Reasoning about Dynamic Information Displays

J.C. Campos (1) and G.J. Doherty (2)

(1) Departamento de Informática, Universidade do Minho, Campus de Gualtar, 4710-057 Braga, Portugal. e-mail: Jose.Campos@di.uminho.pt

(2) Department of Computer Science, Trinity College, Dublin 2, Ireland. e-mail: Gavin.Doherty@cs.tcd.ie

Abstract. With increasing use of computing systems while on the move and in constantly changing conditions, whether it is via mobile devices, wearable computers or embedded systems in the environment, time plays an increasingly important role in interaction. The way in which information is represented in an interface is fundamental to interaction with it, and how the information is used in the users tasks and activities. *Dynamic* representations where the user must perceive changes in the information displayed over time pose a further challenge to the designer. Very often this information is integrated with information from the environment in the performance of the user's tasks. The diminutive size and limited display capabilities of many ubiquitous and mobile computing devices further motivate careful design of these displays. In this paper we look at how time can be taken into account when reasoning about representational issues from the early stages of design. We look at a model which can be used to reason about these issues in a structured fashion, and apply it to an example.

1 Introduction

Reasoning about design, in the context of interactive systems, implies reasoning about how useful and easy to use those systems will be. In this context, assessing the quality of a design is no easy task. Although there are a number of human-factors oriented studies which have resulted in design guidelines and rules, these cannot be turned directly into a set of properties that all systems must obey. In a specific design context, whether a guideline is applicable or not is always debatable. It might even be the case that a guideline is wrong [19]. This is especially true when designing for novel interaction techniques and paradigms.

Time plays an important role in the "interaction experience" [13], and as such should be considered when designing interactive systems. For users learning a new system, time plays a role in associating actions with their effects, and in building a sufficiently good conceptual model of the system to predict future states of the system. Even for experienced users, time plays a role in reasoning about the cause of events or effects perceived in the system, particularly where continuous dynamic information is involved.

Issues of representation are fundamental in what we perceive and the way we think and solve problems [10]. The increasing use of novel physical form factors is likely to increase the importance of external representations in information technology applications [23]. A particular subset of information representations are dynamic information representations - those which rely on observation over a period of time.

In this paper we look at how time can be taken into account when reasoning about issues of representation from the early stages of design. Thus our focus is on *dynamic* information displays. Specifically, we propose a model for reasoning about representational issues where time is involved, and apply it to an example information display. This work builds on previous work on representational reasoning in [6] and [5]. By performing careful analysis of these issues we hope to identify useful properties and guidelines for dynamic information displays, and to understand the scope and validity of any properties proposed.

2 Information Representations and Time

Time plays a role in both how systems are used and perceived. In [8] it is discussed how small differences in the time it takes for a user to perform different actions can influence the interaction strategies adopted. The time the system takes to react to user input can also have a great impact on the usability of the system [4, 3, 20]. For example, if the system takes too long to respond to user actions, it will be difficult for the user to establish what effects are associated with his or her actions.

Time plays an even more important role in emerging interaction techniques and paradigms. Mobile and wearable devices operate in environments where conditions are continuously changing, but sensing and adapting to changing context, and accessing remote resources may take time [11]. Attentional demands due to other tasks the user is performing simultaneously may further reduce the time which can be spent observing information presented in the interface. In mobile applications, parameters such as quality of service, or even intermittent provision of service, have a great impact on interaction, and the time dimension is involved when studying how to design such interfaces. Also, multimedia and augmented reality systems usually involve continuous dynamic information being presented to the users. Listed below is a summary of some of the different ways in which time can affect the interaction between user and system.

Attentional demands. Other simultaneous activities reduce the amount of the users time which can be devoted to interaction.

Task demands. Task has a timed component, that is, some portion(s) of the task must be completed within a certain tim interval.

Observation/learning/causal demands. If interface is not fast enough, then it does not support learning, causal reasoning.

User satisfaction. Very slow interfaces are not engaging or satisfying to use.

Error free interaction demands. There is a tradeoff between the pace at which the user can or must perform a task, and the likelihood of making an error. Interface design can have a big effect on the pace at which a task proceeds.

Dynamic perceptual demands. Only by observing interface over a period can the needed information be extracted.

