Automatic Generation of Human-Computer Interfaces from BACnet Descriptions

Lawrence Henschen(✉), Julia Lee, and Ries Guthmann

Northwestern University, Evanston, IL 60208, USA
henschen@eecs.northwestern.edu, j-leeh@comcast.net,
s2t9k3@u.northwestern.edu

Abstract. We present a methodology by which interfaces can be generated for application areas that have standards for definition of systems within that application area. The methodology includes organizational rules that describe the general nature of information for the application area, operational rules that describe the way users interact with that data, and optional user preference rule by which users can tailor the interface for a more meaningful experience. We show that by developing multiple sets of operational and user preference rules our approach can provide for universal access. We demonstrate the methodology for applications defined by BACNet, a standard for defining building control and monitoring systems and which can be used to also define general Internet of Things systems. We provide a brief description of BACNet objects and show how the application area leads to organizational, operational, and user preference rules for BACNet systems. We also illustrate the approach applied to a second application area to show the generality of the method.

Keywords: User interfaces · Automatic interface generation · Universal access BACNet

1 Introduction

The Internet of Things (IoT) is the next revolution in computing. Although many of the envisioned IoT applications are meant to be totally autonomous, many other IoT applications will involve a significant amount of human interaction. The breadth of application areas is incredibly large – health, assisted living, automation, agriculture, smart buildings and cities, just to name a few. Still, IoT nodes across this vast array of applications have great similarities. Many user interfaces for IoT have been developed for specific applications, for example [1, 2]. However, it will be grossly inefficient to have to develop user interfaces for each new IoT application from scratch. Moreover, various network interfaces, such as REST [3], focus on network structure and/or the command aspect of interacting with sensor nodes and do not address the issue of robust and usable user interfaces. There are some proprietary IoT development systems [4, 5], but these do not have all the features needed for robust development and, of course, are proprietary. Our recent work has focused on the development of a rule-based method for automatically generating a user interface from the descriptions of the underlying objects

N. Streitz and S. Konomi (Eds.): DAPI 2018, LNCS 10921, pp. 71–84, 2018.
https://doi.org/10.1007/978-3-319-91125-0_5

in the application – nodes, sensors, controls, hierarchical objects, etc. Moreover, by varying the rule sets interfaces for a variety of users with special needs (visually impaired, manually impaired, etc.) can be generated. Such a methodology will make the development of user interfaces for IoT applications efficient and simple and at the same time provide for universal access to IoT applications.

In [6] we proposed the use of mark-up languages to describe IoT nodes with sensors and actuators. While this approach holds the promise of being a general method for use in literally any application, it does have a drawback that is quite serious for many IoT applications. Messages containing marked up text are orders of magnitude larger than messages coded in carefully designed bit/byte packed representations. For applications in which the nodes are battery operated, the size of marked-up messages may preclude their use, especially when wireless communication is used. Unfortunately, a bit/byte packed representation for one application will not be usable by other applications, thus precluding the development of a generic methodology for automatic generation of user interfaces across many applications.

In this paper we propose the use of a generic approach for automatically designing interfaces for broad application areas. We show how the presence of formal standards facilitates the generation of rules that determined how to automatically construct the interface for particular application in the chosen broad area. We illustrate this approach with a particular application area, building control, using the formal representation system BACNet [7, 8]. BACNet is both robust in its representational power and formally defined in such a way that good user interfaces can be generated automatically. Its message format is much simpler than xml-based formats, so it solves the issue of energy loss due to large messages while still being formally defined and generic. We begin with a classification of rules about generating interfaces. We then present a brief overview of BACNet and the reasons why it is a good choice for this work. We then list the main features of BACNet that must be used for the user interface. We show sample rules for generating user interface details from BACNet descriptions and show examples of the use of such rules. We also include a discussion of how alternate sets of rules can be used to increase universal accessibility. We show the generality of our approach by applying our methods to a second application area. We close with a restatement of the general approach and why it is important for both IoT and HCI. We also indicate areas of future work.

