Formal Analysis of Ubiquitous Computing Environments through the APEX Framework

José Luís Silva[1], José Creissac Campos[1], and Michael D. Harrison[2]
[1]Departamento de Informática/Universidade do Minho & HASLab/INESC TEC
{jlsilva, jose.campos}@di.uminho.pt
[2]School of Electrical Engineering and Computer Science, Queen Mary University of London
michael.harrison@eecs.qmul.ac.uk

ABSTRACT
Ubiquitous computing (ubicomp) systems involve complex interactions between multiple devices and users. This complexity makes it difficult to establish whether: (1) observations made about use are truly representative of all possible interactions; (2) desirable characteristics of the system are true in all possible scenarios. To address these issues, techniques are needed that support an exhaustive analysis of a system's design. This paper demonstrates one such exhaustive analysis technique that supports the early evaluation of alternative designs for ubiquitous computing environments. The technique combines models of behavior within the environment with a virtual world that allows its simulation. The models support checking of properties based on patterns. These patterns help the analyst to generate and verify relevant properties. Where these properties fail then scenarios suggested by the failure provide an important aid to redesign. The proposed technique uses APEX, a framework for rapid prototyping of ubiquitous environments based on Petri nets. The approach is illustrated through a smart library example. Its benefits and limitations are discussed.

Author Keywords
Ubiquitous and Context-Aware Computing; Analysis; Modeling; Prototyping; 3D virtual environments.

ACM Classification Keywords
F.3.1 [Logics and meanings of programs]: Specifying and Verifying and Reasoning about Programs -Mechanical verification; H.5.2 [Information Interfaces and Presentation]: User Interfaces - Prototyping; D.2.m [Software Engineering]: Miscellaneous – Rapid prototyping.

INTRODUCTION
The design and engineering of ubiquitous computing environments (i.e., electronically enriched environments that are able to sense and respond to the presence of people) present new challenges. Designing a ubiquitous computing environment entails integrating a number of embedded devices and sensors into a meaningful whole, capable of adequately responding to multiple users and their own devices. Given the potential complexity of the interaction between all these elements, it is difficult to analyze a design thoroughly early in its development. This is further complicated by the critical role that the physical environment plays in these systems. It is not always feasible to deploy early versions of the system within a target environment because of restrictions of cost or availability.

Given this situation prototypes have a particular relevance. Indeed, using prototypes to understand an envisaged design has become a principal research approach in Ubiquitous computing [5]. We are particularly interested in the role of prototypes in evaluating the user experience of a target environment, and have been developing the APEX framework as a solution to this problem.

Previous papers have discussed use of the APEX tool as a model driven approach to the development of prototypes based on virtual environments [16, 17]. One important step in the development of a rapid prototype in APEX is to create a virtual environment that is close enough to the physical target system to provide an adequate and realistic experience for users. This environment is created for the user or users by means of a viewer in Opensimulator[1]. The simulation of the ubiquitous system can be achieved within virtual environment by using a colored Petri net (CPN) model to describe its behavior. By this means it is possible not only to interact with objects within the virtual environment but also with real users (via the viewers), simulated autonomous users that are also modeled in CPN, virtual interaction devices such as PDAs and sensors, and real interaction devices. These environments become prototypes of the envisaged systems, which can be used for evaluation.

As a result of the complex interactions arising from the combination of multiple sensors, devices and users in a physical space, observation of episodic use of the prototype alone is not sufficient to guarantee that some particular feature of the system is a property of the design. It becomes difficult to establish whether observations made about use

EICS'12, June 25–28, 2012, Copenhagen, Denmark.

[1] http://opensimulator.org (last accessed January 20, 2012)

are truly representative of all possible interactions or whether certain characteristics of the system are true in all possible scenarios.

The fact that the behavior is driven by a CPN model makes it possible to analyze the behavior of the prototype systematically and exhaustively using CPN Tools [7]. It is this analysis of interactive systems that forms the discussion of the paper. The application of special purpose heuristics to the design of the ubiquitous system is the basis of the discussion. The next section discusses previous research. The paper then moves from a description of the approach to an example of its application.

The paper makes two of contributions.

- It introduces a method of evaluating ubicomp environments through exhaustive analysis, applying and adapting heuristics chosen from other areas of software engineering and HCI. This evaluation is complemented with an analysis of a simulation in 3D.
- It identifies property patterns in the identification and verification of properties.

The stages of analysis using these property patterns are demonstrated through an example.

BACKGROUND

A number of techniques within HCI support the analysis of the usability of an interactive system from early in its design. These techniques range from paper prototyping and Wizard of Oz, to the development of versions of the systems that can be used during user testing. Other techniques that do not require explicit user testing include the use of expert evaluation techniques such as Heuristic Evaluation and Cognitive Walkthrough.

Ubicomp environments present challenging usability evaluation problems. Because they are embedded within physical environments interactions with them differ from the styles of more traditional systems [8]. Interaction within the environment may be *explicit* and the devices used for interaction with the system subject to standard usability heuristics for small devices, or it may be *implicit* and arise simply as a result of the user changing their context (for example moving in or out of a room). In both cases each user's context plays an important role.

A number of evaluation techniques have been developed for dealing with implicit interactions within ubicomp environments. Kim et al. [8], for example, have presented several ubicomp case studies where evaluation has involved making use of physical space. Other evaluation approaches have aimed to provide early evaluation of a partially functional system by using Wizard-of-Oz techniques. Even these more limited approaches involve large resource investments: in the one case building real space for the ubicomp system, and in the other developing the system to a partially working level. These costs could be reduced by the application of heuristics to a ubicomp application as explored by Mankoff et al. [10] in the context of ambient displays.

Scholtz et al. [18,13] have developed a framework for evaluating ubiquitous computing applications. They developed a set of sample metrics measures based on ubiquitous computing evaluation to assess whether adequate design principles are satisfied and if the design produces the desired user experience. This framework does not provide an exhaustive means of analyzing a developed prototype. Instead the focus is to identify key areas of evaluation and to identify metrics and design guidelines to improve user experience in ubiquitous systems.

