Middleware communication platform for securing data-flow between different automation networks delivered by IOT principles

The building automation systems (BAS) are design to answer comfort, security and energy saving requirements. Energy can represent a significant percentage of the total building expenses. Among energy, an integrated BAS will significantly contribute to decrease of maintenance costs as well. Today, the most popular and well-known standards in BAS are, LonWorks and BACnet, KNX, Modbus which we consider as open solution standards. [1] LonWorks and EIB/KNX are considered as a field-level and control centric solutions (bottom up), while BACnet is considered as more service oriented (top down) protocol positioned more for upper-level functionality. The goal of today building automation solutions is to achieve seamless all building sub-systems (e.g., heating, ventilation and air-conditioning (HVAC), lightning, security, etc.) through both horizontal (data points interactions) as well vertical (management service interactions) domains with extension towards IOT services.

For such a system key is how to establish multi-vendor and multi-protocol interoperability through all sub-systems. In such a @OS system, it is possible to share sensor data among different subsystem like lightning and HVAC together with outbound IOT information’s derived from public services. Open solutions designed on such a way offers numerous benefits, including competitive bidding, consistent installation, consistent maintenance, system integration and interoperability, data acquisition and product interchangeability, as well future proof accessibility of third party services.

Generally, it is impossible to close the building into just one of the protocols; therefore, we still need an integration of several different protocols, to handle the range of building automation needs. It is very important to understand that there are lots of deployed buildings where an automation of the different sub-systems is done by different protocols. Intercommunication between such a subsystems is done through gateway devices. Every protocol defines his own application model with the data representation description (data format, encoding) as well the communication model between applications (data manipulation methods). Since, all application models are specific, it is impossible to establish a direct (i.e., translation/gateway-free) communication across protocol borders, and use of a gateway is a must in such integrations. [3] Gateways are generally containing a database of mappings between network entities (data points) of both sides. The data points correspond to the logical inputs and outputs of the underlying physical process.

The main characteristic of a data point is its present value. Typically, it corresponds to the real world physical representations like room temperature or the state of a switch. The data points from different devices at the same network could be directly connected, distinguishing output points and input points. Often to the data points are attached attributes (Meta data), such attributes are adding a semantic meaning to the present value by describing, for example, the engineering unit of the value. Typically, data points are logically grouped to describe specific functions of the system, including both data points and the processing rules that belong to them. Such logical groups we are calling „functional blocks”[2].

Communication between devices at the same network (horizontal communication) is established through data points. Since, the horizontal communication is involved in the exchange of present values (or alarm indicators), it is possible that the semantics interpretation could be assured at setup time by all communication partners.

By the contrary vertical communication is more stochastic, and it is represented by service related tasks such as accessing and modifying data from the application outside the networked devices. Good examples are: adjusting a set point, retrieving trend logs. Such tasks are often described as management services. Other tasks which are guided by the same principles are, for example, modifying the application itself by changing binding information. Usually such tasks are called an engineering service. Vertical communications most often follow a client-server model, unlike horizontal communications that are following a producer–consumer model.

The gateway functions needed for internetwork communication are limited to a small set of services, such as read value, write value and change-of-value subscription. Gateways can directly translate information between two control networks, providing horizontal connections from data points of one system to data points in another system. To successfully integrate distributed applications spread over heterogeneous networks. We need to translate data points as well application services from one network to another. Every network has an own application model equally important communication services and data structures and gateways are not able to seamlessly translate all specifics. Therefore, we have to create a generic application model which will be able to suit all relevant building automation networks. Such structure will move the mapping from gateway type data point mapping to full application model mapping where both data points as well application services will be translated. The biggest obstacle to such structure is non existence of well defined general building automation
ontology. The BrightCore @OS framework is developed to help in solving of such issues as well to enable seamless service interaction of multi protocol building systems with outbound IOT services.

BrightCore @OS framework includes the tools for an easy integration, maintenance, supervision and development of new automation applications as well object (Prototypes of user-defined constructs, which govern how variable data is processed), and plug-in for horizontal and vertical applications.

The System opens space to ICT and IOT professionals by Shell API, who are not necessarily experts in the automation, to build connections to Enterprise class of applications, as well to build special corporate and IOT service applications like the energy management dashboards, analytics applications, or real time simulations engines. The novelty of the solution is in a complete “syntactic-semantic notion of network infrastructure,” making it expandable to different services bounded directly to data points of the underlying networks.

BrightCore @OS is built precisely with an idea to reduce the number of people engaged in passive maintenance and to help Outsourcing services in building automation control infrastructure and maintain corporate standards.

BrightCore @OS is designed with a service infrastructure in mind. Therefore, combining the technical systems in buildings with IOT services that provided expert’s knowledge is one of its natural tasks.

BrightCore @OS is a distributed software framework with a primary goal to successfully mediate between building automation control networks and their desired uses and service providers.

Control networks are, in essence, structured collections of information and BrightCore @OS acts as a two-way information conduit that connects control networks with microservice server programs called CoreLets and client programs called BrightLets. The actual function and purpose of BrightLets are not predefined. BrightCore enables complete customization of CoreLets and BrightLets through the Shell, an API connecting a BrightLets or CoreLets to the rest of BrightCore. The information held in the control networks is structured in the object containers. This information is processed into values through “objects”, user-defined constructs, which govern how variable data is processed. The mapping of variables to objects, objects to objects, object definitions and everything vital to the functioning of BrightCore is configured in a single program called BrightCore Builder. The BrightCore Builder includes a vector graphics editor for creating interactive graphic representations of variable data. The graphics are handled in BrightLets by the Shell, which has a built-in vector graphic engine. The entire system has a well-defined encapsulated core, and it is completely extendable to be sufficiently versatile for today’s continuously changing market demands. The problem of polytypic networks is solved with the “adapters.” Each type of network has an adapter, a server who monitors wanted network traffic and converts it to the common encapsulated object model which preserves specifics of native network both data points and services. This selected traffic is then sent utilizing event driven mechanism to the microCore. The microCore (could live in Windows or Linux environment) is a server, which uses the system’s configuration to convert the received objects’ data to the general defined object model. Every object has an object class that defines the object’s inputs and how the inputs are processed. The object classes are completely configurable. The BrightCore Builder is a program used for creating and editing the main BrightCore configuration as well for creation of new classes and for the binding of data between dissimilar networks. The variable-to-object property and object-to-object mapping is done through a straightforward drag and drop. The program has an interactive error, and warnings list to help keep track of empty or invalid mappings.

Multiple BrightNode adapters can run simultaneously allowing of interconnection of many systems over IP. We can assume every building as one node of infrastructure. Furthermore, the built-in BrightNode routing service allows data to be transferred from one protocol to another, either on the same altitude system or between altitude systems, across the IP network. Clearly, the system is constructed with an idea to manage an arbitrary number of control networks of multiple arbitrary types, as well as a system that could work as a vertical service from the cloud.

Client and Server programs could be connected to the microCore infrastructure and get the information on past object values as the microCore stores all object changes in an internal database. They can also be notified of object value changes as they happen. Client and server connectivity to the microCore is handled by the BrightCore client/server API - the Shell. The Shell is simple to use, and it enables third party vendors to create new BrightCore clients (called BrightLets and CoreLets) tailored to custom specifications based on MicroService paradigm. The Shell also supports client network actuation - setting the network variable values. The Shell solves the problem of changing user demands – client and server programs can be created and modified as users wish, without compromising system structure and function.