A Preference Modelling Approach to Support Intelligibility in Pervasive Applications

Johnson Fong\textsuperscript{1,2}, Jadwiga Indulska\textsuperscript{1,2} and Ricky Robinson\textsuperscript{1,2}

\textsuperscript{1}The University of Queensland
School of Information Technology and Electrical Engineering
\textsuperscript{2}National ICT Australia (NICTA)
Queensland Research Laboratory, Australia
{jfong, jaga}@itee.uq.edu.au, ricky.robinson@nicta.com.au

Abstract— Context-aware applications do not always adapt their behaviours in ways that users expect due to a variety of reasons. Applications that lack intelligibility are often incapable of offering explanations to users as to why they decided to adapt their behaviours in certain ways, and providing feedback mechanisms for users to take control of any unwanted adaptation. This can lead to loss of user trust, satisfaction and acceptance of these applications. However, providing intelligibility and user control in applications are non-trivial; it involves exposing internal working components that influence the adaptation decisions, rendering them understandable to non-technical users, and enabling user modifications to those components to correct unexpected adaptations. This paper describes a user preference model regarding application adaptations. The goal is to support intelligibility in applications by facilitating users in generation of clear mental models that enable them to understand the links between particular contextual situations and various adaptive actions. It also aims to support user control of application behaviours by assisting developers in the creation of appropriate feedback mechanisms. These are essential in preventing user frustration at erratic application behaviours.

context-awareness; intelligibility; user control; context modelling; preference modelling

I. INTRODUCTION

Context-aware applications adapt their behaviours in responses to users' current context and preferences. These applications may not always adapt or behave the way as users expect due to a variety of reasons, such as (1) users can differ greatly in terms of their requirements and expectations about how their application should behave, and (2) the context and preference information upon which they base adaptation decisions can be imperfect (e.g., out of date, ambiguous and imprecise, etc.) [11]. Hence, context-aware applications should be intelligible [2], e.g., being able to represent to their users what context or situations the applications think they have inferred, how the reasoning took place, and what actions are linked to those situations in a clear manner, etc. Applications with limited intelligibility often cannot explain to users why they decided to adapt in certain ways, and offer no mechanism for users to override any unwanted application behaviour. Such lack of system intelligibility and control can cause users to become increasingly dissatisfied and abandon the applications eventually.

This problem can be mitigated by informing users of the application capabilities and its understanding of the current context, disclosing actions taken by the application, and providing feedback mechanisms for users to take control of its behaviour. In this paper, we present our efforts to address this challenge, focusing on an approach for modelling context-dependant preferences that serve as the basis of decision-making of application adaptations. The model facilitates automatic generations of wide range of explanations recommended by Lim and Dey et al. [2], and supports user-customisable adaptation decision-making of context-aware applications. The model builds further abstractions on top of the situation abstractions and CML context modelling approach proposed by Henricksen's et al [3]. It aims to support users in generation of high level mental models and explanations that enable them to understand the linkages between particular contextual situations and various adaptive actions of the application. The model also facilitates development of feedback mechanisms for non-technical users to formulate their own preferences independently and modify the state of applications to control their behaviours.

The structure of the paper is as follows. Section 2 presents related work in the area of intelligibility and user control of application behaviours, and approaches to context and preference modelling. Section 3 identifies a set of key requirements, introduces our preference modelling approach and presents some examples for illustration purposes. Section 4 and 5 describes how the model supports intelligibility and user control of context-aware application. Finally Section 6 summarises the paper with discussions on future work.

II. RELATED WORK

In this section, we review relevant approaches in the area of intelligibility and user control of context-aware application behaviours, as well as techniques for modelling context and user preferences. The aim is to learn how to expose some of the internal applications components and decision-making processes of adaptations in a way that non-technical users can understand, so that this information can be modified in a user friendly environment.
A. Studies on intelligibility

Dey and Lim et al. performed several user studies for assessing the impact and demand for intelligibility in context-aware applications [2, 9]. Their studies found that context-aware applications providing intelligibility types such as why, certainty and control are helpful in improving users' satisfaction, and should be made available for all context-aware applications. They have implemented [1] an extension to the Context Toolkit to support some of the mentioned intelligibility types. In a similar field of research, Henricksen and Hardian et al. [11] identified a number of internal middleware components (choices, situations and user preferences) that would influence the behaviour of applications. They also extended the PACE middleware to support revelation of these components to users. However, users may not necessarily understand the information being exposed.

