Improving Understanding, Trust, and Control with Intelligibility in Context-Aware Applications

THESIS PROPOSAL

Brian Lim

2 May 2011

Human-Computer Interaction Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213
byl@cs.cmu.edu

COMMITTEE

Anind K. Dey (Chair), Carnegie Mellon University
Scott E. Hudson, Carnegie Mellon University
Aniket Kittur, Carnegie Mellon University
Margarett M. Burnett, Oregon State University

ABSTRACT

Context-aware applications can facilitate people as they carry out their daily tasks. These applications can use a suite of sensors to detect what is happening in the environment and with the user. They can then infer the user intention. This way, they try to understand the contexts of the situation, and consequently act to provide services. For example, a smart phone can recognize that you are in a conversation, and suppress any incoming messages during this period. To minimize obtrusiveness and allow users to focus primarily on their tasks, context-aware applications perform sensing implicitly without explicitly informing users. Furthermore, to better understand the contexts of users in their physical and social environments, context-aware applications are using increasingly complex mechanisms to infer these contexts (e.g., by using machine learning algorithms). This implicit sensing and complex inference can remain invisible when the applications work well and as expected, but become a mystery when the applications behave inappropriately or unexpectedly. In such cases, the lack of understanding of these applications can lead users to mistrust, misuse it, or abandon them altogether. To counter this, context-aware applications should be intelligible, capable of generating explanations of their behavior.

This thesis proposes to investigate how to provide intelligibility in context-aware applications, and evaluate its usefulness to improve user understanding, trust, and control. We explored what explanation types users are interested in for different context-aware applications under various circumstances. We provided explanations in terms of questions that users would ask, such as why did it do X, why did it not do Y, what if I did W, what will it do, how can I get the application to do Y? Early evaluation found why and why not explanation types most effective, among these four explanation types, to improve understanding and trust in intelligent context-aware applications. We have developed a toolkit to help developers implement intelligibility in their context-aware applications, such that they can automatically generate explanations. However, presenting explanations remained an open design exercise, so we explored usability issues of and design principles for providing intelligibility in a prototype of a mobile context-aware application.

Having developed technical and design support for intelligibility in context-aware applications, we next seek to further evaluate intelligibility on improving understanding, trust, and control. First, we shall explore the limits of the helpfulness of intelligibility given the uncertainty of the application. Due to the scrutable and transparent nature of intelligibility, we hypothesize that, while intelligibility helps users better appreciate a highly certain application, it would also expose the inadequacy of a highly uncertain application. We propose to investigate the impact on user impression on context-aware applications with high and low certainty. Next, we shall explore the symbiotic relationship between intelligibility and control. With the greater understanding from intelligibility, users should be able to better control and configure context-aware applications. Conversely, being empowered with the capability to control the application, users would be able to better understand and appreciate intelligibility. So we propose to investigate how the combination of intelligibility and controllability improves understanding, trust, and control of context-aware applications.
1 Introduction

Over the past 20 years many attempts have been made to achieve Weiser’s vision of ubiquitous computing [66] through continued advancements in context-aware computing [16, 52]. Context-aware intelligent systems adapt and tailor their behavior in response to the user’s current situation (or context), such as the user’s activity, location, and environmental conditions. Most of these systems employ complex rules or machine learning models. With the goal of calm computing [27], these systems rely on implicit input often collected without user involvement. Thus users of context-aware applications can have great difficulty reasoning about system behavior [8, 9]. Such lack of system intelligibility (in particular if a mismatch between user expectation and system behavior occurs) can lead users to mistrust the system, misuse it, or abandon it altogether [44].

One mechanism to alleviate this lack of intelligibility in intelligent context-aware systems is through automatically generated explanations. This approach has been employed in several domains including decision making [20], recommender systems [29], end-user debugging [32, 33, 35], and user modeling [12], with the goal of increasing user trust and acceptance of these systems. This thesis proposes to investigate the provision of intelligibility in context-aware applications to help users better understand them, trust them, control them, and hence accept them. For example, a context-aware application may mis-infer your availability to receive phone calls, and allow a colleague to call you at the library. Intelligibility in this application would allow you to learn why this apparent mistake happened. It could tell you how it did correctly infer your location as the library, but that you forgot to set a rule to be unavailable at this place, or that your colleague ignored social norms and called anyway.

1.1 Thesis Approach

This thesis proposes a win-win strategy for both developers and users of context-aware applications. Providing intelligibility requires extra effort for developers to implement applications, and consuming intelligibility also requires more effort for end-users to process and interpret information from these applications. However, we posit benefits to both: intelligibility can improve end-user acceptance of context-aware applications through their improved trust, and intelligibility can improve user control (both perception, and actual ability) of these applications through improved understanding. This is articulated in the thesis statement:

Intelligibility in context-aware applications can improve end-users’ understanding of how these applications work. Consequently, end-users would learn to trust these applications more, and would also be able to more effectively control these applications.

To prove this thesis statement, we approach the problem in three high-level stages. First, we (i) explore what intelligibility is and define it through exploratory work, then we (ii) facilitate and support intelligibility so that it is easier to provide it, and finally, (iii) we evaluate the usefulness of intelligibility towards the thesis goals.

I) Requirements Gathering and Specification

In the first stage, we sought to define a framework for intelligibility. We accomplish this with a literature review of explanations in intelligent systems (Section 2), and empirical work eliciting what explanations potential users of context-aware applications would like to know (Section 0). To this end, we have defined a taxonomy of explanation question types (Section 3).

II) Facilitation, Support, and Guidelines

The next stage implements the requirements determined from the taxonomy of intelligibility, and provides generalized support for intelligibility in context-aware applications through a toolkit and design recommendations. We facilitate the implementation of intelligibility with the Intelligibility Toolkit (Section 4.3), and also explored design and usability issues to derive guidelines for providing and presenting intelligibility (Section 4.4).

III) Evaluation

With the foundational work in place, the final stage is to evaluate intelligibility in context-aware applications. Using the toolkit and design guidelines, we can rapidly prototype intelligibility in context-aware applications to test our hypotheses. We have investigated the individual impacts of different explanation types on user understanding and trust of context-aware intelligent systems (Section 4.1). The proposed work (Sections 5.1 and 5.2) seeks to deepen this investigation, treating intelligibility holistically (i.e., explanation types presented cohesively), and dealing with more realistic context-aware applications.
The following table summarizes the different pieces of work contributing to this thesis:

<table>
<thead>
<tr>
<th>Stage</th>
<th>Project</th>
<th>Section</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>I Requirements Gathering and Specification</td>
<td>Literature Review</td>
<td>2</td>
<td>Completed</td>
</tr>
<tr>
<td></td>
<td>Assessing Demand</td>
<td>4.2</td>
<td>Completed</td>
</tr>
<tr>
<td>II Facilitation, Support, and Guidelines</td>
<td>Intelligibility Toolkit</td>
<td>4.3</td>
<td>Completed</td>
</tr>
<tr>
<td></td>
<td>Designing for Intelligibility</td>
<td>4.4</td>
<td>Completed</td>
</tr>
<tr>
<td>III Evaluation</td>
<td>Intelligibility of Question Types</td>
<td>4.1</td>
<td>Completed</td>
</tr>
<tr>
<td></td>
<td>Intelligibility &amp; Uncertainty</td>
<td>5.1</td>
<td>Proposed</td>
</tr>
<tr>
<td></td>
<td>Intelligibility &amp; Control</td>
<td>5.2</td>
<td>Proposed</td>
</tr>
</tbody>
</table>

See Section 6 for the expected contributions of the proposed thesis.

2 Related Work – Explanations in Intelligent Systems

Research in several domains have explored the impact of explanations to improve user trust and acceptance of intelligent systems, including knowledge-based systems (e.g., [25]), task processing systems (e.g., [24, 27, 42, 53]), intelligent tutoring systems (e.g., [25]), recommender systems (e.g., [29, 49]), end-user debugging (e.g., [32, 34, 45]), case-base reasoning (CBR) (e.g., [34, 55]), and context-aware systems (e.g., [5, 12, 60, 64]), etc. These domains can be separated into two groups as expert systems handling professional tasks, and end-user systems handling “everyday” activities. We discuss how we draw inspiration from these works that have investigated explanations over the past several decades, and identify gaps and opportunities for providing explanations for context-aware applications in ubiquitous computing (ubicomp).

2.1 Expert Systems

Much early research on explanations in intelligent systems were focused on expert systems to help professionals learn how the system has made decisions, or to help novices learn about decision making. As such, several frameworks of explanations have been developed.

2.1.1 Knowledge-based systems

Drawing from explanation facilities of many knowledge-based systems (KBS), Gregor and Benbasat [25] identify three classification methods of explanation type: content, presentation format, and provision mechanism. They found that KBS systems provide four content types of explanations:

1. Trace or line of reasoning. In response to the typical “why” question, this explanation type describes the decision processes taken by the system, why or how it came to its result. Explanations that EMYCIN [62] provided are of this type.
2. Justification or support. Introduced in the Xplain system [58], this type of explanation provides deeper domain knowledge to justify the system’s process. These deep explanations can incorporate different types of knowledge such as analogies, cases, and text books.
3. Control or strategic. Introduced in NEOMYCIN [10], this type of explanation explains the “system’s control behavior, and problem solving strategy.” This provides the user with the design rationale that the developers employed for the application logic.
4. Terminological. Distinguished by Swartout and Smoliar [59], this type of explanation familiarizes users with domain terms and concepts by providing terminologies and definitions.

Gregor and Benbasat's framework is similar to what was later developed by Haynes et al. [27], so we shall defer discussion on how it relates to intelligibility in context-aware applications.

2.1.2 Intelligent Agents

Haynes et al. [27] performed an extensive review of explanations in intelligent agents (systems that “make use of a knowledge-base and algorithm to carry out its responsibilities”), using a wider scope of systems than just KBS. They propose a framework of four main explanation types: ontological, mechanical and operational explanations, and design rationale.

- Ontological explanations provide “what” information to help users make sense of a concept or a component of the system, including:
o **What – identity.** Basic ontological information about the existence of an agent or agent component, or its identifier.

o **What – definition.** Information beyond simply identifying an agent or component and involves providing it with some meaning in context through definitions.

o **What – relation.** Information about the static structural relation between agents or their components, such as spatial information.

o **What – event.** Especially distinguished, this is information about entities that are primitives in describing causal explanations, and can provide temporal information.

- **Mechanistic explanations** deal with the how of agent behavior. The main type of question is "How does it work?" This type of explanations provides information about how different components interact to give rise to more complex actions.

- **Operational explanations** answer the question of "How do I (the user) use it (the system)?" They provide instructions for the user or other agents to enact some agent behavior.

- **Design rationale** explanations deal with why questions at multiple levels from system component constraints to designer intentions to law-like relations. In relation to the taxonomy provided by Gregor and Benbasat, the design rationale spans reasoning trace and strategic. Haynes *et al.* categorize design rationale into four-ports:
  
  o **Deductive-Nomological (D-N).** Explanations referring to some law or law-like relation between entities and/or agents. This is based on the D-N model that suggests that explanations should take the form of deductive statements predicated on well-established truths [28].
  
  o **Functional.** Design intent of the function of a created agent or component.
  
  o **Structural.** Explanations that refer to the structure of the system constraints that cause an entity or event to happen.
  
  o **Pragmatic.** Explanations to requests that depend on the user’s interest value. These explanations are in response to either why not or what if questions.

In an empirical study using a virtual pilot cognitive model intelligent agent, Haynes *et al.* found that most explanation seeking questions (58%) were ontological, followed by mechanistic (19%), then operational (12%) and design rationale (11%).

