LoRa as a secure and wireless complement to KNX

Bram Roelandts

bram@roelandts.cc

Filip Van Craenenbroeck

filip.van.craenenbroeck@5abox.com

I. Abstract

A number of other protocols could make a great complement to KNX, but the rapidly emerging LoRa protocol stands out in particular. First and foremost, LoRa communication is secure by design. Because of its high receiver sensitivity, it can either cover a large geographical area outdoors or provide deeper coverage indoors.

This project investigates combining LoRa and KNX into an iOS visualisation, providing a seamless experience to the end-user while bearing in mind the specifics of both protocols. When integrating both protocols, a few questions arise:

1. To what extend do the protocols interwork? Should LoRa sensor readings appear on the KNX bus, or do the protocols run in parallel all the way up to the iOS app?

2. How is metadata of the two fundamentally different protocols supplied to an iOS visualization in a uniform way?

3. Are there protocol-specific limitations that might have an impact on the visualization and need to be taken into consideration?

The setup for this project consists of a private LoRa network and a wired KNX network, both with different sensors and actors. A gateway capable of handling both protocols securely relays all data to a cloud platform, where the data is decoded and stored. Moving the decoding of data to the cloud inherently enhances security, because LoRa and KNX Secure encryption keys are stored in a heavily secured vault instead of on the gateway. All data is exposed to the iOS visualization through a uniform API, meaning the app seamlessly interacts with both protocols without the overhead of dealing with them separately.

Different classes of LoRa devices exist. Class A is almost exclusively used for battery-powered devices like sensors. Class C is used for permanently powered actors.
Thanks to these different device classes, a wide variety of customer requirements can be easily met with LoRa. Because of the plug-and-play nature of LoRa, both new and existing installations can be easily expanded for greater comfort and more functionality.

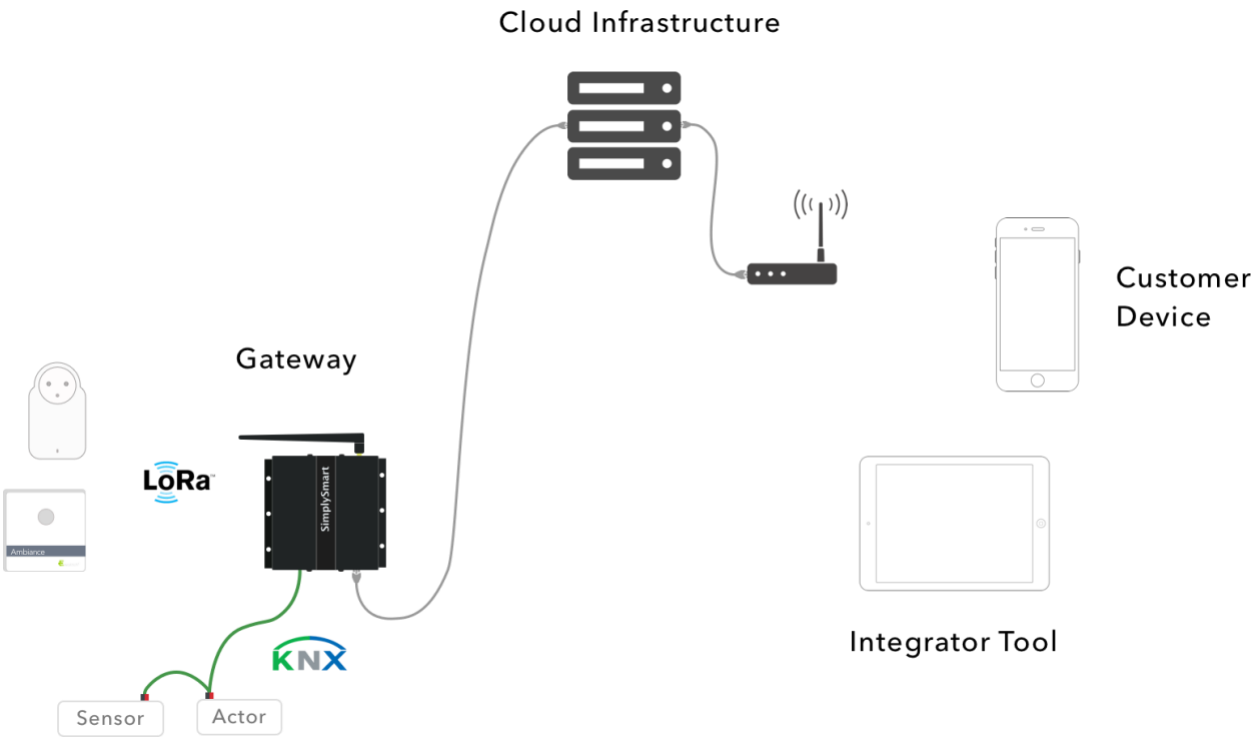
LoRa in combination with KNX Secure makes a great fit for fully secure smart buildings. Security, however, should not be at the expense of user experience. Subsequently, there is the need for seamless integration of the protocols, in order to come to fully secure, fully featured and increasingly comfortable smart buildings.