B. Studies on User Control

User control can be conceptualised as the level of user intervention that is required to operate a context-aware application [4]. This level of control changes depending on a wide variety of factors, such as users' need, situation and expertise, and the types of application, etc. As Van der Heijden [4] argues that the transfer of control from the user to the application would result in an increase in user anxiety when using the application. On the other hand, Barkuus and Dey [6] argue that users are willing to accept the loss of control and a large degree of autonomy from applications, as long as the application’s usefulness is greater than the cost of limited control. In general, the balance between user control and autonomy of applications often needed to be adjusted at run-time, unlike traditional applications where the trade-off can be fixed at design time.

BrDICZka et al. [12] developed a supervised learning scheme for learning situation models that enable users to provide feedback to context-aware services. A simplified situation model is initially acquired using an automatic segmentation process. The model is then adapted to accommodate user preferences and different operating environments through interactions with users using the supervised learning algorithm. Experimental results obtain from their evaluation of the algorithm are encouraging. Their future research would involve improving sensing reliability, and generating explanations for users to understand and correct wrong system perceptions.

C. Context Modelling

There exist a number of high level context modelling approaches, such as ontology based models, spatial models, object-role based model and rule-based models (e.g., using UML enriched with Object Constraint Language) [13]. As adopted in numerous projects such as [15] by Wang et al. and [16] by Ranganathan et al., the main strengths of ontology based context models are in terms of heterogeneity and interoperability. Spatial context model proposed in [17, 18] by Becker, Nicklas and Durr are well suited for location-based applications. Object-role based model has great expressive power for capturing a wide variety of context, such as temporal, historic and ambiguous information [5]. In addition to the mentioned modelling techniques, there are few hybrid approaches that attempt to combine the advantages of different modelling formalisms, such as [14] by Bettini et al. and [19] by Henricksen et al. The former represents raw context data by means of CC/PP profiles, and complex data using OWL ontology. The hybrid modelling technique of the latter is based on CML and ontology language.

D. Preference Modelling

In order to enable users to control and customise behaviours of context-aware applications, there are needs for techniques to elicit and capture user requirements and preferences. Context-dependent preferences are indispensable in supporting decision-making processes of context-ware adaptations. The importance and difficulty of accurately identifying user requirements that are influenced by context, and then mapping these to appropriate behaviours of applications, are well recognized [3].

A quantitative preference model proposed by Agrawal and Wimmers[20] enable users to supply many separate preferences which can later be combined using a scoring mechanism. A score belongs to a set [0 to 1] and [Veto, Indifference], where the numerical scores capture relative desirability. Henricksen [3] proposed a preference model based on this scoring mechanism. Each preference assigns a score to each adaptation choice that is determined according to the context. A final score of a choice is computed using a utility function such as average(), wgtaverage(), override(), and as(). However, these functions may cause difficulties for users to fully understand how their model works, or how exactly a particular adaptation is derived from the user's preferences.

Alternatively, there are number of context-aware systems that model users' adaptation preferences by means of rule-based expressions. Preferences in [10] and [8] are expressed in terms of if-then rules, where as in [7] they adopted the ECA (event-condition-action) rules for expressing user preferences. However, system intelligibility is not the main focus of their models, they focused on other areas such as privacy and adaptability. Hence, there is limited mechanism in place for generating explanations regarding an adaptation and providing control for users with various levels of technological expertise. In addition, situation abstractions for representing high level and complex context information are not employed as rule conditions in these models. Nevertheless, rule-based preferences are expected to have a higher intelligibility than preferences that require some form of calculations, as they represent a closer mapping to users' cognition of preference representations. This is the solution that we pursue in the following section.
In this section, we begin by defining a set of key requirements that guide the design of the preference model. Subsequently, we present the model and focus on how it supports explanations generation for intelligible context-aware applications.