Another relevant area is that of supervisory and manual control, where users operate or control a system in real time. Examples of analysis of this kind can be found in [21] and [2]. In both cases the interaction between a pilot and the automation of an aircraft is analysed regarding the control of the aircraft's climbing behaviour. Both approaches use model checking tools which are discrete in nature. [2] discusses how continuous systems can be analysed with a discrete tool using abstraction. In [7] the use of hybrid automata for the specification and analysis of continuous interaction is discussed. This type of approach allows for a more direct representation of the continuous aspects of interactive systems. All the approaches above tend to concentrate on behavioural aspects of the system, that is, they are concerned with actions which might be taken by the user, and the potential consequences (in terms of the state of the system) of these actions. The issue of what the user can perceive in the interface and how this might affect the actions they take is not addressed.

2.1 Empirical and Analytic Methods

Reasoning about design, in the context of interactive systems, implies reasoning about how useful and easy to use those systems will be. Factors influencing the usability of a system range from pure software engineering concerns to psychological, sociological, or organisational concerns. Hence, in most situations, only actual deployment of the system will give a final answer concerning its usability. It is still the case, however, that there is a need to consider usability issues during the design process.

Usability evaluation methods can be divided into two major groups (see [15, Part III]). *Empirical techniques* rely on building prototypes of the system being developed and testing them using real users under controlled conditions. *Analytic techniques* rely on confronting models of the system with how users are expected to behave. Empirical methods based on detailed prototypes of the system being developed can be useful in validating design decisions in conditions as close to actual usage conditions as possible. The greatest drawback of such methods is that they are difficult to apply in the early stages of design when most decisions have to be made, and hence exclusive use of such methods would require extensive (and costly) development and evaluation cycles. To address this issue discount approaches can be adopted where users more directly participate on the design process by evaluating early prototypes or paper mock-ups of the envisaged system, but it can be difficult to address subtle issues (eg. concerning time and interaction context) using such techniques.

Analytic methods fill this gap. By being based on models of the system, and not requiring a prototype to be built and placed in a plausible interaction context, analytic methods have the potential to allow reasoning about the usability of a system to be carried out early in the design process. The aim of such methods is to identify as many potential problems as possible in the earliest stages of development, thus reducing the number of development and evaluation cycles needed to produce a usable system. Their drawback is that assumptions have to be made about user behaviour, and the results can only be as good as the assumptions that are made.

Traditional analytic methods tend to be carried out manually and more or less informally [15]. This can work in areas where the technology is well known, and the design of the system is not too complex. However, as new interaction techniques and paradigms emerge, or as systems become more complex, this type of approach becomes less likely to deliver. Furthermore, where real-time issues are involved some problems will be difficult to reason about without recourse to clear and concise representations of system designs and properties.

The use of structured models which are amenable to rigorous analysis during development can impact system design at two levels:

- the use of concise mathematical concepts and notations can help in the organisation and communication of ideas.
- mathematical models can allow rigorous reasoning about properties of the system being designed.

Although the modelling process itself can provide valuable insight into the system being designed, the possibility of formally reasoning about the models with the aid of tool support can bring additional benefits at relatively low cost. Hence, in recent years formal verification of interactive systems has been an active area of research (see, for example, [14, 12, 1]).

2.2 Properties and Guidelines

Traditionally, quality will be measured against a set of properties that the system or artifact must exhibit. Trying to devise a meaningful set of properties, that should be true of an interactive system in order to guarantee its quality, is no easy task: there is no magic recipe for easy interactive systems building. Guidelines are (or, at least, should be) of a qualitative and high level nature, which means that they are not easy to verify in a rigorous way. They must first be turned into concrete properties. A typical guideline might be: "*Reduce cognitive load*" [18]. Design rules, on the other hand, are about very specific interface features, which means that formal verification might not be, in many cases, the best approach. A typical design rule might be: "*A menu should not have more than seven entries*". In a specific design context, whether a guideline is applicable or not is always debatable. It might even be the case that, given a specific design context, trying to make the system comply to some general purpose guideline might be detrimental [19]. This is especially true when designing for novel interaction techniques and paradigms, which were not considered when devising currently available guidelines.

In the field of software engineering, lists of properties have also emerged (see, for example, [22, 9]). However, these tend to be governed mainly by the specific style of specification being used, and what that style allows to be expressed. Their relevance towards usability is not always completely clear.