II. Introduction

The paper describes the overall architecture of the solution, the physical setup, provides some basic technical background about LoRa, the metadata handling for both protocols, the integrator tool and the client experience. To finish things off, findings are presented, along with some suggestions for possible areas for further exploration.

1. The architecture

A number of architectural choices were made from the beginning. The hardware on premise should only provide the basic functionality and connectivity. The heart of the system resides in the cloud and allows to quickly develop and deploy new functionality, based upon the assumption that the connection to gateway hardware is always available. The cloud also enables usage of the application from anywhere. The visualization is running only on iOS, as this provides the most uniform experience from a hardware and OS point of view.



The architecture is designed bearing in mind that the end-customer is no technical expert. Therefore, the involvement of the integrator is crucial to set up and configure the visualization. The 'integrator tool' is an iOS application that allows the integrator to, on one hand, leverage the setup done via the ETS file for KNX and, on the other hand, to provision the LoRa devices in a

simple way. Moreover, the integrator tool allows an integrator to prepare and customize the visualization for the end-customer.

The above picture shows a complete overview of all the different components. From actors, sensors and the gateway on the left, to the cloud solution in the middle and the iOS application for integrators and end-users on the right.

2. Hardware

A number of hardware components are used in this project: on one hand the gateway and on the other hand the actors and sensors for both KNX and LoRa.

a. Gateway

The gateway is a Linux-based embedded system equipped with a LoRa card and connected to the KNX network via the serial interface. Its main function is to relay messages between the cloud and the KNX and LoRa devices.

b. KNX devices

The KNX network has a few dozen actors and sensors connected to the gateway via a twisted pair network. There are 2 lines in the KNX network.

c. LoRa

As for physical LoRa devices, both class A and C devices were used (see chapter 4.7). Class A sensors included temperature, brightness, humidity and motion. The only Class C device was a smart wall plug with switching capabilities and active energy measurements.

d. Client device

Both integrators and end-users interact with the installation using an iOS application on iPad and iPhone. Integrators sign in to a dedicated tool for configuring the installation, while end-users enjoy a seamless experience thanks to the pre-configuration by their integrator.

3. Gateway

The gateway connects both LoRa and KNX devices with the cloud.

The main software component related to LoRa is the standard Semtech packet forwarder. Semtech is the developer and maintainer of LoRa. Along with protocol specifications, Semtech maintains a variety LoRa software component implementations. The Semtech packet forwarder's role is to relay encrypted messages between the physical LoRa radio network and the LoRa application server hosted in the cloud over a UDP socket. More details for this architectural decision are given later on.

For KNX, the choice was made to develop our own stack and relay the messages to the cloud via secure MQTT. MQTT is a lightweight publish and subscribe system, designed for low-bandwidth and high scalability.

III. The KNX stack

1. Introduction

We initially started out using knxd (formerly eibd) as protocol stack on the gateway. While knxd is open-source and has quite a large community around it, its original implementation dates more than a decade back. With KNX Secure, tweaking knxd would take too much effort. We therefore chose to develop a KNX stack from scratch.

