Coordination Scheme Editor for building management systems

Maxime Louvel, François Pacull, Maria Isabel Vergara-Gallego

Coordination Scheme Editor for Building Management Systems

Maxime Louvel, François Pacull, Maria Isabel Vergara-Gallego
Univ. of Grenoble Alpes
CEA, LETI, MINATEC
Email: FirstName.LastName@cea.fr

***Abstract*—This paper presents our work on the Coordination Scheme Editor (CSE). This editor provides a web interface to design building automation rules. Rules are designed with a drag and drop programming. Then rules are validated, generated and activated in the buildings.**

CSE allows to define template rules. The rules are then instantiated in building rooms with the available sensors and actuators. CSE also offers a monitoring interface to know which scenario is impacted by one or several faulty sensors/actuators.

The CSE relies on the LINC middleware to provide coordination across several building automation systems and sensors/actuators technologies.

I. Introduction

Building management systems (BMS) have traditionally been proprietary and closed systems. A BMS consists of hardware and software components; in general, software components are proprietary, based either on closed solutions or open industrial standards. Typical examples of standard solutions for BMS are LONworks, KNX, BacNet, and Modbus [1]. When a building is built, the hardware components are installed and a software commissioning is done. Once again this operation is done using proprietary software and, in most of the cases, it requires the intervention of a skilled technician.

Buildings are usually built to last 20 years or more, and, during their lifetime, they have to evolve. Indeed, buildings are usually retrofitted every two or three years to add new equipment, new facilities (e.g. walls, building extension or new material), or simply to reconfigure a poorly functioning building. Another type of evolution is reconfiguration for users comfort or usage. For instance, a wall can be added/removed to create/remove rooms. After such modifications, the BMS must be reconfigured to adapt the building behaviour. To do so, a skilled technician has to come in the building and perform the reconfiguration of the BMS using a proprietary software.

The complexity increases when considering multiple technologies in one building. Today there is a trend to integrate open industry standards meaning that a unique building may integrate several technologies, or several BMS, that work independently and that have to be configured independently. Another trend is to install new devices, such as sensors, that are not originally provided by the BMS. For instance, indoor air quality sensors exist but they are not connected to existing BMS. Typically, when these sensors are installed data are collected by a dedicated system, an off-line analysis is done and guidelines are provided to the facility manager. The facility manager then calls a technician to update the BMS accordingly. Another example is the emerging market of the Energy Sub-Contracting Companies (ESCOs) [2]. ESCOs want to manage the building more efficiently to reduce the energy bill and take some money in the loop. To make their job more efficient they will want to install specific sensors that should interact with the BMS.

Several solutions have been proposed to connect multiple BMS and technologies through a software layer [3], [4]. The work proposed in this paper goes further, by providing a graphical user interface that permits a user (e.g. the facility manager or the building occupants) to monitor the BMS and define the interactions between the connected devices. The use of such interface does not require any special skill. The proposed approach is called Coordination Scheme Editor (CSE). The CSE is a software component based on the LINC [5] coordination middleware. LINC has already been used in building automation applications integrating heterogeneous systems [6], [7]. The CSE provides a web interface to design, monitor, and update BMS rules. This interface is based on the latest web standards (HTML5, SVG and javascript) and blockly [8] to offer an easy drag and drop programming. Once the rules are defined by the user, LINC verifies, compiles, and executes the rules using the real devices. In addition, when a sensor or an actuator becomes faulty, alerts are raised in the CSE to show all the rules impacted by the error.

This paper is structured as follows. First, Section II gives an overview of the middleware LINC; then, Section III describes the CSE and how it can be used to generate, verify, and execute new rules in the building. Section IV explains how the CSE has been used to generate rules in different contexts. Finally, Section V summarizes some existing related approaches and Section VI concludes the paper and presents future directions.

