Interactive System Modelling for the Internet of Things

JUDY BOWEN, The University of Waikato, New Zealand
JESSICA TURNER, The University of Waikato, New Zealand

The rapid growth in the number of Internet of Things (IoT) systems and their increasing use in safety-critical domains has led to the need for an evolution of development methods. Model-Driven Development (MDD) approaches have been used in software engineering to reduce development time and minimise errors in implemented systems. This paper introduces a novel Model-Driven Development (MDD) approach for Internet of Things (IoT) systems which considers interactivity from the perspective of both users and IoT components. A real-world example is presented to explain the motivation for the work and demonstrate the benefits and use of lightweight interactive system models adapted for the design and development of IoT systems.

CCS Concepts: • **Software and its engineering → Model checking**; **Model-driven software engineering**.

Additional Key Words and Phrases: IoT, Interaction Design

1 INTRODUCTION

The continued evolution and maturing of internet connected devices contributing to the Internet of Things (IoT), along with rapid expansion of proposed use-cases has led to an increasing number of IoT systems being used in a variety of ways. Worldwide, the number of IoT connected devices is projected to increase to 43 billion by 2023, an almost threefold increase from 2018 [22]. From large-scale smart-cities and environments [30], such as connected factories [20], smart-homes [4] etc. to individual home appliances and personal items, new solutions and systems are entering the market at a rapid rate. The domains of IoT systems are similarly diverse, but increasingly include areas which can be considered safety-critical, such as within medical settings [3]. Design and development methods for IoT systems, therefore, must similarly evolve to keep up with rapidly-changing technologies and use-cases.

IoT systems are made up of heterogeneous components and typically involve both streaming and static data. The implementation of these systems will often make use of different programming languages to deal with hardware and low-level communication requirements, cloud services and web or mobile applications. One of the many challenges of developing large-scale IoT systems is the co-ordination of the different parts (both hardware and software), along with ensuring the data-flow and functionality is correctly supported. While the focus of much of the IoT research community is on these hard technical challenges, there is another aspect which is equally important, and that is of the user and use. IoT systems may require complex set-up and configuration in order to work effectively, so ensuring the user experience aspects of this are fully considered is crucial. While we may expect users of 'standard' interactive systems to have a consistent interface for interactions (albeit this may switch between different modalities such as mobile, desktop etc.) in IoT systems they may be interacting not only with traditional web and mobile interfaces, but also via different interaction modalities (haptic, auditory, voice etc.) through increasingly diverse hardware.

Model-Driven Development (MDD) approaches are used in software engineering to assist developers in creating applications. MDD uses models of a system which adhere to the system requirements that can be used in a number of different ways, from verifying behaviour to generating code. The primary benefit of MDD is that it reduces time costs in the development process as a result of fewer errors in the implemented system. Depending on the models used in the process, verification and validation (v&v) techniques may also be incorporated to ensure that systems meet their requirements, incorporate safety properties and behave as expected. With regards to IoT

Authors' addresses: Judy Bowen, The University of Waikato, New Zealand, jbowen@waikato.ac.nz; Jessica Turner, The University of Waikato, New Zealand, jessica.turner@waikato.ac.nz.

applications, MDD techniques are used to help address the many complex and diverse challenges that designing and implementing these systems brings.

In this paper we present a MDD approach for IoT systems which focuses on their interactive aspects and their use as interactive systems. The usual HCI considerations of system development are extended due to the heterogeneous nature of IoT. Novel interaction methods, such as indirect inputs and outputs from wearable technology, supplement direct interactions from users, while multiple interfaces provide access to different parts of the system. Mobile devices and web applications, which provide data visualisations or real-time information, allow users to interact in traditional ways, while internet-connected smart devices provide functionality which produces tangible feedback or impacts the users' environment directly. Rather than considering the user as someone who acts upon the system, the users are rather an integrated part of an IoT system, which may also act upon them and the environment they inhabit. This adds complexity to the development of IoT systems and on understanding the human considerations of use. However, it also allows for the abstraction of all parts of the system (including users) from the perspective of their interactive capabilities - what inputs can they provide, what outputs can they process. This abstraction enables a model-driven approach which focuses on interactivity and interactions of the system to ensure all parts can correctly contribute to the total required behaviour of the system.

We start by identifying common challenges for IoT and outline how models and MDD have been applied to these. We then introduce a real-world example to show how the use of interactive system models can be used to support the design and development process for IoT. Following the example we discuss some of the implications of the work presented and end with concluding remarks.

2 RELATED WORK

The use of modelling for the design and development of IoT systems is not a new concept and there are a number of different approaches in use. These approaches typically follow a model-driven development (MDD) structure in which the models are used to describe the requirements of a system. The models are then used to generate code and/or tests for the system to ensure it works as expected. In particular, communication between system components and interactivity of the 'things' must be managed in addition to end user interactions. This presents unique challenges when compared to traditional MDD which relies on an expected hardware and standard protocols. In this section we describe MDD approaches which focus on engineering IoT systems and their interactivity.

2.1 Closed Environments

In MDD for IoT applications closed environments are popular as they making code generation easier. There are different ways to achieve this but this usually includes using a pre-defined set of sensors and actuators running on, or being processed by, a common operating system. For example, ContikiOS[1] is one of many operating systems developed for IoT devices. Asici *et al.* describe a ContikiOS-based model-driven engineering approach for RaspberryPi[2] devices [8]. This approach uses extended meta-models to describe the relations between components of the system, which defines the interactivity between those different components. The models are translated into Petri Nets which allows them to use the models for code generation and verification.

In [25] cMoflon, a tool that generates embedded C code for ContikiOS wireless sensor networks, is proposed. The focus is explicitly on communication between sensors using topologies (a model usually used for describing networks) and state charts which are described using story-driven

[1]See https://www.contiki-ng.org/ (last accessed April 2023)
[2]See https://www.raspberrypi.org/ (last accessed April 2023)

modelling [32]. Similarly, Ammar *et al.* describe a Systems Modelling Language (SysML) approach for developing IoT applications using ContikiOS [6]. SysML is a semi-formal modelling language which describes a system's framework and architecture; it was created as an extension of the Unified Modelling Language (UML). Ammar *et al.* use SysML to simulate wireless sensor networks. Instead of code generation, these are translated to state charts for v&v of communication between sensors.

While working in closed environments assists in simplifying MDD approaches for IoT systems, modelling techniques used for closed environments are only applicable to the devices within that closed environment. The examples given above are only applicable to devices running ContikiOS software and do not extend to other operating systems (such as TinyOS[3] or Windows IoT[4]). Furthermore, the use of closed environments reduces flexibility of approaches as it limits developers to a specific set of devices and programming languages.

2.2 Open Environments

In contrast to closed environments, an open environment describes IoT systems which do not adhere to one type of hardware, protocol or programming language. This introduces new challenges as systems must be designed with flexibility of hardware in mind. We cannot guarantee which hardware components will make up a system, thus code generation becomes more complex. For example, Agrawal *et al.* demonstrate using Raspberry Pi and Arudino technology to build a smart drip irrigation system [2]. The project describes the challenges of connecting these devices, demonstrating the low-level hardware considerations developers must make when working in open environments.

