The APEX framework: prototyping of ubiquitous environments based on Petri nets

José Luís Silva[1,⋆], Óscar R. Ribeiro[1], João M. Fernandes[1], José Creissac Campos[1], and Michael D. Harrison[2]

[1] Dep. Informática / CCTC, Universidade do Minho, Braga, Portugal
{jlsilva, orribeiro, jmf, jose.campos}@di.uminho.pt
[2] Newcastle University, United Kingdom
michael.harrison@ncl.ac.uk

Abstract. The user experience of ubiquitous environments is a determining factor in their success. The characteristics of such systems must be explored as early as possible to anticipate potential user problems, and to reduce the cost of redesign. However, the development of early prototypes to be evaluated in the target environment can be disruptive to the ongoing system and therefore unacceptable. This paper reports on an ongoing effort to explore how model-based rapid prototyping of ubiquitous environments might be used to avoid actual deployment while still enabling users to interact with a representation of the system. The paper describes APEX, a framework that brings together an existing 3D Application Server with CPN Tools. APEX-based prototypes enable users to navigate a virtual world simulation of the envisaged ubiquitous environment. The APEX architecture and the proposed CPN-based modelling approach are described. An example illustrates their use.

1 Introduction

Ubiquitous computing poses new challenges for designers and developers of interactive systems. Because these systems *immerse* their users, the effect they have on the users' experience is an important element contributing to the success of a design. Technology enhancement has the potential to have a profound impact on a built environment transforming a sterile space into a place that is in harmony with its purpose. The experience of checking into an airport can be improved by providing information to travellers when and where they need it. Frustrating delays could thereby be removed through the appropriate use of personalised information. The experience of using a library could be improved by providing personal and clear information about the location of the shelf in a large library where the required book is located. Experience therefore becomes an additional interactive characteristic of ubiquitous systems, to be explored in addition to more traditional notions of usability.

⋆ José Luís Silva is supported by *Fundação para a Ciência e Tecnologia* (FCT, Portugal) through PhD Grant SFRH/BD/41179/2007.

Experience is difficult to specify as a requirement that can be calculated and demonstrated of a system. It is difficult to measure and to obtain early feedback about whether a design will have the required effect. Currently, there are no techniques that can be used to analyse specifications against different notions of experience (for a discussion, see [9]). An important barrier is the difficulty of developing prototypes that could feasibly be used to explore issues of experience.

This paper limits attention to *ubiquitous environments* envisaged as enhancing physical environments. In the envisaged designs, "spaces" are augmented with sensors, public displays and personal devices. Of particular interest in these systems is the way that the user interacts with the environment, as a result of both explicit interaction with the system, and implicit interactions that arise through changes of *context*. Here context could include location, or the steps that have to be taken by a user to achieve some goal (for example check-in, baggage screening, passport control, boarding card scanning).

The paper describes how prototypes can be built to represent the interaction between users, devices and services, as users move within ubiquitous environments. To avoid unnecessary development cost, early designs are explored in this proposal through model-based prototypes explored within a virtual environment. The paper describes a prototyping framework (APEX) that uses Coloured Petri Net (CPN) [11] models. APEX binds a CPN model to a 3D application server (OpenSimulator[3]).

The Petri nets modelling language, being an expressive and graphically informative notation, allows the description of the envisaged design. OpenSimulator provides support for exploring the design based on the Petri net description. Their integration thus allows rapid prototyping of ubiquitous environments, enabling users to navigate a virtual world simulation of the environment to evaluate usability issues, including user experience.

This paper builds on [18]. There, the early concept of the APEX framework was discussed, and some initial results presented. Since then, the framework has been developed, and the modelling approach fully revised. The new models present a number of benefits, including better scalability and support for heterogeneity. The current paper describes the APEX architecture, the new modelling approach, and provides modelling guidelines for developing prototypes.

The structure of the paper is as follows. Section 2 discusses related literature and the goals of the project. Section 3 describes the architecture of APEX. Use of the framework is illustrated by means of a smart library which senses the presence of users, and guides them to the shelves where their required books are located. Section 4 describes how the example is modelled. Section 5 describes usage of the framework. Section 6 presents conclusions and future work.

2 Related literature and goals

Despite considerable advances in the development of ubiquitous systems, there continues to be a tendency (see [5] for a concise overview) for the development

[3] http://opensimulator.org (last accessed June 14, 2010)

and evaluation of ubiquitous systems to be focussed on experimental systems, usually prototype device designs within partial systems. The issue of how to evaluate whole systems in real contexts continues to be a concern, see [2] for a useful discussion of this contrast. Another important aspect of evaluation is how to explore the user experience that a designed system creates. In this respect there is a substantial literature taken from design disciplines, see for example [4]. In design, for example, a typical approach is to use non-functional (for example, clay) prototypes as objects which potential users are asked to carry around in the contexts where the actual system is to be used in order to obtain information about how the proposed design might be experienced. One particularity of the type of systems of interest is that the system is woven into the context, making it harder to prototype.