2 Rules for Generating Interfaces

In our approach we use three sets of rules for generating interfaces – organizational rules, operational rules, and user preference rules. Organizational rules are based on the nature of the application and its data. They describe the general way in which the data in the application area should be presented and the general ways in which a user could interact with the system. We emphasize that these rules depend on the application area only, and they are at a high and very general level. For example, as described in detail in Sect. 5, BACNet objects have several different kinds of properties, and in our example we make use of that general feature about BACNet objects to specify how information should be

organized for rendering. Operational rules describe the ways in which a user actually interacts with the system. Operational rules are further partitioned into output rules (how information is rendered) and input rules (how the user enters information). The main motivation for this portioning will be given in Sect. 5.3. Operational rules are based on the organizational rules but can be tailored for particular user groups, such as visually impaired users or users with impaired dexterity. This allows a system using our approach to provide universal accessibility for the chosen application area. Finally, we allow for an optional set of user preference rules. These rules allow individual users to adjust the operational aspects of the interface. Details of these three sets of rules will be illustrated in Sect. 5.

3 Why Use BACNet?

BACNet [7, 8] is a standard for representing the structure and components of embedded systems, which are the foundation of the IoT. It provides for the definition of objects which can sense or control their environment, objects that can process data that is collected by the sensing objects, objects that define the scheduling of tasks, and many others as indicated in the list in the next section. These are exactly the components that make up most embedded systems. It was originally introduced as a way of defining control systems for buildings, and it is used widely in that application [9]. However, because of its generality and completeness it is also being used in many other applications and may well become a standard tool in the development of IoT applications.

BACNet is formally defined with a standards organization [7] that maintains the standard and provides for future development and expansion. Even in its present state BACNet is more complete than other tools, such as Bluemix [4] and Kaa [5]. Moreover, the standard itself provides for both future expansion and user-defined extensions. (Of course, user-defined extensions may require user-developed interfaces on top of the automatically generated interface that we describe in this paper, but that is a reasonable expectation. However, as explained in Sect. 4.2, if done carefully the method we propose can incorporate the extensions without further programming). BACNet is a generic representation methodology used to describe the elements of a system but does not require the use of proprietary systems or software, as is the case for systems like Bluemix and ones provided by other large companies. Finally, we note that Bluemix and other such systems at present do not offer interfaces for users with special needs, such as visually impaired users or physically impaired users. A key feature of our approach is that providing for universal accessibility is straightforward.

In addition to the features mentioned in the preceding paragraph, BACNet has two other key features that allow for the kind of automatic user interface generation we seek. First, in any system defined by BACNet the host computer maintains a model of the objects in the system. The host computer is typically the one through which the human user will interact with the system, so the HCI portal has complete information about all the objects that need to be rendered. Moreover, the BACNet host must provide access to the information about the objects. Thus, the interface system simply requests information when it needs and issues writes when the user wants some property of the system

changed. Second, there are a limited set of object types and these are well defined. There is a modest set of features that are universal to all BACNet objects, and for each BACNet type there is a modest set of features for objects of that particular type. Thus a system that can generate renderings for this limited set of objects is relatively easy to develop and implement.

To summarize, BACNet is a formally defined and standardized representational system that handles all the features found in embedded systems and IoT "things". It is of modest size. In any application it would likely run on the same computer used for the user interface. It is likely to become widely used in the IoT field. For these reasons, BACNet is a good choice to use as the basis for automatically generated user interfaces for IoT applications. The availability of such a system means that any IoT application immediately has an acceptable user interface as soon as the system itself has been defined in BACNet.

4 Overview of Main BACNet Features

In this section we describe the main features of BACNet that would be used to generate the user interface. Space precludes describing all of the features that would be used for a robust user interface or mentioning BACNet features that are not relevant to the user interface. The interested reader is referred to [8] for a good overview of BACNet objects and other features and to [7] for the formal definition of the standard. In this paper we focus on the main features used in the generation of user interfaces and on ones that illustrate the principles of the method we are proposing. Most of the material in this section is taken directly from [8].

4.1 BACNet Objects

The major elements of BACNet systems are defined as BACNet objects. There is a top-level "object" type, and there are more special types derived from the base object type. A BACNet object may contain other BACNet objects, providing for definition of hierarchical systems. The following is the list of all BACNet object types.

