Noname manuscript No.
(will be inserted by the editor)

Is my configuration any good: checking usability in an interactive sensor-based activity monitor

Muffy Calder · Phil Gray · Chris Unsworth

the date of receipt and acceptance should be inserted later

Abstract We investigate formal analysis of two aspects of usability in a deployed interactive, configurable and context-aware system: an event-driven, sensor-based homecare activity monitor system. The system was not designed from formal requirements or specification: we model the system *as it is*, in the context of an agile development process. Our aim is to determine if formal modelling and analysis can contribute to improving usability, and if so, which style of modelling is most suitable. The purpose of the analysis to inform configurers about about how to interact with the system, so the system is more usable for participants, and to guide future developments.

We consider redundancies in configuration rules defined by carers and participants, and the interaction modality of the output messages. Two approaches to modelling are considered: a deep embedding in which devices, sensors and rules are represented explicitly by data structures in the modelling language and nondeterminism is employed to model all possible device and sensor states, and a shallow embedding in which the rules and device and sensor states are represented directly in propositional logic. The former requires a conventional machine and a model-checker for analysis, whereas the latter is implemented using a SAT solver directly on the activity monitor hardware. We draw conclusions about the role of formal models and reasoning in deployed systems, and the need for clear semantics and ontologies for interaction modalities.

Corresponding author Muffy.Calder@glasgow.ac.uk

School of Computing Science, University of Glasgow, Glasgow G12 8RZ, UK

1 Introduction

We investigate the analysis and automatic checking of usability in a deployed configurable, interactive, context-aware, system. Our aim is to determine if formal modelling and analysis can contribute to improving usability, and if so, which style of modelling is most suitable. The system is event-driven and sensor-based: it provides activity monitoring in a homecare setting.

The system was not designed from formal requirements or specification, so we model the system *as it is*, in the context of an agile development process. The purpose of the analysis to inform configurers about how to interact with the system, so the system is more usable for participants, and to guide future developments.

We consider two aspects of interaction: configuration of the system by rules defined by carers and participants, and the interaction modality of the output messages, for a given configuration. We detect rule redundancy, which is relevant because the the system imposes a small, finite number of rules, and the simultaneous use interaction modalities for outputs from the system, as defined by the rules, which may be confusing to a participant. Analysis is based on formal models that include representations of the devices and sensors, and representations of user configurations specified by a finite set of rules. Two different approaches to modelling are investigated. Since a key feature of the system is user-defined *rules*, we investigate two styles of representation taken from theorem proving and programming languages. The first is a *deep* embedding, in which one language or logic is represented in the data structures of another, and the second is a *shallow* embedding, which is simply a syntactic translation from one language to another. In our *deep* embedding, the devices, sensors and rules are represented explicitly by data structures

in the Promela language (the specification language of the model checker SPIN [Hol03], and non-determinism is used to represent all possible device and sensor states. In our *shallow* embedding, the devices, sensors and rules are represented explicitly in propositional logic. Reasoning about the former involves formulating temporal logic properties and exploration of the underlying state space using the model-checker SPIN; reasoning about the latter involves solving the model using a SAT solver [ES03].

The system we consider is the MATCH Activity Monitor (hereafter, referred to as MAM) system, an experimental platform built on top of the MATCH homecare infrastructure [MG09, Tur12]. The MATCH project (http://www.match-project.org.uk) is a collaborative research project focussing on technologies for care in the home. The MAM is a typical example of an activity awareness system [MdRM09] that allows groups of users to share information about their current status or recent activities. The system detects sensed activities such as movement in rooms, on equipment (e.g. kettle), or by individuals, and it delivers messages to participants and carers such as text messages, a wide variety of sounds and vibration alerts.

Configuring the monitoring tasks is an essential part of sensor-based monitoring of the participants' activities and state. For example, carers reconfigure the systems regularly, to take into account changes in the participants' medical condition, their home situation, and consequent changes to the services and sensors. For these reasons, system configuration is treated as an ongoing process throughout the lifetime of a system, just as we would consider any other kind of interaction or context change.

The system was not designed with verification in mind: there is no formal specification or design, nor are there formal requirements. But, after the system was deployed, it became clear that configurability was an issue for some users and help for them was required, as well as evidence to guide the designers when considering future improvements. The aim of this work is to investigate how formal modelling and analysis can contribute to the provision of that help, by determining better ways to configure the system and configurations that are more usable.

The main contribution of this paper is two different style of modelling of the MAM system and formulation of properties that essentially allow one to ask the the question *is a configuration any good?* More specifically, we consider:

- *are all the rules necessary, are some rules redundant or unnecesarrily complex?*
- *are users being constrained by the number of rules allowed because there redundancies in the rule set?*
- *how does the sytem notify an agent (e.g. a carer, or the cared person, or another automated system) in the event of a specific action or activity and is the notification unambiguous? For example, is it possible to deliver multiple speech outputs at the same time, or to deliver multiple tactile outputs simultaneously?*

We investigate two different approaches to modelling, and in both cases consider detecting rule redundancies and checking for confusing use of interaction modalities. The deep embedding involves developing a model of the entire event-driven system, which includes modelling the current rule set and events that change the status of input devices; the shallow embedding involves modelling the logic of the device and sensor states and rules directly as disjunctive clauses. Whereas in the former rules are represented *in* the modelling language, in the latter, the rules *are* the model. In the latter, reasoning is performed directly on the MAM hardware, whereas the former requires a conventional PC. In both cases parts of the model are generated automatically from the MAM system or logged data about configurations.

The paper is organised as follows. In section 2 we give an overview of the MAM system and in section 3 we discuss how rule redundancy and interaction modality can affect usability. In section 4 we outline our approach to modelling the MAM. In section 5 we give an overview of the deep embedding in the Promela language and in section 6 we give an overview of the shallow embedding in propositional logic. In the following section we discuss how to generate the models and then check properties, using SPIN to reason about the Promela model and a SAT solver to reason about the logic model. Discussion follows in Section 8 and an overview of related work follows in Section 9. Conclusions and future work are in Section 10.