APEX is designed to satisfy three requirements. The first is that it should enable the rapid development of both prototypes and target systems. While there are several existing platforms for ubiquitous computing ([3, 8, 10] are examples), a software tool is required that facilitates the development of prototypes, while simultaneously providing the hooks for the target system.

The second requirement is that a 3D environment can be used to construct simulations that can be explored realistically by users. 3D Application Servers, such as SecondLife™[4] or OpenSimulator, provide a fast track to developing virtual worlds. OpenSimulator, in particular, has the advantage of being open source, which means that the backend can be programmed allowing configurability and extensibility.

Systems such as Topiary [13] enable users to explore prototypes of context-aware application in real world settings. They resort to Wizard of Oz techniques to avoid the actual deployment of sensors. They are targeted to the prototyping of applications running on user devices, and do not support the *enhancement* of the physical space. A different class of systems, such as 3DSim [17], UbiWorld [6] or the work of O'Neill et al. [16], have similar visions to ours (developing simulations of the actual environments).

The third requirement is an approach to modelling ubiquitous computing. While 3DSim and UbiWord envisage the use of programming languages to build the prototypes, we are interested in creating them from models of envisaged systems. A benefit of this approach is the integration of the modelling approach with analytical approaches, to provide leverage on properties of ubiquitous environments that are relevant to their use.

Petri nets constitute an expressive and graphically informative modelling language that has been used to describe virtual environments. Previous modelling approaches based on Petri nets include the use of: Hybrid high-level Nets (HyNets) [14], Flownets [19], Interactive Cooperative Objects (ICO) [15], and Coloured Petri Nets (CPN) [11].

CPN modelling and analysis is supported by CPN Tools, enabling analysis either by simulation (similar to program execution) or by more formal analysis (state space analysis and invariant analysis). Simulation can be used to animate

[4] http://secondlife.com (last accessed June 14, 2010)

the models. State space analysis can be used to check standard properties, such as reachability, boundedness, liveness properties and fairness, as well as specific properties defined using the associated programming language (CPN ML language [12]).

In summary then, given the objectives set forth for APEX, CPN was chosen because: (i) it allows rapid development of prototypes, much faster than equivalent conventional approaches using C#; (ii) it allows analysis of properties of the model (via CPN Tools); (iii) the animation capabilities of CPN Tools allow control of the virtual world simulations directly from the models (hence, the behaviour modelled is exactly what is executed — this improves on current approaches in that in these approaches when a simulation needs to be programmed, what is executed does not necessarily reflect the models and specifications produced in an earlier development stage).

While several approaches aiming at ubiquitous computing prototyping were identified above, they are mostly focused on helping ubiquitous system designers to identify unwanted behaviour in their system, and to support informed decision making in an iterative design cycle. APEX is more focused on the *experience* users will have of the design, and in the use of tools to enable analysis.

The above mentioned approach of O'Neill et al. [16] is the most similar to ours, using models and a 3D simulation for the prototyping of ubiquitous environments. In their case a games engine is used. We believe the use of a 3D application server (OpenSimulator) has some advantages compared with a games engine. It supports the creation of virtual environments in real time using world building tools, and it is easily extendable by the loading of modules. In the case of the games' engine, the environment must be previously fully created using a map editor. Using a 3D application server means the approach is flexible. A variety of clients, customizable in appearance, can access the virtual world on multiple protocols at the same time, and in world application development using a number of different languages is also possible.

3 The APEX framework

The overall architectural view of the APEX framework is presented in Figure 1. Three main components are identified:

- a *virtual environment component*, responsible for managing the physical appearance and layout of the prototype, including managing the 3D simulation and the construction of the virtual environment;
- a *behavioural component*, responsible for managing the behaviour of the prototype, including the description, analysis and validation of the virtual environment's behaviour;
- a *communication/execution component*, responsible for the data exchange among all components and for the execution of the simulation.

OpenSimulator enables the interactive creation of virtual environments. It provides a sufficiently rich *texture* to enable users to visualise the physical char-

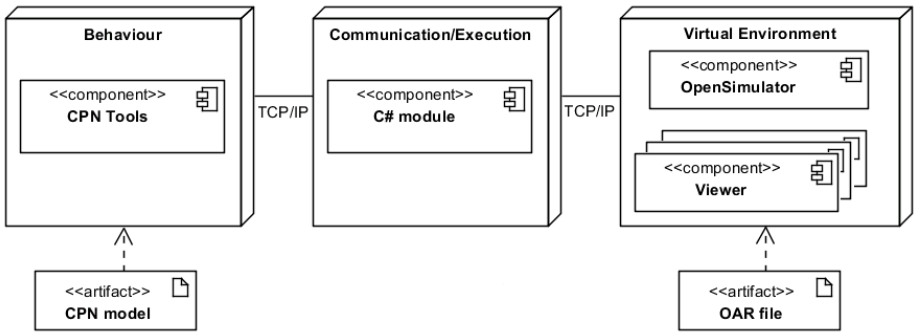

Fig. 1. Logical architecture of the APEX framework

acteristics of the real system. A rich palette of features provides for easy object/environment creation and manipulation. These objects, together with the insertion and manipulation of textures, lighting, animation and sounds also provided, enable a simulation which can create a realistic visualisation of the proposed real system. Pre-defined environments and devices can be used in this creation process.