- Basic object types.
  - Device
  - Analog input
  - Analog output
  - Analog value
  - Binary input
  - Binary output
  - Binary value
  - Multi-state input
  - Multi-state output
  - Multi-state value
  - File

- Process-related object types
  - Averaging
  - Loop
  - Program
- Control-related object types
  - Command
  - Load control
- Meter-related object types
  - Accumulator
  - Pulse converter
- Collection-related object types
  - Group
  - Global group
  - Structured view
- Schedule-related object types
  - Calendar
  - Schedule
- Notification-related object types
  - Event enrollment
  - Notification class
  - Notification forwarder
  - Alert enrollment
- Logging object types
  - Trend log
  - Trend log multiple
  - Event log
- Safety and security object types
  - Life safety point
  - Life safety zone
  - Network security
- Physical access control system object types
  - Access point
  - Access zone
  - Access door
  - Access user
  - Access rights
  - Access credential
  - Credential data input
- Simple value object types
  - Character string value
  - Large analog value
  - Bit string value
  - Integer value
  - Positive integer value
  - Octet string value

  - Date value
  - Time value
  - Date/Time value
  - Date pattern value
  - Time pattern value
  - Date/Time pattern value
- Lighting control object type
  - Channel
  - Lighting output

4.2 Object Properties

Every BACNet object type has a set of associated properties. The BACNet standard specifies that each property has three attributes – property identifier, property datatype, and conformance code. The property identifier is a string describing what that property represents, for example "Present_value" or "Units". The property datatype is a string identifying a primitive datatype, such as INTEGER or REAL or CHARACTERSTRING, or one of the BACNet constructed data types, such BACNetAddress. The conformance code is one of "R" (the property is required and can be read by using the BACNet built-in services), "W" (the property is required and can be both read and written through BACNet services), or "O" (the property is optional). The latter is included mainly to accommodate proprietary extensions to the BACNet standard. The interface system simply uses the BACNet services to obtain the information for all the BACNet objects in that system and to change values of those properties that are writeable.

The following set of properties is common to all BACNet objects:

- Object_identifier – contains the BACNet object type plus the instance number
- Object_name
- Object_type
- Property_list – the list of all properties associated with this object
- Description – an optional property that is a string describing the objects use, purpose, or other aspect
- Profile_name – an optional property that allows extensions to the standard BACNet feature set (typically for proprietary extensions)

Individual object types have additional properties. For example, object types having a value field might also have a "UNITS" property. Objects that sense the environment might have an "EVENT_STATE" property. Many object types have a "STATUS_FLAGS" property. The Analog_Input_Object type, for example, has 29 properties in addition to the six properties mentioned above common to all objects, and the Device object type has fifty additional properties. Although at first glance this might seem like a lot, the rendering mechanism is still quite simple – use BACNet services to obtain the properties and their current values and then render. Whether there are six or thirty-five has no bearing on the complexity of the rendering algorithm.

An important property that occurs in some of the object types, in particular in the Device object type, is the "Object_List" property. For a given object A this property is a list of the objects that are inside A. For example, a sensor node might have several sensing devices (i.e., input objects) and possibly even an actuator device (i.e., an output object). The "Object_List" allows the description of this hierarchical structure. In the interface then, the "Object_List" would allow the user to zoom in to examine or control the internal objects.

We present portions of the property lists for a few sample object types in preparation for the next section. Space precludes showing the entire lists. We only show enough so that the reader will better understand and appreciate the rules used to automatically generate the user interface. The following properties are, of course, in addition to the six mentioned above that occur in all BACNet objects. For the purposes of generating the interface it is not important to understand the meaning of each property, although we have chosen ones whose meaning should be fairly obvious. Similarly, it is not important to understand the specifics of each data type, only to know that each type is well-defined in the standard and thus can be rendered with suitable software.

Sample Properties of the Analog Output Object Type

Property	Datatype
PRESENT_VALUE	REAL
EVENT_DETECTION_ENABLE	BOOLEAN
UNITS	BACNetEngineeringUnits
MIN_PRES_VALUE	REAL
RESOLUTION	REAL
EVENT_MESSAGE_TEXTS	BACNetArray of CharacterString
…	

Sample Properties of the Scheduling Object Type

Property	Datatype
PRESENT_VALUE	any primitive datatype
LIST_OF_OBJECT_PROPERTY_REFERENCES	BACNetDeviceObjectPropertyReference list
WEEKLY_SCHEDULE	BACNetArray[7] of BACNetDailySchedule
EXCEPTION_SCHEDULE	BACNetArray of BACNetSpecialEvents
…	

Sample Properties of the Program Object Type

Property	Datatype
PROGRAM_STATE	BACNetProgramState
INSTANCE_OF	CharacterString
Ref1	proprietary
Ref2	proprietary
…	

