Building Automation: Experience with Dynamic Reconfiguration of a Room

Maxime Louvel, François Pacull, Safietou Raby Thior, Maria Isabel Vergara-Gallego, Oussama Yaakoubi
Univ. Grenoble Alpes, F-38000 Grenoble, France
CEA, LETI, MINATEC Campus, F-38054 Grenoble, France
firstname.lastname@cea.fr

***Abstract*—This paper details a case study in the domain of building automation systems. This case study is the reconfiguration of a room that can be split in two or merged in one according to the current needs. The building is equipped with the LON system, a standard in building automation. Such a reconfiguration is normally done manually by a skilled technician. Thanks to our approach, it can now be autonomous and triggered by various external events such as sensor readings, a remote controller or information from an agenda. To achieve dynamic reconfiguration in this context we rely on LINC, a coordination middleware providing an abstraction layer allowing the encapsulation of hardware and software components. Thus, we can not only integrate sensors and actuators but also legacy systems such as the LON configuration tool. The coordination is provided by coordination rules which can be statically defined at application deployment or dynamically added or removed to reconfigure the system.**

Keywords–Coordination; middleware; building automation.

I. INTRODUCTION

Advances in consumer electronics and embedded systems have leveraged the emergence of connected smart devices for a huge number of applications. These devices communicate using different protocols and physical medium. Such explosion of heterogeneous devices and services, and the necessity of interaction between them, have yielded architectural concepts based on a common software layer called *middleware*. A middleware provides an abstraction view of heterogeneous hardware, services, and protocols. It hides heterogeneity to the application and facilitates software re-utilisation, application deployment, and coordination between different devices to achieve a common goal.

Building Automation Systems (BAS) are a typical example of systems requiring a middleware. Indeed, a BAS contains a huge number of devices and services that use different protocols and that need to interact between them. A BAS typically has the following requirements: i. The *necessity of a common software layer* to access heterogeneous devices, ii. The *compatibility with legacy systems and protocols* to communicate with existing infrastructures and extend BAS capabilities and iii. The *possibility to dynamically reconfigure the system* to allow special operating modes, adaptation to changing devices and space redistribution. Existing middlewares [1] [2] [3] only partially fulfil these needs. They are either too complex to allow dynamic reconfiguration or they focus on a specific part of the BAS needs.

In this paper, we use LINC [4] to answer these challenges. LINC is a resource-based middleware that provides interactions between connected devices or software components through coordination rules. Several technologies have already been encapsulated for building automation. For instance, in [5] LINC was used to control interactions between different devices. In the present paper, we describe another angle of LINC where it is used as an administration layer where devices can be dynamically reconfigured and interactions between entities belonging to the same technology may be changed according to a given context. In this case, LINC is required only for the reconfiguration phase.

This paper presents an example where a BAS, that uses the Local Operating Network (LON) [6] platform, is dynamically reconfigured to allow space reconfiguration. In the case study we consider a room that is split in two (or merged in one) when a removable wall is closed (or opened). Then, all the devices present in the room are automatically reconfigured according to the position of the wall. The rules describing all the actions to perform for reconfiguration are automatically generated, compiled and executed. These actions include removing the current configuration and putting in place the new one. Additionally, LINC provides the possibility to extend the capabilities of the BAS thanks to the encapsulation of new components.

The structure of this paper is as follows. Section II presents some related work; then, Section III describes the LINC middleware and Section IV describes the architecture put in place for dynamic reconfiguration. The case study is described in Section V along with some discussions and results. Finally, Section VI concludes the paper.

II. RELATED WORK

A BAS concerns the control of building services. Extending or modifying the behaviour of an existing building automation infrastructure requires dealing with heterogeneity of communication protocols and legacy systems [7]. There are lots of communication protocols and many of them coexist on the same building.

Therefore, a middleware or abstraction layer [8] is needed in such scenario. Some characteristics as scalability, fault tolerance and flexibility are desired when choosing the proper middleware. For instance, scalability allows extending the capabilities of the BAS by introducing new communication technologies. Wireless technologies are becoming very attractive given its flexibility and easiness to install and deploy [9] [10]. Some efforts trying to integrate legacy BAS technologies and wireless sensor networks technologies can be found in the literature [11] [12]. Besides, some middleware solutions for BAS have been proposed [2] [3]. However, since they target a specific technology or application, these solutions lack of flexibility. Scalability may also be an issue given the complexity of such approaches.

