

# iCAP: Interactive Prototyping of Context-Aware Applications

Anind K. Dey<sup>1</sup>, Timothy Sohn<sup>2</sup>, Sara Streng<sup>3</sup> and Justin Kodama<sup>4</sup>

<sup>1</sup> Human-Computer Interaction Institute, Carnegie Mellon University, Pittsburgh, PA USA  
anind@cs.cmu.edu

<sup>2</sup> Computer Science and Engineering, University of California, San Diego, La Jolla, CA, USA  
tsohn@cs.ucsd.edu

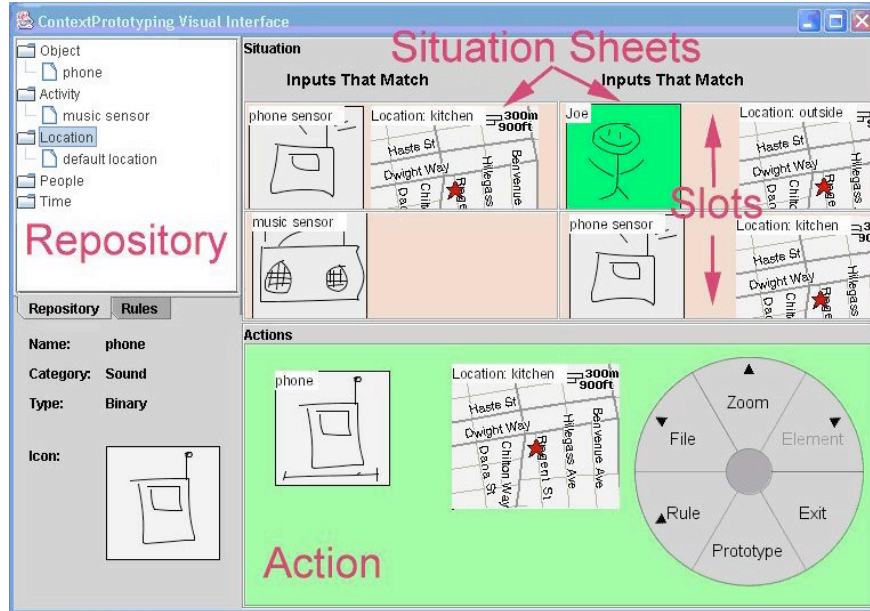
<sup>3</sup> Institute for Informatics, University of Munich, Munich, Germany  
sara.streng@gmx.de

<sup>4</sup> Palm, Inc., Sunnyvale, CA USA  
justin.kodama@palm.com

**Abstract.** Although numerous context-aware applications have been developed and there have been technological advances for acquiring contextual information, it is still difficult to develop and prototype interesting context-aware applications. This is largely due to the lack of programming support available to both programmers and end-users. This lack of support closes off the context-aware application design space to a larger group of users. We present iCAP, a system that allows end-users to visually design a wide variety of context-aware applications, including those based on if-then rules, temporal and spatial relationships and environment personalization. iCAP allows users to quickly prototype and test their applications without writing any code. We describe the study we conducted to understand end-users' mental models of context-aware applications, how this impacted the design of our system and several applications that demonstrate iCAP's richness and ease of use. We also describe a user study performed with 20 end-users, who were able to use iCAP to specify *every* application that they envisioned, illustrating iCAP's expressiveness and usability.

## 1 Introduction

In the past several years, there has been an increased effort and interest in building and deploying context-aware applications. Users' environments contain context information that can be sensed by an application, including location, identity, activity, and the state of nearby people. Context-aware computing involves sensing this context to implicitly provide appropriate information and services. Many groups have developed infrastructures and toolkits to support this next era of ubiquitous computing, however few have focused on empowering end-users in building context-aware applications [5,7,11]. Currently, developing a context-aware application requires developers and end-users alike to either build their application from scratch (involving laborious direct interaction with hardware sensors and devices), or to use an enabling toolkit. While low-level toolkits provide much-needed support for acquiring context [2,3,4,8], large amounts of code must still be written to develop simple sensor-rich applications.



**Fig. 1.** The iCAP user interface has two main areas: a tabbed window on the left that acts as a repository for user-defined objects, activities, locations, people, time and rules and the situation area on the right where these components can be dragged to construct a rule. The rule shown uses 2 situation sheets, where one (on the right) is split into 2 slots. The rule being prototyped is: *If* the phone rings in the kitchen and music is on in an adjacent room *or* if Joe is outside and the phone rings in the kitchen *then* turn up the phone volume in the kitchen.

This inhibits the design of interesting applications, especially for end-users, who end up having little control over how these applications behave. End-users with little technical expertise should be able to exercise control over context-aware systems and rapidly prototype applications. They have more intimate knowledge about their activities and environments than a hired programmer and they need the ability to create and modify applications as those activities and environments change. Without such ability, context-aware applications acting implicitly could significantly annoy users as they fail to meet users' needs.

To address these issues we built the interactive Context-aware Application Prototyper (iCAP), a system aimed at lowering barriers for end-users to build interesting context-aware applications for their instrumented environment, *without requiring them to write any code*. In particular, iCAP allows a user to describe a *situation* and associate an *action* with it. It is a visual, rule-based system that supports prototyping of 3 common types of context-aware behaviors: simple if-then rules, relationship-based actions and environment personalization. In supporting prototyping without writing code, iCAP allows end-users to exert control over sensing systems and dictate application behavior.

