Dynamic Reconfiguration of Building Automation Systems with LINC

Laurent-Frédéric Ducreux, 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
E-mail: Laurent-Frédéric.Ducreux@cea.fr, Maxime.Louvel@cea.fr, François.Pacull@cea.fr, Safietou-Raby.Thior@cea.fr, Maria-Isabel.Vergara-Gallego@cea.fr, Oussama.Yaakoubi@cea.fr

Received: 14 November 2014 /Accepted: 15 January 2015 /Published: 28 February 2015

Abstract: This paper details a new approach for dynamic reconfiguration of building automation systems. To achieve dynamic reconfiguration in this context we rely on LINC, a coordination middleware that provides an abstraction layer allowing the encapsulation of hardware and software components. This paper shows two aspects of dynamic reconfiguration: one using the LINC middleware as an intermediation layer and one using LINC to reconfigure autonomous building management systems.

A case study is presented. It consists in 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 the proposed approach, it can now be autonomous and triggered by various external events such as sensor readings, a remote controller or information from an agenda.

Keywords: Coordination, Distributed systems, Middleware, Building automation, Dynamic reconfiguration.

1. 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, has 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-utilization, the deployment of applications, 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 proper solution for BAS typically has the following requirements:
1) The necessity of a common software layer to access heterogeneous devices;
2) The compatibility with legacy systems and protocols, to communicate with existing infrastructures and extend BAS capabilities;
3) The possibility to dynamically reconfigure the system.
Concerning the last point, several motivations for changing the BAS configuration may be considered:

- **Maintenance**: when it is required to replace a faulty device by a new one of the same type but not necessarily using the same configuration or technology;
- **Retrofit**: when new functions, services, or scenarios need to be put in place, with new devices during the lifetime of the building;
- **New usage**: when the interconnection of different and independent subsystems, not initially considered to be cooperating, is concerned. For instance, we can decrease the energy consumption of a building by using the information coming from the alarm system to infer the fact that the building is empty and control the HVAC (Heating, Ventilation and Air Conditioning) system according to this information;
- **Contextual adaptation**: when the scenarios in place need to be adapted to new building usages. This may be needed either because the user changes (more or less people, different categories of people) or because the building itself changes (space reconfiguration by splitting a room for instance).

Existing middlewares [1-3] only partially fulfill these needs. They lack of flexibility failing to provide dynamic reconfiguration and 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. LINC allows to dynamically reconfigure devices and to redefine the interactions describing how the devices cooperate.

Using LINC for BAS reconfiguration may refer to two different aspects. In the first aspect, the more obvious, LINC provides an abstraction layer on top of the heterogeneous devices and a coordination layer that permits the definition of scenarios managing the building. In this case, LINC is used as an intermediation layer. In the second aspect, LINC is used as an administration layer to reconfigure devices belonging to the same technology. In this context, the devices are self-managed and considered as an autonomous subsystem. Then, LINC is only used during the reconfiguration phase. Once the reconfiguration is completed, the control is given back to the autonomous subsystem and LINC is not used any-more.

These two aspects are complementary and, when combined, offer the flexibility required to fully mastering dynamic reconfiguration of a BAS. We have already discussed and driven the first previous papers [5-6].

This paper, which is an extension of [7], focuses mainly on the second aspect of reconfiguration. To demonstrate how we manage BAS reconfiguration, we present an example where a BAS that uses the Local Operating Network (LON) [8] 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 be performed 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 2 presents some related work; then, Section 3 introduces the main parts of the LINC middleware to make this paper self-contained. Section 4 describes the architecture put in place for the 2 aspects of the dynamic reconfiguration. The case study is described in Section 5 along with some discussions and results. Finally, Section 6 concludes the paper.

## 2. Related Work

A BAS concerns the control of building services. Extending or modifying the behaviour of an existing building automation infrastructure requires dealing with heterogeneous communication protocols and legacy systems [9]. There are lots of communication protocols and many of them coexist on the same building.

Therefore, a middleware or abstraction layer [10] is needed in such scenario. Some characteristics such 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 [11-12]. Some efforts trying to integrate legacy BAS technologies and Wireless Sensor Networks (WSN) technologies can be found in the literature [13-14]. 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:

1) **The nodes or devices level**, where the behaviour of the node is modified, i.e., sampling period, routing mechanism, sensing precision, transmission power,

2) **The system level** where relationships and coordination of devices are modified according to a new defined scenario.