In order to manage interoperability between devices, MDD approaches for IoT systems in open environments describe models at a different level of abstraction by defining models based on their software architecture. For example, Domain-Specific Modelling Languages (DSML) are often used in architecture-focused approaches. They enable descriptions of the software architecture of an application which enables reasoning about the different parts and communication needs of the system. Existing literature shows a common approach used is to extend UML when defining a new DSML for IoT. Examples include: The Interaction Flow Modelling Language (IFML)[5] for modelling the front-end and behaviours of systems; ThingML[6] for the design and implementation of distributed systems; and SysML[7] for describing frameworks and architecture. We describe each of these DSMLs next and give examples of their use with IoT.

IFML builds on UML, which allows developers to make use of content and process models, sequence diagrams etc. Unique to IFML is the ability to describe user interactions, these interactions follow an event-based approach for describing the tasks an end-user may wish to carry out using a system. Using IFML, Brambilla *et al.* describe a MDD approach for creating user interfaces of IoT applications [16]. User interactions are described as events depending on the actions the user may take, for example, a "Press" or "Long Press" on a touch screen. Similarly, they use an extension of IFML which allows them to describe events between IoT devices (e.g. request or push data), thus creating a single process for describing the IoT system and user interface. This allows the use of established design patterns to ensure best practice for the user interface and data synchronisation of the systems created.

[3]See http://www.tinyos.net/ (last accessed April 2023)
[4]See https://developer.microsoft.com/en-us/windows/iot/ (last accessed April 2023)
[5]See https://www.ifml.org/ (last accessed April 2023)
[6]See https://github.com/TelluIoT/ThingML (last accessed April 2023)
[7]See https://sysml.org/ (last accessed April 2023)

SysML is a semi-formal modelling language created as an extension of UML, which allows developers to describe system frameworks and architecture. Kotronis *et al.* use SysML as the basis of a model driven approach which focuses on the criticality requirements of smart health systems [26]. Using SysML they are able to define the critical properties of the system (such as safety critical or mission critical) and then use this to update the system design and ensure required properties are included.

There are also several techniques which use formal modelling methods as the basis for a DSML. In [37] Finite State Machines (FSM) are used to describe how components of the IoT application interact to form the basis of a model-driven development technique. The aim in using FSMs is to allow for rapid prototyping of different components in the overall system. In contrast, in [5] the authors describe a v&v technique using state charts. Their focus is on safety properties of the IoT application, building on techniques previously established in the embedded systems area. Each of these techniques demonstrate ways in which to verify and validate IoT applications.

IoT applications handle large quantities of data, some of which can be sensitive in nature (e.g. personal data of end-users). As a result, data and privacy concerns represent a significant challenge in IoT development, especially when we consider interactivity, such as how users input data and what is output, and to who. In [18] the authors describe a framework which allows data to be managed as a service. They use a meta-modelling approach which defines how data can be accessed and modified depending on stakeholder type. For instance, in healthcare settings, patients are treated differently to healthcare professionals as they have differing requirements. A healthcare professional may need to access multiple patient information while a patient should only be able to access their own information. Their approach allows for data analysis to ensure the correct relationships are present in the system, such that data is communicated correctly between system components and end users and access control for different types of users can be handled appropriately.

Ardito *et al.* describe an approach for building Smart Interactive Experiences (SIEs), examples include interactive museum or educational experiences, using IoT devices [7]. Their approach incorporates a user-defined semantics, working within an open environment, to allow domain experts to configure IoT devices accordingly. The tangible approach they take allows the subject domain experts to describe the interactivity their system needs, assisting developers to understand the end users' goals. While they do not take a MDD approach, their semantics are useful in understanding the interactivity required between the IoT application and end users.

2.3 Summary

In this section we have explored MDD for IoT in relation to interactivity between system components and end users. We have highlighted the differences between open and closed environments. In particular, closed environments make code generation easier as we can work with an expected set of components, meanwhile, open environments take a software architecture approach as the hardware is not guaranteed to be using the same language or operating system. The majority of the approaches above describe MDD approaches which focus on interactivity between components or end users but not both. In the next section we introduce an example which highlights the aspects needed for modelling an IoT system as an interactive system. We then describe our chosen approach and demonstrate it by way of the example.

3 EXAMPLE

Here we introduce an example, based on a real-world project, which uses wearable technology to monitor workers for safety purposes. The system consists of several different components and applications and was designed in conjunction with workers and stakeholders, following a

user-centred design process. The user-centred nature of the development and the focus of the initial project requirements meant that the design-team primarily considered the system under development as an interactive system (albeit with multiple heterogeneous components) rather than an IoT system. This influenced the choice of modelling solutions, as discussed later. The system was developed and implemented by six teams working remotely from each other. Each of the teams worked with a sub-section of the requirements to focus on their part of the development, but also needed to ensure integration and consistency of all interactive parts of the system and interactions between the components shown in figure 1. This provided the motivation for adopting a model-based approach, to ensure the teams could work independently but with a well-defined description of the full solution. It also dictated the types of models chosen as the focus was on ensuring that the teams could manage the interactions correctly. We use this example to show how our modelling process can guide the separate development teams and also provide a high level view of the whole system which allows for validation that the component parts and their users can achieve the required goals (given in the initial requirements).

The components of this IoT project are shown in figure 1.

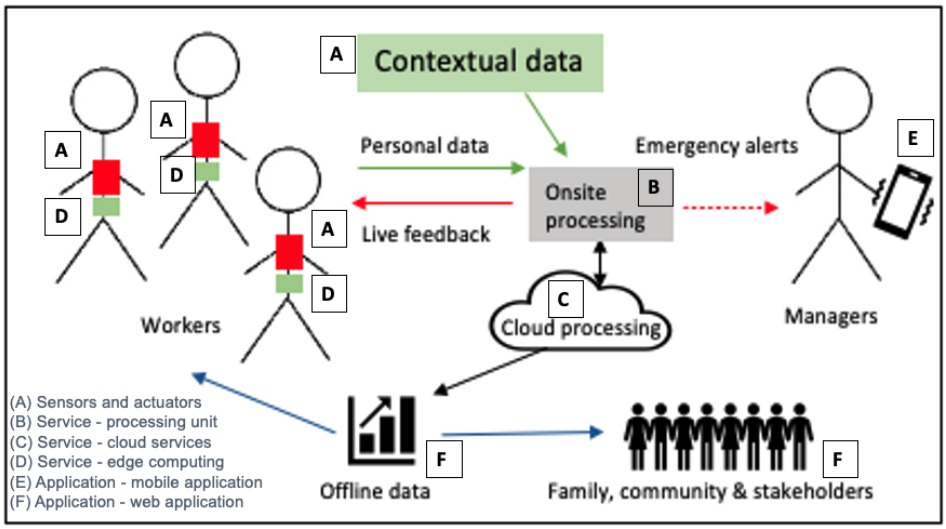

Fig. 1. Components of the Worker Monitoring Project

Each of the labelled components in figure 1 was developed by a separate team (6 teams A - F) alongside a seventh team working solely on the streaming data analysis methods (team G). For each of the component parts the teams needed to manage the data-flow, shown in the logical data-flow diagram in Figure 2 and consider their integration into the full system. Each team started with a high-level understanding of the requirements of their part of the system development, which included the data-flow diagram and the component descriptions given next.

