Interaction Modelling for IoT

Jessica Turner
The University of Waikato
Tauranga, New Zealand
jessica.turner@waikato.ac.nz

Judy Bowen
The University of Waikato
Hamilton, New Zealand
judy.bowen@waikato.ac.nz

Nikki van Zandwijk
The University of Waikato
Hamilton, New Zealand
nv30@students.waikato.ac.nz

***Abstract*—Informal design artefacts allow end-users and non-experts to contribute to software design ideas and development. In contrast, software engineering techniques such as model-driven development support experts in ensuring quality properties of the software they propose and build. Each of these approaches have benefits which contribute to the development of robust, reliable and usable software, however it is not always obvious how best to combine these two. In this paper we describe a novel technique which allows us to use informal design artefacts, in the form of ideation card designs, to generate formal models of IoT applications. To implement this technique, we created the Cards-to-Model (C2M) tool which allows us to automate the model generation process. We demonstrate this technique with a case study for a safety-critical IoT application called "Medication Reminders". By generating formal models directly from the design we reduce the complexity of the modelling process. In addition, by incorporating easy-to-use informal design artefacts in the process we allow non-experts to engage in the design and modelling process of IoT applications.**

***Index Terms*—Formal Modelling; Human Computer Interaction; Formal Methods; Internet of Things**

I. Introduction

In *rapid prototyping*, ideation processes are used whereby designers produce informal design artefacts to communicate and visualise their ideas. These artefacts allow designers to explore various solutions to a specific design problem. Ideation card decks have been proposed as a way to support both experts and non-experts to engage with this rapid prototyping process. There are many different types of cards and card decks, but typically they all provide a common structure around which to discuss and clarify ideas.

While we might expect designers of software to have expertise in the domain, in contrast, developers of Internet of Things (IoT) applications may be hobbyists or people in maker communities who are less technically experienced [1]. Ideation cards have also been developed specifically for designing Internet of Things (IoT) applications [2], [3]. The purpose of the cards is to reduce the complexity of designing an IoT application and support those who may not necessarily be familiar with the relevant technology, allowing them to explore solutions they could not have otherwise considered. Expert designers can therefore easily communicate design ideas with end users or stakeholders and non-experts can engage in IoT design processes for their DIY solutions.

Different types of ideation card decks may focus on different aspects of a design. For example, "The Intelligence Augmentation Design Toolkit"[1] enables designers to investigate the user experience of an IoT application, which incorporate machine learning and artificial intellignence. Alternatively, the "Tiles IoT Inventor Toolkit"[2] provides designers with challenges centering on global issues, such as recycling or traffic congestion etc. These challenges provide the basis for a design but they may also be used to design IoT applications in general. Using this toolkit, a designer can create an IoT application at an abstract level, considering which sensors, actions, services and feedback would be relevant to the end user. As such, they are similar to other informal design artefacts, such as paper-based prototypes, which support end-user and non-expert collaboration.

In contrast, formal models are used for verification and validation of software, interactive systems, and more generally IoT applications. Within safety-critical settings, verification and validation are important to ensure we reduce the potential risk and harm that could be caused by error. By being able to reason about a system formally, we can prove that a system has certain properties (for example safety, liveness, etc.) While the benefits of formal models are evident, they are not communicated as easily to end users and instead require expert knowledge in order to be utilised accurately. Given the success of informal design artefacts for communicating with stakeholders, it would therefore be useful to incorporate the benefits provided by these informal artefacts with those of formal models. This has been recognised in previous work which seeks to make formal models more accessible ([4]–[8]).

Previous methods which have utilised informal design artefacts as the basis for formal models have typically focused on the interface design or tasks that are able to be completed by an end user. However, IoT applications are increasingly being seen as solutions within safety-critical domains (aviation, connected cars, power plants etc.), and therefore design considerations such as correctness and robustness are becoming similarly important. IoT applications present a novel problem as they have components which affect an end user's interaction but may not necessarily be interacted with directly (for example, environmental sensors). Conversely, an end user may have an indirect effect on a sensor or actuator of an IoT application without it being evident (e.g. when sensors react to

[1]See https://futurice.com/ia-design-kit
[2]See https://www.tilestoolkit.io/

the physical presence of a user). As a result, existing methods which combine informal and formal design artefacts do not typically capture the richer range of interaction modes and interactive elements that IoT applications present. Therefore, we must consider extending these methods or developing new ones specifically for IoT applications.

In this paper we present a technique for using informal design artefacts, in the form of ideation card decks, to create formal models of an IoT application. In the next section we present some background to model-driven development in general, and present related work on modelling IoT applications. This is followed by a description of the approach we propose for modelling IoT applications with an explanation of the benefits this provides. In Section IV we describe the ideation cards used in our approach and explain how to generate models from a card-based design. We then introduce the Cards-to-Model (C2M) tool which automates this process and exemplify this with a case study. Lastly, we discuss the outcomes of our work and finish with concluding remarks.

II. Background and Related Work

A. Model-Driven Development (MDD)

In MDD, models of a system are created and then used as the basis to derive or generate code and/or tests [9]. Weyers *et al.* state that models allow a system to be abstracted to a higher level, where focus can be applied to specific aspects [10]. In this way models act as a tool to break complex applications into manageable parts. Some modelling languages may also provide automation enabling generation of code from the model and most have tool support which also allows models to be re-used [11].

A design-focussed approach is given by Masci *et al.* [7] which can be used to prototype and model interactive systems. Their tool, PVSio-web, creates models of systems and analyses them, however in this work the focus is on user interactions and interfaces of medical devices or clinical environments (see [12]) as opposed to IoT-only applications.

There are several MDD approaches for IoT applications which use formal models in their approach [13]–[15]. Souri *et al.* use verification of formal models to prove correctness for the behavioural workflows in IoT applications [16], while Krichen shows how the testing of an IoT application can be completed using formal verification and model-based testing [17]. These works demonstrate the usefulness of using models in an MDD approach for IoT applications. Their focus on formal methods allows for a rigorous model-based approch, but relies on the expertise of formal methods practitioners which we aim to avoid in our own approach.