The analog output object type illustrates the ability of BACNet to model the standard kinds of "things" in an IoT application. The scheduling object type is an example of the ability to model time-dependent aspects of applications. The program object type illustrates one way that proprietary extensions can be made to BACNet in ways that are compatible with the standard and facilitate the incorporation of such proprietary extensions into the automatically generated interface. All of the properties are BACNet standard properties except the Refi properties of the program object type. Most of the properties are self-explanatory. For the program object type, the INSTANCE_OF property is a string giving the name of the (proprietary) function to be called, and the Refi properties are non-BACNet properties that represent the arguments that are to be passed to that function. These examples are sufficient to illustrate almost all the aspects needed for user interfaces to BACNet systems, as will be illustrated in the next section.

5 A Sample Generic User Interface that Can Be Automatically Generated

In this section we present an example of a user interface that can be generated automatically from any BACNet description and describe some of the rules that would be used in the automatic generation process. As noted in Sect. 5, the use of rules for the generation of the interface means that many other renderings are possible, including especially interfaces for users with special needs. Our choice here is only for illustration and is not necessarily the best for all purposes.

5.1 Organization of the Sample Interface

All BACNet objects have the six common properties. Each object type has its own set of additional properties. Some of these properties can be considered as parameters of individual objects, and these properties would typically not be changeable. For example, a particular analog output device would have fixed minimum and maximum output values, such as 0–5 V. Other properties can be changed. For example, the output of an output device should be controllable through the interface so that the user can make that device change its output value. Finally, some properties only become relevant or get values when an event occurs resulting in some kind of notification or alarm.

This suggests a visual interface with four panels. One panel contains the properties common to all BACNet objects. We decide not to display the property list because the properties will be shown in the other panels. A second panel contains parameter information for the object. A third panel contains information about the current value or status of the object. A fourth panel would show information relevant to events occurring in the object or events occurring in the system that affect that object. Such events include scheduled notifications, alerts, and error conditions. The fourth panel might be shown only when a relevant event has actually occurred; appearing would draw attention to the occurrence of the event, similar to the way a pop-up window draws the attention of a user. Learnability, usability, and other important features of good human-computer interfaces are enhanced by this common rendering. No matter what kind of object is being displayed, the user knows where to look to find the four different kinds of information. Figure 1 shows a sample rendering for an analog output device. The plus in front of "Event messages:" indicates a list that can be expanded. Figure 2 shows the same object with an event that occurred in a different object, a temperature sensor in the same room, that is relevant for this analog output object.

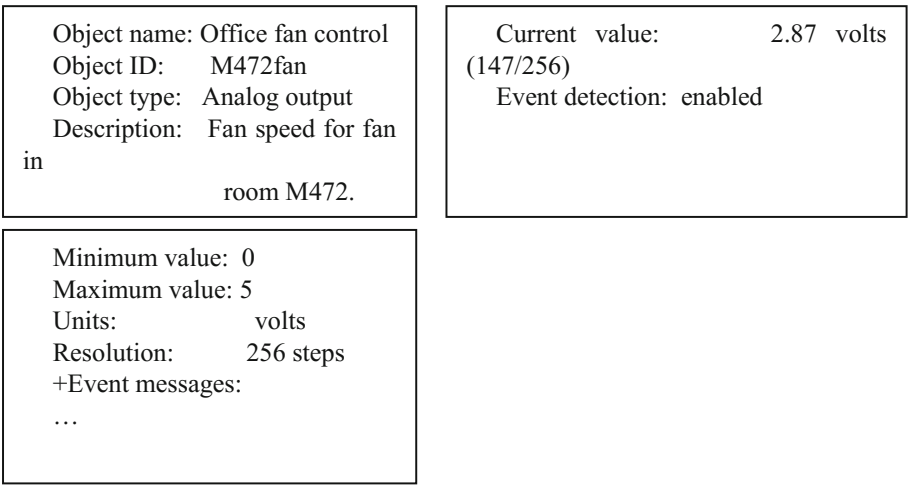

Fig. 1. Sample rendering of an analog output device showing the three main panels.

5.2 Rules for Generating Our Interface

As described in Sect. 2, we organize the rules for automatically generating the interface into three sets - interface configuration rules, operational rules, and user preference rules. Again, because of space restrictions we only present examples of the rules.