Ubiquitous systems prototyping research is mostly concerned with the development of prototypes of isolated devices (e.g. Topiary [9]). Some approaches like 3DSim [15] and VARU [6] develop simulations of actual environments like APEX. The benefit of APEX is that modeling and associated analytical approaches can be combined with simulations. Additionally APEX supports a multilayered development approach: simulation layer (Opensimulator); a modeling layer (using CPN Tools) and a physical layer (using external devices and real users).

Scholtz et al. [14] argue the need to develop interdisciplinary evaluation techniques to address ubicomp properties at early stages in design. Assessment techniques are required to evaluate alternative solutions before deploying the system. The complexity of a physical environment where a number of devices are situated, and the added complexity of real world activities, means that it is hard to assess which observations are representative of the use of the system. Likewise it is difficult to assess informally whether characteristics of the system, assessed against specific heuristics, hold across all possible usage scenarios.

The experience of exploring ubicomp environments depends on individual preferences. However some characteristics of user experience can be expressed as properties of the environment. These properties can complement an understanding of experience based on empirical evaluation of the use of a prototype and should be seen as part of a toolset for evaluating a design. We argue that systematic and exhaustive techniques need to be part of an interdisciplinary approach. We follow Mankoff et al. [10] by developing property patterns from existing heuristics. Property patterns have two roles: i) helping identify interesting properties and ii) helping verify existing properties. For example a property of the *system* requires that there should be feedback for any user of the environment who carries out a particular kind of transaction. This can be expressed as a typical property that takes a standard form. This property pattern would provide the form and would complement evaluation techniques by offering exhaustive analysis of whether a property is true. This would not be feasible by exploring all possible user behaviors through observation.

APPROACH

Usability heuristics [11] are a starting point for analysis using the APEX system. In this approach the analyst is encouraged to explore how well a particular design supports general properties that encourage ease of use. The analyst or team of analysts bring their expertise in human factors or their understanding of the domain to decide where there are issues (for example in relation to ease of recovery or the visibility of the effect of explicit or implicit actions) in the design and propose design improvements.

Tool Support

To achieve a systematic and exhaustive analysis of the CPN model of the behavior of the system, the verification capabilities of CPN Tools are used. These tools provide a modeling and verification environment for Colored Petri Nets. Particularly relevant here is the State Space (SS) tool. The tool generates a *reachability* graph that defines the states that can be reached from some starting state. Each node of the graph represents an execution state. Arcs represent the binding of particular values (e.g. actions) from one state to a new one. Figure 1 illustrates part of one of these graphs. The whole graph represents all possible executions of a ubicomp system showing which actions can be executed in each system state. Each node is numbered and labeled with its number of input/output arcs. Arc and node labels are hidden by default in the tool, but can be checked interactively (e.g. arc caption in Figure 1).

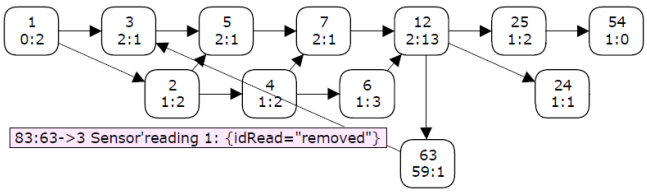

Figure 1- Reachability graph

The process of verification of a property involves applying a predicate to relevant states in the reachability graph. The returned result is either that the predicate is true of all relevant states or that the predicate fails to be true, in which case the path to the failing state is indicated. This path can then be used to explore a situation that may be of interest from the perspective of the design of the ubiquitous system.

Patterns

The approach uses verification patterns adapted from properties that are based on usability heuristics as well as a broader range of properties used in other fields [4]. Of particular interest is a set of property patterns provided by the IVY tool [2]. This tool is a model-based environment for the analysis of interactive systems that is used here as a starting point. In IVY patterns define *property templates*, expressed in temporal logic, which must be instantiated to the particular details of the system and property under consideration. This instantiation process creates a temporal formula that can be verified. Analysis based on the formula is then performed automatically by a model checker.

Applying the patterns in the context of APEX raises a number of challenges. A first challenge is how the property template defined within the pattern relates to the verification process. As explained above verification is achieved using the SS tool by writing predicates over the reachability graph. Hence, the pattern, instead of defining a temporal logic template, must define how the reachability graph is to be explored (in particular defining which predicates are needed) so that verification can be performed. Other challenges concern the interpretation of the patterns, and in particular:

- Who are the users? IVY patterns assume interaction between a user and the device. In APEX the interaction context is richer, involving spaces where several *users* might be present. Hence, when considering user actions and system responses it is necessary to consider how different users affect each other, e.g., an action by one user might trigger a system response directed to a different user. It becomes relevant to consider therefore who carried out an action or caused some change in the system state.
- What are the actions? In a ubicomp setting implicit interaction becomes relevant as well as explicit user action. The system might be responding to conditions arising through implicit user action or changes to the environment. These conditions are typically monitored indirectly through sensors, e.g., a user entering or leaving a room. Hence, rather than actions, *situations* of interest may require characterization.
- What is being analyzed? A general problem not specific to this context is whether the property is addressing the design of the system or the model itself, i.e., whether the property is being used to reason about features of the system's design, or is being used to validate the model itself. This affects the interpretation of the reachability graph. Indeed, while some nodes correspond to states of the ubiquitous system, others correspond to intermediate execution states of the model.

Setting Up the Analysis

The approach is illustrated in the next sections using a smart library context. The example illustrates the choice of property patterns and how these patterns are instantiated in the case of the example and then checked within the APEX framework.