Contributing to this problem is that fact that interactive systems form an increasingly heterogeneous class of systems. In fact, the only common requirement is that the system interacts with a human user effectively. This accounts for systems from airplane cockpits and control rooms of nuclear power plants, to mobile phones and set-top boxes. Additionally, the presence of the human user means that assumptions must be made about the capabilities (physical, cognitive, and perceptual) of the particular users the system is intended to serve. So, instead of trying to establish a list of properties, we should try to identify some specific issues related to interactive systems design, about which the designer might wish to be assured that the system is satisfactory. We do this by paying attention to what is generally true of all systems and all users.

3 A Framework for Analysis

Rather than conjecture about users internal representations of time, we wish to provide a framework which illustrates qualitative differences between design alternatives. We focus on representations of information in the interface which change over time, where this change is relevant to the users tasks and activities.

We start with a reworked version of the model proposed by [6] (see Figure 1). In [5] we show how this model can be used to reason about the properties of the interaction between a pilot and a cockpit instrument during the landing procedure. From

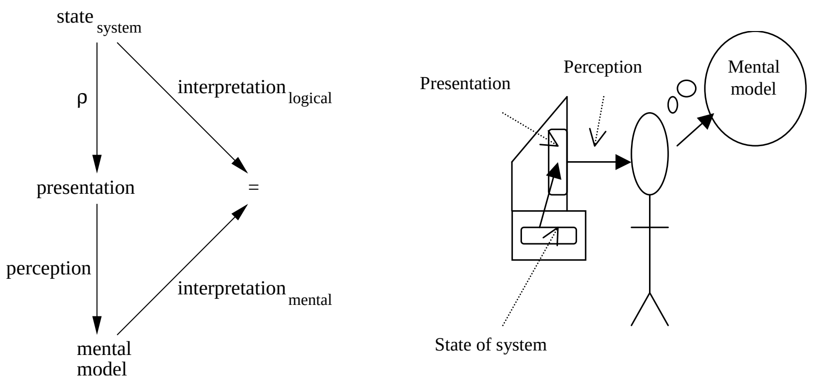

Fig. 1. Basic model

the diagram in Figure 1 it is possible to identify the three basic entities involved in the interaction:

- functional core - the system state and its operations;
- user interface - the presentation and possible user actions;
- user - the person using the interactive system (building a mental or conceptual model of the system and trying to fulfill goals by issuing commands).

The diagram also introduces two basic mappings:

- ρ (presentation mapping) - this map expresses the fact that the presentation of the system can be seen as a reification of the system's state.
- perception - this mapping captures what is assumed the user will perceive of the user interface presentation. It can be seen as a filter which the user applies to the information presented at the interface, in order to construct a mental model of the system state.

How users will perceive the user interface will depend on the actual system being analysed. The type of property and user being considered also has a bearing on what perception relation to consider. Hence, defining an appropriate perception relation for a specific system design requires discussion between software engineers and human-factors experts. The diagram introduces two additional mappings: interpretation$_{\text{logical}}$

and interpretation$_{\text{mental}}$. What these mappings express is that, in order for the presentation to be found adequate, it must allow the user to build a mental model which is sufficient for carrying out the required tasks and activities. Because we are dealing with cognitive issues, assumptions must be made about how the user interprets the presentation. This is captured by the interpretation$_{\text{mental}}$ mapping. Both mappings will be dependent of the specific aspect/property under consideration.

Using this model, we can reason about whether a given presentation is adequate. In purely representational terms, an interface is said to be correct if its presentation enables the user to build (with a set of assumed perceptual and cognitive capabilities) an accurate model of the underlying system state. This can be expressed as:

$$\text{interpretation}_{\text{mental}}(\text{perception} \circ \rho(\text{state}_{\text{system}})) = \text{interpretation}_{\text{logical}}(\text{state}_{\text{system}}) \quad (1)$$

That is, the user's mental model of the system matches the system state, in so far as the model relates to assumptions about what is considered relevant of the interaction process. The equation above relates to what Norman calls the Gulf of Evaluation: "the amount of effort that the person must exert to interpret the physical state of the system" [16].

3.1 Dealing with Time

This basic model is static. No notion of change is present in either the system state, the presentation, or the mental model. To better deal with change, [1] extends the model to accommodate the notions of system operations, interface actions, and user goals. The new version of the model introduces three additional mappings:

- operation$_{\text{system}}$ – system operations map system states to system states. They are intended to represent the basic functionality of the system.
- action$_{\text{interface}}$ - interface actions map presentations to presentations. Each interface action will typically be associated with one or a sequence of system operations. Interface actions can occur as a consequence of user activity, or as a consequence of a system operation.
- goal - goals map mental models to mental models. They are used to capture the intentions of the users. Typically a user goal will cause a number of interface actions to be executed. This mapping can also be used to capture changes in the user's mental model which are caused by changes in the presentation.