Regarding reconfiguration, it can be performed at different levels: i. the *nodes or devices level*, where the behaviour of the node is modified, i.e., sampling period, routing mechanism, sensing precision, transmission power and ii. the *system level* where relationships and coordination of devices are modified according to a new defined scenario. In a Wireless Sensor Network scenario, much effort has been put at the node

 ISBN: 978-1-61208-374-2

level; in this case, reconfiguration is mostly related to reliable communication between nodes and energy efficiency. Some examples of reconfiguration in Wireless Sensor Networks are presented in [13] [14]; furthermore, a middleware may facilitate this task [15]. In this paper, we focus on the system level reconfiguration which allows to adapt an existing system to a new defined scenario so that new coordination rules are generated and executed. Such reconfiguration may be triggered by an external event. Similarly, a context-aware middleware, allows adapting the application, based on information retrieved from sensors or events. Several context-aware middlewares can be found in the literature [16] [1] each of them focusing on a given set of applications. For the specific case of building automation Istoan et al. [17] propose a mechanism to reconfigure dynamically the relationships between devices and services in a building. This proposition remains as a conceptual approach and no real implementation is presented.

III. Overview of LINC

Full description of the LINC middleware may be found in [4]. This section describes the very basic information to make the paper self-contained.

LINC provides a uniform abstraction layer to encapsulate the different software and hardware components. This layer simplifies the integration of legacy components and their coordination. This abstraction layer relies on the associative memory paradigm implemented as a distributed set of bags containing resources (tuples). Following the Linda [18] approach, bags are accessed only through three operations:

- `rd()`: takes a partially instantiated tuple as input parameter and returns a stream of fully instantiated tuples from the bag, where the fields match 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.

A. Examples of components encapsulated as bags

A typical sensor, measuring a physical quantity, can be modelled through a **Sensor** bag containing tuples, that are formed as `(sensorid, value)`. The measured quantity is then retrieved by accessing this bag. To differentiate sensors capabilities (temperature, humidity, and so on), a **Type** bag is added, containing tuples formed as `(sensorid, type)`.

Actuators may be modelled with a bag **command** designed as `(actuatorid, function, parameter1, parameter2)`. Then, when a resource is inserted with a `put()` operation, the bag triggers the awaited action over the targeted actuator adapted with the appropriate parameters. In a similar manner, any other component or service which provides a given protocol or API, such as SOAP, RPC, CORBA, or REST can be encapsulated in a bag or set of bags.

B. Object

Bags are grouped within objects according to application logic. For instance, all the bags used to control a network with a given technology are grouped in the same object.

C. Coordination rules

The three operations described above, i.e., `rd()`, `get()` and `put()`, are used within production rules [19] A production rule is composed of a precondition phase and a performance phase.

1) 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.

2) Performance phase: The performance phase of a production 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* [20], executed in sequence. Each transaction contains a set of operations that are performed in an atomic manner. Hence, we can guarantee that actions that belong 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;
- All the involved bags are effectively accessible. For instance, for a bag encapsulating a remote service we can determine if such service can be actually accessed.

These properties are very important as they ensure that the set of required objects, bags and resources, are actually available "at the same time". More properties offered by LINC in the BAS context are detailed in [5].

IV. Application Architecture for dynamic reconfiguration

Figure 1 presents the architecture of an application based on LINC. The top layer is the application layer and it defines how devices and services interact with each other through the use of bags and coordination rules.

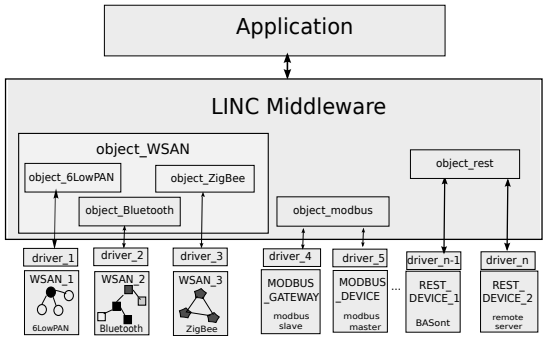

Figure 1: Example of application

The bottom layer corresponds to the hardware or service specific layer. The API and actions provided by each device are encapsulated so that they can be accessed through bags defined in LINC. To facilitate software re-use and evolution, and to reduce complexity, we have defined frameworks defining common features. For instance, all wireless sensor networks share a common set of bags for controlling the sensors and the actuators. Only the communication part which is dependent

 ISBN: 978-1-61208-374-2

on the technology is different. We have thus defined a WSAN (from Wireless Sensors and Actuator Network) object which is inherited by all the object responsible for a technology. Adding a new protocol is basically integrating the driver for the new technology. Hence, when the driver part has been written the object can be coordinated with the rest of the system. More details on this software architecture may be found in [5] [21].