In contrast, "The Interaction Flow Modelling Language" (IFML)[3] is a standard that is used to create abstract descriptions of the front-end of an application. The models used in this approach are semi-formal and as a result do not allow for verification and validation, but may be easier for non-experts to adopt. Bernaschina *et al.* describe how these can be mapped to an existing formal language like Place Chart Nets, a variant of Petri Nets, in order to take advantage of the benefits of formal models [18]. They also describe an online tool (IFMLedit.org) which can be used to create models. In our approach we generate formal models directly from the informal design artefacts, removing the need to map between semi-formal and formal models, thereby providing a more streamlined solution.

Domain specific languages (DSL) have also been proposed for describing IoT applications. For example Khaled *et al.* present the IoT Device Description Language (IoT-DDL) [19], a machine readable DDL for 'things' which aims to support integration and homogenization between disparate technologies. IoT-DDL explicitly tools 'things' to self-discover and securely share their own capabilities which helps to break down the barriers that typically exist when trying to incorporate a wide-range of different 'things' (particularly from different providers) into IoT applications. These types of DSLs help support important properties of IoT design, and alongside the use of MDD contribute to robust design and development processes. However, they all require significant technical understanding in order to be used. In our approach we reduce the technical knowledge required by using easy-to-understand informal design artefacts to generate formal models.

Other MDD approaches for IoT applications have been proposed which focus on aspects such as adaptive IoT [20], management of heterogeneous components [21], run-time management [22], etc. Of most relevance for our work are those which focus on user-interfaces and user interactions. Brambilla and Umuhoza described a visual modelling language for IoT, which is also based on IFML, and which incorporates IoT specific UI design patterns. This provides a mechanism for partial code-generation, as well as validation of proposed solutions. Their use of design patterns is intended also to support user acceptance. In contrast, we seek to support user-centred design approaches by providing a way of incorporating light-weight design artefacts into a more formal model-driven approach. As such, we build on approaches which seek to support interaction design in safety-critical domains through the use of combining models at different levels of focus and abstraction (such as [23]–[26] etc.) As has been described previously, the use of different types of models adopted in such approaches leads to a wide-range of benefits over and above the provision of formal descriptions [27] and we envisage these benefits will also be seen for IoT development.

The approaches described here all require a high level of technical expertise, or effort in translating between semi-formal and formal approaches in order to be utilised. However, our goal is to use informal design artefacts, which are easy for end users to understand, to inform an MDD approach without the need for mapping or translation effort. The benefit of this is that it will reduce the technical expertise required to create IoT applications while retaining the benefits formal models provide.

[3] See https://www.ifml.org/

B. Informal Design Artefacts

In a rapid prototyping environment, designers use various techniques to assist in ideation processes. Research findings suggest that game-like tools which provide tips and guidance are successful in supporting ideation [2], [3]. Peters *et al.* [28] examined 76 analogue design tools and in their study found that card based tools made up the majority. Design cards have increased in popularity over the past 3-5 years [29] and this effect has also been seen in the IoT design domain. We therefore focussed our attention on card based ideation tools as the starting point for our informal design approach.

System design ideation cards are not limited to IoT applications alone. "The Intelligence Augmentation Design Toolkit"[4] aims to make Artificial Intelligence (AI) concepts easier to understand in order to create an application which uses AI. "UX Flowchart Cards"[5] are used to aid in website structure planning, and while they help design a better user experience, they also assist in the design of the system itself.

Ideation cards aid collaboration and are useful for exploring different design ideas. These cards fall into one of two categories, user experience or system design. For example, "Ethics Cards"[6] and "The Tarot Cards of Tech"[7], demonstrate ways to use ideation in designing user experience. Alternatively, Karakuri IoT [30] and the Tiles IoT Toolkit Cards [31] are decks used for the ideation of IoT applications. As our focus is on the design of IoT applications, we consider only cards which describe the system design.

III. Modelling IoT Applications

IoT applications comprise different components and combinations of components depending on their purpose and domain of use. Here we consider IoT applications that incorporate some sort of user interaction. Again these vary in size and complexity. In the simplest case we may have a single sensor or actuator where the only interaction is initial set up. For example, the simple "Smart bulb" system shown in Figure 1 provides a mobile app where the user can set times for the bulb to turn on and off, no other interaction is required. In the most complex cases, however, we may have multiple sensors and actuators interacting with multiple services and users (e.g. a smart city environment). Some of these examples will either be entirely safety-critical or incorporate aspects that are safety-critical. These are the cases of most interest to us, as they necessarily require robust software engineering methods to ensure critical aspects are correct (although we would argue that such robustness and correctness also apply more generally).

Our aim then is to consider three main components of IoT applications (sensors, actuators and 'things'; cloud services; mobile or web applications targetted at users) within the design process to ensure consistency and correctness. That is, we want to ensure that the components described satisfy the user and behavioural requirements correctly. Consider the IoT plant water sensor application shown in Figure 2 as an example. In the simplest case a single sensor is registered to a cloud service which has details of the plant type, soil type, water requirements etc. The sensor is placed into the soil of the plant pot and sends moisture readings to the cloud service at regular intervals. The readings are assessed by the cloud service to determine if any action is required (for example if the soil is too dry or too wet) and if action is required an update is sent to a user's mobile app. The data is also stored and can be used to provide information over time for the specific plant. The mobile app provides notifications to the user based on updates as they arrive, it also enables the user to report actions taken (i.e. if they water the plant) which will clear notifications. The mobile app can also report a history log of notifications and actions, as well as display information about the sensor readings/plant over time. To ensure that the application design will behave as required we need to ensure the following:

- the proposed sensor(s) provide(s) required information
- the cloud service correctly analyses the data
- the cloud service generates required notifications
- the mobile app provides the functionality to inform user of notifications
- the mobile app can provide updates to the cloud service

In order to achieve this we now consider how we might model such a system, we present the details of this next.

