Prototyping and Analysing Ubiquitous Computing Environments using Multiple Layers

José Luís Silva[a,b,c,d,**], José Creissac Campos[a,b,*], Michael D. Harrison[e,f]

[a]*Dept. Informática / Universidade do Minho, Braga , Portugal*
[b]*HASLab / INESC TEC, Braga, Portugal*
[c]*ICS-IRIT, University of Toulouse III, Toulouse, France*
[d]*Madeira-ITI, University of Madeira, Funchal, Portugal*
[e]*School of Computing Science, Newcastle University, Newcastle upon Tyne, United Kingdom*
[f]*School of Electrical Engineering and Computer Science, Queen Mary University of London, London, United Kingdom*

Abstract

If ubiquitous computing (ubicomp) is to enhance physical environments then early and accurate assessment of alternative solutions will be necessary to avoid costly deployment of systems that fail to meet requirements. This paper presents APEX, a prototyping framework that combines a 3D Application Server with a behaviour modeling tool. The contribution of this framework is that it allows exhaustive analysis of the behaviour models that drive the prototype while at the same time enabling immersive exploration of a virtual environment simulating the proposed system. The development of prototypes is supported through three layers: a simulation layer (using OpenSimulator); a modelling layer (using CPN Tools) and a physical layer (using external devices and real users). APEX allows movement between these layers to analyse different features, from user experience to user behaviour. The multi layer approach makes it possible to express user behaviour in the modelling layer, provides a way to reduce the number of real users needed by adding simulated avatars, and supports user testing of hybrids of virtual and real components as well as exhaustive analysis. This paper demonstrates the approach by means of an example, placing particular emphasis on the simulation of virtual environments, low cost prototyping and the formal analysis capabilities.

Keywords: Ubiquitous and Context-Aware Computing, Modelling, Prototyping, Interactive Systems Analysis, 3D virtual environments

*Corresponding author
**Principal corresponding author
Email addresses: `jlsilva@di.uminho.pt` (José Luís Silva), `jose.campos@di.uminho.pt` (José Creissac Campos), `michael.harrison@eecs.qmul.ac.uk` (Michael D. Harrison)

1. INTRODUCTION

Deploying a system prematurely can be a costly process. For this reason many tools have been developed to specify and to prototype early versions of a design so that the implications of the design can be explored. The development of ubiquitous computing environments brings fresh challenges to prototyping. The impact of a potential design cannot be fully understood without understanding the context in which the design is developed. APEX is a framework that allows designers and developers to model, analyse and simulate ubicomp environments. It is designed to support the development of ubiquitous systems that enhance physical environments with sensors and situated elements. Examples of such elements and sensors are public displays, personal devices, wireless and RFID sensors. The purpose of these ubiquitous systems is to improve the experience of people within the environment. They provide services to occupiers of the space, as they change context, through explicit and implicit interactions.

User experience has been defined as "a person's perceptions and responses that result from the use and/or anticipated use of a product, system or service" [16]. Issues such as physical texture of the environment may often have an important impact on the experience that occupants have of a space. Ubiquitous computing environments involve situated elements. For this reason changing the design can often involve costly physical reconfiguration. While realistic evaluation requires exploration in the proposed physical environment, some development and evaluation may be achieved using prototypes and simulation. APEX uses 3D virtual worlds to create an immersive experience of the space. Virtual immersive 3D environments can be used to simulate relevant (including textural) aspects of the target space. Such simulation enables production of early information about use of the system. Target users can experiment with the system and provide early feedback. This paper extends previous work [1, 3] by describing the whole simulation environment. It focuses on the low cost prototyping features of the tool. It illustrates the approach using an example environment within a smart home. The environment alerts carers when a child is likely to be affected by an asthma trigger.

Hands-on evaluation of a prototype is not sufficient in itself to fully recognise the implications of a design. APEX therefore offers a set of patterns, from which properties or heuristics can be developed. These patterns enable further analysis of the system under development. Empirical techniques for analysis are combined with formal analysis seamlessly. The analysis approach echoes the philosophy of Scholtz and others [22] who use a set of sample measures to evaluate ubiquitous computing applications. These measures assess whether adequate design principles are satisfied and if the design produces the desired user experience. APEX allows exhaustive analysis of a developed prototype behaviour against a set of properties derived from patterns that are supported by the framework.

The APEX framework offers three complementary perspectives of a system under development.

1. A 3D simulation of the environment (created in a "virtual world" supported by

the OpenSimulator[2] web based 3D application server) captures the texture and the spatial characteristics of the environment.

2. Rigorous behaviour models of system behaviour, that include sensors and dynamic objects, can be created, analysed and animated using CPN Tools [7] embedded within the environment.
3. External (physical) devices can be connected to the virtual world using Bluetooth.

Each layer supports a specific type of evaluation. The virtual world simulates the smart environment, making it possible for potential users to explore scenarios within the world. Users can interact with the prototype by manipulating physical handheld devices, using designs proposed for the deployed system, or by controlling avatars located in the virtual world. Avatars can also simulate the behaviour of users autonomously. The modelling layer allows the developer or designer to analyse scenarios systematically, using properties offered by the patterns. Several modelling approaches which were considered as the basis for behaviour models include: (Hybrid high-level Nets (HyNets) [26], Communicating Sequential Processes (CSP) [40], Flownets [41], ASUR++ [42], Interactive Cooperative Objects (ICO) [43] and Coloured Petri nets (CPN) [7]. CPN was chosen because of the substantial set of tools available and its expressive power in the APEX context. The integration of physical components in the physical layer, for example smart phones, allows exploration of how the evolving design would work.

In summary APEX supports:

- the design of ubicomp environments and the exploration of design alternatives, with a particular emphasis on how users will experience these designs;
- analysis either by animation (similar to program execution) or by more formal analysis of behaviour;
- multi-layered development, in which analysis can be combined with evaluation of virtual simulations and actual implementations of components of the proposed design;
- the whole prototyping cycle (design, experience, test and analysis);
- multiple users with collaborative features (e.g. speaking, chatting) enabling interaction between users.

Section 2 discusses literature that is related to the framework. The example that is used to demonstrate the capabilities of APEX is introduced in Section 3. The APEX framework (Section 4) is then described. The method of developing a prototype of the example is described in Section 5. The analysis process is described in Section 6. Section 7 briefly outlines the results of a preliminary evaluation of the framework. Finally (Section 8) conclusions and future work are outlined.

[2]http://opensimulator.org/ (last accessed: 3 December 2012)

2. RELATED WORK

The evaluation, simulation and analysis of ubiquitous systems is already a rich area of research. Relevant research can be categorised in terms of: early evaluation of ubiquitous systems, the development of prototypes using virtual environments and analysis techniques.

2.1. Early evaluation of ubiquitous systems

Current prototyping tools (see [5] for an early overview of approaches) are mostly concerned with single devices, rather than systems of systems combining people and devices. Examples are UbiWise [8], UbiREAL[9], d.tools [20], Topiary [12] and Activity Studio [23]. It is recognised that prototypes should be explored within their envisaged setting [6] and therefore tools produce prototypes either for the real world (for example, d.tools, Topiary and Activity Studio) or for a virtual world (Ubiwise and UbiREAL).

d.tools supports the prototyping of physical devices. It combines elements of a real and simulated world, from design to test and analysis. It allows the development of both physical components (e.g. sensors, actuators) and virtual components in a device editor. The behaviour of the device is modelled using statecharts. The statecharts can be animated for user testing and behaviour can also be analysed. Activity Studio, in contrast, is a tool for prototyping and *in-situ* testing of context aware software applications. It supports the testing of low-cost prototypes in experimentally relevant environments over extended periods. It is possible to explore prototypes over time involving several users, either using real sensors or gathering data from users. Topiary, on the other hand, allows users to explore context aware prototypes using a storyboarding approach. Several other tools use Wizard of Oz techniques to avoid the need for physical sensors and actual physical spaces.

These evaluation and prototyping techniques are valuable but they do not address the interplay between device and environment *in situ* [6]. This interplay is crucial to understanding how the system works as a whole. Displays, devices and sensors form an integrated whole that, together with the physical characteristics of the environment, contribute to the texture of the resulting system.

UbiWise and UbiREAL simulate ubiquitous systems using virtual environments. The simulation acts as a development test bed for the devices and software. The APEX framework, in contrast, supports the design of the environment itself, with a particular emphasis on how users will experience it.

2.2. Simulation using virtual environments

Virtual environments enable the exploration of a proposed ubicomp environment. It does this by means of navigation and interaction within a 3D simulation. These simulations can immerse users. They make it possible to achieve user experience closer to that of the proposed target system. The environment must be sufficiently rich and textured to address usability requirements that depend on the target environment. The simulated environment must produce an impression of what it would be like to use systems once deployed.

3DSim [10], UbiWorld [11], the work of O'Neill and others [13] [24] and VARU [14] all develop simulations of actual environments. 3DSim and UbiWorld use programming languages to build prototypes. The advantage of an approach such as APEX is that modelling, and its associated analysis, can be combined with simulation. Vanacken and others [25], for example, use such techniques to model multimodal interaction techniques. O'Neill and others combine 3D simulation with models to identify occurrences of unwanted system behaviours. APEX, in contrast, aims to provide exhaustive analysis support.

Other approaches, for example VARU, provide user experience without supporting analysis. VARU can be used to develop a tangible space, combining virtual and augmented reality. A rendering game engine built on OpenSceneGraph[3] is used to achieve this.

APEX is unique in supporting analysis of ubicomp environments, combining simulation (similar to program execution) and formal analysis (State Space analysis). It also allows evaluations of hybrids of virtual and physical elements.

2.3. Analysis Techniques

Interaction analysis in ubicomp environments presents challenging problems. The physical environment of the system is important. Interaction can be implicit and therefore unconscious [28]. Kim and others [28] have presented several ubicomp case studies where evaluation has involved making use of physical space and implicit interactions. Interaction within these environments can also be explicit. The devices used for explicit interaction should also be analysed. Different techniques, for example standard usability heuristics for small devices, are required in these cases. Whatever the style of interaction, the user's context plays an important role.

Several HCI techniques support the early analysis of interactive system designs. Examples range from paper prototyping and Wizard of Oz techniques, to the development of versions of the systems for user testing. These approaches require large resource investment. Physical space is required to develop the ubicomp system for realistic evaluation however partial. These costs can be reduced by careful application of techniques that do not require explicit user testing. Such techniques include the use of expert evaluation techniques, for example Heuristic Evaluation and Cognitive Walkthrough. A summary of techniques can be found in [27]. The application of heuristics to a ubicomp application, involving ambient displays, has been explored by Mankoff and others [29]. The problem with using these techniques is that their subjectivity leads them to be unreliable [27].

APEX provides a set of patterns. These patterns derive and extend usability heuristics and enable the generation of formal properties for analysis. Analysis is performed on behavioural models relating to the whole ubicomp system. The analysis is systematic and depends on individual judgement only when considering scenarios where the properties fail.

[3] http://www.openscenegraph.org (last accessed: 3 December 2012)

2.4. *Overview*

To summarise the current state of the art, it is clear that different types of prototyping are possible. Techniques are available that allow evaluation of: a single device isolated from its context of use; an application/device and its context of use; and the environment including the applications and devices contained within it.

Several approaches aim at ubicomp prototyping. These approaches, however, typically focus on context aware applications or isolated devices. They do not prototype ubicomp environments as a whole. Some approaches address the issue of experience but do not address the whole environment. The key features of the research related to the APEX tool are summarised in Table 1.

	UbiReal	d.tools	Topiary	3DSim	UbiWorld	VARU	AS[4]	Ubiwise	OW[5]
Application/isolated devices prototyping	yes	yes	yes	yes	yes	yes	yes	yes	no
Unwanted behaviour identification	yes	yes	yes	yes	yes	yes	yes	no	yes
Ubiquitous environement prototyping	no	no	no	yes	yes	yes	no	no	yes
Provide user experience	no	yes	yes	yes	yes	yes	yes	yes	yes
Formal exhaustive analysis support	no	no	no	no	no	no	no	no	no
Whole cycle of prototyping support	no	yes	yes	no	no	no	no	no	yes

Table 1: Prototyping approaches comparison

No prototyping approach focuses on the user experience of a whole ubicomp environment, while at the same time providing the tools to support formal and exhaustive analysis. APEX aims to fill this gap.

3. EXAMPLE DESCRIPTION

APEX will be demonstrated by using an application intended for a smart home. The application is designed to improve quality of life for child asthma sufferers. The system alerts carers when a child is likely to be affected by an asthma trigger. It also offers carers suggestions about how to act. Cabana et al. [2] have indicated that carers need support to help asthmatic children identify asthma triggers within their environment. Relevant information is sent to a carer's mobile device or to a wall mounted display in a room where the carer is located. Asthma triggers vary depending on the individual. They occur when relevant conditions in the environment are met (examples could include occurrence of tobacco smoke, house dust mites, pets, mould, outdoor air pollution or cockroach allergen)[6].

