Reliable control network gateways for building automation networks

Stefan Seifried, Wolfgang Kastner
Automation Systems Group
Technische Universität Wien
{sseifried, k}@auto.tuwien.ac.at

***Abstract*—In building automation systems, the diversity of existing field level networks is a major challenge for fully integrated solutions. Typical approaches achieve integration by shifting the needed complexity to the automation or management layer. In contrast, this paper focuses on leaving the complexity at the field level to provide a homogeneous view to upper layers without compromising reliability. Therefore, a gateway device architecture is advised which enables the seamless integration and communication between field level protocols of building automation networks. The necessary translation between field level protocols is done by utilizing a stateful solution incorporating a unifying information model. The reliability aspect is achieved by combining several gateway devices into a redundant compound.**

I. Introduction

Building Automation Systems (BAS), as a special case of automation systems, are concerned with control and management of building services. The domain core areas are heating, ventilation, air conditioning, lighting/shading. The ultimate goal is to reduce energy while maintaining a certain level of comfort. Also safety and security have to be guaranteed. Automation systems in general have developed a large variety of specific communication protocols tailored to diverse application domains. In BAS, the emergence of Ethernet and IP resulted in a consolidation of communication protocols at the management level. On the other hand, the field level still consists of a large set of maintained and well-established solutions, that are not easily replaceable by Ethernet and IP due to special requirements (cf. requirements given by standards for cabling and installations in buildings) [1].

The diversity of automation system communication protocols gives rise to the integration problem at the vertical and horizontal dimension of the automation pyramid. While the challenge for the vertical integration is information processing across the levels of the automation pyramid, the difficulty in the horizontal direction is in the information translation between disparate communication protocols of the same level. BAS form a specific application domain which enables the construction of a homogenous view, despite different implementations of concepts for specific field protocols [2]. A common information model allows the conversion of information artifacts between field level communication protocols. Another challenge arises in automating critical infrastructures. Specific reliability mechanisms are leveraged by field level protocols and single points of integration (at the vertical or horizontal dimension) would compromise a holistic reliability concept along the automation pyramid. Therefore, a sophisticated integration concept must also consider multiple junctures between or in the layers of the automation pyramid.

While integration at the vertical dimension of the automation pyramid was subject to active research ([2], [3], [4]), the integration problem remains undiscussed for the horizontal dimension at the field level. The goal of this work is to advise a horizontal integration solution, while possibly retaining the level of reliability to interconnect critical infrastructures at the field level of BAS. One major challenge is the communication translation between field devices, part of disjoint networks using different protocols. The difficulty herein lies in rigorous timing requirements, the different modes of communication, event driven and polling, and that communication mechanisms are usually not stateless. The second objective is to achieve the strongest possible fault hypothesis for the interconnected network. Hence, the overall fault hypothesis shall be imposed by the weakest communication protocol and not the integration solution itself.

A promising approach is a gateway device which enables the exchange of information between otherwise disjoint field level networks. The communication networks linked by the proposed gateway device are then referred to as interconnected network. Another important property is seamless integration, meaning that the gateway shall integrate into field networks behaving like a native appliance. A single gateway device would likely compromise the fault hypothesis of an attached field level protocol. Therefore, we propose several such gateway devices are joined together into a redundant compound, called Virtual Redundant Gateway (VRG). The number of gateway devices of a VRG must then be sufficient enough such that the resulting fault hypothesis matches the fault hypothesis of the weakest interconnected field protocol.

This paper limits itself on the basic architecture of such a gateway device. Next, the two main aspects, interconnection (see Section III) and reliability (see Section IV), along with a fault analysis (see Section V) are discussed. Setup and maintenance topics are left out to decrease the overall complexity of the presented concept. Section VI presents a proof-of-concept combining the technologies *KNX* and *ZigBee* to an interconnected network.

II. System Architecture

There are two main tasks that a gateway device, part of a VRG, needs to fulfill. First, it needs to forward and translate the communication between the attached field networks. Second, it has to coordinate its operation with all other members of a VRG.

The interconnection aspect of the gateway solution requires a single device to implement an interface to every attached

field network, as depicted by Figure 1. Integrating the proposed gateway device then as a full member of the individual field networks enables seamless integration without requiring any specific knowledge by other members of the field network. A distinct gateway device needs to enable a stateful communication translation process by storing exchanged information at a local database. Since it can neither be assumed that the interconnected protocols implement the same mode of communication, nor that the utilized communication mechanisms are stateless. Furthermore, the data storage at a gateway device must follow an information model which enables a homogeneous view between all targeted field level protocols.

Seamless integration between otherwise disjoint field networks utilizing different protocols is then achieved by mapping the homogeneous representation into a field network specific view. Hence, messages targeted to a field device from another fieldbus are translated and forwarded with the help of the gateway's local database and its information model. Hence, as depicted in Figure 1, every device in *field network A* that is to be visible in *field network B*, and every device in *field network B* that is to be visible in *field network A* is modeled by means of the information model. The modeled device is then presented like a true native communication partner to the other field protocol, thus achieving seamless integration.

