Ontology-Based Fault Propagation in Building Automation Systems

Henrik Dibowski, Ondřej Holub, Jiří Rojíček

Home and Building Technologies (HBT)
Honeywell
Prague, Czech Republic
henrik.dibowski@honeywell.com, ondrej.holub@honeywell.com, jiri.rojicek@honeywell.com

Abstract - Faults in building automation systems (BAS) can have far reaching consequences, because they can affect other equipment and areas in a building and are hard to isolate. To better deal with that, an ontology-based approach is presented that can reason about the effects of faults and causes of abnormal situations. A rich ontology-based building information model (BIM) is used for modelling BAS formally and semantically. Faults are propagated in this model in both forward and backward direction by applying rules, which are defined with either SWRL or SPARQL/Update. By that, the fault propagation enables an immediate assessment of potential consequences of faults respectively an analysis of root cause(s). It can enhance fault detection and diagnosis by considering BAS as a whole instead of single pieces of equipment only, which provides a better understanding of BAS and improves the decision making and prioritization of the right emergency and maintenance actions. An earlier, shorter version of this paper was presented at SIMS2016 [2].

Keywords - fault propagation and diagnosis; fault detection; building automation system; building information model; ontology; OWL; rules

I. INTRODUCTION

Building automation systems (BAS) can be complex, highly networked, distributed systems that supply and control up to hundreds of rooms and zones in a building. They can consist of different central plants (e.g. boiler plants, chiller plants, air handling units (AHUs), roof top units (RTUs), variable air volume (VAV) systems) that each may contain many different pieces of equipment themselves (e.g. pumps, fans, dampers, heating/cooling coils, air filters, sensors, controllers) and up to thousands of distributed, decentralized field devices (e.g. sensors, operating panels, controllers, dampers, valves). In BAS manifold faults can happen, since basically each of the various pieces of installed equipment or implemented control strategies can fail or function improperly due to misconfigurations, degradations, breakdowns etc. The immediate detection and repair of faults however is critical to the overall operation of the building in order to ensure a comfortable environment for occupants, to avoid inefficient operation of BAS which may result in a higher energy consumption, and to prevent further damage to occupants, equipment or goods (e.g. food in a supermarket).

Computer-aided fault detection and diagnostics (FDD), which can determine and analyze faults in BAS, and solutions for optimized building operation have become prior art. Several solutions are offered by various vendors, e.g. ATTUNE® (Honeywell), PANOPTIX (Johnson Controls Technology Company), EZENICS (Ezenics, Inc.), SCIENERGY (Scienergy, Inc.), SKYSPARK (SkyFoundry, LLC), ANALYTIKA (Cimetrics Inc.) and CLOCKWORKS (KGS Buildings, LLC). FDD algorithms are typically specialized in faults of specific plants or types of equipment, or areas of a building. For example, FDD algorithms for boiler plants typically consider the variables associated with the boiler plant, such as power input to the boiler, supply water temperature, return water temperature, supply water mass flow etc. By analysing historical and/or current data of associated variables, faults like breakdowns, malfunctions, degradations and control failures in the specific plant or equipment can be detected. Such an equipment-centric FDD approach is intuitive and reasonable, because algorithms need to be aware of just small pieces of a BAS instead of facing the whole complexity. This simplifies algorithm development, configuration and reuse for many BAS. An isolated analysis like this however has the serious deficit that it can only be aware of what is going on inside a specific plant, equipment, or area, but is unaware of the complex, potentially far-reaching consequences of faults within BAS. Along the various interrelationships that exist in BAS, based on e.g. energy flows, mass flows, signal flows or control flows, faults can propagate from the root cause to other equipment, areas or control strategies in a building. Such propagations of faults rely on the phenomenon of causality: a cause-effect-relationship between one process, the cause, and another process, the effect [1]. A fault in a central plant is especially severe since it can propagate to the downstream devices in the whole building through the connecting pipes or air ducts. A broken boiler, for example, cannot supply hot water to the different connected AHUs, VAVs, radiators etc., which in turn cannot heat the supply air and the air in the rooms and zones anymore. On the contrary, the effects of other faults can be very local, such as a stuck radiator valve, affecting only the zone the radiator is located in.

Understanding the interrelationships and consequences of faults and how they affect the proper operation of the
building (e.g., availability of rooms, user comfort, energy efficiency) is crucial for properly reacting to them, for supporting the decision making about the proper emergency and maintenance actions and schedules (e.g., repair, configuration, replacement), and finally for improving the overall operation and maintenance of BAS. But so far, determining the interrelationships and consequences has been a complicated, time-consuming, and error prone engineering task, that requires a specific treatment for each individual BAS. It requires engineers to read and know the system documentation, to have an overall understanding of the BAS layout and its interdependencies in mind and to analyse the possible effects by thinking. Thus, no automated analysis and immediate and reliable reaction is possible. Hence, a computer-aided support is urgently needed here.

To solve this deficit, a knowledge-based approach that enables fault propagation in BAS is proposed in this paper. It uses an ontology-based building information model (BIM) as system model, and applies fault propagation rules to automatically propagate faults along causality relationships. The fault propagation can work in both direction, i.e. forward to derive the effects of a known fault in the rest of the BAS (forward fault propagation), and backward to identify the root cause(s) of observed abnormal situations (backward fault propagation).