Recall, configuration rules are rules based on the nature of the application area and its data and provide the general specification of how the user will interact with the system. In our case this set contains the rule about rendering information in panels. Further, for each BACNet object type there is a rule specifying placement of the information about objects of that type into particular panels and the format for displaying

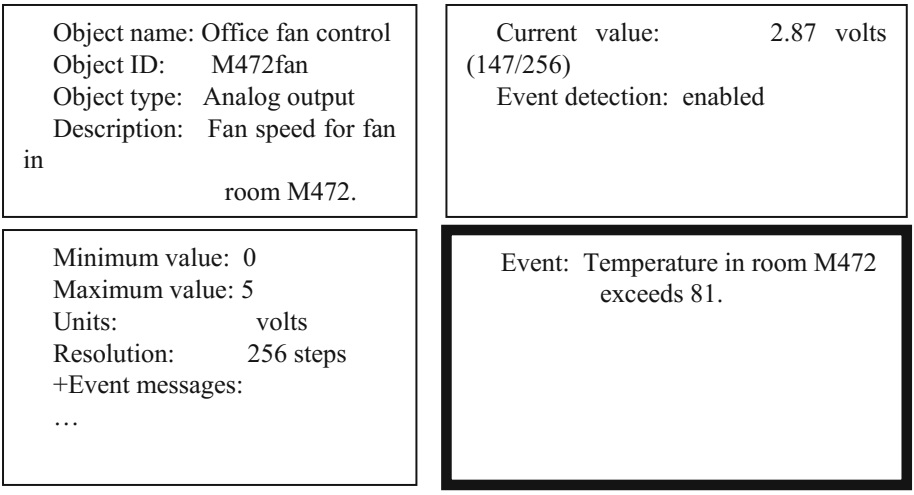

Fig. 2. Sample rendering showing the events panel.

that information. The decision whether or not to make the fourth panel permanently rendered or displayed only when appropriate events actually occur falls into the configuration class.

Operational rules determine the behavior of the interface when users are actually using it. We further divide this class of rules into those dealing with outputs from the interface and those dealing with inputs to the interface. Examples of operational rules for output include expanding/contracting composite values depending on whether the user has clicked the corresponding ± symbol. An example of an operational input rule is to allow the user to click on a ± symbol. Another input rule, one that is particularly important for embedded systems and IoT applications, is to allow users to highlight object output values (such as the fan speed in the example in Sect. 5.1) and type in a new value. BACNet objects can contain other BACNet objects, and the "Object_List" property holds the list of object identifiers of these contained objects. Clicking on an object identifier will cause the referenced object to be rendered. There are a variety of operational rules for how a multiplicity of objects should be handled. For example, the objects being rendered can be stored in stack-like fashion. When a user clicks on an object reference in one object, say A, that object is saved on a stack, and the new object is rendered. When the user has finished with the currently rendered object, the object on the top of the stack is again rendered. Such a rule allows a user to explore the structure of the IoT system by digging into the nested structure of the objects. Of course, there may be other rules for handling such situations.

Of particular importance are operational rules that specify how the interface is to interact with the underlying IoT system. In our example these rules would specify how the information to be rendered is obtained and what is to be done when the user inputs values that are supposedly changeable in the underlying system. For BACNet interfaces these are quite simple. The BACNet standard specifies that BACNet services are to be provided by which properties and their values can be retrieved for any object in the

system (BACNet read services) and property values that are changeable can be written with new values (BACNet write services). This is a key and critical feature for automatically generating BACNet interfaces. It is not necessary to read and parse the BACNet specification of the underlying IoT system, which could be in xml or even computer code. All the interaction can be handled through BACNet services, which are standard in all BACNet systems.

As noted, BACNet read and write services allow access to properties of all objects in a given IoT application, even objects that are not standard in BACNet. Venders who incorporate extensions should provide such access services to compliment the BACNet built-in services. If the vender has mapped the user-accessible features of the extension into properties of any associated objects, then our system can still automatically generate the interface even for the extensions. Consider the Program Object Type illustrated in Sect. 4.2. The *Refi* properties are not in the BACNet standard. However, according the BACNet standard the vendor providing such a function as an extension should also provide means to read and write those properties. Now, from the point of view of automatically generating the interface, our system simply reads the list of properties and their related values and allows the user to write new values for writeable properties. We may decide as part of the configuration to generate a fifth panel for proprietary properties. Finally, if the value typed by the user into the interface for a writeable property is simply passed through to the object and the parsing of such (for example translating a string into a real number) is handled on the vender side, then literally no extra work is required for the interface generation.