The frameworks provided by Gregor & Benbasat and Haynes *et al.* provide a rich design space for different types of explanations. However, they cater to expert systems with users who carry out tasks that require expert decision making. Context-aware applications in ubiquitous computing focus on helping lay end-users in "everyday" activities [2], so their users would require a different set of explanations. For example, we expect the functional purpose of context-aware applications to be clearer than expert systems because, as everyday products, their functional scope would be limited. Therefore, we do not anticipate functional explanation types to be very necessary. Nevertheless, some of these explanation types remain useful for context-aware applications.

In this thesis, the explanations we provide for intelligibility are mainly about the application's line of reasoning, or mechanistic. We treat context-aware applications as inference and decision agents, and, through intelligibility, reveal their reasoning process. We take a user-centered approach, and therefore, also provide pragmatic design rationale explanations to explain to end-users how the application inferred in the context of the user’s goals (why not) or present understanding of the situation (what if). While users should not have to be overly bothered by technical terminology when using everyday applications, to explain some of the low-lying contexts and reasoning traces, terminological explanations may be needed to help users learn relevant explanatory concepts. We also expect users to act on the information they learn from intelligibility, but they would need to know how they can modify or control the context-aware application. Therefore, operational explanations would also be relevant to provide in context-aware applications.

### 2.2 End-User Systems

Research into explanations for KBS or task processing systems tend to focus on trained or reasonably knowledgeable users. However, explanations can be useful for novice end-users to understand unfamiliar programs too, even those that help with their everyday tasks.

#### 2.2.1 Recommender Systems

Currently, explanations of end-user systems are most accessible to people through online recommender systems like Amazon's recommendation of products, Pandora.com's song selection, etc. Herlocker *et al.* (2000) described two sources of errors: model/process, and data. Model/process errors are due to the limited feature space of the computational model used; data errors are due to (i) not enough data, (ii) poor or bad data, or (iii) high variance data. Tintarev [60] classifies the
explanation types used in recommender systems in several types such as **case-based**, **content-based**, **collaborative**, **demographic**, and **knowledge-based**. Much of these explain the recommendations regarding the **similarity** of the attributes of the entities of interest (e.g., speed of camera), of the user (e.g., demographic information), preference similarities between users (e.g., the user preferring low prices).

Even though these similarity-based approaches are highly effective for recommender systems, context-aware applications also use context information about the physical environment and situation. Moreover, context-aware applications can use other types of models to make inferences. From a literature survey of context-aware applications [39], we found that the most popular models are indeed different: rules, decision trees, and naïve Bayes classifiers. Therefore, while explanations have been richly studied for recommender systems, research into explanations for context-aware applications remain an open problem.

### 2.2.2 End-User Programming

Ko and Myers [31] found that end-user programmers of the Alice programming environment [11] asked questions when their expectations are unmet. They asked **why did** questions when something unexpected occurs and **why didn’t** questions when something expected does not happen. Ko and Myers subsequently develop the Whyline system [32, 33] that traverses the program tree to generate reasoning traces within the program code to generate **why did** and **why didn’t** explanations:

- Why did the program do X?
- Why didn’t the program do Y?

Myers et al. [45] extended this capability to end-user “everyday” productivity tools with the Crystal framework to support these explanations in a sample text editor that has auto-correct features. In both applications, users use a GUI menu to ask questions. Following this question-asking approach, Kulesza et al. [35] investigated the provision of **why did** and **why didn’t** explanations for an email client that uses the naïve Bayes machine learning classifier to sort email. Due to the probabilistic nature (rather than deterministic or rule-based) of the naïve Bayes classifier, reasoning traces were not used for the explanations, but a representation of weights from various inputs (keywords). Explanations were provided as a rich visualization of bar charts.

It is intuitive that end-users would also ask why and why didn't questions for other "everyday" applications, and, in the proposed thesis, we take this approach of providing explanations to these questions, but generalize it for context-aware applications. Our work leverages some explanation techniques from Kulesza et al., extending them to explain physical contexts that are more relevant for context-aware applications. Furthermore, the overall approach in end-user programming is to allow the end-user to debug the application when it behaves inappropriately. We broaden the use of explanations to be used in more situations, even when the application is functioning appropriately.

### 2.2.3 Ubiquitous Computing and Context-Aware Applications

Context-aware applications for ubiquitous computing present new challenges for providing explanations to end-users. These applications would penetrate everyday life and have a wide impact on end-users [2]. Furthermore, many of these systems would automatically gather information (contexts) about the user and environment and implicitly take various actions [16].

However, such activity done “quietly” without the user’s knowledge [67], without much transparency, can be disconcerting to users who may like to know how their information is being used. Bellotti and Edwards [8] state that context-aware applications must be intelligible: being able to “represent to their users what they know, how they know it, and what they are doing about it.” Some early **intelligible** context-aware applications provide end-users with a modest amount of explanations to give them insight mainly by providing transparency (showing the application's underlying state) and traceability (showing reasoning trace) information. Cheverst et al. [12] investigated how much users would want to know about rules governing a context-aware system and whether to control it. The system takes actions depending on context changes (and history) and the user model (e.g., preferences), and displays to users its rules of a fuzzy decision tree and its certainty about the inference. Tullio et al.'s interruptibility displays [61] explain how they determine a manager's interruptibility by exposing the values of sensors in the manager's room. Panoramic [68] provides reasoning trace, location status, and history explanations to explain location events through a visualization of parallel timelines of sensed and rule-determined events. Some frameworks and toolkits have also been developed to provide wider support for intelligibility in context-aware applications. PersonisAD [5] defines a distributed framework to support explanations by resolving identities and associations of devices, locations, people, etc. It makes user models **scrutable** so that users can control which parts of their user model can be private or public and visible to the sensing environment. Dey and Newberger [17] provide the Enactors toolkit to support intelligibility and control in context-aware applications by adding the Enactor component to the Context Toolkit. For intelligibility, it allows applications to provide input context values, and reasoning traces. For control, it exposes parameters that the UI layer of the
application can allow users to interact with and manipulate. This thesis extends the scope of intelligibility to allow users to ask more questions of the application's state and inference mechanism. For example, users would be able to ask about an anomaly with a Why Not question, and ask about a possible future scenario with a What If question.

Vermeulen [65] plans to explore the design space for providing and presenting intelligibility in ubicomp systems along the dimensions of: timing (before, during, or after an event), generality (general, or domain-specific), degree of co-location (whether intelligibility is provided in the same UI or separately), initiative (user, or system initiated), modality (visual, auditory, haptic), level of control (not controllable to fully programmable). My proposed thesis takes a different approach to investigate intelligibility in context-aware applications. Rather than explore multiple presentation styles for intelligibility, we have explored the provision of intelligibility from an information-centric perspective. End-users are considered information consumers of explanations, and intelligible applications as information providers through the explanations they can generate, and present. Presentation styles are definitely important for the effective assimilation of explanations and conveyance of intelligible information, but we have treated finding the best solutions for presenting explanations in different applications mainly as a design exercise.

In summary, much research investigating the provision of explanations in intelligent systems have demonstrated a positive impact on user understanding and trust. Research in the domain of context-aware computing is also nascent and has shown some promise, but more work is required to provide stronger support for intelligibility and gain better insight about how intelligibility impacts users. My thesis proposes to deepen this research, and provide concrete contributions towards providing intelligibility in context-aware applications. In the next section, we describe how the nature of context-aware applications pose research questions for providing intelligibility, and how we answer these questions in the completed and proposed work for this thesis.

3 Intelligibility for Context-Aware Applications

As mentioned in the earlier section, context-aware applications use implicit sensing, and intelligent inference to figure out the user's context so as to perform appropriate actions. For ubicomp systems, context-aware applications have primarily been developed to support everyday activities, such as tracking the user's physical activity to monitor her exercise, recognizing activity in the home to provide timely medical assistance, determining her availability to others, providing recommendations based on where she is and what she is doing, reminding her to pick up the milk when she is located at the grocery store, etc. They sense implicitly to minimize obtrusiveness and interruption to the user; they automatically sense the situation rather than require the user to manually tell them what is happening. Context-aware applications are increasingly using sophisticated inference mechanisms due to the growing complexity of contexts they need to understand, particularly for activity recognition. For inference, they use big rule sets and machine learning algorithms to handle many cases, and to be more robust to exceptional cases. All these improves the accuracy in properly and calmly understanding the user's context.

Unfortunately, these two factors of implicit sensing and intelligent inference also make context-aware applications difficult for end-users to understand how they work. This is particularly problematic when the applications behave inappropriately or unexpectedly. In such cases, context-aware applications no longer remain invisible to the user's experience, and become a puzzle instead. The users get frustrated if they cannot understand what has happened and why the application appeared to fail. Eventually, this lack of understanding would lead to a loss in trust in the system's inference and behavior, and the eventual abandonment of them. Without a proper understanding of how context-aware applications work, users would also not be able to effectively control them to improve their performance for subsequent situations. Therefore, we have to make context-aware applications intelligible, so that they can explain what they sense and how they are inferring about the users' contexts.

Intelligible context-aware applications "represent to their users what they know, how they know it, and what they are doing about it." — Bellotti and Edwards, 2001

Starting with a broad idea of intelligibility from Bellotti and Edwards [8], we defined intelligibility for a context-aware application in terms of it being able for a to answer questions that users could ask of it. We draw from the question-answering approach of the Whyline [32], with just why and why not questions, and extend to five questions that are relevant for context-aware applications:

1. **What** is the current value of the context?
2. **Why** is this context the current value X?
3. **Why Not**: why isn't this context value Y, instead?
4. **How To**: when would this context take value Y?
5. **What if** the conditions are different, what would this context be?
The first explanation type, What, allows a context-aware application to explain what it knows through implicit sensing; explanation types Why, Why Not, and How To explain how it knows, through its inference mechanism; explanation type What If, allows the user to know what it would do in a given circumstance.

This thesis intends to prove the hypothesis that intelligibility can improve a user's understanding, trust, and control of context-aware applications. We would especially like to show this with the scope of intelligibility that we have defined based on multiple question types. Specifically, our first investigation sought to answer the research question:

**RQ1** Does intelligibility help users improve their understanding and trust of context-aware intelligent systems?

Even though this has been proven true with narrower forms of intelligibility (transparency, and scrutability) in related work, we explored how supporting the various question types independently affect user understanding and trust in context-aware applications. Our work, presented in Section 4.1, successfully shows that providing some explanation types (Why and Why Not) are more effective than others in improving user understanding and trust.

The successful results from our first study showed that providing intelligibility is a promising avenue for research. Next, we sought to carefully explore the scope of questions that users would ask of context-aware applications. Specifically:

**RQ2** What are the intelligibility needs of end-users in context-aware applications?

Answering this question will help ensure that the intelligibility that we provide will be relevant to users and better satisfy their informational needs. In work presented in Section 0, we conducted user-centered, empirical research to elicit what information users wanted to know of context-aware applications when they behaved under various situations. We found more explanation types, and this expanded our taxonomy of question types to include:

6. **Inputs:** what factors affect this context?
7. **Outputs:** what other values can this context take?
8. **Certainty:** how confident is Laksa of this value?
9. **Control:** how can I control the application to improve it?
10. **Situation:** what else is happening in this situation (not about the application, but about the circumstance)?

We now knew which explanation types users asked of context-aware applications, and which they asked for more under various circumstances. However, it remained difficult for application developers to implement intelligibility in context-aware applications, especially with such a wide range of explanation types. This brings us to the next research question:

**RQ3** How can we support the implementation of intelligibility in context-aware applications?

We chose to provide toolkit support for developers to easily add intelligibility to their context-aware applications (Section 4.3). We developed the Intelligibility Toolkit that provides extensible components to support the automatic generation of explanations, and mechanisms to process the explanation information into simpler forms that end-users may easily interpret. However, this technical contribution did not provide final solutions for how the explanations should be presented to end-users. This left open the next research question:

**RQ4** How can we design intelligibility for context-aware application to be usable for end-users?

We answer this question with a think-aloud usability study described in Section 4.4, where we designed a complex context-aware application that uses multiple input contexts and various rules and machine learning classifiers. In this study, we explored several design principles for intelligibility, and learned how users interpreted explanations from an intelligible context-aware application. Our findings provide insights and design recommendations for providing usable intelligibility in context-aware applications.