This paper is an extended version of the conference paper [2]. Amongst several enhancements and three additional figures, the main improvement consists in the new section V, which describes an alternative, more expressive approach of how the fault propagation rules can be specified and applied.

This paper is structured as follows: In the next section, the prior art of fault propagation in engineering is summarized and analysed. Section III focuses on an ontology-based BIM representation of BAS and explains how to draw inferences with rules. Causality rules and fault propagation rules are introduced in section IV. In section V, which is the main enhancement compared to [2], an alternative implementation of the rules as graph-patterns, expressed with the update language SPARQL/Update, is explained. The principles of forward and backward fault propagation are then described in sections VI and VII. Section VIII describes how the system has been implemented, and section IX finally summarizes the solution and concludes the paper.

II. FAULT PROPAGATION IN ENGINEERING

A. State of the Art

According to [3], fault propagation “is an established technique which uses qualitative relationships to determine how a disturbance at some point in a network of relationships will affect the rest of the nodes in the network.” Fault propagation has widely been addressed in different engineering disciplines, and many literature and implementations can be found. It is clearly dominated by safety critical applications such as in process automation, digital circuits, communication networks or aerospace, where safety analyses and diagnoses are indispensable and the required engineering effort is affordable.

1. Identification of Causality: The first required step for doing fault propagation is to recognize the causalties (i.e. qualitative relationships) between the entities of interest, which are typically variables of a system or the equipment/components. Mainly two different sources of information can be used for that step, which are process data and process knowledge [4].

Prevalent are data-driven methods that use process data for the identification of causalties, i.e. historical and/or current data available from the system. Various methods have been described and applied, like cross-correlation analysis, Granger causality, frequency domain methods, predictability improvement and transfer entropy [5]. A good overview can be found in [4].

Model-based approaches on the other hand use process knowledge that is made available by some kind of system representation. Depending on the domain these can be different kinds of models, like piping and instrumentation diagrams (P&ID), control schematics, network diagrams, circuit diagrams etc. Despite the significant benefit process knowledge can provide, model-driven approaches are less widespread than data-driven methods. The main reason is very likely the lack of machine-interpretable system models and the prevalence of traditional documents containing natural language or graphics not processable by computers, and the need to have engineers doing the extraction of relevant relationships and the identification of causalties manually without or with limited tool-support. Of course, data-driven and model-based approaches can also be used in combination to identify causalties in a system.

2. Causal Models: In order to represent identified system causalties and to enable fault propagation, an appropriate causal model of the system is required and needs to be created. Popular and intuitive are causal graph models, in which nodes represent variables or equipment/components of the system and arrows represent qualitative causal relationships. Several approaches can be found, like signed directed graphs (SDGs) [3], [4], [6], logical causal graphs [7], fault trees [8] and fault semantic networks [9]. Various extensions have been pro-posed, by which the initially pure qualitative characteristics can be combined with quantitative properties, such as probabilities assigned to the arrows of SDGs [10] or logical causal graphs [7], fuzzy states of nodes [11], or fuzzy propagation rules [9]. In [12] an extension of SDGs by propagation times has been proposed, also enabling the computation of the time it takes for a fault to become detectable. A similar approach is that of temporal causal graphs described in [13].

For the manifold causal graph models that exist there is many different methods available that can be applied for

DOI 10.5013/IJSSST.a.18.03.01 1.2 ISSN: 1473-804x online, 1473-8031 print
implementing fault propagation on these models. Applied methods range from various graph algorithms (e.g. depth-first search, breadth-first search, heuristic search) over mathematical or probabilistic computations of different complexity up to ant colony optimization as used in [14]. A few approaches address the task of (semi-)automatically creating the graph models, thus minimizing the required engineering effort. [15] explains the extraction of process topology data from CAEX (Computer Aided Engineering Exchange) files and conversion into topology-based causal models. In [16] a method is presented by which fault trees can be derived from AADL (Architecture Analysis and Design Language) specifications. And [13] shows how temporal causal graphs can be derived from bond graphs.

Another way of representing causal models besides causal graph models are matrix representations. Matrices are used for example in [17] to define causalities and connectivity of process automation systems. They can be processed by search algorithms to exploit feasible propagation paths of faults and display them in process monitoring systems.

Symbolic approaches constitute another big fault propagation category. Instead of graphs or matrices they use a symbolic knowledge representation for modelling the systems and causalities, i.e. symbolic logic as a declarative, well understood knowledge representation and associated reasoning paradigms. First applications can be found in the 80s, such as [18] describing diagnostic expert systems combining probabilistic inference with a symbolic representation of causal relations. This approach is specialized on direct causalities between disorders and effects, but cannot execute fault propagation yet. In [19] a rule-based approach to fault diagnosis was introduced. Diagnostic rules (IF-THEN rules), derived from a SDG, are used for reasoning about fault diagnoses, incorporating also the pathways of fault propagation. The approach developed in [20] is based on a Datalog representation of a system (binary decision diagrams and rules) and applies a Datalog reasoning engine to find out whether faults in components can cause a critical component of the system to fail. An interactive system involving feedback from human experts is explained in [21]. By some kind of ontology fault processes and faults are classified, albeit no formal ontology language is used. A rule-chaining-type reasoning, implemented in LISP, can generate causal chains of events which occur in the system. Another approach mentioning the application of ontologies, but only for modelling the system itself, was presented in [10]. In contrast to [19], SDGs are constructed from the ontological representation and used for fault propagation.