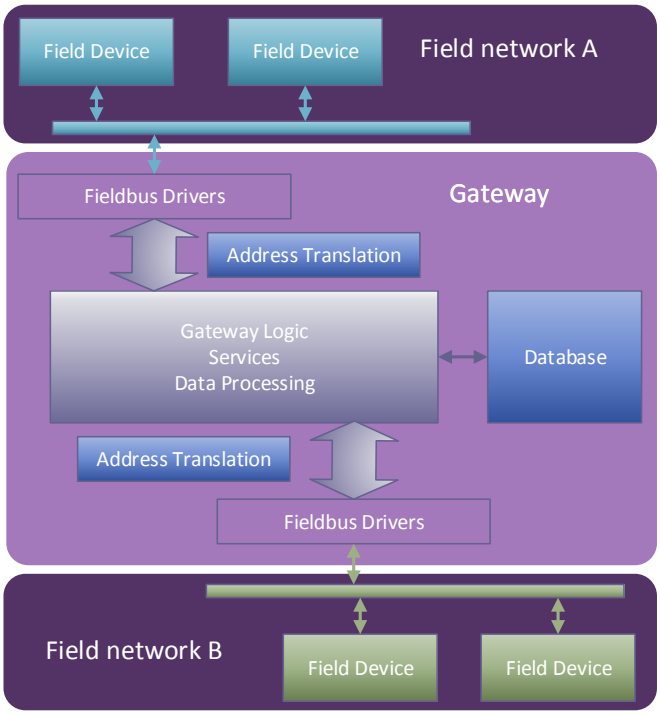

Fig. 1. Gateway Device and Network Topology [5]

For industrial automation, a well defined classification of modeling approaches for mapping foreign entities to another field protocol has been developed by the *INTERBUS Integration* working group [6], in order to map *Interbus* devices to the *Profinet IO device model*. Basically, there are three approaches:

- **Compact topology:** The compact topology concentrates all available field devices in a single modeled device and promotes a monolithic information model design. Hence, all topology related information regarding the modeled network is lost, and it is only suitable for small instances of field networks.
- **Logical topology:** A logical topology promotes the grouping of functionality in entities at the gateway model. Therefore, devices from the originating field network can be combined at the information model.
- **Physical topology:** The physical topology based information model utilizes a strict bijective mapping between field devices and their modeled equivalents.

The topology most relevant to the proposed gateway approach is determined by the instance of the integration problem. The integration of a small field network instance may not require any preservation of the topological structure and can be easily modeled by means of the *compact topology*. On the other hand, another problem instance may require the modeling according to the location of field devices in buildings and hence a *logical* or *physical topology*. Therefore, a well-designed information meta-model must enable the use of all three presented topologies.

The reliability aspect of the proposed solution is realized by following a replication approach. Several single gateway devices are linked together in a redundant way, forming a so called VRG. Hence, the inner state of all gateway devices among a VRG must be synchronized and coordinated. The necessary traffic is distributed across the available field network connections. On the one hand, this eliminates the need for a separate link and additional cabling efforts. On the other hand, a traffic overhead is introduced which might interfere with normal field network operation.

There are two roles foreseen for gateway devices part of a VRG, *master* and *slave*. The *master* is a single, distinct gateway which is responsible for forwarding and translation of traffic between interconnected field devices. Hence, every other gateway, part of a VRG, is a *slave*. A *slave* gateway does not actively participate in ongoing communication between devices of different networks, but is permanently synchronized with the inner state from the *master* gateway. The task of the *slave* gateways is to monitor the *master* for both device faults and link faults at its field network interfaces. Further, if a fault at the current *master* gateway is detected, the *slaves* immediately elect a successor to the *master* among all correct functioning *slave* devices of a VRG.

III. Information Model

As described in Section II, direct message translation between field level protocols attached to a gateway device is infeasible due to timing constraints and differing modes of communication. A general applicable solution is proposed, by establishing a homogeneous view by the means of a protocol independent information model. The purpose of the information model is the storage of runtime data of all attached field networks and to enable seamless integration.

The design of the information model must enable all the different modeling approach topologies outlined in Section II as well as a separation of concerns regarding field protocol specific information (e.g. address information) and user data (e.g. datapoints). Therefore, the well-established Model View

Presenter (MVP) software design pattern is leveraged [7]. The MVP pattern, a sibling from the well known Model View Controller (MVC) pattern, is a hierarchical approach and splits the model into three partitions. The *model* partition defines the available user data and offers information update mechanisms and state-change notification services. The *view* partition is the front-end presented to a single attached field network and is responsible for forwarding service calls to the *presenter*. The *presenter* partition models the field protocol specific information and acts as an abstraction between the user data and a field network instance. Therefore, two disjoint field networks realized by the same protocol attached to a VRG share different *views*. Hence, different entities and functionalities may be exposed.