2 The MATCH Activity Monitor

The MAM system consists of one or more hubs that carries out a finite number activity monitoring tasks, supported by a rich set of sensor-generated events and actuated events that are defined as message types. The messages may be have different modalities: text, audio, tactile, etc., The monitoring tasks are defined by a (finite) set of rules.

Each hub is connected to a set of satellites and other hubs (see architecture diagram in Figure 1). Typically, a hub resides in the home of a person requiring care (i.e.

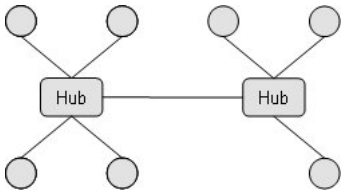

Fig. 1 MAM System Architecture.

Fig. 2 A MAM hub displaying digital photo.

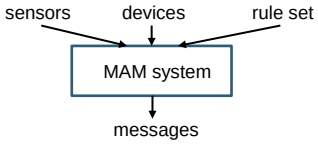

Fig. 3 MAM hub.

a participant) while the satellites are used by carers, clinicians, family and friends. If a user is not interacting with a MAM hub, it operates a digital photo frame application that displays a photo supplied by a user (see Figure 2) in order to make it a non-intrusive part of the participant's home.

Sensors and output messaging devices can be attached to everyday items like coffee cups, bathtubs and doors: for example, the system uses a Jake sensor pack [JAK] for simple movement sensing, and the Shake sensor pack [SHA], for richer sensing capabilities and tactile feedback.

A MAM hub (Figure 3) supports a set of up to eight monitoring tasks, each of which involves the generation of messages based on user-generated and sensor-generated input indicating an event or activity. Monitoring tasks are defined by rules that specify an event or activity to be reported plus the destination and form of the reporting message. Typical rules might state that when a calendar request from a given user to the cared person is received, notify the home by a doorbell earcon[1] on a speaker in the lounge, or the use of a coffee cup (captured via a Jake) in the home of the cared person, should be reported to a carer (i.e. the hub in the home or office of the carer) via a speech message.

[1] A short meaningful audio segment.

The MAM system supports a variety of data sources, message destinations and message interaction modalities (e.g. speech, graphics, tactile, etc.). A list from the current prototype is given in Figure 4.

2.1 Configuration

Each MAM hub can be configured to support up to eight monitoring tasks. This limitation, imposed by the designers, was intentional and is based on empirical evidence, to limit the complexity of the application. Each monitoring task is specified explicitly as a monitoring rule. In addition to simple <input source> <destination, modality> rules, it is also possible to specify combinations of inputs (e.g. a button press or an appointment) or message modalities (e.g. speech and graphics). Rules may also have a guard condition; currently the MAM only supports a *location* condition such that the message is sent if someone is sensed near a specified location.

A user may also choose a system-generated recommendation of the input, destination or modality. The recommendation can be used in an automatic or semi-automatic mode. In the former case, the system will choose the input, destination or modality most commonly associated with the other parameters, based on a history of logged configurations. In the latter case, the system will offer a ranked list of choices, based again on frequency of association, from which the user must select one.

A user interface is supplied for configuration. To configure the hub, a user touches the screen and the photo application fades away, replaced by the MAM application, from which the configuration screen is accessible. Figure 5 shows a typical rule configuration. Note the eight tabs to the left, one for each rule; rule 1 is selected. The rule configuration view is divided into a left-hand panel for specifying input and a right-hand panel for destination and modality. In this case the blue and red buttons on my hub (left-hand panel) have been selected to create messages to be sent to Lucy's machine(s) (right-hand panel). The large vertical green button on the right of the panel is the on-off toggle switch for the rule; when green, the rule is active and when red the rule is inactive. Figure 6 indicates the full functionality of the configuration screen. We note that the user interface to the MAM configuration subsystem (i.e. rule specification) is itself a subject of research and is not considered in this paper.

Even with the rather limited set of inputs, destinations and message types, the configuration space (i.e. number of different possible rule sets) is huge and not

Input Sources	
Calendar	An online calendar scheduling system reports upcoming appointments.
Accelerometer	Small custom-built Bluetooth accelerometers can be placed around the home (e.g. on a phone, teacup, or door) or on a person, in order to detect movement-signalled activity of the instrumented thing/person. This is performed using JAKE and SHAKE devices [WMSH07].
Webcam movement events	Fixed and wireless webcams can be used to provide motion detection. This allows for room occupancy to be detected and reported.
User-generated text	Users can key in their current activity, mood or needs explicitly using an on-screen keyboard.
Abstract Buttons	A user may select an "abstract button" to which no particular meaning has been assigned in advance by the developers of the system (i.e. "the red square") The user may negotiate with other people to assign a particular meaning to these buttons. This concept is derived from MarkerClock [RM07] that uses a similar abstract marker feature.
Message Destinations	
Local Hub	Messages are directed to one of the output devices associated with the local machine.
Registered Users	Messages are directed to specified users; the message will be sent to their hub, if they have one, or to their registered web-based client(s).
Modalities	
Graphical	Notice of an activity is briefly overlaid on top of the hub photoframe; an icon indicates that there is an unread message waiting. Additionally, the message will be added to a scrollable list of messages that is permanently available.
Speech	The content of the message is rendered into VoiceXML and played through any of the device's speakers.
Non-speech audio	A selection of auditory alerts is provided, such as "nature" sounds and "animal" sounds as well as more familiar "alert" noises. Each set of sounds contains multiple .wav files, each of which is mapped to a particular type of alert. As with speech, this can be directed to any distinct speaker.
Tactile	The Shake device (but not the Jake) is equipped with an inbuilt vibrotactile actuator that can be activated. Vibration "profiles" (i.e. vibrate fast-slow-fast, slow-fast-slow) can be used to distinguish between different types of activity.
Email	Predefined activity messages can be delivered to one or more email addresses that the user can specify.