At this stage, we have investigated how to provide intelligibility from gathering requirements, providing technical support, and recommending design principles. This allows developers and designers to more easily and carefully implement, provide, and present intelligibility in context-aware applications. This also allows us to explore our hypotheses about the impact of intelligibility with more realistic intelligible context-aware applications. Logically, we next address research questions relevant to evaluation in light of realistic issues. One concern is that context-aware applications are not always certain of what they infer, and providing intelligibility may not be helpful when they are uncertain. This could be because users learn how incompetent the applications are. This brings up the research question:

**RQ5** When is intelligibility helpful and harmful for context-aware applications with different certainties?

We already know that intelligibility can be beneficial (from answering RQ1), and which situations that intelligibility is particularly needed (from answering RQ2). Answering this question will allow us to further characterize when intelligibility
should or should not be provided based on application certainty. The proposed study described in Section 5.1 also leverages a more realistic context-aware application (based on work described in Section 4.4) than that in Section 4.1. This will allow us to validate the results in Section 4.1 in a more realistic setting. We will once again investigate our hypothesis that intelligibility improves user understanding and trust in context-aware applications.

Our final proposed work looks at the close relationship that intelligibility has with control. Even though end-users of context-aware applications would typically let the application function seamlessly without direct user input, they will still desire a sense of control, and would occasionally need to control and configure the application to override its decision, or improve its performance for future situations. We hypothesize that, with the increased understanding due to intelligibility, users will also be more able to effectively control the context-aware application. By more effective control, we mean that the user will know which parameters to adjust to improve the application's inference and behavior, and that he will also be able to change parameter values such that its inference for the current and, possibly, future situations will be improved. This addresses the final research question that rounds off our thesis objective:

\textit{RQ6) Does intelligibility help users improve their understanding, trust, and control of context-aware applications?}

The work proposed in Section 5.2 will investigate this question with a realistic and interactive context-aware application prototype, where we provide four different versions with and without intelligibility, and with and without controllability.

The following table summarizes the research questions that the proposed thesis tackles and how answering them will satisfy the thesis requirements. Through answering the research questions, we have developed a means to providing a \textit{relevant} form of intelligibility for context-aware applications, and will have demonstrated the usefulness of intelligibility on three aspects of user understanding, trust, and control of these applications.

<table>
<thead>
<tr>
<th>Research Question</th>
<th>Requirements</th>
<th>Section</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Does intelligibility help users improve their understanding and trust of</td>
<td>✓ ✓</td>
<td>4.1</td>
<td>Completed</td>
</tr>
<tr>
<td>context-aware intelligent systems?</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 What are the intelligibility needs of end-users in context-aware applications?</td>
<td>✓</td>
<td>4.2</td>
<td>Completed</td>
</tr>
<tr>
<td>3 How can we support the implementation of intelligibility in context-aware</td>
<td>✓</td>
<td>4.3</td>
<td>Completed</td>
</tr>
<tr>
<td>applications?</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 How can we design intelligibility for context-aware application to be usable</td>
<td>✓</td>
<td>4.4</td>
<td>Completed</td>
</tr>
<tr>
<td>for end-users?</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5 When is intelligibility helpful and harmful for context-aware applications</td>
<td>✓ ✓</td>
<td>5.1</td>
<td>Proposed</td>
</tr>
<tr>
<td>with different certainties?</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6 Does intelligibility help users improve their understanding, trust, and</td>
<td>✓ ✓ ✓</td>
<td>5.2</td>
<td>Proposed</td>
</tr>
<tr>
<td>control of context-aware applications?</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

In the next section, we describe in detail the pieces of work that have been completed for this thesis.

### 4 Previous Work

Several pieces of work have been completed towards the proposed thesis. We have performed an early investigation of the impact of intelligibility on user understanding and trust (Section 4.1), assessed what types of information users would like to know about when using context-aware applications in various situations (Section 0), provided toolkit support for implementing intelligibility (Section 4.3), and explored design and usability issues of intelligible interfaces (Section 4.4). All four of these studies are lab-based, with three of them being conducted as online surveys.

I shall present the completed work in chronological order of when they were completed, revealing the deepening understanding of how to provide and support intelligibility in context-aware applications as we progressed through the research agenda.
4.1 Intelligibility of Question Types for Context-Aware Intelligent Systems

RQ1) Does intelligibility help users improve their understanding and trust of context-aware intelligent systems?

When people try to understand situations, they may ask a variety of questions. This is also true when they are trying to figure out how a device or application works. Our early framing of intelligibility centered around supporting explanations to a set of five questions that users may ask of a context-aware application:

1. **What** is the value of the context?
2. **Why** is this context the current value?
3. **Why Not**: why isn’t this context value Y, instead?
4. **How To**: when would this context take value Y?
5. **What if** the conditions are different, what would this context be?

The first question (What) is non-trivial particularly for context-aware applications that *quietly* take action and do not explicitly indicate their state. Users may want to ask about the underlying reasoning and logic of the application (Why, Why Not, How To), and forecast how the application would behave in a future situation (What If). We hypothesized explanations to some of these question types would help users more than others, in terms of helping them understand the application, and trusting it more. By evaluating this difference, we would be able to identify which explanation types to focus on to maximize user understanding and trust. This section summarizes the work published in [37].

4.1.1 Method: Lab study with an abstract intelligent system

Given the different factors we wanted to investigate and the flexibility of our testing infrastructure, we were able to independently test different intelligibility elements in a series of experiments. We made the tradeoff of conducting controlled, yet simple, experiments with a large number of subjects that we could generalize from, over studying more realistic, yet more complex situations.

We created two web interfaces representing two context-aware intelligent systems. The first was a concrete system that considered three inputs (body temperature, heart rate, and pace) and inferred whether the user is exercising. The system was represented as an input-output interface (see Figure 1). It could be intelligible, showing one of four types of explanations (Why, Why Not, How To, What If). Why and Why Not explanations were provided as reasoning traces in text form. How To and What If explanations were provided as interactive interfaces where users had to provide additional input (see Figure 2 and Figure 3). The second system was an abstract system with anonymized input factors, to remove the confound of prior knowledge about concrete factors affecting how well participants may understand the system reasoning. Under the hood, the system used a simple decision tree for inference, but the participants were not told this.

---

Figure 1. Screenshot of the interface for our intelligibility testing infrastructure.
Our study varied whether we provided participants with explanations, and which explanation type is provided if so (i.e., None, Why, Why Not, How To, What If conditions between-subject). To let participants learn the system, we showed participants 24 examples of the system functioning under different situations (different input and output values). We measured how well participants understood the system by asking them how they thought the system reasoned about the inputs to derive the output value (free text response) for each example. We also asked them for their mental model about how they thought the system worked, overall. These were coded based on how correct the reasons they provided were (Wrong, Rules are some Inequalities, Partially Correct, Fully Correct). We further measured their understanding with 15 test questions of prediction. We presented the interface with a missing input or output, and had participants fill in what they thought would be an appropriate value. Finally, we asked them to self report how much they felt they understood the system (5-point Likert scale), and also asked for a self report of how much they trusted the system output (5-point Likert scale).

The study was conducted online through Amazon Mechanical Turk, and we recruited 211 participants.

4.1.2 Results & Implications: Why & Why Not better explanation types

For the experiment with the abstract intelligent system, we found that Why and Why Not explanation types helped participants more than How To and What If. With Why and Why Not, participants had more correct answers (see Figure 4), had more detailed and correct reasons (Figure 5), had more correct mental models of how the system worked (Figure 6), and reported that they understood the system better (Figure 7a). We believed that this was because it was easier to assimilate the textual explanations of Why and Why Not. Some participants reported that they struggled with how to use the How To and What If interaction facilities. This suggests that How To and What If explanation interfaces would need to be more carefully designed to be useful, or that it may truly not be useful or important to provide these explanation types.

Participants receiving Why Not explanations were not able to provide correctly describe the reasoning of the system as well as those receiving Why explanations (Figure 5). This could be due to the increased cognitive load of reading contra-positive statements of the Why Not explanations. Participants in the Why Not condition also did not trust the system as much as those in the Why condition (Figure 7b).

We did not see as significant results with the concrete intelligent system, but only marginal differences for understanding and trust. This could be due to participants applying their prior knowledge of exercising to understanding how the system works and not paying careful attention to the explanations, as evidenced by the reasons they provided. This persistence of mental model was also found in [60] where participants received explanations, over time, about how an interruptibility system worked.
Our findings suggest that providing reasoning trace explanations for context-aware applications to novice users, and in particular Why explanations, can improve user’s understanding and trust in the system. This work was a necessary first step into understanding the impact of explanations in context-aware applications. From the small set of five questions we considered, we found that some explanation types are more effective than others in improving user understanding and trust. We next sought to expand the scope of intelligibility to include more questions that users may ask of context-aware applications.
4.2 Assessing Demand for Intelligibility in Context-Aware Applications

RQ2) What are the intelligibility needs of end-users in context-aware applications?

In work described in the previous section, we found that some types of explanation were more effective than others in improving users’ understanding and trust of a context-aware intelligent system [37]. However, it is not clear what information users actually want to know and will ask about, and whether there are more explanation types than we had previously considered. In this work, we explored and assessed user demand for intelligibility: which types of questions users want answered, and how answering them improves user satisfaction of context-aware applications. User satisfaction is obviously crucial for adoption and acceptance of such technologies. This section summarizes the work published in [38].

We hypothesize that there are different types of information users are interested in, for different context-aware applications, and different situations. Since people ask information seeking questions due to cognitive disequilibrium [25] and to correct knowledge deficits [62], we believe that satisfying these information demands through intelligibility can lead to better satisfaction when using these applications and improved adoption and acceptance. In order to elicit the information demands users have for context-aware applications under various situations, we conducted a study of the demand for explanations and different types of information in several scenarios users may find themselves in as they use context-aware applications. Using described scenarios instead of actual field deployments allows us to quickly and more effectively study and understand the impact of different information on intelligibility and satisfaction, without having to implement and deploy a variety of applications, any of which could fail for reasons independent of our main focus.

4.2.1 Setup: Scenarios of Four Context-Aware Applications

To investigate the demand for intelligibility in the space of context-aware applications, we selected four prototypical context-aware applications: (i) a desktop interruptibility management application (an Instant Messenger plugin based on [6]), (ii) a remote person monitoring peripheral display (inspired by Digital Family Portrait [46]), (iii) a context-aware reminder application (based on CybreMinder [15]), and (iv) a mobile context-aware tour guide (based on CyberGuide [1]). All applications in this study behaved according to models of learned decision trees. For each application, the scenarios intentionally spanned a range of incorrect, appropriate and unexpected or anomalous, but not necessarily wrong behavior, to probe directly at the issues of intelligibility and satisfaction. For brevity, we just describe the scenarios for the first application.

We designed the instant messenger (IM) auto-notification plugin based on recent [6] work on a predictive model to determine how long a buddy would take to respond to a message. Our application uses the responsiveness prediction to determine the subject’s interruptibility [22], and either forwards or suppresses incoming IM messages. We developed four main scenarios for this application where the subject is in various states of busyness:

1. Rushing to reach an imminent deadline,
2. Taking a break and surfing the Internet,
3. Reading a work-related book, and
4. Returning from a protracted informal meeting.

For each scenario, the user receives an IM message from

- A colleague regarding critical work, or
- A friend regarding a fun video.