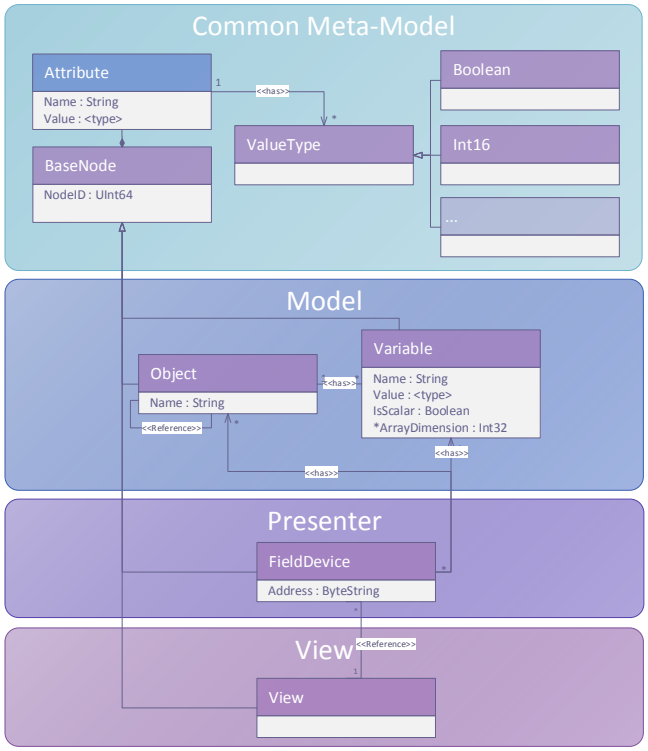

Fig. 2. MVP based Information Model

A. Model

An appropriate meta-model describing the *model* part must consider complex data structures besides common primitive types. Advising a design from scratch is not a good practice, given the vast amount of available field level protocols that must be considered. Hence, the meta-model specified by OPC Unified Architecture (OPC UA) [8] is used as a reference, but is trimmed down to the needed subset of features.

The fundamental entity of the proposed meta model is the *BaseNode*. Elements derived from *BaseNode* can be directly addressed using the *NodeID* attribute. The value of the *NodeID* must be unique throughout the whole gateway device. *Attributes* are simple key-value pairs that model constant parameters further characterizing an entity. An *Attribute* consists of two elements, a *Name* and a *Value*. The *Name* identifies an *Attribute* uniquely in the context of an arbitrary *BaseNode* derived entity type. Thus, no two *Attributes* with the same *Name* are allowed to exist throughout an inheritance relation. The *Value* part of an *Attribute* is encoded according to one of the defined *ValueTypes*, also used for data *Variables*.

A *Variable* is an entity derived from *BaseNode* and resembles arbitrary user data values. The set of *Attributes* consists of mandatory properties *Name*, *Value*, *IsScalar*, and an optional *ArrayDimension*. *Variable* types are intended to be put in the context of an *Object*. Also *Variables* are foreseen that are not tied to an *Object*. The modeled *Value* of a *Variable* is either a scalar or an array type, determined by the *IsScalar* attribute. In case the *IsScalar* property is set to *false*, the *ArrayDimension* attribute is required. The *ArrayDimension* defines the fixed number of values this *Variable* may hold.

The purpose of an *Object* element is to group contextual related *Variable* types into a single entity and to further allow the modeling of deeply nested data structures. An *Object* is allowed to have an arbitrary number of *References* to other *Objects* or *Variables*. The only mandatory *Attribute* for an *Object* is the *Name*, so it may be identified in the context of another *Object* without knowing its *gateway* wide unique *NodeID*.

The *ValueType* element represents the supported data encoding formats of the *Value* entries for both *Attribute* and *Variable* entity types. The available *ValueTypes* are predefined by the meta-model and are a subset of the simple datatypes already predefined by OPC UA.

B. Presenter

The *presenter* partition models the field protocol specific data. This is done by adding an additional *BaseNode* derived type, called *FieldDevice*. The only mandatory attribute of the *FieldDevice* is *Address*, and represents the native address for a single field level protocol. It is encoded in the modeled field protocol's native encoding and is therefore stored as an arbitrary *ByteString*. Additional information can be linked to *FieldDevices* in form of *Variable* entities. Further, functionality can be grouped by the means of the *Object* class.

C. View

The *view* partition models the view presented to an attached field network. An instance of a *View* entity references all instances of *FieldDevice* designated for an attached field network. Therefore, a protocol binding layer must be implemented at the gateway, which is capable of interacting with both the native protocol stack and an instance of the *View* entity.

IV. Gateway Architecture

The reliability aspect of the VRGs proposed replication approach is based around a holistic *fault hypothesis* for the interconnected network. Each field network and the VRG itself form a so called Fault Containment Region (FCR). Faults are contained within their respective FCR, unless the *fault hypothesis* is violated. For an interconnected system of FCRs, the FCR with the weakest *fault hypothesis* determines the *fault hypothesis* for the interconnected system. Therefore, it is required that the VRG is designed so that its *fault hypothesis*

is at least as strong as the weakest one of all connected field level protocols.

As mentioned in Section II, all replication and coordination traffic of a VRG shall be handled by utilizing the interconnected communication network in order to fulfill the *seamless integration* property. Therefore, three building blocks are foreseen to aid the gateway in distributing traffic and detecting VRG faults: The bus guard, the bus load monitor and the redundancy manager.

A. Bus guard

The *bus guard* is responsible for detection and reporting of link faults at an arbitrary field network interface of a gateway device. Therefore, an end-to-end connection monitoring mechanism is utilized, by sending *heart-beat* messages between the *slaves* and the *master* gateway members of a VRG. The monitoring transaction is started by a *slave* gateway device sending a *heart-beat* message and is closed when an acknowledge message has been received from the *master* gateway.