Fig. 4 MAM Activity Monitoring Task Parameters

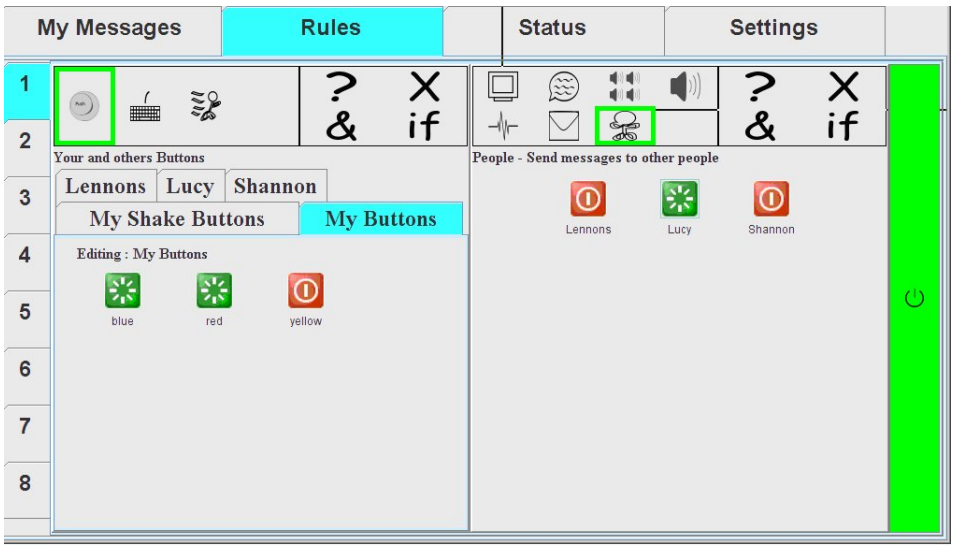

Fig. 5 Sample task configuration screen.

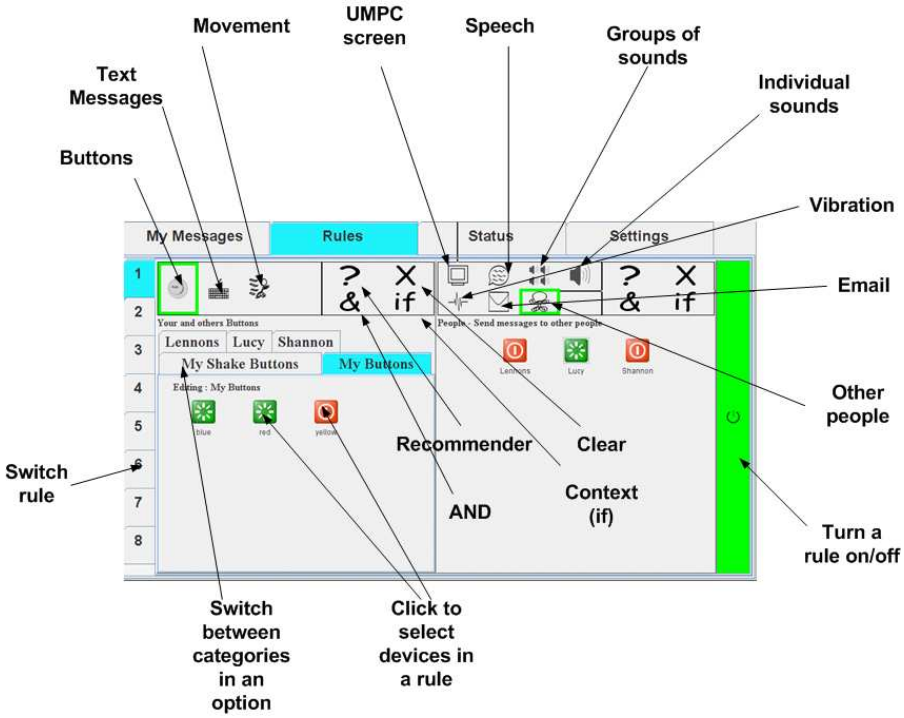

Fig. 6 Sample task configuration screen: functionality.

all configurations are desirable. In particular, configurations may be unacceptable or ineffective for users because of the inappropriate use of certain interaction modalities for particular users or situations, or the rules may be inefficient themselves, due to redundancy.

3 Usability

We consider interaction and usability in the context of users being constrained to define only a small, finite number of rules, and the confusion that can arise from rule overlappings, redundancies, and the interaction modalities of output messages.

3.1 Redundant rules

Rules are typically added to the system by non-expert users, or at least by occasional users. It is possible to create redundant rules, through overlappings or repetition, which may confuse the user about the meaning of the rule set and also unnecessarily constrain the number of (useful) rules that can be defined (recall, the MAM interface allows up to 8 rules). This conjecture was confirmed by the MAM developers during user evaluation studies: they found that many test subjects indicated they had trouble understanding complex rules – many subjects only wanted to define disjoint (sets of) simple rules but found they had unintentionally defined more complex rules.

We define a rule to be redundant if it can be removed from the system without affecting how the system operates. We do not aim to remove such rules automatically, in most cases, further interaction with the user is required to determine which rule should be removed. For example, while it might be most efficient (from the point of view of minimising the number of rules) to retain the most general rule (if one exists), a user may wish to retain a particular redundant rule because they anticipate later changes to the rule set, or because different subsets of rules are defined by different types of user (e.g. nurse, family member) and that provenance needs to be preserved. The implementation and analysis of the best form of interaction with the user is an issue for the MAM design team, we have therefore focussed on providing the information about which rules are redundant, and why.