To create a prototype, besides creating the virtual environment, the developer needs to extend the CPN base model provided. APEX uses CPN Tools to model the behaviour of the virtual environment. Models of each type of dynamic object/device in the environment (e.g., sensors, displays, personal devices) need to be inserted into the global model of the environment. Adequate models must either be available or must be created using CPN Tools. Section 4.2 will provide a more detailed description of how that can be done.

Once the CPN model and the environment are created a component of the framework binds them together. To achieve this, transitions in the CPN link the behaviour described by the models to the respective objects in the environment.

Several users can be connected to the simulation using different viewpoints onto the OpenSimulator server. Users can navigate and interact with the virtual world simulation of the envisaged ubiquitous environment, enabling the evaluation of usability and experience issues with the proposed design.

3.1 Behavioural component

This component is responsible for driving the simulation using the information from the model, and to send relevant data to the virtual environment. It contains the CPN tools, which use CPN models to describe the behaviour of the virtual environment in response to user actions and context changes.

A generic CPN base model is provided from which virtual environment models can be derived. The aim in developing this base model was to develop a generic style of CPN relevant to the modelling of virtual environments, including models that can be instantiated to the physical space in which the system

is to be defined to operate. The model consists of: (a) a module to initialise the simulation, and to establish the connection between the CPN model, as represented by CPN Tools, and OpenSimulator; (b) a module that receives user data (for example user identity and position) from OpenSimulator when a user moves and uses it to update appropriate tokens; (c) modules describing the behaviour of each device in the system. An example is presented in Section 4.

3.2 Virtual environment component

This component sends information about the simulation (e.g. user position) to the behavioural component which takes a *decision* and sends indications to reflect these changes in the simulation. It contains the OpenSimulator server and viewers for each client who connect to it.

The OpenSimulator server is responsible for maintaining the virtual environment information available to viewers. The features of the 3D simulation include location, the viewing aspect and the physics of each of the objects in the environment. Pre-defined environments and objects can be saved/loaded in/from Opensim ARchive files (OAR). All the different entities (object, terrain, textures, etc.) are packaged in these files in the format used by Opensimulator to keep data within an archive. The server enables the connection of several users from, possibly, different locations to the same virtual environment via the web through appropriate viewers.

Viewers interact with the server and are used to define features of the 3D simulation presented to users, and to allow users to navigate and interact within the simulated environment. Interaction is achieved both explicitly by a user using (virtual) devices, and implicitly through changes of context. Possible viewers include the Hippo OpenSim Viewer[5] or the Linden Lab's Second Life viewer[6]. However, a number of alternative compatible viewers exist[7]. Note that, currently, some of these alternative viewers only enable the environment exploration without providing any modelling tool.

The behaviour described in the previous section is linked to the objects which are identified by unique names. For instance, to open a gate in the simulation, the CPN model of the gates must indicate in its open transition code the identifier of the gate to open. Objects identifiers are easily accessible through the properties panel provided by the viewer and associated to each object of the environment.

3.3 Communication/execution component

This component is a DLL (dynamic-link library) responsible for loading the simulated ubiquitous environment into the OpenSimulator server, and for using the CPN models to drive it. It is positioned between the two other components managing the exchange of information between them.

[5] http://mjm-labs.com/viewer/ (last accessed June 14, 2010)
[6] http://secondlife.com/support/downloads (last accessed June 14, 2010)
[7] http://opensimulator.org/wiki/Connecting (last accessed June 14, 2010)

Communication in the CPN models is achieved through Comms/CPN [7], a CPN ML library for connecting between CPN Tools and external processes, provided with the CPN Tools. The BRITNeY Suite [20] also enables the communication between CPN models and a Java-based animation package. Comms/CPN is more adequate and simple to use for our case. Unlike Comms/CPN the BRITNeY Suite has a more general purpose, providing more features besides the communication package, which make it more complex to use.

In order to use Comms/CPN a module must be loaded into the external process. Java and C modules are available with the distribution. However, OpenSimulator modules (DLLs) are developed in C#. No alternatives were found for this communication so a new C#/CPN communication package has been developed. With this development the communication of the CPN models, using the Comms/CPN functions, and C# processes becomes possible.

The developed module sends information to CPN Tools when changes in the environment happen, and is responsible for changing the environment in response to data sent by CPN Tools. Additionally, it handles the loading/saving of OpenSimulator objects/environments and the execution of commands invoked by the user in the viewer. When inserted in the OpenSimulator server location, this DLL is automatically loaded by the OpenSimulator. After the establishment of the communication between the CPN model and the simulator, by the evocation of a function in the CPN model (explained in the next section), the APEX is ready to use.