II. Overview of LINC

To make this paper self-contained, this section presents an overview of the middleware LINC. More details on LINC can be found in [5]. LINC provides a uniform abstraction layer to encapsulate software and hardware components. This abstraction layer relies on the associative memory paradigm implemented as a distributed set of bags containing resources (tuples). Inspired by Linda [9], bags are accessed through three operations:

- `rd()`: takes a partially instantiated tuple as input parameter and returns from the bag a stream of fully instantiated tuples matching the given input pattern;
- `put()`: takes a fully instantiated tuple as input parameter and inserts it in the bag;
- `get()`: takes a fully instantiated tuple as input parameter, verifies if a matching resource exists in the bag and consumes it in an atomic way.

Bags are grouped within objects according to application logic. For instance, all the bags used to manage a set of devices that communicate using the same communication technology can be grouped in the same LINC object. LINC objects are able to execute rules that manipulate resources in its own bags or in the bags of other objects.

The rest of this section describes the properties of LINC and the existing frameworks that facilitate the development of building automation applications and user interfaces. Finally, it describes how rules can be injected into the system, and activated/deactivated at run-time.

A. Coordination rules

The three operations `rd()`, `get()` and `put()` are used within production rules [10]. A production rule is composed of a precondition phase and a performance phase.

Precondition phase: The precondition phase is a sequence of `rd()` operations which detect or wait for the presence of resources in several given bags. The resources are, for instance, values from sensors, external events, or results of service calls. In the precondition phase:

- the output fields of a `rd()` operation can be used to define input fields of subsequent `rd()` operations;
- a `rd()` is blocked until at least one resource corresponding to the input pattern is available.

Performance phase: The performance phase of a LINC rule combines the three `rd()`, `get()` and `put()` operations to respectively verify that some resources (e.g. the one(s) found in the precondition phase) are present, consume some resources, and insert new resources.

In this phase, the operations are embedded in one or multiple *distributed transactions* [11], executed in sequence. Each transaction contains a set of operations that are performed in an atomic manner. Hence, LINC guarantees that actions belonging to the same transaction, are either all executed or none. This ensures properties such as:

- Some conditions responsible for firing the rule (precondition) are still valid at the time of the performance phase completion (e.g. the presence sensor still detects someone);
- All the involved bags are effectively accessible (e.g. bags encapsulating sensors or an actuators).

B. LINC for buildings management systems

In the context of building automation, LINC provides the PUTUTU framework [6], [7], which integrates more than 20 wireless and wired technologies commonly used in BMS and WSAN (Wireless Sensor/Actuator Networks). PUTUTU is formed of a set of LINC objects that encapsulate different communication technologies. These objects inherit from three basic objects which contain special bags to facilitate the manipulation of information and the development of BMS rules. The three basic objects are:

- *WSAN_Sensor Object*: This object is used to encapsulate sensors. Among others, it provides three bags: `Sensors` to store sensor values in the form: $(id, value)$, `Type` to associate the identifier of a sensor with the type of data it provides (i.e. $(id, type)$), and `Location` to manage the spatial location of the sensor (i.e. $(id, location)$). The first two bags are usually filled by the driver that encapsulates the technology and the last bag is filled by the application when the binding is done.
- *WSAN_Actuator Object*: This object encapsulates actuators. As the `WSAN_sensor object` it provides a bag `Type` to keep information about the device type and a bag `Location` to keep information about the location of the device. It provides also the bag `Command` which is used to send a command to the actuator. Then, to control an actuator, a resource is put in the bag `Command`. The resource contains the `id` of the actuator, the `command` to apply, and possible `parameters`. When the resource is added, the command is sent to the actuator through the driver encapsulating the protocol.
- *WSAN_Sensor_Actuator object*: This object inherits from the last two objects and it is used to encapsulate technologies that provide sensors and actuators devices.