The developer intends to design a demonstrably safe system that creates a user experience that will encourage use. Examples of properties to be guaranteed include:

[4]Activity Studio [23]
[5]O'Neill' work [13] [24]
[6]http://www.cdc.gov/asthma/triggers.html (last accessed: 3 December 2012)

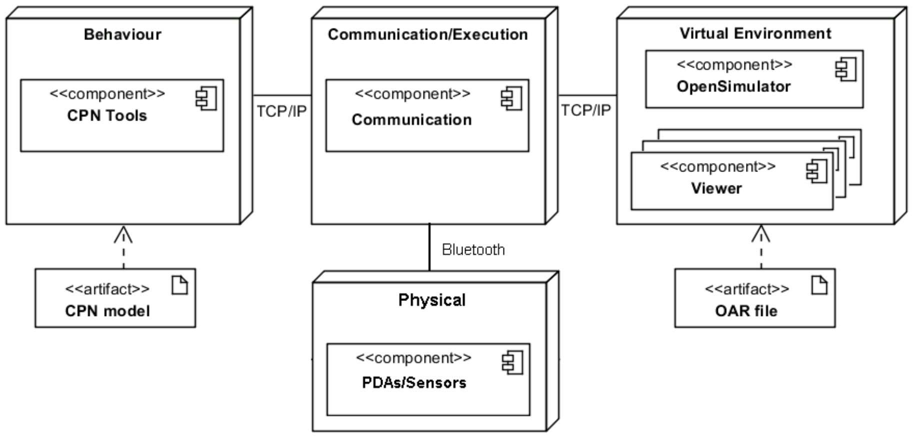


Figure 1: Logical architecture of the APEX framework

- “whenever a child is in danger carers are alerted”;
- “wherever carers go they will receive information about their child when they need to be alerted”.

APEX will be used to develop a model of the proposed design that guarantees these properties. The aim is also that users are able to explore a system prototype immersively within a virtual environment that is driven by the model. Systems such as these are hard to assess through observation alone. There are two reasons for this.

- The target physical environment is complex, involving a number of devices in specific and significant locations.
- The activities that are representative of the use of the system are complicated and impact significantly on the effectiveness of the system.

Analysis and exploration are required in combination. It is not possible to explore all behaviours through observation. At the same time, the simple analysis of properties does not provide a rich enough assessment.

4. THE APEX FRAMEWORK AND PROTOTYPING

APEX enables the development of immersive 3D prototypes. It combines OpenSimulator based virtual worlds and CPN based [7] behavioural models with physical devices. The framework coordinates these different components.

4.1. APEX architecture

The architecture of the framework has four components (see Figure 1).

- A virtual environment component manages the 3D simulation and the construction of the virtual environment.

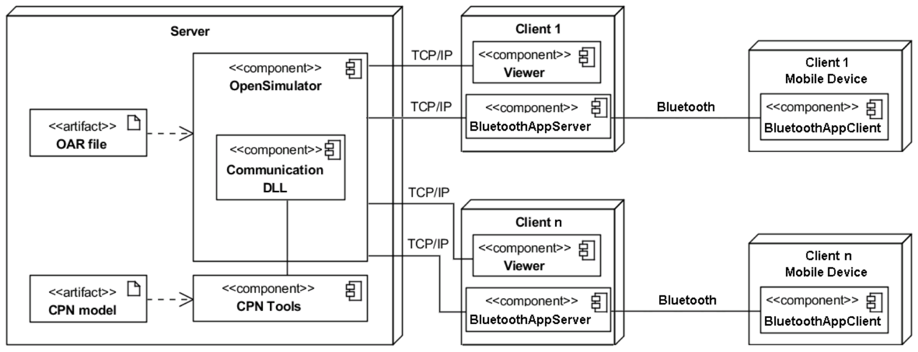


Figure 2: Physical architecture of the APEX framework

- A behavioural component manages the behaviour of the prototype and provides access to the analysis tools.
- A physical component supports connections to physical external devices, such as smart phones and sensors.
- A communication/execution component exchanges data between the other three components during simulation. This component brings the elements together to create a realistic environment.

The architecture aims to make it easy to move between the different layers during development.

4.2. Virtual Environment Component

The virtual environment is distributed across a server machine which hosts OpenSimulator and client machines. A viewer is used to interact with the prototype (e.g. *Second Life Viewer*[7]) in the user's client machine (see Figure 2). Viewers are typically transparent to use. They make it possible to achieve immersive web access for multiple concurrent users. It is important to create a realistic object world to ensure immersion and representative user experience. Immersion levels can range from a desktop to a CAVE (e.g. CaveSL[8]) depending on the viewer used (see [17] for a discussion).

3D application servers offer several advantages over simple virtual environments. It is possible to connect several clients to the same virtual space and thereby to assess the experience of multiple users. OpenSimulator environments can be accessed through a variety of different viewers. Besides the *Second Life* viewer itself, a number of other third party viewers can be used[9]. The main goal of these viewers is to provide

[7]Second Life Viewer: http://secondlife.com/support/downloads/ (last accessed: 15 July 2013)

[8]CaveSL website: http://projects.ict.usc.edu/force/cominghome/cavesl/index.html (last accessed: 3 December 2012)

[9]Third party viewers to connect to Second Life or OpenSimulator: http://wiki.secondlife.com/wiki/Alternate_viewers#nonlinden (last accessed: 3 December 2012)

client access to the environment. However they do differ and this can lead to different user experiences. Some are developed for specific uses (e.g. SL Military), others to support specific visualisations (e.g. stereoscopic 3D visualisation) or specific hardware configurations (e.g. multiple display usage).

Viewers provide limited support for virtual object creation compared with game engines. However, most OpenSimulator viewers do allow the creation and editing of objects and textures. A more detailed explanation of how object editing can be achieved is presented in Section 5. They also enable the description of behaviours using OpenSimulator scripts (written in Linden Scripting Language). Environments are generated as OAR (Opensim ARchive files) reusable packages. Viewers such as the SecondLife viewer and Cool VL Viewer[10] provide further support for polygon meshes using the widely available ".collada" format. These polygon meshes can be downloaded from online repositories. Hence thousands of developed objects can be used off-the-shelf including buildings, furniture, everyday household objects, cars and planes. This can be done using shared repositories of objects such as the Google 3D Warehouse[11]. Objects can also be created using external 3D computer graphics software, such as Blender, Maya, 3DS Max or Google Sketchup, before being imported into the environment. By these means the component enables the creation of complex environments.

4.3. Behavioural Component

The behaviour model coordinates the behaviours of dynamic objects and sensors that compose the environment. The model specifies flows of information and represents the generic structure of the ubiquitous environment. Behaviours can be selected from a collection of predefined modules from the APEX library. These can be combined with tailor-made modules as necessary. A generic *CPN base model* is provided to aid the development of the model of the virtual ubiquitous computing environment. This base model creates a generic CPN style that is relevant to the modelling of virtual environments. It contains modules that:

- initialise the simulation, and establish the connection between the CPN model, as represented by CPN Tools, and OpenSimulator;
- receive data (for example sensors' data) from OpenSimulator and use it to update appropriate tokens;
- describe the behaviour of each device in the system.

A detailed description of the base model is out of the scope of this paper (see [3] for further information). However, its use will be illustrated when extending it to create a prototype of a new ubiquitous environment in Section 5.

The behaviour model is specified using CPN. A CPN model consists of a set of modules that interact with each other through a set of defined interfaces. A module responsible for opening a gate for a user is presented in Figure 3 to illustrate the CPN

[10]Cool VL Viewer: http://sldev.free.fr/ (last accessed: 15 July 2013)
[11]Google 3D Warehouse: http://sketchup.google.com/3dwarehouse/ (last accessed: 3 December 2012)

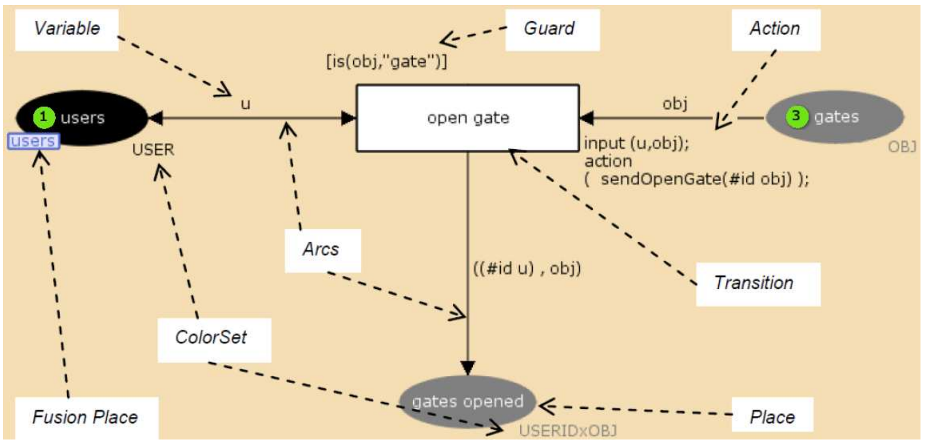


Figure 3: CPN graphical syntax

notation. Each module contains a network of places (represented by ovals), transitions (represented by rectangles), and arcs connecting transitions with places. Each place can contain tokens (represented inside the place by dots with a number indicating the quantity) that carry data values described as token colours. Transitions use variables to manipulate tokens and move them from place to place. The arcs are annotated with the names of the variables that flow through them. Each place can only carry tokens of the type known as the colour set of the place. These types include string, product, record as well as the usual basic types. The CPN components (places, arcs and transitions) can have CPN ML constructs that affect the behaviour of a net associated with them. These are called actions when effecting transitions (for example *sendOpenGate* in Figure 3). Transitions can have Boolean expressions (called guards) that restrict the execution of transitions to when the guards are satisfied. The CPN hierarchy supports places called *Fusion* places that define a set of functionally identical places. Elements of *Fusion* place sets can be used in different modules but they are functionally unique. So anything that is happening within a *Fusion* place set also happens to all other places in the set. The graphical representation of these places is illustrated by the place *users* in Figure 3.

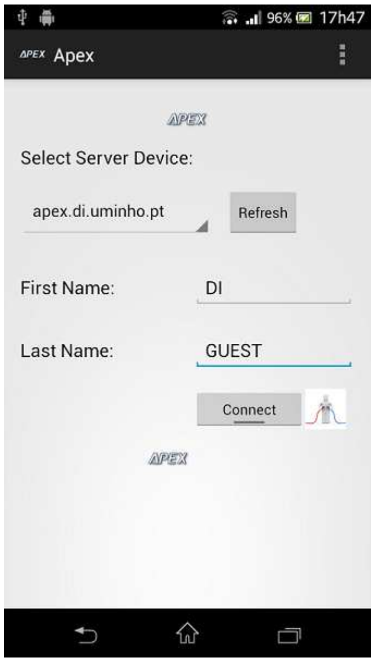
CPN Tools enables the automatic simulation of CPN models, where, through transitions, tokens are moved in the network of places. Options such as the number of steps of the simulation, and the delay in milliseconds of each transition, are provided by the tool. The user can pause or stop the simulation at any time. The CPN simulation works by binding tokens (present in places) to variables of corresponding types present in outgoing arcs. This is done non-deterministically by CPN Tools. Tokens are selected to satisfy the type of the variables on the arcs, and the guards of the destination transitions. Alternatively, the simulation can be stepped through manually by the analyst. This can be done by selecting the token for each binding that is required to execute a step in the model. In the example of Figure 3, when the *open gate* transition is executed, a token from the place *gates* moves to the place *gates opened*. The identifier (*#id u*) of the user, to whom the gate was opened, is added to the token. The token present in the *users* place remains there because the arc connected to the transition is bidirectional. In the case of these arcs, tokens are only queried by transitions and not consumed. During a simulation many transitions can be enabled at the same time. In these cases only one transition is chosen and executed in each iteration. This selection is automatically done by the CPN Tools. The selection uses a fair algorithm that takes into consideration previous selections. However, more recent versions of CPN Tools make it possible to associate priorities to transitions. These priorities enable the modeller to specify which transition will fire first when there is more than one transition enabled at the same time. More information can be found in [39]

The State Space (SS) tool, that is also part of CPN Tools [4], can be used for analysis (e.g. whether a carer is warned when a child approaches an asthma trigger). The tool generates a reachability graph that indicates the states, that specify a specific property, that can be reached from some starting state. Each node of the graph represents an execution state, while arcs represent actions that lead from one state to another, see for example Figure 16 (page 22). The whole graph can be used to represent all the possible executions of the ubicomp system subject to predefined constraints. The arcs and labels of the graph can be checked interactively by the tool. Verification of a property requires the application of a predicate to relevant states in the reachability graph. The returned result is either that the predicate is true of all states, or that it fails to be true and examples, for which it is false, are provided. These examples are then used as a basis for exploration of a situation that may be of interest from a design point of view.