In this model change is represented explicitly, which allows us to reason about Norman's Gulf of Execution [16]: the *distance* between what the user wants to achieve (the goal), and what is possible at the interface.

$$\text{goal} \circ \text{perception} \circ \rho(\text{state}_{\text{system}}) = \text{perception} \circ \text{action}_{\text{interface}} \circ \rho(\text{state}_{\text{system}}) \quad (2)$$

The equation compares the result of executing an interface action with the goal the user had in mind. The model in Figure 2 is biased towards event based systems and it still does not explicitly account for time. Furthermore, change is not always directly connected to the concept of action. For example, the analysis in [6] concerns determining whether the representation used in an information display (a progress bar) enables the user to perform a task (to detect progress). The analysis concerns relationships between system states and user mental models which change over time, regardless of how the change comes about. In order to better express such situations we change the model to that in Figure 3.

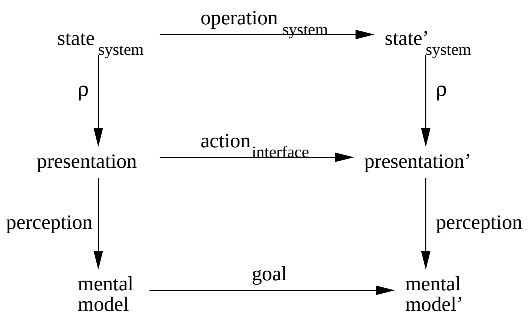

Fig. 2. Model based on actions

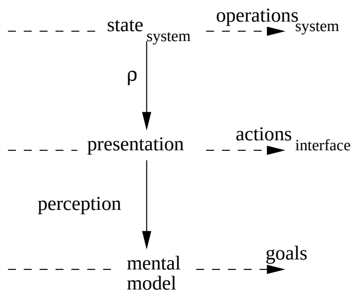

Fig. 3. Model based on time

The diagram shows how all three levels of the model change continuously with time.Note that if we define a number of discrete time slices we will be in the situation of the previous model, since each time slice can then be seen as an action. Using this model, the relation between system state and user's mental model can be expressed using the following type of equation:

$$\forall_{t_1,t_2} \cdot \mathrm{interpretation}_{\mathrm{logical}}(\mathrm{System}, t_1, t_2) = \mathrm{interpretation}_{\mathrm{mental}}(\mathrm{MentalModel}, t_1, t_2) \quad (3)$$

What the equation expresses is that the evolution of the user's mental model over time must match the evolution of the state of the system. Similar equations can be written to express relations between the presentation and the system state or the user's mental model.

4 An example

In this section we present an example of how the model introduced in the previous section can be used to reason about the perceptual properties of a user interface. We will show that we can use this reasoning to validate design decisions during the early stages of development. The notation used is that of the PVS tool [17], which is also used to explore some of the properties proposed.

Let us consider an interactive system with a dynamic information display - a GPS device. Location and direction information is continuously updated (although there can

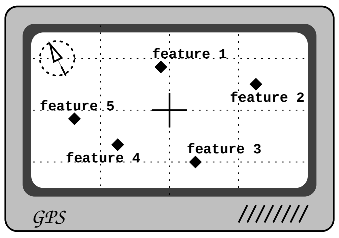

Fig. 4. Initial design

be discontinuities in the data itself due to the nature of the technology). We will consider a device that is capable of displaying the position of a number of features relative to its current physical location. We assume that no device level actions are available to the user, but the user can move the device around and look at the annotated map it presents.

We will consider a design goal which is relevant to common navigation tasks: users should be able to know where they are.

Device model Developing the device and interface models is a relatively straightforward recording of the envisaged design. The state of the device can be defined by the position of each relevant feature plus the location and heading of the device:

```
wposition: TYPE = [# ypos: real, xpos: real #]
State: TYPE = [# features: [name → wposition],
                 location: wposition,
                 heading: Angle #]
System: TYPE = [Time → State]
```

We are using cartesian coordinates, and assuming an infinite plane. This is a simplification, actual coordinates are polar. A polar coordinates based model is also being developed and the results of the proofs are basically the same (even if the proofs are slightly more complex). The present approach helps simplify the models and proofs, and does not interfere with the properties that will be explored.