Other components, such as BMS software tools and databases, can be mapped to obtain information regarding the building, to add information regarding new discovered devices, and to reconfigure the devices. For instance, in [12] the software tools of a LONWorks [13] BMS have been encapsulated in LINC in order to automatically reconfigure the BMS.

C. User interfaces

LINC provides graphical user interfaces based on the Model-View-Controller(MVC) [14] approach. The MVC mechanism has been implemented above the bag paradigm. This mechanism permits to create interfaces that represent the real system, the user is able to interact with the real system, and the interface is updated when there is an event from the system. A framework that provides user interfaces has been developed [15]. The framework permits creating web interfaces based on the latest standards (HTML5, javascript and CSS). A LINC user interface is provided by a LINC object and can be accessed through any web browser. The framework provides three main types of objects which are based on the MVC mechanism and that are used to create user interfaces:

- *Object SVGInterface*: This object is able to render Scalable Vector Graphic (SVG) files. The SVG file is previously designed and javascript code is attached to support user interaction (e.g. click on predefined graphical elements). Graphical elements in the SVG are modified dynamically (e.g. change color, shape, text, hide, show,

and so on), accroding to events from the system, thanks to the MVC mechanism.

- *Object Plot*: This object generates different types of plots (as SVGs) and pies to visualize data.
- *Object Layout*: This object provides the layout of the main interface. The interface contains several LINC interfaces, that can be provided by different LINC objects (SVGInterface, plot, layout, or other) and may also include external web interfaces. The layout is defined using an HTML web-page or a SVG file previously designed.

D. Rule generation and execution

In LINC, rules can be added in a running system. The rule is simply added as a resource in the form of `(r_id, context, rule)`. When the resource is added, the rule is verified and compiled. If there is no error, the rule is executed by a LINC object. The context is an information meaningful for the application. It can be used to activate or deactivate a group of rules. For instance, a building may have different rules for different day periods, or it could have some rules for normal functioning and others for emergency situations.

Additionally, the execution of a rule is also controlled by resources in the `RulesId` bag of coordinator objects. This bag contains tuples shaped as "(rule_id, context, status)", the `status` can be either `ENABLED` or `DISABLED`. When the rule is compiled, a `rd` to this bag is added at the beginning of the precondition and the performance of the rule, to check if the rule is `ENABLED`. Adding these `rd` operations permit stopping the execution of a rule by simply replacing the resource `("rule_id", "context", "ENABLED")` to `("rule_id", "context", "DISABLED")`. Indeed, the `rd` operation of the rule status causes the transaction to fail. In this way, a rule can enabled/disabled by adding a resource on a bag. Combining this with the `context` information, it is possible to enable/disable all the rules of a given context.

III. Coordination Scheme Editor

This section describes how the CSE works. First it describes its architecture. Then, it details how new rules can be defined and verified using the graphical interface of the CSE. Finally, a brief overview of the building monitoring interface provided by the CSE is presented.

A. CSE Overview and Architecture

The architecture of the CSE is illustrated in Figure 1. Existing communication technologies for building automation are integrated with LINC using the PUTUTU framework [6]. In this way, heterogeneous sensors and actuators can be accessed and manipulated by the application. Besides, the system integrates a database that keeps information regarding the devices deployed in the building(s). This information permits the application to associate a device identifier with the type of information it can provide, the actions it can perform, its location, and so on.

The user interface has been developed using the MVC GUI framework provided by LINC. As seen in Figure 2,