In WSN scenarios much effort has been put at the node level; in this case, reconfiguration is mostly related to reliable communication between nodes and energy efficiency. Some examples of reconfiguration...
in Wireless Sensor Networks have been presented [15-16]; furthermore, a middleware may facilitate the task of reconfiguration [17]. In this paper, we focus on the system level reconfiguration which allows adapting an existing system to a new defined scenario. Therefore, new coordination rules that modify interactions between the devices 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 [1, 18] each of them focusing on a given set of applications. For the specific case of building automation Istoan, et al. [19] proposes a mechanism to reconfigure in dynamic way the relationships between devices and services in a building. This proposition remains as a conceptual approach and no real implementation is presented.

3. 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 [20] approach, bags are accessed only through three operations:

- \( \text{rd}(\cdot) \): 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;
- \( \text{put}(\cdot) \): takes a fully instantiated tuple as input parameters and inserts it in the bag;
- \( \text{get}(\cdot) \): takes a fully instantiated tuple as input parameter, verifies if a matching resource exists in the bag and consumes it in an atomic way.

3.1. 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 \((\text{sensorid}, \text{value})\). The measured quantity is then retrieved by accessing this bag. To differentiate sensors capabilities, i.e., temperature, humidity, and so on, a Type bag can be Added; this bag contains tuples formed as \((\text{sensorid}, \text{type})\). Actuators may be modelled with a bag command designed as \((\text{actuatorid}, \text{function}, \text{parameter1}, \text{parameter2})\). Then, when a resource is inserted with a \(\text{put}(\cdot)\) 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 a set of bags.

3.2. Object

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

3.3. Coordination Rules

The three operations described above, i.e., \(\text{rd}(\cdot)\), \(\text{get}(\cdot)\), and \(\text{put}(\cdot)\), are used within production rules [21]. A production rule is composed of a precondition phase and a performance phase.

The **precondition phase** is a sequence of \(\text{rd}(\cdot)\) operations which detect or wait for the presence of resources in a set of 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 \(\text{rd}(\cdot)\) operation can be used to define input fields of subsequent \(\text{rd}(\cdot)\) operations;
- A \(\text{rd}(\cdot)\) is blocked until at least one resource corresponding to the input pattern is available.

The **performance phase** of a production rule combines the three \(\text{rd}(\cdot), \text{get}(\cdot), \text{and} \text{put}(\cdot)\) 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 [22], executed in sequence. Each transaction contains a set of operations that are performed in an atomic manner.

Hence, LINC can guarantee 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;
- 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].

The Listing 1 illustrates a simple example of LINC rule. The precondition and the performance phase are separated by the symbol “::”. 

In the precondition, we verify if the sensor presence_1 has detected someone and if the luminosity is below 200 lux. Two rd() operations are used to get the information from the sensors. In the first case, we verify directly that the value is equal to "true" in the second case we compare the value of the sensor thanks to the INLINE_ASSERT token which allows the verification of a condition. If the precondition is fulfilled, the performance phase is triggered and the light is set to 80 % of its capability.

During the performance phase, we verify if the presence and the luminosity information did not changed and we perform a put() operation in the bag Actuators. The put() operation changes the state of the lights. The performance phase is executed only if all the defined operations are possible, i.e., the rd() of the conditions and the put() of the new light intensity.

A full description of the LINC middleware may be found in [4].

4. Application Architecture for Dynamic Reconfiguration

LINC naturally provides an abstraction layer to deal with the heterogeneity of connected devices. LINC also provides a coordination layer that facilitates the development of applications requiring the interaction of several BASs or independent systems. These two layers are well suited for the first aspect of reconfiguration, i.e., using LINC as an intermediate layer.

To deal with the second aspect of reconfiguration, we have defined an administration layer to encapsulate the legacy tool used to reconfigure a BAS technology. The flexibility of LINC allows managing both aspects within the same environment. Then, LINC can be used to coordinate different systems and devices and/or to reconfigure autonomous subsystems.

4.1. BAS Integration

To simplify the integration procedure and facilitate software re-use and evolution we have developed a framework dedicated to BAS technologies. A significant number of wired and wireless protocols have been encapsulated.

The framework defines features that are common to a set of devices. For instance, all WSNs share a common set of bags for controlling the sensors and the actuators. Depending of the capability of a technology, the dedicated LINC object inherits from sensors, actuators or both, as shown in the Fig. 1. Only the communication part which is dependent on the technology is different.

4.2. Reconfiguration of Devices and Systems

In several applications, reconfiguration of devices at run-time may be needed. For instance, we may need to change device parameters and/or the relationships between devices, or we may need to perform a software update without having to stop the application.

In LINC, the coordination of devices from different technologies is done through coordination rules. These rules define the different configurations and interactions between devices. For instance, what action must be done on the light when the button is pressed twice. In this case, the reconfiguration consists in replacing the existing coordination rules according to the new desired 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 framework 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 [23]. Then, the reconfiguration is performed by dynamically inserting new coordination rules to reconfigure the different controllers.