The time between two subsequent *heart-beat* messages is determined by the *HeartBeatInterval* parameter. Also a *grace counter* with an upper limit of *GraceLimit* subsequently failed *heart-beat* messages is foreseen. Hence, a link fault is detected and reported after a time span of $GracePeriod = GraceLimit \cdot HeartBeatInterval$. A successful *heart-beat* transaction resets the *grace counter* to zero.

The proposed *end-to-end connection monitoring* method reliably detects link faults between two different gateway devices. However, it is not capable of determining the gateway at which the fault occurred or if the link between the involved interfaces is faulty. A state of the art solution is to add a third communication partner part of the field network under test. However, instead of a *heart-beat* message a similar service like a *ping* message used in Ethernet based networks is required. A *ping* can be any field protocol telegram with a defined payload that is acknowledged and does not modify the state or trigger any action at the receiving node.

The communication partner utilized for such *ping* messages is called Virtual Reference Node (VRN). Dependent on the fault hypothesis of the field network under test, the VRN consists of a single or several fieldbus participants. A call order has to be defined for the members of a VRN at a VRG prior to the activation of a gateway field network interface. The VRN call order enables consistent results of the *ping* service in case one of the VRN members fails, without having to contact every member of the VRN.

B. Bus load monitor

The *bus load monitor* provides a metric that enables an appropriate distribution of the synchronization and coordination traffic induced by the usage of a VRG. The purpose of this component is to avoid to put strain on an attached communication network that already experiences high amount of traffic.

The implementation of the *bus load monitor* is highly dependent on the specific field protocols of the interconnected network. One possible solution is to listen to passing traffic on a network interface and derive a metric proportional to the available bandwidth. However, such a metric is only able to estimate the load situation at a local network segment. Hence, the stated approach is only feasible if the sniffed segment incorporates the members of the *VRN* and the *VRG*. Another load monitoring approach is to measure the round trip time of busguard messages. As a prerequisite, the idle round trip time has to be known to derive a meaningful metric. Moreover, the communication path of the telegrams has to be deterministic.

In conclusion, the implementation of the *bus load monitor* and its metric needs to be tailored to a specific field network. In any case, a good overall load metric needs to combine several applicable methods and generate a weighted average of the individual outcomes.

C. Redundancy manager

The *redundancy manager* component handles the information model replication and coordination among all members of a VRG. Wired fieldbus networks mostly allow a *passive* replication approach, where *slave* gateways sniff communication traffic. However, wireless field protocols require an *active* replication solution, since it is not guaranteed that the same messages arrive at all gateways equally. Although both replication mechanisms are foreseen for the VRG, the *active* approach is discussed in detail, since it is generally applicable.

In case of an *active* replication mechanism, a field level protocol needs to foresee some kind of generic message type. This generic message must be capable of unicast communication and allow an arbitrary payload. The replication procedure itself is triggered by an arbitrary *data_change_notification* occurring at the *master* gateway. The *redundancy manager* then decides by the metric given by the *bus load monitor* which network interface will be used for the synchronization transaction. A native field telegram, encapsulating the synchronization frame, is then sent via the chosen field network interface to all the *slave* gateway devices. A receiver of such a field message forwards the synchronization frame to its *redundancy manager*, which will update its local information model.

Another important aspect of the *redundancy manager* is to maintain proper operation of the VRG according to the fault hypothesis. As already stated, the *master* gateway is responsible for replication and the *slave* gateway devices are responsible for fault monitoring. However, it still remains to be discussed, how a gateway device can assume the role of a *master* gateway. A possible algorithm to solve this task is presented in Figure 4.

All correct functioning members of a VRG need to reach a conclusion about the new master with a minimal amount of communication traffic. A simple implementation of such an election is based on an arbitrary fixed priority order between all members of a VRG. Such an order can be achieved by assigning a fixed unique number to every *gateway*, member of a VRG, in advance. When the election process is triggered, every *gateway* device, member of the VRG sends its own priority in form of a *master_advertisement* message to all other devices part of a VRG. After the *MasterAdvertisementTimeout* has passed, no new priority values are accepted and the gateway device with the highest advertised priority value is elected as new *master*.