3.2 Interaction modalities

Output devices are classified according to user interaction modalities. For example, earcons and speech are audio, screen pop-ups and text messages are visual and vibration alerts are tactile. The usability of a configuration may depend on the appropriate use of the different modalities. But this is a subtle and context-dependent

concept, subject also to personal preference. There are a number of dimensions to consider, for example, the use of interaction modality may indicate priority or urgency, e.g. audio may indicate higher priority than visual. Or, there may be standard ways of using modalities for certain users. For example, always use visual to deliver a message about medication use to a participant (unless they are visually impaired). Visual output devices should be avoided for severely visually impaired participants, though, it may be appropriate to visual to notify other participants and carers.

The dimension we consider here is one that is less personal: the simultaneous use of the same interaction modality for more than one message. For example, simultaneous audio outputs, e.g. two speech outputs, or two earcons such as an animal sound and a door bell, may confuse a participant and result in the loss of messages. The MAM offers a rich set of earcons and so this type of confusion is quite likely. Overuse of tactile interaction e.g. vibration, may also result in the user being unable to differentiate between two different messages.

Consider the following five MAM modalities: graphical, speech, non-speech, tactile, and email. We could find no standard taxonomy or ontology of interaction modalities or guidance for their use in the literature to inform our analysis. But based on user trials, we propose that, at least in the MAM system, it is not a problem to use graphical or email modality for simultaneous delivery of multiple messages – because messages can be delivered sequentially, with an appropriate additiona message indicating there are more messages to be uncovered/read. But, the audio and tactile modalities *should not* be used for multiple messages simultaneously because to do so can be confusing to the participant. We conclude it is acceptable to deliver messages in different interaction modes, simultaneously, but that messages should not be delivered simultaneously in the same mode, for some modes.

The exact ontology of interaction modes is critical in our models; for example, we have chosen to distinguish between the audio modalities of speech and non-speech. We note this may may be too fine-grained from a usability point of view, or indeed, not fine-grained enough. For example, we could distinguish between animal earcons, {lion, elephant, dog} and nature earcons {wave, forest, wind}. In the future, further empirical investigation of the MAM system may inform a different ontology.

4 Modelling the MAM

Our approach is based on the principle that we model the system as it *is implemented*, rather than an idealised view of what it *should* do. To this end, we derived configuration-parameterised models from the actual deployed system through a combination of code inspection, analysis of observed behaviour, and interviews with the designers. The parameter (the configuration) is instantiated automatically from internal data or logfiles.

The key aspects of the MAM system from a modelling perspective are it is event-driven and rule-based. Events include (but are not restricted to) direct user interaction with the hub, such as pushing buttons and editing rules, and indirect user interaction such as movement captured by a webcam or external actions such as messages received from other users. The rules describe how activities are monitored, and so define how the system reacts to events, which may include changes to the current set of rules. Thus after every event, the rules are applied. Rules are defined via the graphical interface and stored internally as instances of pre-defined (parameterised) *evaluation functions* that return input or output devices. Evaluation functions are the basic building blocks of the MAM system, and so they are also fundamental to the model(s).

Each rule consists of an input and an output evaluation function, and a list of parameters. Both input and output functions can be themselves compositions of other (evaluation) functions. Composition of input evaluation functions is interpreted as a disjunction, meaning that an event will be triggered if either evaluation function is true, whereas the composition of output functions is a conjunction, meaning that if the conjunction of the output functions is triggered, then the result of both evaluation functions will be used.

The MAM system designers did not define a textual format for user defined rules, but during the modelling process we found it helpful to have one and so we implemented a small procedure to extract textual rules from internal evaluation functions. An example rule set thus derived is shown in Figure 10.

We note that while users interact with the system, not only as the subjects of sensing and communication, but as active participants in the configuration process, users are not modelled explicitly. Rather, we represent explicitly the events that could occur as a consequence of their actions. This means we make no distinction between user interaction and any other change of context: while there may be intent associated with the former, from a modelling point of view both are simply aspects of state that may be captured by propositions (whose validity may be temporal).

In the remainder we assume a single hub that can take input from one or more satellites or additional hubs, and a single rule set, though it would be a simple

matter to extend the model to include multiple hubs and rule sets and dynamic rule sets. We investigate two different styles of modelling, the first is a deep embedding in the specification language Promela: a high level language for specifying concurrent processes, communication, and data types, and the second is a shallow embedding in standard propositional logic.

5 A deep embedding in Promela

In the deep embedding model in the Promela language, the hub is represented by a data type that encapsulates the input and output device types. An example of a simple hub data type with four buttons (coloured and Shake), a single movement sensor (Jake), a movement actuator (Shake), and an (audio) speaker, is shown in figure 7.

The input and output sensors and devices and the current rule set are all represented by processes, which are composed concurrently thus:

$$sensor_1 \| sensor_2 \| \ldots \| device_1 \| device_2 \ldots \| rules$$

A coordination mechanism is required to ensure that the rules are applied repeatedly, after *every* event. We use a global boolean variable (called *event*) for coordination, i.e. to indicate whether or not an event can be processed.

```
typedef hub{
     bit red;
     bit yellow;
     bit blue;
     bit shake_in_b;
     chan audio_out = [5] of { mtype };
     chan screen_popup = [5] of { mtype };
     chan screen_list = [5] of { mtype };
     byte webcam;
     byte jake_in_m;
     byte shake_out;
     mtype text_in }
```

Fig. 7 hub datatype.

In the following sections, we give specifications of some example input and output devices/sensors, and rule sets.

5.1 Inputs

Each input device/sensor is represented by a process and a corresponding global variable. For example, a button press is represented by a single bit variable and a process that nondeterministically assigns the values 0 or 1, as given in Figure 8. Note the use of atomicity to ensure that the *event* synchronisation variable is updated in the same computation step.

```
proctype button()
{do
 if event  ->
   :: atomic{red_button != 0 -> red_button=0;event=0}
   :: atomic{ red_button != 1 -> red_button=1;event=0}
  fi
od}
```

Fig. 8 Example of a button process.