2. Implementation

To avoid complex installation, compilation and configuration, we decided not to develop in C. Instead, we chose the relatively unknown Luvit framework (see luvit.io). Luvit is built on top of Lua, a language with a more readable syntax than C, and uses libUV for asynchronous I/O, a perfect match to handle a protocol like KNX. In addition to more readable and maintainable code, Luvit is also easily portable to other platforms. If you've a KNX twisted pair interface wired up to the serial port, it's simply a matter of installing Luvit and off you go.

3. Stack functionality

Not the whole KNX specification was implemented just yet. The implementation consists of host protocol communication with the Opternus TPUART chip and read/write from and to the KNX bus. In addition to standard serial communication, basic KNXnet/IP communication was implemented. The ability to program KNX devices over a remote connection is a must-have, so KNXnet/IP Tunneling is also present.

While we deliberately decided to leave out certain optional KNXnet/IP service families for now, there is still some work to be done on mandatory families, too. They are mostly small things, an example being the repetition of a telegram when no ack is received.

Because writing a protocol stack is fairly tedious and requires thorough testing, we are also considering the possibility of open-sourcing our implementation. This would allow for more extensive testing and contribution by third-parties, helping the stack to mature faster.

4. Gateway interface

As explained later in this document, all metadata, like group addresses and their data point types, was moved to the cloud. In some cases, it actually makes way more sense to embed logic right on the gateway, so that's still possible, too. In a NodeRED-like environment (see nodered.org), where programming is done via a graphical user interface using nodes and flows, we've made it possible for integrators to interact with the KNX installation in a very straightforward way. We've made five nodes to offer integrators all the tools they need.

- **KNX Write**: Takes a group address and binary value as parameters, converts the parameters into a telegram and sends the telegram on the bus.
- **KNX Read**: Takes a group address as parameter and directly sends a value read request on the bus.
- **KNX Listen**: Outputs any telegram coming from the bus. Output includes source address, destination address and the binary payload.
- **KNX Encode**: Takes a DPT and plaintext value as parameters, and outputs an encoded binary value.
- **KNX Decode**: Takes a DPT and binary value as parameters, and outputs a decoded plaintext value.

Thanks to these simple yet powerful nodes, integrators can create interesting flows with ease. Example use cases are a flow getting the current time from the system and sending it onto the bus every 12 hours and a flow sending out an email when a leak has been detected by a KNX sensor. The possibilities in this environment really are rather extensive.

5. Functionality overview

KNXnet/IP Services (server only)		
	Core	Supported
	Device management	Supported
	Tunneling	Supported
Bus communication over the serial port		Supported

IV. LoRa

1. Chapter overview

- Exploring if and how LoRa complements KNX
- Introduction to various LoRa network components and their function.
- LoRa device provisioning
- LoRa device classes

2. Why LoRa?

KNX remains the de facto standard for building automation. Especially with the recent addition of KNX Secure, KNX is the preferred choice for new installations. When it comes to upgrading existing installations, however, there is an increasing demand for a plug-and-play wireless solutions. To meet today's standards, the wireless solution needs to be secure by design, have a reasonable range and consume very little power, opening a whole new world of possibilities with battery-powered devices.

The rapidly emerging LoRa protocol immediately stands out, as it meets all of the above criteria. LoRa provides AES-128 encryption, has an outdoor range of up to 10 km and already has dozens of manufacturers making various battery-powered devices.

In addition to matching the above criteria, there are two additional factors making LoRa a very attractive choice. Firstly, LoRa devices are, just like KNX, produced by several different companies, eliminating dependency on a single manufacturer. Secondly, the original developer of LoRa, Semtech Corporation, actively maintains a variety of open-source LoRa software for the client and the gateway. This makes it easier to get started with LoRa.