B. Criticism

Despite fault propagation has been studied for many years and widely been addressed in literature, prior art has several limitations and gaps. First of all, to the best of our knowledge, there is no approach that addresses fault propagation in BAS and their specific characteristics adequately. Secondly, the causal models (e.g. graph based models, matrix representation, symbolic approaches) that need to be created in order to realize fault propagation are yet another (partly redundant) system model. Their creation from process data and/or process knowledge can be a laborious, expensive and error-prone task requiring the work of qualified engineers. Only few approaches discuss solutions how these models can be derived (semi-)automatically. Thirdly, the causal models are in most cases limited in their expressiveness, which seems to be a compromise to keep their creation and processing simple. Faults are typically represented as binary states (true means fault and false no fault) or by a probability value; and causalities between them simply indicate that they can have some kind of impact on each other, eventually differentiating between a positive and negative influence or associating a weight or probability. A detailed differentiation of the various types and reasons of faults that can manifest in equipment/components or variables (e.g. ok/impaired/broken state of equipment or controller, insufficient/sufficient dimensioning or energy supply, too low/too high value) or the types of causalities (e.g. energy flows, mass flows, signal flows or control flows) however is mostly missing or not well addressed. This semantics of a system, its nodes, fault types and causalities is not explicitly available in the causal models and cannot be utilized for fault propagation.

Our approach tries to overcome those issues and limitations. We propose a qualitative, symbolic, knowledge-based fault propagation approach for BAS. It can operate on the system model, i.e. a BIM formalized as ontology, directly and automatically. This is essential since it is simply not affordable in BAS domain to spend extra time and money for creating causal models for every single building. Unlike in other safety critical applications, the budget is very limited here. Our approach exploits the rich semantics of the model and faults to better differentiate between different types of faults and their specific effects on the system, and can thereby derive detailed diagnoses and fault propagation explanations.

III. KNOWLEDGE-BASED BUILDING INFORMATION MODEL

The BAS domain and with it our fault propagation approach proposed in this paper can much benefit by the existence of a mature, machine-processable system modelling approach called building information model (BIM). “A BIM is a digital representation of the physical and the functional characteristics of a facility. As such it serves as a shared knowledge resource for information about a facility, forming a reliable basis for decisions during its life cycle from inception onward” [22]. A BIM can cover up to five dimensions, i.e. a 5-D BIM is a model that includes 3-D shape information, schedule and time constraints as fourth dimension and a cost dimension [22]. Building information modelling is often associated with
Industry Foundation Classes (IFC) [23]. It is a neutral, non-proprietary, open data exchange specification for sharing BIM data among software applications digitally and facilitating interoperability between them. IFC models can include many different subsections such as architectural, mechanical, electrical, plumbing, sanitary and fire protection information.

Especially the following aspects that can be found in a well-defined BIM are of interest for the fault propagation approach:

- Building structure, consisting of floors and contained zones and rooms, and their properties
- Installed plants and equipment and their type and location
- The physical and logical causal interrelationships between plants, equipment, zones and rooms based on building structure (energy flow i.e. heat loss/input through windows and walls from/to neighboured rooms/zones and outside), air duct layout (air mass flow), pipes (water mass flow), cables and logical bindings (signal flow, control flow).

A. Ontology-Based BIM

According to our experience, the best technical solution for specifying BIMs is to use a formal ontology language, such as OWL (Web Ontology Language) [24]. OWL is an expressive logical language with formal syntax and semantics based on description logics theory, and is perfectly suitable for domain modelling, classification, semantic search and deriving new insights by reasoning. The automatic conversion of BIMs from IFC to OWL has been addressed already, e.g. as a conversion of the full model [25] or pieces of interest in order to keep the model simple [26]. For the implementation of the fault propagation approach it is reasonable to use an ontology vocabulary specialized for BAS, such as BASont [27]. Additionally, we suggest to extend the scope of the BIM to include also the BAS variables (i.e. datapoints) and their semantics (i.e. what type of information they represent, such as supply water temperature of a boiler, supply air mass flow of an AHU etc.). Their semantics can be modelled ontologically as described in [28].

B. Combining BIM with Rules

BIMs formalized with OWL allow it to do reasoning and derive logical inferences. Using SWRL (Semantic Web Rule Language) [29] logical IF-THEN rules can be specified and executed by a reasoner, e.g. Pellet, Jess or KAON 2. We use this rule-based approach to show how to implement fault propagation in BAS, as will be shown later on in the following Chapters IV, VI and VII.

Combining an ontology, which de facto is a knowledge base, with rules executed by an inference engine per definition makes our system a knowledge-based system (KBS), and the fault propagation a knowledge-based approach.