4 Modelling with CPNs (The example)

As previously stated, a generic CPN modelling approach was developed to enable the easy creation of new ubiquitous systems prototypes. In this section the modules of this approach are described. Figure 2 presents the setup model. As will be discussed, this model needs small modifications only when being adapted to different applications. The model in figure 3 deals with user position and is generic. The developer then needs to develop a module for each device type present in the ubiquitous system. Figure 4 presents the module for a specific type of object present in the example used (a gate). How these modules are created will be described in section 4.5

4.1 The example

The example used to illustrate the system is a smart library. Books are identified by RFID tags and are stored on bookshelves. Screens are used to provide information to library users. A registred library user is allowed entry/exit via gates. When a registered user arrives at the entry gate, a screen displays which books have been requested by the user (e.g., earlier via a web interface) and opens the entry gate. The system guides the user to the required books through the use of sensors that recognise the user's position in real-time. As the user approaches the book's location a light with a specific colour is turned on. Hence several users

looking for books in nearby locations can distinguish their own request. When the book is removed, the light on the book is turned off. As the user returns to the exit gate a personalised list of requested and returned books is displayed on a screen by the gate which is opened so that the user can leave.

4.2 Modelling approach

There are a number of styles of specification that can be achieved using CPN. These styles vary according to the extent to which the semantics of the underlying objects are made explicit in the structure of the CPN specification, or encoded into the tokens. The following two extremes are possible:

- placing all the semantics in the tokens, in other words, minimising the number of places in the net;
- using places to characterize each different relevant situation (user action, context change, etc.), thereby adding transitions that explicitly describe aspects of the semantics of the objects.

A small example is presented to clarify these two approaches. Suppose a device which can be in two different states (*on* and *off*) is to be modelled. Following the two approaches above, two different results will be reached. In the first, the model will consist of only one place, and one transition from and to this place. The place will hold tokens with a semantics which can represent all the different states of the device. The state of the device will be encoded as an attribute (a colour) of the token representing the device. The transition will be responsible for changing the colour of the token, reflecting the new state of the device. In this situation all the meaning is in the value of the tokens.

Following the second approach, the model will be represented by two places each representing a possible state of the device, and by transitions between them (two in this case). No semantics will be carried by the token, all the meaning will be represented by the structure of the model. The state of the device is known by looking to the position of the token, i.e. at the place which holds the token.

In APEX, a mixed approach is used where the states of the dynamic objects (open, closed, etc.) are modelled as places and user actions and context changes modelled as transitions. Each device and user is represented in the CPN model as a token in the respective place. Each of these tokens has an identifier which is used as the identifier of the objects present in the simulation.

The users and object features (e.g. identifier, position) are modelled as attributes in their respective tokens. These values are used by CPN ML functions together with instructions (e.g. *open*, *close*) to indicate changes that must be reflected in OpenSimulator. Section 4.5 will provide a description of how this is done. The guards on the transitions as well as the functions associated with transitions are responsible for part of the behaviour of the system. Both of these are modelled in the CPN ML language, so this behaviour is modelled functionally.

This combination gives more expressiveness to the ubiquitous systems modelling while avoiding clutter in the CPN specification. In the next sub sections, the approach will be illustrated using the example.

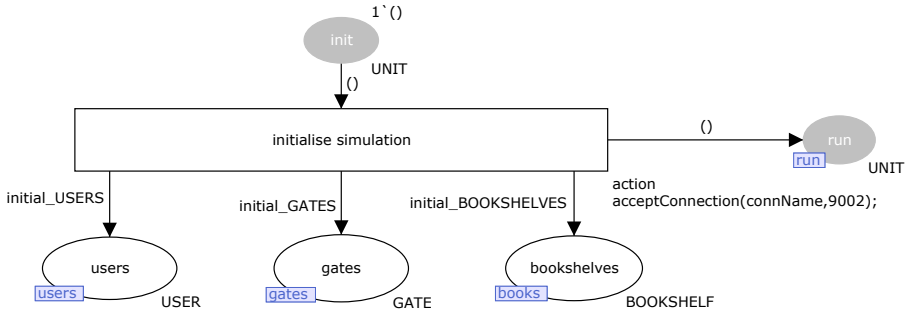

Fig. 2. The CPN module to setup the library simulation.

4.3 Setting up the simulation

The initial conditions of the simulation are defined in the CPN module shown in Figure 2. Firing the "`initialise simulation`" transition sets the initial configuration of the simulation, and executes the associated CPN ML code. For example, "`acceptConnection(connName,9002)`" is a function of the Comms/CPN library used to establish the connection between CPN Tools and OpenSimulator.

In this case the configuration includes three places: "`users`", "`gates`" and "`bookshelves`". Fusion tags (inset into the lower left corner of the places) enable instances of these places to appear in other parts of the CPN model. Hence, these places are called *fusion places*. The utilization of these places will be described in section 4.5.