Although not strictly necessary, the inclusion of rules that allow users to set preferences for some interface features provides for a more flexible and user-friendly experience. In the case of our BACNet interface, we allow users to specify how certain values are rendered. For example, in the BACNet context binary values can represent 0 and 1, ON and OFF, ACTIVE and NOT_ACTIVE, and a variety of other meanings. Analog values could be rendered as numbers within a range, slide bars, meters, and a variety of other ways. Each individual object can have its own representation different from other objects, even objects within the same object type. For example, a user viewing a binary input object representing a door would likely want to see the values as OPEN and CLOSED, while the two values for a binary input object representing a light should be rendered as ON and OFF. Allowing the user to specify renderings other than the default rendering provides for a richer and more meaningful interaction experience by making the rendering of each individual object match more closely to the user's real-world concept of that object. Note that extensions to the standard may suggest additional user preference rules.

5.3 Alternate Rule Sets and Universal Access

It is easy to see that by changing the operational and user preference rule sets the system can be made to generate different kinds of interfaces, in particular interfaces for users with special needs. We illustrate this idea for a few cases.

A visually impaired user would likely prefer voice output. Some BACNet object types have relatively few properties so that reading all the properties and their values is

acceptable. Others have too many to make reading the entire set of information useful. A configuration rule would determine which objects fall into which category. For those in the second category, additional configuration rules would determine the mechanism for a user to get the information that was not rendered at the time the object was first presented. The partition into panels still applies, but the initial rendering of an object would read the information to the user instead of display it on a screen. The operational rules, then, might be to speak the common properties (name, ID, description) and their values for every object and inform the user that information about the configuration (panel 2) and current values (panel 3) are available on request. The amount of information about the current values associated with an object is relatively small for all object types, so the interface could simply read the material that would be printed in panel 3 for a visual interface. If the user requests configuration information, the interface can read out the list of properties and ask which one(s) the user wants to hear about. Properties with composite values (such as date or a list) would have the name of the property read and then be expanded only if the user requested. User preference rules would include ones used in the visual interface, the difference being that the values would be rendered by voice instead of screen display. For example, the user would hear that a door was OPEN and a fan was OFF. However, there would be rules specifically for visually impaired users, such as selecting between a terse mode and a verbose mode.

By altering the operational rule sets interfaces can be obtained for many different classes of users – visually impaired users who are still good with keyboards, visually impaired users who also need voice input, users with normal vision but no manual dexterity, etc. Separating the operational rules into output rules and input rules makes the goal of universal access, i.e. access by individuals of all kinds and with all kinds of special needs, much easier to attain. We propose the development of a variety of default classes such as “general default”, “visually impaired”, “physically impaired (can’t use hands to enter data)”, etc. that automatically change rendering and user input to match individual users. Following the principle of user-centered design, these rule sets would be developed by studying the needs of different classes of users working in the specific environment of the Internet of Things in such applications as building control, sensor networks, etc. For example, users with normal vision would likely not object to the automatic display of the information in panel 2, even when this information was somewhat lengthy; sighted users just focus their eyes on the parts of the screen in which they are interested. Visually impaired users, on the other hand, would likely object to a long list of object parameter information being read. Studies could reveal information about how visually impaired users actually use such information and appropriate rules for rendering it developed for those users.

5.4 Application of the Method to Another Area

Finally, we illustrate our method in a different application area, namely transportation systems. These could include taxi companies, bus companies, delivery services, and many other similar activities.

For this general application area there would be two kinds of organizational categories – a map and a list of vehicles. This suggests an organizational rule that specifies

the rendering of a map plus the vehicles that are located within the map. Vehicles might partition into three sets – vehicles that are in service but currently busy, vehicles that are in service but available for assignment, and vehicles that are currently not in service. Location of the vehicles must be included in the interface as well as the ability to dig into the details of particular vehicle and to send messages to individual vehicles. There would be no obvious separate panels, as was the case for the BACNet interface. The reader can easily imagine other general characteristics of transportation systems.

Operational rules for a sighted user with normal manual dexterity might include the display of the map and the locations of the vehicles. The three different kinds of vehicles could be distinguished by using colors or shapes, a choice that could be left for user preference. Location would be indicated by position on the map. Clicking on a vehicle would open up the details of that vehicle and provide text box for typing messages to be sent to that vehicle. There might be zoom-in and zoom-out buttons for the user to focus on smaller or larger areas.