Alternatively to SWRL, SPARQL/Update [30] can be used for specifying rules. SPARQL/Update is an update language for RDF graphs, which is the syntactical data model underlying OWL. Supporting negation, complex filtering and many more makes it even more expressive than SWRL. SPARQL/Update queries can be executed by a query engine, such as ARQ from Apache Jena API. It is shown in Chapter V in detail, how SPARQL/Update can be used for specifying and applying rules for fault propagation.

IV. CAUSALITY AND FAULT PROPAGATION RULES

BAS are mostly unique systems, specialized towards the specific layout and characteristics of a building. Nevertheless, there are well known patterns used in BAS design (e.g. types of plants/equipment and their composition, ductwork layout, pipework layout etc.) and principles that rule in BAS. Compared to other more heterogeneous systems, such as process automation, we are in an advantageous position to formalize these circumstances as expert knowledge in terms of rules. The fault propagation approach can be realized by defining SWRL rules and applying them to an ontology-based BIM. We distinguish two categories of fault rules, which are executed in succession: 1) causality rules, i.e. rules for determining and materializing causalities (1st step); 2) fault propagation rules, i.e. rules for propagating faults in BAS (2nd step).

The rules can be best explained with an example. In Figure 1 a part of a BAS is shown, consisting of a boiler plant, AHU and a zone.
The boiler plant provides hot supply water (supply water temperature $T_{sw}$), which is circulated by a pump (supply water mass flow $w_{sw}$) to the AHU and radiator and back to the boiler. The AHU supplies air to the zone. It can control the supply air temperature ($T_{sa}$) by heating the air with a heating coil. The supply water mass flow inside the heating coil ($w_{sw\_ahu}$) can be controlled by a valve, resulting in a specific heat gain ($Q_{hc}$). This can heat the supply air, which is moved by a fan (supply air mass flow $w_{sa}$). Another valve at the radiator controls the radiator supply water mass flow ($w_{sw\_rad}$) and the resulting radiator heat gain ($Q_{rad}$). AHU and radiator can thus heat the zone air temperature ($T_{za}$).

Besides the water and air mass flow, Figure 1 already shows some of the causality relationships between the equipment and variables that exist in this simple example. They are visualized as directed arrows. A causality relationship between an equipment and a variable, typically a controlled variable, expresses that faults of the equipment can affect the variable. And causalities between variables indicate that faulty states of variables can propagate to other variables. A zone air temperature $T_{za}$, for example, can be affected by several other variables, such as radiator heat gain, AHU supply air temperature and mass flow, and, not displayed in Figure 1, air temperature of neighbouring zones and outside air (heat loss/input through windows and walls).

A. Causality Rules

Causality rules can be defined and applied to automatically determine the causality relationships between variables of a BAS. They are based on expert knowledge about the typical variables associated with specific equipment and the interrelationships that exist in BAS based on e.g. energy flows, mass flows, signal flows or control flows.

Some causality rule examples specified in SWRL are shown in Figure 3. Please note that the class and property prefixes were omitted in these examples for obtaining a better readability. In order to better understand the rules, an excerpt of the syntax of SWRL used in our examples is given in Figure 2.

From this Figure 2 follows how one has to read the rules. Rule 1 from Figure 3 for example defines the following: “For individuals of class ‘Boiler’ that have an object property relationship ‘hasVariable’ to an individual (?var1) of class ‘SupplyWaterTemp’ and an object property relationship ‘supplies’ to some individual having an object property relationship ‘hasVariable’ to some other individual (?var2) of class ‘HeatGain’ INFER the object property relationship ‘hasCausalRelationship’ between the individuals matching variables ?var1 and ?var2.”
Described in a more intuitive way, the causality rules from Figure 3 have the following semantics:

- Rule 1: The supply water temperature of a boiler, which supplies a downstream equipment (e.g. AHU, radiator), has a causal relationship to the heat gain of the downstream equipment (e.g. heat gain of AHU heating coil or radiator).
- Rule 2: The heat gain of an AHU heating coil has a causal relationship to the supply air temperature of the AHU.
- Rule 3: The supply air temperature of an AHU has a causal relationship to the air temperature of the zones it supplies.
- Rule 4: The heat gain of a radiator located in a zone has a causal relationship to the zone’s air temperature.

Applied to the ontology-based BIM of the BAS example shown in Figure 1, the four causality rules can create (i.e. materialize) five causality relationships, which are represented as object properties in the ontology. This is shown in Figure 4.

The five materialized causality relationships are the ones, which are labelled with the corresponding rule names. The other causality relationships were created by other rules that are not shown in this paper.

The execution of causality rules on an ontology-based BIM is the first step of the fault propagation approach, and this step needs to be done only once. As a result of this step, the causality relationships (object properties) are materialized and remain in the ontology. The fault propagation rules can use them in all subsequent steps as often as needed. This keeps the fault propagation rules simple and performant. The fault propagation rules are explained in the following section.