As a brief indication of the process consider the following specific property that concerns the illumination of a book light. The light turns off depending on the user action (user taking the book or moving away) but also depends on the actions of other users (taking the book) and also of the state of the system (light already turned off). This property is an instance of a particular pattern, namely the feedback pattern. It requires that in all paths through the environment, and for all states in the paths, it is true that if the light is on for the book that the user wants and the user takes this book then in every next state the book light is turned off. This

property relates to a specific user who takes the book, the one who has reserved it. The system would leave the light on in the book if the wrong person takes it, hence indicating that they are taking the book without reserving it.

For the property to be verified, the model must be converted to a form that will allow CPN Tools to check the truth of the property. CPN Tools require that the model be deterministic and "small" so as to reduce the search space used during analysis. The SS tool (part of CPN Tools) uses brute force to bind each variable to each of its possible known values, creating the reachability graph. Because in normal conditions the model exchanges information with the virtual world simulation and/or actual physical external devices, the model can in principle be of unlimited size. This large open model must therefore be translated into a closed one, so that it is tractable within the SS tool.

Closing the model means isolating it from external components. This is achieved by defining finite sets of possible values for all the variables in the model that previously held values acquired externally. CPN Tools defines a set of up to one hundred elements as *a small color set*. APEXi, a component of APEX, is used to initialize *small color sets* semi-automatically. The tool provides an interface (see Figure 5) to enable analysts to supply or select desired values that can be used to populate as tokens the relevant places of the CPN model as represented by a chosen scenario.

Property patterns are explored in APEX in the next section. They are represented formally and then instantiated for the example ubicomp environment.

PROPERTY SPECIFICATION PATTERNS FOR UBICOMP ENVIRONMENTS

Patterns provide a basis for analysis serving two roles: they aid the process of elicitation of appropriate properties; they help the analyst use CPN Tools to perform the analysis of the instantiated property. A number of relevant property patterns are now described. These patterns are adapted from those supported by the IVY tool [2] to the ubicomp context.

In [3] patterns are expressed as CTL templates to be instantiated to concrete actions and predicates. We express the patterns as predicates over the CPN model's reachability graph where actions are represented as transitions and effects as predicates over states. States are defined by the values of the attributes in the model and capture relevant configurations/conditions of the system.

The Consistency Pattern

Justification: Consistency is a heuristic that has widespread relevance, including in the Ubicomp area [10].

Intuition: The consistency pattern defined in [3] captures the requirement that a given event Q always causes a defined effect R (expressed as a predicate over the states before and after Q). There is optionally an additional predicate (a guard) that constrains when the system behaves consistently.

In the ubicomp context: the event (Q) is either an explicit or an implicit action by the user which might change the environment or the state of the system. Implicit actions and environment changes are expressed in terms of the values read by the sensors in the system. The effect of the action (R) is a change in the state of the system as a whole. Whether users perceive the change in environment is an important element of the effect. Context plays an important role. If an action by some user is being analyzed then the presence of other users might also influence the response. Hence, the gate in the library may not close when a user leaves its neighborhood because of the presence of another user. These various dimensions add to the texture in which the pattern can be used beyond providing values for Q and R. The context of the analyzed environment is described by the tokens defined by *small color sets* initialized by the APEXi tool.

The algorithm: The algorithm to be followed is presented in Figure 2. For simplicity sake this algorithm assumes that the given effect happens in response to the particular event(s)/state(s) being considered only (this can be checked with the Precedence pattern below). The functions in the figure are used to identify, in the reachability graph, counter examples for the property being verified. The *counterExampleNodes* function identifies the nodes of the counter example by firstly identifying relevant nodes (corresponding to the effect R – *identifyRelevantNodes* function). Nodes correspond to states of the reachability graph. From the identified relevant nodes (returned by the *identifyRelevantNodes* function) the algorithm attempts to identify alternative paths were the desired effect is not verified. The *counterExampleNodes* function is applied to the set of relevant values of the selected scenario (using *map*) and the resulting list of nodes is held in the *CONSISTENCY* variable. If the list is empty, the property holds. The underlined pieces in Figure 2 are the parts that need to be instantiated. They identify the places in the Petri net that are relevant for the property being verified. A concrete instantiation of this algorithm is presented in Figure 6.

The Feedback Pattern

Feedback is a particular use of the consistency pattern where a user action Q always causes a perceivable effect R. In the ubicomp context the action (R) represents a change that is observable in the environment though it should be noted that the person causing the system's response might not necessarily be the same as the person who observes the response. Even if it is the same person, the fact that the response might be triggered by an implicit interaction or an environment change begs the question of whether the response will be salient enough. At this stage issues such as salience are not being considered, rather the concern is to guarantee that feedback is always provided. It is likely that evaluating the salience of a particular feedback will require input from the simulation (an example of synergy between

the formal and empirical analysis - but see [12] for a formal treatment of salience).

```
fun identifyRelevantNodes obj =
  PredAllNodes (fn n => cf(obj,Mark.MODULE'PLACE 1 n) > 0)
-------------------------------------------------------------------------------
fun counterExampleNodes u =
 let
  val nodes = identifyRelevantNodes u
 in
  let
   val predecessorsNodes =
   SearchNodes (nodes, fn _ => true, NoLimit, fn n => InNodes n, [], op ^^)
  in
    let
      val nodesPredecessorsNodesWith2orMoreSucessors =
      SearchNodes (remdupl(predecessorsNodes), fn n => not(contains nodes  [n])
                    andalso length( OutNodes n) >= 2, NoLimit, fn n => n, [], op ::)
    in
      SearchNodes (remdupl(nodesPredecessorsNodesWith2orMoreSucessors),
                    fn n => not( contains (map (fn x=> Reachable(n,x) andalso
                    length(NodesInPath (n,x))>2) nodes) [true]), NoLimit,
                    fn n => n, [], op ::)
    end
  end
end
-------------------------------------------------------------------------------
val CONSISTENCY =
  map(fn u => counterExampleNodes u) (UpperMultiSet (Mark.MODULE'PLACE 1)
```

Figure 2 - The consistency/feedback pattern algorithm

The Reachability Pattern

Justification: reachability is a basic property over which other properties are derived (e.g. precedence, completeness). It can be used to demonstrate that the system can reach a specific state or situation.