In several applications, reconfiguration of devices at run time may be needed. For instance, we may need to change device parameters and the relationships between devices, or we may need to perform a software update.

Usually with such architecture, the coordination of devices of different technologies is done through coordination rules. These rules define the different configurations and interactions between devices. For instance, what action has to be done on the light when the button is pressed twice. Thus, reconfiguring is removing some coordination rules and replacing them by new ones, taking into account the new behaviour.

Alternatively, when all the concerned devices use a technology that integrates one controller, i.e., LON or KNX, we can use LINC to reconfigure these controllers. To do so, we extended the application described above with the encapsulation of the legacy system used to reconfigure the controllers of the devices. In the case of LON it is the LNS22O tool [22]. Then, the reconfiguration is done by dynamically inserting new coordination to reconfigure the different controllers.

The reconfiguration procedure will be started by the application as a response to a given event, i.e., a request from the user, the presence of a resource, or the value of a given sensor. We now detail how it is possible to dynamically generate rules and how the activated rules can controlled.

A. Dynamic rules generation

When performing reconfiguration, coordination rules must be consistent with the new configuration or scenario. To ensure this consistency, rules can be automatically generated corresponding to contextual information. In LINC, rules are seen as resources in bags. Hence, they can be added, enabled or disabled at run-time. To add a new rule, a resource is added in a dedicated bag of the object called `AddRules`. This bag receives resources of the form `(package,source)` where `package` is the logical name of the group of rules and `source` is the actual code of the rules.

When the reconfiguration is triggered, the device is reconfigured and new coordination rules are generated and added in the `AddRules` bag. When a resource is added in this bag, the rule is dynamically compiled. This compilation includes syntax verifications, and various checks to prevent potential issues at execution time. If the rule contains no detectable error, the object starts to execute it right away. At this point, the new scenario is set-up.

B. Control of rules execution

In a rule-based system it might be difficult to know which rule is executed at any time. Ensuring that a rule is really not active may be even more difficult. To overcome such issues, LINC uses a specific bag, called `RulesId`, containing the rules enabled. When a rule is compiled (when the system is started, or dynamically added as explained previously), an operation `rd(ruleId, "ENABLED")` is added in the beginning of the precondition and in every transaction of the rule. Hence, removing a rule only requires to remove the corresponding resource in the bag `RulesId`. Indeed, no new instance of the rule may be started. If an instance reaches the performance phase, it will be aborted because the resource `rd(ruleId, "ENABLED")` is not in the bag.

Obviously, the same pattern may be applied to a group of rules by using a specific bag mapping the rules id to some other information that is meaningful for the application. A typical example in a BAS is to have two set of rules. The first set are the rules for daily life (e.g., managing lights, heating and comfort). The second set of rules are dedicated to emergency situation (e.g., to coordinate the lights with the evacuation procedure). In case of an emergency, the comfort rules must be disabled to prevent unexpected situation. This can be done with a bag containing the current mode i.e., `comfort` or `emergency`. A `rd("comfort")` (respectively a `rd("emergency")`) on the this bag is added in every precondition and every transactions ensuring that only one mode is used at a time.

Another important aspect of rules execution is that they can be run on a distributed way. This characteristic provides scalability and the possibility to integrate a considerable number of technologies. For the particular case of building automation, several BAS can be easily integrated. Moreover, rules can be replicated on several machines so that we guarantee its execution even if there is a faulty machine.

V. Case Study: Reconfiguration of a Building Automation System

The context of this case study is the SCUBA (Self-organising, Co-operative and robUst Building Automation) project [23]. It aims to provide novel architectures, services, and engineering methodologies for robust, adaptive, self-organising, and cooperating monitoring and control systems. The case study presented in this paper consists in reconfiguring a LON system when a room is split in two (or merged in one). This implies to change the binding between devices to adapt the system to the current situation: Which button trigger what and which sensor control what.