Movement sensors such as a Jake, Shake or webcam, are represented as an integer variable, and as above, their respective processes nondeterministically assign values to that variable. In the current prototype, for the movement sensors, we have chosen the abstraction: 0 represents no movement and 1, 2 and 3 represent low, medium and high levels of movement respectively. Text based inputs, such as messages from other hubs, are represented as an mtype variable (an mtype is a Promela enumerated type). Again, the associated processes assign values nondeterministicaly.

Together, these processes act as sources of events for the system, whereas output devices act as sinks.

5.2 Outputs

Each output device is represented as a process with an associated global variable or channel to which messages are written. The process continuously polls for a value being assigned to the variable/written to the channel, and then it resets the variable or reads the message off the channel. For example, the speaker on the hub is modelled as a channel and a process as given in Figure 9.

```
proctype speaker(chan in_chan)
{mtype audio_file;
  do
 if event ->
   :: atomic{in_chan?audio_file; event = 0};
 fi
 od}
```

Fig. 9 Example of a speaker process.

5.3 Rules

The rule set is represented by a single process, and each rule is represented by a conditional statement, consist-

ing of a guard and a compound statement. More precisely, we represent a rule as a single statement $C \rightarrow A$, where C is a guard statement made up of a disjunction of statements representing the condition of the rule and A is compound statement consisting of a sequence of statements representing the action. For example, the rule "when the red console button is pressed play the doorbell earcon on the hub speaker" maps to the Promela statement
$(this.red > 0) \rightarrow this.audio\_out!earcon\_doorbell$. Rules can also be context sensitive. For example, "If the red button is pressed then, if the webcam has recently detected movement inform me with synthesised speech else send me an e-mail". In this case the definition of recent is a system parameter, which we represent by a global variable.

The Promela representation of the example rule set is in Figure 11. In this example there are two hubs, one that belongs to the user and one that belongs to Bill. The user's hub is referred to by the global variable *this* and Bill's hub by *Billh* (both have type *hub* as defined in Figure 7). Again note the use of atomicity to ensure that *all* the rules are applied in one computation step.

The representation of a rule set in Promela is generated automatically from within the MAM system, or from logfiles of configurations.

5.4 Interaction modality

We do not represent the interaction modality of each output message explicitly, but rather the *counts* of the usage of specific modalities, which is implemented by the addition of simple counters. We track only the interaction modalities speech, non-speech,and tactile, represented by the counters *speech_count*, *nspeech_count*, *tactile_count* respectively. These are all initially set to 0, and incremented in the (atomic) action sequence of the rules. They are reset to 0 after the atomic step. For example, the rule set from Figure 11 is augmented as shown in Figure 12.

6 A shallow embedding in propositional logic

We now turn our attention to a shallow embedding using propositional logic. In this representation, the rules, sensors and device states, and modality counts are modelled by literals and disjunctive normal form clauses.

6.1 Literals - inputs and outputs

Each simple input type is represented by a single literal. For example, a literal represents a button press or receipt of a message from someone. Similarly, simple output types are also represented as literals. More complex input functions such as movement, which has low, medium and high inputs, can be represented as one literal per input value. A clause then needs to be added to ensure the input values are consistent. For example, if the literal for a movement level high is true, then both medium and low should also be true. To ensure this, the clauses $low \vee \neg medium$ and $medium \vee \neg high$ are added, which will need to be done for each movement detection device. However, if only one rule takes input from a movement detection device, then the input can be treated as a simple input device and the clause can be omitted. Classes of output can be modelled in one of two ways, similar to inputs. If only one rule uses the individual elements from a class, then the class can be represented as a single literal. Otherwise, each of the class members is represented as a single literal and there is an additional literal for the class. For example, the MAM earcon class nature is represented by $wave \vee forest \vee wind \vee \neg nature$.

There are several ways to represent the use of interaction modalities by output messages. Since we do not actually need the *counts*, values can be partitioned into three classes: no use, once, and more than once. We simply introduce two boolean variables per modality, the first variable is set to true on the first use of the modality, and the second variable is set to true on the second use.

6.2 Clauses - rules

Rules are represented by literals and clauses. Assume an indexed set of rules. We represent each (indexed) rule r_i by a new literal r_i, which is set to true to indicate it has been triggered, and a set of clauses defined as follows and summarised in Figure 13. There are four types of clause associated with a rule. First, a clause $r_i \vee \neg c$ is added for each condition c that triggers rule i. Second, to ensure r_i is not true if none of its conditions are met, the following clause is added: $c_1 \vee c_2 \vee \ldots \vee c_n \vee \neg r_i$. Third, if rule i is triggered then the appropriate outputs must be set to true, thus the clause $\neg r_i \vee a$ is added for each action a associated with rule i. Finally, to ensure that actions are only taken if associate with a rule, the clause $r_1 \vee r_2 \vee \ldots \vee r_n \vee \neg a_i$ is added, where r_1 to r_n are all the rules that are associated with action a.

7 Model generation and analysis

Both the Promela and logic models depend on the given configuration; we generate each model instance auto-

1.	$\forall r_i \in R$	$\forall c \in r_i$	$r_i \vee \neg c$
2.	$\forall r_i \in R$	$\forall c_j \in r_i$	$c_1 \vee c_2 \vee \ldots \vee c_n \vee \neg r_i$
3.	$\forall r_i \in R$	$\forall a \in r_i$	$\neg r_i \vee a$
4.	$\forall a \in R$	$\forall r_j.action(r_j, a)$	$r_1 \vee r_2 \vee \ldots \vee r_n \vee \neg a_i$

Fig. 13 Clauses required to represent a given rule set R.

matically from rules stored in the internal MAM evaluation function format. This enables us to generate models automatically from logfiles of user trials, and at runtime.

7.1 Analysis - Promela model

We used the model checker SPIN to generate the entire state space and verify LTL (linear temporal logic) properties that encode rule redundancy and interaction modality.