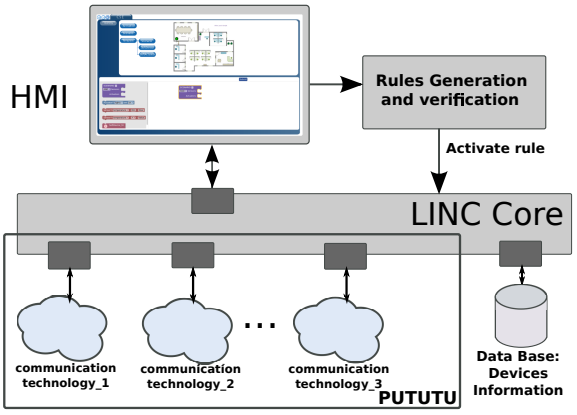

Fig. 1: CSE architectures

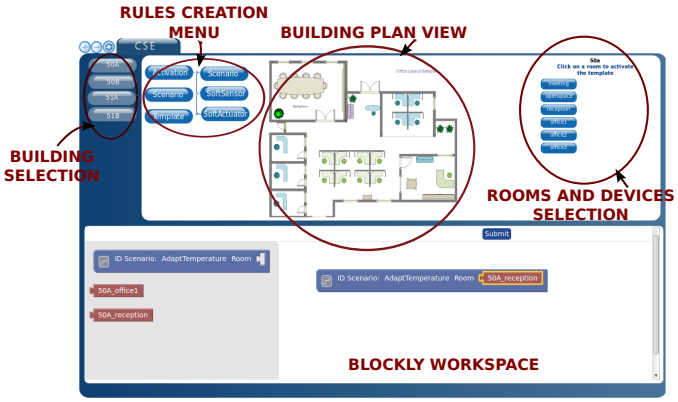

Fig. 2: CSE Interface

the interface consists of a Layout interface (managed by a Layout object) which contains several SVG sub-interfaces, all provided by the CSE LINC object. The interface displays the plan of the building and floor of interest. It lists the rooms of the floor and displays the different devices available in every room. Finally, through the use of a blockly-based interface, new rules, that describe interactions between the devices, can be defined by the user and added into the system.

B. Rule design

Rules can be defined by the user, through the creation of `Templates`. Defined `Templates` are then activated in the desired room(s). The blockly interface provides blocks that can be selected by the user to create or update the template.

The user can also create a dedicated rule (without a template) by directly choosing the devices available on a specific room of the building. The same interface is used for dedicated rules or templates.

1) Templates definition: A template permits the creation of generic rules, using a set of devices, that can be activated later on a given room or set of rooms. The template can correspond to a scenario (i.e. a set events generating actions for actuators), a soft-sensor (i.e. combination of real sensors or external values), or a soft-actuator (i.e. to act on several actuators).

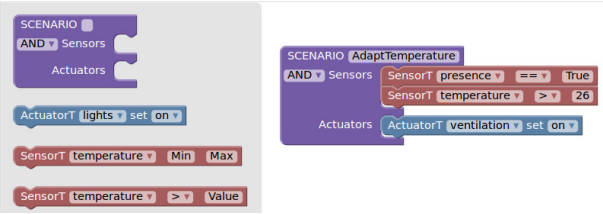

Fig. 3: Scenario Template Creation Example

Template Scenario: A template scenario defines a rule that combines several sensors and actuators. A scenario is formed of a set of conditions, such as values of sensors, to trigger actions to a set of actuators. Figure 3 illustrates the definition of a template scenario that detects the presence of someone in the room, using a presence sensor; then, it determines if the room temperature is higher than "26 degrees Celsius", using a temperature sensor. If the two previous conditions are true, the ventilation of the room is turned on.

Template Soft-Sensor: The value of a soft-sensor is a function that combines the values of a set of real sensors. Figure 4 shows an example of definition of a soft-sensor. In this case, the value of the soft-sensor will be "true" if the presence sensor value is "true" and the humidity is higher than or equal to "50%".

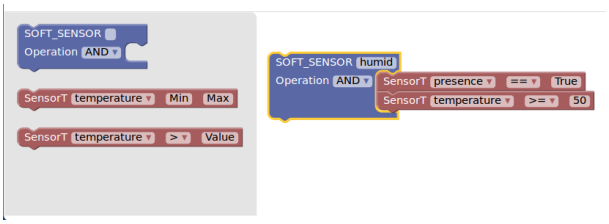

Fig. 4: Soft Sensor Template Creation Example

Template Soft-Actuator: Acting on a soft-actuator means acting on several actuators. For instance, all the lights in a room can be combined on a single actuator, so that turning on this actuator will turn on all the lights in the room.