Sensors: Capture data from the workers and the environment and provide the data to any services that process the data, which for this example are the edge computing service and the processing unit service. For each type of data that is collected there should be a corresponding behaviour to send the data to the respective services. The sensor data consists of biometric data collected from the workers (id, heart-rate, galvanic skin response, activity, location) and environmental data (ambient temperature, humidity, location).

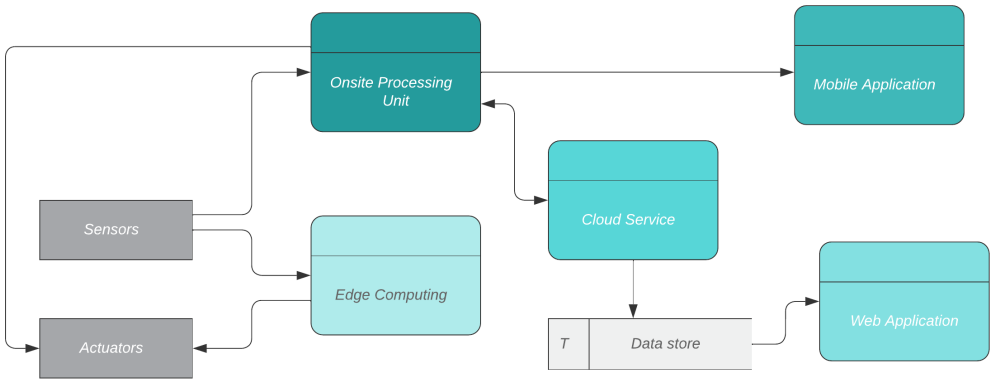

Fig. 2. Logical DFD for the Worker Monitoring Application

Actuators: Provide outputs based on triggers received from the edge computing service and the processing unit service. These include lights and vibrations on the workers' wearable technology.
Processing Unit Service: The processing unit collects data that is streamed from the sensors. It performs functions to check the data, generate any immediate feedback to the edge computing devices, actuators and mobile application, as well as sending all of the streamed data to the cloud processing service.
Cloud Service: The cloud service receives data from the onsite processing unit service. It performs functions to generate graphs and visualisations from the data over time, which is stored and used by the web application.
Edge Computing Service: The edge computing service performs real-time checking of the streaming data from the sensors, sends and receives data to/from the processing unit and provides real-time alerts to the actuators.
Mobile Application: The mobile application provides emergency data about the workers to inform supervisors and managers when intervention is required. It receives data from the processing unit service.
Web Application: The web application provides data visualisation for the workers, families, community and stakeholders to show health and alert status over time. It receives data from the cloud processing service.

The goal of the modelling process we describe was to enable the developer teams to ensure that the system being built met the requirements and provided correct functionality. While this is challenging for any interactive system, for IoT systems the number of heterogeneous components and sub-systems adds to the complexity. For each of the components of our example, shown in Figure 1 we needed to ensure that they would behave and interact as required so that the total system met the requirements. While we typically consider interactions between human users and system interfaces, in IoT solutions which incorporate wearable technology we also want to consider indirect interactions. As such, the data produced from the wearable technology and the outputs to the wearable actuators can be considered in a similar manner as direct user input/output. This led to the requirements for the modelling approach, which was adopted by each of the teams (and the project coordinator) such that the following could be ensured:

(1) There are sensors for all required data.

(2) There are actuators for all required responses/outputs.
(3) The combination of all services allows for required handling of data.
(4) The data processing methods correctly analyse the data.
(5) The application provides the required functionality for the users.

These properties can be considered at different levels of abstraction, particularly with respect to the data. For example, the worker and environmental data is streamed at different time intervals, but for simplicity of the initial modelling time-related considerations were abstracted away and it was assumed that transfer of data occurs as required. As the system was implemented and refined, these lower-level requirements could then be addressed . In our descriptions in the rest of the paper we will initially consider the functional behaviours of the system and how these interact with data (collect, transmit, process, respond to, etc. as shown in Figure 2) rather than the detail of data types and transmission protocols. We discuss this further in Section 6.

As we have discussed in Section 2 there are many different approaches that can be taken when modelling IoT systems. Rather than create yet another modelling language or notation we instead took a more pragmatic approach based on our consideration of IoT systems as a specialised subset of interactive systems. This allows us to take an existing modelling approach used for interactive systems and extend it to be useful for IoT systems. While this has similarities to the DSML approaches described in Section 2, it takes advantage of the fact that web and mobile applications, along with cloud-based, fog and edge data-processing systems form a large part of IoT systems. The models we propose are designed and well-suited to ensuring the correctness of functionality and interactivity of such applications and can be easily extended to consider additional components such as sensors and actuators. In the next section we will demonstrate how the existing modelling approach can be extended to support the additional requirements related to the IoT components in order to satisfy the properties above.

4 MODELLING PROCESS

We build on an existing approach used for modelling interactive systems, which is the Presentation Model process [12]. Presentation Models (PModels) consist of four parts. The PModel describes interactive components (widgets) of an application; the Presentation Interaction Model (PIM) describes the availability of the behaviours of the application and gives formal semantics to the interactive behaviours; the Presentation Model Relation (PMR) gives the formal semantics of the functional behaviours by mapping them to operations in a formal specification; finally, a formal specification describes the state and behaviours of the system. The four models can be used independently or combined for a variety of purposes, including test-generation, model-checking of safety properties, v&v etc. (see [15] for full details of this). PModels were created to describe interactive systems in general, we specialise this for IoT to address the properties identified in the previous section. While it is outside of the scope of this paper to repeat a full description of the PModel process, we provide enough detail in what follows to explain how we extend the process and how we use it for our purpose of reasoning about IoT systems.

In order to ensure the IoT systems meet all requirements, are consistent and complete (i.e. meet the four criteria defined above), we will use the models to identify the following:

- The sensors capture the required data:

(1) Use the PModels to ensure we have the necessary sensors in the system for each type of data.

- The actuators respond to the required data:

(2) Use the PModels to ensure we have the necessary actuators in the system for each type of data.

- The services correctly analyse the data:
  (3) Use the PMR to find the specified operations related to sensor and actuator behaviours.
  (4) Ensure the specified operations match data input/outputs to related sensor/actuator behaviours.
  (5) Ensure the operations in the specification describe correct behaviour.
- The application provides the required functionality for the users:
  (6) Confirm from the previous 5 steps that the required data inputs and outputs are available for the user and the functions and services needed to provide functionality exist.

This is not the full process, we also need to test the UI and designs to ensure usability of the system required for users to access and comprehend the data outputs. A description of some of these activities for the example system can be found in [9, 19], but in this work we focus on the modelling and use of models only.

We begin with a high-level description of the presentation model process and how it is used for interactive systems. We then show how we can extend the approach in order to address the points listed above.