There are 16 scenarios (2 actions x 4 main x 2 messages).
4.2.2 Free-form elicitation of information demand

For the first experiment, we recruited 250 participants from Amazon Mechanical Turk. We showed them one of four application surveys, where each survey presented multiple scenarios of the application behaving differently for various situations. We presented short 5-second video clips and short textual descriptions about what is happening. After a scenario is presented, participants are shown 2-5 (depending on the application and scenario) instances of the scenario, one at a time, with different application responses, represented as screenshots along with text (e.g., see Figure 8, right), where the behaviors may be appropriate, strange, or incorrect. We then asked them how they felt about the situation and application behavior, and them what information they would like to know about the situation or what the application just did.

![Diagram of hierarchy of intelligibility types]

Figure 9: Hierarchical representation of intelligibility types that users want to find out about.

From our participants' free-text responses of information need, we coded 10 types of explanations arising from various situations (summarized in Figure 9). We found that participants asked different questions for different circumstances. Hence, we also coded for circumstances:

- **Criticality.** Whether the situation presented is critical. Situations involving accidents or medical concerns with the Digital Family Portrait and work-related urgency for the IM Auto-Notification were considered highly critical. Due to the profound influence of the high criticality of the fall and incontinence scenarios in the Digital Family Portrait survey, these scenarios were excluded from consideration of the other moderators.
- **Goal-Supportive.** Whether the situation is motivated by a goal the user has (CybreMinder scenarios only).
- **Recommendation.** Whether the application is recommending information for the user to follow or ignore (CyberGuide scenarios only).
- **Externalities.** Whether the application is perceived to have high external dependencies (e.g., getting weather information from a weather radio station) vs. being perceived as “self-contained.” CybreMinder and CyberGuide had perceived high external dependencies.

4.2.3 Follow-Up Study: Validation of information demand

We ran a follow-up study to validate the findings of the first study. We reused the applications and scenarios from experiment 1, with one application per survey. Experiment 2 is designed as a between-subject study for the intelligibility types (10 conditions plus a None condition). Participants are assigned to a version of the survey with only one type of intelligibility information provided (including None). We summarize our results in terms of design recommendations for when developers of context-aware applications should implement intelligibility.

4.2.4 Design Recommendations

We present some recommendations to developers of context-aware applications about which intelligibility types to provide and under what circumstances. Developers can identify which moderators and situations apply for their applications and then use the suggested recommendations, e.g., whether the application usually behaves inappropriately, whether it is goal-supportive. Implementing all the types of intelligibility we investigated is excessive and may even be detrimental. We present the different types and offer advice about when and how they should be implemented. This is summarized in Table 1.

1. **Input.** Users may only have a moderate interest in knowing more about the application’s input sources or sensor readings, but if they perceive the application is heavily dependent on external sources (high externality), they may want to know more about the inputs.
2. **Output.** In typical application use, users are not interested in the output alternatives. However, they may suspect that the application is capable of more action than it is exhibiting. Providing intelligibility explaining the output (action) capabilities of the application is particularly important for applications that make recommendations (such as tour guides), especially when users want to seek better options. This should be provided automatically during early stages of usage to improve
users’ awareness.

3. **Why.** Answering why questions is an essential intelligibility requirement as such questions are very common. However, users may also have high expectations that the why explanations are very informative and a simple reason trace may not be sufficient to fully satisfy the users’ enquiries. Visualizations could be used to augment a trace.

4. **Why Not.** Such explanations are good for high risk (high chance of inappropriateness) circumstances and goal-supportive functions. However, we caution against implementing it in all types of context-aware applications because generating these explanations for all alternative possibilities may be non-trivial.

5. **How.** Users are somewhat interested in knowing how the application arrives at its outcomes, and particularly like such explanations. However, how explanations can get cumbersome to produce for applications with complex or learned logic. Users may have to use an interactive facility to specify the constraints in which to obtain an action [37].

6. **What If.** This explanation type also involves user interaction in specifying input conditions and the application simulating what would happen. We recommend what if explanations for non-recommenders and more self-contained applications, for which users indicated strongest demand.

7. **What Else.** The what else explanation provides information closely related to the situation explanation. The latter provides situational awareness, while the former provides more information about what the application has done. Unsolicited demand for what else information is low, but becomes significant when participants are aware of its availability. This indicates an intrinsic need for this type of explanation. What else explanations are also important in critical situations when users hope that the application is doing more to remedy or handle the critical situation.

8. **Certainty.** Providing certainty information is particularly important for applications that are goal-supportive where users want to know how certain the application is in its decision or action, and for applications that rely heavily on external sources and sensors. Certainty values that we provided for this study were over 90%, but we suspect that if low certainty accuracies are reported to the user, this may hurt their impression of the application. As applications can have varying levels of certainty when in use, it may not be wise to always show certainty information.

9. **Control.** For context-aware applications, this intelligibility type would support users changing parameters in the conceptual model that were originally set by the developers or learned by the system. However, when users adjust such parameters, they may be making poorer choices than the developers or the underlying machine learning algorithm, and may ultimately hurt the application accuracy. Nevertheless, it may be important to allow users to change these settings, but caution them about the danger of doing so.

10. **Situation.** Though users are less interested about what else is happening when the application responds, there is moderate interest in increasing their real-world situational awareness (what else). This would involve making applications sense more related events and contexts (e.g., for a monitoring application, what is the historical trace of events before an anomaly) than their primary function, and to reveal such knowledge to the user. This is more important in cases when the application acts in highly critical situations.

<table>
<thead>
<tr>
<th>Intelligibility Type</th>
<th>General</th>
<th>Inappropriateness</th>
<th>Criticality</th>
<th>Goal-Supportive</th>
<th>Recommendation</th>
<th>Externalities</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>L</td>
<td>H</td>
<td>L</td>
<td>H</td>
<td>L</td>
<td>H</td>
</tr>
<tr>
<td>Input</td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Output</td>
<td></td>
<td>2</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Why</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Why Not</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>How</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>What If</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>What Else</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Certainty</td>
<td>1</td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Control</td>
<td></td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Situation</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

L=Low, H=High, 1=recommended, 2=highly recommended

Table 1: Design recommendations of which intelligibility types to implement depending on the circumstances encountered by and functionality of the candidate context-aware application. For example, if the application is inaccurate, it would have low Appropriateness, and we recommend the explanation types: Why, Why Not, How, What If, and Control.
Our findings suggest that some explanation types (e.g., Why, Certainty, Control) should be made available for all context-aware applications, while some are more useful for specific contexts (e.g., why not for goal-supportive tasks). We believe that context-aware application developers can take these recommendations on when and how to provide different types of intelligibility features and dramatically improve user satisfaction with, and acceptance of, their context-aware applications. The following step in the research agenda was to support developers to implement these explanation types in context-aware applications.

### 4.3 Intelligibility Toolkit

**RQ3** *How can we support the implementation of intelligibility in context-aware applications?*

Having elicited what questions users are interested to ask of context-aware applications, we have found a reasonably large set of explanation types that applications may provide to be intelligible. However, it would be a substantial effort for developers to implement these explanations for all context-aware applications for which they would like to provide intelligibility. To support the implementation of intelligibility, and lower the barrier to providing them in context-aware applications, we developed the Intelligibility Toolkit. Its focus was to provide the automatic generation and processing of explanations from the underlying context model of a context-aware application. The Intelligibility Toolkit provide an extensible, standardized framework with which the explanations can be (i) queried, (ii) generated, (iii) post-processed, and (iv) presented. This section summarizes the work published in [39].

From a review of literature from the last 10 years, we determined the most popular decision and inference models used in context-aware applications. The Intelligibility Toolkit currently supports the generation of explanations from the four most popular models: rules, decision trees, naïve Bayes classifiers, and hidden Markov models (HMM).

The Intelligibility Toolkit also initially supports a subset of 8 explanation types elicited from our previous work, particularly those that can be automatically generated. These are categorized into mode-independent (treating applications as input-output functional, stateful systems) and mode-dependent explanation types (varies depending on model used).

<table>
<thead>
<tr>
<th>Model-Independent Explanation Types</th>
<th>Model-Dependent Explanation Types</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Inputs</strong> explanations inform users what input sensors <em>(e.g., thermostat, GPS coordinates)</em> and information sources <em>(e.g., weather forecast, restaurant reviews website)</em> that the application employs so that users can understand its scope. Inputs should be described by their name and possibly also with some description of what they mean or refer to.</td>
<td><strong>Why</strong> explanations inform users why the application derived its output value from the current (or previous) input values. For rule-based systems, this returns the conditions <em>(rules)</em> that were true such that the output was selected.</td>
</tr>
<tr>
<td><strong>Outputs</strong> explanations inform users what output options the application can produce. This lets users know what it can do or what states it can be in <em>(e.g., activity recognized as one of three options: sitting, standing, walking)</em>. This helps users understand the extent of the application’s capabilities. Outputs explanations can also be used to help ask model-based explanations Why Not and How To <em>(see below)</em> by allowing the user to select an alternative desired output.</td>
<td><strong>Why Not</strong> explanations inform users why an alternative output value was <em>not</em> produced given the current input values. They could provide users with enough information to achieve the alternative output value, but not necessarily so <em>(e.g., see the naïve Bayes explanation algorithm later)</em>.</td>
</tr>
<tr>
<td><strong>What</strong> explanations inform users of the current <em>(or previous)</em> system state in terms of output value; this makes the application state explicit. Input values are obtained by recursively asking What on the Inputs. When a user asks a why question, she may actually be asking for What.</td>
<td><strong>How To</strong> explanations answer the question &quot;In <em>general</em>, how can the application produce alternative output value X?&quot; This is in contrast to asking when a specific event occurs.</td>
</tr>
<tr>
<td><strong>What If</strong> explanations allow users to speculate what the application would do given a set of user-set input values.</td>
<td><strong>Certainty</strong> explanations inform users how <em>(un)certaint</em> the application is of the output value produced. They help the user determine how much to trust the output value.</td>
</tr>
</tbody>
</table>

| **Independently** explanation types depend on the context model. **Dependently** explanation types depend on the model. |

### 4.3.1 Toolkit Architecture

We implemented the Intelligibility Toolkit, extending the Enactor framework [17] of the Context Toolkit [16]. Enactors contain the application logic of the context-aware application and contain References that monitor the state of input Widgets. Each Reference contains a *rule* and is triggered when its rule is satisfied. This is managed automatically by the discovery mechanism of the Context Toolkit.
We added an *output* property and a *list* of its values for the Enactor to represent its output value, and output options. Each output value is associated with a Reference. Furthermore, we extended the Enactor framework to support classifiers. We used Weka [51] for the decision tree and naïve Bayes classifiers, and Jahmm [23] for HMM classifiers. Next, we describe the components we added to the Enactor framework to support a wider range of intelligibility (see architecture diagram in Figure 10).

**Explanation Data Structure**

We define an explanation in terms of one or multiple reasons (*e.g.*, multiple reasons for Why Not). Each reason can be a singular conditional (*e.g.*, one certainty value for a Certainty explanation) or a conjunction (*e.g.*, multiple conditionals for a Why explanation). The conditional is the atomic unit of an explanation (*e.g.*, certainty=90%, temperature<24°C). Furthermore, there can be negated conditionals (*e.g.*, not temperature≥24°C). Formally, we define explanations in Disjunctive Normal Form (DNF), *i.e.* a disjunction (OR) of conjunctions (ANDs) of conditionals (see Figure 11; see example in Figure 12). The standardization of explanation information supports R4 such that there is a standard way to pipe and feed different explanation types.

**Explainers**

The Explainer is the main component of the Intelligibility Toolkit that contains the mechanisms and algorithms to generate explanations based on the decision model. There is a generic Explainer that generates explanations for model-independent types, and subclasses of Explainer for each of the four decision models supported.

Model-independent explanations are generated using the architecture of the Enactor framework to provide the corresponding information. *Outputs explanations* report the name and definition property of each context type (Widget attribute) used in the application. *Outputs explanations* report a List of output values for Enactor. *What explanations* report the current Enactor output value. To obtain input values (*Input What explanations*), handles for the input contexts are obtained via the Input explanation, and through those, the *What* explanation. A *What If explanation* sets input context values set by the user (through UserInputsQuerier), and tests it on all References. It reports the output value associated with the Reference that gets triggered.