C. Template activation

Once a template is defined, the user can activate it on a room. For that the user can choose the desired building and floor. Then, the list of defined templates and possible rooms will be displayed on the interface, and the user can bind a template to a room.

Once the user submits the activation, LINC will check if the room has all the devices needed by the template. If this is the case, LINC will generate the appropriate rule, using the available devices, and add the rule into the system to execute it. Listing 1 shows the rule generated for an instance of the template scenario shown in Figure 3. Line 1 and 7 are used to control the rule execution. Line 2 corresponds to the first condition of the template ($presence == "True"$) and matches resource `("pr_23", "True")`. Line 3 and 4 correspond to the second condition of the template ($temperature > 26$).

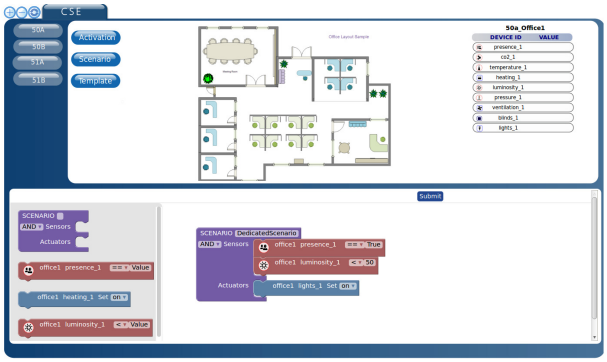

Fig. 5: Dedicated Scenario Example

Line 3 reads the resource in the sensor bag and line 4 checks if the temperature is higher than 26. If matching resources are found, the performance is triggered. Line 8 and 9 checks tha the sensor values did not change and line 10 sends the command `on` to the ventilation system.

```
["RulesId"].rd("r0033", "temp_sc", "ENABLED") &                    1
["Sensors"].rd("pr_23", "True") &
["Sensors"].rd("t_34", t_val) &                                    3
INLINE_ASSERT: t_val > 26 &
::                                                                 5
{
  ["RulesId"].rd("r0033", "temp_sc", "ENABLED");                   7
  ["Sensors"].rd("pr_23", "True");
  ["Sensors"].rd("t_34", t_val);                                   9
  ["Actuators"].put("v_42", "on");
}.                                                                 11
```

Listing 1: Rule generated from Template

A rule instantiated from a template can be deactivated by removing the corresponding resource in the bag `RulesId` of the coordinator object (i.e. the CSE object). It is also possible to disable all the instantiated rules of template thanks to their context information (e.g. `"temp_sc"`).

D. Dedicated rules

Using the devices available in the building, it is also possible to write rules. In this case no template is defined, the devices are chosen from the list of available devices. The Figure 5 shows how a dedicated rule is created, available devices per room can be chosen from the list in the right side of the interface, then they can be used as a blockly block, in the blockly workspace, to create the rule.

Dedicated rules are also verified before their execution, and they can be activated/deactivated when desired.

E. Rule verification

When a template is instantiated or a rule is designed, several verifications can be done. If an error is detected, it is shown on the interface and the rule is not activated. Examples of errors that can be detected are:

- values tested for the sensors are out of the sensor range;
- invalid command for an actuator;
- invalid type in free entry text (e.g. a string when expected an integer).