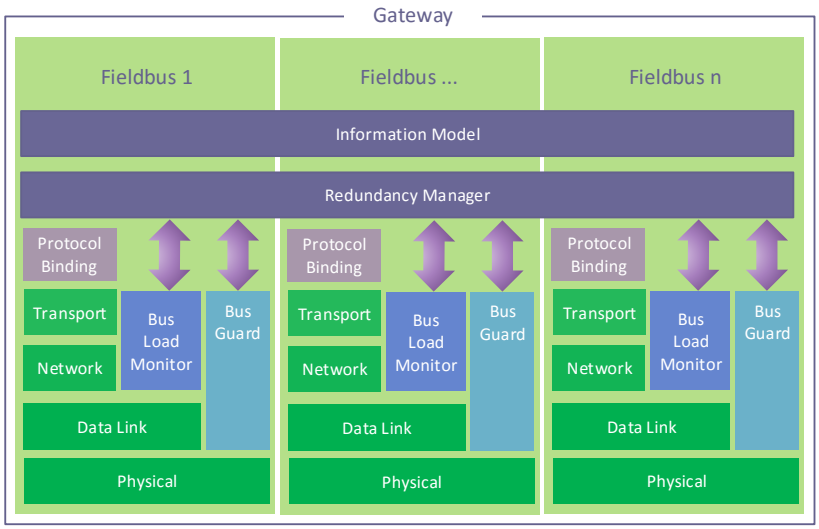

Fig. 3. Gateway Architecture

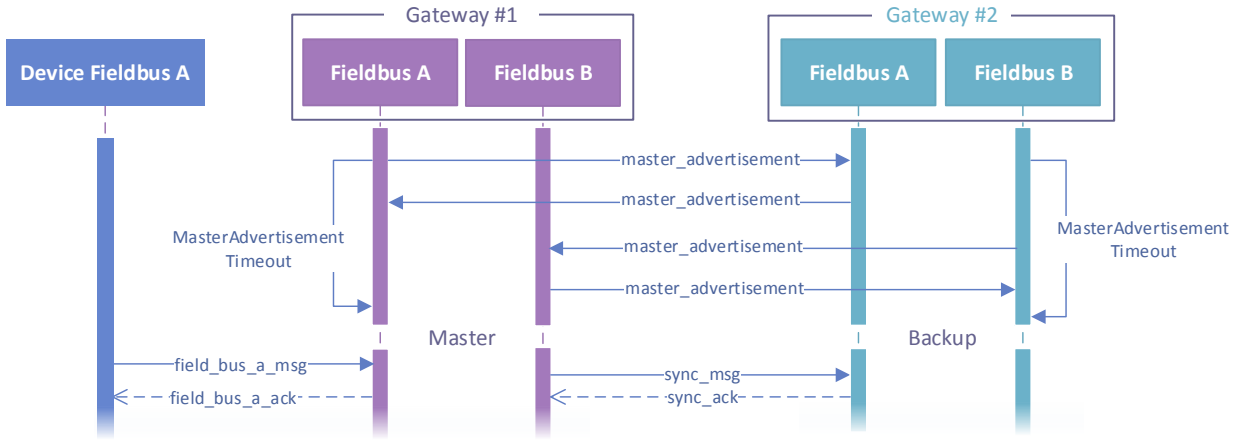

Fig. 4. Election Algorithm

The main requirement for a reliable and consistent election mechanism is a *reliable group communication*. Either a *master_advertisement* message is delivered to every participating, correct functioning *gateway device* or to none. Otherwise, the list of priority values may differ from *gateway* to *gateway* and the election process is inconsistent.

The election process itself is triggered by the *bus guard* or a received *master_advertisement* message. When a *master gateway* receives a *master_advertisement* message, it assumes that at least one communication link failed. Further, the old gateway *master* assumes the role of a *fail-silent* device and does no longer participate in any communication.

V. Fault Analysis

The fault analysis of a VRG is shown by a case analysis. Without loss of generality, it is assumed that the VRG consists of two distinct gateway devices. A fault at an arbitrary field network interface is referred to as link fault and the outage of a complete gateway is called device fault. It is further assumed that a device fault at a VRG member is fail-silent. The fault hypothesis for such a VRG states that it is capable of tolerating one link fault or one device fault.

1) ***Link fault at master gateway****:* Assume a link fault at an arbitrary field device interface of the current *master* gateway. The *slave* gateways detect the failure by means of their *bus guards*. Each *slave* device starts the *GracePeriod* due to unacknowledged *heart_beat* messages. After the *GracePeriod* has expired, the *slave* gateways perform a *self-test* using the respective VRN as a reference to avoid erroneous triggering of the election process. Further, the *slave* devices start to multicast the *master_advertisement* messages simultaneously on all available interfaces. The *master gateway* also receives the *master_advertisement* messages through its remaining operational network interface. The current *master* gateway shuts down operation and remains fail-silent. After the *MasterAdvertisementTimeout* has expired, the *slave* gateway with the highest priority is selected among all *gateway* devices that participated in the master advertisement. Thus, the highest priority *gateway* assumes the role of the new *master* device and continues regular operation. All remaining *slave* gateways, part of the VRG, start to monitor the new *master*.