As described in Section II, there are different approaches we can take to modelling this design in order to be able to consider the consistency and correctness properties. Here

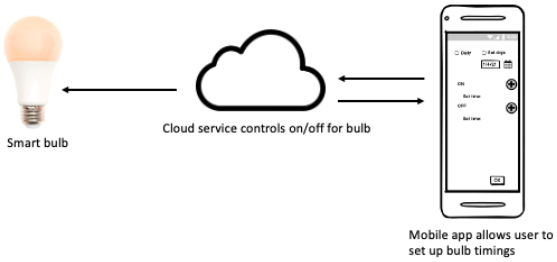

Figure 1. Smartbulb IoT Application

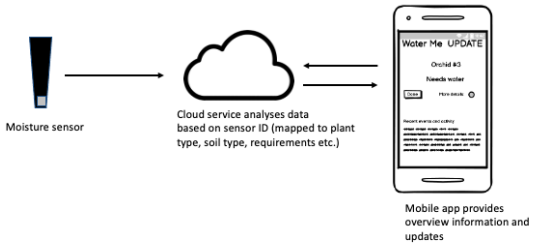

Figure 2. Plant Water Sensor IoT Application

[4]See https://futurice.com/ia-design-kit
[5]See https://www.uxflowcharts.com/
[6]See http://ethicskit.org/ethics-cards.html
[7]See https://www.artefactgroup.com/case-studies/the-tarot-cards-of-tech/

we will use the 'Presentation Model' approach [32] which combines different models for different parts of the system. The advantages of this approach are that it uses lightweight models for the interactive components which lend themselves to design artefacts, while also providing a formal semantics. The semantics for the interaction model enables combination with a formal specification of the system behaviour, which in turn allows for formal reasoning and model checking of the entire system (see [25] for a detailed explanation of this). The presentation model approach was developed for interactive systems more generally rather than IoT solutions, however it is easily extended to capture the information of an IoT application, which we describe next.

Originally presentation models were designed to capture the narrative (both the meaning and interaction possibilities) of design artefacts such as prototypes, scenarios, storyboards etc. The approach consists of three main components:

1) Presentation model(s) (pmodels) describing the interface elements (widgets) of a system along with their behaviours
2) Presentation interaction models (PIMs) describing the navigational properties of the system
3) Presentation model relation (PMR) which connects the described behaviours of the widgets with a formal specification

The formal specification is assumed to have been developed separately from the design artefacts (following the usual process for specification creation) and while it is typically given in the Z notation [33] it could be any formal language (e.g. ML within Petri Nets [34]). In order to make this approach applicable for IoT applications we extend the categories of interactive widgets to include IoT sensors and actuators, and so describe these in pmodels. In addition we include both the service behaviour of the cloud and functional behaviour of the mobile app within the formal specification. As such, the PMR relates the sensors and mobile apps with relevant system functions. Consider again the example in Figure 2, we describe the sensor and the widgets of the mobile screens (for simplicity only one is shown in the image but there will be more in the total design) in pmodels, describe the navigation within the mobile app in the PIM and relate the behaviours of all widgets and sensors to the formal specification in the PMR. The specification describes the functionality of the cloud service as well as the mobile app. We therefore have the following:

Presentation model

```
WaterMe app is sensors:wmMobile
sensors is
(moistureSensor,dataInput,(S_updateData))

wmMobile is notification:detail:mainScreen

notification is
(plantInfo,display,(S_plantNotification)),
(doneButton,actioncontrol,(S_confirmAction,
                                  I_main)),
(detailButton,actioncontrol,
              (S_showFullDetail,I_detail),
(historyLog,display,(S_showHistory))
```

Note we have omitted the pmodels for the 'main' and 'detail' screens of the mobile app for brevity. The first line of the model declares the total IoT application (WaterMe) as the conjunction of all of the sensors and wmMobile pmodels (presentation models are a conservative extension of set theory, allowing all standard set operations). The pmodels (wmMobile and notification) consist of tuples describing the widgets/sensors. Each tuple gives the name of the widget/sensor, a high-level category description and a set of associated behaviours. Behaviours pre-fixed with an 'S' are functional behaviours whereas those pre-fixed with an 'I' are interactive behaviours of the UI. Figure 3 shows the presentation interaction model for the system. It is a transition diagram which shows how the I_behaviours enable a user to navigate through the system. It is an abstraction of available behaviours (each state represents the behaviours of the pmodel with the same name) and defines the meanings of I_behaviours by visualising their effects from each state they are available within.

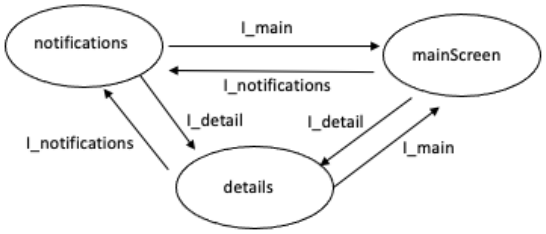

Figure 3. Presentation Interaction Model (PIM)

PMR

$S\_updateData \mapsto UpdateOp$
$S\_plantNotification \mapsto NotifyOp$
$S\_confirmAction \mapsto ActionPerformedOp$
$S\_showFullDetail \mapsto NotifyOp$
$S\_showHistory \mapsto OutputFullOp$

The PMR relates each S-behaviour in the models to operations in the formal specification, thereby giving formal meanings for the S-behaviours. Note that at this level of abstraction we are not considering connectivity and dataflow between sensors and the cloud service, while these are important aspects they are considered separately from the design models and therefore do not form part of our discussions here.

The Z specification describes the functional behaviour of the system. In its simplest state we consider the system as a collection of sensors with associated data, described as:

WaterMeSystem
$sensors : Id \pfun Data$

Each of the operations described in the PMR (right-hand side of the pairs in the relation) are also described in the specification, along with any other relevant operations and predicates. For example, the 'UpdateOp' would be described as:

$$
\begin{array}{|l}
\hline
\textit{UpdateOp} \\
\Delta \textit{WaterMeSystem} \\
np? : \textit{Id} \rightarrow \textit{Data} \\
\hline
\textit{sensors}' = \textit{sensors} \oplus np? \\
\hline
\end{array}
$$

where a new reading from a sensor (again described as a function from id to data) overrides the existing sensor id/data pair in the system. It is not our intention to describe Z specifications here, rather we provide these snippets as examples to demonstrate how the PMR enables formal meaning to be given to the S-behaviours from the pmodels.