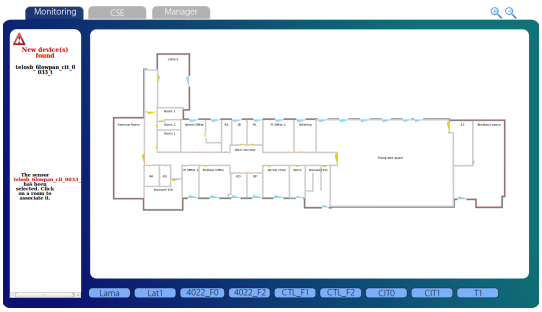

Fig. 6: CSE monitoring interface

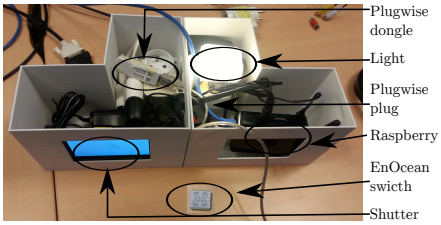

Fig. 7: Building emulation

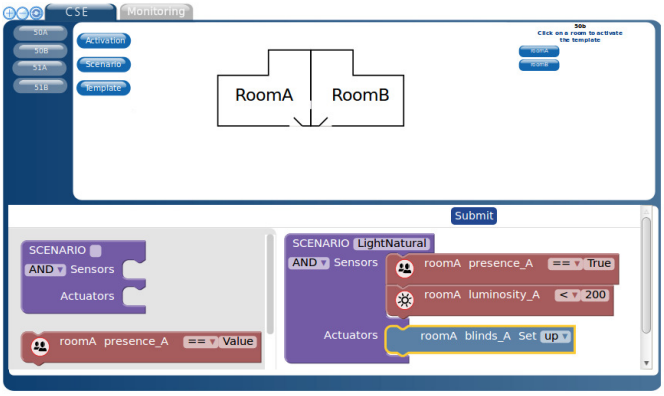

Fig. 8: Building Emulation Scenario

F. Monitoring

The CSE also provides an interface to monitor the rules deployed in the buildings. The floor plan of a room is displayed along with sensor values and actuator staus.

In addition, the monitoring interface informs when a new device, that has not been bound, is discovered and allows the user to bind the new device to a given location. After the binding is done, the new devices can be used in rules. Figure 6 displays the monitoring interface where the building plan is visualized on the right, and new discovered devices appear on the left part of the window.

Finally, CSE offers a monitoring interface of all the rules designed with CSE. It is possible to filter the rules based on building, room, template generation/instantiation date, and the status of the rule. The status of a rule can be activated, stopped, deleted or error. Error status means that one or more devices used by the rule became faulty. Thanks to the template instantiation, the facility manager can then easily link the faulty device to a template scenario. For instance, if the presence sensor `"pr_23"` becomes faulty, used in Listing 1, the rule `"r0033"` is impacted. This rule was generated by instaniating the template 3 on a given room. Hence the facility manager will know that the temperature of the room is not properly controlled anymore.

IV. Case studies

A. Lighting system control

This use case has been implemented in the FP7 SCUBA European project [16]. The demonstrator integrates the LONWorks [13] Building Automation System (BAS) with external information. LONWorks is a technology largely used to control devices in large buildings. However, the BAS uses only information coming from within the building. When the building is built (or retrofitted), a LONWorks technician comes and manually configures the connections between the sensors (e.g. movement or buttons) and the actuators (e.g. light, shutter or Heating Ventilation and Air Conditioning (HVAC)).

The goal of this scenario is to control the lighting system of the building, using external information regarding the power consumption of the building and the current energy price. The idea is to decrease the light intensity to remove/decrease picks of power consumption or to save money when the energy price is high. Then, several lighting levels are defined, according to the occupancy of the room and external information.

First, the LONWorks system has been integrated in LINC [12] as well as the ligthing algorithm, running in the cloud. Then, different devices of the building were mapped in the CSE according to their position in the building. A soft sensor and a soft actuator have been defined. The soft sensor takes inputs from the lighting algorithm. The soft actuator is the LONWorks device controlling the lights in the room. Finally, three template scenarios were defined to adapt the ligth intensity. The first one switches off the light when the room is empty. The second one sets the light to 500 lux when the room is occupied and no constraints are imposed on energy consumption. The last one sets the light to 150 lux if the external algorithm detects that energy price is high or the demand is high in the building. With the CSE the facility manager can create his own rules according to his objectives.