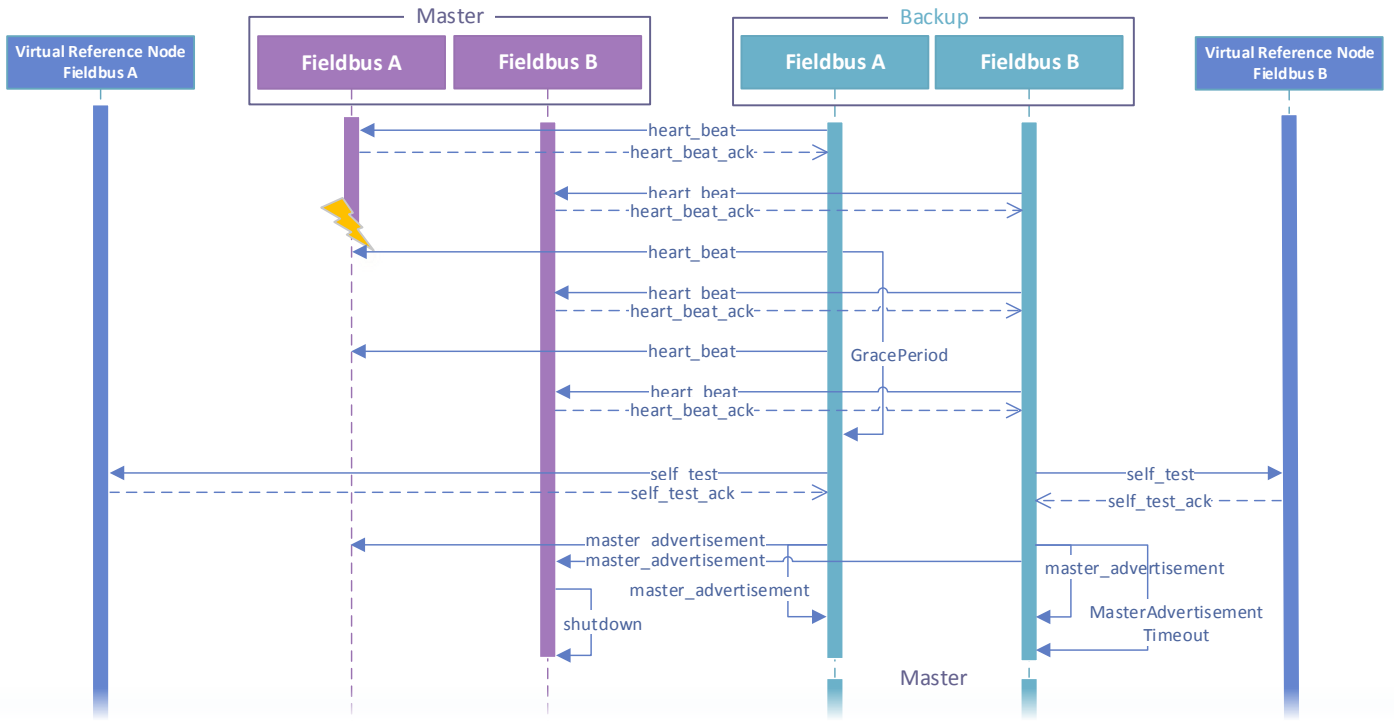

Fig. 5. Link fault at *master gateway*

2) ***Device fault at master gateway****:* Assume a device fault at the current *master* gateway. The *slave* gateways again recognize the outage of the *master* via their *bus guard* components. After the *GracePeriod* has expired, the *slaves* start a *self-test* of their fieldbus interfaces by contacting the VRNs. After the *self-test* is successfully completed, the *slave* devices start to send *master_advertisement_messages* on all available field network interfaces, utilizing reliable multicasts. After the *MasterAdvertisementTimeout* has expired, all *slave* gateways choose a new *master* by comparing the received priority values from the advertisement phase. The device with the highest priority value is selected, and immediately continues normal operation. All remaining *slaves*, part of the VRG, start to monitor the new *master* device.

3) ***Link fault at slave gateway****:* Suppose a link fault at an arbitrary *slave* gateway and an arbitrary field device interface thereof. The faulted device is then unable to send *heart_beat* messages. Hence, the *bus guard* of the failed interface starts the *GracePeriod*. After the *GracePeriod* has expired, the *slave* performs a *self-test* by contacting the VRN. Since no messages can be sent via the faulted network interface, the *self-test* fails. Hence, the affected device shuts down operation and removes itself from the VRG.

4) ***Device fault at slave gateway****:* Assume a device fault at an arbitrary *slave* gateway of a VRG. Such a failed *gateway* ceases to communicate with any other member of a VRG. Synchronization traffic from the *master* device remains unacknowledged. Hence, the failed *gateway* is removed from the VRG.

5) ***Link fault at master gateway with pending transaction****:* Suppose a link fault at the *master* gateway, while the master is either busy with a synchronization or field network triggered transaction. In such a case, a distinction must be made, whether the faulty interface is involved in a pending transaction or not.

At first, assume that none of the ongoing transactions is linked to the faulted field network interface. The transaction is completed, despite any incoming *master_advertisement_messages*, but no new transactions are started despite synchronization transactions. Therefore, the *master* gateway finishes any pending synchronization tasks by means of the correct functioning network interfaces. Finally, the faulted *master* shuts down.

Now consider, that the faulted interface interferes with a pending gateway transaction. In case of a transaction originating from a field network, it can be recovered by means of the retransmission capabilities of the network itself as soon as the new *master* device is operational. On the other hand, a synchronization transaction will be simply retried on another fieldbus interface. Transactions which involve field devices will be marked as incomplete and retried from the new *master*. Again, the old *master* shuts down after all synchronization tasks are finished and the new elected *master* resumes operation.

6) ***Device fault at master gateway with pending transaction****:* Assume a device fault at a *master* gateway with an ongoing transaction. In either case, the transaction is aborted and inadvertently lost. Therefore, the application running on top of the field network must be able to handle the inconsistent state. However, a new *master* device is elected according to the previous cases and operation is resumed.

7) ***Link fault at slave gateway with pending transaction****:* Assume a link fault at an arbitrary *slave* gateway while

a synchronization transaction is pending. Regardless if the synchronization transaction is affected by the faulted network interface or not, the *slave* device will sense the failure through the *bus guard*. Hence, the faulted device will remove itself from the VRG after the *self-test* failed.