CPN Tools can support the simulation of systems that involve many users. The modelling layer allows the creation of scenarios that use programmed avatars. These scenarios are designed to simulate the experience of situations where there are several users. Avatars can be modelled using different navigations through the environment. By this means one real user can appear to experience situations where many users are present. For example, several avatars can be programmed to arrive at an asthma trigger at the same time. Hence behaviour of the system can be observed when many children are within proximity of an asthma trigger. APEX can be used to model the programmed avatars' movement. This makes it possible to simulate implicit interactions within a ubiquitous environment. Explicit interactions are not currently supported. Avatar movement is either defined manually or uses previously recorded information taken from real users exploring the simulation. The framework supports switching between non-programmed (driven by real users) and programmed avatars in real time.

4.4. Physical Component

The APEX physical component (see Figure 1) connects external devices, such as smart phones and sensors, to the framework. Receiving sensor data, as well as sending information to actual implemented components in the physical world, can be achieved using this component. Systems can evolve gradually by replacing virtual entities with physical entities. The connection between external devices and the virtual world is achieved via Bluetooth using the Communication/Execution component. This is discussed in the next section. A Bluetooth client application is installed in each of the mobile devices. At the same time a Bluetooth server application is installed on the



Select Server Device:

apex.di.uminho.pt

Refresh

First Name: DI

Last Name: GUEST

Connect

Figure 4: Bluetooth Client Application installed in a smartphone running Android

client machines (running in parallel with the viewer – see Figure 2). Clients communicate with OpenSimulator via TCP/IP. APEX detects mobile devices automatically. It links them to relevant avatars in the virtual environment by using login information established when users connect the mobile device (see Figure 4).

Users can interact interchangeably, either with physical objects in the physical layer, or with virtual objects in the simulation and modelling layer. Different combinations of physical and virtual objects can be used as prototypes of the system at different stages of the development process. Interaction with physical devices enables users to experience *physical* aspects of the proposed target ubiquitous environment. For example, the smart phone application used in the proposed design could be prototyped either as a simulated smart phone or as the actual smart phone.

4.5. Communication/Execution Component

The communication/execution component (see Figure 1) coordinates the components of the framework. This component recognises changes in the Virtual Environment or Physical component and notifies relevant Behavioural and/or Physical components. Changes are triggered explicitly as a result of direct user action, or implicitly by sensors in the environment. Actions, triggered by the Behavioural component, are reflected in both the virtual environment and the physical devices. Communication between the model and the communication/execution component is achieved using functions (for example, *sendUserInfo* – see Figure 11, page 18) based on predefined ones within CPN Tools [15].

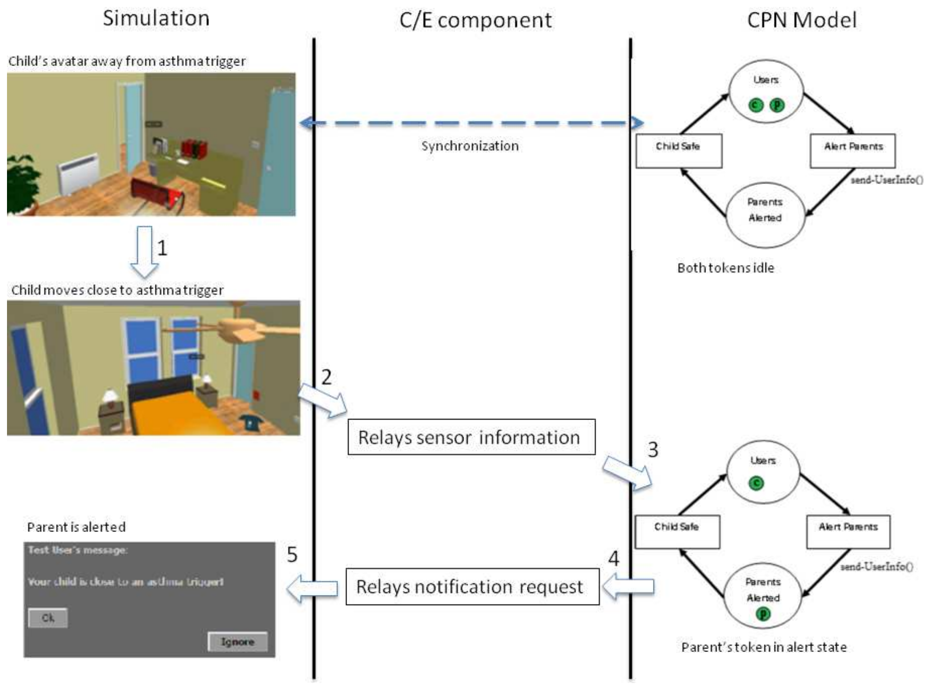

Simulation
C/E component
CPN Model
Child's avatar away from asthma trigger
Users
Child Safe
Alert Parents
Parents Alerted
send-UserInfo()
Synchronization
Both tokens idle
1
Child moves close to asthma trigger
2
Relays sensor information
3
Parent is alerted
Test User's message:
Your child is close to an asthma trigger!
Ok
Ignore
5
Relays notification request
4
Parent's token in alert state

Figure 5: Overview of the asthma alert working process

Communication in CPN Tools is achieved by means of Comms/CPN [15]. A CPN ML library connects CPN Tools to external processes. An appropriate module must be loaded into the external process (in this case OpenSimulator) so that Comms/CPN can be used. Java and C modules are available with the distribution for this purpose. OpenSimulator modules (DLLs) are developed in C#. Hence a new C#/CPN communication module (DLL) was developed enabling the communication between CPN models and C# processes.

Figure 5 illustrates the sending of an alert to carers when their child comes close to an asthma trigger. The left column represents the virtual environment, the middle column the Communication/Execution component, and the right column the CPN model responsible for specifying the behaviour of the parent alert system. To improve readability, the figure presents a simplified version of the actual model. At the first step the avatar of the child is in a room with no asthma triggers. A token that represents the child is in the *Users* place (token *c* in the figure). The carer token is also on the *Users* place because he or she has not yet been alerted (token *p* in the figure). The place *Users* holds the children and carers not alerted. In step 1 the child's avatar moves close to an asthma trigger. At this moment the APEX communication/execution component is notified of the identity of the avatar approaching the trigger. This is done by the presence sensor which is located near the trigger (step 2) . In step 3 a state change takes place in the alert system. The carer's token is moved to the *Parents Alerted place* using the

Alert Parents transition. This transition is accompanied by an associated action (*sendUserInfo()*). A result of this transition is that a notification request (with the identifier of the carer's avatar) is sent to the APEX communication/execution component (step 4). Finally, the APEX communication/execution locates the avatar and sends it the alert message (step 5).

This process is fully automatic once the system is set up. Set up is achieved by extending the *CPN base model* with relevant modules. In addition the dynamic objects are put in the simulation layer. Consistency across the multiple representations at different layers, is maintained by the communication/execution component. Each dynamic object/sensor in the simulation layer contains a unique identifier, used to represent it in the modelling layer. The predefined behaviour of sensors can be configured using the viewer (detailed description in Section 5). A script is used to define dynamic objects behaviour. Data is exchanged between the layers, as strings that carry the identification of the relevant object, as well as the event that is being communicated. This information enables the update of the receiver component to reflect the changes in the sender component. Information exchange occurs in both directions. The scripts linked to the dynamic objects are designed to both react to changes in the environment, and to effect changes in the object, to assure consistency with the state of the CPN model.

5. USING APEX TO DEVELOP A PROTOTYPE

This section describes how APEX was used to create the virtual environment and associated behavioural models for the asthma example. The multi-layer approach is illustrated through this example.

5.1. Virtual Environment

The target environment for the proposed system is the Aware Home[12] at Georgia Institute of Technology (GaTech). The home has two identical floors with nine rooms on each floor. It was originally designed to explore emerging technologies and services in the home. The prototype is explored in a virtual environment that represents the space, the sensors and the people within the Aware Home. The spatial organisation and position of the various sensors is indicated in the floor plan (Figure 6). 16 presence sensors are used in the prototype to detect the location of people. 16 environment sensors detect environment conditions (for example smoke or air quality). Presence and collocated environment sensors are distributed across rooms. Their locations are indicated by the numbers in Figure 6. They are used to detect users and environmental conditions everywhere in the house. The Cool VL viewer was used to create the virtual environment. 3D models of the Aware Home, developed at GaTech using Google Sketchup, were used as a starting point. The virtual environment is to be sufficiently close to the physical target system to provide an adequate and realistic experience for users. The rooms were furnished in part by uploading furniture objects selected from an on-line 3D warehouse[13]. New objects were also developed within the viewer by

[12]Aware Home: http://awarehome.imtc.gatech.edu (last accessed: 4 December 2012)
[13]3D warehouse: http://sketchup.google.com/3dwarehouse/ (last accessed: 4 December 2012)

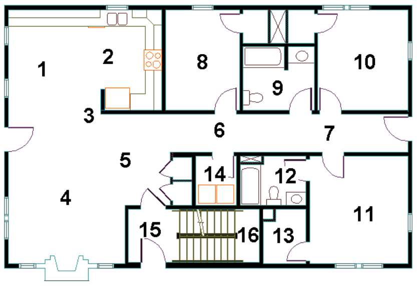


Figure 6: Aware Home floor plan (without furniture) with inserted sensors (one presence sensor and one environment sensor present in each number)

linking basic shapes to build the desired elements (e.g. chairs, bookshelves, tables – see Figure 8 for an example of constructing a chair). The resulting virtual environment within Opensimulator is illustrated in Figure 7.

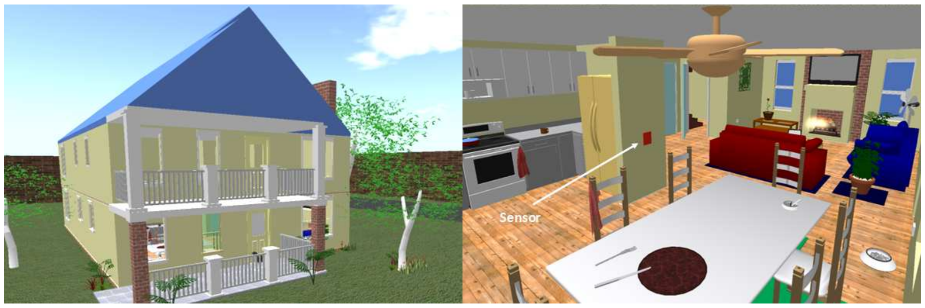
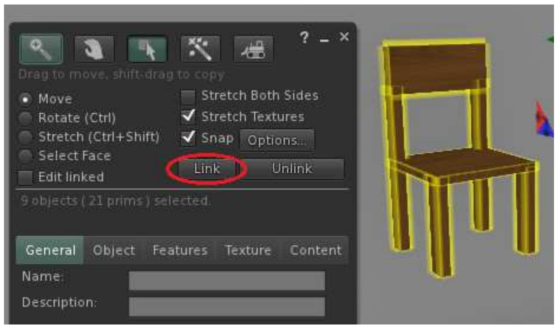


Figure 7: Aware Home 3D environment



Drag to move, shift-drag to copy

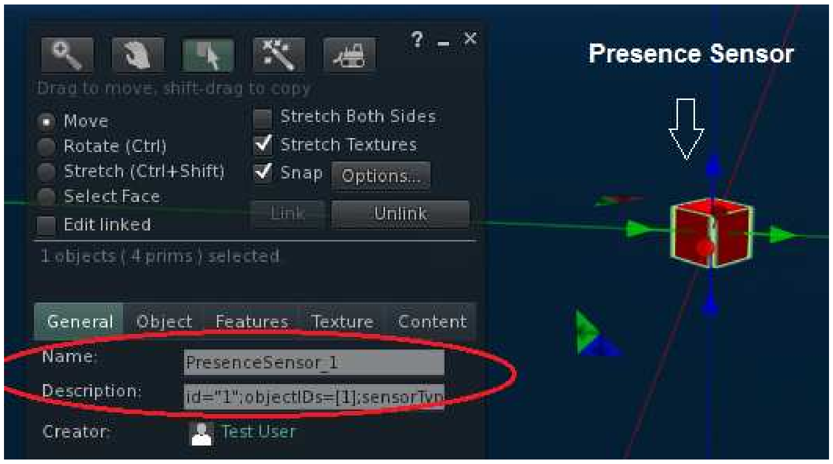
[✓ Move]
Rotate (Ctrl)
Stretch (Ctrl+Shift)
Select Face
Edit linked

Stretch Both Sides
[✓ Stretch Textures]
[✓ Snap]

9 objects (21 prims) selected.