The most common context-aware applications are described naturally as a collection of rule-based conditions. A museum tour guide is a common example [1]: “*when a user stands in front of a particular exhibit, show them content related to that exhibit.*” Another canonical example is a smart home [15], with rules like “*turn on the lights whenever someone is in the room.*” iCAP is built on this rule-based paradigm with two separate parts: a visual rule building interface and an underlying rules engine. The interface (Fig. 1) allows users to build, prototype, test and deploy applications. It also separates users from difficulties in dealing with low-level sensor inputs or toolkit-level details. The engine is an event-driven database that evaluates and executes a user’s rules (which specify an application). Context inputs to trigger rules can come from a user-controlled Wizard-of-Oz interface, or from an instrumented context-aware environment. Users of our system could successfully specify and test rules using iCAP.

iCAP supports three common types of context-aware applications. The first type, *simple if-then rules*, are rules where an action is triggered when a condition is satisfied. As discussed earlier (with the tour guide example), many basic applications can be described in this way. The next type is *relationship-based actions*. Humans are naturally relational, and think in terms of *personal*, *spatial* and *temporal* relationships. For instance, I am aware that my roommate (personal) entered the living room five minutes ago (temporal), and that my room and his bedroom are connected by a hallway (spatial). When the system recognizes that my roommate is in the next room, it can prompt me to ask him about going grocery shopping. iCAP provides the necessary support to develop rules that are built on these types of relationships. Lastly, it supports *environment personalization*, where an environment satisfies the differing preferences of its occupants. For instance, one user may enjoy bright lights and rock music, while another prefers dim lights with classical music. To satisfy both users, the environment must account for their preferences and adjust itself accordingly.

The next section describes a formative study we conducted with target end-users to understand their conceptual models of context-aware applications. We then present the iCAP interface through the building of an example context-aware behavior. We then describe how the design of iCAP was guided by our findings. We show that iCAP supports users’ conceptual models through a user study we conducted, where *every* user successfully specified *every* application that they envisioned. We also demonstrate that iCAP can cover an important design space in context-aware computing. We then survey previous research in the areas of context-aware computing and rule-based systems, providing further motivation for our work. We conclude with a discussion of the limitations of iCAP and future directions for this research.

## 2 User Conceptual Models of Context-Awareness

To guide the design of an end-user prototyping environment for context-aware applications, we conducted 90-minute interviews with 20 subjects (9 female, 11 male) with ages ranging from 19 to 52. All participants were experienced with computers but had no experience with programming. Our goal for this study was to gain an understanding of users’ conceptual models of context-awareness: how they want to build context-aware applications and the types of applications they want to create.

To provide some context for the interview, participants were provided with a description of a smart home, a concept that most were already familiar with. This generic smart home had sensors that could sense a large variety of environment state and user activities and could execute services on behalf of users. We intentionally were vague about sample applications and did not discuss any means for specifying them, so as not to bias our study results. Then participants were asked to create their first context-aware application. Specifically, they were asked to describe how, when and where they would want music to play in their smart home. This included both how they listen to music now and hypothetical situations in their smart home. The music scenario was chosen because it is easy to understand and is unconstrained, allowing for creativity and flexibility in specifying application behavior. The last part of the study was more freeform with subjects creating their own scenarios and applications that they found useful and desirable in their home. Here our goal was to collect a large set of user-specified scenarios to better understand users' needs, and descriptions of those scenarios to understand how they naturally conceptualize them.

## 2.1 Study Results

We collected a total of 371 application descriptions from our subjects, including those from the music scenario. Scenario domains were widespread and included temperature and lighting control, cooking, bathing and watching television or controlling entertainment devices. There was quite a bit of overlap among subjects. From these scenarios and descriptions, we uncovered some interesting findings.

First, *every subject* described their applications in terms of *if-then rules*, using the form “*if I ...*” or “*when I ...*” in a particular situation, perform this action’. Fewer than 5% of the rules created used declarative statements instead, for example, “*The nightlight in the bathroom should dim at night*” or “*During parties, usually play hip hop, going out music.*” The uniformity of this result across users was surprising. Almost one-quarter (23.5%) of all rules involved explicit Boolean logic (*e.g.* use of ‘and’ or ‘or’ statements). An example is “*If it is nighttime **and** my roommate’s door is closed with the light off, turn the television off.*”

Second, subjects specified a wide variety of rules, but *most rules* (78.7%) fell into the *simple if-then category*. The remaining rules were divided among the temporal, spatial and personal relationship rules evenly (7, 7, and 6.5% respectively). Less than 1% of rules focused on environmental personalization. However, environment personalization depends on knowing the preferences of the environment’s occupants and 14% of specified rules required knowing some user preference. For example, “*When my alarm clock goes off, turn the volume up louder than normal. But when I wake up, turn it down to normal.*” requires knowledge of what normal is for this person.

Third, the rules that subjects wanted to create were *less complex* than we expected. We broke down each rule into a set of constituent elements, trying to determine the kinds of language used in describing them. After analyzing all the rules, we came up with six categories: Activity, object, location, time, person and state. The rules used an average of 2.5 (SD=0.71) instances of these categories. While this varied across users from a low of 2.0 to a high of 2.9, the results were fairly uniform. An example

rule is “*When I’m cleaning, whether it’s the dishes, vacuuming, or other cleaning activity, play music and have it follow me where I go,*” and it consists of 3 activities and 1 variable location. Another example is “*When I open my pajama drawer, then turn on the hot water in the shower,*” consisting of 2 objects. In general, the older the subject, the more complex the rules were.

We found that 56.4% of all rules involved objects (*e.g.* radio) or the state of objects (*e.g.* radio on/off or volume). In decreasing order of importance were activities (19.1%), locations (12.8%), time (7.6%) and people other than the subject (4%). We were surprised that subjects focused so much on objects, expecting that they would describe rules more in terms of the activities they were performing or wanted to have performed for them. Similar to other work, though, subjects did not mention sensors or sensing [23], only the devices they would normally interact with in their home.