8) ***Device fault at slave gateway with pending transaction****:* Assume a device fault at an arbitrary *gateway device* part of the group of *slaves* of a VRG. All synchronization attempts from the *master gateway* will then fail, meaning that the *backup* device is removed from the logical *gateway* compound.

VI. Proof-of-Concept

A proof-of-concept gateway implementation has been carried out utilizing *beaglebone black* single board computers. The testbed interconnects *KNX* and *ZigBee* by a VRG consisting of two *gateway* devices. The choice of the communication was driven by the fact that two different physical media and network topologies are utilized. *ZigBee* communicates via radio and resorts to a *mesh* network topology. The *KNX* installation is built upon wired connections and employs a traditional bus topology. Interconnection with the gateway has been established with the use of USB dongles designated to the individual communication protocols.

The internal software architecture of the proof-of-concept gateway devices was strictly designed according to Figure 3. However, the *transport*, *network*, *data link* and *physical* layer are packed into *communication stack* building blocks. The following enumeration gives a brief overview of each component utilized by the proof-of-concept gateways:

- **KNX communication stack:** *knxd*[1] acts as a communication stack connecting a *KNX* network to the *protocol binding* layer and further to the *information model*. The communication stack provides a complete *API*, including message monitoring, filtering and forwarding. The physical connection is done via a *USB* attached bus coupler device that forwards incoming messages via a *tty* software interface.
- **ZigBee communication stack:** The interface to the *ZigBee* network is achieved via the *Texas Instruments Z-Stack Linux Gateway*. This *gateway* driver solution was intended as a demonstration project for a *gateway* between Ethernet and IP based applications and *ZigBee/Home Automation (HA)*. However, the original implementation has been adapted by the proof-of-concept and was integrated via local *unix* sockets into the *gateway* software architecture.
- **Information Model:** The *information model* is implemented with the help of a *mysql* relational database. A simple database schema directly tailored to the test scenarios was created and used for the storage of the different entities of the *information model*.
- **Protocol Binding:** The *protocol binding* layer utilizes the stored entities at the *information model* via atomic database transactions. Thus, the *mysql* database already provides the necessary synchronization to avoid concurrent access to stored information. The address field of each message retrieved from the fieldbus interface of the proof-of-concept *gateway* device is matched against the list of *FieldDevice* entities referenced by the corresponding *View*. In case a match has been found, the message is processed and the *information model* is updated accordingly.
- **Bus Guard:** The *bus guard* transmits periodic messages to the field network address, which are automatically acknowledged by the communication stacks of the target *gateway* devices. The configuration parameters of the proof-of-concept gateway device are hard-coded. The *heart-beat* interval is set to one second, whereas the *grace period* is set to three consecutive failed transmission attempts.
- **Bus Load Monitor:** The instances of the *bus load monitor* building block attach themselves to the field protocol stacks as passive listeners in promiscuous mode. Thus, simply all incoming and outgoing messages are counted and a load indicator is built. Such an indicator is calculated by means of a *moving average* algorithm. More sophisticated factors regarding the load indicator, like the signal strength or link quality index for *ZigBee*, are currently not incorporated into the *bus load monitor*.
- **Redundancy Manager:** Synchronization between individual *gateways* is achieved by subscribing to change notifications from the *information model*. Furthermore, any message directly addressed to the *gateway* device is interpreted as *gateway-to-gateway* communication. Hence, such telegrams are always directed to the *redundancy manager*, while messages with differing addresses are matched against the list of *FieldDevice* entities.

Finally, all outlined software components are interconnected via *Unix* sockets and the message queue library *ZeroMQ*[2]. *ZeroMQ* provides a high performance message passing system with *publish/subscriber* mechanisms and is able to serialize arbitrary data structures. There is also support for various programming languages. All custom software core components of the proof-of-concept gateway device have been written in C, with some minor bindings written in Python.

Different application scenarios from the domain of BAS with varying complexity have been tested, like lighting (see Figure 6) and temperature control. Also the fault analysis cases outlined in Section V have been examined. Device faults have been simulated by pulling the power plugs on the gateway devices, while link faults have been provoked by disconnecting the USB dongles.

VII. Conclusion and Outlook

The paper presents a horizontal integration approach between different field level protocols, while retaining the level of reliability of the field network with respect to the weakest fault hypothesis involved. The VRG further leaves the complexity needed for the integration of different protocols at the field level of the automation pyramid. The translation process