B. Fault Propagation Rules

Fault Propagation rules describe qualitatively how faults propagate along the causalities in a BAS, i.e. how faults of equipment manifest in faulty states of variables, and how faulty states of variables propagate and affect other variables. Fault propagation rules thus define the qualitative effects different types of faults can have. Some fault propagation rule examples are shown in Figure 5.
The fault propagation rules from Figure 5 have the following semantics:

- **Rule 5**: A broken state of a boiler results in the boiler supply water temperature being "unheated".
- **Rule 6**: An "unheated" supply water temperature affects causally related variables of type heat gain (materialized by Rule 1) to be in state "no heat gain".
- **Rule 7**: A heat gain variable being in state "no heat gain" affects a causally related supply air temperature (materialized by Rule 2, thus applying to AHUs) to be in state "no heating".
- **Rule 8**: If heating is required, a supply air temperature in state "no heating" has a lowering impact on causally related zone air temperatures (materialized by Rule 3, thus applying to AHUs).
- **Rule 9**: If heating is required, a heat gain variable being in state "no heat gain" has a lowering impact on a causally related zone air temperature (materialized by Rule 4).

**C. Discussion**

If defined in such a generic way, causality rules and fault propagation rules will be reusable rules that can be used for any BAS. The rules can be applied to an associated ontology-based BIM directly, without the need and overhead to create a separate causal model.

In the two sections VI and VII it will be demonstrated how fault propagation rules can be applied to realize forward fault propagation and backward fault propagation.

**V. EXPRESSING RULES WITH SPARQL/UPDATE**

In this Chapter, an alternative to using SWRL for specifying the causality and fault propagation rules is presented, which is the update language SPARQL/Update [30]. The reason why such an alternative can be beneficial and provide a higher functionality is the limitations of the rule language SWRL, as will be shortly explained in the following paragraph.

SWRL is a rule language with a limited expressiveness. It supports only the logical AND and INFER operator, but neither OR nor NOT can be expressed. SWRL was designed to support monotonic inference only. Hence, the rules cannot modify, i.e. retract or remove, existing information from an ontology. Monotonicity is also the reason why negation (NOT) is not supported. SWRL rules can basically infer and add new datatype property values (Literals) and object property relationships to an existing ontology. But it cannot change existing information, nor, and more importantly, it cannot create new individuals natively. Applying SWRL rules requires to run a logical reasoner. This comes along with a processing overhead, where the required triples need to be identified and extracted from the ontology, transferred and translated into the reasoner’s internal data representation, on which the reasoner can operate. Once the reasoner has finished the inferencing, the results need to be translated back into triples, which need to be added back to the ontology. Because of that overhead, and because the architecture of existing reasoners requires to have all data in main memory, SWRL reasoning cannot be applied to large RDF triple stores.

SPARQL/Update, which is an update language for RDF graphs, provides a greater expressiveness and functionality than SWRL does, and it can be used as alternative to SWRL. The most important advantage compared to SWRL is its support of OR (UNION), negation as failure (FILTER NOT EXISTS), its ability to change existing information (INSERT, DELETE) and to create new individuals (BIND, INSERT). Besides, it offers strong filtering capabilities (FILTER), optional graph patterns (OPTIONAL) and conditional logic (IF, BOUND). And all of this can be executed by a standard SPARQL query engine, even on RDF triple stores and very large data sets.

But before we show how to use more of those advanced features for fault propagation, let’s start with three basic examples first. The first SPARQL/Update query, which is shown in Figure 6, is the equivalent of the SWRL causality rule 1 from Figure 3. The rule body is defined in the WHERE part of the query, and the rule consequence in the INSERT part. The WHERE part defines a graph pattern, which is to be matched with the RDF graph, i.e. the ontology-based BIM. It defines the same things as the rule body of the SWRL causality rule 1, but as a graph pattern instead, i.e. it looks for individuals of class ‘Boiler’ that have an individual of class ‘SupplyWaterTemp’ associated via the object property relationship ‘hasVariable’, and that furthermore have a ‘supplies’ relationship defined to an...
individual, which has a variable of class ‘HeatGain’ associated. Or with other words: It looks for boilers and their supply water temperature variable, and downstream equipment and their heat gain variables, which is supplied by the boiler. Finally, for each supply water temperature and heat gain variable pairs that are matched by this whole graph pattern, a ‘hasCausalRelationship’ object property relationship is added to the ontology by the INSERT part of the SPARQL/Update query.

A second SPARQL/Update query example is shown in Figure 7.

This is the equivalent of the SWRL fault propagation rule 5 from Figure 5. For each successful graph pattern match, it relates the boiler supply water temperature variable with the state individual ‘Unheated’, which means that it is in state unheated.

As third example, another fault propagation rule is shown in Figure 8, which is the fault propagation rule 8 from Figure 5. It materializes a ‘hasLoweringImpact’ relationship between two variables, which expresses a lowering impact of an unheated supply air temperature to the causally related zone air temperature.