### A. Requirements

The process of deciding what adaptation to perform for any context-aware application involves evaluating user needs and preferences. Hence, the principal requirements of a preference model are:

- **To possess high standards of usability.** The model should conceptually simple that enable non-technical users to formulate a variety of preferences efficiently and easily.
- **To provide adequate expressive power so the preferences can be incrementally combined to form more sophisticated and elegant preferences.**
- **To support externalisation of situations and preferences evaluation** into middleware to ease software engineering of applications by reducing there complexity.
- **To support conflict resolution** mechanism that are based on preferential knowledge interactively acquired from users in the context of specific problem situation, and
- **Generic** enough to be applicable to arbitrary context-aware applications [11].

### B. Overview of the preference model

As discussed in previous section the preference model in Henricksen's framework is designed for developers [3]. Here we discuss a more user-centred version aimed to provide support for generating explanations of adaptations and user customisation of adaptation decision-making process. We choose to extend Henricksen's framework as it offers formal context and preference modelling abstractions, and supports externalisation of their evaluation to middleware. The evaluation of context and preferences can either be application initiated (i.e., when an evaluation is on a user request, such as requesting an application to select the best communication channel for contacting a colleague), or triggered by context changes (i.e., when there is a change in contextual situation that would trigger a notification for which the application has subscribed). Both types of evaluation are supported by the programming models in the framework known as Situation Based Branching (SSB) and Situation Based Triggering (SST).

In this section we propose a preference model that builds on top of the CML situation abstractions [19], and employs a rule-based system where each preference is composed by a set of rules (φ). Each rule is either one of the following:

- **IF Situ [B1] ELSE [B2],**
- **IF Situ [B1],**

where \([B_1,...,B_n]\) can either be an action \([\text{THEN } \text{Act}]\), or another set of rules \([\phi]\) (e.g., IF Situ [B1] ELSE [B2]). The word "Situ" represents a set of situation predicates which describes the contexts in which preferences apply. Situations are defined using first order logic using context facts of a CML context model [5]. Situations can be combined using logical connectives, \(\land\) (conjunction), \(\lor\) (disjunction) and \(\neg\) (negation). The word "Act" represents an action to be performed when the situations hold, where its parameters can either be bound or unbound. An optional keyword FORBID can be specified in front of the action. The word indicates any rule associated with the action will not be evaluated, which effectively prevents the action to be executed during the lifetime of the preference evaluation.

Overall, a preference can simply be written as follows:

\[
\text{PrefName} \left( V_1,...,V_n \right) : \phi \rightarrow \text{IF Situ} \left[ B_1 \right] \text{ELSE} \left[ B_2 \right] | \text{IF Situ} \left[ B_1 \right] \rightarrow \text{Act} | \text{FORBID?} \text{Action}
\]

In the following section, we present some example preferences and illustrate reuse of preferences.

### C. Preference Examples

In this section, we illustrate some example preferences being applied to a smart home that provides context-aware services designed to assist elderly in maximizing independence and maintaining a high quality of life. Figure 2 shows a preference that is used to decide how the smart home application should behave in a situation when an occupant has fallen in the house.

\[
\text{OccupantHasFallenPref}(\text{Occupant}) : \\
\text{IF } \neg \text{IsConscious}(\text{Occupant}) \{ \\
\text{IF } \text{IsDoctorFor}("\text{Dr Joe}", \text{Occupant}) \text{ THEN contact}("\text{Dr Joe})" \} \\
\text{ELSE contact}("\text{ANY_Doctor}") \}
\]

\[
\text{IF } \text{IsNextOfKinFor}("\text{Ms Helen}", \text{Occupant}) \land \text{Available}("\text{Ms Helen})" \text{ THEN contact}("\text{Ms Helen})")
\]

\[
\text{IF } \text{WorkingHr}() \land \text{IsDoctorFor}("\text{Dr Ken}", \text{Occupant}) \text{ THEN contact}("\text{Dr Ken})")
\]