Model-specific explanations are generated from different Explainers for each model. Our initial implementation includes two broad types of explanations to generate explanations from the four model types: rule and decision tree explainers, and Bayesian explainers.

**Rule and Decision Tree Explainers**

Rules and decision trees can be modeled in tree form, and hence can be transformed into multiple trees of Disjunctive Normal Form (DNF), each tree representing a different output value. Figure 12 and Figure 13 demonstrate DNF trees that provide...
Why, Why Not, and How To explanations for the problem described in Table 2. The explainers for rules and decision trees are implemented in RuleExplainer and DecisionTreeExplainer, respectively.

<table>
<thead>
<tr>
<th>Input Conditionals</th>
<th>Output Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a ): Activity = Sitting \n( b ): Noise = Quiet \n( c ): Latitude near Office's \n( d ): Longitude near Office's \n( e ): Schedule = In Meeting</td>
<td>( \otimes ): Availability = Yes \n( \otimes ): Availability = Somewhat Not \n( \otimes ): Availability = Not</td>
</tr>
</tbody>
</table>

Table 2. Pedagogical example of input conditionals and output values for rule and decision tree. This describes an application to infer a user's availability based on his activity, the noise level around him, his proximity to his office (by latitude, longitude), and his schedule.

Input state \((a, \neg b, \neg c, d, \neg e)\): user is sitting in a noisy place at latitude not near the office, longitude near the office, and is not in a meeting.

Bayesian Explainers (Naïve Bayes and Hidden Markov Models)
Bayesian models make inferences based on conditional probabilities of multiple factors. To explain these models, we extend the idea of weights of evidence demonstrated in [48] that transforms the constituent probabilities into an additive model, where each probability becomes a piece of evidence voting for or against an inference. The sum of these evidences indicate the overall evidence for the inference.

The naïve Bayes is explained as the sum of evidence (see Figure 15 for an example):
1. Prior probabilities of selected class value,
2. Due to each feature value

The hidden Markov model includes a time transition component, and is explained as the sum of evidence (see Figure 16 for an example):
1. Prior probabilities of selected state,
2. Due to each state transition, and
3. Due to each feature value at sequence step

While we have implemented one Explainer per model, additional Explainers can be developed to support other types of models we have not covered, and also to generate different explanation methods for existing model types. For example, we implemented the weights of evidence method from [48] to explain Bayesian models (naïve Bayes, HMM), but there are other explanation methods that could also provide different explanations, e.g., [43, 50].
Queries
To have an explainer generate an explanation, the program can pass a Query to the Explainer. Queries encapsulate information about which context the user is interested in, which question the user is asking, and at which time the user is asking about the context. For example, to ask why activity is inferred as it was at 4pm, we create the query (in pseudocode):

```
new Query("activity", Query.WHY_QUESTION, "4pm")
```

and pass it to an Explainer. The base Query takes the current inputs and output values and is used for explanation types What, Why, and Certainty. AltQuery extends Querier and includes an alternative target output value to facilitate explanation types Why Not, and How To. WhatIfQuery extends Querier and allows the setting of input values, supporting What If explanations. Query can be extended to employ different constraining mechanisms, such as querying based on time.

Reducers
Explanations generated from Explainers may be unwieldy to an end-user in two ways: (i) too many reasons (e.g., numerous ways to achieve a target output value), and (ii) each reason being too long (e.g., numerous inputs with required values to cause the output value). The latter case occurs when many sensors and feature values are used to build the models, as is the case for accurate learned systems. Reducers simplify the Explanation Data Structure so that the explanation is easier for users to interpret. Reducers can also be used to pre-process explanations before presenting them, e.g., stripping privacy-sensitive information, when explaining contexts to a social contact.

Presenters
Explainers produce explanations in the form of Explanation Data Structures, and Presenters render them in a form presentable to end-users, e.g., as text, visualization, or interactive graphical interface. Developers can build different Presenters to suit their target user and device form factor. Furthermore, even if the explanation is large, a developer may elegantly present it (e.g., see Figure 16), instead of reducing it.

4.3.2 Validation: Four Demonstration Applications
We built four applications demonstrating how the Intelligibility Toolkit can be used to provide explanations for different context-aware applications. Figure 14, Figure 15, and Figure 16 show screenshots for three of the applications (excludes rule-based), illustrating how the explanations may look like for different underlying decision models. More examples and tutorials can be viewed and downloaded at the website we deployed for the toolkit at [http://www.contexttoolkit.org](http://www.contexttoolkit.org).

![IM with Bob from Alice](image)

**Figure 14:** IM Autostatus. Demonstration of various explanations from an IM responsiveness prediction plugin that uses a decision tree to predict when a buddy would respond.
How to read Why Not: The top bar indicates the average evidence \( \Delta g > 0 \), i.e. more evidence for Sitting than Standing. The next bar indicates the evidence due to prior probabilities \( \Delta h < 0 \) is in favor of Standing (i.e. user is more likely to be standing). Each of the following bars indicates the difference in evidence for each feature, \( \Delta f \), and whether they are in favor of Sitting or Standing.

Why Not inferred Standing but Sitting? Because (i) the prior likelihood indicates that it is pre-disposed to inferring Standing rather than Sitting, (ii) the values of Mean(x), Mean(y), Energy(x), Energy(y), etc., support the inference for Sitting, while (iii) the values of Mean(z), Energy(z), etc., support the inference for Standing. The user can interpret that the z-axis could be instrumental to infer Standing.

How To + What If explanation: User selects values of inputs, and see how the corresponding evidence changes along with the average overall evidence (bar at top), to see if the threshold (vertical bar) is crossed.

The Intelligibility Toolkit provides automatic generation of eight explanation types (Inputs, Outputs, What, What If, Why, Why Not, How To, Certainty) for the four most popular decision model types (rules, decision trees, naïve Bayes, hidden Markov models) in context-aware applications. It supports the generation of explanation structures (through Explainers), querying mechanisms to specify questions and constrain explanations (through Queries), simplifying complex explanations (through Reducers), and presenting the explanations to end-users and other subsystems (through Presenters). The toolkit is also extensible to support new explanation types, model types, reduction heuristics, and presentation formats.

The Intelligibility Toolkit makes it easier for developers to provide many explanation types in their context-aware applications. This ease also allows developers to perform rapid prototyping of different explanation types to discern the best explanations to use and the best ways to use them. This brings us to focus on the next stage in providing intelligibility — how to design intelligible interfaces for different contexts, so that end-users can quickly and effectively assimilate the explanations.

4.4 Designing for Intelligibility

**RQ4: How can we design intelligibility for context-aware application to be usable for end-users?**

In the previous section, we described the Intelligibility Toolkit that provides support for application developers to easily add intelligibility in their context-aware applications without needing develop their own explanation generation algorithms. The Intelligibility Toolkit also provides other components to facilitate the Reduction and Presentation of the explanation data structure. However, it is still a difficult problem to design usable and effective interfaces for intelligibility. In this work, we explored design and usability issues of making intelligible interfaces for context-aware applications. We focused on a mobile,
social-awareness application, and provide generalizable design recommendations for designing several explanation types. This section presents the work published in [40].

### 4.4.1 Laksa Prototype

We created a real context-aware application prototype, Laksa, to provide a vehicle to explore the design, implementation, and use of intelligibility in a sophisticated, multi-factor context-aware application. Laksa is a mobile application that shares people’s availability status. Laksa is an acronym for Location, Activity, Connectivity (κ), and Social Awareness that describe its function. Status is determined from six lower-level contexts: Place, Motion, Sound activity, phone Ringer, Schedule, and who is enquiring (Contactor).

**Availability:** Available, Semi-Available, Unavailable — is determined based on rules regarding the following 6 factors. The contactor interprets the availability and decides whether to contact and how (call, text, email, etc.).

**Contactor (Who is Enquiring):** Family, Friend, Coworker, and Default (to check user's default status) — categorizes person contacting or enquiring about the user.

**Place:** Home, Office, Café, Library, etc. — represents the semantic location of the user. It is computed by sensing latitude and longitude from the Skyhook Wi-Fi API (uses a hybrid GPS, Wi-Fi, and cell tower positioning algorithm), and matching to a pre-determined named location that the user specifies. To convey accuracy, it also reports the sensed distance error and detected number of access points.

**Motion:** Sitting, Walking, Cycling, Placing the phone Flat, etc. — represents the user’s physical activity inferred with the phone placed in a front pants pocket. Inferences are made with a decision tree trained using activities from several users. Features extracted from the accelerometer are similar to [36]: e.g., mean and standard deviation for three axes, phone orientation angles, and signal powers.

**Sound:** Talking, Music, and Ambient Noise — represents the sound activity that Laksa recognizes from what it can hear from the phone's microphone. Inferences come from a naïve Bayes classifier trained on sound samples. Features extracted are similar to [41]: e.g., mean and standard deviation of power, low-energy frame rate, spectral flux, spectral entropy, spectral centroid, bandwidth, Mel-Frequency Cepstral Coefficients (MFCCs).

**Phone Ringer:** Silent, Vibration, or Normal.

**Calendar Schedule:** Personal, Work, or Unscheduled.

While one could compare the use of explanations for different contexts (e.g., Place, Motion, Sound) and different decision models (rules, decision trees, naïve Bayes), for this formative work, we focus on exploring the design and use of explanations across this breadth of factors.

### 4.4.2 Design of Intelligibility

We employed several design strategies to make the explanations more user-friendly. Figure 17 and Figure 18 show some explanation user interfaces (UIs) resulting from the strategies described next.

**Reducing and Aggregating Explanations**

We expect that users of context-aware applications would rather be focused on their day-to-day tasks than dedicate too much attention to technical details. Hence, it is important to simplify and reduce the provided explanations to be concise and salient. We have done this for Laksa by aggregating explanation types (e.g., What value and Certainty rating shown together), reducing the number of reasons, length of reasons / number of input features (e.g., omitting MFCCs for sound), combining explanations for simultaneous consumption (e.g., presenting x-y-z accelerometer values in 2D diagrams), simplifying names (e.g., "spectral entropy" renamed to "pitch pureness", "low-energy frame rate" renamed to "periods of silence"), and normalizing numerical values to lay scales (e.g., 0 to 100). While this may compromise comprehensiveness, it is intended to make Laksa more intelligible.

**Visualizing Explanations**

Users should more quickly assimilate visual explanations because of the higher bandwidth of diagrams vs. text. Hence, we provide several visual representations: icons for context and feature values, dynamic diagrams that change when values change, and even animation and sound (to hear pitches and pitch ranges). Visualizations are customized for the context domains (e.g., map for Place, physical 2D phone diagram for Motion). Since Sound is not visual, we chose to explain sound by metaphor (e.g., showing a pan flute to represent pitches and ranges; see Figure 18c).
This is the core information users see first to determine what Availability status Laκsa has inferred. It shows the user or buddy’s pic, name, and Availability status; shows time since last change.

To ask for an explanation, first select which context you would like to ask about by selecting one of these seven Context Tabs. Each tab shows the current value of the context (What explanation), e.g., Sound = Talking.

Next, you ask a specific question from the drop-down menu. The Question Panel adapts to the question selected (e.g., Why, Why Not, What If).

The resulting explanation is rendered in the Explanation Panel.

(a) Map visualization explaining with 'bubble' components. Blue bubble indicates estimate of current location; green dotted bubble indicates specified region of named location.

Why is my Place inferred as "Office"? Because your estimated location bubble overlaps with the bubble specifying where Office is.

(b) Physical Diagram visualization (two shown here) illustrating the interpreted mechanical motion of the phone leading to its inference of cycling. Dotted lines and arrows or shading show boundary conditions of rules of the decision tree model. Users can mouse over the diagram to see textual explanations with numerical values.