3. Field of application

LoRa complements KNX because of a few reasons. LoRa has a high receive sensitivity, is mostly battery powered, with a lifetime between 5 and 15 years, and offers solutions within the domain of analysis, smart metering, volume measurements and tracking. LoRa offers additional services in certain fields, sometimes with easier installation. A number of applications LoRa is well-suited for are listed in the below table.

Analysis	Smart meters	Volume monitoring	Device tracking
Air quality	Electricity	Glass/garbage container	Movement
Temperature	Water	Oil/Fuel tanks	Location
Humidity	Gas	Food containers	Tagging
Presence	Renewable energy		

4. Network components

Network server

The network server peers with all device endpoints (via LoRaWAN) that are configured to be members of applications defined by the server, and are connected via gateways within their coverage area. There is no given standard for how a network server is located within the LoRaWAN network topology. It may be embedded within a LoRa gateway, or it could be co-located with application servers in a data center.

The network server provides three key functions:

- Authentication and authorization of devices
- Management and optimization of the network
- Interfacing with upstream application servers

The network server receives messages from devices, managing device authentication, application message routing, and management of gateways and devices. It will dynamically select the best gateway for device data routing, as well as de-duplicating packets and optimizing radio resources via the Adaptive Data Rate (ADR) function.

Application server

The Application Server handles all the application layer payloads of the associated 3 end-devices and provides the application-level service to the end-user. It also generates all 4 the application layer downlink payloads towards the connected end-devices. Depending on the implementation the message payload is made available via interfaces like MQTT, HTTP or other.

For more detailed information, please refer to the LoRa specification [1], which can be obtained upon request.

5. Network management and optimization

If supported by an end-device, the network server can perform Adaptive Data Rate (ADR) adjustments, based on the received SNR (Signal to Noise Ratio) of each device. In effect, the use of ADR results in devices spending less time on air, improving radio resource efficiency, whilst also managing the reliability of messages.

As illustrated in the network architecture, duplicated messages can be received at the network server from one device transmission that is received and forwarded by multiple gateways. The network server should de-duplicate these messages and respond to, or forward the message once. See [2].

6. Device provisioning

The LoRaWAN specification defines two ways to provision devices.

OTA, which stands for Over-The-Air activation and uses a pre-shared application key. A network session key and application session key are derived from this application key. A LoRa device sends out a join request and then derives the keys from the join response. If the session keys are compromised, it's simply a matter of re-joining the network.

ABP, which stands for Activation-By-Personalization, uses pre-shared session keys. An advantage of ABP is that devices don't have to join the network and don't need the processing power to derive session keys. A disadvantage, however, is that the device has to be physically re-programmed in case the session keys would get compromised.

[1] *LoRa Backend Interfaces. (2018). LoRa Alliance.*

[2] *GitHub. (2018). LoRaWAN Overview. [online] Available at: https://github.com/Fluent-networks/floranet/wiki/LoRaWAN-Overview [Accessed 10 Oct. 2018].*

7. Device classes

The LoRaWAN specification defines three device classes. The key difference between classes is how frequent they come online and are able to receive messages. A timeframe in which a LoRa device is online is called a receive slot.

Class A The communication is always initiated by the device and the gateway can only respond. The device opens up two short receive slots immediately after it transmits a message. In between these transmissions, typically every few hours or even days, the device goes into deep sleep. Class A devices can last multiple years on a single battery charge, because the LoRa radio is turned off most of the time.

Class B devices open up more receive slots compared to class A. In addition to the slots immediately after transmission, class B devices have scheduled receive slots at fixed times. Very few devices to this day actually use class B.

Class C devices have continuous receive slots, only closed when the device itself is transmitting. This allows the gateway to reach the device at any time and not wait for the device to contact the gateway before anything can be sent back. Because the LoRa radio is always online and constantly draws power, running a class C device of a battery is practically impossible.

The functionality of a LoRa device is implicitly constrained by its device class. Class A is commonly used for measurements and occasional events. Because one can only transmit to the device when it comes online, class A type devices can't be used for switching lights, for example. Class C devices have continuous receive slots, so switching a light using class C happens within seconds

8. Implementation

The combined network/application server is hosted in the cloud. For every installation, a new instance will be created. The gateway will only relay the message from and to the cloud.