Intuition: The reachability pattern captures the requirement that the system can always evolve from one specific state S to another state Q.

In the ubicomp context: environmental situations are represented as states based on particular distinguishing features, for example, a book being illuminated, or a user being at a given location. Some features of the state are likely to be directly controlled by the system (the light on the book), while others are observed (the user's position), although the observed features might be indirectly influenced by the system (e.g., the gate, when it opens, enables the user to move inside the library). Depending on the complexity of the system, establishing how these influences work will not be easy and can be aided by formal verification.

The algorithm: uses the reachability graph and identifies desired states with identifying attributes. For each identified state S the algorithm checks whether it is possible to reach a new state Q with the desired environment attributes. An instance of the algorithm can be found in Figure 7.

The Precedence Pattern

Justification: The precedence pattern describes relationships between a pair of events/states where the occurrence of the first is a necessary pre-condition for the occurrence of the second.

Intuition: This pattern captures the requirement that a state or event S precedes another state or event P. The occurrence of the second is enabled by the occurrence of the first.

In the ubicomp context: This property can be used to verify that some event or state does not occur without the satisfaction of a pre-condition. Consider for example the property concerned with illuminating the book. The first state (S), triggered by a user action (for example, as the user approaches the book), is a pre-condition for the occurrence of the second state (P – book light turned on). The property requires that the light will never turn on without a relevant user approaching the book. Note that this does not guarantee that the light will always turn on when a relevant user as would be required by a consistency property.

The algorithm: identifies the second states (P) of the reachability graph based on attributes of the environment and then identifies each predecessor. The presence of state S characterized by specified attributes is verified.

Other Patterns

Several other patterns were also adapted, for example: Reversibility (the effect of a given action can eventually be undone); Possibility (some event or state is always possible throughout the execution of the system); Universality (some condition always holds); or Eventuality (some event or condition must eventually hold at some point).

DEVELOPING A MODEL FOR ANALYSIS OF A SMART LIBRARY

The patterns are now applied to an example. The analysis process requires an initial setup before property patterns can be instantiated.

Introduction to the Example

A "smart library" identifies books stored on bookshelves using Radio Frequency Identification (RFID) tags. Screens provide context sensitive information to library users. A registered library user is allowed entry or exit via gates. When a registered user arrives at the entry gate, a screen displays which books have already been requested by them (using a web interface at their desktop for example) and opens the entry gate.

The system recognizes the users' position in real-time by means of presence sensors. The users are guided to required books by further screens. As the user approaches the book's location a light with a distinctive color is turned on allowing several users looking for books in nearby locations to distinguish their own request. When the book is removed, the light on the book is turned off. As the user returns to the exit gate a personalized list of requested and returned books is displayed on a screen by the gate. The gate is then opened so that the user can leave.

While the example is not based on any specific existing system, similar systems could be used to support dispatch in relation to e-shopping or for guiding people inside a building (e.g. hospital or airport). Indeed, a method and system for localizing objects among a set of stacked objects equipped with improved RFID tags has been patented [1] suggesting the feasibility of the physical implementation of the system.

The Model

An APEX prototype of the library example is described more fully in [16]. Here the process of using it for verification is now illustrated.

Creating the prototype involves creating the virtual environment and extending the APEX CPN *base model*. The *base model* underpins the behavioral model of the ubiquitous system. Specific behavior relating to the library system is added to the *base model* and this animates the environment so that it is appropriate for evaluation based on user exploration of the virtual space as well as being a basis for verification.

A number of modules are added that simulate the behavior of gates, books, PDAs and displays. The Gate module is described using CPN in Figure 3 and holds information about the users, the devices and the sensors present in the environment. The purpose of the gates module is to open a gate when a user with appropriate "entering" permission is in the proximity of a presence sensor associated with the gate. The Gate module consists of a transition to open a gate and another one to close it. Whether the module will open or close the gate is based on information held by a number of places: *Dynamic Objects* (e.g. gates, screens), *Users* and *P_sensors* (presence sensors).

The opened or closed state of the gates is recognized through two places: the *Dynamic Object* place (holds tokens for closed gates) and the *gates opened* place (holds tokens for opened gates).

A function is used to identify the type of the objects that are being dealt with in the *Dynamic Objects* place. A particular concern is to identify the gates because the *Dynamic Object* place holds objects other than gates. The *is* function is designed to receive a dynamic object and a string as arguments and to compare the type of the object against the string to check whether there is a correspondence. In the case of the gate further information is required to decide whether to open or close the gate. This information includes whether: (i) a user is near a presence sensor; (ii) the presence sensor affects the gate; (iii)nobody is near the presence sensor.

Three functions are used to capture these conditions: *userNearPresenceSensor, objAfectedByPresenceSensor* and *nobodyNearPresenceSensor*.

A further module, responsible for providing directions to users, is presented in Figure 4. The module uses the positions of the requested book and the user to send information to the relevant PDA about which direction should be followed. The means of getting the direction and sending it to the appropriate PDA is associated with the *show direction* transition. In particular the *sendUserInfo* function is used to send information to a specified user. The identifier of the sensor used to obtain the direction is forwarded by the module to the *PDAs with the new direction info* place to be used to decide when to display default information (*show default* transition).

This combination of modules (along with others which pick up books and notify relevant users) can now be analyzed.