Figure 2. A preference example OccupantHasFallenPref, and the result of its evaluation.
The preference name (OccupantHasFallenPref) and a variable (Occupant) are shown at the top row, follow by a set of well-formed rules. The meanings for each of the rules are very much self-explanatory. They imply that the occupant prefers to contact Dr Joe if she has fallen unconscious, otherwise contact her next of kin and/or Dr Ken if the incident occurred during office hours. Preference OutgoingCallPref in Figure 3 illustrates the usage of the keyword FORBID. The preference forbids the use of synchronous channel, such as voice call, when the callee is busy and the matter is not urgent, even if the callee is a doctor.

```
OutgoingCallPref(Callee, Priority):
IF IsOccupied(Callee) AND Urgent(Priority) THEN
  FORBID synchronousChannel(Callee)
ELSE asynchronousChannel(Callee)
```

Figure 3. A preference illustrates usage of FORBID.

```
FireAlarmActivatedPref(Occupant):
IF OccupantNotHm(Occupant) THEN evaluatePref
  (OutgoingCallPref, "FireService", "High")
ELSE promptAction(Occupant)
```

Figure 4. A preference demonstrates reuse of an existing preference.

Figure 4 demonstrates how to reuse an existing preference (OutgoingCallPref) to ease the creation of another preference FireAlarmActivatedPref. The preference implies that when a fire alarm is activated and no body was home, place an emergency call automatically. Otherwise, in case it was a false alarm, prompt the occupant for action if she was home. By reusing existing preferences of other events, complicated preferences can be expressed in a straightforward manner.

By eliminating the utility functions for combing various ratings in Henricksen’s model, the preference model becomes easier for users to understand. Users can express preferences by directly stating what appropriate actions should be taken for particular situations, and predict the eventual outcome of a preference evaluation with ease. The evaluation of preferences can either be application initiated or triggered by context changes as supported by the Henricksen's programming models. The preference model is designed to be generic, i.e., be applicable to arbitrary context-aware applications that have infrastructure support for the management of context (using high level context models and situations). For example, the smart home application [23], the virtual handover application [21] and the emergency command centre prototype [22] are sample applications that suitable for adopting the preference model in their development. Next section we describe methods for generating explanations of preference evaluations so users can better understand how decisions of adaptations are made.

IV. SUPPORT FOR INTELLIGIBILITY

The preference model supports intelligibility of context-aware applications by facilitating explanations generation regarding an outcome of a preference evaluation. For each preference evaluation, an application can generate explanations for a range of explanation types mentioned previously by Dey et al. [2] as shown in Figure 5.

To explain how each explanation type is achieved, we use a sample result of a preference evaluation, as shown in Figure 2, where the preference is evaluated against the current context. Assuming the following is the current context: the occupant has fallen, occupant is conscious, Ms Helen is the occupant’s next of kin, and Ms Helen is available, and it is currently outside working hours. Given the current context and the result of the preference evaluation, the application can generate explanations for each explanation types by, firstly using following methods inspired by Lim et al. [1], to identify relevant situations or an outcome of the evaluation. Secondly, their descriptions are being retrieved from a repository - a semantic manager [11] that stores descriptions and meanings of various situations, preferences and adaptations in textual/multimedia format.

What explanations inform users what exactly the outcome of the evaluation is (e.g., an execution of an action or an evaluation of another preference, etc.). This explanation returns a brief description that is associated with that outcome.

Why explanations inform users why did the application produce a particular outcome. This explanation selects the situations that were satisfied to produce the outcome and returns their descriptions.

Why Not explanations inform users why an alternative outcome was not produced. This explanation identifies unsatisfied situations, retrieves their descriptions and returns their negation.

How To explanations inform users how can the application produce an alternative outcome. This explanation selects the rule that links to the alternative outcome, and returns the description of each situation that comprises the rule.

Control explanations inform users how to make modifications to the preference and its rules. The explanation directs users to an adaptive feedback system for editing situations and preferences.