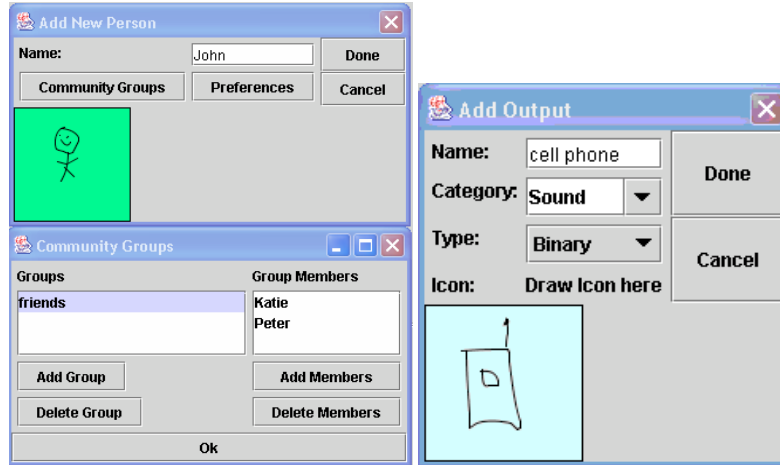
Finally, we learned how subjects naturally conceptualized context-aware applications, along three different dimensions (situation, action, and preference). First, all the rules specified by our subjects had the system detecting a situation of interest. But, there was variation on whether the system was detecting a subject’s state or the state of the house. An example using subject’s state is “*When I’m cleaning, ... play music.*” An example using house state is “*When the water [on the stove] starts to boil, turn off the heat.*” Rules that involved detecting the subject’s state were far more common (70.9% vs. 29.1%). Second, on the action side, subjects viewed the smart home in two different ways: as a piece of technology that they command to perform a context-aware behavior; and, as something that can assist them in performing their own tasks. An example of the former is “*If I leave the house, turn off the lights*”, while an example of the latter is “*When close friends are over, they know a lot of my music, so I’d like to expose them to some new quirky stuff*”. Similar to the results of Truong *et al.* [23], viewing the house as something to command was far more common, covering all but 7 of the 371 rules. Third, subjects greatly preferred (86% vs. 14%) to state specific behaviors rather than to state preferences about what they would like their home to do. A command example is, “*When the water on the stove starts to boil, turn off the heat.*” A preference example is “*It would be good if music was playing that was based on what I was cooking. Like salsa music for Mexican food.*”

Our findings from this study guided the design of iCAP, our end-user visual prototyping system for context-aware applications. Our interface was designed to support the ways in which users overwhelmingly preferred to specify rules (describe a subject’s situation and command the house to act on that state, describing specific behaviors rather than preferences). In the next section, we describe the iCAP interface through an example application.

### 3 iCAP Interface

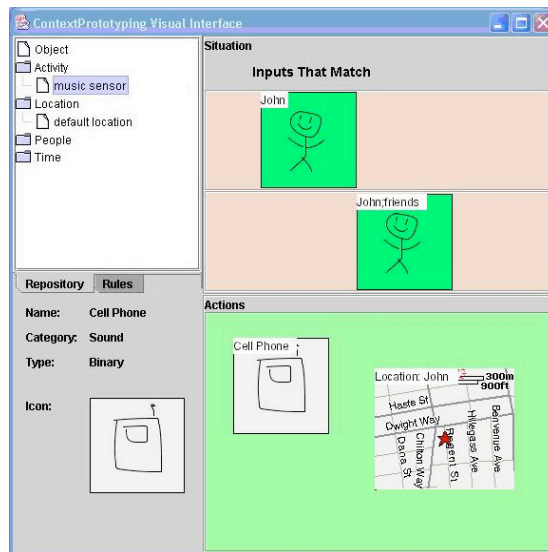
One can imagine a buddy alert application where a user, John, wants to be notified when his friends are in a location adjacent to his. His rule would be:

*IF I am in location1 AND my friend is in an adjacent location (location2)  
THEN beep my cell phone*



**Fig. 2.** (a) Creation of person and his personal groups (b) Creation of “cell phone”

The user first creates the people and artifacts involved in the rule and adds them to the repository (top left of Fig. 1). For each person, the user sketches an icon or selects an image that will be associated with that person, and specifies that it is part of a “friends” group (Fig.2a). He then creates a new output, the “cell phone” and specifies the category of the output (sound), the type of output (binary: ON/OFF, for simplicity), and a sketch/image of the output (Fig. 2b). It is now ready to use in designing rules by simply dragging the appropriate icon onto the rule sheets.



**Fig. 3.** Rule layout for the example rule

The user selects “New Rule” and the system creates one visual area, or *sheet*, on top, in beige, for entering the situation (IF) and another sheet on bottom, in green, for the action (THEN) (Fig. 3). The example situation has two conditions that are specified by laying out sheet: “John is in location1” and “any friend of John’s is in location2”. The user splits the single input sheet into two vertical “slots” that will be related by a conjunction. In the first slot, he drags in the icon of John from our set of inputs. In the second slot, he again drags in the object of John, but specifies that it represents John’s friends. Since he does not specify locations for either slot, the system automatically assigns them variable locations.

To specify the action, the user drags in the cell phone icon from our set of outputs to the action sheet, he sets the action to be “turn on”. The location of this device should be John’s location, so he drags in a Location object (map icon), and sets the location value to be John’s location. Thus the action sheet has two icons that together specify the action “turn on John’s phone”

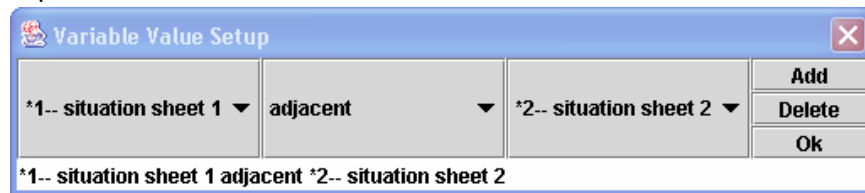


Fig. 4. Resolving the variable value relations.

Name	Sensors	Current Value	Possible Value
office thermometer	thermometer	55.4	55.4
bedroom thermometer	thermometer	60.8	60.8
bedroom lights	light intensity	9	9
office presence	IDENTITY	peter	peter
bedroom presence	IDENTITY	katie	katie peter katie john

-- office thermometer => thermometer @ office state: 55.4  
 -- bedroom presence => IDENTITY @ bedroom state: katie

**Event Log**                      **Simulate Values**

Fig. 5. Prototyping mode where users can simulate values and see event changes throughout the system in the event log.