Figure 8: Linkage of the elements composing a chair

After being created, the virtual environment needs to be set up to work with the CPN model. Each dynamic object and sensor that is present in the virtual environment must be configured through the viewer. This is done by accessing the property panel



- [● Move]
- Rotate (Ctrl)
- Stretch (Ctrl+Shift)
- Select Face
- ☐ Edit linked
- ☐ Stretch Both Sides
- ✓ Stretch Textures
- ✓ Snap

1 objects (4 prims) selected

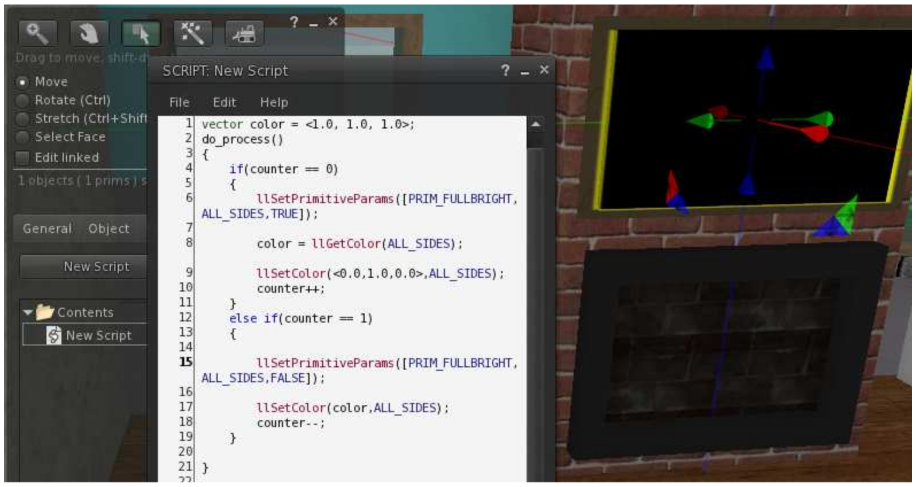
Name:	PresenceSensor_1
Description:	id="1";objectIDs=[1];sensorTyp
Creator:	Test User

Figure 9: Sensor's attributes

of the object (see Figure 9). Some environment conditions must be satisfied so that it is possible to create appropriate animations.

- Each *dynamic object* in the virtual environment (represented by a token in the CPN model) must:
  - have a unique ID present in the field Name. This ID is used to identify the objects in both layers (thus linking/associating a token to the correct object in the virtual environment);
  - indicate its object type using the field Description (e.g. *object type* = *screen*).
- Each sensor must be loaded from the pre-defined sensors provided (OAR files). Alternatively new ones can be defined. Figure 9 illustrates sensor features as follow.
  - The fields *Name* and *Description* must be changed to reflect the desired values.
  - The *objectIDs* list present in the *Description* field of the *Presence Sensors* represents the *Ids* of the objects that the sensor affects.
  - The *sensorType* present in the *Description* field of the *Sensors* indicates the type of the sensor.
  - The *threshold* present in the *Description* field of the *Sensors* represents the distance from which the sensor reacts.

The behaviour of the dynamic objects in the virtual environment is triggered by associated CPN modules (the link being established by common token and object IDs). It is made concrete by LSL (Linden Scripting Language) scripts. When a screen is in the show alert CPN state, this must be reflected in the environment. The concrete



```
vector color = <1.0, 1.0, 1.0>;
do_process()
{
    if(counter == 0)
    {
        llSetPrimitiveParams([PRIM_FULLBRIGHT,
ALL_SIDES,TRUE]);

        color = llGetColor(ALL_SIDES);

        llSetColor(<0.0,1.0,0.0>,ALL_SIDES);
        counter++;
    }
    else if(counter == 1)
    {

        llSetPrimitiveParams([PRIM_FULLBRIGHT,
ALL_SIDES,FALSE]);

        llSetColor(color,ALL_SIDES);
        counter--;
    }

}
```

Figure 10: Dynamic Object Script association

mechanism for doing this is a script linked to the relevant object in the environment. The script is triggered by the behavioural component. In the smart home example, scripts are associated with screens to display alerts. Figure 10 shows part of the script that specifies how information to be displayed is linked to a public screen dynamic object. The script is linked in the example to the object by using the object script association feature provided by the viewer.

5.2. The behaviour model

An APEX behaviour model combines "off the shelf" and purpose developed modules to express the behaviour of the different components in a particular ubiquitous computing environment. The model for the asthma system prototype uses two modules, described in Figure 11 and Figure 12, that hold information about the users and the sensors present in the environment.

The purpose of the first module is to alert carers when their child is too close to an asthma trigger. Places are associated with different hues to improve readability. No semantics is associated with these colours. The *Alert Parent* transition is defined in the "alerts carers" module, while the *child safe* transition removes the alert. Whether the module alerts or removes alerts depends on information held in *Users* (which contains carers and children that are not in danger) and *P_sensors* (which contains the presence sensors) places. Access to the values, held in places, is by means of the variables associated with arcs (for example, *u* and *u1* represent users and *ps* represents a presence sensor). The alert has behaviour that is described by the *Alert Parent* transition in Figure 11. The system alerts the carer as the child approaches a trigger related to a specific allergy. The transition can fire when (expression between square brackets) a child *u*, with a parent *u1* is near a presence sensor *ps* and the parent has not already been alerted. The presence sensor is modelled using the *userNearPresenceSensor* function. Firing the transition results in the update of carer information with

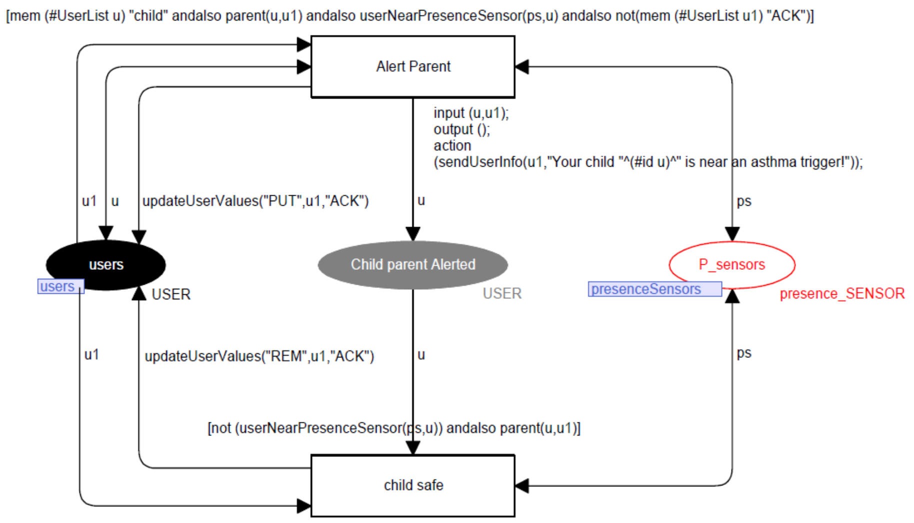

[mem (#UserList u) "child" andalso parent(u,u1) andalso userNearPresenceSensor(ps,u) andalso not(mem (#UserList u1) "ACK")]
Alert Parent
input (u,u1);
output ();
action
(sendUserInfo(u1,"Your child "^(#id u)^" is near an asthma trigger!"));
u1
u
updateUserValues("PUT",u1,"ACK")
u
ps
users
users
USER
Child parent Alerted
USER
P_sensors
presenceSensors
presence_SENSOR
u1
updateUserValues("REM",u1,"ACK")
u
ps
[not (userNearPresenceSensor(ps,u)) andalso parent(u,u1)]
child safe

Figure 11: Parents alert system behavioural model

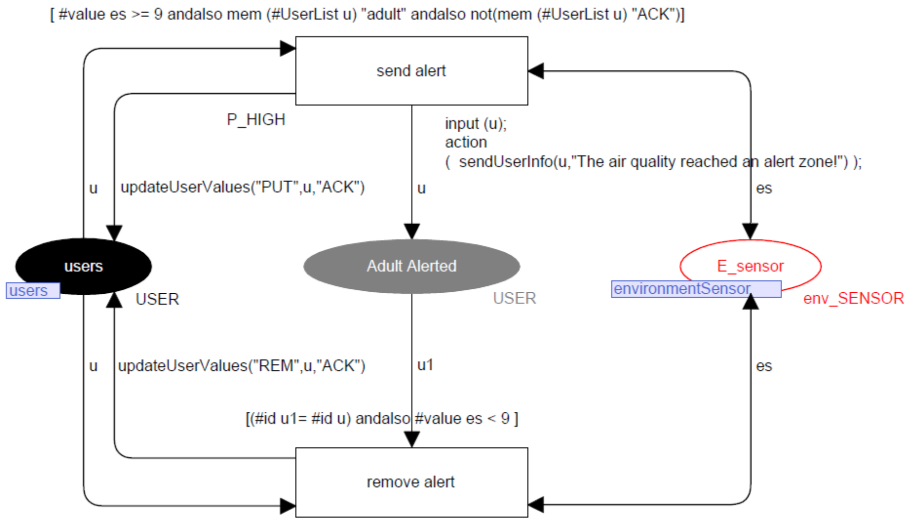

[#value es >= 9 andalso mem (#UserList u) "adult" andalso not(mem (#UserList u) "ACK")]
send alert
P_HIGH
input (u);
action
(sendUserInfo(u,"The air quality reached an alert zone!"));
u
updateUserValues("PUT",u,"ACK")
u
es
users
users
USER
Adult Alerted
USER
E_sensor
environmentSensor
env_SENSOR
u
updateUserValues("REM",u,"ACK")
u1
es
[(#id u1= #id u) andalso #value es < 9]
remove alert

Figure 12: Air quality alert system

a warning (meaning that the carer is alerted). This is described by the *updateUserValues("PUT",u1,"ACK")* function executing while placing the tokens in the *users* place. Additionally, the *Alert Parent* transition moves the child's token into the *Child parent Alerted* place. During this transition carers are alerted using the *sendUserInfo* function. This function, with the respective user and message as parameters, enables the communication/execution component to trigger a script in the relevant objects(s) in the virtual environment. When the child is no longer close to an asthma trigger, indicated by *not(userNearPresenceSensor(ps,u))*, the token is removed from the *Child parents Alerted* place. The earlier warning is then removed from the relevant carers using *updateUserValues(REM,u1,ACK)*.

The module described in Figure 12 sends an alert when the environment reaches one of its alert states (e.g. air polluted). The environmental sensor (*E_sensor*) models environmental air quality as an integer. When the value is greater than or equal to 9 an alert zone is reached. This module is structurally similar to the one just described. The main difference is that an arc is removed that was used in the previous model to identify the carer of a specific child. This kind of association is not necessary because the environment alerts are sent to all adults. The illustrated components are chosen for their simplicity to aid explanation. It is not the present purpose to elaborate the CPN modelling approach. A detailed description of more elaborate models can be found in previous work [3].

6. USING APEX TO EVALUATE A PROTOTYPE

The ubicomp environment prototype described in the previous section can now be evaluated using APEX. A detailed description of how to set up the environment, and a list of commands provided by APEX, can be found at the APEX website[14].

6.1. Analysis of User Experience

Different user experiences can be elicited with various versions of the prototype. Variations can be produced using different combinations of physical and virtual components. Example combinations that were explored in this case included: simulating a smartphone or tablet as a popup window, and a more immersive option with the smartphone itself.