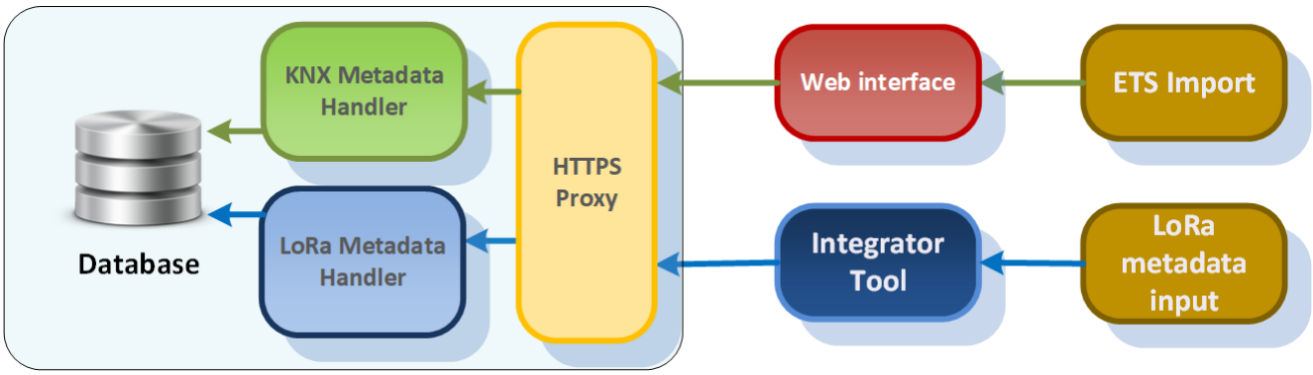
During the provisioning phase, via the integrator tool the keys are sent to the application server and stored in a secure vault.

Another possibility could have been to implement the network/application server directly onto the gateway. This would have required major modifications to the hardware to ensure the secure storage of the keys.

V. Metadata handling

1. Introduction

The metadata flow is handled in the way as presented in this picture:



2. KNX metadata

The ETS .knxproj file serves as a basis for input of the metadata. The integrator uploads the project file using a web interface, the backend extracts information from the file and stores it in the cloud. That metadata information is then retrieved by the integrator tool.

Templates allow the integrator to input their group address structure into the integrator tool. Some structuring possibilities were presented in the document 'KNX Project Design Guidelines' by the KNX Association. The template will allow them to link different group addresses into meaningful and useable objects (switchable light, dimmable light, RGB lighting, blinds, curtains, heating, etc.).

Anything that falls outside the predefined template, can still be individually mapped. While an integrator may reuse their structure across projects, every installation is slightly different. Manual mapping accommodates these small changes, while retaining the same template. The integrator is also free to leave out any group addresses that are not relevant to the visualization.

The only remaining manual effort is to assign a space (room and floor) to these meaningful objects. Although building information is available in the ETS project file, an actor often is not physically present in the room where the function takes effect. While the binary actor controlling the kitchen lights might be in the basement, the end-user thinks of these lights as being in the kitchen.

Using the ETS project file as the primary source of information avoids any tedious replication for the visualization. Moreover, any changes to the ETS project file simply require a re-import of

the file on the web interface. Even in the early stages of the installation, the file can be uploaded and a basic visualization is already available. Later on, changes to the installation do not require the integrator to simply start over, as they are incorporated into the visualization in a simple and straightforward way. This makes the visualization a core part of the installation, rather than an add-on.

As a starting point, each object is named as found in the ETS file. Both integrators and end-users can further customize names of objects and spaces. This allows the customer to further tailor the visualization for an even more personalized experience, as each user can individually make on-device changes.

3. LoRa metadata

The interpretation of a binary payload sent by LoRa device is entirely defined by the manufacturer. Based upon information from the datasheet, messages can be properly encoded and decoded. To spare the integrator the bit-by-bit decoding, a device library was put into place. The integrator can choose from a list of supported devices and the backend will automatically take care of the bit-by-bit interpretation for that device.