A. Benefits of the Approach

Although the pmodel approach was designed for the description of interactive systems in general, rather than IoT applications, we can see from the example above that it is possible to model IoT applications in this way. Now we consider the benefits of doing so. Once we have initial design ideas for an IoT application we want to ensure that the proposed system will do what is required. Note we are not addressing requirements engineering here, other research has investigated the use of design specifications, such as behavioural specifications, within model-based engineering approaches [35], [36]. Such research could be integrated with the work we propose in this paper, but here, for simplicity, we assume a known set of requirements. The properties we want to consider, therefore are:

- Can we gather the required data?
  1) Use the pmodel to ensure we have the necessary sensors in the system.
- Can we evaluate the data as required?
  2) Use the PMR to find the specified operations related to sensor behaviours.
  3) Ensure the operations in the specification describe correct behaviour (e.g. through model-checking).
- Do we provide the necessary information back to the user?
  4) Use the pmodel to ensure we have the required data outputs.
  5) Use the UI design to ensure usability of the system required for user to access and comprehend the data outputs.
  6) Ensure the operations in the specification provide required data.
  7) Use the PMR to find the outputs connected to the Z operations.

Details of how to undertake each of these steps (via model-checking and refinement for example) have been described in other works [25], [37], [38]. While we discuss these further in Section VII, our intention in describing them here is to motivate the use of the pmodel approach and demonstrate the benefits of the model-generation approach we present next.

IV. IoT Ideation Cards

In this work we made use of the Tiles IoT Toolkit Cards in order to design IoT applications. Mora *et al.* describe these as "a set of 110 cards which consist of 6 different types and a workshop activity, utilising a booklet and board, to help teams come up with IoT innovations" [31]. The purpose of the cards is to engage non-experts in idea creation of an IoT application via object augmentation. The cards consist of the following categories:

- **Things:** everyday objects that could be enhanced using technology (e.g. fridge, pens, shoes).
- **Sensors:** used to connect to everyday objects to collect data.
- **Services:** ways to provide or store information (e.g. internet services, social networks).
- **Human Actions:** how a human can physically affect an object.
- **Feedback:** how the object communicates to the end user.
- **Missions:** goals to spark ideas that give purpose and value to the product.
- **Criteria:** reflect and evaluate the design.
- **Scenarios:** problems and challenges that the IoT solution may attempt to address.
- **Personas:** potential users of your system who are in the target audience.

The "Mission" booklet is an instruction manual which describes how cards should be used. First, a mission or scenario is selected, followed by personas which are in the target audience of the application. Next, things, sensors and services are selected followed by human actions and feedback to describe how the things and services are used. In addition to the cards, ideas can be discussed and sketched. Lastly, criteria cards are chosen to find ways to improve the idea.

In a study by Sintoris *et al.*, the Tiles IoT Toolkit was used with groups of engineering students to aid ideation [39]. Results from the the study found that students concluded that the tool supported fast development of new ideas and elaboration of those ideas. This highlights the flexibility of and ease of use for the Tiles IoT Toolkit, demonstrating its applicability for our work.

V. C2M Tool: Generating Models from a Design

We created the C2M tool (Figure 4) to automate the process of using a card design to generate pmodels, a PIM and PMR. The Tiles IoT Toolkit cards were designed to be used in a tangible process, this is useful for understanding and ideation, but not ideal for model creation. This is because in order for a design to be processed the card design must be digitised. C2M allows existing designs to be added or new designs created directly in the tool. Once the design is created, C2M automatically generates the necessary models. In addition to

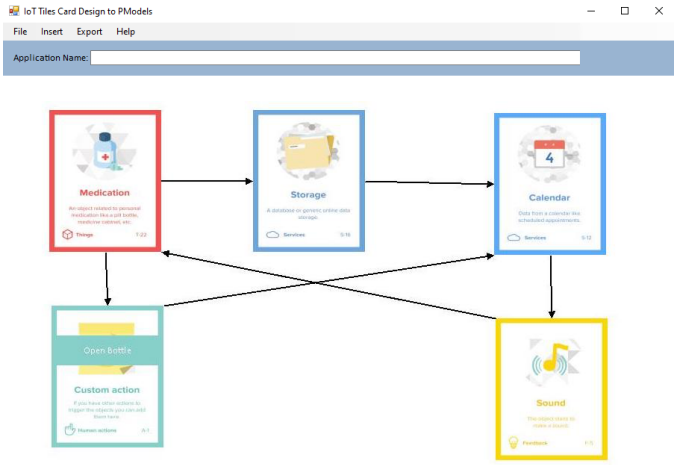

Figure 4. Screeenshot of C2M Tool

generating the models, the uploaded design is saved as an image and can be reused.

In order to upload or create a design in C2M, users of the tool must first select the ideation card deck that they wish to use. Several different card sets have been included for use in the tool, but full model-generation has so far only been defined for the IoT Toolkit cards. The user can select different cards from the chosen deck and place them in the digital design space in the same way as they would place them on a surface in a tangible process. Within the tool we use the cards listed above, but exclude the criteria, scenario, persona and mission cards as they are simply tools for sparking ideas or reflection and do not describe meaningful properties of the designed application. In addition, we provide an annotated version of the "Service" card, which we call "HI_Service" to indicate that this is a service that the user will interact with (as opposed to cloud storage for example). The 'arrow tool' allows the user to define connections between each card in the design which shows which items are connected to each other.

VI. Case Study: Medicine Reminders

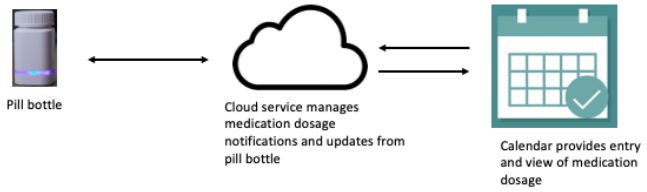

Figure 5. Medication Reminder

In order to describe how the tool converts the cards into the presentation model we introduce an example which we will use to demonstrate the process. We propose a simple IoT application which uses a smart pill bottle to remind patients to take their medication, see Figure 5. The medicine and time to be taken are uploaded to a cloud service which populates a calendar. When it is time to take the medicine, the smart pill bottle will make a sound. When the patient opens the bottle to take their medicine the cloud service is notified that the appropriate action has occurred. While this application is simple, it is an example of a safety critical system where the use of validation and verification of a formal model is important. If a patient takes too few or too many doses of a medication, this could result in undesirable and potentially life-threatening consequences.