Such functionality is achieved by inheriting from a dedicated object that encapsulates the appropriate tool provided with the technology. This object
contains the basic primitives to reconfigure an autonomous subsystem.

Fig. 2 shows the example of the LON integration that inherits from the **object_wsan_sensor_actuator** and the **object_admin_LON**. Such an object is able to manage the two aspects that concern the reconfiguration, i.e., interaction with external devices and reconfiguration of the autonomous system.

4.3. Example of Application

Fig. 3 presents the architecture of a typical application using the framework. The top layer is the application layer and it defines how devices and services interact (e.g., the scenarios) with each other. These interactions make use of bags and coordination rules.

4.4. Dynamic Rules Generation

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.

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, removed, 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.

4.5. 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 that are enabled at a given moment. 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 (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 contains the rules for daily life (e.g., lights, heating and comfort management). 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. An \texttt{rd(“comfort”) or, respectively, an rd(“emergency”) on this bag is added in every precondition and every transaction to ensure 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.

5. Case Study: Reconfiguration of a Building Automation System

The context of this case study is the SCUBA (Self-organizing, Co-operative ad robUst Building Automation) project [24]. It aims to provide novel architectures, services, and engineering methodologies for robust, adaptive, self-organizing, and cooperating monitoring and control systems.

The case study presented in this paper consists in reconfiguring a LON system where a room is split in two (or merged in one). This implies to change the bindings between devices to adapt the system to the current situation: Which button triggers what and which sensor controls what.

The example presented in this work illustrates a requirement more and more important in BAS where people want to optimize the room usages while keeping comfort and optimal energy consumption. To 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.

5.1. 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 and the selected technology. 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 devices. Typically, these modifications are planned in advance and they are done on purpose by a technician.

Changes in the space configuration are something more dynamic, unplanned, and can be triggered by any occupant. Physical reconfiguration of a room needs to be synchronized with the involved devices, 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.

5.2. Overview of LON

LON [8] is a control network system designed by Echelon Corporation. LON is widely used in existing BAS 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. Fig. 4 shows an example of two bindings between three LON devices.
In the example, one 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.

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 [23]. 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.

5.3. Reconfiguration of a Room

The platform used for the case study is illustrated in Fig. 5. 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.

![Fig. 5. T1 Platform.](image)

Each office is equipped with sensors (temperature, luminosity, and presence), a HVAC system, dimmable spotlights, motorized 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.

The detection of the wall position, 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 433 MHz frequency band.

Room reconfiguration requires redefining 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.

**Encapsulation of LON in LINC**

For encapsulation, a software layer (driver) has been built on top of the LNS Server provided with the LON controller. This driver permits to access LON administrative services (for reconfiguration). The LON_object has been placed just on top of the driver layer and it contains bags associated to a specific LNS. Then, the request of 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.

**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. This rule has no precondition and it is always triggered once. Also, the rule is automatically generated from predefined configuration stored in a dedicated component called BASont [6] provided by one of the SCUBA project partners. Alternatively, this information could be stored in a database.
The second rule (Listing 4) generates the rules presented in Listing 2 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.

The rule executes as follows:
- A `rd()` operation on the bag `TriggerConfig` retrieves the configuration according to the wall position when it changes;
- 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;
- 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;
- The `generateConfigRule` method generates the LINC rule for the new room configuration following the format of the rule presented in Listing 2. The `COMPUTE` operation permits calling an external method, in this case `generateConfigRule`;
- The generated rule is put in the bag `AddRules` of `LON_Object` in order to be executed to switch to the new configuration.

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. The bridging of these technologies may provide communication between different
buildings or different rooms on a building for example.

Besides, LINC provides several characteristics that are 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 about these characteristics 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.

6. Conclusions

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 only putting a resource on a bag (send the reconfiguration command) and the system generates on the fly the rules according to the new scenario. This shows the reflexive aspect of the middleware, where LINC rules are able according to a given context to generate LINC rules that are at their turn taken into account by the system.

This paper described a case study presenting the reconfiguration of a BAS 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 devices: temperature, luminosity, and presence sensors, an HVAC system, dimmable spotlights, motorized 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. This automatic reconfiguration done through LINC brings three advantages. First, it is quicker since we go from a couple of hours to a couple of seconds. Second, it is safer since as it is done automatically, we avoid the always possible human error. And, finally, it guarantees that in case of issue in the reconfiguration we at least stay in a consistent state and minimize the inconveniences.

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.

Acknowledgments

The authors would like to thank Franck Bernier for sharing his knowledge in LONworks. This work has been partially funded by the FP7 SCUBA project under grant number 288079 and Artemis ARROWHEAD project under grant agreement number 332987.

References

[9]. W. Kastner, G. Neugschandtner, S. Soucek, H. Newmann, Communication Systems for Building