This case study illustrates a requirement more and more important in BAS where people want to optimise the room usages. To better explain the scenario, we first introduce some basic concepts and definitions regarding building automation and the LON technology, which is the imposed legacy technology. Then, we describe the reconfiguration procedure and we discuss the obtained results.

A. Building Automation

A BAS is a network of software and hardware components that sense, control, and act on the environment and communicate between them. It ensures the operational performance of the facility as well as the comfort and safety of building occupants.

Nowadays, it is possible to monitor and control several systems of a building. Typically, these systems work independently and can communicate between them thanks to a given interconnection technology which can be a standard or a proprietary solution. In general, several communication technologies are present in a building.

At deployment time, the interconnections between devices are defined according to the disposition of the building. Configuring such interconnections is complex, time demanding, and requires the intervention of qualified personal. Reconfiguration of the BAS may be required in order to respond to the needs of occupants, or to replace or upgrade some equipments.

 ISBN: 978-1-61208-374-2

Typically, these modifications are planned in advance and they are done on purpose by a technician.

Changes in the space configuration is something more dynamic, unplanned and that can be triggered by any occupant. Physical reconfiguration of a room needs to be synchronised with the involved equipments, i.e., sensors, actuators, controllers. For example, consider a button to switch on or off the lights in a room. If this room is combined with another room by removing a wall, the lights of the second room must also be connected to the button of the first room.

B. LON

LON [6] is a control network system designed by Echelon Corporation. LON is widely used in existing building automation systems and it allows communication between devices coming from different manufacturers using a common protocol called LONtalk. LON devices are physically linked together and they embed a special controller called the Neuron Chip. The latter is associated with a transceiver that allows its reconfiguration and communication with other devices. A unique identifier permits to identify inputs and outputs of each device on the network. Although devices are physically connected, they are not able to exchange messages unless a logical connection, called a `Binding`, is created between them. Figure 2 shows an example of two bindings between three devices. In the example, on of the Network Variable Outputs (NVO) of the `Device_1` is associated to the Network Variable Inputs (NVI) of two devices: the `Device_2` and `Device_3`.

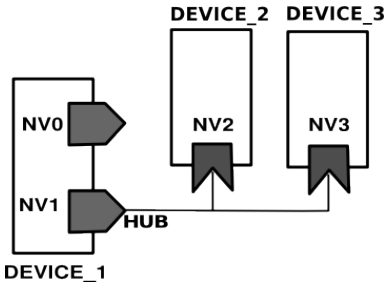

Figure 2: Binding between devices

LON networks are designed, monitored, and managed through a dedicated MS-Windows software stack. This stack is based on the LON Network Services (LNS) tool [22]. The tool gives access to a database storing LON designs and provides a complex API to interact with the network. An LNS Server is provided on top of the LON Networks as an ActiveX component. Usually, these tools allow reconfiguring the network connections by manually creating or removing bindings between network variables. However, the user can only act on one binding at a time. A classical room configuration typically contains dozens of bindings and reconfiguration needs to be done by a skilled technician. Hence, space reconfiguration with the LON technical tool rapidly becomes too costly and time demanding.

C. Reconfiguration of a Room

The platform used for the case study is illustrated in Figure 3. It is called T1 and is located in the Schneider premises in Grenoble, France and is one of the use case sites of the SCUBA project. It consists of two separate rooms (Room A and Room B) with a removable wall. The wall may be open to combine both offices to form a single office, or it may be closed to obtain two different offices.

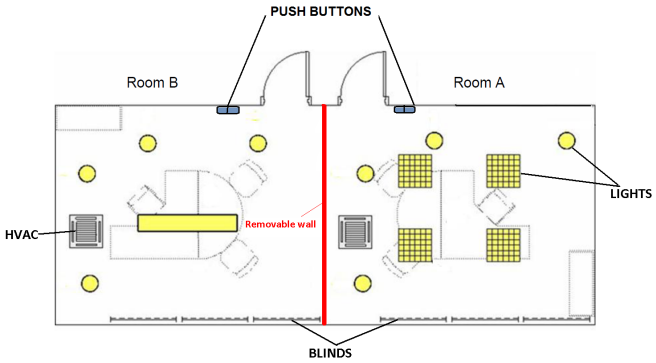

Figure 3: T1 Platform

Each office is equipped with sensors (temperature, luminosity, and presence), a Heating, Ventilation and Air Conditioning (HVAC) system, dimmable spotlights, motorised venetian blinds, and push buttons. All these devices use the LON technology.