A possible interpretation, at the logical level, of the design goal is:

```
logical_at(s: System, p: wposition, t: Time): boolean =
    location(s(t)) = p
```

Interface model The user interface presents to the user a *window* around the user's current position. The location of the device is always represented at the center of the screen and features' position represented in relation to that point. The display incorporates a compass (north attribute) and features' coordinates are adjusted according to how the device is positioned. Obviously some scaling factor will be needed, and only those features that fall inside the window determined by the screen size and that scaling factor are represented[1].

[1] The "lift" type is used to model partial functions. "lift[X]" is the type "X∪{bottom}". "up: X → lift[X]", and "bottom: lift[X]" are constructors. "up?: lift[X] → boolean" is a recognizer (has the value been constructed with "up"?), and "down: lift[X] → X" is the accessor for "up".

```
screenpos: TYPE = [# ypos: screenlat, xpos: screenlong #]
Display: TYPE = [# features: [name → lift[screenpos]],
                   north: Angle #]
Interface: TYPE = [Time → Display]

ρ(s: System): Interface = λ (t: Time):
   (# features :=
      λ (n: name):
       LET p = features(s(t))(n),
           y = convertypos(p, location(s(t)), heading(s(t))),
           x = convertxpos(p, location(s(t)), heading(s(t)))
       IN IF up?(y) ∧ up?(x) THEN % a valid screen position?
            up((# ypos := down(y),
                  xpos := down(x) #))
          ELSE bottom
          ENDIF,
      north := heading(s(t))#)
```

User's mental model Based on the interface the user will form an image of the world around. We will assume the mental image created of the relative position of a feature is based both on the perceived distance of the feature and on the perceived angle of the position of the feature in relation to the center of the screen and the direction the user is heading.

```
uposition: TYPE = [# distance: nonneg_int, angle: Angle #]
Image: TYPE = [# features: [name → lift[uposition]],
                 angle: Angle #]
UserM: TYPE = [Time → Image]

perception(i: Interface): UserM = λ (t: Time):
  (# features := λ (n: name):
                 IF up?(features(i(t))(n)) THEN
                    LET origin = (# xpos := floor(screenw/2),
                                    ypos := floor(screenh/2) #),
                        d = distance(down(features(i(t))(n)), origin),
                        a = angletoheading(down(features(i(t))(n)))
                    IN up((# distance := d, angle := a #))
                 ELSE bottom
                 ENDIF,
      north := north(i(t))#)
```

Analysis - positioning subtask Based on the mental model above, a user knows he is at some feature when the distance to that feature is zero. The interpretation of the first design goal at the user level is:

```
user_at(u: UserM, n: name, t: Time): boolean =
    IF up?(features(u(t))(n)) THEN
```

```
        distance(down(features(u(t))(n))) = 0
      ELSE  FALSE
      ENDIF
```

Following from the model introduced in Section 3, testing whether the user mental model is an adequate representation for the identified subtask can be done by attempting to prove the following theorem:

```
∀ (s: System, t: Time,
                    n: {n₁: name | up?(features(ρ(s)(t))(n₁))}):
  user_at(perception(ρ(s)), n, t)=logical_at(s, features(s(t))(n), t)
```

The initial attempt at proof fails because ρ decreases the *definition* of the coordinates. Hence it might happen that the screen shows the user at some feature, when in fact the user is not *exactly* there. The impact of this inaccuracy depends on the scale factor used when presenting information on the screen. Ideally, the user will be able to identify the feature by looking around – that is, if the resolution leaves the user *close enough* to the feature. How can we analyse this notion of *close enough* at modelling time? Users will be looking, not only at the user interface, but also at the world around them. The interactive device is just another information resource for the user. This means that the most relevant question is not whether the user mental model is *in sync* with the system model (although that is relevant), but whether the user's mental model obtained from the device is *consistent* with the one obtained directly from the world. We need to include some representation of the world in the analysis.

Environment model First we must consider the relation between the world and the device. There are two aspects here that deserve consideration:

– the information that is received is inaccurate, and the degree of accuracy can vary.
– the information is not sensed continuously, the device will periodically sense information from the environment (ie. satellites) using a defined sample rate.