After the three basic examples, which do not add anything that cannot be expressed with SWRL, let’s consider the more advanced features of SPARQL/Update. With the powerful keywords UNION, OPTIONAL, IF, BOUND, FILTER, FILTER NOT EXISTS, more comprehensive rules can be defined that contain OR, negation as failure or even conditional logic. This allows for more expressive, sophisticated rules, and an overall more advanced inferencing capability. An example SPARQL/Update query, which uses the OPTIONAL, IF, BOUND and FILTER keywords, is illustrated in Figure 9. This query is an advanced version of the fault propagation rule 8 (cf. Figure 5, Figure 8). It comprises negation as failure and conditional logic to distinguish between two different cases: 1) There is no other (possibly upstream) heating device (e.g. boiler, radiator), which can directly or indirectly heat the zone; 2) There is at least one other (upstream) heating device that can directly or indirectly heat the zone. This is expressed by the OPTIONAL graph pattern, which tries to match other upstream variables, which are causally related to the zone temperature, and which belong to a heating device. If no other heating device can be found, which is evaluated by the following IF-BOUND statement, then an inevitable lowering impact (‘hasInevitableLoweringImpact’, i.e. hard consequence) of the unheated supply air temperature on the zone air temperature is deduced and materialized by the INSERT statement, otherwise just a possible lowering impact (‘hasLoweringImpact’, i.e. possible consequence). These two different object properties can be used by the fault propagation to distinguish between possible and inevitable consequences of faults in a BAS. Here it means that whenever it is known that there is no other heating device, which is causally related to the zone, then a breakdown of the only existing heating device inevitably results in a too cold zone when heating is required.
Figure 9. Fault propagation rule 8 as SPARQL/Update query with OPTIONAL, IF and BOUND keywords.

Another advantage of SPARQL/Update compared to SWRL is the capability of creating individuals by the rules. In all shown examples in this paper, the equipment variables are only associated with general, predefined state individuals via object property relationships. With the expressiveness of SPARQL/Update however, one can define for example equipment specific rules, which materialize, i.e. create, equipment specific state individuals, which are not shared with other variables and other equipment. Those state individuals can get particular variable-specific data attached, e.g. the paths along which a fault propagated, or explanation strings.

However, there is also an important disadvantage of using SPARQL/Update compared to SWRL reasoning. SWRL reasoners are true rule processing engines, as we know it from expert systems and artificial intelligence. The rule engines implement a particular, sophisticated rule execution strategy, i.e. they decide which rules to select and apply in which order, so that the whole reasoning process is accelerated and optimized. The reasoning process, be it either forward chaining or backward chaining, is automatically controlled by the reasoner itself and automatically stops when all applicable rules have been fired and no further facts can be inferred. A query engine on the contrary typically does not have this kind of ability. It simply runs queries in the particular order as they are invoked by the user or application. In order to turn a SPARQL query engine into a rule-based system, one has to implement a rule execution strategy. SPARQL/Update rules need to be selected and called in the appropriate order and number of times. A particular generic fault propagation rule, for example, needs to be executed as many times as a fault can maximally propagate in a system, because it performs only one fault propagation step per execution.

This extra work of implementing such a query execution strategy however is doable, despite a basic approach would certainly not reach the perfection of sophisticated reasoners. A simple approach is to run the causality rules first, and as often until no further triples can be inferred. Then all fault propagation rules can be alternately executed N times, where N is the length of the longest path of ‘hasCausalRelationship’ relations in the ontology-based BIM. This approach ensures completeness, i.e. that all causality relationships are materialized and faults are completely propagated in all possible directions. It is however not optimal in terms of minimal number of executed queries to completely propagate all faults. But even this could be improved by implementing more advanced strategies, if performance is critical.

VI. FORWARD FAULT PROPAGATION

Now that the causality and fault propagation rules have been introduced in the previous two chapters using two alternative languages (SWRL vs. SPARQL/Update), it is now time to explain how the fault propagation works in both the forward direction (this chapter) and backward direction (next chapter). The fault propagation requires an ontology-based BIM, in which all causality relationships have been materialized by the causality rules. So here we assume that the causality rules have been executed already.

Forward fault propagation is the procedure of deriving the effects a fault can cause in a BAS. It applies the fault propagation rules in forward direction (forward chaining) and can thus identify which equipment and rooms/zones might be affected by the root fault and how.

The workflow of forward fault propagation is shown in Figure 10.
It starts with a fault reported by the BAS (e.g. detected by a FDD algorithm) or by a user (e.g. field engineer, building manager, service personnel). The fault propagation rules (either specified in SWRL or SPARQL/Update) are then executed (“fired”) incrementally to deduce the manifestation of the fault in the corresponding variables and their propagation along the causalities. By that all potential consequences of the fault are derived and known. Because fault consequences can have various impacts (e.g. a lowering or increasing impact on a zone air temperature) depending on the current situation (e.g. heating required, cooling required, ventilation required) a post-processing needs to be performed. All derived consequences are weighed up and an overall conclusion and explanation of the effects of the fault is derived. The explanation is output to a specialized, subsequent algorithm for further processing (e.g. emergency and maintenance scheduler, room occupancy planner etc.) or to a user.

Let’s consider the example of a broken boiler in the BAS from Figure 1 during cold weather season where heating is required.

Given the ontology-based BIM with materialized causality relationships and the two facts (triples) “Boiler1 hasState Broken” and “Site1 heatingRequired true”, fault propagation Rules 5 to 9 can be fired and infer the consequences as shown in Figure 11. Based on the inferences, and because no other equipment for heating the zone besides the AHU and radiator exist, the post-processing can derive and output a final explanation: “The air temperature in zone 1 cannot be controlled, because the radiator does not work and the AHU cannot heat the supply air. If heating is required, the zone air temperature will be too low.”