Why isn’t Motion inferred as “Cycling”? Because the phone is oriented more than the drawn angle, the vigorousness detected is below the illustrated value, etc.

(c) Metaphorical visualization (two shown here) of sound feature values that was sensed and computed at a specified moment. This can show current values sensed, or average values for each possible outcome.

What details (Inputs) affected the Sound inference? 48% of the sound heard was relatively silent (Periods of Silence); the range of pitches heard was from 1140 to 2623Hz (this can be played aurally for the user to listen to), etc.

(d) Weights of Evidence visualization. Bar chart visualization showing weights of evidence voting for or against the inference of Talking. Weights are represented by the length of each bar; color and positioning indicate evidence for or against. Not all factors shown.

Why isn’t Sound inferred as "Talking"? Because most of the factors vote for the inference of Music (especially hidden features, the sound Volume, Periods of Silence, etc); only Pitch Fluctuation votes for Talking.

**Figure 17. Screenshot of the Laκsa showing how to use the components in the core and intelligibility user interface.**

**Figure 18: Explanation Visualizations rendered in the explanation panel.** Some examples and their interpretations.

**Explaining in Relatable Terms**

As we deploy intelligibility in real-world prototypes, users need to understand how the contexts and recognition features relate to the real world. Therefore, we include Description explanations that describe what each context factor and feature mean. These are presented in physical rather than system terms (e.g., there are more *periods of silence* for talking than ambient noise, because there are significant pauses in speech that are relatively quiet). Furthermore, features for Motion and Sound were scaled to physically meaningful names and values (e.g., *vigorousness* to represent accelerometer signal power in Watts). We also chose to visualize explanations for motion using *first principles*. For example, orientation information is shown as the orientation of the phone relative to the ground (see Figure 18b). Note that ensuring that terms are domain relatable may require significant domain knowledge, rather than just naïvely applying effective features identified in the literature about activity recognition (e.g., [36, 41]).
Providing Explanations for Control
We chose to provide explanation types for each context only if users could leverage the information to improve Laksa, or change their behavior. What If explanations are only provided for the top-tier context Availability. For other contexts, users would not be able to meaningfully change the input features (e.g., changing the entropy or frequency to influence sound). Furthermore, we omit trivial explanations, e.g., asking What If one is at a specific coordinate location to learn which semantic place Laksa would infer the user being at; asking Why Not questions about manually set contexts (e.g., schedule or ringer mode).

We iterated on the design of Laksa with these strategies and feedback from colleagues who are active HCI researchers.

4.4.3 Method: Scenario-Driven Think-Aloud User Study
To explore the use of the intelligibility features in Laksa, we conducted a scenario-driven user study where participants think aloud as they use it. This was an exploratory study where we investigated how and why participants used intelligibility, and how this use impacts their understanding of Laksa. We conducted an in-situ controlled study rather than a field deployment for two main reasons: (i) to present participants with a lower-fidelity interface to elicit more feedback from the think aloud study, and (ii) to avoid having serious usability issues that could confound results in a field study, where it would also be harder to monitor participants’ usage and rationale.

4.4.4 Results: Patterns of Intelligibility Use
We recruited 13 participants (8 females) with a mean age of 26.4 years old (range: 18 to 37). We ran each participant for 2.5 hours on average (range: 2 to 3.5). Due to the length of each scenario, each participant experienced between 3 and 6 scenarios (median=4), selected to try to balance coverage.

We transcribed the think aloud and interview data, segmented by speaker (interviewer/interviewee). The transcript was coded by question type (e.g., Why, Why Not, What If), goal/intention/rationale (verified during interviews), feature requests, breakdowns/struggles (e.g., too many questions to choose from), and extent of (mis)understanding. We formed sequence models of the usage of question types, and consolidated them (e.g., see Figure 19). We interpreted the findings and models to identify causal factors. This was assembled into higher-level themes using an affinity diagram (selection based on theme convergence, importance, and novelty).

We found different usage of explanations for the three different situational dimensions presented in the scenarios: exploration/verification, social awareness, and fault finding.

Exploration / Verification
We observed that participants explored all explanation types as they tried to learn how Laksa functioned, and more about the scenarios. Naturally, participants used the Description explanations to remind them what the terms and concepts meant, and used the Inputs explanation to examine deeper states that affect inferences (e.g., for S3, P11 used Inputs explanation of Availability to see a “summary of her statuses”). How To explanations were used in two ways: P11 appreciated learning new concepts about how Sound inference can be made (through periods of silence, pitch ranges, etc.); for S3, P6 checked to see when she would be Unavailable. Finally, some participants asked What If to preemptively test out troublesome or critical situations to see how Laksa would respond under those circumstances; e.g., for S3 during a lull period, P3 and P5 confirmed that they remained available to family members in an emergency.

Social Awareness
For S6 and S7, participants had less knowledge of the true situation of their buddy than about themselves. We observed that they mainly focused on Input explanations of Availability (or equivalently, What explanations of the lower-level contexts). P6 first checked the Input values of Availability to determine the state of her friend (in S7). Participants would then form stories about what they believed their buddy could be doing at the time; e.g., P9 having seen the Motion inferred as “Other” (placed flat) for S6, P9 figured his friend was “probably sleeping or taking a nap or eating or something that would probably involve not wanting a phone call.” When he subsequently looked at the Sound inference, he said “sound is 78%, oh he is listening to music so I guess I could intrude if I want to but then I get he might be with someone else too.” Most participants did not continue by asking other questions. However, for S6, P7 asked When would her friend be Semi-Available (How To) to learn the respective rules her friend had set.

However, we found that once participants discovered that the availability status could be inaccurate or erroneous, they explored other questions and investigated more deeply. This is similar to how participants investigated anomalies with explanations about themselves (discussed next).
Fault Finding

Participants used explanations most when discovering that Laksa behaved unexpectedly. Figure 19 summarizes the sequences of how participants asked for various explanations (labels refer to observations described in the following text). The choices of questions were slightly different when investigating the top-tier Availability than the lower-tier contexts (Place, Motion, & Sound).

<table>
<thead>
<tr>
<th>Top-tier Rule-based Context (Availability)</th>
<th>Lower-tier Sensed / Inferred Contexts (Place, Motion, &amp; Sound)</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Diagram of question flow for Availability" /></td>
<td><img src="image" alt="Diagram of question flow for Sensed / Inferred Contexts" /></td>
</tr>
</tbody>
</table>

Figure 19: Consolidated sequence models of explanation use for fault finding. Participants make a range of decision choices for which question type explanation to ask for. '?' indicates participant desire to ask questions; arrows indicate transitions of using each explanation; only prominent behavior is noted here.

Top-tier, Rule-based Availability

When participants realized that the Availability was wrong, some instinctively first selected the Why question (Figure 19: 1). P6 first asked why about her Availability in S3 and S4, and why her Place was inferred as Office in S4. When explaining her rationale to first select why (S4), P12 said "because I want to know why...why I'm available." This suggests a linguistic cue for asking Why first.

Alternatively, sometimes participants first inspected the state of the application by asking for Input values (2). For S3, through the Input values, P11 discovered that "the talking and possibly the work is unavailable in the schedule [were] the two that led my status to be unavailable." For S4, P6 and P12 more quickly discovered that the Place inference (as Office) was erroneous (should be Library instead).

If participants had an expectation of what the availability should be, they would ask about the expected outcome using Why Not or How To questions (3a, 3b). These represented different strategies to address this goal-oriented query. Interestingly, some participants asked How To instead of Why Not, even though the latter was more concise; e.g., for S4, P7 asked When would status be Semi-Available (How To) to manually identify erroneous conditions out of three. Had she asked "Why isn’t status Semi-Available", she would have seen only one condition that was specifically identified to be relevant to the scenario. Furthermore, some participants misunderstood the How To explanation, e.g., for S7, when P4 asked "When would Sound be Unavailable," he interpreted the requirement that his friend has to be Talking to represent that his friend was currently sensed as talking; for S4, P7 examined the rules for when she would be Semi-Available (as she had expected status to be), found the condition Place = Library, and misunderstood that to mean that Laksa had correctly inferred her location. Possibly, some terminology put off users from using Why Not more: P6 complained about the excessive use of negatives (contrapositives).

Some participants simulated conditions they expected to be true with the What If question (4), to see if Laksa would infer an expected Availability. For S4, P8 asked What If, setting Place to Library (which she believed to be ground truth), and Ringer to Silent (which she believed she should have set). This resulted in the status of Semi-Available, which was correct unlike the actual inference of Available, and indicated that while the rule was executed correctly, possibly something was sensed
wrongly. Unfortunately, participants were prone to carelessness: while setting up the expected state for S2, P6 changed 3 contexts (Place = Café, Motion = Sitting, Contacter = Friends), but failed to notice her schedule was set to Work (a pivotal factor to determine her status to friends). She expected her status to appear as Available, but it appeared as Unavailable instead; for S4, P13 forgot to set Place to Library (left as inferred value Office) when trying to verify the inferred availability.

Using the aforementioned strategies, some participants may decide to add a new rule or modify one to fix the anomaly, and this may be a satisfactory solution. However, most of the problems in the scenarios are due to faults at the lower-tier context inference. To investigate further, they would select the suspect context by clicking on the respective tab.

Lower-tier, Sensed / Inferred Contexts

Once again, participants instinctively asked Why (5); e.g., for S2, P9 asked Why to see “that map thing,” the Map visualization showing which place his location overlaps with; for S2, P6 asked Why Motion was inferred as Walking, only paying attention to which features were listed, and not carefully interpreting the boundary conditions; for S3, P6 first asked Why Sound was inferred as Talking, but found that unhelpful.

Participants also paid attention to the inference Certainty (6). For S6, after noting a Sound certainty of 73%, P10 mentioned that as long as it was above 50%, that was “good enough.” For S7, P12 accepted the inference for Sound as Music (93% certainty), because it was above 90%. Subsequently, he was confused when this inference turned out to be wrong. When in doubt of the current inference, some participants also made it a point to find out which other Output values were plausible through their Certainties (7); e.g., for S6, P6 checked the certainty of which Sound was inferred as Talking (8%), became wary that the actual inference (Music at 73%) may be wrong, and became hesitant to contact his buddy. For S4, P12 wanted to see whether Kaxa recognized squatting, and seeing Certainties of 67% for Standing and 33% for Cycling, he accepted that “cycling is kind of like squatting” and “partially standing.”

When asking about an expected outcome or exploring an alternative outcome with a noticeably high certainty, participants similarly demonstrated two dominant exploration strategies: asking Why Not (8b), or How To together with Inputs (8a). Asking Why Not provides a concise explanation that directly compares the actual inference with the desired outcome. For S4, immediately after noticing a wrong Place inference, P5, P6, P12 asked Why Not to see how their location bubble did not overlap with the bubble for Library. For S5, P5 used the Why Not Physical Diagram to explore how Running was not inferred (Cycling was). For S7, P6 became uncertain after noticing, from the Weights of Evidence visualization, that Pitch Fluctuation strongly voted for inferring that her friend was Talking, while every other factor voted for Listening to Music (see Figure 18d). However, we found that many participants disliked the explanations as being too technical, particularly the Physical Diagrams for Motion. In fact, for S5, focused on Cycling, P7 avoided the Motion diagrams of the Input, Why, and Why Not explanations. For S6, P9 found the Weights of Evidence visualization explaining “Why isn’t (Why Not) Sound Ambient Noise” confusing (difficult to remember what icons meant), and preferred to just look at the Output Certainty of Ambient Noise. Other participants alternatively used the How To explanation in conjunction with Inputs, by manually comparing the two explanations; e.g., for S6, P9 repeatedly toggled between the How To and Inputs metaphorical diagrams for Sound. He studied Pitch Pureness to see if the current Input value (≈29) was “just about right” compared to the average value for Music (≈34), and accepted the Sound inference as Music. After several exposures to both techniques, P12 realized (in S5) that the Why Not explanation for Motion provided similar information to How To + Inputs, and required less effort to inspect.