The "thing" in this application is a smart pill bottle which is connected to two services, cloud storage and calendar and time service (which is an interactive service). The cloud storage service will keep information relevant to the medication, such as what type of medication is in the bottle. The calendar and time service manages when to take the medication, including dates and times. The calendar and time service sends a notification when it is time to take the medication, this is provided as feedback in the form of an audible sound coming from the bottle. The only action that can be taken by the end user is to open the bottle and take the medication, this then triggers a response from the bottle to the calendar to say that medication has been taken. Note that in this scenario we consider a reasonable user who would not deliberately avoid taking their medication. The calendar may be provided as a mobile app or web page, and we imagine in a full implementation that this would allow the user to manage the notifications. However, in this example we simplify this and assume that it provides only a visual guide to the user (where the initial data may be populated by the pharmacist when a prescription is filled for example). Figure 6 displays the Tiles IoT Toolkit card design for this particular application.

The medication card is connected to both the cloud storage and custom sensor card "Bottle Lid". The connection to the cloud storage card indicates that the *thing* card sends information to the storage service, while the sensor card connection indicates that the *thing* controls and/or responds to the "Bottle Lid" sensor. The cloud storage service is connected to the calendar and time service, indicating that data from the storage service is passed to the calendar and time service. The calendar and time service is connected to the sound feedback card which demonstrates that the feedback card requires information from the calendar and time service in order to notify the user at the right time. The sound card is connected to the medication card as the sound is made by the bottle to notify the user. The "Open bottle" and "Close bottle" action cards describe the actions that the "Bottle Lid" sensor responds to. The "Bottle Lid" sensor sends information to the calendar and time service to indicate whether or not the bottle has been opened by the user. This illustrates that the cards and their connections can have different meanings depending on how they are placed in the design. These will need to be correctly interpreted when we construct the model.

Next we describe how the C2M tool generates the pmodels, PIM and the PMR from the design. The types of the selected cards (indicated by different colours in Figure 6) and the connections between them are important when defining the

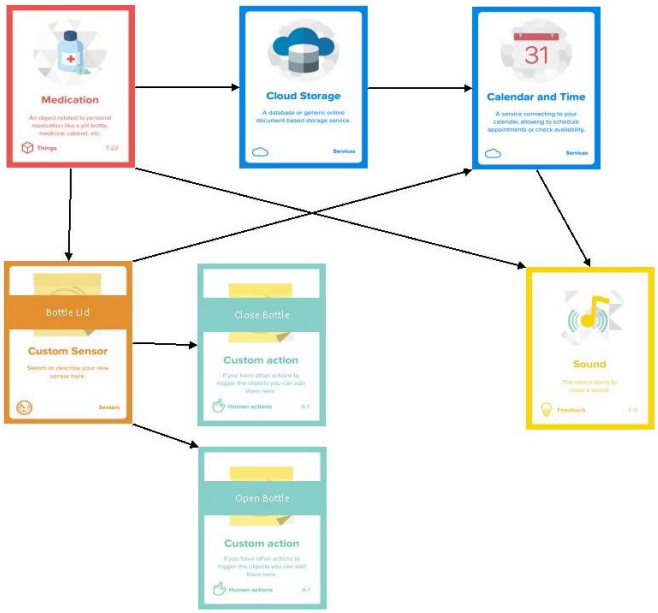

Figure 6. Medication Reminder IoT application in C2M Tool

pmodels, as these define different behaviours and states in the IoT application.

For each of the types of cards and connections we define how they will be modelled.

- Thing - pmodel and widget
- Sensors - widget
- Service - behaviour
- HI_Service - behaviour
- Action - behaviour
- Feedback - widget and behaviour

In the design for the smart medication application shown in Figure 6, we have seven cards and eight connections which are used to create the models as described below. To give names to the pmodels, widgets and behaviours the tool takes the names from the cards (given by the user). The name of the overall system being modelled is taken from the name of the saved design file given in the 'Application name' input on the tool interface.

The "Medication" card is a *Thing* card and will therefore be modelled as a pmodel. The "Cloud Storage" card is a *Service* card and will be specified as the behaviour "S_CloudStorage". The "Calendar and Time" card is an *HI_Service* and will become a behaviour called "S_UpdateCalendar". The "Sound" card is a *feedback* card and will be turned into a widget with the behaviour "S_Sound". The "Bottle Lid" card is a *sensor* card and will be turned into a widget. Lastly, the "Open bottle" and "Close bottle" are both *action* cards, they will therefore become the behaviours "S_OpenBottle" and "S_CloseBottle" respectively.

We use the above information to create pmodels by investigating the connections between each of the cards. Firstly, we determine which widgets to include in the pmodel, these are determined by the connections between cards. In this example there is only one pmodel, the "Medication" pmodel and associated widget "Medication". The thing card is connected directly to both the custom sensor "Bottle Lid" and feedback card "Sound", therefore these widgets are included in the Medication pmodel. Note that widgets which are associated with the pmodel or feedback cards are categorised as responders, while sensors are categorised as action controls. This is because any card which is to become a widget may only be an action control if it is connected to associated action cards, otherwise we assume it is a responder.

Next, we determine which behaviours are associated with which widgets. A behaviour is associated with a widget if there is an incoming transition from that widget, with the exception of the feedback card (as its behaviour is directly associated with its widget). Therefore, in our example: the medication widget has exactly one associated behaviour from the "Cloud Storage" service card; the bottle lid widget has three associated behaviours, two from the action cards "Open Bottle" and "Close Bottle", and another from the "Calendar and Time" HI_Service card; lastly, the sound widget is associated in this example only with its direct behaviour ("S_Sound") as it has no outgoing transitions.