4.1 Presentation Models for Interactive Systems

The presentation model process in its existing form enables us to describe any interactive system [12]. We use the mobile application ('E' in figure 1) as an example to demonstrate this. Figure 3

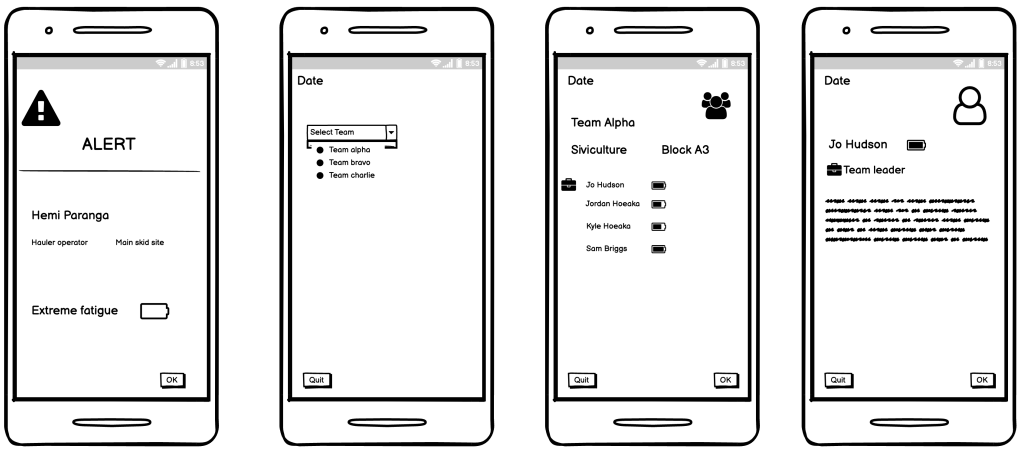

Fig. 3. Worker Alert Mobile Application Prototype

shows a simplified prototype of the mobile application which provides information about the work teams and individual workers. Once the manager has logged in they can select a work team to view from the list of all teams. The team detail screen shows a list of all team members with an overview of their details and fatigue status, selecting one of the team members provides a more detailed view of that person's details. If an alert is escalated (where a worker has ignored fatigue warnings for a designated period of time) the mobile application generates the alert as shown in the first screen of figure 3 and provides an audible alarm. The manager can clear the alarm page (which will return them to whichever screen they were viewing previously), but if the worker statistics do not improve the alert will continue to be re-issued.

The PModel for this prototype describes each of the screens separately (in a PModel) by way of its interactive elements (widgets) and assigns names, categories (which, at the highest level

describe whether the widget generates or responds to actions and behaviours) and behaviours to each widget. For example, the first model below describes the screen used to select a team to view. It consists of four widgets. The first is *TeamList* a drop-down list of all teams, when a team is selected two behaviours occur. A system action is generated (denoted by prefix 'S') S_SelectTeam and an interactive action is generated (denoted by prefix 'I') I_TeamView. Informally we understand that selecting a team from the list navigates us to a new screen where the details of the selected team are shown. The formal meaning of the 'S' and 'I' behaviours are given by the PMR and PIM below. textitQuitButton is the control which allows the user to exit the application (quit or off buttons are always labelled with the default behaviour 'Quit'). The *DateLabel* displays the current date, as such it responds to a system behaviour which provides the current date. Finally, the *AlarmActivator* widget is described twice as it both responds to a behaviour (S_Alert) which is the trigger for the audible alarm as well as generates a behaviour which switches the device to the alarm screen (I_Alert). The remaining screens are described in a similar manner, where each widget is given its associated category along with the behaviours generated or responded to by the widget.

```
TeamSelect is
   TeamList, ActionControl, (S_SelectTeam, I_TeamView),
   QuitButton, ActionControl, (Quit),
   DateLabel, Responder, (S_CurrentDate),
   AlarmSound, Responder, (S_Alert, I_Alert)

TeamView is
   TeamName, Responder, (S_SelectTeam),
   TeamData, Responder, (S_SelectTeam),
   WorkerList, Responder, (S_SelectTeam),
   WorkerList, ActionControl, (S_SelectWorker, I_WorkerView)
   OKButton, ActionControl, (I_TeamSelect),
   QuitButton, ActionControl, (Quit),
   DateLabel, Responder, (S_CurrentDate)
   AlarmActivator, Responder, (S_Alert)
   AlarmActivator, ActionControl(I_Alert)

WorkerView is
   WorkerName, Responder, (S_SelectWorker),
   WorkerRole, Responder, (S_SelectWorker),
   WorkerInfo, Responder, (S_SelectWorker),
   OKButton, ActionControl, (I_TeamView),
   QuitButton, ActionControl, (Quit),
   DateLabel, Responder, (S_CurrentDate)
   AlarmSound, Responder, (S_Alert, I_Alert)

Alert is
    AlarmInfo, Responder, (S_Alarm),
    OKButton, ActionControl, (I_TeamSelect, I_TeamView, I_WorkerView)
    AlarmSound, Responder, (S_Alert)
```

The mobile application as a whole is described as the concatenation of the widget descriptions (the semantics of PModels are a conservative extension of set theory), which is:

```
MobileApp is TeamSelect : TeamView : WorkerView : Alert
```

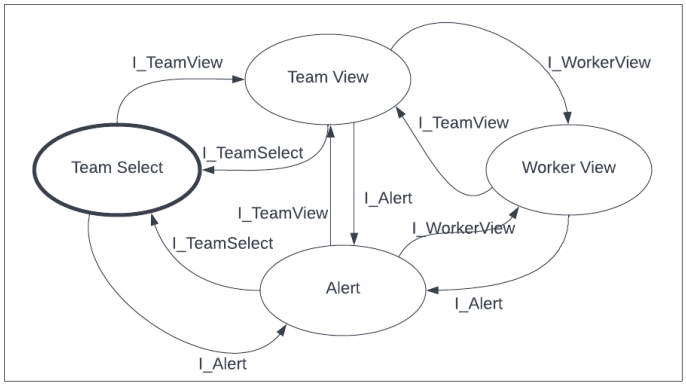

Fig. 4. Presentation Interaction Model

Figure 4 shows the PIM of the mobile application. Each state in the model represents a PModel of the same name, and is therefore an abstraction of all of the behaviours described in the model. Transitions between states represent the navigation between screens of the application and are labelled with the I-behaviours that provide the navigation, hence, the PIM gives meaning to the I-behaviours of the PModel by way of the transition diagram and show the availability of behaviours to a user by way of the state transitions.

The final part of the model is the PMR, which for this example is as follows:

$S\_SelectTeam \mapsto TeamDetailsOp$
$S\_SelectWorker \mapsto WorkerDetailsOp$
$S\_Alert \mapsto AlarmOp$
$S\_CurrentDate \mapsto DateOp$

Each of the S-behaviours from the PModel are related to an operation in a formal specification of the system. The formal specification describes the total state of the system and its operations (using a suitable language such as Z, B, Alloy etc.). The PMR, therefore, gives meaning to the S-behaviours through this relation.

The approach presented above is the standard use of PModels for interactive systems. In addition to its use for a mobile application it could also be used for the web application ('F' in figure 1), again this would be the standard use of PModels so we omit the description of this here. Of more interest is how we might extend this to incorporate additional aspects present in our IoT solutions ('A', 'B', 'C' and 'D' in figure 1). Then we can consider how we can address the requirements for the models given at the start of Section 4.