Finally, the rule is saved and the user is shown a drop down menu to resolve any relations between the variable location values (Fig. 4). He specifies an adjacency relationship, indicating “John’s location should be adjacent to his friend’s location.” iCAP uses information provided by the user or known by the underlying context

infrastructure to determine what locations are adjacent to each other. The rule is saved and appears in the rule panel on the left. He can test his buddy alert application by selecting the prototype mode from a pie menu (see Fig. 1). This launches another window where he can either control (simulate) the relevant context inputs or connect to an existing context infrastructure, and see if his rule behaves as expected (Fig. 5).

## 4 The iCAP System

As almost all of the applications in our study were specified as rules (if <situation> then <action>), iCAP supports the building and execution of context-aware rules and has two main components: a visual rule-building interface and a rules engine that stores the built rules and evaluates them when the rule is being run. The iCAP interface (Fig. 1) provides a simple way for users to design and prototype context-aware applications. The user only needs to deal with defining necessary elements (objects, activities, location, people, time) and use them to visually create situations and associated actions, or if-then rules. No coding is involved, which is a tremendous benefit for non-programmers allowing them to easily explore context-aware computing.

We chose a visual prototyping system because visual programming languages have proven effective in taking advantage of user's spatial reasoning skills [22]. Our study showed that the average user-specified rule has low complexity, so a visual environment seems quite appropriate, supporting both simplicity and familiarity. We iterated on the design of the user interface multiple times, starting with paper prototypes through to the final interface which we present here. At each stage, we obtained feedback from local experts and test subjects. The interface has one window with two main areas (Fig. 1), a tabbed window on the left that is the repository for the user-defined elements and rules, while on the right, a rules area where these components can be dragged to construct a conditional rule. The rules area is split into two areas, one (top) for situations (IF) and one (bottom) for actions (THEN). We built iCAP on top of SATIN, a toolkit for building sketching-based applications [9].

After a number of rules have been defined, the entire rule set can be tested using the iCAP rules engine. The engine can either be set to simulate the context-aware environment, or be used in conjunction with a real context-aware environment via the Context Toolkit [4], an open-source toolkit for enabling programmers to build context-aware applications. Users can interact with the engine in simulation to change the value of defined inputs and evaluate the behavior of the rules being tested. In simulation mode, outputs are visually updated onscreen, whereas with a real environment, real outputs in the physical environment are controlled. With the engine, users can quickly design and test their applications, without having to create an entire infrastructure for collecting or simulating context and without writing any code.

### 4.1 iCAP Interaction

Interaction with iCAP has three steps. First, users create elements relevant to their rules (if they do not already exist in the repository). Second, these elements are

dragged and composed to create rules. Finally, the entire rule set can either be simulated or connected to a live context sensing infrastructure in the prototyping mode.

**Creating Elements** There are five categories of elements in the repository: objects, activities, locations, people and time. These categories (and their order in the repository) come directly from the analysis of the rules from our study (the sixth category found in our study, object state, was combined into the object category). iCAP contains a repository of elements that is populated either automatically by the real sensor-enhanced environment that iCAP is connected to, or manually by the user (or a combination). When the user creates an element, it is associated with a user-sketched graphical icon and added to the repository. Objects have, by default, a binary (on/off) mode and, optionally, a gradient mode (range from 1-10) for objects with different levels (*e.g.* volume, lighting). An example object is the phone in Fig. 1. In addition, objects have a content field used to simulate output modalities. For example, with a music player we could set the content field to “Beethoven Symphony”, which would output that string when turned on, to simulate the playing of a classical music piece. iCAP treats activities like objects. Activities either have a binary mode or a gradient mode. Activity objects detect user or home activities and can only be dragged onto the situation sheet.

People and location elements are created similarly to objects. These objects are essential for relationship and personalization rules. By default, there always exists an “I” people object, since users preferred to write rules about their own state and activity. People objects are created with a name and optional fields (Fig. 2a) for preferences as well as community groups. The system recognizes 3 predefined preferences for a person: lighting, sound, temperature, and uses them to configure objects for environment personalization. Custom preferences such as a music category can be defined with preference being ‘classical music’. People objects can also be created with community groups such as ‘friends’ or ‘family’, allowing the creation of general rule structures such as “*if a family member...*” or “*if a roommate...*”

Location elements specify that a condition or action must take place in a particular location. They simply require a name upon creation. Optionally, a user can indicate what locations are connected together, allowing the creation of rules that take advantage of these spatial adjacencies. In addition, the user can specify whether environment personalization should be turned on for this location, and, if so, the location will attempt to satisfy the preferences of its occupants using the objects in its vicinity.

Time elements allow users to indicate that a situation or an action must occur at a specific day/time. By default, elements correspond to the current time and day.

**Constructing Rules: Simple if-then rules, Relationship-based Action and Environment Personalization** Subjects in our study had strong tendencies to build rules that were command-oriented and specific, making it much easier to support rule construction. After users create necessary elements, they can use them to define rules. iCAP supports the construction of simple if-then rules, spatial, temporal and personal relationship-based actions and environment personalization. As *simple if-then* rules were quite common in our study, iCAP focuses on making these easy to build. Users build these rules by dragging and dropping elements onto the situation and action sheets for each rule. After dragging each icon, the users specify conditions (*e.g.* <, <=, >, >=, =, != and combinations for ranges) governing the behavior of the input.

Our study showed that rules that use simple Boolean logic were quite common so to support these rules, we implemented Pane and Myers' match form scheme [16]. Their matching layout uses the vertical dimension with a label "objects that match" to represent the AND operator and the horizontal dimension for the OR operator. They showed that this layout provides a simple, intuitive way to fully express Boolean logic, for non-programmers, including both children and adults. By default, a rule contains a vertical and a horizontal split to make Boolean logic rule creation more efficient. Fig. 1 illustrates this, describing a rule that turns up the phone volume in the kitchen if the phone rings in the kitchen (top left) AND music is on in an adjacent room (bottom left) OR if Joe is outside (top right) AND the phone rings in the kitchen (bottom right). Users can subdivide the situation sheet more complex logic is needed.