Thirdly, with the descriptions of situations or outcome returned from individual explanation types, they are concatenated with appropriate strings (e.g., +because+, +when+, and +not+, etc.) using a concatenation method [24] by Halpin et al., to produce readable explanation phrases as shown in Figure 5.
allowed to create new situations. In addition, detail introduce new rules with existing situations, but are not preference. Each rule comprises of various situations given permissions to read, modify and create new rules in a intermediate be revealed to and accessed by users with account

adaptation decisions. Thus, a feedback system should understand of the application operations and its experience to an application. This can affect their have varying levels of expertise in technology, and levels of design of appropriate feedback mechanisms is that users can explicit feedback to the decision-making through interaction adaptations performed by context-aware applications do not always result in behaviours that users expects, due to variability in human preferences and imperfect sensing of context information, etc. This can cause annoyance and hinder user objectives. This paper outlined a rule-based preference model built on top of CML situation abstractions, aimed to address the issue of intelligibility and user control in applications behaviours. We described specific methods to generate a wide range of explanations from the preference model to support application intelligibility. In addition, we presented a user model to support the design of a feedback system, which allows a variety of users to override undesirable adaptations. The user model defines how a user's level of expertise will affect his/her interactions with the system.

As part of our future work, we intend to carry out a user study to evaluate the effectiveness of the preference model, as well as the explanation generation methods and the user model. The study will investigate their impact on system intelligibility, and how well they improve user control for various users as they are asked to override unwanted behaviours of context-aware applications. It is possible that some users may still find the preference model too complex, as they are not used to express preferences with if-then-else clauses. This is one of the questions that we aim to answer from the user study; whether it is necessary to investigate more user friendly ways e.g., using graphical programming languages for representing preferences, or relax the first requirement of the model (usable by non-technical users) to non-context management experts, etc.

V. SUPPORT FOR USER CONTROL

In order to provide user control of application behaviours, users not only have to understand how the adaptation decisions are made, but also be able to provide explicit feedback to the decision-making through interaction with applications. One of the challenges related to the design of appropriate feedback mechanisms is that users can have varying levels of expertise in technology, and levels of experience to an application. This can affect their understanding of the application operations and its adaptation decisions. Thus, a feedback system should account for the level of users' expertise by adopting a user model, which is to determine how much information should be revealed to and accessed by users with respect to their requirements and capabilities. The aim is to avoid over-exposing application information and overwhelming users with unfamiliar system features, which can cause confusion to users.

In this section, we discuss a user model that is based on three levels skill acquisition. The user model is designed to be adopted by our preference model to facilitate the development of feedback mechanisms that account for users' expertise. It supports users' customisation of preferences based on three levels of expertise: novice, intermediate and advanced. At the novice level, users are given permissions to read, modify and create new rules in a preference. Each rule comprises of various situations connected using logical connectives. Novice users can introduce new rules with existing situations, but are not allowed to create new situations. In addition, detail explanations regarding an evaluation of a preference, like of Figure 5, are generated for users. They are expected to understand how situations are linked to various adaptations, and able to override undesirable actions by modifying the rules.

At the intermediate level, users have the privilege to access the situation level of preferences. Users are able to identify what situations are being used by the rules, and modify the predicate logic that defines the situations. Situations are comprised of context facts. Users can create new situations with existing context facts that are available in the current context model. In the case where the necessary context facts are not available, only users with advanced level privilege can extend the context model to include necessary context facts. Users are able to create and modify context facts in the current context model by employing tools such as [5]. As advanced users inherit all the privileges of the pervious levels, they are allowed to introduce new situations with the newly added facts, combine the situations to form new rules, and ultimately create new preferences with the rules they recently introduced.

VI. FUTURE WORK AND CONCLUSIONS

<table>
<thead>
<tr>
<th>Explanation Types</th>
<th>Explanations</th>
</tr>
</thead>
<tbody>
<tr>
<td>What is it doing?</td>
<td>Contact Next of Kin “Ms Helen” - A voice message and a SMS have been sent to Ms Helen’s Mobile</td>
</tr>
<tr>
<td>Why</td>
<td>Because the occupant has fallen conscious, Ms Helen is occupant’s next of kin and Ms Helen is available.</td>
</tr>
<tr>
<td>Why Not contact Dr Joe, Dr Ken or Any doctor</td>
<td>Didn’t contact Dr Joe because the occupant is conscious.</td>
</tr>
<tr>
<td>Dr Joe or …</td>
<td>Didn’t contact Dr Ken because it’s not within working hours. …</td>
</tr>
<tr>
<td>How To contact Dr Joe, Dr Ken or …</td>
<td>Contact Dr Joe when occupant is not conscious and Dr Joe is the doctor for the occupant.</td>
</tr>
<tr>
<td>Control</td>
<td>User may edit situations and preferences using an adaptive feedback system</td>
</tr>
</tbody>
</table>