Rule redundancy is encoded by associating an LTL property with each rule, and then checking whether or not that property holds in the model generated without that rule. More precisely, recall that conditions (i.e. inputs) are disjunctions and actions (i.e. outputs) are conjuctions. For each Promela representation of rule r of form $X \rightarrow Y_1, \ldots Y_n$, where X is a guard and $Y_1, \ldots Y_n$ is a sequence of statements, the associated property is the postcondition associated with the compound statement $Y_1, \ldots Y_n$ (a conjunction of propositions encoding the assignments Y_i) will always eventually occur after the guard X (a disjunction of conditions) becomes true. Namely, for rule r, define a mapping f such that $f(r) = \Box(f(C) \rightarrow \Diamond f(A))$, where $f()$ maps guards and assignments to propositions in the obvious way, and $\Box$ and $\Diamond$ are the globally and eventually operators, respectively. For example, the rule $(this.yellow > 0) - > this.shake\_out = 1$ maps to the LTL property
$\Box((this.yellow > 0) \rightarrow \Diamond(this.shake\_out == 1))$. We then define a rule r in the rule set R to be redundant if for model $\mathcal{M}(R|r)$, which represents R without r, $f(r) \models \mathcal{M}(R|r)$. Namely, the consequences of X hold, even when the rule r is not in the rule set. A rule set R is defined to contain no redundant rules if $\forall r.f(r) \not\models \mathcal{M}(R|r)$.

Appropriate use of interaction modality of output messages is encoded as LTL safety properties (something "bad" does not happen), e.g. $\Box((speech\_count <= 1)$, $\Box((tactile\_count <= 1)$, etc. Specifically, given such a property ϕ, we check $\phi \models \mathcal{M}(R)$; if it is false, there is an inappropriate simultaneous use of an interaction modality in the set $\mathcal{R}$.

7.2 Analysis - logic model

We used the open-source SAT solver *miniSAT* to check satisfiability of the logic model(s), calling it from a script that generates the appropriate model. Athough SAT solving is in general NP-complete, such solvers are highly efficient for many practical applications, and this proved to be the case here.

To detect redundant rules, we solve the model once for each atomic condition in the rule, to check if atomic condition c from rule r_i is redundant. We add a clause for each input literal, setting the literal related to c to true and all the rest to false. All literals that represent the actions from rule r_i are set to true. All clauses related to the rule being checked are removed. If the resultant model is satisfiable then condition c from rule r_i is redundant, if all conditions associated with rule r_i are redundant, then r_i is redundant.

To check for inappropriate use of interaction modality, we solve the model when the second variable associated with that modality is true.

7.3 Complexity and comparison

In the deep embedding, a typical search depth is of the order 10^6 and state space of order 10^8, with verification times between 10 and 30 minutes on a conventional PC. While this representation is straightforward and can be extended easily to accommodate more devices or a different interaction taxonomy, it clearly cannot be used to deliver feedback to the configurer at run-time. In the shallow embedding, a typical model consists of order 100 literals and clauses, with each instance of the problem requiring less than a thousandth of a second to solve on a conventional PC. All instances were solved with propagation alone, no search was required. Given this stunning improvement on complexity (which is not a reflection on the SPIN model-checker but the different styles of deep and shallow embedding), we then reimplemented the entire process on the actual MAM hardware (no separate PC required). Using that hardware, reading in and checking a rule set requires approximately 5 seconds. The majority of this time is taken to read and parse the log file. Each individual SAT model requires approximately 15 thousandths of a second to solve and thus we were able to offer analysis at run-time.

8 Discussion

The aim of this paper is to determine if formal modelling and analysis can contribute to improving usability of a system in which configuration is an ongoing

process and when configured, (possibly different) users interact with the system, according to the context as perceived through a variety of input sensors and devices.

Modelling is after deployment, so the goal of analysis is to help configurers of the current system, and, in the spirit of agile development, designers of future increments. This means that we have not modelled an idealised system, but one that has been designed and engineered in the context of specific practices and personal conventions. This presents non-trivial challenges for any modelling process. We have focussed on two specific aspects of usability, not full functionality, thus our approach may be regarded as an application of formal methods *light* [JJW96].

A key question to ask is which style of modelling is most suitable. We presented two models differentiated by the style of representation: deep or shallow. But the distinction also reflects two quite different abstractions: in the former there is a clear representation of events and computation paths, whereas the in the latter the system is essentially a knowledge base. To an extent, in this application, one could argue that the state of a sensor encapsulates a set of computational paths (or at least what is required to know them) and so we do not need to study the paths themselves. Thus the knowledge base approach may be more appropriate for this type of context-aware system. In both cases, the models are straightforward. The novel aspect of this work is not the models themselves, but the process of deriving them and consideration of how analysis will be used.

How to detect and resolve redundancy depends on an agreed understanding of modalities, priorities, and more generally, context. One contribution of our formal modelling and analysis of MAM design has been to expose the need for clear semantics and ontologies for interaction modalities and context, with respect to acceptability and usability. For example, a user may not care about the simultaneous use of earcons *unless* one of them has been generated by a certain condition. Delivering messages via the television and the beeper simultaneously may be acceptable, unless one of the messages is considered significantly more urgent than the other, or has arisen because of an unsafe context. It has become clear from our discussions with the designers of the MAM that priorities on rules and messages are a likely extension, thus reinforcing the need for clear ontologies. This also illustrates a well known benefit of the formalisation process, which is not the model itself, but exposure of what needs to be clarified in order to develop a model.

We note that we have not experimented with the selection and presentation of feedback to configurers, but rather we have focussed on how to produce the analysis that will form the basis of that feedback.

Further, since we can directly incorporate logfiles into model(s), we can analyse sets of models to detect frequently occuring pitfalls and to explore design decisions concerning the ranking of recommendations. For example, currently the system makes recommendations based on historical associations. It would be interesting to investigate whether, in practice, recommended rules lead more often to redundancy or inappriate use of interaction modality. If so, it might be advisable (for the designers) to add some form of checking and filtering at the recommendation stage. Further, we might also analyse logfiles for other kinds of usability issues. For example, in at least one configuration from a logfile we noticed overlapping rules in the sense that inputs overlap (e.g. report two different messages for the same input), and in another we noted a circularity amongst rules, e.g. the message output for one rule was the input for another. This is distinct from process mining, which attempts to recover information from logfiles; here, we treat logfiles as data.