Using the process described above, the C2M tool generates the following presentation model and PMR for this design:

Presentation model

```
MedicationReminder is Medication

Medication is
(Medication, Responder, (S_CloudStorage))
(BottleLid, ActCtrl, (S_OpenBottle,
S_CloseBottle, S_UpdateCalendar))
(Sound, Responder, (S_Sound))
```

PMR

$S\_CloudStorage \mapsto CloudStorageOp$
$S\_Sound \mapsto SoundOp$
$S\_OpenBottle \mapsto OpenBottleOp$
$S\_CloseBottle \mapsto CloseBottleOp$
$S\_UpdateCalendar \mapsto UpdateCalendarOp$

The models created are at a high level of abstraction and act as a starting point within the modelling process. As the design is further refined the models will be likewise refined to match the more detailed information. If this includes details of interactive parts of the application (e.g. the Calendar) which includes I-behaviours then we can subsequently create the PIM (which is already an automatic process from the presentation model). As the design process moves beyond the high-level ideas created from the design to more detailed considerations (perhaps using sketches, prototypes, wireframes etc.) we can use a refinement process (see [37] for details of this) to ensure we capture the original intentions of the design. We discuss this further in VII.

A. Using the Models

In Section III we outlined the properties of an IoT design (in that case the smart bulb example) that we would like a model-based approach to support, we return to these now and discuss how these are supported with reference to the model features described in Section III-A. While the properties we described were specific to the plant sensor example, we can generalise them to all IoT systems as follows:

- the proposed sensor(s) provide(s) required information
- the cloud service correctly analyses the data
- the cloud service generates required notifications
- the system/actuator(s) provides the functionality to inform user of notifications
- the system can provide updates to the cloud service

Where the IoT system includes a mobile app we consider this within the system properties in the list above.

The first property we wanted to support was the ability to ensure that the proposed sensor(s) provide(s) the required information. There are two parts to this, firstly ensuring we have the necessary sensors described; secondly ensuring the sensors send data to the correct services/things. In our smart pill bottle example we can identify from the presentation model that the "BottleLid" sensor generates a behaviour "S_UpdateCalendar". We can use the PMR to identify which operation in the formal specification describes this behaviour. We might then determine properties of the behaviours we want to guarantee using model-checking. For example, when the pill bottle is opened we want the calendar service to check that there is a corresponding date and time that matches (medication is due) and that we update the calendar to show this has been satisfied. While a description of model-checking Z specifications is outside of the scope of this paper, the process of determining predicates of interest from the models to match the sensors will typically follow these patterns: Identify the sensors and their behaviours; use the PMR to find the appropriate specified operations; create the predicate to model-check the behaviour in the specification (this can be done by creating invariants or using the linear temporal logic option in Pro-B/Z, see [25] for a description of this).

The second property relates to the handling and manipulation of any data by the cloud service. As we have a formal specification describing this service it provides us with the basis for reasoning about its behaviour (using any of the standard formal methods techniques we wish to adopt). In most cases, the behaviours will relate to the inputs from sensors and outputs to actuators, which are captured in the first property (above) and the third property (following). However, having a full specification also allows us to consider any other behaviours of the cloud service which may otherwise be hidden within such a design process. For example, if a medication regime includes a monthly dosage then we must ensure that calculations around months with different numbers of days (including leap years) are handled correctly.

The third property, ensuring the cloud service generates required notifications follows a similar pattern. Within the formal specification we identify the behaviours of the cloud service, and we use the PMR to identify which sensors/actuators respond to any of these behaviours. In our example the medication system as a whole responds to the calendar cloud storage and the sound actuator responds to the SoundOp. Again, detailed requirements for these behaviours can be model-checked.

The fourth property is the the same as the first, but we consider the actuators (responders) in the pmodel in this instance and check the behaviours they respond to. This may include displays and notifications in mobile apps in some instances, but for our example is related just to the "Sound" actuator which responds to the "S_Sound" behaviour. We can use this information to check properties of the operations in the specification which are responsible for describing how and when the notification is generated.

The fifth property considers inputs to the system that are not sensor-based (for example user-entry via a mobile app). These are straightforward as they follow typical Model Based Testing processes for mobile apps more generally and are not specific to just IoT systems.

VII. Discussion

The example we have given above demonstrates the feasibility of generating the models from the design cards. It also shows how the models may then be used to consider properties of the design which may relate to requirements generally and safety properties in particular. Of course there is much more to say about the use of formal models for these types of activities. While demonstrating examples of this is beyond the scope of this paper it is worth revisiting the process of refinement of the high-level design (such as that given by the cards) to more substantive design artefacts such as prototypes. Translating a design, such as that given by the cards, to an implemented system, requires many steps. If we are dealing with design by non-experts (we revisit this term shortly) then there is no guarantee they have the technical knowledge or expertise to select and combine the sensors and actuators of the system, much less set up and program the cloud services. As such the designs they propose may not be implementable in the form given. In fact, this is another advantage of having formal models derived from the cards, inconsistencies may be exposed as part of the modelling, or model-checking process. Similarly, as the design is refined towards the implementation, changes will be made which may not preserve original properties of the design. Having formal models with embedded refinement processes (see [37], [38] for some examples) will support this transition.

We use the term "non-experts" in different contexts within this paper with different meanings. There are people who may be hobbyists or makers with a desire to create IoT systems who have no expertise in any of the technical skills typically required (programming, soldering, selecting hardware components and managing power differentials etc.) While the cards enable them to articulate their design ideas, the subsequent models will be of little interest (or benefit) to these users.

However, there are also designers who do have some of these technical skills, but are non-experts in formal modelling. The ability to work with end users (from the first group) via the cards and then have formal models provides them with the ability to include the structure that formal methods can provide to their development. Thirdly there are technical experts with experience in formal modelling, but they themselves may be non-experts in design or user-centred processes. The work we describe here gives them a shared vocabulary (via the design to models process) with the rest of the design team and enables them to validate the proposed systems.

The examples we have presented here are all necessarily small with few components. As the size and complexity of envisaged systems grows there is a danger that end-users will not be able to use the cards to capture all of the complexity. The models we have used here all have their own strategies for managing size and complexity (reported in a range of literature) and some of these are suitable to tackle this problem. For example, using partial models at different levels of abstraction to focus on specific areas of systems rather than trying to model everything in a monolithic fashion addresses many of the concerns that arise (although this requires appropriate mechanisms to combine different models within later parts of the development process [27]). In the same way, using the cards to design specific parts of a system in isolation provides the same benefits, and also contributes to the creation of these partial and more lightweight formal models.