B. Building emulation

The second use case is a portable demonstrator, that emulates a buidling containing several sensors and actuators technologies. Figure 7 is a picture of the demonstrator. The demonstrator integrates a lighting control device (Plugwise plug that can be on or off), a shutter (graphical interface on a small screen), and emulates a presence and luminosity sensor (EnOcean switch with four buttons). The building control and the CSE run in a Raspberry PI platform.

With CSE we designed several scenarios to adapt the light and the shutter according to the information coming from the EnOcean button. Figure 8, shows a rule example, where the shutters are controlled according to sensor values (EnOcean

switch). This emulation shows that we can easily implement rules with CSE integrating several heterogenous technologies.

V. Related work

With the increase of embedded communicating devices in applications, such as building automation and smart homes, visual programming frameworks are an interesting approach to provide the user with the tools necessary to control their own system. Such framework must deal with several challenges. First they have to support the integration of heterogeneous devices and software/hardware components. Then they should allow the generation, verification, and execution of distributed rules. And finally, they should provide an intuitive visual programming language. Based on LINC, the CSE answers all these challenges and provides a complete solution.

Dealing with heterogeneity in these applications has been studied in the literature [4], [17], however, no solution integrates a friendly user interface with a mechanism to coordinate distributed devices while hiding heterogeneity. Blockly has already been used to program wireless sensor network applications in smart environments [18]. Similar to the CSE, the authors provide a web interface based on blockly permitting the definition of rules. Then, the proper code is generated. Nevertheless, this approach does not support heterogeneous devices (i.e. only CoAP capable devices are encapsulated). Other visual programming approaches such as ClickScript [19] and Labview [20] require some programming skills and are not suited for non-experienced users.

VI. Conclusion

This paper presented the Coordination Scheme Editor (CSE). CSE is a web-based tool used to design, monitor and control building automation rules. CSE relies on the LINC middleware and rules can thus be designed with heterogeneous hardware and software technologies.

CSE offers a visual programming language based on the latest web standard (HTML5, SVG and javacript) and Blockly. This provides an easy to use HMI than can be used by facility manager, building owners or event building occupants. CSE interface allows to design template rules that can be instantiated across buildings from a single web interface. At design and instantiation time, rules are verified to check for instance that only valid commands are sent to actuators. CSE also provides a monitoring interface that can display the status of all the rules. The monitoring interface also shows all the rules and templates impacted by a faulty sensor.

Future work will focus on providing more advanced verification with CSE. Indeed currently CSE only verifies that the generated rules are correct. However there is no verification on the potential conflicts between rules. A typical conflict is when two rules can be activated at the same time (i.e. same triggering condition) and they have different effect in the system (e.g. start vs. stop the ventilation).

Acknowledgment

This work has funded by the Artemis ARROWHEAD (grant 332987) and the H2020 TOPAs project (grant 676760).

References

[1] W. Kastner, G. Neugschwandtner, S. Soucek, and H. M. Newman, "Communication systems for building automation and control," *Proceedings of the IEEE*, vol. 93, no. 6, pp. 1178–1203, June 2005.

[2] E. Stuart, C. A. Goldman, P. Larsen, and D. Gilligan, "The us esco industry: recent trends, current size and remaining market potential," *Lawrence Berkeley National Laboratory, American Council for an Energy-Efficient Economy Proceedings, Chicago, Illinois*, 2014.

[3] D. Bruckner, J. Haase, P. Palensky, and G. Zucker, "Latest trends in integrating building automation and smart grids," in *IECON 2012 - 38th Annual Conference on IEEE Industrial Electronics Society*, Oct 2012, pp. 6285–6290.