Operational rules for a visually impaired user who does not use a keyboard for input would be different. Rather than display a visual map, the coordinates of the four corners might be read out to the user or possibly a name associated with the area of focus on the map. Rendering of the vehicles within the area of focus might be done by first speaking the numbers of vehicles in each class and then letting the user issue a voice command requesting more information, such as location, about one or more of those classes. A user might request the IDs of the vehicles and then request the details about a particular one, again by voice input. The idea, as with the BACNet example, is that a careful study of how visually impaired users would want to interact with the system will suggest a suitable set of operational rules for that class of users.

Unfortunately, there is no standards organization for defining transportation systems, as there was for the building control industry. Therefore, many of the issues involved with the interface interacting with the system itself will not be handled automatically. This shows the importance of having such standards and, by the way, suggests that industries would profit from the development of such standards.

6 Conclusion

We have described a system that can generate user interfaces for any standard BACNet-defined application system. It is anticipated that BACNet will become widely used for implementing Internet of Things applications. A system like we described would therefore have a major impact on the development of the IoT because engineers who develop IoT applications will have an immediate user interface and will not have to devote additional time and effort beyond the design of the BACNet application itself. Moreover, developers will not be forced into using commercial or proprietary systems for their IoT applications in order to avoid having to spend time and money on the interface; they will get the interface automatically. We also illustrated that our approach is general and can be applied to other application areas besides building control.

We made two major contributions to the HCI community. First, we have suggested that in areas of endeavor for which formal standards exist for defining applications those

formal standards be used to implement an interface system that will automatically generate a user interface for any application defined in that standard. Second, we have demonstrated that a rule-based approach to the implementation of such an interface system can be used to achieve universal access in that area of endeavor, in some cases even in the absence of such standards.

Several key features of BACNet made it suitable for our approach. As noted already, BACNet has a standards organization and a formal definition. BACNet allows for non-standard extensions, but these extensions can be made in a way that complies with the formal definition of BACNet and, thus, is amenable to automatic interface generation like we have described. BACNet objects are simple enough so that information about them can be rendered through both visual and audio outputs and inputs to the interface can be accepted by keyboard or voice or any of several other ways. Thus, it is relatively easy to achieve universal access for BACNet applications through the use of multiple rule sets. None of this is possible for general web pages. There are no standards, and even if there were the amount of information on typical pages is way more than can be easily rendered through non-visual means such as reading.

Future work includes, first, the design of a representation system for defining the rule sets. This would allow for universal access without the need to hand code a particular style of interface, as we have done for the demonstration system. Independent of that, ethnographic studies need to be done to determine the best way to render and interact with users with special needs, after which appropriate rule sets can be developed for those classes of users.

References

1. Kosnik, D., Henschen, L.: A Web-enabled data management interface for health monitoring of civil infrastructure. In: Proceedings of the 15th HCII International Conference, vol. 2, pp. 107–113 (2013)
2. Keller, I., Lehmann, A., Franke, M., Schlegel, T.: Towards an interaction concept for efficient control of cyber-physical systems. In: Shumaker, R., Lackey, S. (eds.) VAMR 2014. LNCS, vol. 8525, pp. 149–158. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-07458-0_15
3. Wikipedia. https://en.wikipedia.org/wiki/Representational_state_transfer. Accessed 04 Feb 2018
4. IBM Developer Works. https://www.ibm.com/developerworks/cloud/library/cl-bluemix foundry/index.html. Accessed 04 Feb 2018
5. Kaa Project Home Page. http://www.kaaproject.org. Accessed 04 Feb 2018
6. Henschen, L., Lee, J.: Human-computer interfaces for sensor/actuator networks. In: Kurosu, M. (ed.) HCI 2016. LNCS, vol. 9732, pp. 379–387. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-39516-6_36
7. BACNet Home Page. http://www.bacnet.org/. Accessed 04 Feb 2018
8. Newman, H.: BACNet: The Global Standard for Building Automation and Control Networks. Momentum Press, New York (2013)
9. Schachinger, D., Stampfel, C., Kastner, W.: Interoperable integration of building automation systems using RESTful BACnet Web services. In: IECON 2015 – 41st Annual Conference of the IEEE Industrial Electronics Society, pp. 003899–003904 (2105)