4.2 Extending the Modelling Approach for IoT

We now show how the existing modelling approach is extended to enable its use for IoT solutions. This provides one of the contributions of this work, a light-weight modelling approach for IoT solutions that supports the development of their heterogeneous parts in a consistent manner and which ensures they interact correctly with each other as well as provide correct interactivity for the user.

Sensors and actuators in IoT solutions represent mechanisms of data collection, outputs and user interactions (either directly through manipulation, or indirectly such as hearing or seeing an output). As such, we can consider them in a similar manner to the widget descriptions given in the PModel above. Rather than grouping them within interface windows or screens, we can group them according to the structure of the solution. For our example we consider all of the sensors and actuators ('A' in figure 1) together, as they are located in the same physical space, but we might equally have components in different places that we might want to group separately. The choices we make about such groupings will typically depend on the abstraction level of our models, our consideration of the data (see later discussions) and the organisation of the development teams. We describe the sensors and actuators using a PModel as follows:

```
Sensors is
    WorkerSensor, Sensor, (S_WorkerData),
    AlertOutput, Actuator, (S_Warning),
    EnvironmentSensor, Sensor, (S_EnvData)
```

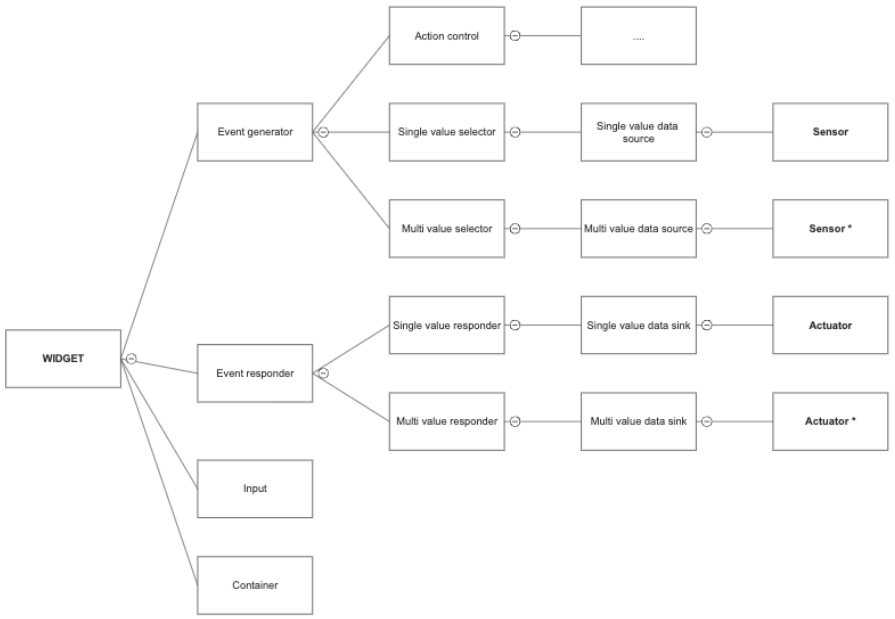

Fig. 5. Part of the Widget Category Hierarchy

Although we have labelled the components explicitly as sensors and actuators, we could similarly abstract this to ActionControls and Responders (as in the the previous part of the example). In fact, the categories that are used in PModels are described in a hierarchical tree structure, with the root node being the most general 'widget' and leaf nodes being concrete instantiations of these within the relevant code. Figure 5 shows where sensors and actuators are located within the hierarchy. It is always possible to be more abstract by using the category of an ascendant (e.g. a sensor could be described as an event generator) or less abstract by using the category of a descendant (e.g. a leaf node below sensor could be a specific type of sensor, such as heart-rate monitor or a sensor defined by a particular type of data). The 'Sensor' and 'Actuator' categories are used where there is a single

source of data sensed or responded to, while 'Sensor*' and 'Actuator*' are used when there is more than one type of data. As such, our sensors are more accurately described in the PModel as:

```
Sensors is
    WorkerSensor, Sensor*, (S_WorkerData),
    AlertOutput, Actuator, (S_Warning),
    EnvironmentSensor, Sensor*, (S_EnvData)
```

Both the worker sensor and the environmental sensor collect multiple types of data, whereas the actuator responds only to a single type of data (which is the warning).

The S-behaviours for the sensors and actuators relate to data which is collected, and subsequently transferred, throughout the system. While the S-behaviours give a picture of how the data moves through the system, they are abstract with respect to data type, temporal properties and transfer protocols. We discuss this further later.

The cloud and edge services ('C' and 'D' in figure 1) can be treated in the same way the functional core of an interactive system is treated in the PModel approach. This means that a formal specification can be used to describe the system state and operations, and these can be added to the PMR with associated S-behaviours from the models of sensors and actuators. In some cases, of course, this might seem to be a heavyweight approach, especially for small, non safety-critical services. We expect that a similar pragmatic approach is taken as in the wider PModel methodology, whereby specification snippets and partial models are used as required rather than there being an expectation that every part of the system is formally modelled. A wider discussion on the use of lightweight formal methods can be found in articles such as [1, 24] etc. and the use of interactive system models at different levels of abstraction and in various combinations is given in [15].

5 PRACTICAL APPLICATION

We now return to the motivation for creating the models, and using the workplace monitoring example, demonstrate their use to ensure that the required properties hold. At the start of Section 4 we outlined the six steps needed to satisfy the high-level properties that we wanted to be certain our system adheres to:

(1) Use the PModel to ensure we have the necessary sensors in the system for each type of data.
(2) Use the PModel to ensure we have the necessary actuators in the system for each type of data.
(3) Use the PMR to find the specified operations related to sensor and actuator behaviours.
(4) Ensure the specified operations match data input/outputs to related sensor/actuator behaviours.
(5) Ensure the operations in the specification describe correct behaviour.
(6) Confirm from the previous 5 steps that the required data inputs and outputs are available for the user and the functions and services needed to provide functionality exist.

In order to address these we do, of course, need the system requirements which typically form a large part of the basis for the models. We describe relevant parts of these for our example (which are a subset of the real-world example) as required to describe the process here.

Our first steps (1 and 2) require that we have necessary sensors and actuators for the required data collection and expression. The workplace monitoring project requires that we collect two distinct types of data: the workers' personal biometric data and the environmental data that forms the context for meaning of the biometric data. There are multiple components to each of these data types (e.g. from the workers we collect heart-rate (HR), galvanic skin response (GSR), step counts etc.), but for ease of description of process and simplicity we will group these together here into two distinct data types - personal and environmental. Development team A worked on the sensors

and actuators, supported by Team G who were responsible for streaming data handling and data analysis algorithms. The model guided development team A and provides information to further development teams (discussed below). Data output requires notification when alerts are raised. The PModel for these sensors and actuators, as described in section 4 is:

```
Sensors is
    WorkerSensor, Sensor*, (S_WorkerData),
    AlertOutput, Actuator, (S_Warning),
    EnvironmentSensor, Sensor*, (S_EnvData)
```

The model shows how the data from the worker sensors is managed into a single output (similarly the environmental sensors) captured by the s-behaviour (S_WorkerData) while the actuator must respond to a single trigger, represented by S_Warning. From this we can identify that there is a sensor for each of the required (simplified) data collection types and an actuator for the data output (alert).