To manage the space reconfiguration of the platform, we consider two configurations:

- **configuration 1:** The wall is open, i.e., the platform consists of a single room. In this configuration, LON bindings are created between push buttons (Room A and Room B) and actuators (lights, blinds) of both rooms. In addition, a single temperature sensor is used to control both HVACs. The two presence sensors are used with an ”OR” logical and any of them can signal the room occupancy.
- **configuration 2:** The wall is closed i.e., the platform consists of two separated rooms. Both rooms have an individual control; then, push buttons of Room A only manage actuators of Room A, and the same applies for Room B. HVACs are controlled separately by their respective temperature sensor. The two presence sensors are independent.

Detecting the presence of the wall, in order to trigger reconfiguration, is done thanks to an external device that may use an arbitrary communication protocol encapsulated by LINC. In our case study, we have used a simple wireless magnetic sensor, that operates in the 433Mhz frequency band.

Changing the room configuration requires reconfiguring LON bindings. We now detail how the LON tools have been encapsulated in LINC, and how automatic reconfiguration is made possible. The approach significantly decreases the time and cost, compared to a manual reconfiguration.

1) Encapsulation of LON in LINC: For encapsulation, a software layer (driver) has been built on top of the LNS Server. This driver permits to access LON administrative services (for reconfiguration). The `LONobject` has been placed just on top of the driver layer and it contains bags associated to a specific LON network service. Then, requesting a given service is reduced to a simple standard operation on a bag (`rd()`,`get()` or `put()`).

The most important bags in our scenario are:

- *CreateBinding* A `put()` operation on this bag, with the binding information, creates the binding;

- *RemoveBinding:* A `put()` operation on this bag, with the binding identification, removes the binding.

Internally, when a `put()` operation is done on one of these bags, the corresponding APIs in the LON ActiveX layer are called, implementing the new binding configuration.

2) Automatic Reconfiguration of LON: Listing 1 gives an example of a generated rule to switch between two configurations. The rule starts with the removal of bindings and then, it creates bindings for the new configuration. The precondition and performance of a rule are separated by the symbol "::". Hence, this rule has no precondition[1] and it is always triggered once.

This rule is automatically generated from predefined configuration stored in a dedicated component called BASont [21] provided by one of the SCUBA project partners. Alternatively, this information could be stored in a database.

```
::
{
# remove bindings
["LonObject","removeBinding"].put("Locations.Room 2 , R2
     Push Button.Switch[1] ,
     nvoSWsetting[1]@@@Locations.Room 2 , R2 Room
     Box.ConstLightCtrl[0] , nviCLsetting") ;

#...
# create the new bindings
["LonObject","createBinding"].put("Locations.Room 1 , R1
     Push Button Left.Switch[0] ,
     nvoSWsetting[0]@@@Locations.Room 1 , R1 Light
     Controller (LRC 5141).LightCotroller[0] ,
     nviChannel_1") ;
#...
}.
```

Listing 1: Generated Rule

Once generated, the rule is inserted in the `AddRules` bag of the `LONObject` upon a condition concerning the presence of the wall. To do so, we use three rules detailed in Listing 2 and 3.

```
["LonObject","Sensors"].rd("magnetic_1", "opened") &
::
{
["LonObject","Sensors"].rd("magnetic_1", "opened") &
["LonObject","TriggerConfig"].put("wall_opened") &
}.

["LonObject","Sensors"].rd("magnetic_1", "closed") &
::
{
["LonObject","Sensors"].rd("magnetic_1", "closed") &
["LonObject","TriggerConfig"].put("wall_closed") &
}.
```

Listing 2: LINC rules that trigger reconfiguration

The first set of rules (Listing 2) takes the value of the magnetic sensor, that detects the position of the removable wall. The wall is typically moved by the user of the room. When a new value is sensed, the rule inserts the required configuration in the bag `TriggerConfig` of the `LONObject`. As observed in the rule, the information obtained from the magnetic sensor is used to trigger the reconfiguration process. Thus, we can also trigger very easily the change with an event coming from a remote controller or a time-based order from an agenda. We only need to add a rule for each case.