We develop a simple environment model that accounts for the current (real) position of the user/device only:

```
World: TYPE = [Time → wposition]

at(w: World, p: wposition, t: Time): boolean = w(t) = p
```

Updated system model We now update our system model to make its derivation from the environment model explicit (see sense below). For brevity we present here only what needs to be changed:

```
State: TYPE = [# features: [name → wposition],
                 location1: wposition,
                 location2: wposition,
                 location3: wposition,
                 heading: Angle #]
features_map: Map
```

```
induce_error(p: wposition): wposition
induce_error_def: LEMMA
  ∀ (p: wposition): distance(p, induce_error(p)) ≤ error

sense(w: World): System =
  λ (t: Time):(# features := features_map,
                 location1 := induce_error(w(t − lag)),
                 location2 := induce_error(w(t − lag − samplerate)),
                 location3 := induce_error(w(t − lag − 2 × samplerate)),
                 heading := heading(t) #)
```

Updated interface model The interface model presents the current location as a mean value of the system's last measured locations. The ρ mapping becomes:

```
ρ(s: System): Interface = λ (t: Time):
   (# features := λ (n: name): LET p = features(s(t))(n),
                                   location = mean_location(s(t)),
                                   y = convertypos(p, location),
                                   x = convertxpos(p, location)
                               IN IF up?(y) ∧ up?(x) THEN
                                    up((# ypos := down(y),
                                    xpos := down(x) #))
                                  ELSE bottom
                                  ENDIF,
          north:= -heading(s(t)) #)
```

Updated user model Finally, the user model must be updated to consider also the image obtained from the world:

```
inviewingdistance(p: wposition, n: name): boolean =
      distance(p, features_map(n)) ≤ viewingdistance
observe(w: World): UserM = λ (t: Time):
    (# features := λ (n: name):
                      IF inviewingdistance(w(t), n) THEN
                         LET d = distance(features_map(n), w(t)),
                             a = angletonorth(features_map(n), w(t))
                         IN up((# distance := floor(d), angle := a #))
                      ELSE bottom
                      ENDIF,
         north := bottom #)
```

We are considering that by direct observation of the world the user will not be able to know where the North is.

Analysis - integrating device and environment information We now have to test whether both images are consistent. The theorem for this is:

```
∀ (w: World, t: Time,
          n: {n_1: name | inviewingdistance(w(t), n_1)}):
  user_at(perception(ρ(sense(w))), n, t) = user_at(observe(w), n, t)
```

During the proof it becomes necessary to prove that the user will build the same perception of being at some specific point, regardless of that perception being obtained from the interface or directly from the environment. There are a number of factors that prevent us from proving this equivalence:

– the scale factor used to convert to screen coordinates;
– the error margin introduced by the sensing process;
– the time lag in screen refresh introduced by the sample rate.

Starting with the simplest case we can consider a user who is stationary at some location. In this case the effect of the sample rate is eliminated since all readings are being taken from the same location. The error margin and the scaling factor, however, can interact to create problems. Two situations can be considered:

– the error margin is smaller than the scaling factor — in this case the screen will indicate a zero distance when the user is at a feature. However, if the scale factor is too large the screen might indicate the user is at a feature even if the user is not.
– the error margin is greater than the scaling factor — in this case the screen might indicate that the user is not at the feature even when he/she is. In this case it can also be shown that the user interface is not stable over time. Even when the user does not move, the position indicated by the screen will change as the error affects the sensing procedure. The larger the difference between error margin and scaling factor, the greater this effect will be.

The problem is that there is uncertainty introduced by the device (both at the logical level and at the presentation level) that is not presented to the user - an approximation is presented as a precise position. Using more samples for the approximation will only reduce uncertainty if the user is stationary, and will decrease the responsiveness of the device. The alternative is to make the uncertainty apparent on screen. This can be done by presenting not *the* position, but a set of possible positions to the user. One way of doing this is to present a circle indicating the uncertainty margin. Anther possibility would be to present all of the different readings made by the device. In this latter case it can be expected that the cognitive load on the user will be greater than in the former case. This is because the user would have to infer, from the points, the likely area of uncertainty. If we choose the former possibility the interface model becomes:

```
Display: TYPE = [# features: [name → lift[screenpos]],
                   north: Angle,
                   uncertainty: nonneg_int #]
```

With this model the definition of being somewhere becomes:

```
user_at(u: UserM, n: name, t: Time): boolean =
    IF up?(features(u(t))(n))
      THEN distance(down(features(u(t))(n))) < uncertainty(u(t))
    ELSE FALSE
    ENDIF
```

That is, the user is considered to be at some location if the location falls inside the area of uncertainty. Initially a reasonable value for the uncertainty can be defined based on the known error margin and scaling factor(screen size).

