A visual programming approach based on domain ontologies for configuring industrial IoT installations

Matúš Tomlein

Department of Computer Science
Aarhus University, Denmark
matus@cs.au.dk

Kaj Grønbæk

Department of Computer Science
Aarhus University, Denmark
kgronbak@cs.au.dk

ABSTRACT

Industrial products tend to be customized by third-parties for different use cases. This is currently supported by adding external Programmable Logic Controllers (PLC) to installations. As IoT software ecosystems become wide-spread, the need for such hardware will decrease. However, removing the controllers opens the challenge of configuring distributed industrial installations. This paper argues for modeling application requirements on industrial installations so that they can be evaluated automatically based on information about targeted installations. GIMLE, a visual language for modeling application requirements using expressive domain knowledge, is proposed. GIMLE enables modeling requirements on physical features of installations, which hasn't received significant attention in the related work. A study with domain experts is used to reflect on the proposed process. The scalability of the visual language is evaluated using a model for a real-world application. The support for reuse of requirements enabled us to build models that can add support for new installations without significant changes and with a slower increase in size the more components can be reused.

ACM Classification Keywords

I.2.4 Artificial Intelligence: Knowledge Representation Formalisms and Methods: Representation languages; D.1.7 Software: Programming Techniques: Visual Programming

Author Keywords

End-user programming; visual programming; modeling.

INTRODUCTION

The Internet of Things has the potential to spur innovation in the industrial domain just like in the smart home. By opening software ecosystems, it can enable third-party developers to create applications that satisfy needs that would otherwise be uncovered [8]. However, the industrial domain also opens new challenges in the configuration of smart environments [17]. While research in the smart home domain has focused on empowering end-users to configure and connect heterogeneous

products [11], industrial installations are controlled and monitored by applications that work with dozens of sensors and actuators with specific roles and placements within the installations. They have requirements on the specific sensors, actuators and installation types that they support. The extensive requirements make the configuration difficult and error-prone when left completely to the installers or end-users.

Thus, the requirements of applications have to be explicit and possible to evaluate automatically. This means taking expertise from domain experts and modeling their knowledge so that it can be reasoned about automatically. Although research and practice in software variability management has enabled customizable software products [26] and semantic service descriptions have been used for composition of distributed and heterogeneous services [23], we see gaps when it comes to modeling requirements on the physical parts of installations in ubiquitous settings, such as the placements of devices in installations, their interconnections, roles and relations. Supporting domain experts, who may not be software developers, in building expressive models of the installations remains a challenge [18].

This paper presents GIMLE (Graphical Installation Modeling Language for IoT Ecosystems), which proposes to use general-purpose visual programming styles to model domain knowledge through expressive ontological requirements. The work was done in cooperation with our industrial partner—a large corporation producing physical and software-intensive products used in building infrastructure. It was further consulted with domain experts in two other institutions. This paper provides an analysis of real-world use cases from the domains of water movement and cooling technologies that was used to guide the design of the visual language. The use cases and iterative feedback from domain experts were used throughout the development of GIMLE. The paper evaluates the scalability of the language using a model for an existing application and discusses reflections of domain experts on the proposed process from a preliminary study.

BACKGROUND

Industrial products, such as water pumps or sensors, may be applied in different types of installations or use cases. Their configuration tends to depend on the use case that the products are placed in. The configuration or custom application logic may also be contributed by third-party solution providers who customize the products to niche markets.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

IoT 2017, October 22–25, 2017, Linz, Austria

© 2017 Copyright held by the owner/author(s). Publication rights licensed to ACM. ISBN 978-1-4503-5318-2/17/10... 15.00

DOI: <https://doi.org/10.1145/3131542.3131552>

Figure 1: Comparison of the involvement of end-users versus developers of applications in the related work.

The current practice in customization of industrial installations makes use of external Programmable Logic Controllers (PLC) that connect to, control and monitor products in the installations. PLCs provide a reliable and real-time operation with multiple I/O connecting to sensors and actuators or SCADA systems. However, the PLCs are an additional hardware that is a barrier for entry and increases the development and operation costs for solution developers that want to bring their applications to the market.

By connecting the sensors and actuators to the Internet, the need for external controllers has decreased. They can be customized by run-time updates and can potentially provide software ecosystems that enable the deployment of third-party application packages [24, 15]. Moreover, they may be controlled and monitored from the cloud in cases where real-time operation is not required. However, taking away the external controllers means eliminating the boundaries between different use cases. By installing an external controller in an installation, one restricts the possible use cases to those supported by the controller, which is not the case if only multi-purpose devices are used.