Rules that leverage *spatial relationships* or adjacencies (*e.g.* the next room) are supported during location creation, when users are able to specify what locations are adjacent to the newly created location. When two or more locations are placed on the interface, users can specify the relationship between them (if any). Relative *temporal relationships* such as "before" and "after" are also supported for creating rules. By default, each situation sheet is labeled with "Inputs That Match". A user can click on this label and change it to "Time Ordering", with the first column representing the first event, the second representing an event happening after the first one, and so on. The situation would then be satisfied if these events happened in the desired order. Objects can be set to keep track of a certain time period of activity (*e.g.*, 5 minutes before) or a relative event ordering (*e.g.*, after the next person who walks in). These temporal relationships further exhibit the power of iCAP in building conditional rules.

*Personal relationships* and *environment personalization* are tightly integrated together. Personal relationships are supported through the use of community groups. Examples of community groups include family, friends and co-workers, allowing rules to be created about any of these groups or the individuals in these groups. We support personalization by allowing individuals to set preferences and community groups. By setting a flag for a location, a user can indicate whether the environment should take these preferences into account and change itself when users are in that location. When a person enters a room (with a set flag), the location analyzes the preferences of each person present and tries to satisfy them. We support combinations of personal relationships and personalization, for example, in combining the preferences of all my family members. Preference aggregation to a single result is performed by following a set of heuristics. Because environment personalization was rare in our study, we only support a few common heuristics (*e.g.*, oldest person wins and person who has been in the room the longest wins) and others can only be added by writing code. In future versions, we will allow end-users to visually create these.

**Ambiguity and Conflict Resolution** iCAP allows users to initially create rules that are ambiguous or conflict with each other. A small fraction of the rules from our study used terms like "anyone" and "anywhere". When users do not specify the name of a person or a location or provide a time, these values default to wildcards that will match any value. If when the user saves her rule, there are multiple wildcards, the system prompts the user to disambiguate them. For example, a spatial relationship rule that performs an action when I am in an undefined location and my friend is in an

undefined location, could be disambiguated by setting the 2 undefined locations to be equal or adjacent (or their complements). Also, when rules are saved, the rules engine checks other saved rules for this user to determine whether any of the rules could potentially conflict and highlights these rules for the user to resolve the conflicts or ignore them. If 2 rules conflict at runtime, iCAP, by default, executes the rule that was most recently updated.

## 4.2 Rules Engine

The rules engine sits between the visual interface and a real or simulated context-aware infrastructure. Rules are represented in the engine as a situation and an associated action, with each represented as a Boolean tree. Non-leaf nodes represent Boolean operators (*e.g.*, AND, OR) and leaf nodes contain comparisons (*e.g.*, John is in the bedroom or temperature > 15°) or actions (*e.g.*, call Anne's pager). The rules engine supports all general comparison operations (*e.g.*, =, >, <), as well as relative temporal, spatial and personal relationships for evaluating rules, as described earlier.

**Evaluating the Rules** Evaluation of rules is based on context input received from either a Wizard-of-Oz interface (Fig. 5) and/or a real context environment. Since a real context sensing environment may not be available, we provide three different modes for rule evaluation. The first is a pure simulation interface giving the user control over all inputs through the Wizard-of-Oz interface. This mode allows users to use inputs that may not exist in their environment. The second mode is a real environment only mode. In this mode, all sensors and devices developed in the visual interface are bound to real objects through the context-sensing environment. The sensors (and interpretations of those sensors) and devices that are available in the physical environment are made available for use in iCAP. If a sensor or device made by the visual interface cannot be mapped to an appropriate object in the real environment it is discarded along with any rules that depend upon it. The last mode is a combination of the first two. We call it a "map all possible" mode, where the system maps all possible objects to the real environment, and for those that cannot be mapped, they are allowed to be simulated in the simulator.

**Integration with the Context Toolkit** The engine provides an interface to the Context Toolkit (CTK) [4], to support the automatic population of the iCAP repository, passing of events from the real environment to the rules engine, and to support the rules engine in executing actions in the real environment. Leaf nodes in our Boolean trees act as queries to a discovery service in the CTK, enabling them to bind to real-world sensors and actuators in a user's smart environment.

If either of the two modes that rely on a real context environment are active, the rules engine will attempt to bind sensors and devices to the Context Toolkit. For sensors and actions, the engine will construct a description query using each element's location, name and type (Boolean/gradient). This description query is given to the CTK's discovery system to locate any components in the environment that can provide the requested information. If an appropriate component is found, it will be the sole provider for the sensor (or service for the action) in the rules engine. iCAP is

able to send events to and receive events from a real context infrastructure, making it a real tool ready for deployment.

There does need to be a common naming scheme between objects in iCAP and in the Context Toolkit. We anticipate additional features in iCAP that would show all available components and services in the CTK and allow the user to choose from these when composing rules. Our main goal here is to demonstrate that iCAP can send events to and receive events from a real context infrastructure, making it a real tool ready for deployment.

## 5 Validation

In the preceding sections, we motivated and described the design of iCAP. Although its iterative design was guided by our findings from our formative study, we still need to determine how usable it is and whether it supports the conceptual models we discovered. Here, we validate iCAP by answering the following questions:

- Can users use iCAP to easily and accurately build context-aware rules?
- Does iCAP support the conceptual models that we elicited in our formative study?
- Does iCAP support the construction of an appropriate range of rules?

We address the first two questions through an initial user study we conducted on iCAP. We address the third question through an analysis of the set of rules provided by our subjects in our formative study, by showing how iCAP covers an important design space from the literature and by showing how iCAP enables end-users to build canonical applications taken from the literature.