VIII. Conclusions

In this paper we have presented a technique for using informal design artefacts, in the form of ideation card decks, to create formal models of an IoT application. We have demonstrated that particular models can be adapted to describe IoT systems, and that it is possible to generate these models from a card design using an automated process within the C2M tool. We have also given examples of some of the benefits this provides and outlined others.

There are some limitations inherent in the work as proposed here. The way in which models are generated from the card design does place limitations upon what a designer should or should not with the cards. While this is to be expected (in order to generate a model, we need to be able to somewhat control or define the format of that model) it may have implications on the design process. This can also be addressed through a "tidying up" or refinement of card-based designs as a prior step to model-generation. It would be interesting to further explore both of these options to identifying advantages or disadvantages of either. At present the model-generation is limited to one set of design cards only, however as we have other card sets implemented within the tool extending the transformation rules to include these is ongoing work.

So far our work has focused on the process from the designers' perspective, ensuring the tool can support the ideation activities and then the models can be generated. The next steps will be to look at the models and work with model-checking experts to understand the implications of the types of models that are generated to determine if further improvements are required. This will then enable us to move to end-end testing of the process.

References

[1] G. Tanganelli, C. Vallati, and E. Mingozzi, "Rapid prototyping of iot solutions: A developer's perspective," *IEEE Internet Computing*, vol. 23, no. 4, pp. 43–52, 2019.

[2] S. Daly, C. Seifert, S. Yilmaz, and R. Gonzalez, "Comparing ideation techniques for beginning designers," *Journal of Mechanical Design (1990)*, vol. 138, no. 10, 2016.

[3] M. Fiadotau and M. Sillaots, "Comparing ideation techniques for games education," *International Conference on Game Jams, Hackathons and Game Creation Events 2020*, pp. 22–25, 2020.

[4] J. Bowen and A. Hinze, "Reasoning about interactive systems in dynamic situations of use," in *The Handbook of Formal Methods in Human-Computer Interaction.* Springer Publishing Company, Incorporated, 2017, pp. 319–341.

[5] J. Bowen and S. Reeves, "Generating obligations, assertions and tests from ui models," *Proceedings of the ACM on Human-Computer Interaction*, vol. 1, no. EICS, pp. 1–18, 2017.

[6] J. Turner, J. Bowen, and S. Reeves, "Model-based testing of interactive systems using interaction sequence," *Proceedings of the ACM on Human-Computer Interaction*, vol. 4, no. EICS, pp. 1–37, 2020. [Online]. Available: https://doi.org/10.1145/3397873

[7] P. Masci, P. Oladimeji, P. Mallozzi, P. Curzon, and H. Thimbleby, "Pvsio-web: mathematically based tool support for the design of interactive and interoperable medical systems," *EAI Endorsed Transactions on Collaborative Computing*, vol. 2, no. 7, pp. 42–44, 2016. [Online]. Available: https://doi.org/10.4108/eai.14-10-2015.2261720

[8] C. Martinie, P. Palanque, E. Bouzekri, A. Cockburn, A. Canny, and E. Barboni, "Analysing and demonstrating tool-supported customizable task notations," *Proc. ACM Hum.-Comput. Interact.*, vol. 3, no. EICS, Jun. 2019. [Online]. Available: https://doi.org/10.1145/3331154

[9] C. Prehofer and L. Chiarabini, "From internet of things mashups to model-based development," pp. 499–504, 2015. [Online]. Available: https://dio.org/10.1109/COMPSAC.2015.263

[10] B. Weyers, J. Bowen, A. Dix, and P. Palanque, "State of the art on formal methods for interactive systems," in *The handbook of formal methods in human-computer interaction*. In The Handbook of Formal Methods in Human-Computer Interaction. Springer International Publishing AG, 2017, pp. 3–57. [Online]. Available: https://doi.org/10.1007/978-3-319-51838-1

[11] S. Mellor, A. Clark, and T. Futagami, "Model-driven development," *IEEE Software*, vol. 20, no. 5, p. 14, 2003. [Online]. Available: https://doi.org/10.1109/MS.2003.1231145

[12] P. Masci, P. Mallozzi, F. D. Angelis, G. Serugendo, and P. Curzon, "Using pvsio-web and sapere for rapid prototyping of user interfaces in integrated clinical environments," *EAI Endorsed Transactions on Collaborative Computing*, vol. 2, no. 7, pp. 42–44, 2015. [Online]. Available: https://doi.org/10.4108/eai.14-10-2015.2261720

[13] A. Pintus, D. Carboni, and P. A, "Paraimpu," *Proceedings of the 21st International Conference on World Wide Web*, pp. 401–404, 2012. [Online]. Available: https://doi.org/10.1145/2187980.2188059

[14] X. Nguyen, H. Tran, H. Baraki, and K. Geihs, "Frasad: A framework for model-driven iot application development," *IEEE 2nd World Forum on Internet of Things (WF-IoT)e*, pp. 387–392, 2015. [Online]. Available: https://dio.org/10.1109/WF-IoT.2015.7389085

[15] C. de Farias, I. Brito, L. Pirmez, F. Delicato, P. Pires, T. Rodrigues, I. dos Santos, L. Carmo, and T. Batista, "Comfit: A development environment for the internet of things," *Future Generation Computer Systems*, vol. 75, p. 128,Äì144, 2017.

[16] A. Souri and M. Norouzi, "A state-of-the-art survey on formal verification of the internet of things applications," *Journal of Service Science Research*, vol. 11, no. 1, pp. 47–67, 2019. [Online]. Available: https://doi.org/10.1007/s12927-019-0003-8

[17] M. Krichen, "Improving formal verification and testing techniques for internet of things and smart cities," *Mobile Networks and Applications*, 2019. [Online]. Available: https://doi.org/10.1007/s11036-019-01369-6

[18] C. Bernaschina, S. Comai, and P. Fraternali, "Ifmledit.org: model driven rapid prototyping of mobile apps," *IEEE/ACM 4th International Conference on Mobile Software Engineering and Systems (MOBILESoft)*, pp. 207–208, 2017.