The related work in Industrial Internet of Things has aimed to make manufacturing systems autonomous by creating a virtual representation of them, called the digital twin [21]. Digital twins provide a semantic understanding and introspection of installations and their state. Although they have partly been applied to support configuration [10, 12], the related work hasn't enabled domain experts to build configuration models that can reason about digital twins to produce application recommendations.

COMPARISON TO RELATED WORK

This section positions the work within the related work in configuration of smart environments. It further argues for the novelty of the approach.

Roles of end-users and developers in configuration

When comparing with the related work on configuration in ubiquitous computing environments, we find it useful to compare based on its focus on building configuration models in contrast to configuration solutions [22]. Related work that builds configuration models tends to involve developers in modeling sets of correct configurations of a product and their

constraints or requirements. On the other hand, work focusing on configuration solutions tends to empower end-users in configuring a real-world instance of the product.

We use this distinction to give a broad overview of the related work as shown in Figure 1. At the one end of the scale, in many smart home cases, the end-users are empowered to connect and configure their devices and services using various approaches in end-user programming [7] or mashup editors [14]. In this case, the developers only annotate their APIs based on the targeted platforms. Similarly, in smartphone ecosystems, the end-users are in control of searching for and installing the apps while developers model some basic application requirements (e.g., type or capabilities of the smartphones) using manifest files [2]. Research in the Web of Things [16, 17] and service composition in ambient intelligence [23] has aimed to abstract the configuration of smart environments and instead let end-users to specify their targeted goals while reasoning is used to compose the available services based on their semantic descriptions provided by their developers. At the other end of the scale, we see Dynamic Software Product Lines [4] and autonomic computing [5] where developers provide complete and verifiable variability models, such as feature models [9], that adapt at run-time mostly without the involvement of end-users.

One can see that the more autonomous the platform is, the more responsibility is put on supporting the developers in building configuration models. The contribution of this paper is in building a configuration model that models requirements on the contextual information of installations. Our goal is to filter applications based on a description of the installation as an input from the end-users or installers. This relates the problem to the methods used in the Web of Things literature, however instead of using goals as an input from the end-users, we use contextual information about installations to evaluate the configuration models created by the developers.

Modeled knowledge and intelligibility

Using ontologies to model the properties and capabilities of services in IoT environments has been studied in the related work [20]. However, modeling the requirements and conditions under which they can be applied has used only limited vocabularies [9] that are not expressive enough to model domain knowledge regarding the physical properties of the installations such as the layouts of devices.

Furthermore, more expressive ontological models have used semantic rules or SPARQL queries [16] that query the models to define the requirements of services. Creating such queries requires specific programming skills that make the process inaccessible to some domain experts and may result in complex models. Visual programming has been used to make the queries more accessible [3], however the visual representation has mirrored the structure of the queries and thus didn't provide any benefits with regards to scalability and complexity of the queries.

Contributions of this work

This paper fills a gap by involving domain experts in modeling expressive requirements of software components that work

Figure 2: A representation of a simplified physical installation to illustrate the characteristics of the targeted installations.

Figure 3: A visualization of modules in cooling installations with abstracted sensors and actuators. The squares represent independent modules. The connections show the flow of liquids and shapes show different types of devices.

with domain ontologies and model the physical parts of installations in ubiquitous computing environments. It further aims to make the process accessible to non-programmers.

ANALYSIS OF TARGETED INSTALLATIONS

This section provides an analysis of the targeted types of installations. The analysis was done based on documentation and discussions with two experts in the domain of water movement at our industrial partner, one domain expert at the Slovak Association for Cooling and Air Conditioning Technology¹ and an expert in monitoring of cooling circuits at the company ClimaCheck². It focuses on the relevant aspects of the installations that guided the design of the visual language.

Differentiating features of installations

Figure 2 shows a simplified example of an installation that we target. When investigating how to differentiate this kind of installation from other related installations where the products may be applied, we identified the following factors:

Devices and their characteristics. The specific set of devices, resources and their characteristics can give an indication of the targeted use case. When looking at the characteristics of devices, information about the manufacturers, product types, sensed facets and their units as well as technical information such as the capacity of pumps or value range of sensors are relevant. In certain cases, these characteristics may be identified automatically [19].

Resources. Similarly to the taxonomy by Vanderhulst et al. [25], we distinguish between computer-augmented devices (e.g., sensors, actuators) and resources without a computing platform (e.g., water tank, manual valves). Items without a

computing platform, which we call resources, often fulfil important roles in the installation. We also use the term to refer to “dumb” devices that perform functions in the installation, but are not discoverable or controllable through software, e.g., some valves.