The fault propagation paths, i.e. how faults have propagated in a BAS, are tracked by our system. A mechanism for recording the paths is incorporated in the fault propagation rules (omitted in the figures in this paper for simplicity). For every step a fault is propagated, the corresponding path is encoded as String and attached to the destination node, i.e. the variable it propagated to, as OWL datatype property value. The String encoding contains the IRI of the equipment or variable the fault propagated from (source node) and its state, followed by the destination node IRI and its state.

As an example, the propagation of the boiler fault to the boiler supply water temperature being in state unheated (cf. Figure 11) can be encoded by the String “Boiler1-hasState-Broken->Tsw-hasState-Unheated”, with “Boiler1”, “Broken”, “Tsw”, “Unheated” being placeholders for the corresponding IRIs of the individuals. Whenever the fault did not originate at the source node, but was propagated there from an upstream node, the tracked path stored in the source node is concatenated with the current path and stored in the destination node, thus iteratively building up the complete path over the whole fault propagation chains. The complete fault propagation path starting from “Boiler1 hasState Broken” to “Tsa hasLoweringImpact Tza” (cf. Figure 11), given the situation “Site1 heatingRequired true”, is encoded like this:

\[(Boiler1\text{-}hasState\text{-}Broken\rightarrow Tsw\text{-}hasState\text{-}Unheated}
\rightarrow Qhe\text{-}hasState\text{-}No\_Heat\_Gain)\&(Site1\text{-}heatingRequired\text{-}true)
\rightarrow Tsa\text{-}hasLoweringImpact\text{-}Tza\]

VII. ACKWARD FAULT PROPAGATION

Backward fault propagation is the procedure of identifying the root cause(s) of an abnormal situation in a BAS. Unlike forward fault propagation it does not follow
the direction faults propagate, i.e., in the direction of the causality relationships, but considers the opposite direction. Given an abnormal situation it follows the fault propagation paths backwards and determines all possible root causes. So it starts from the other end of fault propagation chains.

The workflow of backward fault propagation is shown in Figure 12. It starts with a reported abnormal situation, e.g., it is too cold in a zone, or a radiator is not heating. Then it performs the backward fault propagation. At a first glance it seems natural to realize it by a backward chaining of the fault propagation rules. Aside from backward chaining is not supported for SWRL, implementing it in the following way is more efficient: The solution applies the fault propagation rules in the forward direction as a first step also. But instead of starting with one or a few given, known faults, all equipment is assumed to be faulty at the beginning, unless known otherwise. This means that all fault states the equipment in a BAS can be in, are set active in the ontology-based BIM. Then a reasoner (if rules are implemented with SWRL) or alternatively a query engine (if rules are implemented with SPARQL/Update) is started, executing the fault propagation rules until all possible fault consequences and their fault propagation paths have been derived. This whole inferencing needs to be done only once at the beginning.

After that step the BIM contains all fault propagation paths of all possible faults modelled in the ontology. Backward fault propagation then simply needs to take all fault propagation path Strings attached to the variable(s) related to the abnormal situation (e.g. Tza in case of the zone temperature being too cold). These paths, which can be numerous, describe all potential propagation paths and root causes of the abnormal situation. All potential root causes for the zone air temperature Tza being too cold, for example, are shown in Figure 13.

Then, under regard of the current situation, these fault propagation paths are analysed. An elimination procedure starts where all unfounded fault propagation paths have to be sorted out. The elimination can be supported by using available measurements from the BAS, which can prove or disprove the variables along the propagation paths being in faulty state or not. Also FDD algorithms (e.g., checking the proper operation of an AHU by checking that the supply air temperature meets the set-point) can be scheduled to run and/or FDD algorithm results can be used in order to detect and confirm or eliminate faults. If no data is available, even a field engineer can be sent to check equipment for faults. Based on that information, the knowledge-based system can eliminate fault propagation paths or confirm them.

Figure 12. Backward fault propagation workflow.
Whenever the proper operation of a plant or equipment has been confirmed, the system can apply reasoning to eliminate faults in other upstream parts of the BAS as well whose proper operation is a prerequisite. It can for example deduce from a properly operating AHU during heating season that the upstream boiler plant must be operating as well. All fault propagation paths originating from a broken boiler can be eliminated based on this deduction.

The elimination procedure is repeated, until only one or a few fault propagation paths and root causes remain, which can be deduced to be the root cause(s) (deduction by elimination) and optionally be confirmed by the mentioned methods. The paths contain the equipment or control faults and their manifestation and propagation in the BAS. This diagnosis, together with the whole fault propagation path, is finally output to a user or subsequent algorithm.

VIII. IMPLEMENTATION

The knowledge-based system for fault propagation has been implemented and successfully been applied at Honeywell. It is implemented in Java and uses the Apache Jena API as ontology framework. For processing SWRL rules, the Pellet reasoner is used, and for the alternative SPARQL/Update-based approach, the query engine Apache Jena ARQ is applied. The OWL ontologies and SPARQL/Update queries were defined in TopBraid Composer and the SWRL rules with Protégé.