The second rule (Listing 3) generates the rules presented in Listing 1 when the wall position changes. For each configuration, there is a list of corresponding bindings which are stored in the bag `Bindings`. This bag belongs to the object `BindingDB` and associates the list of bindings to the configuration: `"wall_closed"` or `"wall_open"`. This bag also contains the list of all bindings to erase, associated to `"clear_bindings"`, in order to have a clean configuration.

```
["LonObject","TriggerConfig"].rd("T1", wallPosition) &
["BindingDB","Bindings"].rd("clear_bindings",                    26
     allBindings) &
["BindingDB","Bindings"].rd(wallPosition,
     wallPositionbinding)&
COMPUTE: generatedRule, Rulesname =                              28
     lonFunctions.generateConfigRule(allBindings,
     wallPositionbinding, wallPosition)&
::
{                                                                30
["LonObject","AddRules"].put(Rulesname, generatedRule); ;
}.                                                               32
```

Listing 3: LINC rule that generates reconfiguration rules

The rule executes as follows:

1) A `rd()` operation on the bag `TriggerConfig` retrieves the configuration (wall position) when it changes (line 25);
2) the next `rd()` operation on the bag `Bindings` with the variant name `"clear_bindings"` retrieves the list of all possible bindings on the T1 platform independently of the actual configuration (line 26).
3) the next `rd()` operation is performed on the bag `Bindings` with the wall's position read from the bag `TriggerConfig` to get the list of bindings to be created (line 27).
4) the `generateConfigRule` method generates the LINC rule for the new room configuration following the format of the rule presented in (Listing 1). The `COMPUTE` operation permits calling an external method, in this case `generateConfigRule`.
5) the generated rule is put in the bag `AddRules` of `LonObject` in order to be executed to switch to the new configuration (line 31).

Note that, once the rule has been generated and executed, the middleware is not used anymore; the BAS continues to work autonomously thanks to the LON controller.

After the encapsulation of LON, the reconfiguration mechanism can be applied to any number of configurations involving any number of rooms. Thanks to LINC, the approach is not restricted to a given technology and it allows extending the BAS functionalities using new devices and services. Considering another standard would only require to develop the appropriate driver following the same architecture than for the LON system. Even if these types of systems are based on a quite heavy API, using several software levels, the process is repeatable. Once the driver is done, the mechanism for reconfiguration presented is this case study can be applied.

The possibility to add and remove rules dynamically, along with encapsulation of multiple technologies, provides a way to integrate an existing infrastructure with new sensors and actuators on a dynamic way. For instance, as soon as a new sensor is detected, we can execute new rules to allow its interaction with existing devices. In addition, we can also bridge two or more existing infrastructures that use different BAS technologies, i.e, LON, BacNet, KNX, Modbus. This bridging provides communication between different buildings or different rooms on a building for example.

Besides, LINC provides several characteristics that are

[1]

 ISBN: 978-1-61208-374-2

desirable for applications such as building automation. Firstly, it allows scalability thanks to its distributed nature. LINC also provides graceful degradation so that alternative actions are executed when there is a system fault. In addition, LINC uses transactions guaranteeing the execution of all the actions defined in a rule, more details can be found in [5]. When reconfiguration is performed, the latter characteristic is fundamental, since it guarantees that the system is on a consistent state after reconfiguration.

VI. Conclusion

In this paper, we have presented our approach for dynamic reconfiguration in the context of building automation. We have relied on the coordination middleware LINC to provide a non intrusive reconfiguration of a legacy system. Once the BAS have been encapsulated in LINC, it is possible to reconfigure connected devices. Reconfiguration allows changing relationships between devices or software changes as a response to an event.

Given that LINC allows adding and generating new rules on a dynamic way, reconfiguration and adaptation to new environmental conditions can be performed easily. As soon as encapsulation is done, reconfiguration requires putting a resource on a bag (send the reconfiguration command) and generating the rules according to the new scenario.

This paper has described a case study presenting the reconfiguration of a building automation system based on the LON technology. The case study is a room that can be split in two or merged in one with a removable wall. This reconfiguration concerns a significant set of equipments: temperature, luminosity and presence sensors, an HVAC system, dimmable spotlights, motorised Venetian blinds, and push buttons. In LON, such reconfiguration is normally done manually by a skilled technician. With our approach, a dynamic reconfiguration can be triggered by an external device that determines the position of the removable wall. This mechanism provides a fully automatic system, where the only action required from the user is to remove/put in place the wall. Here, the middleware is only used to reconfigure the LON controller. After reconfiguration, the system continues to run autonomously.