Roles and relations of devices. As shown in Figure 2, the same type of a device may fulfil different roles in the installation, e.g., a master or a slave pump. Roles can be set by the installers, but in some cases may also be inferred from other information about the installation, such as device layouts or networking of devices.

Networking of devices. Devices in industrial installations may be interconnected using various technologies, e.g., Modbus, ProfiBus, Bluetooth, analogue pulse connections, relays. Each provide different trade-offs in cost and function and thus they are not likely to be replaced by IoT protocols in the foreseeable future.

Layout of devices. The flow of liquids and placements of devices within the flow are useful predictors of the targeted use case. Placement of devices within the flow can also suggest their roles within the installation. For instance, a pressure sensor placed before water boosting pump performs a different role than one placed after it.

Modularity of installations

We were able to analyse larger cooling installations in the scale of supermarkets. To illustrate the structure of such installations, we give an abstracted example in Figure 3. One can observe several patterns in the installations:

Modularity of installations. The installations are separated into independent modules. Modules are independent in the sense that there is no direct communication between them and they are controlled and monitored by independent Programmable Logic Controllers. The analysed cases contained up to 30 sensors and 10 other devices per module.

Repeating sections of modules. Modules may contain multiple repetitive sections containing similar layouts and sets of devices. Similar sections can also be found across modules.

Archetypes of devices. We can also observe certain archetypes of device and resource layouts that are frequent in similar installations. For instance, a water tank containing three water level switches connected via analogue connections to pumps.

Expressing differences between installations

To guide our efforts to model the requirements of applications, we analysed how the required properties and differences between installations were described by the experts and in documentation. We note several patterns of conditions that were used to discuss the use cases.

Alternatives. A single use case can often be supported by alternative sets of devices, resources and their layouts. In cases where more alternatives were possible, it was relevant for the application to know which alternative is present.

¹<http://szchkt.org>

²<http://www.climacheck.com>

Negative conditions. We noticed the experts differentiate variations in use cases by devices or layouts that they do not contain. For instance, one use case would require a different placement of a pressure sensor if an additional device is present in the installation than if it is not.

Describing layouts. Layouts of devices within the studied domains were described in two ways: as a containment relation (e.g., sensor is placed in the main circuit) and as a relative order within directed flows of liquids with respect to other devices (e.g., sensor is placed before a compressor and after a pump). When talking about the relative order of devices within a flow, one may sometimes ignore some devices that are not relevant. For instance, for a flow with devices A, B and C, one can enquire whether A is placed before C, disregarding that B is placed between them.

Contributions

The described analysis enabled us to identify aspects of installations that have to be modeled in order to differentiate between use cases. These installation features constitute a more expressive ontology than traditional configuration ontologies, which tend to only model internal aspects and connections of products [22]. It further showed that industrial installations are modular and, in some ways, repetitive. This implies a need for reuse of modeled conditions within and across use cases. We also identified a need to support alternative and negated conditions, which is in line with other variability models [9].

GIMLE: A VISUAL LANGUAGE FOR MODELING APPLICATION REQUIREMENTS

GIMLE is a visual language for modeling application requirements using expressive domain knowledge. It enables building requirements on ontological knowledge, such as the layouts of IoT installations, that can be processed to recommend applications along with their initialization options.

First, this section argues for the underlying technologies chosen to design and implement GIMLE. Based on the chosen graphical representation, it proposes visual concepts for modeling their requirements and considerations made to facilitate modularity and reuse.

Graphical style and representation

Our goal with the introduction of a graphical abstraction over configurations was to improve the understandability of configurations and increase the efficiency of configuration development and maintenance. Graphical interfaces are common in configuration problems, however they tend to be limited to a certain vocabulary that doesn't scale to the description of the physical parts of installations.

Our analysis showed a need to support reuse of configurations across and within use cases, definition of alternatives, negative conditions and variable level of detail. These requirements suggest a need for a programming-like environment, which led us to consider visual styles used in end-user programming. Two commonly used visual styles in end-user programming of smart environments are visual data-flow [14] and block-based visual programming [6]. These styles have been applied

for creating mashups and also as a replacement for textual programming languages.

Our modeling task is different from the usual application of these styles, so we aimed to investigate whether they can be adapted for it. We made use of two frameworks that are available for the visual styles: Blockly for block-based programming and NodeRED for visual data-flow. Both styles are based on the notion of connecting blocks, the main difference lies in how the blocks are connected—in Blockly they are stacked and in NodeRED they are connected by wires. Another difference is in how the blocks are configured—in NodeRED one has to double-click a node to see its configuration, while in Blockly the configuration is always visible. Both Blockly and NodeRED come with predefined blocks for simple programming. We discarded these blocks and implemented new ones tailored for modeling the requirements on devices, networking, installation properties and layouts.