We note that while in the current prototype the configurers are humans, in future, the system might autonomously configure itself in response to a context change and so automatic checking of usability aspects would become more urgent.

9 Related Work

Modelling and reasoning about interactive, context-aware systems has been recognised as a significant challenge [CH97, CRB07] for at least two decades now. Much formal analysis is focussed on techniques for requirements involving location and resources, within a tradtional waterfall framework. For example, [CDP09] employs the Ambient calculus for requirements and [CE07] employs a constraint-based modelling style and temporal logic properties. There are various knowledge based approaches, such as the language CML (context modelling language) [HI06, BBH+10], which is based on conceptual modelling techniques for databases. There is related work developing policy conflict handling mechanisms from telecommunications systems in the MATCH project [TCW07, WT08]. Our work shares similar motivations but is complementary in that it investigates in detail how (configuration) policy rules for the MAM can be modelled and verified. Some work has been done on better integration of formal analysis techniques within the context of interactive system interfaces (e.g. [CH08]), but there is little work on analysis in the context of a more agile software development process. One exception is [RBCB08], where a model of salience and cogni-

tive load is developed and a usability property is considered. The model is expressed in a higher order logic, and the property is expressed in LTL. In some cases, analysis revealed inconsistencies between experimental behaviour and the formal model, which led them to suggest refinements to the rules and also new studies of behaviour.

Another exception is Kristoffersen's work on automatic usability assessment based on structured user interface specifications [Kri09]. This study analyses principles of usablity (consistency, synthesisability, etc.) by generating a rewriting logic model (written in the language Maude) automatically from an XUL specification of an interface, and then checking state invariants that encode some of those principles. While this work focusses on user interface design, which is not our concern, we note that two conclusions accord with ours. First, attempts at formalising design ambitions may make them more trivial. We have a similar issue: formalising rule redundancy and inappropriate use of interaction modes has resulted in somewhat trivial properties (albeit, ones that can be checked automatically and quickly). Second, while XUL has many drawbacks as a design language, it does offer many advantages compared to most other automatic usability evaluation methods based on models, because there is no "impedance mismatch". Kristoffersen refers to this as a problem when the representation of the application intended for analysis is not be an accurate image of the application. While we do not have the exact equivalent of XUL here, we have worked directly with the so-called *evaluation functions* of the MAM system, and thus we also have minimised the impedance mismatch.

We note that an early version of this work is presented in [CGU09], the main differences here are we provide more details of the distinction between deep and shallow embeddings, the purpose and derivation of the models, and modelling interaction modalities.

10 Conclusions and future work

We have reported two modelling and analysis approaches for usability aspects of a interactive, configurable, activity monitoring system.

The system was already deployed, having been designed and engineered in the context of specific practices and personal conventions. The goal of our formal analysis is to help users (configurers and participants) configure the system better, and guide designers in the context of an agile development process.

We considered redundancies in configuration rules defined by carers and participants, and the interaction modality of the output messages, for a given system configuration. We developed two different approaches to modelling. One is a deep embedding in a specification language in which devices, sensors and rules are represented explicitly by data structures in the modelling language and non-determinism is employed to model device and sensor states. The other is a shallow embedding in propositional logic in which the rules and device and sensor states are represented directly in propositional logic. The former requires a conventional machine and a model-checker for analysis, whereas the latter is implemented using a SAT solver directly on the activity monitor hardware. In both cases, the models are closely aligned with the internal evaluation functions and parts of the models are generated automatically from actual configuration data or log files. We conclude that the state of a sensor effectively encapsulates a set of computational paths and so we do not need to study the paths themselves, thus a logic based representation may be best (and most efficient) for this type of context-aware system. In both types of model, the chosen ontology of the user interaction modality is crucial and there is a strong need for clear ontologies of interaction modalities. This illustrates a well known benefit of the formalisation process, which is not the model itself, but the exposure of what needs to be clarified in order to develop a model.

Longer term, our plans for further work fall into four areas. We will carry out user trials to investigate the best forms of interaction with, and feedback to, human users, when we detect redundancies and inappropriate use of interaction modalities. We will gather empirical evidence about the whether or not recommended rules contribute to these problems. We will also investigate ways to present and use analysis results in the context of non-human agents. We will investigate incorporating aspects of stochastic user and sensor behaviour, performance, and real-time into the model and properties. For example, we could consider probabilistic abstractions of log files as a representation of the configuration process, and thus obtain a probabilisitic model of the whole system. Finally, we will further investigate ontologies for interaction modalities, especially in the context of priorities.

Acknowledgements

This research was supported by the *VPS* project *(Verifying interoperability in pervasive systems)*, funded by the Engineering and Science Research Council (EPSRC) under grant number EP/F033206/1. We also acknowledge support from the MATCH Project, funded by the Scottish Funding Council under grant HR04016.

References

[BBH+10] Claudio Bettini, Oliver Brdiczka, Karen Henricksen, Jadwiga Indulska, Daniela Nicklas, Anand Ranganathan, and Daniele Riboni. A survey of context modelling and reasoning techniques. *Pervasive and Mobile Computing*, 6(2):161 – 180, 2010.

[CDP09] A. Coronato and G. De Pietro. Formal specification of a safety critical pervasive application for a nuclear medicine department. *IEEE International Conference on Advanced Information Networking and Applications Workshops*, pages 1043–1048, 2009.

[CE07] Antonio Cerone and Norzima Elbegbayan. Model-checking driven design of interactive systems. *Electron. Notes Theor. Comput. Sci.*, 183:3–20, 2007.