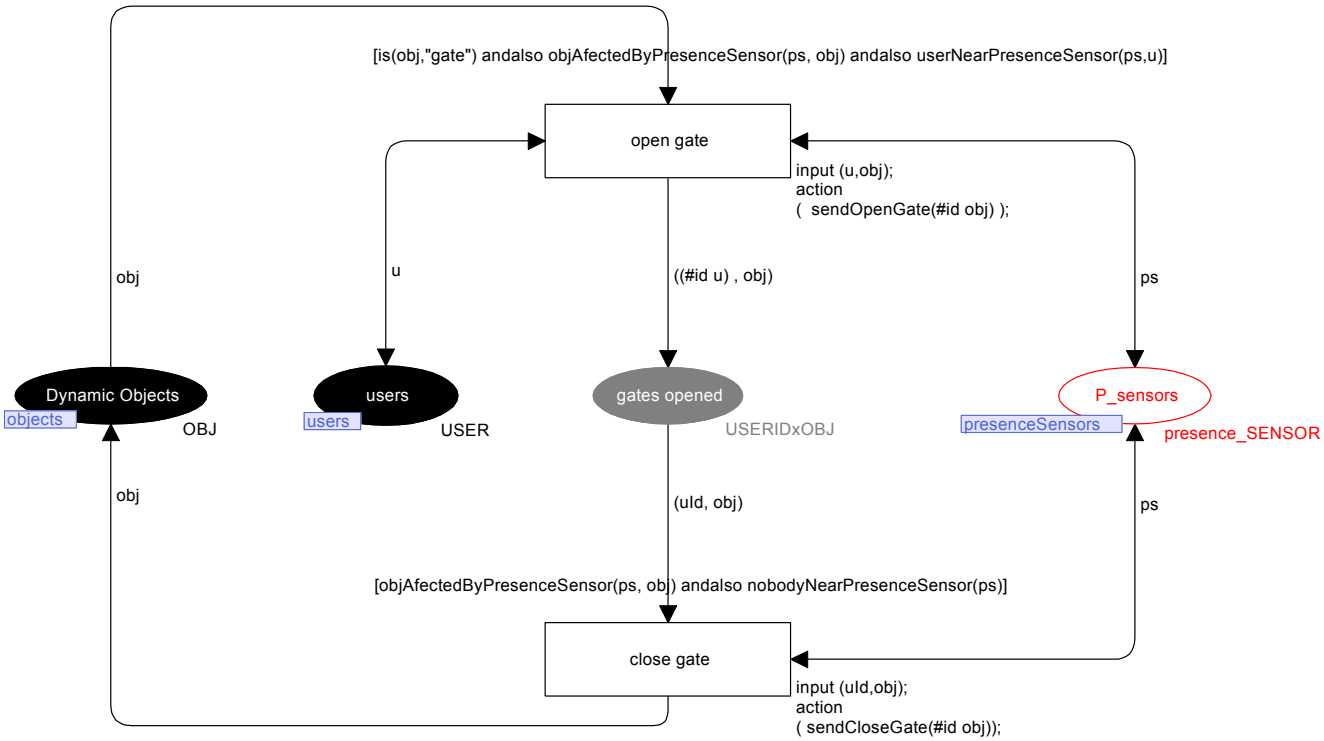

Figure 3 - Gates module

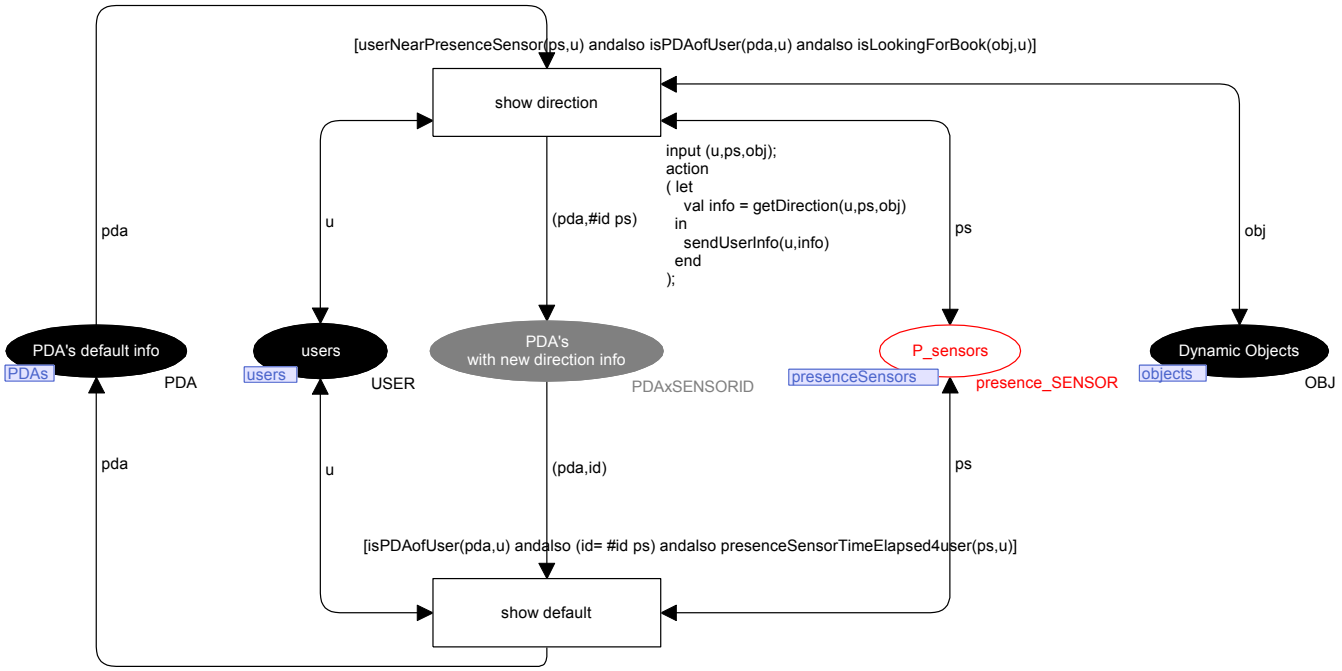

Figure 4 - User's PDA book direction module (open version)

Setting Up the Model for Analysis

The model must be transformed into a closed model containing *small color sets* in order to reduce the state space for analysis using the SS tool. This is achieved by removing the non-deterministic elements of the model, isolating it from external components using scenarios that limit the behavior of the open elements.