Annotations at the bottom right side of the places indicate the type of token each place can hold. Place "`users`" holds "`USER`" tokens representing information about users in the virtual environments. This particular place is mandatory, since whatever the model the handling of users must be supported. The remaining places ("`gates`" and "`bookshelves`") hold tokens representing devices. These places are system dependent and will vary for each prototype. The colour (structure) of the tokens which these places can hold is defined in CPN Tools, and characterises the information held in the model for each type of device.

Besides establishing the connection between the CPN tools and OpenSimulator, and initializing user and device places, the "`initialise simulation`" uses two places to control the execution of the CPN model: "`init`" to limit execution of the transition to one occurrence, and "`run`" to inform other CPN modules that the simulation is running.

4.4 Reading users' positions

Figure 3 presents the CPN module that collects users' data from the OpenSimulator. Transition "`read user id`" reads a user identifier sent by the OpenSimulator server (c.f., "`receiveString()`" function on the code block associated with the transition). A token with the value of the read user identifier is in-

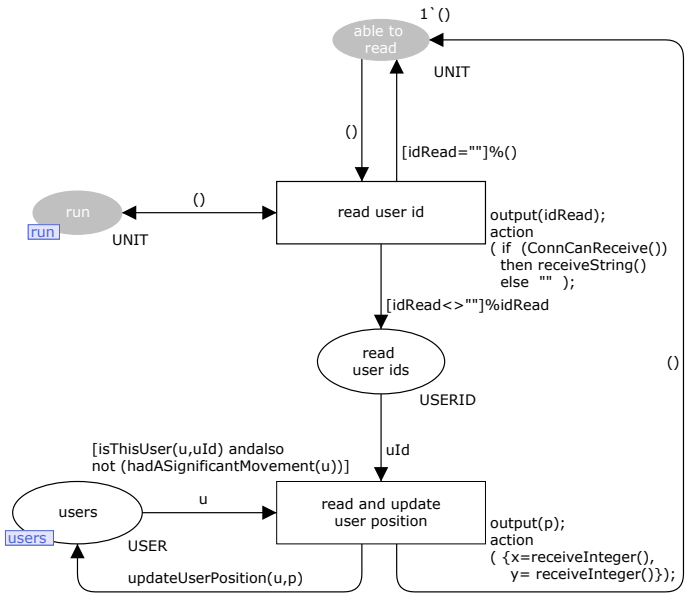

Fig. 3. The CPN module for acquiring users' data.

troduced in the "`read user ids`" place (c.f., "`output(idRead)`"). This is used to read the new position by means of the transition "`read and update user position`", which also updates the relevant user token (taken from the user's fusion place). The new coordinates x and y are read in the *action* part of the transition using the function "`receiveInteger()`" and a pair of coordinates (p) is produced. This pair is then used to update the user position through the "`updateUserPosition`" function. The expression "`isThisUser(u,uId)`", in the guard of this transition, guarantees that the user token which is updated corresponds to the previously read identifier. In this model the number of users remains constant during each simulation session. To add more users to the system one must add the corresponding tokens in the place "`users`". The automatic addition and deletion of users at runtime, in accord with the users connected to the simulation, is planned. This will be achieved via the addition of new models to generate user tokens, and by enhancements to the C# module.

CPN modules for reading the user's position, and for managing devices' behaviour execute concurrently. Precedence of devices' transitions over data acquisition transitions is guaranteed through the guard "`not (hadASignificant-Movement(u))`" on the transition "`read and update user position`". Movement of a user is significant (for a device) when the new position is "near" the device. Hence, if a user is near a device, no new data will be acquired until the device has processed the current data.