4.4.5 Discussion: Themes Of Intelligibility Use

We created an affinity diagram of our coded findings to map out core issues and patterns of use. Here, we present the top three high-level themes of how and why participants used or failed to use the intelligibility features.

Information Overload and Explanation Detail

While we intend explanations to express a comprehensive view of what Laksa knows and how it infers, we run into the problem of information overload. Truly comprehensive explanations are too long and complicated. Even though we took several steps to reduce the explanation complexity, participants still complained about the remaining complexity. P1 pointed out (as expected) that there were too many questions to choose from, and she did not necessarily know which was best for her goals, P3, P7, P8, and P9 also complained about the large number of reasons provided for Availability explanations (up to 9), and the large number of input features described for Motion and Sound. In fact, P3 suggested showing up to 3-4 reasons; most participants only paid attention to 1-3 features of Motion (especially just vigorousness and movement) and Sound (periods of silence, pitch range, pitch pureness). When trying to determine her friend’s availability for S6, P10 grew tired of asking for explanations. She felt “like it was information overload,” and that she “started getting less information about what he was doing.” She started doubting whether her friend was truly listening to music, or sleeping instead. Clearly, our lay users had a low threshold for explanation detail.

24
Furthermore, participants preferred Certainty explanations because of its single value (e.g., P9). Similarly, P12 eventually showed a preference for the more concise Why Not instead of How To explanation for explaining Sound. While participants found the Motion and Sound input feature details interesting, they also found them too technical for such a lay-user application (e.g., P3, P7).

**Prior Knowledge and Relatability**

It is well-known that prior knowledge plays a role in learning and understanding, and we observed how that influenced how participants used and interpreted explanations; e.g., since Laκsa uses access points to sense location, the geographical distribution of these points affects the inferred location, and the distance error. However, P4 did not know this, and had no idea in S4 that Place was wrongly inferred as not being the Library.

This problem was more widely manifest in a usability issue: some participants wanted to see explanations framed in terms of the real world; e.g., for S5, P1 wanted Motion vigorousness to be stated as “within the walking or running range” or in terms of exercise "high or low intensity". She found forces and orientation unhelpful to understand her exercise. While this can clearly help users in making sense of input values, this is less feasible if an input has multiple ranges of values for certain outcomes (e.g., multiple ranges of orientation angles for sitting due to different resting angles of the phone in a pocket).

The participants’ use of Inputs and How To explanations for a comparative method to derive a Why Not explanation also suggests this need to frame sensor features in terms of well-understood activities. For S7, to convince themselves that a sound heard was really talking, participants looked at the feature values in Inputs (representing the current state), and values typical of Talking (found by asking How To).

**Different Strategies in Problem Solving**

We observed that some participants employed suboptimal problem solving strategies to try to figure out how Laκsa made inferences. We observed a lack of strategy and logical fallacies such as causal oversimplification. Even when provided with various explanation tools, some participants did not know how to effectively use them. For S2, P11 sequentially explored questions in the drop-down list of questions, while others (e.g., P6, P12) chose the Why explanation instinctively. Many participants also ask How To when they wanted to get a Why Not explanation. This required them to do extra work to figure out which reason is relevant, when Why Not would automatically select it.

Participants also exhibited common logical fallacies. Many participants exhibited causal oversimplification [13], because as they looked at reasons (e.g., from Why, Why Not), they mistakenly fixated on a single factor and ignored others. P6 and P11 felt that Schedule was the “explicit way” of saying whether they were available (S2). P4 thought that since his friend was at the office (S7), then he must be Unavailable, even though he also would have needed to be talking. Having formed the wrong conception that his friend must be busy, P4 stubbornly tried to look for clues to verify his hypothesis, rather than revise it. He was thus unable to figure out the problem without help from the experimenter.

**Blame Shifting**

![Figure 20: Blame shifting during S4 about mistakenly receiving a phone call in the library.](image)

We observed participants shifting blame attribution in several scenarios, particularly, S4 where the participant received a loud call while at the library (see Figure 20). Participants changed who or what they attribute blame to after carefully considering what they know, and viewing Laκsa’s explanations. Immediately after receiving the interruption, P1, P7, P8, P12 reacted instinctively and were annoyed with Laκsa (1a), while P6, P13 were self-judgmental and felt they forgot to silence the ringer appropriately (1b). On reflection, P7, and P12 realized their coworker should have seen their Place and known not to call them. They would then blame their coworker for violating social norms (2). After looking at the availability status displayed, participants realized Laκsa was indeed wrong, and all participants shifted blame to Laκsa (3a, 3b). However, after viewing explanations and finding out that the status error was due to a poor location sensing (and both Library and Office in tight proximity), participants had different reactions. P8, P12 continued to blame Laκsa for its imprecise sensing, and even became harsher in their judgment because they (incorrectly) expected indoor location sensing to be as precise as contemporary GPS devices.
(4a). On the other hand, P6, P7, P13 forgave it because they understood how challenging it was to sense location in the given circumstance, and understood that they would have to change a setting to improve sensitivity (4b). This supports findings about reduced blame attribution when autonomous robots explain their actions [30]. While one might assume explanations improve perception and trust of Laksa, this observation reveals explanations to be a double-edged sword: revealing the difficulty of some situations, or exposing a lack of competence.

4.4.6 Design Recommendations For Intelligibility

Drawing from our design experience and user study, we present some recommendations on how to improve the usability of providing explanations that can be applied more generally to context-aware applications.

Reducing and Aggregating Explanations

We had originally employed this strategy before the user study when designing Laksa, and have found this to be even more crucial based on our study results. In fact, participants demanded an even lower level of detail, wanting to see more details only as needed. Hence we continue to recommend this requirement. One compromise would be to allow incremental access to more detail on demand and offer significantly reduced explanations initially. Alternatively, it could be even better to present them in a form that is concise but does not compromise by omitting any reasons. Finding How To explanations cumbersome due to the large number of tabs to see multiple reasons, P1 suggested just presenting the rules in a table instead. This would provide a bird's eye view of the rules and yet be much easier to access. Furthermore, because some participants found some features for Motion and Sound to be overly technical, it may be sufficient to filter them out of explanations rather than making them physically meaningful, or provide metaphors to explain them. However, it is unclear whether users want to see them when encountering more serious and esoteric debugging problems.

Retooling Explanations with Simpler Components

Several participants (e.g., P6, P9, P13) referred to explanations of Place as “the bubble thing” or “map thing” instead of noting which question they wanted to ask. They used the simple bubble components of Place (Figure 3a) to investigate various questions (Figure 21). Therefore, it may be better to design simple explanation components that can be used to answer a number of questions than to explicitly answer those questions through different automatically generated representations: i.e., use explanation components with a smaller vocabulary set expressive enough to convey most of the explanation types we have employed. Unfortunately, it is difficult to design reusable, simple explanation components for contexts like motion and sound, because they depend on a wider and more diverse range of contexts that may not be represented equivalently.

Figure 21: Simple "bubble" components simultaneously for explaining seven questions about Place (What, Inputs, Outputs, Why, Why Not, How To, Certainty). What: Place inferred as at place A. Inputs: (Latitude, Longitude) coordinates of sensed current user location. Output possibilities: place A, B, etc. Why at A: because sensed location bubble overlaps with bubble of place A. Why Not at B: because sensed location bubble not overlapping with place B. How To be inferred at B: need location and B bubbles to overlap. Certainty of inference: distance error and number of access points.

Streamlining Questioning

Being aware of her lack of understanding to effectively use the explanation questions, P7 suggested a flow chart to help guide users to ask optimal questions. We propose the streamlined flow of questions as shown in Figure 22, where only 1-3 question types are accessible at any given time. This helps reduce information overload when choosing explanations. It also better supports framing of information by allowing quick comparisons between Why and Why Not, and Inputs and How To (to support the two observed why-not strategies). Since Inputs and What If explanations are popular, they are made relatively easier to access. When users first want to ask questions, they can seek the mechanistic rationale by asking Why, explore the system state through Inputs, or explore the Certainties of alternative Output values. As convenient shortcuts, users can ask Why Not having seen the alternative Output values, and ask about alternative input values through What If after viewing the current Input values. Users can also explore Output values through the What If UI by looking at the possible values each context may take.
Non-Mechanistic Explanations
We found that users need more types of explanations to properly ground themselves in the domain the application operates in, and to educate them about good problem solving and debugging strategies to fully understand the program functionality.

Given the deep knowledge that complex context-aware applications rely on to make decisions, and evidence that some participants lacked sufficient prior knowledge to relate technical behavior to real-world and domain phenomena, we can see that simple textual descriptions are not sufficient to scaffold the automatically generated explanations. This suggests context-aware applications need to have access to information about complex real-world concepts that are not necessarily core to the application.

Moreover, we found that since some users did not effectively leverage the explanation facilities provided, intelligible applications may need to teach them problem solving strategies. One solution may be to provide examples of end-user debugging with the explanation tools.

Following the model-centric approach of supporting intelligibility in context-aware applications, this work takes a design and usability-centric approach to explore how to present usable explanations to help users better understand an intelligible context-aware application. Our findings emphasize the importance of making explanations usable and quickly consumable (by reducing information overload), relating the application behavior to the physical world (to increase the relevance of the information), and supporting effective problem solving and debugging strategies (so that users can quickly understand the application issues before giving up). We suggest a need for streamlining explanations while maintaining access to the rich explanation capabilities, and for integrating domain knowledge in explanations. With a better understanding of how users use the question type explanations, we can better design explanations to help understand sophisticated context-aware applications.

4.5 Summary
We have explored intelligibility from a simple set of explanations of question types (Section 4.1), investigated a more comprehensive list of information types that users want to know about context-aware applications (Section 4.2), developed a toolkit to support and expedite the development of this expanded requirement for intelligibility (Section 4.3), and iterated on the design of an intelligible context-aware application, deriving several usability issues and design guidelines for providing and presenting intelligibility (Section 4.4). These lay the groundwork for a framework which specifies what to provide to make context-aware applications intelligible, and how to concretely provide intelligibility in these applications.

Our early evaluation of intelligibility in context-aware intelligent systems (Section 4.1) only dealt with four explanation types, and was based on simple, abstract intelligent systems. As we have gained greater insight into how to provide intelligibility, we wish to continue to evaluate intelligibility for more realistic context-aware applications.

5 Proposed Work
In the proposed follow-up, we will focus on the evaluation of intelligibility towards proving the thesis statement. Unlike our earlier evaluation, we will conduct experiments with realistic context-aware applications, with rich intelligibility explanations. We will leverage the Intelligibility Toolkit and design guidelines that we have developed to build these intelligible prototypes. Specifically, we will refine the Laksa prototype and use all or a subset of it for the subsequent studies.

We plan two follow-up studies as controlled, lab-based experiments, rather than free-form explorative studies. In particular, we are interested in secondary factors that would influence the impact of intelligibility: uncertainty, and control. Real context-aware applications make inferences with incomplete information, and are subject to uncertainty. Depending on how well their inference mechanism have been designed, some applications can be particularly uncertain, and have poor accuracy.
This not only compromises its core performance, but will also cause them to generate explanations about their poor performance. Hence, intelligibility may be harmful to these inaccurate applications, and should be avoided. Conversely, an accurate application will be able to use intelligibility to demonstrate how well it makes its inferences. In our first follow-up study, we propose to investigate the interaction effect between the accuracy (or certainty) of context-aware applications, and providing intelligibility (Section 5.1).

For the second study (Section 5.2), we choose to investigate the mutual relationship intelligibility has with controllability. Controllability would be the capability to configure parameters of a context-aware application. We expect intelligibility to help participants better control the application to improve its performance. On the other hand, we expect the opportunity to control the application to enhance the usefulness of intelligibility. Previously, we have noted many participants complaining about the pointlessness of explanations if they could not do anything about the information. This study gives us the opportunity to specifically explore this relationship.