[4] A. Veichtlbauer and T. Pfeiffenberger, "Generic middleware for user-friendly control systems in home and building automation," *International Journal on Advances in Networks and Services Volume 6, Number 1 & 2, 2013*, 2013.

[5] M. Louvel and F. Pacull, "Linc: A compact yet powerful coordination environment," in *Coordination Models and Languages*. Springer, 2014, pp. 83–98.

[6] F. Pacull *et al.*, "Self-organisation for building automation systems: Middleware linc as an integration tool," in *IECON 2013-39th Annual Conference on IEEE Industrial Electronics Society*. Vienna, Austria: IEEE, 2013, pp. 7726–7732.

[7] L.-F. Ducreux, C. Guyon-Gardeux, S. Lesecq, F. Pacull, and S. R. Thior, "Resource-based middleware in the context of heterogeneous building automation systems," in *IECON 2012-38th Annual Conference on IEEE Industrial Electronics Society*. Montreal, Canada: IEEE, 2012, pp. 4847–4852.

[8] "Blockly: A visual programming editor," https://developers.google.com/blockly/, accessed: 2016-04-15.

[9] N. Carriero and D. Gelernter, "Linda in context," *Commun. ACM*, vol. 32, pp. 444–458, 1989.

[10] T. Cooper and N. Wogrin, *Rule-based Programming with OPS5*. San Fransisco: Morgan Kaufmann, 1988, vol. 988.

[11] P. A. Bernstein, V. Hadzilacos, and N. Goodman, *Concurrency control and recovery in database systems*. New York: Addison-wesley, 1987, vol. 370.

[12] L.-F. Ducreux, M. Louvel, F. Pacull, S.-R. Thior, M.-I. Vergara-Gallego, and O. Yaakoubi, "Dynamic reconfiguration of building automation systems with linc," *Sensors & Transducers*, vol. 185, no. 2, p. 68, 2015.

[13] U. Ryssel, H. Dibowski, H. Frank, and K. Kabitzsch, *IEHandbook LONWorks*. Berlin: Vde-Verlag, 2010.

[14] G. E. Krasner and S. T. Pope, "A description of the model-view-controller user interface paradigm in the smalltalk-80 system," *Journal of Object Oriented Programming*, vol. 1, pp. 26–49, 1988.

[15] L.-F. Ducreux, C. Guyon-Gardeux, M. Louvel, F. Pacull, S. R. Thior, and M. I. Vergara-Gallego, "Rapid prototyping of complete systems, the case study of a smart parking," in *2015 International Symposium on Rapid System Prototyping (RSP)*. IEEE, 2015, pp. 133–139.

[16] "SCUBA: Self-Organazing, Co-operative, and robUst Building Automation," http://www.aws.cit.ie/scuba//, accessed: 2016-04-21.

[17] A. Fernbach, W. Granzer, and W. Kastner, "Interoperability at the management level of building automation systems: A case study for bacnet and opc ua," in *Emerging Technologies Factory Automation (ETFA), 2011 IEEE 16th Conference on*, Sept 2011, pp. 1–8.

[18] M. A. Serna, C. J. Sreenan, and S. Fedor, "A visual programming framework for wireless sensor networks in smart home applications," in *Tenth IEEE International Conference on Intelligent Sensors, Sensor Networks and Information Processing, ISSNIP 2015, Singapore, April 7-9, 2015*, 2015, pp. 1–6.

[19] L. Mainetti *et al.*, "A novel architecture enabling the visual implementation of web of things applications," in *21st International Conference on Software, Telecommunications and Computer Networks, SoftCOM 2013, Split-Primosten, Croatia, September 18-20, 2013*, 2013, pp. 1–7.

[20] G. W. Johnson, *LabVIEW Graphical Programming: Practical Applications in Instrumentation and Control*, 2nd ed. McGraw-Hill School Education Group, 1997.