We used the help of two domain experts at our industrial partner to get iterative feedback on the use of the visual styles to intelligibly model physical properties of installations. Similarly to a study conducted by Dahl et al. [6], we found that both interfaces got less readable and cluttered with growing complexity. We saw a need to separate the details of the modeled properties of installations from the general logical flow of the requirements. This led us to consider combining the visual styles and using the visual data-flow style for building the logical flow of requirements and the block-based style for modeling the required properties of installations. This is illustrated in Figure 4, which shows a block-based environment shown upon double-clicking a single node within a visual-data flow environment.

Logical flow of conditions

The rules are structured as a chain of nodes, which each present a separate requirement. Several types of requirement nodes are provided according to the domain concepts to be modeled (e.g., device networking, layouts of devices). The requirement chains start with a "Start" node and end with a "Recommend" node. The recommendations are for applications that should be suggested to the installer. An example of a chain is shown in the upper part of Figure 5.

The chains of nodes provide a concept similar to variables in programming languages. For instance, a node that specifies a requirement for a temperature sensor will assign a name to such sensors that match its criteria. Other nodes that follow this node may then add more requirements to the temperature sensor, e.g., that it is connected to Modbus or placed in a location with some other devices. At the end of the chain, the variables will only contain such devices that have passed all the previous requirements. These devices can then be assigned as initialization parameters for application recommendations. For instance, if an application requires a pressure and a temperature sensor placed in the same room, the chain of nodes will produce initializations of the application with such sensors that match the defined requirements.

In the analysis (Section 4.3), we identified a need to describe requirements using devices or layouts that the installations

Figure 4: Block-based visual environment embedded in a visual data-flow framework. It is opened after double-clicking a node and enables configuring the details of the requirement that the node models.

Figure 5: A chain of nodes beginning with a “Start” node and resulting in an application recommendation. The chain calls a function defined below it, which requires a layout. The function is called twice with different parameters.

Figure 6: A node that provides two outputs—upper is followed if it is satisfied, lower if the devices that it requires are not matched.

Figure 7: Configuration with two chains of nodes—upper one is used to satisfy a requirement for a water inlet that is then reused in the lower chain to recommend an application.

don’t provide. For instance, for application that requires a different placement of a pressure sensor if another device is present in the installation than if it is not. To facilitate this, we added a second output to the nodes. The top output is followed in case the requirement defined in the node was satisfied, while the bottom output is followed in case it failed to be satisfied. A device node providing both a positive and a negative output is shown in Figure 6.

To model alternative requirements, one can simply connect multiple chains to the outgoing port of a node. Branched requirements are all evaluated independently, similar to an “if—if” statement in programming languages. For an “if—else” statement, one could use the negative output of nodes as discussed above.

Modeled requirements

Upon double-clicking a certain node, a pop-up with options to model the details of the requirement or recommendation using a Blockly interface is shown. The pop-up provides a palette with types of modeled features on the left, which can be dragged to the canvas on the right and configured according to the requirement. Each node may define multiple of the same kind of requirements. For instance, a single device node may require three different sensors. We found it useful to group such requirements to limit the length of the chains of nodes.

The provided options for modeling requirements reflect the installation features that we found as predictive of use cases in the analysis (Section 4.1). These include requirements on device characteristics, device roles, installation layouts and networking between devices. The analysis showed the need to model installation layouts using containment relations and by describing the order of devices in flows. We were able to provide this functionality by embedding blocks within each other and connecting them in ordered chains. An example is shown in Figure 4.

The concrete set of options for modeling installation requirements are domain dependent. The blocks are populated using an external domain ontology which defines the supported device types, types of locations and more. While our targeted domains led us to provide the mentioned filters, we argue that the same interaction could be provided for other ontologies as well. There are numerous ontologies that describe various aspects of IoT installations [1]. Adding support for these ontologies to GIMLE is only a matter of defining the structure of the blocks and binding them to the classes and attributes in the ontologies. In our future work, we aim to generalize GIMLE and extend it to other domains.

Support for modularity and reuse

Section 4.2 discussed the modularity of bigger installations. We found that similar sets and layouts of devices are often repeated within and across modules. To prevent repetitive chains of conditions and support reuse of conditions between use cases, we introduced several concepts in the language.

The NodeRED environment enables reuse of chains of nodes within the visual language using subflows. However, subflows can’t receive parameters, which we found necessary in order

to adjust for slight differences in how the reusable chains of nodes fit within the installation. For instance, when extracting conditions describing a layout of devices, we needed to choose which circuit the devices are placed in.