So far we have considered a stationary user. It is also relevant to consider what happens when the user is moving. If we suppose the user moves in order to reach some location, then it is relevant that he or she knows when he/she has arrived at the destination (consider a driver using the GPS device and wanting to leave the road at some specific junction). We can write a function "has_arrived(m: UserM, ($t1$, $t2$): Time, n: Name): boolean" which tests if between two instants in time the user has arrived at a location. A useful property of the system would be to guarantee that when the user arrives at the location, the device also indicates arrival. This can be expressed as:

$$\begin{array}{l} \forall\ (w\text{: World, } (t1, t2)\text{: Time,} \\ \qquad\qquad n\text{: } \{n_1\text{: name } \mid \text{ inviewingdistance}(w(t),\ n_1)\})\text{:} \\ \quad \text{has\_arrived}(\text{perception}(\rho(\text{sense}(w))),\ t1,\ t2,\ n) \\ \quad = \\ \quad \text{has\_arrived}(\text{observe}(w),\ t1,\ t2,\ n) \end{array}$$

We find however that this theorem cannot be proved. If the user is moving fast enough he/she might reach the destination before the device is able to indicate that on screen (in the example of the driver mentioned above, this would amount to going by the junction before the GPS device had time to warn him about it). This is because, due to the sampling rate, when the velocity of the user increases the readings of position become further apart. If the user is moving the uncertainty increases. Since the uncertainty presented at the screen has a fixed value, at some point the "real" uncertainty will exceed that presented and the user might already be at the destination while the screen still shows him some way apart.

Several solutions to this could be considered. Interpolation of the position based on current velocity is one possibility. The use of a variable uncertainty area is another. In the first case we try to diminish the uncertainty by making predictions about future readings. This solution might present problems if there are quick variations on the heading and speed. In the second case an attempt is made to better represent uncertainty on screen. This does not solve the problem, it simply highlights that the user's position cannot be determined exactly. To completely solve the problem it is necessary to make changes at the device level, in order to obtain more exact readings of position, which is beyond the scope of the interface design.

We have presented an analysis which relates speed of movement, scale, display size, accuracy of device and frequency of updates, size and distance of features. The value of the analysis is in the complex issues which are raised, in a framework which allows clear and careful consideration.

This discussion gives an idea of the sort of tradeoffs and problems that must be considered and addressed when designing system with a dynamic component. It also shows how the model proposed in Section 3 can be used to reason about representational aspects in the presence of time considerations. Additional design alternatives could of course be considered and analysed in a similar manner. While it is always tempting to simply add more information (percepts), the limited "real estate" in many ubiquitous devices motivates careful consideration of the representational aspects of a display with respect to the user's tasks and activities.

5 Conclusions

The manner in which information is presented in an interactive system has a profound affect on our ability to perceive and reason about that information. This issue is even more vital in modern computing systems, where the technology and form factor can constrain the interaction. Furthermore, the variety of environments and situations of use (which may not always be amenable to a given form of interaction) add further challenges, many of which are time-related. Given that dynamically sensed information is a central part of many ubiquitous computing applications, dynamic information displays will be increasingly common. The constraints imposed by the physical form factors motivate careful design of these displays. We believe that work on the significance of information representations such as that of Hutchins [10] to be particularly relevant to these emerging technologies. However, such work does not provide a concrete and methodical basis for analysis (although it does provide a potential theoretical basis for analysis).

In previous work we have examined the issue of representational reasoning [6, 5]. This work, however, did not explicitly take time into consideration. To address this issue, we have presented a model which allows both time and environmental factors to be considered when reasoning about the usability of a representation. The example has shown how considerations about the users' goals, and scenarios of usage, can help in analysing alternative design options. By using rigorous analysis, it is possible to uncover assumptions concerning interaction and perception, which are implicitly made during the design of the interface. It can be used whenever deemed useful to validate specific aspects of the rationale behind a design decision, this can be particularly valuable where there is a complex relationship between real-world information available to the user, and sensed information displayed dynamically on the device.

One important aspect of the use of rigorous approaches is the possibility of providing automated support for the reasoning process. In this case we have used a first order theorem proving tool (PVS) to explore the proposed properties. While it is fair to argue that theorem proving tools can be hard to apply, we have found PVS to be helpful in validating the models for consistency, and quickly highlighting design problems when attempting to prove the theorems.