Next we consider the behaviours associated with these sensors and actuators, i.e. what happens to the data collected or what triggers the actuator (step 3). Again, we omit a description of *how* the data collection triggers the action, as there may be a variety of different triggers (discussed later). In the PModel above we have three S-behaviours, we use the PMR to identify which operations describe these in the formal specification. We show the relevant part of this relation below:

$S\_WorkerData \mapsto CalculateRiskOp$
$S\_Warning \mapsto AlertLevelOp$
$S\_EnvData \mapsto CalculateRiskOp$

To satisfy step 4, we must ensure that for each of the related operations in the specification the data dependencies (required inputs and outputs to operations) are met. Note, we are not considering correctness of behaviour here (see later step for this), rather we are ensuring that the data flow shown in Figure 2 is correctly implemented. Different specification languages define inputs and outputs to operations in different ways. As the specification for this example is written using the Z language, we show how we manage the dependencies in Z. In general, the requirement for related s-behaviours in the PMR is as follows:

$S\_SensorAction \mapsto DataConsumingOperation$
$S\_ActuatorAction \mapsto DataProducingOperation$

S_SensorAction is any s-behaviour of a widget under the sensor category and S_ActuatorAction is any s-behaviour of a widget under the actuator category. A 'DataConsumingOperation' is a Z operation with a defined input, e.g.

$$\Delta DCOperation$$
$$\delta System$$
$$i? : DATA$$
$$\vdash$$
$$Predicate$$

In Z, the convention is to decorate inputs to an operation using the '?' symbol after the name. Similarly, a 'DataProducingOperation' is a Z operation with a defined output, e.g.

$\Delta DPOperation$

$\delta System$
$o! : DATA$

$Predicate$

Where by convention the '!' symbol is used after the name of an output of the operation.

For the operations defined in the PMR snippet above, we therefore check that *CalculateRiskOp* is a data consuming operation and *AlertLevelOp* is a data producing operation. Given the following specifications for these operations (again these have been simplified for the purpose of demonstration), we are satisfied that this step is correct.

$\Delta CalculateRiskOp$

$\delta System$
$pd? : DATA$
$cd? : DATA$

$detail\ of\ operation\ calculation\ omitted$

$\Delta AlertLevelOp$

$\delta System$
$alert! : DATA$

$riskLevel = High \land alert! = True$

In the description of *CalculateRiskOp* the two data inputs, *pd?* and *cd?* represent the personal and contextual data from the sensors (we omit the detail of the data processing calculations). In *AlertLevelOp* when the *riskLevel* observation (which is one of the *System* properties) has a value of High, then the output *alert!* has a corresponding value of *True*.

Step 5 concerns the behaviours of the specified operations. Numerous examples exist in the literature discussing model-checking of formal specifications of interactive systems (see [17, 27, 35], so here we describe the process at a high level only. The formal specification describes all of the possible states of the system and the operations that change that state. Initial model-checking can be performed to ensure there are no deadlocks or live-locks in the model (specification) and that any invariants included hold true in all states. For our example, using a specification written in Z, we use the ProZ plugin for ProB, an animator and model-checker for the B language [8]. Where the operations have parameter boundaries we might also define predicates relating to behaviours, which can also be checked in ProB using linear temporal logic (a detailed examination of the use of this method for safety-critical systems can be found in [13]). For example, the *CalculateRiskOp* receives data from the workers and environment and determines the current level of risk, we use model-checking to ensure the behaviour of this operation is correct, particularly that it cannot identify 'no risk' in any safety-critical situation or generate false positives for identified risk that lead to incorrect alerts. Alerts are escalated through the system based on a number of criteria, including where existing alerts are in place and risk is still being identified. Again, the model can be checked to ensure the correct parts of the system are notified when these escalations occur. For the data visualisation components of the system the properties we want to check in the model relate to privacy controls and sharing of data. This also enables us to ensure outputs from operations

[8]https://prob.hhu.de/ (last accessed April 2023)

(the observations marked with a ! in the Z specification) will be correct, and map properly to the actuators or other outputs of the system (step 6).

In addition to the steps described above which use the models to support the development teams in designing and building parts of the system correctly, there are, of course, further steps. Once the individual parts of the system were implemented these were tested, using both model-based testing (in a similar manner to methods described in [14, 33]) and user evaluations. A user-centred process was followed to ensure that usability issues were adequately addressed for all outputs. In the example, this meant considering the usability of not only the mobile and web applications (E and F in figure 1) but also the actuators (part of A in figure 1) as these provide haptic and visual feedback to the workers. For the mobile and web applications we performed usability evaluations, which, for the workplace project, had to take into account the different end-user groups with differing requirements. For example, when designing web applications to display personal data to the users we had to find ways of ensuring that the data was displayed in meaningful ways for the different stakeholders, and that privacy could be controlled by end-users who were not habitual computer users. Evaluation of the haptic and visual feedback required a number of different studies, to begin with we needed to explore suitability of different feedback mechanisms for use within an outdoor hazardous work environment, and then further investigate their use with the workforce (more information on this can be found in [9]). In general, the type of evaluation required will depend on the nature of the data outputs, but these will not necessarily be restricted to GUIs.

6 DISCUSSION

In Section 2 we discussed how different modelling techniques can be used to support design and creation of IoT applications. We assume an open approach, this is based on our experiences from the presented example, where we found that a mixture of off the shelf and bespoke hardware was required, as well as a variety of cloud and web services. As we have discussed earlier, such an approach introduced complexities with respect to the communication between the heterogeneous components of the system. We do not attempt to solve or address such problems with our models, rather we focus on the interactivity from the perspective of data flow and user requirements, which is one way to consider the aspect of interoperability. The lightweight nature of our models (although underpinned by formal semantics) naturally lends itself to flexibility, as it is easy to adapt and extend the models to different types of IoT components and sub-systems. Our approach also assumes the use of partial models and the use of formality only where required (as opposed to end-end modelling of all parts) which adds to the flexibility of use.

Data security and privacy are important concerns for IoT developers. The example we have presented relies on collection of personal data from workers, as well as commercially sensitive data from work environments. As such there is a need for robust cyber-security to ensure the data cannot be intercepted by malicious entities. Our models do not address these aspects as we recognise that this requires a different focus from our own. In the example presented data security was considered separately from the design and development aspects we have outlined in this work. Similarly, data privacy and the use, ownership and data responsibilities formed a large part of the example project, although these considerations have not been described in this paper. This consisted of ensuring the correct data was collected and communicated between parts of the system, and also that it was handled, displayed and stored in appropriate ways given the sensitive nature of the personal data from the workers and commercial sensitivity of some of the environmental data [10, 11]. At the present time this is the focus of ongoing work as we find ways of modelling these data properties that are easy to use alongside our other modelling techniques. Our goal is that these are also able to be used in similarly lightweight ways that allow us to focus only on the parts considered most important. This supports our pragmatic approach that uses partial and

sketchy models as necessary to reduce the overhead of the modelling work, whilst still capitalising on their use in supporting development.