Figure 5. Explanation generated from the result of the OccupantHasFallenPref evaluation.

Our methods support generation of five common types of explanations demanded by users [2]. Having users understand how adaptation decisions are made, and which rules and situations contributed to the final action, the users may decide to personlise the internal workings of applications according to their preferences. Next section we describe how the preference model can facilitate the design of a feedback system to provide user control of applications for achieving more desirable context-aware behaviours.

In order to provide user control of application behaviours, users not only have to understand how the adaptation decisions are made, but also be able to provide explicit feedback to the decision-making through interaction with applications. One of the challenges related to the design of appropriate feedback mechanisms is that users can have varying levels of expertise in technology, and levels of experience to an application. This can affect their understanding of the application operations and its adaptation decisions. Thus, a feedback system should account for the level of users' expertise by adopting a user model, which is to determine how much information should be revealed to and accessed by users with respect to their requirements and capabilities. The aim is to avoid over-exposing application information and overwhelming users with unfamiliar system features, which can cause confusion to users.

In this section, we discuss a user model that is based on three levels skill acquisition. The user model is designed to be adopted by our preference model to facilitate the development of feedback mechanisms that account for users' expertise. It supports users' customisation of preferences based on three levels of expertise: novice, intermediate and advanced. At the novice level, users are given permissions to read, modify and create new rules in a preference. Each rule comprises of various situations connected using logical connectives. Novice users can introduce new rules with existing situations, but are not allowed to create new situations. In addition, detail explanations regarding an evaluation of a preference, like of Figure 5, are generated for users. They are expected to understand how situations are linked to various adaptations, and able to override undesirable actions by modifying the rules.

At the intermediate level, users have the privilege to access the situation level of preferences. Users are able to identify what situations are being used by the rules, and modify the predicate logic that defines the situations. Situations are comprised of context facts. Users can create new situations with existing context facts that are available in the current context model. In the case where the necessary context facts are not available, only users with advanced level privilege can extend the context model to include necessary context facts. Users are able to create and modify context facts in the current context model by employing tools such as [5]. As advanced users inherit all the privileges of the pervious levels, they are allowed to introduce new situations with the newly added facts, combine the situations to form new rules, and ultimately create new preferences with the rules they recently introduced.

VI. FUTURE WORK AND CONCLUSIONS

Adaptations performed by context-aware applications do not always result in behaviours that users expects, due to variability in human preferences and imperfect sensing of context information, etc. This can cause annoyance and hinder user objectives. This paper outlined a rule-based preference model built on top of CML situation abstractions, aimed to address the issue of intelligibility and user control in applications behaviours. We described specific methods to generate a wide range of explanations from the preference model to support application intelligibility. In addition, we presented a user model to support the design of a feedback system, which allows a variety of users to override undesirable adaptations. The user model defines how a user's level of expertise will affect his/her interactions with the system.

As part of our future work, we intend to carry out a user study to evaluate the effectiveness of the preference model, as well as the explanation generation methods and the user model. The study will investigate their impact on system intelligibility, and how well they improve user control for various users as they are asked to override unwanted behaviours of context-aware applications. It is possible that some users may still find the preference model too complex, as they are not used to express preferences with if-then-else clauses. This is one of the questions that we aim to answer from the user study; whether it is necessary to investigate more user friendly ways e.g., using graphical programming languages for representing preferences, or relax the first requirement of the model (usable by non-technical users) to non-context management experts, etc.
ACKNOWLEDGEMENT

NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program; and the Queensland Government.

REFERENCES