### 5.1 User Study

Although the design of iCAP was guided by our findings from our formative study and we followed an iterative design process, we still need to determine how usable it is and whether it supports the conceptual models we discovered. To do this, we conducted a study of 20 non-programmers (age range from 23-67, 10 males, 10 females) using iCAP to perform a set of open-ended tasks and fixed tasks. As with our formative study, we described the concept of a smart home, which all subjects were familiar with. We asked each subject to write down succinct descriptions of 3 different applications they would find desirable and useful. Subjects were presented with iCAP and a short tutorial and asked to create and test their 3 applications. We were most interested in seeing whether iCAP allowed users to specify rules in ways natural to them. The applications our subjects came up with spanned a wide range of domains, overlapping the results from the previous study, but including new ones like reminders to take medicine, notifying users when a baby woke up, and routing phone calls to a user's location (landline phone vs. cell phone). Users were asked to implement their open-ended rules as close to their written specification as possible.

In the second part of our study, users created a set of 7 rules we specified to test general usability. These rules were taken from the set of user-specified rules from our formative study and included 3 straightforward if-then rules with varying complexity

(using 2, 3, and 4 elements respectively), 1 if-then rule that required users to resolve ambiguity in a person’s identity, 1 spatial relationship rule that required users to resolve ambiguity in location, 1 personal and 1 temporal relationship rule.

In both study parts, users had no trouble creating rules, or the elements for each of their rules. All users were able to complete all rules in a reasonable amount of time, including the time to create the necessary elements for each rule and test the rule, using Wizard-of-Oz testing, verifying that the correct action was taken when they manually set the appropriate contextual conditions. From our own experiences, the time end-users took to complete each rule is less than it would take a programmer using the Context Toolkit to create an application that supports it (Table 1). More importantly, each rule was implemented by an end-user who, without iCAP, would not have been able to do so.

**Table 1.** Average rule completion time (and standard deviation), in minutes.

Type	Rule	Time
If-Then	If I’m sleeping, turn the stereo off. (complexity level is 2)	3.05 (0.67)
If-Then	If I’m in the living room after 10pm, dim the living room lights. (complexity level is 3)	3.36 (0.74)
If-Then	When I walk into the kitchen and turn on the stove, turn on the television with the volume low. (complexity level is 4)	3.58 (0.71)
If-Then	If anyone is sleeping at noon, turn on his/her alarm clock (person identity ambiguity)	3.17 (0.48)
Spatial	If I am in a room next to Karen, page me. (spatial ambiguity)	2.86 (0.54)
Personal	If my roommates are not home, turn my favorite music on high.	3.72 (0.83)
Temporal	When I go from the kitchen to the bedroom, turn the lights on in the bedroom.	3.54 (1.23)

Users were able to specify elements, specify preferences (*e.g.* my favorite music) and create community groups (*e.g.* roommates), although the latter two were slightly more time consuming. A few users objected to the amount of time element creation took and suggested that this be done automatically. (Note that iCAP can populate the element repository automatically using the Context Toolkit to collect components from a smart space, however, we wanted to study how users created these elements on their own.) Users did not have any difficulty creating rules that involved resolving ambiguity (*e.g.* *if anyone is sleeping, turn on his alarm clock*). In each case, they simply created their rules with ambiguity in them, and resolved that ambiguity when iCAP prompted them to. All users were able to use the Pane and Myers matching scheme [16] to correctly define ‘AND’ and ‘OR’ relationships. While spontaneously thinking aloud, several users referred to the label on the sheet and said “it says inputs that match, so that means everything on this sheet is an AND relationship”. Because iCAP automatically divides the situation area into 4 quadrants for creating Boolean relationships, actually specifying Boolean rules was quite simple for them. However, some users had trouble with the overloaded column operator. In most cases, columns support ANDing two expressions together, but when used in temporal rules, they represent an ordering of events. This overloading confused a few users who had trou-

ble remembering where to click to achieve a temporal event ordering. Finally, users were able to successfully test their rules using the Wizard-of-Oz prototyping interface. For each rule, they verified that the correct action was executed when they manually set the appropriate contextual conditions.

During the first phase of our study, iCAP readily supported users in implementing *every one of the applications* they had described. There was always a simple and direct mapping from the written rule to the visual specification in iCAP. We attribute this to the fact that this group of users shared the same predominant mental models with those in our formative study. The majority of the rules specified were simple if-then with only a few relationship-based rules, and consisted of about 3 elements each. In their descriptions, users focused on home objects and their own activities, and not sensors. For example, one wrote “*when I close the kitchen door, lock the door and turn off the kitchen light,*” readily specified visually with door, location (kitchen) and people (I) objects for the situation and door, light, and location objects for the action.

During the second phase of the study, users were again able to specify *every rule asked of them* using iCAP. Table 1 summarizes the types of rules and average times (with standard deviation) required to specify them.

**Summary** Our evaluation of iCAP showed that it was both usable and supported users’ conceptual models. Users were successful in building rules specified by users and by us. These rules spanned the range of simple if-then rules, environment personalization and personal, spatial and temporal relationship-based actions. In the first, open-ended phase of the study, users often constructed rules quite different than those envisioned by us, and were always successful in specifying them in iCAP. While users had some issues with iCAP, all indicated that they would use it if hardware to outfit a smart home were readily available. One user said iCAP was “exactly what I imagined a visual programming interface would be like to develop these applications.” Another user described it as “easy to use” and “fun and exciting to be able to easily build applications for my home.”