To support reuse with parametrization, we enabled defining functions. Functions serve the same purpose as subflows, however they may receive parameters that can be used within them. Use of a function is demonstrated in Figure 5.

For some applications, expressing all their requirements in a single chain of filter nodes resulted in long sequences of rules. Using functions helped in extracting parts of the chains, however the functions were only visually hiding the long chains of rules.

We found it useful to separate the requirements of applications into partial requirements that can each be satisfied by one or more independent chains of rules. Chains of partial requirements result in a “Satisfy” node where a certain requirement can be satisfied with some arguments, e.g., textual values, devices or locations. They can then be required using a “Require” node in the main chain before recommending an application. Partial requirements proved useful for inspecting problems since one could look at which of them were satisfied and with which arguments. Furthermore, they can be used to create requirements for commonly occurring archetypes of device layouts as suggested in Section 4.2. An example of a partial requirement is shown in Figure 7.

Contributions

This section discussed the visual language used to model requirements of applications. By combining two visual styles, we separated the logical sequence of requirements from their configuration. This enabled a more expressive configuration model than traditional graphical representations provide [13]. By providing functions and partial requirements, we enabled reuse within the configuration that reflects the repetitive nature of industrial installations.

The visual models are translated into semantic rules in the Notation3 language and processed using semantic reasoning. This process is motivated in our previous work [24].

FEEDBACK ON THE PROCESS FROM DOMAIN EXPERTS

Since GIMLE builds on existing visual programming styles, evaluation of their usability is beyond our scope and is provided by the related work [6]. However, this paper presents two novel contributions that we aimed to evaluate: a method to model requirements on physical properties of installations and a process that changes the current practice of configuring industrial installations.

We use a real-world use case to evaluate modeling physical properties of installations and discuss it in Section 7. This section aims to involve domain experts in configuring a physical installation and use their experience to reflect on the proposed process.

Structure of the sessions

We involved domain experts at our industrial partner with experience designing and configuring the targeted systems.

The participants weren’t involved during the design of the system and were first introduced to the system at the sessions. Due to this focus and the busy schedules of such experts, only three employees at our industrial partner were able to participate. Despite the low number of participants, their experience enabled insightful reflections that would be difficult to obtain without this target group.

To give them a concrete understanding of the proposed process, we built a demo physical setup and involved them in the whole process from building requirements of applications to setting up the installation. It contained the following devices from different manufacturers: 1 gateway, 3 pumps, 2 remote controllers, 2 flow sensors, 3 temperature sensors, 2 water level sensors. The pumps and temperature sensors were connected over ModBus to the gateway. The gateway was used as a proxy to enable accessing these devices from the cloud. The flow and level sensors and remotes were connected to the Internet over WiFi and exchanged messages over the MQTT protocol. The installation was controlled from the cloud.

Participants with the following roles were involved: an application manager (participant A), application engineer (participant B) and a product manager (participant C). The sessions took place in a room where the setup was installed. At the start of each session, the participants were introduced to the system and the demo setup. They were guided through building a configuration for one part of the setup (using 4 devices) and commissioning it. The participants were then asked to create a configuration using 8 devices. They all managed to build the configuration with little help in 13, 20 and 18 minutes. The configuration was then used to initialize the targeted devices.

Afterwards, they were shown an existing configuration for a real-world application. They were asked to explore the configuration and describe/draw a diagram of an installation that the configuration targets. They were asked to think out loud and allowed to ask questions. The participants successfully completed this task in 10, 19 and 11 minutes.

Additional 45–60 minutes were used for a semi-structured interview, where the participants reflected on the overall process. The participants were led to compare the process to the current practice and reflect on the challenges in applying the proposed approach. Their qualitative feedback was collected in semi-structured interviews.

Qualitative findings

Although the domain experts remarked that it requires training for one to fully understand the language, they were positive about being able to learn and make use of the visual language. Participant C compared the language to other notations: “It’s not more difficult than going into any other language or tool that you need to learn. For me it looks quite similar to some PLC programming that I do.” Participant B said the visual language was fun to work with. They also appreciated the potential of automating parts of the configuration that are currently performed by the installer. They observed that it can save significant time for the installers which is currently spent on configuring such systems.