[1] knxd – *https://github.com/knxd/knxd*

[2] ZeroMQ – *http://zeromq.org/*

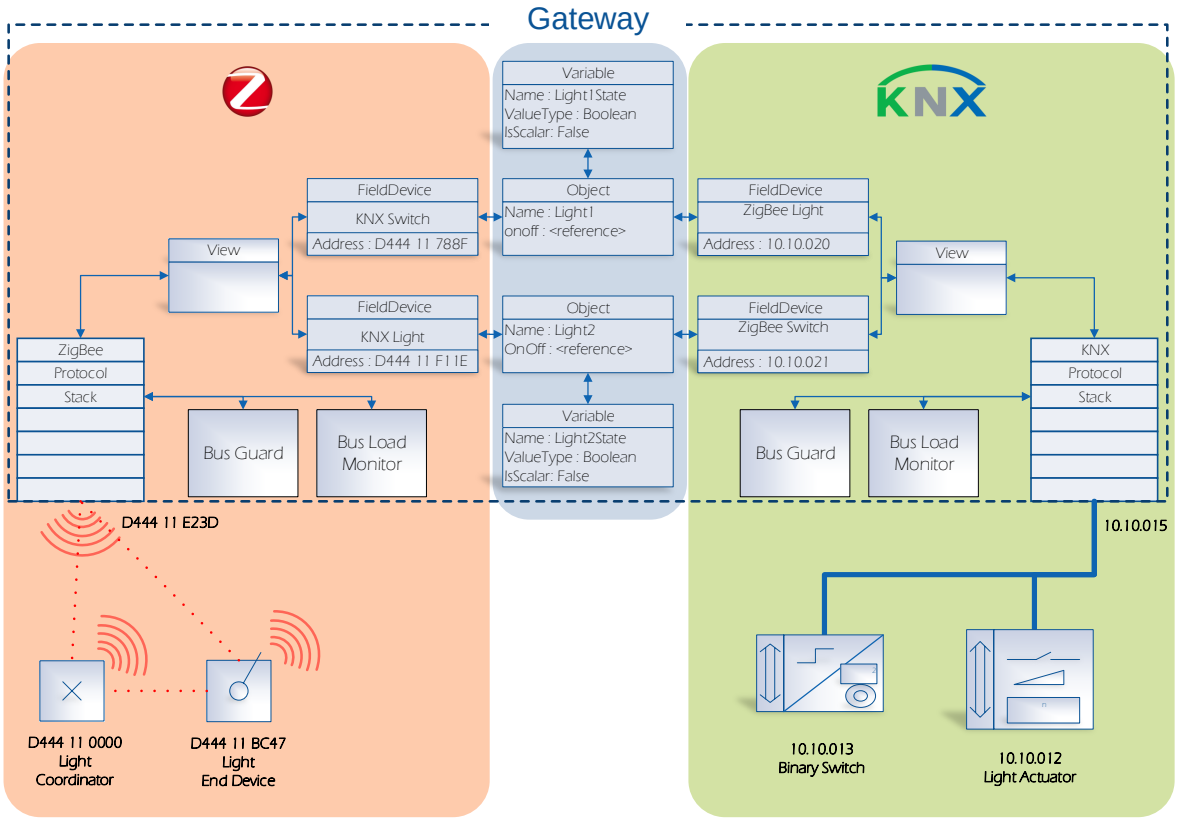

Fig. 6. Lighting test scenario

between different field level protocols has been designed using a stateful approach, resting upon a common *information model*. A stateful approach allows aggregation and dissipation of telegrams, and a more flexible handling of data in general. In contrary, a stateless translation process enables direct end-to-end delivery of data, and does not need a replication mechanism, but can only be applied to a restricted number of protocols [9]. Hence, a stateful approach is more flexible and allows the interconnection of a wider range of field protocols.

The fault hypothesis of an interconnected system is given by the fault hypothesis of its weakest component. The proposed approach avoids being the weakest component by offering a replication mechanism that can be scaled to the fault hypothesis of the interconnected field level protocols. Hence, a VRG is able to keep the integrity of the field network with the weakest fault hypothesis. Nevertheless, the scalability of the replication mechanism is bound by the communication bandwidth of the available field communication links. Furthermore, an analysis of the impact of replication traffic on normal fieldbus operation still has to be done.

The ability to achieve seamless integration of different field level protocols of the BAS domain has been demonstrated by the proof-of-concept implementation. However, the important topic of deployment and configuration of the VRG remains undiscussed. The described gateway approach lacks of a proper management service for configuration and dynamic assembly of a VRG so far. Hence, the next steps will be to include a *gateway* management service leveraging services like *joining*, *leaving* and initial *creation* of a VRG.

References

[1] W. Kastner, S. Soucek, C. Reinisch, and A. Klapproth, "State of the Art in Smart Homes and Buildings," in *Industrial Communication Technology Handbook, Second Edition*, 2014.

[2] C. Reinisch, W. Granzer, F. Praus, and W. Kastner, "Integration of heterogeneous building automation systems using ontologies," in *Proc. of the 34th Annual Conference of IEEE Industrial Electronics*, 2008.

[3] S. Soucek and D. Loy, "Vertical Integration in Building Automation Systems," in *Proc. of the 5th IEEE International Conference on Industrial Informatics*, 2007.

[4] H. Jarvinen, A. Litvinov, and P. Vuorimaa, "Integration platform for home and building automation systems," 2011.

[5] T. Sauter, "Fieldbus Systems: Embedded Networks for Automation," in *Embedded Systems Handbook, Second Edition: Networked Embedded Systems*, 2009.

[6] J. Jasperneite, "INTERBUS," in *Industrial Communication Technology Handbook, Second Edition*, 2014.

[7] Y. Zhang and Y. Luo, "An architecture and implement model for model-view-presenter pattern," in *Proc. of the 3rd IEEE International Conference on Computer Science and Information Technology*, 2010.

[8] "IEC 62541 - OPC Unified Architecture Specification," 2010.

[9] L. Krammer, S. Seifried, and W. Kastner, "A fault-tolerant Backbone for IEEE 802.15.4 based Networks," in *Proc. of the IEEE International Conference on Industrial Technology*, 2014.