The proposed use of APEX simulations is with target users. However, meaningful feedback about the design prototype can also be derived by developers as they explore the implications of their design decisions. Improvements in the design resulted from developer exploration of the simulation. For example, it became clear through simulation that some users prefer to be notified using fixed displays, while others would rather use their smartphones. Users do not always have their smartphones with them, so the system should allow carers to choose how the alert is to be transmitted. Figure 13 shows a carer experiencing a proposed alert system. The small window (bottom right corner) represents the user's (virtual) smartphone set up to receive alerts prior to

[14]http://wiki.di.uminho.pt/twiki/bin/view/Research/APEX/Documentation (last accessed: 2 December 2013)

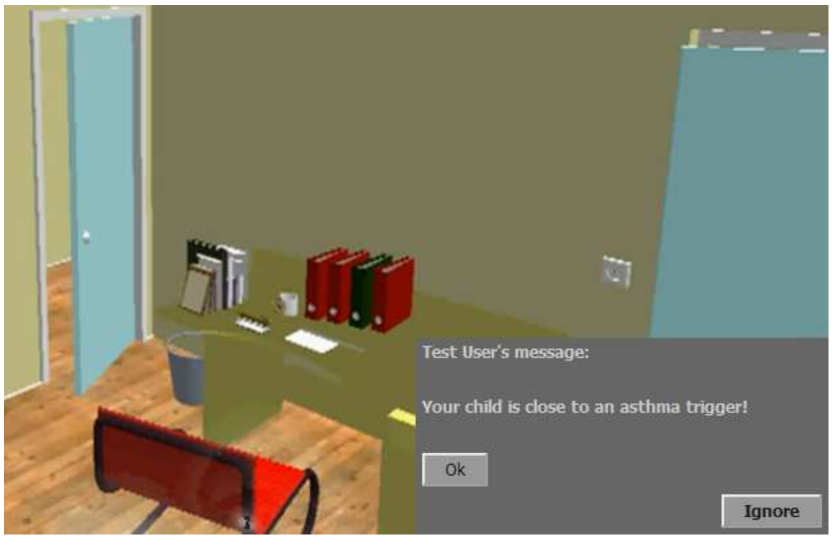


Figure 13: Aware Home alert system user experience

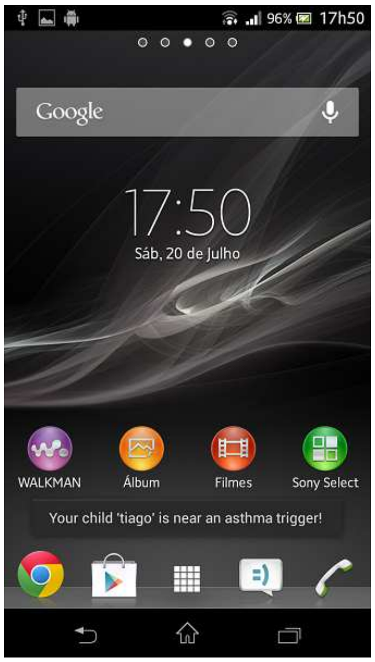
the availability of the physical phone itself. The physical devices could alternatively be connected through Bluetooth when available. Figure 14 (page 21) illustrates the reception of a carer alert ('Your child "Tiago" is near an asthma trigger!') via a smartphone connected to APEX. Figure 15 illustrates the carers' reception of an action plan via their (simulated) smartphone when their child is near an asthma trigger (dust mites).

User experience of this virtual environment offers new dimensions of evaluation. It is possible to address features such as salience, user preference or line of sight. Where carers receive alerts at fixed displays a number of issues were identified. Firstly, the displays were not always positioned to guarantee a continuous line of sight. Secondly, some form of acknowledgement was required from an alerted carer within a specific time-out to ensure that the carer was aware of the alert. A more persuasive (e.g. louder or harsher) alert could be sent if there is no response. Alternative solutions, designed to take these issues into account, can be developed and evaluated with little effort.

This approach to the development of ubicomp environments is flexible. It focuses on providing user experience, whatever resources are available, and broad because it embraces important aspects required to prototype ubiquitous environments.

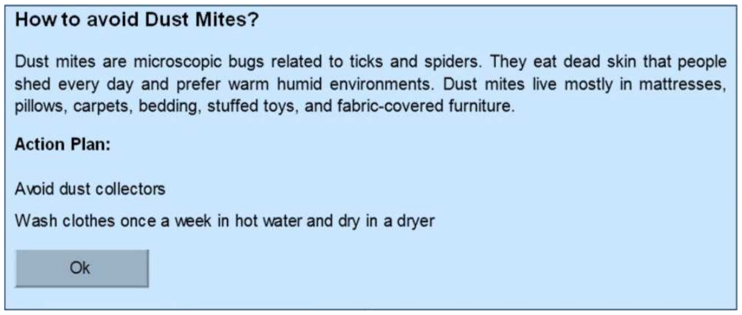
6.2. Formal Behavioural Analysis

APEX makes it possible to complement observation of the simulated system with an exhaustive analysis of possible user behaviours, using the behavioural component. This is achieved by checking that properties of the system hold, subject to specific and defined constraints. Standard templates are made available in APEX. They are designed to help the developer discover properties that are relevant to evaluating the system under design. APEX provides property patterns that combine these property templates with analysis assistance. The patterns explain how to use the relevant CPN Tools to check the instantiations of the templates that have been created.



Your child 'tiago' is near an asthma trigger!

Figure 14: Asthma trigger parent's alert via their smartphone


How to avoid Dust Mites?

Dust mites are microscopic bugs related to ticks and spiders. They eat dead skin that people shed every day and prefer warm humid environments. Dust mites live mostly in mattresses, pillows, carpets, bedding, stuffed toys, and fabric-covered furniture.

Action Plan:

Avoid dust collectors

Wash clothes once a week in hot water and dry in a dryer

Figure 15: Asthma trigger parent's action plan via their smartphone

A ubicomp system design is typically complex. The number of states that would require exploration, to check the truth of a property, is likely to be intractably large. Analysis is therefore focused by constraining the system. State reduction is achieved by narrowing considerations of the model to a restricted closed version, using experimental data derived from the virtual and physical layers. These data, we call scenarios, make it possible to consider all paths within the context of only those conditions that are likely to be encountered at run time. The mechanism for construction of the scenarios is described in the next section. It is necessary, for example, to limit the values each variable might take (e.g., the range of pollution levels read by a sensor).

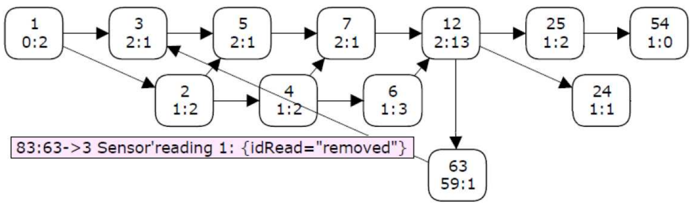


Figure 16: Reachability graph

CPN Tools uses the State Space (SS) tool to bind each variable to each of the provided known values using brute force. The tool creates a reachability graph of the type illustrated in Figure 16. The highlighted arc 83, in the reachability graph, represents the firing of the transition *reading* with *idRead* bound to the value "*removed*". Nodes 63 and 3 correspond to the states, before and after the firing of the transition respectively. CPN Tools makes it possible to inspect nodes. The state corresponding to a node can be visualized using the CPN Tools simulation facility. Values, and the choice of binding elements generated by the model assumptions, can be explored by this means.

Properties generated from the templates can be checked by queries relating to the generated graph (see the *CPN Tools State Space manual* [4] for more details). For example, the *Reachable* query returns a path from the original node to a defined destination node. This can be used to determine whether conditions such as "if after parents have been alerted they can be alerted again" hold. *ListDeadMarkings* finds system states from which it is impossible to do anything. In other words if the system reaches them no further action can be taken. *ListLiveTIs* provides the list of live transitions that can always be enabled again. These standard queries can be augmented by further predicates that are specific to particular properties required of the model.

6.2.1. Patterns

A pattern captures a known solution to a given recurring problem. The use of verification patterns has already been established in other fields of software engineering [31]. Indeed, a specific set of patterns, inspired by usability heuristics [26], has already been selected for use in analysing interactive devices [32]. Applying patterns, in the context of APEX, raises fresh challenges. Property templates, developed for other purposes, may not be relevant to ubiquitous systems. If the template is to be of value then developers should be able to understand how properties, instantiated from the template, can be checked using CPN tools. Property instances must be translated into a form that is meaningful for analysis. Guidance should be provided about how the CPN Tools algorithms are created. APEX models combine behavioural models with rich interaction context including devices and users. Unlike the simpler analysis of individual devices, it is important to be clear when applying the patterns:

Who are the users? Several users might be present in the environment. User actions may involve different sub-groups within the environment (for example, carers or children). System responses may affect other parts of the community, e.g., an action by one user might trigger a system response directed to a different user.

What are the actions? In a ubicomp setting interaction may be implicit as well as explicit. Response may occur as a result of implicit user action or changes to the environment, e.g., a user entering or leaving a room.

What is being analysed? The broader idea of system, implied by ubicomp, leads to concern about whether the analysis 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 systems design, or is being used to validate the model itself.

Verification involves constructing an algorithm over a reachability graph that can be executed within the SS-tool. The pattern template is parameterised on events and system responses that appear in the model. By this means the property can be related to a specific scenario. Events and system states are represented as tokens defined in relevant places in the behaviour model. The algorithm is instantiated by identifying relevant tokens and places. Example patterns are now presented.

The Consistency Pattern.

Intuition Consistency [33] captures the requirement that a given event or system state condition always causes a defined effect. An event may be an implicit or explicit user action.

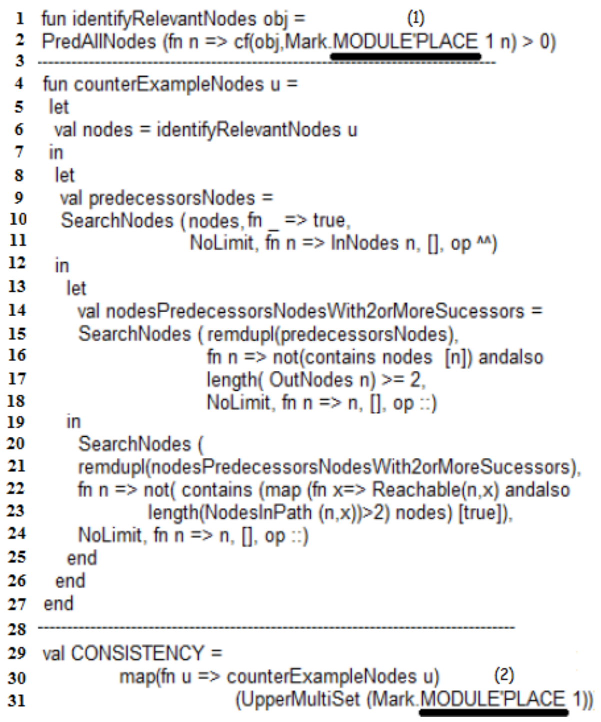
For example, an event may be a change in the environment that the system has sensed. This would occur if the temperature of the space had reached a threshold. Alternatively, the pattern could be defined in relation to the temperature being above or below the threshold (a state condition) instead of having reached it (an event). The defined effect is described in terms of the state of the system as a whole. It could be defined in terms of a personal profile, the presence of other users or environment information. The effect of the event in the environment may or may not be perceivable by users.

An example consistency property in the asthma system is “a child near an asthma trigger is always detected by a sensor”.

Algorithm Figure 17 presents the algorithm skeleton used to verify Consistency properties (written in CPN ML).

The result of the verification is given in the CONSISTENCY variable (line 29 in the figure). The variable contains node identifiers for which the verification fails. It will be empty if it succeeds. The nodes represent counter-examples to the particular instance of the property template being verified. They illustrate states of the system that falsify the property.

To instantiate the template, appropriate places in the Petri net model must be identified. These define the relevant event/state condition and system response. This is done by instantiating the underlined terms in the figure. Term (1) – line 2 in Figure 17 – must correspond to places where the effect is observed. Term (2) – line 31 – corresponds to places that hold tokens that together produce the event or represent the state condition for which we want to analyse the system response.



```
1  fun identifyRelevantNodes obj =                (1)
2  PredAllNodes (fn n => cf(obj,Mark.MODULE'PLACE 1 n) > 0)
3  ------------------------------------------------------------------------
4  fun counterExampleNodes u =
5   let
6    val nodes = identifyRelevantNodes u
7   in
8    let
9     val predecessorsNodes =
10    SearchNodes (nodes,fn _ => true,
11                      NoLimit, fn n => InNodes n, [], op ^^)
12   in
13    let
14      val nodesPredecessorsNodesWith2orMoreSucessors =
15      SearchNodes ( remdupl(predecessorsNodes),
16                        fn n => not(contains nodes  [n]) andalso
17                        length( OutNodes n) >= 2,
18                        NoLimit, fn n => n, [], op ::)
19    in
20      SearchNodes (
21      remdupl(nodesPredecessorsNodesWith2orMoreSucessors),
22      fn n => not( contains (map (fn x=> Reachable(n,x) andalso
23                 length(NodesInPath (n,x))>2) nodes) [true]),
24      NoLimit, fn n => n, [], op ::)
25    end
26   end
27  end
28 ------------------------------------------------------------------------
29 val CONSISTENCY =
30             map(fn u => counterExampleNodes u)         (2)
31                         (UpperMultiSet (Mark.MODULE'PLACE 1))
```

Figure 17: Consistency/feedback property algorithm skeleton

Looking at lines 29 to 31, it can be seen that the value of the CONSISTENCY variable is determined by applying (using the *map* function) the *counterExampleNodes* function to all relevant tokens in the scenario to be verified. These tokens are calculated by the *UpperMultiSet* function from the places instantiated in term (2).