[CGU09] M. Calder, P. Gray, and C. Unsworth. Tightly coupled verification of pervasive systems. *Proceedings of the Third International Workshop on Formal Methods for Interactive Systems (FMIS 2009), Electronic Communications of the EASST*, 2009.

[CH97] J.C. Campos and M. D. Harrison. Formally verifying interactive systems: a review. In *Design, Specification and Verification of Interactive Systems 97*, pages 109–124. Springer, 1997.

[CH08] J. C. Campos and M. D. Harrison. Systematic analysis of control panel interfaces using formal tools. In *XVth International Workshop on the Design, Verification and Specification of Interactive Systems (DSV-IS 2008)*, volume 5136 of *Lecture Notes in Computer Science*, pages 72–85. Springer-Verlag, July 2008.

[CRB07] P. Curzon, R. Rŭkėnas, and A. Blandford. An approach to formal verification of human-computer interaction. *Formal Aspects of Computing*, pages 513–550, 2007.

[ES03] Niklas Eén and Niklas Sörensson. An extensible SAT-solver. In Enrico Giunchiglia and Armando Tacchella, editors, *SAT*, volume 2919, pages 502–518. Springer, 2003.

[HI06] Karen Henricksen and Jadwiga Indulska. Developing context-aware pervasive computing applications: models and approach. *Pervasive and Mobile Computing*, 2:37 – 64, 2006.

[Hol03] Gerard J. Holzmann. *The SPIN model checker: primer and reference manual*. Addison Wesley, Boston, 2003.

[JAK] Jake project. http://code.google.com/p/jake-drivers/.

[JJW96] Cliff B. Jones, Daniel Jackson, and Jeannette Wing. Formal methods light. *Computer*, 29(4):20–22, April 1996.

[Kri09] S. Kristoffersen. A preliminary experiment checking usability principles with formal methods. *IEEE Second International Conference on Advances in Computer-Human Interactions*, pages 261–270, 2009. DOI 10.1109/ACHI.2009.26.

[MdRM09] P. Markopoulos, B. de Ruyter, and W. E. Mackay. *Awareness Systems: Advances in Theory, Methology and Design*. Springer, 2009.

[MG09] T. McBryan and P. Gray. User configuration of activity awareness. *Lecture Notes In Computer Science*, 5518:748–751, 2009.

[RBCB08] R. Rŭkėnas, J. Back, P. Curzon, and A. Blandford. Formal modelling of salience and cognitive load. *ENTCS*, pages 57–75, 2008.

[RM07] Y. Riche and W.E. Mackay. Markerclock: A communicating augmented clock for the elderly. *Proc. Interact 07. Part II, Lecture Notes In Computer Science*, 4663:408–411, 2007.

[SHA] Shake users group. http://www.dcs.gla.ac.uk/research/shake/.

[TCW07] Kenneth J. Turner, Gavin A. Campbell, and Feng Wang. Policies for sensor networks and home care networks. *in Mohammed Erradi (ed.), Proc. 7th. Int. Conf. on New Technologies for Distributed Systems*, pages 273–284, June 2007.

[Tur12] Kenneth J. Turner, editor. *Advances in Home Care Technologies: Results of The MATCH Project*. IOS Press, 2012.

[WMSH07] J. Williamson, R. Murray-Smith, and S. Hughes. Shoogle: excitatory multimodal interaction on mobile devices. *Proc. SIGCHI Conference on Human Factors in Computing Systems*, 4663:121–124, 2007.

[WT08] Fang Wang and Kenneth Turner. Policy conflicts in home care systems. *Proc. 9th Int. Conf. on Feature Interactions in Software and Communications Systems*, pages 54–65, 2008.

If the red or blue buttons are pressed	then	play the rocket earcon
If my webcam detects movement	then	display a pop-up message on my screen and display a message on the screen list
If I receive a message from Bill or Bill presses his red button	then	inform me using synthesised speech
If I receive a message from Bill	then	send a vibration message via the shake
If the red button is pressed	then	send a message to Bill and inform me using synthesised speech
If the jake senses movement	then	send a vibration message via the shake
If Bill presses his red button	then	inform me using synthesised speech
If the yellow button is pressed	then	send a vibration message via the shake

Fig. 10 Example rule set.

```
proctype rules()
{
do
if (event) -> atomic{
  ::(this.red == 1) || (this.blue == 1) -> this.audio_out!ec_rocket;
  ::(this.webcam > 2) ->  {this.screen_popup = me; this.screen_list = me};
  ::(this.text_in == Bill || Billh.red == 1) -> this.audio_out!speech;
  ::(this.text_in == Bill) -> this.shake_out = 1;
  ::(this.red ==1) -> {Billh.text_in = me; this.audio_out!speech};
  ::(this.jake_in_m > 1) -> this.shake_out = 1;
  ::(Billh.red == 1) -> this.audio_out!speech;
  ::(this.yellow == 1) -> this.shake_out = 1;}
event = 1;
od
}
```

Fig. 11 Promela representation of example rule set.

```
proctype rules()
{
do
if (event) -> atomic{
  ::(this.red == 1) || (this.blue == 1) -> {this.audio_out!ec_rocket; speech_count++};
  ::(this.webcam > 2) ->  {this.screen_popup = me; this.screen_list = me};
  ::(this.text_in == Bill || Billh.red == 1) -> {this.audio_out!speech; nspeech_count++};
  ::(this.text_in == Bill) -> {this.shake_out = 1; tactile_count++};
  ::(this.red == 1) -> {Billh.text_in = me; this.audio_out!speech; nspeech_count++};
  ::(this.jake_in_m > 1) -> {this.shake_out = 1; tactile_count++}
  ::(Billh.red == 1) -> {this.audio_out!speech; speech_count++};
  ::(this.yellow == 1) -> this.shake_out = 1;  tactile_count++}
event = 1; speech_count = 0; nspeech_count = 0; tactile_count = 0
od
}
```

Fig. 12 Promela representation of example rule set with interaction modality counts.