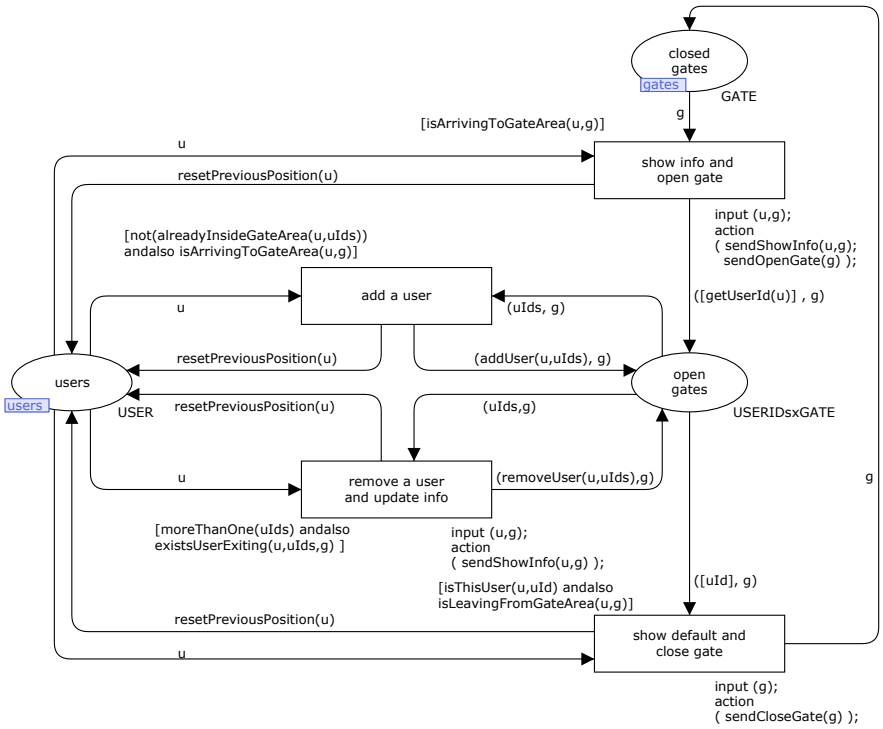

Fig. 4. The CPN module for a entry gate device.

4.5 Modelling the devices of the system

Each device type in the ubiquitous environment simulation needs a corresponding CPN module describing its behaviour. It is envisaged that a library of models will be made available for supported devices. When new (unsupported) devices are to be used, a new model must be developed and added to the library. This section explains the process through the example of the entry gate.

Device behaviour is modelled through a combination of fusion places, normal places, transitions, functions (described in the CPN ML language) and conditions. State transitions play an important role in this process since it is through them that the connection between the model and the simulation is accomplished, via the associated CPN ML functions. These functions are also responsible for describing functional behaviour not structurally expressed by the net.

Fusion places are the basis for the creation of these behavioral modules. They establish the link between the models of the devices and the setup model presented in figure 2. The device model for the entry gate model is presented in Figure 4. In this example the users and gates fusion places hold the (user and gate) tokens needed to model the behaviour.

The entry gate is equipped with a sensor to capture a user approaching it. Transition "`show info and open gate`" represents the actions of the entry gate. It displays requested books on the screen and opens the entry gate. Functions "`sendShowInfo`" and "`sendOpenGate`" are responsible for these actions sending relevant instructions to OpenSimulator. These actions occur when the gate's sensor detects a registered user arriving at the entry gate (modelled by "`isArrivingToGateArea(u,g)`" evaluating to true). When the gate is open and another registered user enters the gate area the transition "`add a user`" occurs and this user is included in the set of users that are near the gate. This set of users is represented in the tokens held by the place "`open gates`". As already stated, each place has an associated token type which it holds. In this case the type of this place is "`USERIDsxGATE`". It means that each token is a product of a set of user IDs ("`USERIDs`") and the gate ("`GATE`") which they are near.

When the transition "`show default and close gate`" is taken, default information is displayed on the screen and the gate is closed. For this to happen a user must have moved away from the gate, and there should be no more users near it. If other users are near the gate, the transition "`remove a user and update info`" removes the designated user from the list of users who are near the gate (function "`removeUser`") and, if that user's information was being displayed, the information currently on the screen is changed to one of the other user's (function "`sendShowInfo`").

As explained, the different CPN models are connected via *fusion places*, enabling token flow between them (e.g. the models in figures 3 and 4 are connected to the setup model via the "`users`", "`gates`" and "`run`" *fusion places*). Put together they form the model of the envisaged ubiquitous system.

The focus of this paper had been the architecture of the framework, and the CPN-based modelling. Of course, a virtual environment to match the model must also be developed. This is done through the virtual world viewer. Once both models and virtual world simulations for all devices are in place, animation of the envisaged ubiquitous system can start.

5 Support for design

As stated in Section 1, APEX supports both the design and the analysis of ubiquitous systems. The developer creates the CPN model, as illustrated in Section 4. Depending on the new types of devices that are used the developer is required to modify a small piece of the Communication/Execution C# module responsible for reflecting the changes in the simulation. The code responds to changes in objects of the environment consistent with the state of the CPN model. As an example, Figure 5 is a code snippet that searches for objects where changes are directed to occur by the CPN model and makes the changes. In this snippet an *open* or *close* action is received and the position of the gate is changed accordingly.

A typical runtime configuration of the framework (see figure 6) will involve deploying the OpenSimulator server, CPN tools, and the Communica-

```
foreach (KeyValuePair<Scene, List<SceneObjectGroup>> kvp in HelloWorldModule.scene_prims)
 {
         foreach (SceneObjectGroup sog in kvp.Value)
         {

                 //Object: gate

                 if (sog.Name.Equals("gate"))
                 {
                     if (action.Equals("open"))        //Open the gate
                         sog.AbsolutePosition = new Vector3(130.7f,130.9f,25.9f)
                     else                              //Close the gate
                          if (action.Equals("close"))
                             sog.AbsolutePosition = new Vector3(128.6f,130.9f,25.9f)

                 }

                 //Object: screen

                 …
         }
 }
```