References

1. J. C. Campos. *Automated Deduction and Usability Reasoning*. DPhil thesis, Department of Computer Science, University of York, 1999. Also available as Technical Report YCST 2000/9, Department of Computer Science, University of York.
2. José C. Campos and Michael D. Harrison. Model checking interactor specifi cations. *Automated Software Engineering*, 8(3/4):275–310, August 2001.
3. A. Dix and G. Abowd. Delays and temporal incoherence due to mediated status-status mappings. *SIGCHI Bulletin*, 28(2):47–49, 1996.

4. Alan Dix. The myth of the infi nitely fast machine. In D. Diaper and R. Winder, editors, *People and Computers III — Proceedings of HCI'87*, pages 215–228. Cambridge University Press, 1987.
5. G. Doherty, J. C. Campos, and M. D. Harrison. Representational reasoning and verifi cation. *Formal Aspects of Computing*, 12:260–277, 2000.
6. Gavin Doherty and Michael D. Harrison. A representational approach to the specifi cation of presentations. In M. D. Harrison and J. C. Torres, editors, *Design, Specification and Verification of Interactive Systems '97*, Springer Computer Science, pages 273–290. Springer-Verlag/Wien, June 1997.
7. Gavin Doherty, Mieke Massink, and Giorgio Faconti. Using hybrid automata to support human factors analysis in a critical system. *Formal Methods in System Design*, 19(2), September 2001.
8. Wayne D. Gray and Deborah A. Boehm-Davis. Milliseconds matter: An introduction to microstrategies and to their use in describing and predicting interactive behaviour. *Journal of Experimental Psychology: Applied*, 6(4):322–335, 2000.
9. M. D. Harrison and D.J. Duke. A review of formalisms for describing interactive behaviour. In R. Taylor and J. Coutaz, editors, *Software Engineering and Human Computer Interaction*, number 896 in Lecture Notes in Computer Science, pages 49–75. Springer-Verlag, 1995.
10. E. Hutchins. How a cockpit remembers its speeds. *Cognitive Science*, 19:265–288, 1995.
11. C. Johnson. The impact of time and place on the operation of mobile computing devices. In B. O'Conaill H. Theimbleby and P. Thomas, editors, *Proceedings of HCI 97*. Springer-Verlag, 1997.
12. C. Johnson, editor. *Proceedings of 8th International Workshop on Interactive Systems, Design Specification and Verification*, volume 2220 of *Lecture Notes in Computer Science*. Springer, 2001.
13. Chris Johnson and Phil Gray. Temporal aspects of usability (workshop report). *SIGCHI Bulletin*, 28(2), 1996.
14. P. Markopoulos and P. Johnson, editors. *Design, Specification and Verification of Interactive Systems '98*, Springer Computer Science. Eurographics, Springer-Verlag/Wien, 1998.
15. William M. Newman and Michael G. Lamming. *Interactive System Design*. Addison-Wesley, 1995.
16. Donald E. Norman. *The Psychology of Everyday Things*. Basic Book Inc., 1988.
17. S. Owre, J. M. Rushby, and N. Shankar. PVS: A prototype verifi cation system. In D. Kapur, editor, *Automated Deduction — CADE-11*, number 607 in Lecture Notes in Artifi cial Intelligence (subseries of Lecture Notes in Computer Science), pages 748–752. Springer-Verlag, 1992.
18. J. Preece et al. *Human-Computer Interaction*. Addison-Wesley, 1994.
19. Jef Raskin. *The Humane Interface*. ACM press, 2000.
20. Chris Roast. Designing for delay in interactive information retrieval. *Interacting with Computers*, 10:87–104, 1998.
21. John Rushby. Using model checking to help discover mode confusions and other automation surprises. In *(Pre-) Proceedings of the Workshop on Human Error, Safety, and System Development (HESSD) 1999*, Liège, Belgium, June 1999.
22. Bernard Sufrin and Jifeng He. Specifi cation, analysis and refi nment of interactive processes. In M. Harrison and H. Thimbleby, editors, *Formal Methods in Human-Computer Interaction*, Cambridge Series on Human-Computer Interaction, chapter 6, pages 154–200. Cambridge University Press, 1990.
23. B. Ullmer and H. Ishii. Emerging frameworks for tangible user interfaces. *IBM Systems Journal*, 39(3&4), 2000.