Let’s have a closer look at the BAS and fault propagation example from Figure 1, which, amongst other examples, has been implemented and applied in an extended version. In such a BAS layout, consisting of a central boiler plant, AHU, and zones equipped with radiators, at least nine different types of variables are relevant for controlling the zone air temperatures (see Figure 1). Already in a simple BAS with two zones only there exist 22 causality relationships between the overall twelve variables. We identified and modelled 14 types of equipment states related to boiler plants, AHUs and radiators, which are health state ok/impaired/broken, energy supply sufficient/insufficient/no energy, dimensioning sufficient/undersized, controller state ok/faulty/broken and valve/ damper closed/in between/open. Overall 48 fault states can be induced into the BIM or be derived by the rules for the boiler plant, AHU (including fan, heating coil, valve and dampers) and two radiators. These equipment faults can result in 31 different fault state types of causally related variables, with overall 45 fault states of variables modelled in our example. Based on that, the fault propagation rule set is able to infer 138 different single step fault consequences.

The fault propagation as it has been demonstrated so far is a pure symbolic, qualitative approach. It does not incorporate probabilities, weights, heuristics or other quantitative methods. Instead of that, we distinguish between two types of fault consequences based on their severity, which are inevitable consequences and possible consequences. Inevitable consequences are the ones which definitely happen when a specific fault occurs. A broken boiler (i.e. a boiler that cannot heat the supply water) for example will inevitably cause all the consequences shown in Figure 11 during heating season, if it is the only boiler in the BAS. An impaired boiler (i.e. a boiler that is still operating, but with a lower efficiency due to e.g. degradation), undersized boiler, boiler with insufficient energy provided, or boiler with faulty controller, however, may possibly result in these consequences in specific situations (e.g. during very cold weather season demanding a high heating amount), but not necessarily. Without any further computations and analyses the system simply cannot reliably predict what might happen. On the other hand, being able to predict the particular consequences in all situations, would require to have very precise, detailed, first-principle models or simulators available, which is simply not feasible in the resource constrained building automation domain. Our symbolic approach constitutes a doable compromise here.

Fault propagation results can be output to other specialized algorithms for further processing, such as emergency and maintenance scheduler, room occupancy planner, energy calculators or room comfort optimizer. Ideally, the ontology-based BIM, including the propagated faults and their propagation paths, is handed over or hosted by an SPARQL endpoint that allows for querying it. The algorithms have then comprehensive knowledge available to start further computations and initiate actions. Alternatively, the communication can be API-based.

For interacting with a user and presenting the fault propagation results, dedicated user interfaces can be created dynamically, based on the context information available in the ontology-based BIM. The user interfaces are context-dependent, and can particularly show the fault explanation together with all (potentially) affected equipment and variables. Current values and historical trends for these variables can be visualized, thus allowing the user to observe the real situation and consequences in the BAS.

IX. CONCLUSION

This paper presented an ontology-based fault propagation approach, which can enhance fault detection and diagnosis in BAS by making manual, time-consuming engineering tasks solvable by a computer. Implemented as a knowledge-based system that operates with rules, forward and backward fault propagation can be performed automatically. It combines an ontology-based BIM, which models a BAS formally and semantically, with expert knowledge formalized as rules to automatically determine causalities in a BAS and to propagate faults along the causalities. The fault propagation approach enables an immediate assessment of consequences of possible faults respectively an analysis of the root cause(s), which allows for quickly reacting to faults. The fault propagation approach and rules can be universally applied to any given BAS, for which an appropriate ontology-based BIM is provided.
The rules can be defined with either SWRL and executed by a reasoner, or as SPARQL/Update queries, which can be processed by a SPARQL query engine. SWRL reasoning has the advantage that the reasoner itself takes care of the proper forward or backward chaining rule execution strategy. SPARQL/Update however is more expressive and allows for far more sophisticated, graph-based queries and inferences on the ontology graph.

The fault propagation approach enables a new paradigm. Instead of modelling and analysing single pieces of equipment of a BAS alone (equipment-centric FDD), BAS can be considered as a whole. The knowledge-based system is aware of the complex, potentially far-reaching consequences of faults. It forms an abstraction and coordination layer for BAS, and can coordinate and combine existing, equipment-centric FDD algorithms and results.

Altogether, the fault propagation approach provides a better understanding of BAS and their behavior, and supports the decision making and prioritization of the right emergency and maintenance actions. Faults can even be simulated in advance to know what might happen then, thus providing a higher awareness of the faults’ severity and enabling the preparation of emergency strategies in advance. A fault causing unacceptable air temperature conditions in a frequently used meeting room, for example, should be scheduled for repair. In the meantime, all planned meetings must be rescheduled to other available, unaffected meeting rooms. Their availability influences the priority of the repair. Faults in a supermarket can have even more severe consequences. An air temperature increase can be very critical to the food. If a certain temperature is exceeded, the food will have to be disposed. Here, an immediate notification and reaction, e.g. moving the food to another place, is essential.

ACKNOWLEDGMENT

The research leading to these results has received funding from the People Programme of the European Union’s Seventh Framework Programme under REA grant agreement no. 324432 (project AMBI – Advanced Methods for Building Diagnostics and Maintenance). Patent Pending.

REFERENCES