Participants A and B remarked that the process of modeling the characteristics of installations requires a different mindset. Upon analysis of the comments and questions, we identified the following ways in which the new process confused them at first:

1. In the current practice, each product in the installation is configured separately at the installation site by connecting it with its inputs (e.g., sensors), choosing the control algorithm (e.g., PID control) and setting its parameters (e.g., various set-points). On the other hand, we are aiming to create a model that can be used to reason about any installation. This changes when the configuration model is created—instead of at the installation site, it is developed in advance. It also requires the use of logical operations (alternatives, negations) that are not necessary when dealing with a configuration for a single installation.
2. The current practice doesn't involve a process where a formal model of the installation is created. Thus, the experts are not used to modeling such properties of installations as the layouts of devices other than in documentation.
3. We introduced a concept of applications that abstracts the particular control algorithms and set-points for individual products. This separates the details of the configuration from the installation model. Being not used to such abstraction, the experts were missing the details of the configuration. However, one could also adapt the proposed approach to deal with control algorithms and their parameters instead of applications.

Participant B raised a concern related to how installations not covered in the configuration would be handled. Since the configuration aims to model installations by their specific features, it is to be expected that there will be special cases that were not considered in the configuration. Thus, it may be necessary to also provide a manual initialization of applications using manually entered parameters. Although the configuration model should cover the majority of cases, for uncovered cases, the installer would have to manually choose the correct application and enter the initialization parameters.

Although we address the problem of scalability of the visual language to larger applications (see Section 7), we haven't investigated how the visual language scales as the number of applications increases. This was brought up participant C: "It is quite easy now that you have two (applications), but what if you had ten?". The issue the participant had in mind was differentiating between the applications when a new user is exploring the configuration. The participant related the concern to a different problem he faced with a graphical model of a system, where by adding an additional layer of abstraction the model became more accessible to newcomers.

Contributions

The participants were able to make use of the visual language in a relatively short time and were positive about its goal to automate some of the actions usually performed by installers. We identified several ways in which the proposed approach required the experts to adapt their "mindset" from the current

Figure 8: Configuration model of the ClimaCheck solution.

Figure 9: Number of requirement nodes (Y-axis) based on how many templates of ClimaCheck are supported (X-axis).

practice. The current practice doesn't involve a process for creating an abstract model of installations, which requires configuring each installation independently. Finally, the experts raised concerns regarding supporting special cases and the need for a higher-level overview of the configuration.

SCALING TO LARGER CONFIGURATIONS

This section discusses the scalability of modeling physical properties of installations using GIMLE. A model implemented based on the specification of a real-world application, ClimaCheck³, is used to discuss the scalability of the approach. The model was discussed in two sessions with the product owner of ClimaCheck and adapted based on their feedback. The discussion enabled us to confirm the relevancy of the model, fix misunderstandings in the configuration and extend it to cover more cases.

ClimaCheck enables analyses and evaluation of performance of cooling systems. It uses various kinds of sensors placed throughout cooling circuits. It is crucial for the application to know the type of the monitored circuit and roles of each of the sensors it receives data from. It provides 25 standard templates for the placement of sensors in circuits. The simplest template contains 9 sensors and 4 other devices in a single circuit, while the most complex one has 30 sensors and 10 other devices placed in two circuits.

³<http://climaccheck.com>

Currently, ClimaCheck is distributed with an external controller that connects to the sensors in the installation. When setting up ClimaCheck at an installation site, one has to choose from the available installation templates. Upon choosing the template, each sensor in the template has to be manually bound to a sensor in the installation.

Given information about the installations, the GIMLE model for ClimaCheck enables operation without an external controller and an automated choice and initialization of the installation templates. It can assess whether any given circuit is supported by the application. Furthermore, it classifies the circuit based on the templates and can produce recommendations for the initialization of the application.

Structure of the model

The model was structured into five parts with increasing level of detail. The first part provided the highest-level view of the model, while the latter ones delved into requirements within smaller and smaller sub-sections of the installations. As shown in Figure 8, the model was structured into the following parts:

- A Start of the configuration, where we decide if the current installation provides one or two circuits. Subsequently, we check that the circuits are fully supported by the application using requirements that are satisfied in the rest of the model.
- B For each circuit, this part tests which devices are present or not. Based on this, it can decide the type of the circuit and call functions in part C and D that identify the sensors within each of its parts.
- C This part contains functions placing requirements on sections within one half of the cooling circuit. For each section, it requires that it provides the necessary sensors. The requirements are satisfied in section E.
- D This part is similar to part C, however deals with the other half of the circuit.
- E Finally, this part locates sensors in the sub-sections of the cooling circuit and assigns them their roles.

This structure of increasing level of detail enabled reuse of requirements across the layers. Each function in section E that locates the concrete sensors is called on average 4.33 times from section C and D. This minimizes the number of times that the properties of each sensor are described in section E despite the varying placement of sensors.