5.1 Intelligibility and Uncertainty

RQ5) When is intelligibility helpful and harmful for context-aware applications with different certainties?

Our earlier studies have shown great promise for the efficacy of intelligibility in context-aware applications, but they have assumed the use of systems that have reasonably high certainty in their actions, and that, while fallible, generally take appropriate actions. Intelligibility would enhance the positive impression a user may have of an application, and reveal how it intelligently tries to figure out what is happening even for difficult sensing and inference situations. Unfortunately, because of these difficulties in sensing and inference, applications can be uncertain of their actions, often resulting in users having a negative impression of these applications. It is hoped that intelligibility would help bring up this shortfall, and raise a user's impression of a context-aware application. However, is there a certainty below which intelligibility would not help, but may actually harm a user's impression of the application? If this were the case, the user could lose even more trust in the application's capability and precision. So an application with sufficiently low certainty would not benefit from adding intelligibility, and instead, the developer should focus on improving its certainty instead.

5.1.1 Hypotheses

We believe the user's impression of an application impacts her trust of it. We define that a user has a good impression of a context-aware application when she perceives it to be highly certain of its inference, feels that it generally behaves appropriately, and she agrees with what it is doing. As illustrated in Figure 23, we hypothesize:

Figure 23. Hypothesis that Intelligibility will improve user impressions when an application is certain of its actions, but it will harm impressions when it is uncertain.

H1a: Above a certainty threshold, intelligibility improves a user's impression of a context-aware application.

H1b: Below the threshold, intelligibility harms the user's impression of the application. This could be due to the user realizing how poorly the application is performing.

We hypothesize that this effect on impression is due to the increased understanding provided by intelligibility:

H2: Providing intelligibility helps increase a user's understanding of the application.

While H2 has been shown to be true in [37], we seek to verify those results, as H1 depends on this. Thus, H2 in combination with H1b hypothesizes that a gain of understanding about a low certainty application leads to a loss in impression. To test these hypotheses, we designed and ran a large-scale, between-subject lab study, described in the next section.

5.1.2 Approach

We will present two scenario-driven lab studies where we investigate the interaction between intelligibility and application uncertainty. For the first study, we will manipulate the provision of Intelligibility in three levels (None, Certainty-only, Full),
Intelligibility and Control

and Certainty in six levels (50, 60, 70, 80, 90, 100%), in a between-subject design for an online survey. We will design two context-aware applications (location-aware, and sound-aware) to explore the impact of certainty on intelligibility for applications with differing complexity. Even though simulated, these realistically and faithfully mocked-up applications will be based on the functional Laika prototype that we previously developed (see Section 4.4), and will behave realistically in the situations. Participants will be shown 10 scenarios to experience the applications under their various versions. Collectively, the scenarios will be representative of the application certainty (e.g., for 60%, the application behaved appropriately for 6 out of 10 scenarios). We plan a follow-up think-aloud study to add greater context to our quantitative findings. Our anticipated contributions are:

1. Understanding how users respond to intelligibility in context-aware applications under different levels of certainty; and
2. Identifying when, how, and why intelligibility is helpful or harmful as a result of application certainty.

5.1.3 Evaluation: Understanding and Perception of Accuracy

We will investigate the interaction between the provision of intelligibility, the certainty of the application, and the impact on understanding and impression using a large-scale, controlled lab study. The study will be deployed online through Amazon Mechanical Turk to allow us to collect input from a large number of participants and span many levels of certainty and intelligibility (as in [37, 38]). We will ask verification questions to filter out unconscientious participants.

Experimental Conditions

We vary Intelligibility and application Certainty as independent variables in a between-subject experiment, across two applications, for a total of 3×6×2=36 conditions. We will recruit about 15 participants per condition for each application.

Intelligibility (3 conditions: None, Certainty-only, Full)

We varied whether participants were provided with explanations where they only saw the application inference (None), or additionally saw a rich explanation visualization (Full). We included an intermediate intelligibility level, where we provided just Certainty percentage only, to investigate how much value the explanation visualizations add over just showing certainty.

Certainty (6 conditions: 50%, 60%, 70%, 80%, 90%, 100%)

We varied certainty as six intervals (rather than a dichotomy) to be able to observe any trends that may arise.

Measures

We are interested in measuring how much participants understand the application for each intelligibility condition, and whether this affects their perception of certainty, feeling of whether the application behaved appropriately, and how much they agree with the application’s inference.

Understanding. For each scenario, we ask participants what they think about why the application inferred what it did, and why not something else (free-text). These responses would be coded by counting the number of correct and wrong reasons to get a measure for how many reasons participants can produce.

Perceived Certainty. For each scenario, we ask participants how certain they believed the application was in its inference (as numerical input 0 to 100%). After the scenarios, we asked for their overall sense of the certainty.

Perceived Appropriateness. For each scenario, we measure what the participant felt about the appropriateness of the application behavior, on a 7-point Likert scale from Very Inappropriate to Very Appropriate.

Agreement. For each scenario, we measure how much the participant agreed with the application’s inference, given the ease or difficulty of making the inference; on a 7-point Likert scale from Strongly Disagree to Strongly Agree.

Much of our previous work had shown positive implications for providing intelligibility, describing when and how it helps users better understand and trust context-aware applications. However, we expect this work to provide a cautionary tale to providing intelligibility. It could suggest a minimum bound for application certainty and confidence before intelligibility is beneficial. Alternatively, it could also suggest a different form of intelligibility, or a different strategy of presenting explanations when the application confidence is particularly low. Nevertheless, this work will help refine our recommendations on when and how to provide intelligibility in context-aware applications.

5.2 Intelligibility and Control
RQ5) Does intelligibility help users improve their understanding, trust, and control of context-aware applications?

Intelligibility can help users learn to trust context-aware applications, but users should also use their improved understanding to better control these applications. However, there are various control models for users to manipulate context-aware applications and this may influence what kinds of explanations are best to facilitate effective control. We intend to investigate how intelligibility can help users more knowledgeably and effectively control and configure context-aware applications. We propose a lab study with the provision of intelligibility and control as a single independent variable with four values (Non-intelligible and Non-controllable, Intelligible only, Controllable only, and both Intelligible and Controllable), and user understanding, perception of trust, and control effectiveness of the application as dependent variables. We hypothesize that understanding, trust, and control would be greatest with both intelligibility and controllability, but least with neither (see Figure 24).

![Figure 24. Hypothesis that, singularly, intelligibility and controllability would each improve understanding and trust, and together, they would further improve understanding and trust.](image)

### 5.2.1 Approach

At this stage, we are interested to see how intelligibility can improve a user's operational understanding of how a context-aware application works. This requires an interactive prototype of an intelligible context-aware application. This application will be a derivative of our Laxsa prototype from (Section 4.4). Similar to studies from our previous work, we describe several scenarios to our participants. We task participants to try to understand how the application makes its inferences, and to adjust parameters to improve its performance. We will provide a reasonably accurate application, but not perfectly accurate so as to allow the user to improve the application's accuracy.

### 5.2.2 Evaluation

Participants would be randomly selected to be in one of the four intelligibility and controllability conditions. They would learn about the application through in-situ scenarios, and would be tasked with improving their inference in some scenarios. Participants would be videotaped, and asked to think-aloud when they use the application, so that we can clearly observe how they make use of the core, intelligibility and controllability features. The core features would just be the application inference and action. At regular intervals, we would ask participants about their perception of trust, and how much they understand the application. Occasionally, for participants with the controllable versions (Controllable-only, or Intelligible and Controllable), we would task them to use the controllability features to improve the application's behavior.

### Plan for Implementation

For the application platform, we plan to develop a single context-aware application based on our Laxsa prototype from (Section 4.4). The intelligibility features would remain similar to what we design in the previous study (Section 5.1), but it will have extra controllability features. For the location context, the parameter of control will be the threshold of the place bubbles, and which radio signals to use (GPS, Wi-Fi, Cellular). For the sound context, control will be facilitated through adjusting the weights or positions of the bar chart representing the evidence of each factor (see Figure 17d). This interaction will be similar to the control feature described in [34]. For the availability context, the control will be through adding, removing, or editing rules of availability based on the lower-level contexts.

### Measures

We are interested in measuring how much participants understand the application for each condition, and how well they can control the application to improve its behavior.
**Inspective Understanding.** For several scenarios, we ask participants what they think about why the application inferred what it did, and why not something else. These responses would be coded by counting the number of correct and their quality to get a measure for how many reasons participants can produce.

**Control Score.** This measure would only apply to participants using controllable versions of the application (Controllable-only, or Intelligible and Controllable). For several scenarios, we task participants to use the controllability features to try to improve the application’s behavior. The result would be whether the inference is correct, what the inference certainty would be, and how it affects other inferences in the survey (i.e., whether inferences of other situations are affected).

**Operational Understanding.** We also record the thought and action processes that participants express while controlling the application. These responses would be coded by counting the number of correct procedures and actions that the participants take.

**Agreement.** For each scenario, we measure how much the participant agreed with the application’s inference, given the ease or difficulty of making the inference; on a 7-point Likert scale from Strongly Disagree to Strongly Agree.

**Perception of Trust.** After the scenarios, we measure the participant’s sense of trust of the application in a set of four counter-balanced 7-point Likert scale questions.

We aim to recruit about 10 participants per condition (total 40), and run each participant for about one to two hours, including training time.

As we apply intelligibility towards more realistic context-aware applications, we have to investigate how users operationally make use of the information they learn from the explanations. This work will explore how intelligibility may help users better engage with context-aware applications, rather than just passively learning more about how they work.

5.3 **Schedule**

We present schedule for the remaining proposed work in terms of semesters:

<table>
<thead>
<tr>
<th>Schedule (Semesters)</th>
<th>Proposed Work</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spring to Summer 2011</td>
<td>Intelligibility &amp; Uncertainty study</td>
</tr>
<tr>
<td>Summer to Fall 2011</td>
<td>Intelligibility &amp; Control study</td>
</tr>
<tr>
<td>Spring 2012</td>
<td>Write dissertation</td>
</tr>
</tbody>
</table>

My plans are to graduate at the end of the 2011-2012 academic year, though I can have continual funding support (from my sponsor, A*STAR, Singapore) till August 2012.

6 **Expected Contributions**

In summary, we expect this thesis to make the following contributions:

1. Define a theoretical framework (question-based explanations) for intelligibility in context-aware applications.
2. Deliver a technical toolkit that implements this framework, and provides automatic support for explanations. Develop design recommendations for deploying intelligibility in context-aware applications.
3. Demonstrate the usefulness of intelligibility for context-aware applications for improving user understanding, trust, and control.

The thesis statement hypothesizes the usefulness of intelligibility, but we had to first define it. This is done through Contribution 1. Next, we have to enable context-aware applications to provide intelligibility, and we accomplished this with Contributions 2 through a toolkit and design guidelines, respectively. Finally, we will prove our hypothesis that intelligibility improves user understanding, trust, and control through three evaluation studies for Contribution 3.

<table>
<thead>
<tr>
<th>Contribution</th>
<th>Project</th>
<th>Section</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Defining intelligibility</td>
<td>Assessing Demand</td>
<td>4.2</td>
</tr>
<tr>
<td>2</td>
<td>Supporting intelligibility</td>
<td>Intelligibility Toolkit</td>
<td>4.3</td>
</tr>
<tr>
<td></td>
<td>Designing for Intelligibility</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Demonstrating the usefulness of intelligibility to improve user understanding, trust, and control</td>
<td>Intelligibility of Question Types</td>
<td>4.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Intelligibility &amp; Uncertainty</td>
<td>5.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Intelligibility &amp; Control</td>
<td>5.2</td>
</tr>
</tbody>
</table>
7 References