Fig. 5. OpenSimulator objects behavior code

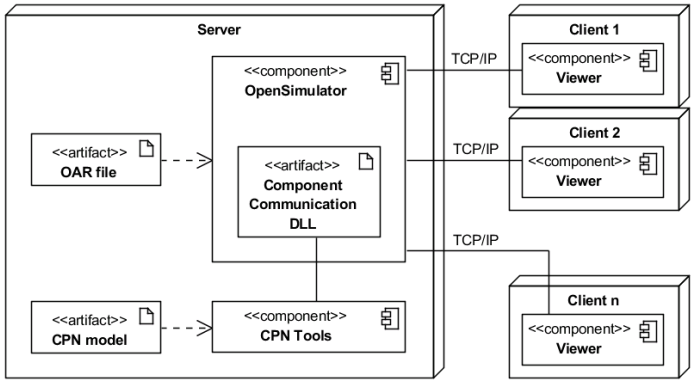

Fig. 6. Physical architecture of the APEX framework

tion/Execution module on a server. Once the CPN model is loaded, the server is ready to allow free exploration and interaction with the virtual environment. At this point, exploration and interaction with the virtual environment is possible. Currently this is achieved by means of viewers deployed on client machines. It is envisaged that higher fidelity prototypes will be possible, for example, using a CAVE system.

Using these prototypes, it becomes possible to test different design alternatives with real users, without the cost of developing the actual system. As an illustration, figure 7 shows a user collecting a book. As the (registered) user approaches the gate (step a) the gate opens and the user is able to enter the library (step b). Once the user is close to the book, the light on the book is turned on so that the user can quickly identify it (step c).

Validating the usefulness of these prototypes in assessing users' experience of the envisaged systems will be the subject of a next phase in the project. However, the literature on virtual reality for purposes such as education, training

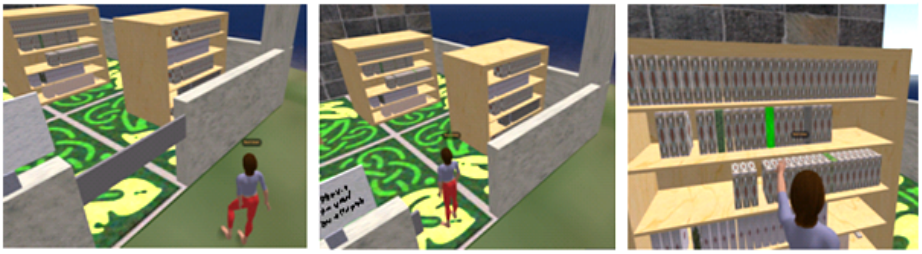

Fig. 7. Viewer interface - user common path

or medical treatment, contains good indications these systems provide for a rich enough experience to allow relevant results to be reached (e.g., see [1] for some interesting papers on the applicability of virtual reality to behavioural sciences).

In addition to exploring the environment, it is also possible to use the viewer to manipulate it, load objects into the environment and to save and clear the environment. This is achieved in the viewer by an avatar "shouting" commands: *load-oar file*, *save-oar file* and *clear*.

Besides exploration of the prototype, analysis of the models can also be considered. Using the State Space tool, provided with the CPN Tools, properties can be check in the model. For instance, reachability properties (e.g. all the states are reachable, a state is reachable from another one) can be expressed using functions provided by the State Space tool for this effect (e.g. AllReachable(), Reachable(node,node)). In the example, given specific assumptions about user behaviour, captured by adding an automated avatar to replace free user interaction, the model can be used to check properties such as that the required book will always be reached, collected and taken out of the library.

6 Conclusions and Future work

The user experience of ubiquitous environments is a determining factor in their success. Enabling early exploration of the characteristics of such systems will help anticipate potential user problems and reduce the cost of redesign. However, the deployment of prototypes in the target environment is, in many cases, infeasible. This happens both because of the cost of deploying such prototypes, and because doing it can be disruptive to the ongoing system. Alternatives must be sought that capture the experience of being immersed within the proposed ubiquitous system, without the cost of actually fielding it.

This paper described one such alternative. A simulation-based prototyping framework for ubiquitous computing systems. The framework brings together the expressive and analytic power of Petri nets, with the possibility of exploring a 3D virtual simulation of the modelled system. Petri nets constitute an ex-

pressive graphical notation. Development of the models and 3D environments is accelerated by the use of the CPN base model, and pre-defined devices. By enabling potential users to explore the simulation of the system before deployment, it becomes possible to have a low-cost approach to the prototyping problem.

Ongoing work on the development of the framework is addressing a number of technical issues in order to better support developers and users. One immediate aspect is the possibility of adding users to the simulation at runtime. In the current version of APEX, the number of users must be set at the start of the simulation run. This will be fixed in the next version of the framework. Another goal is reducing the amount of information exchanged by CPN Tools and APEX to a minimum. This is relevant both to prevent CPN Tools from running out of resources, and because it is envisaged that simulations will be deployed via the web. Connecting the simulation to user devices via bluetooth is also being addressed. This will encourage a more immersive and realist usage experience by allowing mixed reality. It also allows the possibility of moving progressively as part of the design and implementation process from a simulated system to a real system. Exploring the formal analysis of the models is also being considered. This requires the development of simulated users (capturing assumptions about user behaviour) to allow for a complete analysis. Hence, combining this feature with the the previous one, progress will be made towards a mixed economy of simulated and actual components of a proposed design. This will also support exploring how different levels of abstraction can be accomplished and supported. For example, supporting and enabling the migration of devices at the physical level via Bluetooth, at the virtual level as virtual devices in OpenSimulator, at the model level as CPN models.