4. The LoRa device library

Every manufacturer has their own interpretation of bytes sent by their devices. To overcome this roadblock, a library of LoRa device profiles was put into place. The library is frequently updated with the latest device profiles and dynamically updated from the backend to the integrator tool.

In a way, you could think of items in the library as KNX Product Files, but applied to LoRa. Let's say you add a new device, enter its keys and select 'NKE Watteco Smart Plug' as device model. The integrator tool then knows this LoRa devices contains a binary switch component and four different measurements (Active Power, Active Energy, Reactive Power and Reactive Energy). The backend infrastructure in its turn knows how to interpret any bytes coming from this device. Because LoRa payloads are completely freeform, there simply is no way around manually building up a library of devices and how to decode their message payload.

Building and maintaining such a library requires the involvement of the LoRa device manufacturers, who supply device information and payload decoding. This information already exists in the datasheet, but still at present still requires manual addition to the library. The current library only contains a fraction of devices currently available on the market. On one hand, devices can be added upon integrator request. Another possibility would be to allow third-parties to add new devices by supplying a file containing device information and decoding. While this is currently out of scope, it might become relevant in the future.

VI. Integrator tool

1. Introduction

The integrator is without doubt an important person throughout the lifetime of an installation. Not only did he configure the KNX installation using ETS, he also installed and provisioned the LoRa devices. He perfectly knows how he configured each and every component of the installation, so it is only logical to have him prepare the visualization for end-users.

Given the importance of the integrator, a dedicated tool for integrators became the centerpiece of the visualization. Our goal with this integrator tool is to make the integrator's life as easy as possible. The integrator enters his address structuring logic, uploads the initial ETS file and is good to go. Later modifications made in the ETS file only take a few minutes to import once the initial configuration has been completed. When the configuration is ready, it's simply a matter of releasing the configuration to the end-user with the press of a button.

2. In a nutshell

What exactly does the integrator tool do?

- Leverage metadata from the ETS project file. This metadata includes group addresses and their DPT, physical devices and building information.
- Allow an integrator to insert their group address structuring logic.
- Using that logic, combine a number of group addresses into a single function. For example, two addresses 'on/off kitchen lights' and 'fb kitchen lights' are combined into 'kitchen lights'.
- Manually tweak and fine-tune the configuration. There might be some exceptions to the address structure and some functions might need a more user-friendly name. The integrator tool allows you to override the ETS file to achieve an even better experience.
- Map building metadata not present in an ETS file. ETS does provide information about the location of physical actors. The function 'kitchen lights', for example, might be controlled by an actor in the basement. End-users, however see 'kitchen lights' to be in the kitchen. That's why the integrator has to manually map all functions to a room in a building.
- Add new LoRa devices. Enter their associated keys to provision the device, either using ABP or OTA. Select the make and model from our LoRa device library, which automatically contains information about what functions this LoRa device offers and how incoming bytes should be interpreted.
- To configure settings directly on a LoRa device, the integrator tool offers a chat-like commissioning tool.

- Release the whole configuration to the end-user(s) with the push of a button. Once the end-user opens the application, the changes are applied instantly.
- Later on, if the installation changes and evolves, the integrator simply uploads the latest ETS file. All manual modifications and address logic are seamlessly applied to the new project file.

VII. End-user visualization

1. Introduction

As mentioned at the beginning, the goal is to offer a seamless experience towards the end-user, combining KNX and LoRa. The application can be downloaded from the App Store and installed on multiple iOS devices . Once the user logs in, the configuration prepared by the integrator will serve as base configuration for every device. The configuration can be tailored on every device according to the end-customer needs. In this device-centric approach, any changes are kept locally on device and every user customizes the visualization how they like.

Different from other visualizations, the experience is primarily focused on the ease-of use of the combined KNX and LoRa installation. Complex integrations like audio and video were currently left out to keep the visualization minimalistic and straightforward. The main benefit to the user is a straightforward way of interaction with their installation and additional functionality like scenes and schedules.