The *counterExampleNodes* function (lines 4 to 27) identifies states where the desired effect is verified (variable *nodes* - line 6). This is done using the *identifyRelevantNodes* function (line 1 and 2), which uses *PredAllNodes* to calculate all nodes that have markings in the places instantiated in term (1) (i.e., those places where the effect is observed). The *counterExampleNodes* function then takes these states and explores the alternative behaviours that might have occurred in the presence of the event or condition. These are models of highly concurrent systems that may be reacting to a number of different events simultaneously. The algorithm checks whether these other behaviours also satisfy the desired effect.

The exploration of alternative behaviours is done in three steps. First (lines 9 to 11), a search is made for all predecessores of the nodes in variable *nodes*. The result is available in variable *predecessorsNodes*. Then (lines 14 to 18), the result of that search is filtered to consider only those nodes that have alternative behaviours (variable *nodesPredecessorsNodesWith2orMoreSucessors*). Finally (lines 20 to 24), a search is made for those nodes that have behaviours in which the desired effect is not observed. If any node is found then a state has been found where the desired effect was not observed.

The Feedback Pattern.

Intuition To provide adequate feedback is a key principle in Human Computer Interaction. Feedback is understood in the APEX context to be a response reflected in the environment as a whole to specific events. Events can be either explicit or implicit actions by the user. Response represents an observable change in the environment.

The person that causes the system's response is not necessarily the same as the person to whom the response is directed. For example, in the case of "whenever a child is in danger then carers are alerted", the event is an implicit action that occurs when the child "approaches" an asthma trigger. The response which is defined in the model will be the effect of changing the environment so that parents are alerted. How salient, for example visible, the feedback is makes this property pattern different from consistency. Evaluating the salience of feedback will require complementary evaluation at the simulation layer.

An example feedback property in the asthma system is "while a child is in a danger zone, an alert is provided by the system".

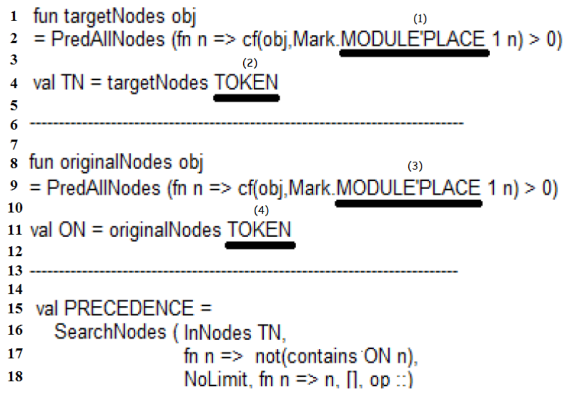
Algorithm Feedback is a specialisation of the consistency pattern. The consistency requirement is further specialised to ensure that the response event is always perceivable by relevant user(s). This can be achieved by using the algorithm for the consistency pattern, with the added restriction that to verify a feedback property the relevant nodes being analysed (underlined term (2) - line 31 in Figure 17) should correspond to the perceivable effects being produced.

The Precedence Pattern.

Intuition This pattern captures the requirement that some event (the consequent) must always be precede by some other event (the antecedent). The consequent event is enabled by the antecedent event. If the consequent event occurs then the precedent event must have occurred. An example of an instance of this pattern would be a requirement that if the carer has been alerted then a child must have approached an asthma trigger. This complements the Feedback property that states that while a child is in a danger zone, an alert is provided by the system.

Algorithm The algorithm skeleton for this pattern is presented in Figure 18. It identifies the consequent states (variable *TN* – line 4) and then identifies each predecessor and whether it satisfies the antecedent (variable *ON* – line 11). The *PRECEDENCE* variable (line 15) contains the list of predecessor nodes not satisfying the antecedent. If this list is empty then the property is verified true.

As before, the underlined terms in the algorithm presented in Figure 18 are the terms to be instantiated. The tokens (TOKEN - lines 4 and 11) to look for, and the place in the model that needs to be searched (MODULE'PLACE - lines 2 and 9), must be provided with appropriate values for both the consequent (terms (1) and (2)) and the antecedent (terms (3) and (4)).



```
fun targetNodes obj
= PredAllNodes (fn n => cf(obj,Mark.MODULE'PLACE 1 n) > 0)

val TN = targetNodes TOKEN

---------------------------------------------------------------------------

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

val ON = originalNodes TOKEN

---------------------------------------------------------------------------

val PRECEDENCE =
   SearchNodes ( InNodes TN,
                 fn n =>  not(contains ON n),
                 NoLimit, fn n => n, [], op ::)
```

Figure 18: Precedence property algorithm skeleton

The Reachability Pattern.

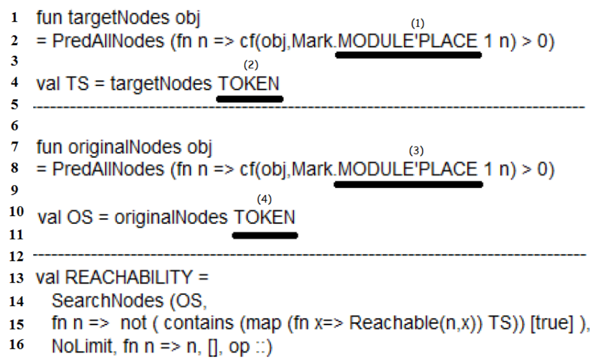
Intuition Reachability requires that the system be able to reach a specific state or situation. It asserts that the system can always evolve from one specific (source) state to another specific state (the target state).

Reachability properties relevant to the example are "when no alarm is raised, a situation can be reached where no child is in danger but a carer is being alerted", or "wherever children are in danger, carers can always receive information about them". Some features of the state are likely to be directly controlled by the system as in the case of the carers being alerted, while others are observed as in the case of the child's position. It is important to recognise that observed features might be indirectly influenced by the system when instantiating these properties.

Algorithm For each source state the algorithm checks whether it is possible to reach a new state with the desired environment attributes. The algorithm skeleton is presented in Figure 19. Terms (1) and (2) are the identified target states, while terms (3) and (4) are the source states. An instance of the algorithm for verifying the second property above, in a particular scenario, can be found in Figure 20. Term (3) was instantiated with place *ChildInDanger* from the *Movement* module, while term (4) defines the specific child considered in the scenario. Terms (1) and (2) describe a particular carer being alerted. The variable *REACHABILITY* (line 19) contains the states from which a counter-example exists.

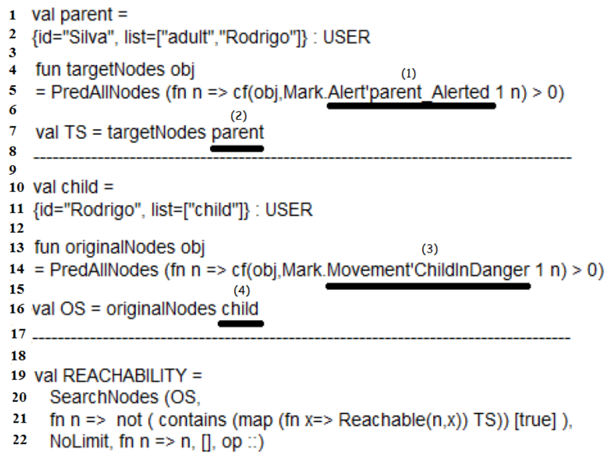
6.2.2. *Making the behaviour model tractable*

Limiting the ranges of values, for each domain of the mode, will have the effect of reducing the size of the model. This will ease analysis. It is important that the range of scenarios is sufficient to ensure that analysis is complete. Some avatars' positions (e.g. "a child is far from the house") in the present case are not relevant, while other positions must be considered (e.g. "a child is near an asthma trigger"). The simulation layer in the APEX tool can be used to create the samplings that make up the scenarios. The



```
fun targetNodes obj
= PredAllNodes (fn n => cf(obj,Mark.MODULE'PLACE 1 n) > 0)

val TS = targetNodes TOKEN
-------------------------------------------------------------------------------

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

val OS = originalNodes TOKEN

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

Figure 19: Reachability property algorithm skeleton



```
val parent =
{id="Silva", list=["adult","Rodrigo"]} : USER

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

val TS = targetNodes parent
-------------------------------------------------------------------------------

val child =
{id="Rodrigo", list=["child"]} : USER

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

val OS = originalNodes child
-------------------------------------------------------------------------------

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

Figure 20: Instance of the reachability property algorithm

open functions, that take values from the simulation or physical layers, are modified to make a random choice from the limited set of values defined by the scenario (see Figure 21). A set of system behaviours can then be constructed as a basis for exhaustive analysis.

The APEXi tool, a component of APEX (see Figure 22) facilitates the selection of values for relevant tokens to make up the scenario. This tool reduces configuration effort prior to analysis. The APEXi tool allows automatic insertion of values to construct the deterministic model. The model can then be evaluated. The selection of adequate values for analysis is an important step that the analyst must consider carefully. The APEXi tool reuses part of the APEX communication/execution component responsible for exchanging information with CPN models. Modules that receive values, sent by the tool, also use functions of the Comms/CPN library [34] that connects CPN Tools with external processes. Figure 23 illustrates how APEXi is connected to the remaining components.

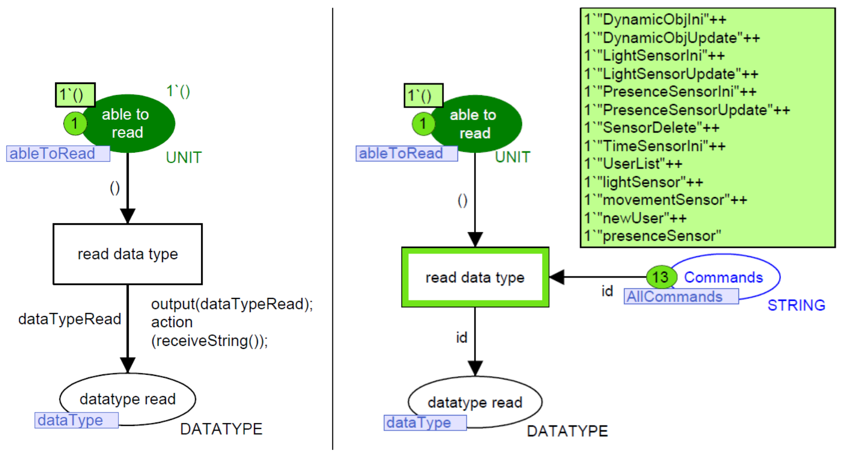


Figure 21: Data type reading - open (left) vs. closed (right)

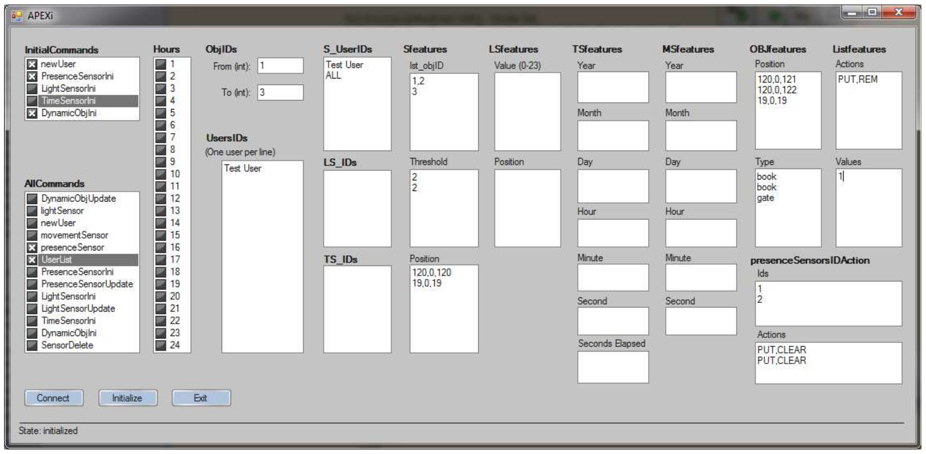
6.2.3. *Example of property instantiation and model reduction*

Property patterns are designed to assist the development of suitable properties. For example, the property "wherever carers go they can always receive information about their child" is verified by instantiating the reachability algorithm skeleton (Figure 19, page 27). The pattern checks whether it is possible to reach one state given another state as a starting point (reachability between two nodes of the reachability graph). This can be expressed in the terms of the template as "for every carer position and every child position a carer alert state can be reached". Figure 20 (page 27) shows the algorithm skeleton associated with this pattern being instantiated. The *targetNodes* and *originalNodes* functions identify the relevant nodes. The places (i.e. *Alert'parent_Alerted* and *Movement'ChildinDanger*) identify the nodes to be used in the analysis. Concrete tokens (i.e. carer and child) are identified in these places (see underlined terms in Figure 20). The execution of the pattern identifies the parent's alerted nodes (returned by the *targetNodes* function). The pattern then requires the identification of all nodes in which the child is in danger (returned by the *originalNodes* function). Finally, the identification of any node from which an alert should have been made and was not are held in the REACHABILITY variable. Such a situation would occur when the system did not reach any carer's alerted node despite the child being in danger.