## 5.2 Context-Aware Design Space

As further validation for iCAP, we show that iCAP facilitates the building of a wide variety of context-aware applications, supporting almost all the rules obtained in our formative study, an important design space in context-aware computing and canonical applications taken from the literature. We analyzed each of the 371 rules from our formative study, and found that iCAP could *support all but 12* of them. Three rules involved actions to be repeated based on time intervals that we did not support (e.g. “*Remind me to feed my bird every 3 days*”). Five rules involved too much ambiguity to create them, either in the specification of what the action is (e.g. “*When close friends are over, they know a lot of my music, so I’d like to expose them to some new quirky stuff*”) or in which action to perform (e.g. “*If not in my room with friends over, then turn the music off or turn it down*”). Four rules were limited by iCAP’s ability to express complex concepts (e.g. “*It would be good if music was playing that was based on what I was cooking. Like salsa music for Mexican food.*”). Overall, iCAP

can support the vast majority of context-aware behaviors that users want to build, including all the behaviors from the first part of the iCAP user study.

iCAP also supports the 4 application categories defined by Schilit in his seminal work on context-awareness: Context-triggered actions, automatic contextual reconfiguration, contextual information and commands, and proximate selection [21].

Allowing users to visually create applications with *context-triggered actions* (if-then rules specifying how a context-aware system should adapt) has been the basis of our work in iCAP and we have described several examples. An example of *automatic contextual reconfiguration* taken from [21] is reconfiguration to share an object/resource among people. iCAP supports this through environment personalization where, for example, a room's lighting devices are adjusted to meet the preferences of its occupants. An example of *contextual information and commands* is Schilit's location browser that presents information relevant to a user's location. We support this in an application that delivers a list of people in adjacent rooms to a user's cell phone and updates this when the user enters a new location. We built a *proximate selection* (*i.e.* local objects are emphasized or made easier to choose) application that displays available output devices in the user's current location on fixed displays in the room.

Finally, we have used iCAP to build a variety of canonical context-aware applications taken from the literature including tour guides [1], reminder systems [12], and environment controllers [15]. Tour guides can be built by creating a collection of rules that present user-defined content when someone enters a particular location, similar to Schilit's contextual information and command category. Content delivery can be customized based upon a user's profile or preference, *e.g.*, to display information about the building information to a user interested in architecture. A reminder system can be built in much the same way, by creating a number of rules that deliver content, a reminder, when the user is in a particular situation. For example, "*when I am in a room adjacent to Katie, remind me to give her the book I borrowed from her*". Finally, iCAP supports home automation systems or environment controllers applications through rules that control heating and lighting conditions and environment personalization when occupants of a space have differing preferences.

## 6 Related Work

**Context-Aware Computing** Since Weiser's vision of ubiquitous computing [24] more than a decade ago, many groups have explored the domain of context-aware applications. Architectures and applications such as stick-e notes [17] and GeoNotes [6] focused on allowing end-users to contextually share data with each other by placing virtual objects in a context-aware environment. However, many of these types of applications were being written from scratch with a high development cost [18]. One must interface to an external sensing system, gather the appropriate sensor data, develop a rule-based engine, and execute the desired actions. With these architectures, there is little support for rapidly prototyping applications. While existing infrastructures that enable programmers to build context-aware applications, such as JCAF, SOLAR, Context Toolkit and Context Fabric [2,3,4,8], support many of these steps, they do not provide any interface for allowing end-users to use them.

Various commercial home automation products like X10 ActiveHome and Vantage QLink, are readily available, however these mainly provide interfaces to directly control the hardware in a home. QLink provides a text-based configuration interface for end-users but it does not support the use of context and it requires in depth knowledge of the hardware deployed which most end-users would not have. In essence, existing systems do not provide support for context-aware application prototyping by end-users, thus demonstrating a need for a system like iCAP.

**Visual Rule-Based Systems** We chose to make iCAP a visual environment for users to prototype context-aware applications for reasons of simplicity and intuitiveness. Visual programming languages have proven effective in taking advantage of user's spatial reasoning skills [22]. This programming style is not only simple and effective for many types of users, but is especially intuitive for end-users. When applied to a new domain like context-aware computing, visual programming provides tools needed to allow creative end-users to easily build novel applications. Although a text environment could have been used, Pane and Myers [16] found that in rule generation tasks, users generated more accurate Boolean rules using their graphical technique than textual methods. We have applied this technique to the context-aware domain. While text-based interfaces provide increased expressiveness, we focus on a graphical interface here to increase end-users' ease of use in creating applications. Our formative study showed that users think about context-aware applications in terms of rules, and indeed most context-aware applications can be *and are* described this way, so we chose to make iCAP a rule-based system despite well-known drawbacks of rules: limited expressiveness, may be easily broken, hard to detect and deal with rules that have conflicting actions. In addition to the work of Pane and Myers described above, Mackay, *et al.* showed in the Information Lens project that people with little computer experience could create and use rules effectively to filter email [14].

In earlier work, we presented a Cappella, a programming by demonstration interface for end-users to build context-aware applications [5]. While it addresses a similar problem space as iCAP, it is intended for building applications that are difficult for a user to express directly. We were inspired by the CAMP system, a magnetic poetry system for allowing end-users to construct capture-and-access applications. Similar to iCAP, the CAMP interface was grounded in a study of users' conceptual models about the application domain. However the variety and complexity of rules we found in our study does not match well to the restricted vocabulary used by CAMP. Agent-sheets capitalizes on the idea of visual rule-based programming by allowing end-users to establish relationships among different autonomous agents [19]. Although it could be expanded to support context-aware applications, it still required a high level of expertise to use [20], and supports limited sensing and actuation. In contrast, we aim to provide even novices with the ability to build context-aware applications. The Alfred system uses recordable speech-based macros to support users in building applications for a smart environment [7]. However, Alfred focuses on rules based on explicit user interaction (pressing a button, speaking a phrase), and unlike iCAP, does not support conditions based on contextual cues. The Jigsaw Editor addresses novice users by supporting end-user reconfiguration of home devices using a novel jigsaw puzzle metaphor [11]. However, the creators recognize the limits of a constrained

metaphor and state that they “do not seek the richness of programming expression allowed by iCAP” [11]. Our goal is to support the building of expressive applications by novice users, trading off some learnability for this expressiveness. Mobile Bristol and Topiary demonstrate the value of supporting designers in building location-aware systems [10,13], however they do not provide support for applications or rules without interfaces.