Instead of remote access being optional, it is built right in. Whether the end-user is on local Wi-Fi or a few thousand kilometers away, the visualization works in exactly the same way. No dabbling with port forwarding on the customer's router, no additional steps to enable access from anywhere.

2. The visualization in a nutshell

Favorites: Group your favorite devices and control them all from one screen, anywhere you happen to be in the world.

Rooms: Access and control your devices by room and by floor to see all of the devices in your installation.

Scenes: We've taken a fresh approach to scenes. Users define them, directly in the app. They're easily customized, any time the user wants, from anywhere. No more calling the integrator every time the user decides to adjust the lights.

Scheduled tasks: Create a scheduled task to execute daily, weekly, yearly, or just once. Since these tasks affect everyone in the environment, every user can see and edit them.

Notification center: With the notification center widget, users don't even have to open the app to control their installation. Just swipe to the right on the lock screen and control up to nine favorite devices.

Shipshape: Did you leave lights on or the windows open? Find out instantly in one simple overview, and with just one touch of a button, make sure everything is shipshape. Instant peace of mind!

Authentication: With facial or fingerprint recognition, the visualization provides another layer of security to the sensitive parts of your installation, such as access and alarms. The installation has end-to-end security and global control.

3. Leveraging the power of iOS

Integration with Siri Shortcuts

Siri Shortcuts, newly introduced in iOS 12, allows users to create scripts on their iOS device. Power users can easily create a shortcut which for example orders a coffee in the Starbucks app, determines the travel time to work depending on traffic and plays the 'Morning' playlist on the living room speaker. While platforms like IFTTT (see ifttt.com) support time-based workflows and various external triggers, Siri shortcuts can only be invoked using Siri or by tapping them. To run the above example shortcut, the user could simply go like "Hey Siri, morning routine".

For users to access third-party apps in their shortcuts, the respective app developers need to integrate with the SiriKit framework first. For our visualization, integration with Siri Shortcuts could mean controlling devices in the building and activating scenes. The user from the example above could then extend their morning routine by also turning on the kitchen lights, dimmed to 80%. While, at present, the project does not yet integrate with SiriKit, it certainly is on the roadmap.

Integration with Siri Intents

Although Intents are technically a part of Siri Shortcuts, they deserve a paragraph of their own. Intents allow third-party apps to take advantage of the advanced machine learning built into iOS. Let's say there is this app for ordering pizzas to your doorstep and the app developers integrated it with Siri Intents. When the user orders a pizza at lunchtime every day, by the end of the week iOS will suggest on the lock screen to buy that very same pizza like previous days. In this project, the app would notify the system every time a scene is triggered. If the system detects a certain pattern, it will automatically suggest triggering that scene on the lock screen at the right time.

4. Visualization conclusion

The best part of the whole visualization? There's absolutely no distinction between KNX and LoRa devices. Selecting favorites, creating scenes and schedules and adding functions to the notification center all work protocol-independent. Were it not for the slight latency of LoRa class C devices, users would have no way of finding out which protocol powers what. Does that mean this project successfully accomplished its goal? In a way, it does. KNX is now wirelessly and securely complemented by LoRa, thanks to a uniform metadata structure and streamlined communication API.

VIII. Summary

In this paper, we presented how a protocol like LoRa can be seamlessly integrated into a visualization allowing new use cases to be handled. The implementation is based on a cloud architecture and currently runs on iOS only. A key component is the integrator tool, which allows leveraging the ETS .knxproj file information and uses a device metadata library for LoRa, but also allows integrators to pre-configure the visualization before it is handed over to the customer. Right from the initial setup and throughout the lifetime of the visualization, it becomes easy to maintain and update the visualization.

IX. Conclusion

The proposed architecture can be extended to include other protocols, for example Modbus, BACnet, M-Bus, etc. The complexity is hidden and, depending on the use case, the integrator is able to offer new services without being a protocol expert.