Analysis of the model

The configuration contained a total of 138 nodes in all the chains of rules. There were 21 functions, which were called from 43 places in the configuration. There were 10 types of requirements, which were required in 15 places and could be satisfied in 21 different ways. There were 99 positive and 12 negative links between nodes.

We were able to keep the length of chains relatively short thanks to the use of functions and requirements. The median length of a chain was 3 nodes, while the longest one had 5 nodes.

The high number of functions and requirements enabled good reuse of definitions in the configuration. It is shown by the fact that only 38 device definitions were required even though the 25 templates each contained between 13 and 40 devices.

We further investigated how does the size of the model grow as we increase the number of supported ClimaCheck templates. Figure 9 shows a chart where the X-axis shows the number of supported templates and the Y-axis shows the size of the model. One can see that the model increases on a logarithmic-like scale, where as more templates are supported, more requirements can be reused among them. Templates 0–15 contain a single circuit, whereas templates 16–26 contain two circuits. Since two-circuit templates contain the same sections as single-circuit templates, supporting more circuits doesn't significantly change the model. The use of parametrized functions and requirements made adding support for new templates possible without changing the existing parts in the configuration.

Contributions

The exercise showed that the model is expressive enough to describe the templates provided by ClimaCheck and differentiate between them. Using the concepts such as parametrized functions and partial requirements, we were able to structure the model in layers with increasing level of detail. This enabled tackling the repetitive nature of the installation templates. We showed the model grows slower the more templates are supported and the more requirements can be reused among them. Furthermore, adding new templates doesn't significantly change the models.

CONCLUSION

Industrial products, such as pumps or sensors, may be applied in different types of installations in which they are configured in different ways. Currently, external controllers are provided by third-parties to customize the products. Unless support for modeling targeted installations is improved, customization without the use of external controllers will result in more complex configuration of the installations. This paper fills the gap of modeling requirements on physical properties of IoT installations and proposes a novel process for configuring industrial installations.

GIMLE is a visual language that provides the expressiveness and scalability to model the installations. It makes use of two visual programming styles in order to separate the logical flow of the requirements from their details. Due to the repetitive nature of industrial installations, support for reuse within the models is a crucial part of the language. We were able to evaluate expressiveness and scalability of GIMLE using real-world use cases. Reflection with domain experts showed a need for the process and uncovered ways in which to clarify how it fits within the current practice and prepare it for use in production settings.

Acknowledgment

This work has been supported by IFD grant no. 3067-00001B for the project entitled: MADE—A platform for future production.