Further development of the framework will involve its evaluation with users and developers. User evaluation concerns the fidelity of the results. Whether prototype environments can be used effectively to enable users to experience the design. Developer evaluation is concerned with the approach's agility. It is concerned with the ease with which accurate prototypes can be developed for ubiquitous environments.

References

[1] CyberPsychology & Behavior. Volume 6, Number 3/4 (2003)

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

[3] Braubach, L., Pokahr, A., Moldt, D., Bartelt, A., Lamersdorf, W.: Tool-supported interpreter-based user interface architecture for ubiquitous computing. In: Interactive Systems. Lecture Notes in Computer Science, vol. 2545, pp. 89–103. Springer-Verlag (2002)

[4] Buchenau, M., Suri, J.: Experience prototyping. In: Proceedings Designing Interactive Systems (DIS'00). pp. 424–433. ACM Press (2000)

[5] Davies, N., Landay, J., Hudson, S., Schmidt, A.: Rapid prototyping for ubiquitous computing — guest editors' introduction. IEEE Pervasive Computing 4(4), 15–17 (2005)

[6] Disz, T., Papka, M., Stevens, R.: UbiWorld: an environment integrating virtual reality, supercomputing, and design. In: Proceedings of the Heterogeneous Computing Workshop. pp. 46–59 (April 1997)
[7] Gallasch, G., Kristensen, L.: Comms/CPN: A communication infrastructure for external communication with design/CPN. In: Jensen, K. (ed.) 3rd Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools (CPN'01). pp. 75–90. DAIMI PB-554, Aarhus University (2001)
[8] Garlan, D., Siewiorek, D., Smailagic, A., Steenkiste, P.: Project Aura: toward distraction-free pervasive computing. IEEE Pervasive Computing pp. 22–31 (April-June 2002)
[9] Harrison, M., Campos, J., Doherty, G., Loer, K.: Connecting rigorous system analysis to experience centred design. In: Law, E., Hvannberg, E., Cockton, G. (eds.) Maturing Usability: Quality in Software, Interaction and Value, pp. 56–74. Human Computer Interaction Series, Springer-Verlag (2008)
[10] Harter, A., Hopper, A., Steggles, P., Ward, A., Webster, P.: The anatomy of a context-aware application. Wireless Networks 1, 1–16 (2001)
[11] Jensen, K., Kristensen, L., Wells, L.: 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 (2007)
[12] Jensen, K., Kristensen, L.M.: Coloured Petri Nets – Modelling and Validation of Concurrent Systems. Springer (2009)
[13] Li, Y., Hong, J., Landay, J.: Topiary: a tool for prototyping location-enhanced applications. In: UIST '04: Proceedings of the 17th annual ACM symposium on User interface software and technology. pp. 217–226. ACM (2004)
[14] Massink, M., Duke, D., Smith, S.: Towards hybrid interface specification for virtual environments. In: Duke, D., Puerta, A. (eds.) Design, Specification and Verification of Interactive Systems '99. pp. 30–51. Springer-Verlag (1999)
[15] Navarre, D., Palanque, P., Bastide, R., Schyn, A., Winckler, M., Nedel, L., Freitas, C.: A formal description of multimodal interaction techniques for immersive virtual reality applications. In: INTERACT 2005. Lecture Notes in Computer Science, vol. 3585, pp. 170–183. Springer-Verlag (2005)
[16] O'Neill, E., Lewis, D., Conlan, O.: A simulation-based approach to highly iterative prototyping of ubiquitous computing systems. In: 2nd International Conference on Simulation Tools and Techniques. pp. 1–10. ICST (2009)
[17] Shirehjini, A.A.N., Klar, F.: 3DSim: rapid prototyping ambient intelligence. In: Proceedings of the 2005 joint conference on Smart objects and ambient intelligence: innovative context-aware services: usages and technologies (2005)
[18] Silva, J., Campos, J., Harrison, M.: An infrastructure for experience centred agile prototyping of ambient intelligence. In: Calvary, G., Graham, T., Gray, P. (eds.) Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems. pp. 79–84. ACM Press (2009)
[19] Smith, S., Duke, D., Massink, M.: The hybrid world of virtual environments. Computer Graphics Forum 18(3), C287–C307 (1999)
[20] Westergaard, M., Lassen, K.B.: The britney suite animation tool. In: Donatelli, S., Thiagarajan, P.S. (eds.) Petri Nets and Other Models of Concurrency - ICATPN 2006. Lecture Notes in Computer Science, vol. 4024, pp. 431–440. Springer (2006)