The APEXi tool has been designed to ease scenario creation. Values associated with instances of behavior and dynamic objects are automatically inserted into *small color sets*. Figure 5 shows values provided to APEXi as follows:

- one user (Test User - *UsersIDs* field) desiring the book with identifier 1 (*values* field);
- two presence sensors, one at the entrance and the other close to the bookshelf (*Sfeatures* field);
- two books with identifiers 1 and 2 (*ObjIDs* and *OBJfeatures* fields);
- one gate with identifier 3 (*ObjIDs* and *OBJfeatures* fields).

These values set up a scenario where the feedback property is going to be analyzed. Once external library services have been related to gates, book lights and behaviors as determined by user position and user requests for selected values. The model becomes adequate for analysis. A number of similar scenarios should be selected with the help of human factors or domain specialists to complete the analysis. For instance it does not make sense to analyze gate behaviors without users.

INSTANTIATING PROPERTY TEMPLATES

Having developed an appropriate model for analysis and selected a scenario, it is now possible to proceed. Analysts may know which property they want to prove (e.g., by observing real users as they interact with the simulation), but they can also have difficulties in their identification. The templates help them in this task. By capturing (and thus guiding the analysis towards) potentially relevant features of a design, they help the analyst discover appropriate properties.

Additionally, using property templates makes it easier to verify properties because algorithms to verify each of the property patterns can be reused.

Three property templates are considered in relation to this example: feedback, reachability and precedence. The other templates mentioned in the paper have similar application.

Feedback

Parameters for property templates are instantiated with user interactions, environment changes and features or states of the environment. An instance of the feedback pattern is whether the books always respond to relevant approaching users. The feedback pattern parameters are instantiated with the following values: action Q is defined as the implicit action occurring when the user approaches the bookshelf (the proximity of the user to the bookshelf is detected by presence sensors in the environment); effect R is defined as changing the environment so that the relevant light is switched on; guard S is defined as stating that the light must initially be off for this property to hold.

This pattern identifies a relevant feedback property: "*when a user approaches the appropriate bookshelf the book lights up (unless it is already on)*".

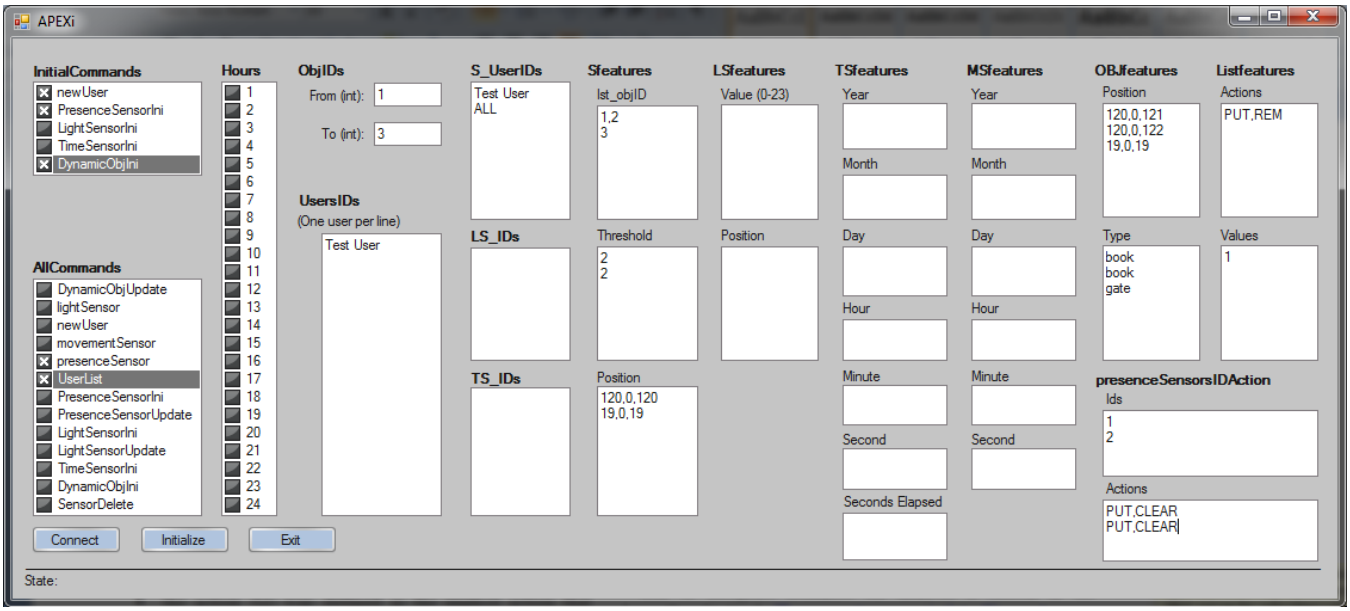

Figure 5 - APEXi tool with selected value used to analyze the feedback property pattern

Reachability

The reachability template can be instantiated similarly: state Q is the situation where a book that a user is looking for is picked up by another person (stops being available); state S is the situation when the user is notified.

This example instantiation identifies the property: "*If a book that a user is looking for is picked up by another person (stops being available), the user is notified*".

Precedence

The precedence template can be instantiated: state S corresponds to the relevant user being near the bookshelf; state P to the light being turned on.

This identifies "*the light does not turn on while the relevant user is not near the bookshelf*".

For analysis it is also necessary to know how many users this property will be applied to as well as which actions are considered (implicit or explicit), what is going to be analyzed, the environment or the mode itself. The selection of adequate scenarios for analysis is critical to the results obtained.

CHECKING THE MODEL USING THE SS TOOL

Checking the properties of the model is considered in this section. This process uses the APEX tools, specifying and instantiating the algorithms in CPN Tools as specified by the appropriate pattern.