Checking the property in the example returns no nodes. For the selected scenario it has been demonstrated that wherever carers go they can receive alerts about their child. In other words the property is true.

6.3. *Complementarity of the analyses*

The interpretation of analysis results requires care. A property proved true is a property of the system. This is relevant but it does not guarantee that a design solution is appropriate from a human perspective. For example one of the properties, described above, guarantees that the system will always provide feedback. However the elements of the environment that are assumed to provide feedback may not actually be recognised effectively by the nominated group of users. These issues of salience must be


APEXi

InitialCommands
- [✓] newUser
- [✓] PresenceSensorIni
- [ ] LightSensorIni
- [ ] TimeSensorIni
- [✓] DynamicObjIni

AllCommands
- [ ] DynamicObjUpdate
- [ ] lightSensor
- [ ] newUser
- [ ] movementSensor
- [✓] presenceSensor
- [✓] UserList
- [ ] PresenceSensorIni
- [ ] PresenceSensorUpdate
- [ ] LightSensorIni
- [ ] LightSensorUpdate
- [ ] TimeSensorIni
- [ ] DynamicObjIni
- [ ] SensorDelete

Hours: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24

ObjIDs
From (int): 1
To (int): 3

UsersIDs (One user per line): Test User

S_UserIDs: Test User, ALL

LS_IDs

TS_IDs

Sfeatures
lst_objID: 1,2 / 3
Threshold: 2 / 2
Position: 120,0,120 / 19,0,19

LSfeatures
Value (0-23)
Position

TSfeatures: Year, Month, Day, Hour, Minute, Second, Seconds Elapsed

MSfeatures: Year, Month, Day, Hour, Minute, Second

OBJfeatures
Position: 120,0,121 / 120,0,122 / 19,0,19
Type: book / book / gate

Listfeatures
Actions: PUT,REM
Values: 1

presenceSensorsIDAction
Ids: 1 / 2
Actions: PUT,CLEAR / PUT,CLEAR

State: initialized

Figure 22: APEXi interface

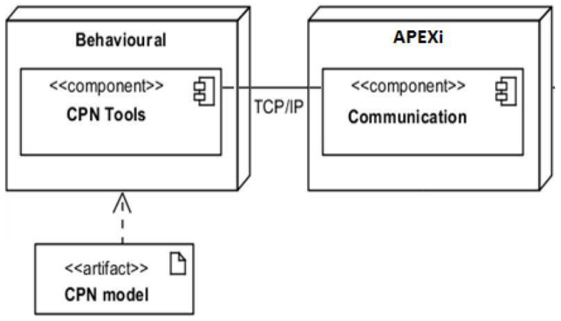


Figure 23: APEXi tool connection to the APEX behavioural component

explored through an evaluation of user experience in the simulation layer. Those tests, however, are not enough to guarantee the correct behaviour of the system. The combination of the two approaches provides the best guarantee that a system's design exhibits appropriate characteristics.

When the property fails the behaviour model can be directly animated with the failed behaviour. This allows selective step by step execution of the CPN models. It also provides a means of seeing how the system reacts in particular situations and enables an exploration of counter-examples generated through exhaustive analysis.

In summary, it has been demonstrated how the APEX framework provides a multi-layered prototyping approach where each layer supports a specific type of evaluation. The analysis provided by these three layers can be used to create analyses that complement each other to provide the full coverage of a proposed solution.

7. EVALUATION OF APEX PROTOTYPING APPROACH

7.1. Introduction to the study

An initial evaluation of APEX was conducted to obtain feedback about its effectiveness in supporting the design and development of ubicomp prototypes. Because APEX is aimed at software engineers and developers within a multi-disciplinary context, we wished to evaluate whether the tools facilitate cross disciplinary evaluation of alternative designs during development. Twenty seven post graduate software engineers at the University of Minho participated in this preliminary study. They were all male and their age varied between 21 and 27 years. The study has a number of features. It introduced participants to CPN and to APEX. On the basis of this introduction they were required to solve a problem using APEX. The problem involved designing and producing a prototype using a provided virtual environment. When they completed the task participants were asked to complete a questionnaire.

Each participant was given a set of instructions about how to use APEX. These instructions described available options and viewer functionalities. A starter model that simulated the aware home was provided so that they had a structure upon which the solution could be built. This consisted of a CPN model and a virtual environment. Each participant was provided with instructions to enable them to configure the virtual environment and to connect and synchronise it with the CPN model. The exercise required students to develop and compare different added functionalities designed to help elderly people find their way to the bathroom at night.

A number of indications and alternative options were offered as hints in the briefing notes. These included:

- putting lights on the floor to be turned on when the person leaves their bed in the dark;
- using a presence sensor to detect whether the person has left their bed;
- using an additional presence sensor to turn the light off when the person has returned to bed.

Functions were provided to facilitate the development of solutions based around these issues. It was indicated to participants that these functions could be invoked from the CPN transitions. Participants were invited first to develop solutions, and then to comment on the adequacy of their solutions, as well as any problems with proposed solutions. After completing the task, participants were invited to modify their designs by adding a facility to turn the light off when the person returns to bed, both by using a presence sensor and a timer.

Participant progress was monitored using a grounded theory approach [35]. This provided a preliminary understanding of how easily the APEX system could be used to produce a prototype. The anticipated development process was grounded in the following phases: CPN interpretation; CPN development; virtual environment configuration and examination of the prototype.

The questionnaire[15] filled in by participants after the exercise addressed five aspects (as defined in the standard USE questionnaire [45]): participant characterization; usefulness; ease of use; ease of learning; and user satisfaction. Subjects were asked to answer on a 7 point Likert scale with values from -3 (strong disagree) to $+3$ (strong agree). The questionnaire included an open question on the framework's strong and weak points, and enabled the participants to make any further comments they wished.

7.2. Results

All participants provided prototypes of possible solutions. The study focus was to evaluate how effectively APEX supported the partipants' development and evaluation activity. As is to be expected, given the constrained design space, all solutions were fairly similar, and close to what was expected. Solutions typically consisted of a set of lights on the floor leading from the bed to the bathroom. The number, position and dimension of the lights varied between the solutions. Some users chose few lights but with larger dimensions while others chose more lights with smaller dimensions. The shapes of the lights used were squares and rectangles. These were chosen in preference to the other shapes that could have been chosen. All participants chose to put the lights on the floor. The reasoning behind this option might have been to avoid disturbing other persons sleeping but this rationale was not provided. Some solutions considered the problem of getting up from either side of the bed by including sensors and lights on both sides. These solutions triggered the turning on of different sets of lights on each side of the bed by independent sensors. In general the sensors were located similarly close to the bed and to the bathroom. Participants successfully identified the limitations of their initial solutions (e.g. the turning off of the light before the person reached the bathroom) and were able to improve them. They did this in a variety of ways, for example by using additional presence sensors and a timer to ensure that the lights were turned on or off as appropriate to the situation. The selected value for the timer was adjusted by experiment in the environment and was finally very similar in all solutions.

The focus of this study was not to consider how well the tool enabled exploration of the design space. This of course is an interesting question and a topic for future consideration. The focus here, however, was on the participants use of the platform.

The observed phases of the participants' development provided an overall understanding of where participants spent their time. While the results were inconclusive in providing clear patterns, they did provide useful insights. These results must be interpreted in the context of the actual example. As might be expected, the CPN modeling phase $(39.9\%, \sigma = 9.5)$ and the virtual environment configuration phase $(30.2\%, \sigma = 9.2)$, being the least familiar, were the most time consuming. Prototype examination $(15.5\%, \sigma = 6.7)$ and CPN interpretation $(14.4\%, \sigma = 4.8)$ were less time consuming. Unsurprisingly, as participants became more familiar with APEX activities, they took less time to complete the tasks. The proportions of time spent in the different phases remained similar however. The students spent between 42 minutes and 1 hour and 28 minutes to create the prototype.

[15]available at http://wiki.di.uminho.pt/twiki/bin/view/Research/APEX/Documentation (last accessed: 23 January 2013)

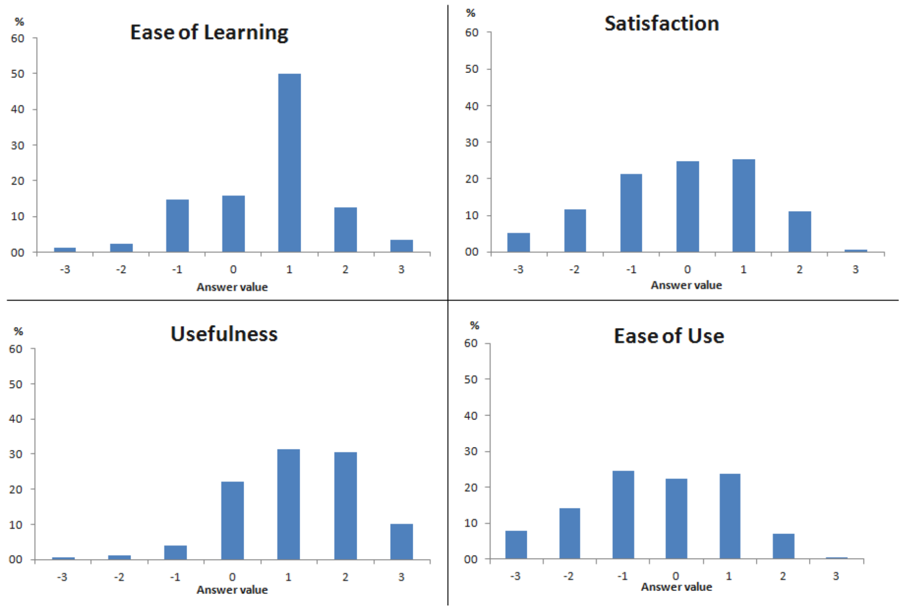


Figure 24: Questionnaire results.

In general, the questionnaires indicated (see Figure 24) a positive reaction to the tool, with all criteria but one obtaining a mode of 1. Participants found it relatively easy to learn how to use the framework. They found that it provided useful features that were easy to memorise. Overall the framework was found to provide results that met their goals in the proposed exercise. The weakest aspect was ease of use, with a mode of -1 (a median of 0). Participants commented that the inherited CPN Tools interface was difficult to use. The moded style of interface [44] differs from other comparable software development and modelling tools. Furthermore the lack of an undo facility in APEX made it difficult to recover from user error. This led to frustration in some cases.

7.3. Analysis

The results were effective in providing early warning of features of APEX that should be improved. They also provide information about how the system is likely to be used by software engineers. Work is being done to improve APEX features particularly in terms of ease of use. A tool is being developed that improves the automation of the development, setting up and deployment of the prototypes.

The results indicated that subject to minor usability improvements the tool is feasible and appropriate for use by developers. The solutions developed were adequate and the framework successfully enabled software engineers to identify issues relating to their solutions and to improve them by analysing the results of the execution pf the solution in the virtual environment. Most importantly, it indicated that APEX can be used by software engineers with no particular relevant skills. This is corroborated by our ongoing experience using the framework.

Further evaluation with end users is being carried out to determine whether generated APEX simulations are adequate to help understand how users would experience a specific proposed ubiquitous computing environment. This evaluation includes a consideration of the extent to which immersion was possible, and a consideration of the extent to which the physical features of the space to be built were recognised through the simulation. It is also clear that different stakeholders will have an interest in the prototypes generated. At this stage it is assumed that users will be the main stakeholders. However it is equally likely for example that the *client* will require a “birds eye view” of the system in order to understand the design concept. These further developments are scheduled for future work.

8. CONCLUSIONS AND FUTURE WORK

The aim of rapid prototyping frameworks should be to make complex system development easier and more efficient. For this purpose these frameworks should be flexible and extensible. APEX supports the development of prototype ubiquitous computing environments. It provides tools to aid the creation of various types of immersive prototypes. It achieves this by using techniques such as stereoscopic 3D and multiple-displays. The exploration of user experience, using alternative ubicomp designs, becomes feasible with APEX. The use of external physical devices also helps immersion by providing a more realistic means of user interaction. How much immersion is enough, and when simpler solutions provide sufficiently similar user experience, are topics that have been discussed by Bowman and McMahan [30].

APEX also offers support for formal verification techniques to reason about the behaviour of the systems. Evaluating the environment behaviour using formal techniques guarantees an exhaustive exploration of possible interactions between the different devices and users in the environment. This is not possible with user testing. Formal analysis cannot however guarantee that a proposed design solution provides an adequate experience. Formal analysis does not guarantee that:

- feedback is salient;
- feedback can be seen by the user;
- feedback will have specific physical characteristics.