The work of Sequeiros *et al.*, demonstrates the importance of modelling tools working in unison for IoT applications in order to capture the behaviour of IoT, Cloud and Mobile systems [34]. They cite a lack of a common framework between different models as a major challenge in security modelling for IoT. While their focus is on security, we have similarly found that the heterogeneity of IoT systems presents challenges when finding appropriate modelling solutions. We believe that in our work we demonstrate that a focus on interactivity within the system is one way of addressing such challenges while reducing workload for developers as they only need to understand one set of models. In future we will explore how our interaction models may be used in a common framework (as suggested by Sequieros) alongside other models which focus on the security and privacy aspects for data.

As discussed in Section 4, PModels can be used as the basis for test-generation, however, they are not currently used for code generation. The level of abstraction used in the models does not currently provide enough information to generate code and the models were designed to support reasoning at a higher level of abstraction. Furthermore, code generation for IoT systems, where languages and hardware are typically heterogeneous, is a much harder problem when compared with traditional interactive systems. While further work is required to add code generation process to this approach, we can make use of the existing test generation process to ensure the final application behaves as expected.

The contribution of our work has been to demonstrate how models developed for interactive systems can be extended to support the development of IoT systems. While we have chosen a particular set of models for this purpose, there are similar approaches which we believe could be applied successfully in a similar manner. The work of Harrison *et al.* uses PVS to formally describe interactive systems [23] this can be used in conjunction with front-end prototypes created using the PVISio tool [29]. While the prototypes that can be developed are currently more suited to traditional interactive systems, it might be possible to extend the approach in the manner we have detailed to incorporate the sensors, actuators and other hardware that contributes to IoT systems. For our work we have also considered the initial design of systems, but from an end-user perspective using design and ideation cards [36], such as the Tiles toolkit [31] and Generonimos [21]. This enables lightweight design ideas to be converted into partial models which can then be used to support the development process further. Similarly, interaction models that focus on design from user task perspectives [28] would be equally appropriate for the types of interactions needed within IoT systems.

While we focus in this work on the mechanisms of interactivity, the human factors associated with the use of such systems is equally important. The models we have extended, have been shown in previous works, to support development of usability testing and to help identify usability issues early on in the development process which gives us some assurance that we can extend these benefits to IoT systems. However, the complexity of the types and methods of interaction provided by IoT may require new ways of evaluating and supporting the human experience, which we do not consider in this work.

7 CONCLUSIONS

In this paper we have presented a real-world IoT system and demonstrated how interactive system models can be extended to support the development process. While there are many existing model-based approaches that exist for IoT, these typically focus on the specific challenges, such as security, communication, data privacy etc. Our contribution here is to show that we can also consider higher-level design challenges to ensure all of the required interactivity between users, IoT components

and the underlying processing and service parts of IoT systems is incorporated into their design. By focusing on interactivity, we have demonstrated that we can use one set of models to describe the web, mobile and IoT components of an application using a common modelling language. The primary benefit of this is that it gives developers a single language to execute their design ideas, allowing for a lightweight modelling approach while taking advantage of the v&v techniques formal modelling provides. In future, this will allow us to explore automated model checking and to build on this process for further MDD techniques for IoT applications.

REFERENCES

[1] Sten Agerholm and Peter Larsen. 1998. A Lightweight Approach to Formal Methods. *Lecture Notes in Computer Science* (09 1998). https://doi.org/10.1007/3-540-48257-1_10

[2] Nikhil Agrawal and Smita Singhal. 2015. Smart drip irrigation system using raspberry pi and arduino. In *International Conference on Computing, Communication & Automation*. IEEE, 928–932.

[3] Naoufal Ainane, Mohamed Ouzzif, and Khalid Bouragba. 2019. Health Monitoring System in a Smart Home. In *Proceedings of the 4th International Conference on Smart City Applications* (Casablanca, Morocco) *(SCA '19)*. Association for Computing Machinery, New York, NY, USA, Article 114, 5 pages.

[4] Muhammad Raisul Alam, Mamun Bin Ibne Reaz, and Mohd Alauddin Mohd Ali. 2012. A Review of Smart Homes—Past, Present, and Future. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 42 (2012), 1190–1203.

[5] Domenico Amalfitano, Nicola Amatucci, Vincenzo De Simone, Vincenzo Riccio, and Fasolino Anna Rita. 2017. Towards a Thing-In-the-Loop Approach for the Verification and Validation of IoT Systems. In *Proceedings of the 1st ACM Workshop on the Internet of Safe Things* (Delft, Netherlands) *(SafeThings'17)*. Association for Computing Machinery, New York, NY, USA, 57–63. https://doi.org/10.1145/3137003.3137007

[6] Nesrine Ammar, Hela Chaieb, and Ridha Bouallegue. 2016. From modeling with SysML to simulation with Contiki Cooja simulator of wireless sensor networks. In *2016 30th International Conference on Advanced Information Networking and Applications Workshops (WAINA)*. IEEE, 760–765.

[7] Carmelo Ardito, Giuseppe Desolda, Rosa Lanzilotti, Alessio Malizia, Maristella Matera, Paolo Buono, and Antonio Piccinno. 2020. User-defined semantics for the design of IoT systems enabling smart interactive experiences. *Personal and Ubiquitous Computing* 24 (2020), 781–796.

[8] Tansu Zafer Asici, Burak Karaduman, Raheleh Eslampanah, Moharram Challenger, Joachim Denil, and Hans Vangheluwe. 2019. Applying Model Driven Engineering Techniques to the Development of Contiki-Based IoT Systems. In *2019 IEEE/ACM 1st International Workshop on Software Engineering Research Practices for the Internet of Things (SERP4IoT)*. 25–32. https://doi.org/10.1109/SERP4IoT.2019.00012

[9] Judy Bowen and Annika Hinze. 2021. Designing for Inaccessible People and Places. In *Human-Computer Interaction - INTERACT 2021 - 18th IFIP TC 13 International Conference, Bari, Italy, August 30 - September 3, 2021, Proceedings, Part IV (Lecture Notes in Computer Science, Vol. 12935)*, Carmelo Ardito, Rosa Lanzilotti, Alessio Malizia, Helen Petrie, Antonio Piccinno, Giuseppe Desolda, and Kori Inkpen (Eds.). Springer, 546–556.

[10] Judy Bowen and Annika Hinze. 2022. Participatory Data Design: Managing Data Sovereignty in IoT Solutions. *Interact. Comput.* 34, 2 (2022), 60–71.

[11] Judy Bowen, Annika Hinze, Christopher Griffiths, Vimal Kumar, and David Bainbridge. 2017. Personal Data Collection in the Workplace: Ethical and Technical Challenges. In *HCI 2017 - Digital make-believe. Proceedings of the 31st International BCS Human Computer Interaction Conference, BCS HCI 2017, University of Sunderland, St Peter's campus, Sunderland, UK, 3-6 July 2017 (Workshops in Computing)*, Lynne E. Hall, Tom Flint, Suzy O'Hara, and Phil Turner (Eds.). BCS.

[12] Judy Bowen and Steve Reeves. 2008. Formal Models for User Interface design artefacts. *Innovations in Systems and Software Engineering* 4, 2 (2008), 125–141.