The approach presented in this paper opens the way to new trends in building automation at a larger scale. Indeed, based on a middleware such as LINC, it will be possible to provide such automatic reconfiguration across several buildings. Future work will focus on automatic reconfiguration through several buildings, using different BAS.

Acknowledgement

This work has been partially funded by the FP7 SCUBA project under grant nb 288079 and Artemis ARROWHEAD project under grant agreement number 332987.

References

[1] A. Saeed and T. Waheed, “An extensive survey of context-aware middleware architectures,” in Electro/Information Technology (EIT), 2010 IEEE International Conference on, May 2010, pp. 1–6.

[2] M. Sarnovskỳ, P. Kostelník, P. Butka, J. Hreňo, and D. Lacková, “First demonstrator of hydra middleware architecture for building automation,” in Proceedings of the scientific conference Znalosti 2008, 2008.

[3] C. Aghemo et al., “Management and monitoring of public buildings through ICT based systems: Control rules for energy saving with lighting and HVAC services,” Frontiers of Architectural Research, 2013, pp. 147 – 161.

[4] M. Louvel and F. Pacull, “Linc: A compact yet powerful coordination environment,” in Coordination Models and Languages, ser. Lecture Notes in Computer Science, 2014, pp. 83–98.

[5] 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.

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

[7] W. Kastner, G. Neugschwandtner, S. Soucek, and H. Newmann, “Communication Systems for Building Automation and Control,” Proceedings of the IEEE, June 2005, pp. 1178–1203.

[8] S. Krakowiak, “Middleware architecture with patterns and frameworks,” 2007.

[9] W. Guo and M. Zhou, “An emerging technology for improved building automation control,” in Systems, Man and Cybernetics, 2009. SMC 2009. IEEE International Conference on. IEEE, June 2009, pp. 337–342.

[10] V. C. Gungor and G. P. Hancke, “Industrial Wireless Sensor Networks: Challenges, Design Principles, and Technical Approaches.” IEEE Transactions on Industrial Electronics, 2009, pp. 4258–4265.

[11] O. F. et al., “Integrating building automation systems and wireless sensor networks,” in Emerging Technologies and Factory Automation, 2007. ETFA. IEEE Conference on, Sept 2007, pp. 1376–1379.

[12] M. Jung, C. Reinisch, and W. Kastner, “Integrating building automation systems and ipv6 in the internet of things,” in 2012 Sixth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), July 2012, pp. 683–688.

[13] S. Marcel et al., “Proactive reconfiguration of wireless sensor networks,” in Proceedings of the 14th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems, ser. MSWiM ’11, 2011, pp. 31–40.

[14] M. Szczodrak, O. Gnawali, and L. P. Carloni, “Dynamic reconfiguration of wireless sensor networks to support heterogeneous applications,” in Proc. of IEEE DCOSS Conf., may 2013, pp. 51–61.

[15] P. Grace, G. Coulson, G. S. Blair, B. Porter, and D. Hughes, “Dynamic reconfiguration in sensor middleware.” in MidSens, ser. ACM International Conference Proceeding Series. ACM, 2006, pp. 1–6.

[16] K. E. Kjær, “A survey of context-aware middleware,” in Proceedings of the 25th Conference on IASTED International Multi-Conference: Software Engineering, 2007, pp. 148–155.

[17] P. Istoan, G. Nain, G. Perrouin, and J.-M. Jezequel, “Dynamic software product lines for service-based systems,” in Proceedings of the 2009 Ninth IEEE International Conference on Computer and Information Technology - Volume 02, ser. CIT ’09, 2009, pp. 193–198.

[18] N. Carriero and D. Gelernter, “Linda in context,” Commun. ACM, vol. 32, 1989, pp. 444–458.

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

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

[21] P. François 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.

[22] E. Corporation, LNS TM for Windows® Programmer’s Guide. USA: Echelon Corporation, 1996-2000.

[23] S. F. Project, 2011-2014, http://www.aws.cit.ie/scuba/.

 ISBN: 978-1-61208-374-2