These are issues that are better addressed by performing user tests.

The value of the APEX framework is that these broader questions can be addressed cooperatively through the multiple layers. Each layer supports a specific type of evaluation: observation of the behaviour of virtual objects, and user reaction to them within a virtual world (in the simulation layer); analysis of the model (in the modelling layer); observation of real objects (e.g. actual smart phones) connected to the virtual world, and user reaction to them (in the physical layer).

The framework also 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. To summarise, it is possible to explore the design from a variety of perspectives.

The paper has illustrated, through an example, the ability to interchange APEX layers to enrich the exploration of a design, and has demonstrated briefly how different features of a ubiquitous environment are explored in these different modes.

Avatars are controlled by real users through the viewer in the illustrative example. This allows human users to experience the developed ubiquitous environment immersively. An alternative possibility is for an "out of the box" view of the system to be achieved through the use of programmed avatars who are driven by a module that sends the avatars behaviour to the simulation. A path is defined in the model for each avatar. The use of programmed avatars allows created scenarios to be tested or analysed with combinations of programmed and non-programmed avatars. In practice this enables the analysis to be focused on consistencies and accuracy (for example, of the information sent to parents) in the presence of several avatars with reduced costs. This "out of the box" view could be of value in helping other stakeholders understand the implications of a design.

The example also illustrates how APEX enables reasoning, formal modeling and analysis, while at the same time providing valuable feedback about how users experience ubiquitous environments. The results of a preliminary evaluation of the use of the APEX tools show that it is useful and easy to learn. Further developments would make APEX an effective tool for developers.

Further development of the framework are focusing on: tools to automate the setting up and deployment of the prototypes; the connection of isolated sensors that are not integrated into smart phones; a tool within the framework (already in prototype) that enables the semi automatic selection of input values to be used in the test of the prototyped environment; the development of different perspectives of value to other stakeholders.

9. ACKNOWLEDGMENTS

Work carried out in the context of the APEX project, funded by ERDF – European Regional Development Fund – through the COMPETE Programme (operational programme for competitiveness) and by National Funds through the FCT - Fundação para a Ciência e a Tecnologia (Portuguese Foundation for Science and Technology) – within project reference FCOMP-01-0124-FEDER-015095. José Luís Silva was supported by Fundação para a Ciência e Tecnologia (FCT, Portugal) through PhD Grant SFRH/BD/41179/2007. Thanks to GaTech members for collaboration, especially Gregory Abowd for hosting and Mario Romero for having provided the Aware Home 3D model.

References

[1] Silva, J. L., Campos, J., Harrison, M., 2012. "Formal analysis of ubiquitous computing environments through the APEX framework". In Proceedings of the 4th ACM SIGCHI symposium on Engineering interactive computing systems (EICS '12). 131–140. ACM.

[2] Cabana, M. D., Slish, K. K., Lewis, T. C., Brown, R. W., Nan, B., Lin, X., Clark. N.M., 2004. “Parental management of asthma triggers within a child’s environment”, Journal of Allergy and Clinical Immunology, Volume 114, Issue 2, 352–357.

[3] Silva, J. L., Ribeiro, O., Fernandes, J. M., Campos, J. C., Harrison, M. D., 2010. “The APEX framework: prototyping of ubiquitous environments based on Petri nets”, Proceedings of the Third international conference on Human-centred software engineering, HCSE’10, 6–21, 16, Springer-Verlag, Berlin, Heidelberg.

[4] Jensen, K., Christensen, S., Kristensen, L.M., 2006.“CPN Tools State Space Manual”. University of Aarhus.

[5] IEEE Pervasive Computing. Special issue on rapid prototyping. (2005). Volume 4, Issue 4.

[6] Abowd, G. D., Hayes, G. R., Iachello, G., Kientz, J. A., Patel, S. N., Stevens, M. M., Truong, K. N. 2005.“Prototypes and Paratypes: Designing Mobile and Ubiquitous Computing Applications”, IEEE Pervasive Computing, 4(4),67–73.

[7] Jensen, K., Kristensen, L.M., Wells, L., 2007. “Coloured Petri Nets and CPN Tools for modelling and validation of concurrent systems”. International Journal on Software Tools for Technology Transfer (STTT), (9)3-4, 213–254.

[8] Barton, J.J., Vijayaraghavan. 2003. “UBIWISE, a simulator for ubiquitous computing systems design”. Tech. Rep. HPL-2003-93, HP Laboratories, Palo Alto.

[9] Tamai, M., Nishigaki, K., Kitani, T., Shibata, N., Yasumoto, K., Ito, M., Nishikawa, H., Yamamoto, S. 2006. “Ubireal: Realistic smartspace simulator for systematic testing”. Ubiquitous Computing , 459–476.

[10] Nazari, S., Klar. A., 2005. “3DSim: Rapid Prototyping Ambient Intelligence”. SOcEUSAI conference. 303–307.

[11] Disz, T.L., Papka, M.E., Stevens, R., 1997. “UbiWorld: An Environment Integrating Virtual Reality, Super-computing, and Design”. In Proceedings 6th Heterogeneous Computing Workshop, 46–57.

[12] Li, Y., Hong, J., Landay, J. 2004. “Topiary: a tool for prototyping location-enhanced applications”. In Proceedings of the 17th annual ACM symposium on User interface software and technology. 217–226. ACM.

[13] O’Neill, E., Lewis, D., Conlan, O. 2009. “A simulation-based approach to highly iterative prototyping of ubiquitous computing systems”. In 2nd International Conference on Simulation Tools and Techniques. 1–10.

[14] Irawati, S., Ahn, S., Kim, J., Ko. H., 2008. “VARU Framework: Enabling Rapid Prototyping of VR, AR and Ubiquitous Applications”. Virtual Reality Conference, 201–208. IEEE.

[15] Gallasch, G., Kristensen, L.M., 2001. “Comms/CPN: A Communication Infrastructure for External Communication with Design/CPN”. In Proceedings of Third Workshop and Tutorial on CPNs and CPN Tools, 79–93. Department of Computer Science, University of Aarhus.

[16] ISO DIS 9241-210:2008 [3] standard.

[17] Moreira, R., 2011. Master Thesis: “Integrating a 3D application server with a CAVE”. University of Minho. http://wiki.di.uminho.pt/twiki/bin/view/Research/APEX/Publications (last accessed - January 21st 2012).

[18] Bateman, E.D., Hurd, S.S., Barnes, P.J., Bousquet, J., Drazen, J.M., FitzGerald, M., Gibson, P., Ohta, K., O’Byrne, P., Pedersen, S.E., Pizzichini, E., Sullivan, S.D., Wenzel, S.E., Zar, H.J., 2008. “Global strategy for asthma management and prevention: GINA executive summary”. The European respiratory journal : official journal of the Euro-pean Society for Clinical Respiratory Physiology, vol. 31, no. 1, 143–78.

[19] Hong, H., Jeong, H. Y., Arriaga R. I., Abowd, G. D., 2010. “TriggerHunter: designing an educational game for families with asthmatic children”. Proceedings of the 28th of the international conference extended abstracts on Human factors in computing systems, 3577–3582.

[20] Hartmann, B., Klemmer, S. R., Bernstein, M., Abdulla, L., Burr, B., Robinson-Mosher, A., Gee, J., 2006. “Reflective physical prototyping through integrated design, test, and analysis”. Proceedings of the 19th annual ACM symposium on User interface software and technology, 299–308.

[21] Abowd, G., Hayes, G., Iachello, G., Kientz, J., Patel, S., Stevens, M., Truong, K., 2005. “Prototypes and paratypes: designing mobile and ubiquitous computing applications”. IEEE Pervasive Computing 4(4), 67–73.

[22] Scholtz J. and Consolvo, S., 2004. “Toward a framework for evaluating ubiquitous computing applications”. Pervasive Computing, IEEE, vol. 3, no. 2, 82–88.

[23] Li, Y. and Landay, J. A., 2008. “Into the wild: low-cost ubicomp prototype testing”. Computer, vol. 41, no. 6, 94–97.

[24] ONeill, E., 2004. Master Thesis: “TATUS a Ubiquitous Computing Simulator”. University of Dublin. http://www.tara.tcd.ie/handle/2262/827 ((last accessed - January 21st).

[25] Vanacken, L., De Boeck, J., Raymaekers, C., Coninx, K., 2008. “Designing context-aware multimodal virtual environments”. Proceedings of the 10th international conference on Multimodal interfaces, 129–136.

[26] Nielsen, J., 1994. “Enhancing the explanatory power of usability heuristics”. Proceedings of the SIGCHI conference on Human factors in computing systems: celebrating inter-dependence, 152–158.

[27] Ruksenas, R., Back, J., Curzon, P., Blandford, A., 2007. “Formal modelling of salience and cognitive load”. Proceedings 2nd Int. Workshop on Formal Methods for Interactive Systems, 57–75.

[28] Kim, S. and Kim, S., “Usability challenges in ubicomp environment”. Proceeding of International Ergonomics Association, 4–4.

[29] Mankoff, J., Dey, A., Hsieh, G., Kientz, J., 2003. “Heuristic evaluation of ambient displays”. Proceedings of the SIGCHI conference on Human factors in computing systems, no. 5, 169–176.

[30] Bowman, D., McMahan, R., 2007. “Virtual reality: how much immersion is enough?”. Computer, vol. 40, no. 7, 36–43.

[31] Dwyer, M. B., Avrunin, G. S., Corbett, J. C., 1999. “Patterns in property specifications for finite-state verification”. Proceedings of the 21st international conference on Software engineering, 411–420.

[32] Campos, J. and Harrison, M., 2008. “Systematic analysis of control panel interfaces using formal tools”. Interactive Systems Design, Specification, and Verification, 72–85.

[33] Campos, J. and Harrison, M. D., 2009. “Interaction engineering using the IVY tool”. Proceedings of the 1st ACM SIGCHI symposium on Engineering interactive computing systems, 35–44.

[34] Gallasch, G., and Kristensen, L. M., 2001. “Comms/CPN: A communication infrastructure for external communication with design/CPN” in Kurt Jensen (Ed.): 3rd Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools, 75–90.

[35] Adams, A., Lunt, P., Cairns, P., 2008. “A qualitative approach to HCI research”. Research Methods for Human-Computer Interaction, cup, P. Cairns and A. L. Cox, 138–157.

[36] Silva, J. L., Campos, J. C., and Harrison, M. D., 2009. “An infrastructure for experience centered agile prototyping of ambient intelligence”. In Proceedings of the 1st ACM SIGCHI symposium on Engineering interactive computing systems. ACM, New York, NY, USA, 79–84.

[37] Basuki, T., Cerone, A., Griesmayer, A., and Schlatte, R., 2009. “Model-checking user behaviour using interacting components”. Formal Aspects of Computing, vol. 21, 571–588.

[38] Westergaard, M. and Lassen, K., 2006. “The britney suite animation tool”, in Proceedings of the 27th international conference on Applications and Theory of Petri Nets and Other Models of Concurrency, 431–440.

[39] Westergaard, M. and Verbeek H.M.W., 2011. “Efficient Implementation of Prioritized Transitions for High-level Petri Nets”. Proceedings of the International Workshop on Petri Nets and Software Engineering (PNSE11). Vol. 723 of CEUR Workshop Proceedings, 27–41.

[40] Hoare, C., 2004. “Communicating sequential processes”. Prentice Hall International. p.260.

[41] Massink, M., Duke, D., and Smith S., 1999. “Towards hybrid interface specification for virtual environments” in Design, Specification and Verification of Interactive Systems. Vol. 99, 3051.

[42] Dubois, E., Gray, P., and Nigay, L., 2002. “ASUR ++ : A Design Notation for Mobile Mixed Systems” in Proceedings of the 4th International Symposium on Mobile Human-Computer Interaction, 123–139.

[43] Navarre, D., Palanque, P., Bastide, R., Schyn, A., Winckler, M., Nedel, L., Freitas, C., 2005.“A formal description of multimodal interaction techniques for immersive virtual reality applications” in Proceedings of the 2005 IFIP TC13 international conference on Human-Computer Interaction, 170–183.

[44] Ratzer, A., Wells, L. , Lassen, H. , Laursen, M., Qvortrup, J., Stissing, M., Westergaard, M., Christensen, S., Jensen, K., 2003. “CPN tools for editing, simulating, and analysing coloured Petri nets” in Proceedings of the 24th international conference on Applications and theory of Petri nets, 450–462.

[45] Lund, A.M., 2001. “Measuring Usability with the USE Questionnaire”. STC Usability SIG Newsletter, 8:2.