[13] Judy Bowen and Steve Reeves. 2013. Modelling Safety Properties of Interactive Medical Systems. In *Proceedings of the 5th ACM SIGCHI Symposium on Engineering Interactive Computing Systems* (London, United Kingdom) *(EICS '13)*. ACM, New York, NY, USA, 91–100.

[14] Judy Bowen and Steve Reeves. 2013. UI-design driven model-based testing. *Innov. Syst. Softw. Eng.* 9, 3 (2013), 201–215.

[15] Judy Bowen and Steve Reeves. 2017. Combining Models for Interactive System Modelling. In *The Handbook of Formal Methods in Human-Computer Interaction*, Benjamin Weyers, Judy Bowen, Alan J. Dix, and Philippe A. Palanque (Eds.). Springer International Publishing, 161–182.

[16] Marco Brambilla, Eric Umuhoza, and Roberto Acerbis. 2017. Model-driven development of user interfaces for IoT systems via domain-specific components and patterns. *Journal of Internet Services and Applications* 8, 1 (2017), 1–21.

[17] José Creissac Campos, Manuel Sousa, Miriam C. Bergue Alves, and Michael D. Harrison. 2016. Formal Verification of a Space System's User Interface With the IVY Workbench. *IEEE Trans. Hum. Mach. Syst.* 46, 2 (2016), 303–316.
[18] Priscila Cedillo, Wilson Valdez, Paúl Cárdenas-Delgado, and Daniela Prado-Cabrera. 2020. A Data as a Service Metamodel for Managing Information of Healthcare and Internet of Things Applications. In *Information and Communication Technologies*, Germania Rodriguez Morales, Efraín R. Fonseca C., Juan Pablo Salgado, Pablo Pérez-Gosende, Marcos Orellana Cordero, and Santiago Berrezueta (Eds.). Springer International Publishing, Cham, 272–286.
[19] R. Charles and D. Golightly (Eds.). 2021. *Supporting Safer Work Practice Through the Use of Wearable Technology*. CIEHF.
[20] Hao Ran Chi, Chung Kit Wu, Nen-Fu Huang, Kim-Fung Tsang, and Ayman Radwan. 2023. A Survey of Network Automation for Industrial Internet-of-Things Toward Industry 5.0. *IEEE Transactions on Industrial Informatics* 19, 2 (2023), 2065–2077. https://doi.org/10.1109/TII.2022.3215231
[21] Kate Compton, Edward Melcer, and Michael Mateas. 2017. Generominos: Ideation Cards for Interactive Generativity.
[22] Fredrik Dahlqvist, Mark Patel, Alexander Rajko, and Jonathan Shulman. 2019. Growing opportunities in the Internet of Things. (2019).
[23] Michael D. Harrison, Paolo Masci, and José Creissac Campos. 2018. Formal Modelling as a Component of User Centred Design. In *Software Technologies: Applications and Foundations*, Manuel Mazzara, Iulian Ober, and Gwen Salaün (Eds.). Springer International Publishing, Cham, 274–289.
[24] Cliff Jones. 1996. Formal Methods Light. *ACM Comput. Surv.* 28 (12 1996), 121. https://doi.org/10.1145/242224.242380
[25] Roland Kluge, Michael Stein, David Giessing, Andy Schürr, and Max Mühlhäuser. 2017. cMoflon: model-driven generation of embedded C code for wireless sensor networks. In *European Conference on Modelling Foundations and Applications*. Springer, 109–125.
[26] Ch. Kotronis, M. Nikolaidou, G. Dimitrakopoulos, D. Anagnostopoulos, A. Amira, and F. Bensaali. 2018. A Model-based Approach for Managing Criticality Requirements in e-Health IoT Systems. In *2018 13th Annual Conference on System of Systems Engineering (SoSE)*. 60–67. https://doi.org/10.1109/SYSOSE.2018.8428764
[27] Karsten Loer and Michael D. Harrison. 2002. Towards Usable and Relevant Model Checking Techniques for the Analysis of Dependable Interactive Systems. In *17th IEEE International Conference on Automated Software Engineering (ASE 2002), 23-27 September 2002, Edinburgh, Scotland, UK*. IEEE Computer Society, 223–226.
[28] Célia Martinie and Philippe A. Palanque. 2020. Task models based engineering of interactive systems. In *EICS '20: ACM SIGCHI Symposium on Engineering Interactive Computing Systems, Sophia Antipolis, France, June 23-26, 2020*, Judy Bowen, Jean Vanderdonckt, and Marco Winckler (Eds.). ACM, 16:1–16:2.
[29] Paolo Masci, Patrick Oladimeji, Yi Zhang, Paul L. Jones, Paul Curzon, and Harold W. Thimbleby. 2015. PVSio-web 2.0: Joining PVS to HCI. In *Computer Aided Verification - 27th International Conference, CAV 2015, San Francisco, CA, USA, July 18-24, 2015, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 9206)*, Daniel Kroening and Corina S. Pasareanu (Eds.). Springer, 470–478.
[30] Saraju P. Mohanty, Uma Choppali, and Elias Kougianos. 2016. Everything you wanted to know about smart cities: The Internet of things is the backbone. *IEEE Consumer Electronics Magazine* 5, 3 (2016), 60–70.
[31] Simone Mora, Francesco Gianni, and Monica Divitini. 2017. Tiles: a card-based ideation toolkit for the internet of things. In *Proceedings of the 2017 conference on designing interactive systems*. 587–598.
[32] Ulrich Norbisrath, Ruben Jubeh, and Albert Zündorf. 2013. Story Driven Modeling.
[33] Miguel Pinto, Marcelo Gonçalves, Paolo Masci, and José Creissac Campos. 2017. TOM: A Model-Based GUI Testing Framework. In *Formal Aspects of Component Software - 14th International Conference, FACS 2017, Braga, Portugal, October 10-13, 2017, Proceedings (Lecture Notes in Computer Science, Vol. 10487)*, José Proença and Markus Lumpe (Eds.). Springer, 155–161.
[34] João B. F. Sequeiros, Francisco T. Chimuco, Musa G. Samaila, Mário M. Freire, and Pedro R. M. Inácio. 2020. Attack and System Modeling Applied to IoT, Cloud, and Mobile Ecosystems: Embedding Security by Design. *ACM Comput. Surv.* 53, 2, Article 25 (mar 2020), 32 pages. https://doi.org/10.1145/3376123
[35] Jessica Turner, Judy Bowen, and Steve Reeves. 2020. SeqCheck: a model checking tool for interactive systems. In *EICS '20: ACM SIGCHI Symposium on Engineering Interactive Computing Systems, Sophia Antipolis, France, June 23-26, 2020*, Judy Bowen, Jean Vanderdonckt, and Marco Winckler (Eds.). ACM, 7:1–7:6.
[36] Jessica Turner, Judy Bowen, and Nikki van Zandwijk. 2021. Interaction Modelling for IoT. In *2021 28th Asia-Pacific Software Engineering Conference (APSEC)*. 120–129. https://doi.org/10.1109/APSEC53868.2021.00020
[37] Ruowei Xiao, Zhanwei Wu, and Dongyu Wang. 2019. A Finite-State-Machine model driven service composition architecture for internet of things rapid prototyping. *Future Generation Computer Systems* 99 (2019), 473–488.

Received October 2022; revised February 2023; accepted April 2023