Feedback

To verify the first property "*when a user approaches their requested book the book's light turns on"* we use and instantiate the algorithm of the feedback pattern (see Figure 6). The *identifyRelevantNodes* function of the algorithm is instantiated with the place where the search (i.e. *Books'LightedBooks*) starts to identify the relevant nodes. This function identifies those nodes of the reachability graph where there are books with lights switched on. The other generic part of the algorithm is also instantiated, with place *AnimationSetup'Dynamic_Objects* used to identify the nodes used in the analysis.

```
fun identifyRelevantNodes obj =
PredAllNodes (fn n => cf(obj,Mark.Books'LightedBooks 1 n) > 0)
----------------------------------------------------------------------------
fun counterExampleNodes u =
 let
  val nodes = identifyRelevantNodes u
 in
  let
   val predecessorsNodes =
   SearchNodes (nodes, fn _ => true, NoLimit, fn n => InNodes n, [], op ^^)
  in
    let
      val nodesPredecessorsNodesWith2orMoreSucessors =
      SearchNodes (remdupl(predecessorsNodes), fn n => not(contains nodes  [n])
           andalso length( OutNodes n) >= 2, NoLimit, fn n => n, [], op ::)
    in
      SearchNodes (remdupl(nodesPredecessorsNodesWith2orMoreSucessors),
           fn n => not( contains (map (fn x=> Reachable(n,x) andalso
           length(NodesInPath (n,x))>2) nodes) [true]), NoLimit, fn n => n, [], op ::)
    end
  end
end
----------------------------------------------------------------------------
val FEEDBACK =
          map(fn u => counterExampleNodes u)
                    (UpperMultiSet (Mark.AnimationSetup'Dynamic_Objects 1))
```

Figure 6 - Book's light behavior property (feedback)

After being instantiated, this concrete algorithm identifies, in this case, those nodes where the user is near the desired book and the book's light has not turned on (*FEEDBACK* variable in Figure 6). The algorithm identifies firstly the nodes in the reachability graph where the user is already detected near the bookshelf, but the system is still to react. From these nodes the system can either turn the book's light on, or alternatively choose to process some other relevant event. Selecting the second alternative (doing something else), creates the executions from which a node with the book's light on is not reached (*counterExampleNodes* function). The resulting list of nodes is empty. This means that

for the analyzed scenario (considering the values provided) there is no system execution containing a node where the light should be turned on but was not.

Summing up, the feedback property algorithm was used to verify the property template. As stated the instantiation is simply accomplished, in this case, by indicating the places where the relevant nodes used by the algorithm should be reached.

Reachability

The second property, "*if a book a user is looking for is picked up by another person (stops being available), the user is notified"* is now addressed. This property is a reachability property, and its verification follows the pattern. Reachability properties demonstrate whether it is possible, from one state, to reach the other state (reachability between two nodes of the reachability graph). This is translated in the stated property as if, for every user looking for the same book and every picked up book state, a user notification state is reachable. The property pattern followed describes how the algorithm can be instantiated to check reachability properties.

This property is again executed using a specific scenario. Properties are parameterized using the selected values from the *small color sets* specified in APEXi. For example, in the property verification algorithm (Figure 7), the user *Silva* and the book with id equal to 1 (*userIDxOBJ* and *book* variables) are used because these are elements that compose the new selected scenario for analysis (APEXi selected values). Obviously this scenario should have at least two users looking for the same book.

The idea behind the demonstration of this property is to identify states from which a user picks up a book and the system is not able to reach a notification state for users looking for this book. In other words the aim is to find counter examples where the system does not have the required properties. Figure 7 shows the instantiation of the reachability pattern algorithm. This is achieved by instantiating the *targetNodes* and *originalNodes* functions to identify the relevant nodes (see underlined pieces in Figure 7). The places used to identify the nodes to be used in the analysis (i.e. *BookPickUp'User_Notified* and *BookPickUp'OBJ_deleted*) and concrete tokens to be identified in these places (i.e. *userIDxOBJ* and *book*) are provided. By this means the desired property can be verified.

The execution of this concrete algorithm identifies firstly all notification nodes (returned by the *targetNodes* function). When these have been identified, all nodes at which the book is picked up are identified (returned by the *originalNodes* function). The final stage is to identify any node in which the book is picked up and from which no notification can be made, i.e. no notification node is reachable (hold in the *REACHABILITY* variable). Checking this property using the algorithm (with each of the three users of the selected scenario as parameter) returns no nodes (*REACHABILITY* variable value) which means that for the selected scenario (three users looking for the same book) whenever a user picks up a book it is possible to notify all users looking for the book.

```
val userIDxOBJ =
("Silva",{id="1",objType="book",position={x=120,y=0,z=121}}) : USERIDxOBJ

fun targetNodes obj
= PredAllNodes (fn n => cf(obj,Mark.BookPickUp'Users_Notified 1 n) > 0)

val TS = targetNodes userIDxOBJ
--------------------------------------------------------------------------------

val book =
{id="1",objType="book",position={x=120,y=0,z=121}} : OBJ

fun originalNodes obj
= PredAllNodes (fn n => cf(obj,Mark.BookPickUp'OBJ_deleted 1 n) > 0)

val OS = originalNodes book
--------------------------------------------------------------------------------

val REACHABILITY =
   SearchNodes (OS,
   fn n =>  not ( contains (map (fn x=> Reachable(n,x)) TS)) [true] ),
   NoLimit, fn n => n, [], op ::)
```

Figure 7 - Notification property (reachability)

Precedence

The third property "*the light does not turn on while the relevant user is not near the bookshelf*" follows the precedence property pattern. To reach a state where the light is on, a relevant user must be near the bookshelf. The precedence algorithm consists in firstly identifying the nodes where the light is on and secondly analyzing their predecessors to check the presence of a user close to a bookshelf. The return of zero nodes means that for the selected scenario the property is always true.

Patterns help developers to verify identified properties and then use relevant algorithms for checking the properties.