**Summary** There is a need for a context-aware prototyping environment that enables end-users to build rule-based context-aware applications. By building upon work in visual rule-based systems, we address this by providing an effective prototyping tool, empowering end-users to build interesting context-aware applications that cover an important design space in context-aware computing.

## 7 Conclusions and Future Work

In this paper, we presented iCAP, a visual prototyping system for context-aware applications. iCAP is a visual rule-based environment that supports end-users in prototyping context-aware applications without writing any code. iCAP provides two main benefits: opening up the design space of context-aware application design to a larger group of users than just programmers, and giving control of what should happen in a context-aware environment to the people it most affects, the end-users.

iCAP supports users in designing and implementing a context-aware application, testing it under simulated and real conditions and revising it, as needed. In particular, it supports the creation of if-then (or situation-action) rules that are triggered by contextual cues, the building of spatial, temporal and personal relationship-based rules, and the building of environment personalization systems.

iCAP’s design was based on a formative study of 20 end-users that demonstrated the appropriateness of rules as a mental model for end-user construction of context-aware applications. After constructing and iterating on iCAP, we validated its usefulness in two ways. First we ran a user study with 20 end-users who successfully used iCAP to create *every* application they envisioned or were asked to create by us, in less time than it would take to program them. Our subjects told us that iCAP was a powerful system they would like to use in the future. We then showed that it could be used to build almost all the applications from the formative study, canonical context-aware applications and ones that covered Schilit’s design space.

Most systems designed for end-users have more constrained functionality than systems designed for programmers. While we have used iCAP to build a wide variety of context-aware applications, it is not as expressive as existing programming systems for building applications [2,3,4,8]. To make it more expressive, we need to support more sophisticated Boolean logic, the ability to activate and deactivate rules based on contextual cues and support for ambiguous context. In addition, we would like to extend iCAP to support context-based retrieval systems that tag captured information with contextual cues to aid future retrieval [12]. iCAP can already capture contextual cues, so we would need to add the ability to store those cues persistently and attach them to user-provided content and provide a mechanism for querying the cues and content. We would like to increase the expressiveness of iCAP while, at the same time, maintaining its ease of use and increasing its learnability to improve users’

performance in creating rules. One approach we will explore is to provide support for both visual and textual specification of rules. Finally, we will deploy iCAP in a real environment to understand how users will use it in practice, dealing with rules that evolve over time, conflicting rules and more complex rules.

## References

1. Abowd, G.D. *et al.* Cyberguide: A mobile context-aware tour guide. *ACM Wireless Networks* 3(5). pp. 421-433, 1997.
2. Bardram, J.. The Java Context-Awareness Framework (JCAF) – A service infrastructure and programming framework for context-aware applications. *Pervasive 2004*, 98–115.
3. Chen, G. and Kotz, D. Solar: An open platform for context-aware mobile applications. *Pervasive 2002*. 41-47.
4. Dey, A.K., Salber, D. and Abowd, G.D.. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human-Computer Interaction Journal*, 16(2-4), 97-166, 2001.
5. Dey, A.K. *et al.* a Cappella: Programming by demonstration of context-aware applications. *CHI 2004*. 33-40.
6. Espinoza, F. *et al.* Geonotes: Social and navigational aspects of location-based information systems. *UBICOMP 2001*. 2-17.
7. Gajos, K., Fox, H. and Shrobe, H. End user empowerment in human centered pervasive computing. *Pervasive 2002*. 134-140.
8. Hong, J.I. and Landay, J.A. An infrastructure approach to context-aware computing. *Human-Computer Interaction Journal*, 16(2-4). 287-303, 2001.
9. Hong, J.I. and Landay, J.A. SATIN: A toolkit for informal ink-based applications. *CHI 2000*, 63-71.
10. Hull, R., Clayton, B. and Melamed, T. Rapid authoring of mediascapes. *Ubicomp 2004*, 125-142.
11. Humble, J. *et al.* ‘Playing with your bits’: user composition of ubiquitous domestic environments. *UBICOMP 2003*, 256-263.
12. Lamming, M. and Flynn, M. Forget-me-not: Intimate computing in support of human memory. *International Symposium on Next Generation Human Interfaces 1994*. 125-128.
13. Li, Y., Hong, J.I. and Landay, J.A. Topiary: Tool for prototyping location-enhanced applications. *UIST 2004*, 217-226.
14. Mackay, W.E. *et al.* How do experienced Information Lens users use rules? *CHI '89*. 211-216.
15. Mozer, M.C. The neural network house: An environment that adapts to its inhabitants. *AAAI Spring Symposium on Intelligent Environments*. 110-114, 1998.
16. Pane, J.F. and Myers, B.A. Tabular and textual methods for selecting objects from a group. *IEEE International Symposium on Visual Languages 2000*. 157-164.
17. Pascoe, J. The Stick-e Note Architecture: Extending the interface beyond the user. *Intelligent User Interfaces 1997*, 261-264.
18. Pascoe, J., Ryan, N. and Morse, D. Issues in developing context-aware computing. *HUC 1999*. 208-221.
19. Repenning, A. and Citrin, W. Agentsheets: Applying grid-based spatial reasoning to human-computer interaction. *IEEE Symposium on Visual Languages 1983*. 77-82.
20. Scerri, P. and Reed, N. The EASE actor development environment. *Swedish AI Society 2000*.
21. Schilit, B., Adams, N. and Want, R. Context-aware computing applications. *Workshop on Mobile Computing Systems and Applications*, 1994.
22. Shu, N.C. Visual Programming: Perspectives and Approaches. *IBM Systems Journal*, Vol. 28. 525-547, 1989.
23. Truong, K.N., Huang, E.M. and Abowd, G.D.. CAMP: A magnetic poetry interface for end-user programming of capture applications for the home. *Ubicomp 2004*, 143-160.
24. Weiser, M. Computer for the 21<sup>st</sup> century. *Scientific American*, 265(3). 94-104, 1991.