REFERENCES

1. B. Balaji, A. Bhattacharya, G. Fierro, J. Gao, J. Gluck, D. Hong, A. Johansen, J. Koh, J. Ploennigs, Y. Agarwal, M. Berges, D. Culler, R. Gupta, M. Kjærgaard, M. Srivastava, and K. Whitehouse. 2016. Brick: Towards a Unified Metadata Schema For Buildings. In *Proc. of the 3rd ACM Int'l Conf. on Systems for Energy-Efficient Built Environments (BuildSys '16)*. ACM, 41–50.
2. T. Berger, R. Pfeiffer, R. Tartler, S. Dienst, K. Czarnecki, A. Wasowski, and S. She. 2014. Variability Mechanisms in Software Ecosystems. *Information and Software Technology* 56, 11 (2014), 1520–1535.
3. P. Bottoni and M. Ceriani. 2015. Using blocks to get more blocks: Exploring linked data through integration of queries and result sets in block programming. In *2015 IEEE Blocks and Beyond Workshop (Blocks and Beyond)*. 99–101.
4. R. Capilla, J. Bosch, P. Trinidad, A. Ruiz-Cortés, and M. Hinchey. 2014. An Overview of Dynamic Software Product Line Architectures and Techniques: Observations from Research and Industry. *J. Syst. Softw.* 91 (May 2014), 3–23.
5. C. Cetina, P. Giner, J. Fons, and V. Pelechano. 2009. Autonomic Computing through Reuse of Variability Models at Runtime: The Case of Smart Homes. *Computer* (2009), 37–43.
6. Y. Dahl and R. Svendsen. 2011. *End-User Composition Interfaces for Smart Environments: A Preliminary Study of Usability Factors*. Springer Berlin Heidelberg, 118–127.
7. Z. Drey, J. Mercadal, and C. Consel. 2009. A Taxonomy-Driven Approach to Visually Prototyping Pervasive Computing Applications. In *1st IFIP Working Conf. on Domain-Specific Languages*, Vol. 5658. 78–99.
8. U. Eklund and J. Bosch. 2014. Architecture for embedded open software ecosystems. *Journal of Systems and Software* 92 (2014), 128 – 142.
9. P. Giner, C. Cetina, J. Fons, and V. Pelechano. 2009. *A Framework for the Reconfiguration of Ubicomp Systems*. Springer Berlin Heidelberg, 1–10.
10. C. Greenhalgh, K. Glover, J. Humble, J. Robinson, S. Wilson, J. Frey, K. Page, and D. De Roure. 2008. Combining System Introspection with User-Provided Description to Support Configuration and Understanding of Pervasive systems. In *2008 Third Int'l Conf. on Pervasive Computing and Applications*, Vol. 2. 673–679.
11. V. Heun, J. Hobin, and P. Maes. 2013. Reality Editor: Programming Smarter Objects. In *Proc. of the 2013 ACM Conf. on Pervasive and Ubiquitous Computing Adjunct Publication (UbiComp '13 Adjunct)*. ACM, 307–310.
12. P. Hirmer, U. Breitenbücher, A. C. F. da Silva, K. Képes, B. Mitschang, and M. Wieland. 2016. Automating the Provisioning and Configuration of Devices in the Internet of Things. *CSIMQ* 9 (2016), 28–43.
13. L. Hotz, A. Felfernig, M. Stumptner, A. Ryabokon, C. Bagley, and K. Wolter. 2014. Configuration Knowledge Representation and Reasoning. In *Knowledge-Based Configuration*. Elsevier, 41–72.
14. R. Kleinfeld, S. Steglich, L. Radziwonowicz, and C. Doukas. 2014. Glue.Things: A Mashup Platform for Wiring the Internet of Things with the Internet of Services. In *Proc. of the 5th Int'l Workshop on Web of Things (WoT '14)*. ACM, 16–21.
15. Fei Li, M. Vogler, M. Claessens, and S. Dustdar. 2013. Towards Automated IoT Application Deployment by a Cloud-Based Approach. In *Service-Oriented Computing and Applications, 2013 IEEE 6th Int'l Conf. on*. 61–68.
16. S. Mayer, N. Inhelder, R. Verborgh, R. Van de Walle, and F. Mattern. 2014. Configuration of smart environments made simple. In *Internet of Things (IOT), 2014 Int'l Conf. on the*. 61–66.
17. S. Mayer, D. Plangger, F. Michahelles, and S. Rothfuss. 2016. UberManufacturing: A Goal-Driven Collaborative Industrial Manufacturing Marketplace. In *Proc. of the 6th Int'l Conf. on the Internet of Things (IoT'16)*. ACM, 111–119.
18. F. Michahelles and S. Mayer. 2015. Toward a Web of Systems. *XRDS* 22, 2 (Dec. 2015), 62–67.
19. R. Minerva, A. Biru, and D. Rotondi. 2015. Towards a definition of the Internet of Things (IoT). *IEEE Internet Initiative* 1 (2015).
20. C. Perera, A. B. Zaslavsky, M. Compton, P. Christen, and D. Georgakopoulos. 2013. Semantic-driven Configuration of Internet of Things Middleware. *CoRR* abs/1309.1515 (2013).
21. R. Rosen, G. von Wichert, G. Lo, and K. D. Bettenhausen. 2015. About The Importance of Autonomy and Digital Twins for the Future of Manufacturing. *IFAC-PapersOnLine* 48, 3 (2015), 567 – 572.
22. T. Soininen, J. Tiihonen, T. Männistö, and R. Sulonen. 1998. Towards a General Ontology of Configuration. *Artif. Intell. Eng. Des. Anal. Manuf.* 12, 4 (Sept. 1998), 357–372.
23. T.G. Stavropoulos, D. Vrakas, and I. Vlahavas. 2013. A survey of service composition in ambient intelligence environments. *Artificial Intelligence Review* 40, 3 (2013), 247–270.
24. M. Tomlein and K. Grønbaek. 2016. Semantic Model of Variability and Capabilities of IoT Applications for Embedded Software Ecosystems. In *Software Architecture (WICSA), 2016 IEEE/IFIP Conf. on*.
25. G. Vanderhulst, K. Luyten, and K. Coninx. 2010. Pervasive maps: Explore and interact with pervasive environments. In *Pervasive Computing and Communications (PerCom), 2010 IEEE Int'l Conf. on*. 227–234.
26. K. Villela, A. Silva, T. Vale, and E. S. de Almeida. 2014. A Survey on Software Variability Management Approaches. In *Proc. of the 18th Int'l Software Product Line Conf. - Volume 1 (SPLC '14)*. ACM, 147–156.