DISCUSSION

Evaluating a ubicomp environment by analyzing its behavior exhaustively does not guarantee that the proposed design solution provides an adequate experience. As seen with the feedback property pattern, a system satisfying this property could mean that at some level the system provides feedback but nevertheless the crucial elements in the environment that are actually required for feedback are missing from the analysis. Is the feedback provided salient? Can the feedback be actually seen by the user? What will the feedback look like physically? These are issues raised through analysis at the modeling layer. The value of the APEX framework with its multilayered prototyping approach is that these broader questions can be addressed. Each layer supports a specific type of evaluation: observation of virtual objects' behavior, and user reaction to them, within a virtual world (in the simulation layer); analysis of the model (in the modeling layer); observation of real objects (e.g. actual smart phones) connected to the virtual world, and users reaction to them (in the physical layer).

The framework supports a development process in which virtual, physical or mixed elements are explored depending on the availability of these components. The initial stages of development can be achieved entirely in terms of a CPN

model. Further development can be moved into the virtual world before moving wholly or partially into the physical world. In summary it is possible to explore the design from a variety of perspectives.

CONCLUSIONS

This paper introduces a method of evaluating ubicomp environments through exhaustive analysis, applying and adapting heuristics chosen from other areas of software engineering and HCI. Ubicomp environments pose new challenges when compared with traditional interactive systems. The introduced approach enables the successful exhaustive analysis of ubicomp environments through property patterns. These patterns were instantiated in a variety of ways in the context of ubicomp environments leading to the identification of procedures to verify different property templates. The proposed property templates aim to help developers match properties and then to write predicates over the reachability graph making easier the demonstration of properties using APEX. More property patterns (algorithms) emerged through the analysis with the stated property templates. Due to space limitations only some of them have been presented.

APEX through CPN provides a way to analyze exhaustively and formally every portion of the system behavior for selected scenarios and to demonstrate properties on it. The APEX multilayer approach complements this analysis.

ACKNOWLEDGMENTS

This work is financed by the ERDF – European Regional Development Fund through the COMPETE Programme (Operational Programme for Competitiveness) and by the Portuguese Government through FCT – Portuguese Foundation for Science and Technology, within project ref. PTDC/EIA-EIA/116069/2009 (APEX project). José L Silva is further funded by the Portuguese Government, through FCT, under grant SFRH/BD/41179/2007. Michael Harrison is supported by the UK EPSRC (EP/G059063/1) funded CHI+MED project.

REFERENCES

1. Bauchot, F., Clement, J.-Y., Marmigere, G., Picon, J. Method and structure for localizing objects using daisy chained RFID tags. 2007. United States Patent Application Publication Pub. No. US 2007/0257799 A1.
2. Campos, J. and Harrison, M. Systematic analysis of control panel interfaces using formal tools. *Interactive Systems. Design, Specification, and Verification*, (2008), Springer LNCS 5136, 72–85.
3. Campos, J. and Harrison, M.D. Interaction engineering using the IVY tool. *Proceedings of the 1st ACM SIGCHI symposium on Engineering interactive computing systems*, ACM (2009), 35–44.
4. Dwyer, M. and Avrunin, G. Patterns in property specifications for finite-state verification. *, ICSE Proceedings*, IEEE (1999), 411-420.
5. Gellersen, H., Kortuem, G., and Schmidt, A. Physical prototyping with smart-its. Pervasive Computing, 3, 3 (2004), 74–82.
6. Irawati, S., Ahn, S., Kim, J., and Ko, H. Varu framework: Enabling rapid prototyping of VR, AR and ubiquitous applications. Virtual Reality Conference, 2008. VR'08. IEEE, IEEE (2008), 201–208.
7. Jensen, K., Kristensen, L.M., and Wells, L. Coloured Petri Nets and CPN Tools for modelling and validation of concurrent systems. International Journal on Software Tools for Technology Transfer 9, 3-4 (2007), 213-254.
8. Kim, SH., Kim, SW., Park, HM. Usability Challenges in Ubicomp Environment, In the Proceeding of International Ergonomics Association (IEA'03) (Seoul, Korea, Aug 24-29, 2003)
9. Li, Y. and Hong, J.I. Topiary: a tool for prototyping location-enhanced applications. Proc. 17th annual User Interface Software and Technology ACM 6, 2 (2004), 217–226.
10. Mankoff, J., Dey, A., Hsieh, G., and Kientz, J. Heuristic evaluation of ambient displays. Proceedings of the SIGCHI conference on Human factors in computing systems, ACM (2003), 169-176.
11. Nielsen, J. Enhancing the explanatory power of usability heuristics. Proceedings of the SIGCHI conference on Human factors in computing systems, ACM (1994), 152–158.
12. Rukšenas, R., Back, J., Curzon, P., and Blandford, A. Formal modelling of salience and cognitive load. Proc. 2nd Int. Workshop on Formal Methods for Interactive Systems: FMIS, (2007), 57–75.
13. Scholtz, J. and Consolvo, S. Toward a framework for evaluating ubiquitous computing applications. Pervasive Computing, IEEE 3, 2 (2004), 82–88.
14. Scholtz, J., Arnstein, L., Kim, M., Kindberg, T., and Consolvo, S. User-Centered Evaluations of Ubicomp Applications User-centered Evaluations of Ubicomp Applications. Intel Corporation 10, May (2002).
15. Shirehjini, A.N. 3DSim: rapid prototyping ambient intelligence. Smart objects and ambient intelligence, October (2005), 303–307.
16. Silva, J., Ribeiro, Ó., Fernandes, J., Campos, J., and Harrison, M. The APEX framework: prototyping of ubiquitous environments based on Petri nets. Proc. Human-Centred Software Engineering, Springer LNCS 6409 (2010), 6–21.
17. Silva, J.L., Campos, J.C., and Harrison, M.D. An infrastructure for experience centered agile prototyping of ambient intelligence. Proceedings of the 1st ACM SIGCHI symposium on Engineering interactive computing systems, ACM (2009), 79–84.
18. Theofanos, M. and Scholtz, J. A Framework for Evaluation of Ubicomp Applications. First International Workshop on Social Implications of Ubiquitous Computing, CHI, (2005), 1-5.