[19] A. E. Khaled, A. Helal, W. Lindquist, and C. Lee, "Iot-ddl–device description language for the "t" in iot," *IEEE Access*, vol. 6, pp. 24 048–24 063, 2018.

[20] M. Hussein, S. Li, and A. Radermacher, "Model-driven development of adaptive IoT systems," in *2017 MODELS Satellite Event*, B. L., G. E., D. R. D., J. J., Ghosh, G. J., C. P., F. M., Kokaly, J. R. J., G. M., B. N., C. P.J., Collet, and P. A., Eds., vol. 2019. Austin, United States: CEUR-WS, Sep. 2017, pp. 17–23.

[21] C. M. Sosa-Reyna, E. Tello-Leal, and D. Lara-Alabazares, "An approach based on model-driven development for iot applications," in *2018 IEEE International Congress on Internet of Things (ICIOT)*, 2018, pp. 134–139.

[22] F. Ciccozzi and R. Spalazzese, "Mde4iot: Supporting the internet of things with model-driven engineering," in *Intelligent Distributed Computing X*, C. Badica, A. El Fallah Seghrouchni, A. Beynier, D. Camacho, C. Herpson, K. Hindriks, and P. Novais, Eds. Cham: Springer International Publishing, 2017, pp. 67–76.

[23] M. D. Harrison, P. Masci, J. C. Campos, and P. Curzon, "Verification of user interface software: The example of use-related safety requirements and programmable medical devices," *IEEE Trans. Hum. Mach. Syst.*, vol. 47, no. 6, pp. 834–846, 2017.

[24] P. Masci and C. A. Muñoz, "An integrated development environment for the prototype verification system," in *Proceedings Fifth Workshop on Formal Integrated Development Environment, F-IDE@FM 2019, Porto, Portugal, 7th October 2019*, ser. EPTCS, R. Monahan, V. Prevosto, and J. Proença, Eds., vol. 310, 2019, pp. 35–49.

[25] J. Bowen and S. Reeves, "Modelling safety properties of interactive medical systems," in *ACM SIGCHI Symposium on Engineering Interactive Computing Systems, EICS'13, London, United Kingdom - June 24 - 27, 2013*, P. Forbrig, P. Dewan, M. Harrison, and K. Luyten, Eds. ACM, 2013, pp. 91–100.

[26] J. Turner, J. Bowen, and S. Reeves, "Model-based testing of interactive systems using interaction sequences," *Proc. ACM Hum. Comput. Interact.*, vol. 4, no. EICS, pp. 85:1–85:37, 2020.

[27] J. Bowen and S. Reeves, "Combining models for interactive system modelling," in *The Handbook of Formal Methods in Human-Computer Interaction*, B. Weyers, J. Bowen, A. J. Dix, and P. A. Palanque, Eds. Springer International Publishing, 2017, pp. 161–182.

[28] D. Peters, L. Loke, and N. Ahmadpour, "Toolkits, cards and games, a review of analogue tools for collaborative ideation," *CoDesign*, pp. 1–25, 2020.

[29] R. Roy and J. P. Warren, "Card-based design tools: A review and analysis of 155 card decks for designers and designing," *Design Studies*, vol. 63, pp. 125–154, 2019.

[30] A. Munoz, U. Florin, Y. Eriksson, Y. Yamamoto, and K. Sandstrom, "The karakuri card deck." In Proceedings of the International Conference on Engineering Design, 2020, p. 807.

[31] S. Mora, F. Gianni, and M. Divitini, "Tiles: a card-based ideation toolkit for the internet of things," 2017. [Online]. Available: https://doi.org/10.1145/3064663.3064699

[32] J. Bowen and S. Reeves, "Formal models for user interface design artefacts," *Innov. Syst. Softw. Eng.*, vol. 4, no. 2, pp. 125–141, 2008.

[33] M. C. Henson, M. Deutsch, and S. Reeves, *Z Logic and Its Applications*. Springer: Monographs in Theoretical Computer Science. An EATCS Series, 2008, pp. 489–596.

[34] S. Jaidka, S. Reeves, and J. Bowen, "A coloured petri net approach to model and analyze safety-critical interactive systems," in *26th Asia-Pacific Software Engineering Conference, APSEC 2019, Putrajaya, Malaysia, December 2-5, 2019*. IEEE, 2019, pp. 347–354.

[35] Y. C. Law, W. Wehrt, S. Sonnentag, and B. Weyers, "Generation of information systems from process models to support intentional forgetting of work habits," in *Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems, EICS 2017, Lisbon, Portugal, June 26-29, 2017*, J. C. Campos, N. Nunes, P. Campos, G. Calvary, J. Nichols, C. Martinie, and J. L. Silva, Eds. ACM, 2017, pp. 27–32.

[36] T. R. Silva, M. Winckler, and H. Trætteberg, "Ensuring the consistency between user requirements and GUI prototypes: A behavior-based automated approach," in *Human-Computer Interaction - INTERACT 2019 - 17th IFIP TC 13 International Conference, Paphos, Cyprus, September 2-6, 2019, Proceedings, Part I*, ser. Lecture Notes in Computer Science, D. Lamas, F. Loizides, L. E. Nacke, H. Petrie, M. Winckler, and P. Zaphiris, Eds., vol. 11746. Springer, 2019, pp. 644–665.

[37] J. Bowen and S. Reeves, "Refinement for user interface designs," *Formal Aspects Comput.*, vol. 21, no. 6, pp. 589–612, 2009.

[38] ——, "Supporting multi-path UI development with vertical refinement," in *20th Australian Software Engineering Conference (ASWEC 2009), 14-17 April 2009, Gold Cost, Australia*. IEEE Computer Society, 2009, pp. 64–72.

[39] C. Sintoris, I. Mavrommati, N. Avouris, and I. Chatzigiannakis, "Out of the box: using gamification cards to teach ideation to engineering students," *Ambient Intelligence*, vol. 11249, pp. 221–226, 2018. [Online]. Available: https://doi.org/10.1007/978-3-030-03062-9_17