www.nfpa.org/.

[2] American Society of Heating, Refrigerating and Air-Conditioning Engineers website. http://www.ashrae.org/.

[3] A. Hedge. Linking environmental conditions to productivity. In *Eastern Ergonomics Conference and Exposition*, New York, June 2004.

[4] Energy Star website. http://www.energystar.gov.

[5] Energy & Environmental Building Alliance website. http://www.eeba.org/.

[6] R. Harper. *Inside the Smart Home*. Springer London Ltd, 2003.

[7] P. I-Hai Lin and H.L. Broberg. Internet-based monitoring and controls for hvac applications. *Industry Applications Magazine, IEEE*, 8(1):49–54, Jan/Feb 2002.

[8] R. Mueller, J.S. Rellermeyer, M. Duller, and Gustavo. Demo: A generic platform for sensor network applications. In *Mobile Adhoc and Sensor Systems, 2007. MASS 2007. IEEE Internatonal Conference on*, pages 1–3, Oct. 2007.

[9] K. C. Lee and H.-H. Lee. Network-based fire-detection system via controller area network for smart home automation. *Consumer Electronics, IEEE Transactions on*, 50(4):1093–1100, Nov. 2004.

[10] P. Riederer. MATLAB/Simulink for Building and HVAC Simulation - State of the Art. In *Proceedings of the 9th International IBPSA Conferene*, 2005.

[11] Y. Zhu, E. Westbrook, J. Inoue, A. Chapoutot, C. Salama, M. Peralta, T. Martin, W. Taha, M. O'Malley, R. Cartwright, A. Ames, and R. Bhattacharya. Mathematical Equations as Executable Models of Mechnical Systems. In *ICCPS 2010*, 2010.

[12] Damien Cassou, Benjamin Bertran, Nicolas Loriant, and Charles Consel. A generative programming approach to developing pervasive computing systems. In *Proceedings of GPCE '09*. ACM, October 2009.

[13] J. Bruneau, W. Jouve, and C. Consel. Diasim, a parameterized simulator for pervasive computing applications. In *Proceedings of Mobiquitous'09*.

[14] T. Kuehn. *Fundamentals: 2005 Ashrae Handbook*. Amer Society of Heating, 2005.

[15] A. K. Dey, G. D. Abowd, and D. Salber. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human-Computer Interaction*, 16(2):97–166, 2001.

[16] M. Martin and P. Nurmi. A generic large scale simulator for ubiquitous computing. In *MobiQuitous 2006*. IEEE Computer Society, July 2006.

[17] W. Bobenhausen. *Simplified design of HVAC systems*. John Wiley and Sons, 1994.

[18] J. I. Levenhagen. *HVAC control system design diagrams*. McGraw-Hill Professional, 1999.

[19] A Guide to Energy-Efficient Heating and Cooling by Energy Star.

[20] J. J. Barton and V. Vijayaraghavan. Ubiwise, a ubiquitous wireless infrastructure simulation environment. Technical report, Hewlett Packard, 2002.

[21] E. O'Neill, M. Klepal, D. Lewis, T. O'Donnell, D. O'Sullivan, and D. Pesch. A testbed for evaluating human interaction with ubiquitous computing environments. In *TRIDENTCOM '05*, 2005.

[22] R. Morla and N. Davies. Evaluating a location-based application: A hybrid test and simulation environment. *IEEE Pervasive Computing*, 3(3):48–56, Juil-Sep 2004.

[23] V. Reynolds, V. Cahill, and A. Senart. Requirements for an ubiquitous computing simulation and emulation environment. In *InterSense '06*, 2006.

[24] COMSOL website. http://www.comsol.com/.

[25] M. Tiller. *Introduction to Physical Modeling With Modelica*. Kluwer Academic Publishers, 2001.