Architecting a Reusable Platform for Pervasive Augmented Reality Games based on Petri Net Models

Tiago Agostinho,
Ivo Cosme,
Licínio Roque
Informatics Eng. Dep.
University of Coimbra
3030-290 Coimbra, Portugal
tacagostinho@gmail.com,
ivocosme@gmail.com,
lir@dei.uc.pt

Fernando Milagaia,
Fausto de Carvalho
Portugal Telecom Inovação, SA.
Aveiro, Portugal
fernando-j-milagaia@ptinovacao.pt,
cfausto@ptinovacao.pt

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Abstract
Pervasive Games require complex technical challenges to be overcome and their design space is yet mostly unexplored. In this paper we present a reusable platform for quickly designing, deploying and evaluating and managing multiplayer Augmented Reality Games. The games can explore visual and aural AR techniques, GPS and QR interface mechanics, and are structured as a set of activities defined using Petri Net models. These ARGs are typically played outdoors, using a smartphone client application. Evaluation of the platform involved field testing with realistic game designs exploring Cultural Heritage scenarios.

Author Keywords
Augmented Reality Games; Game Platform; Game Design.

ACM Classification Keywords
K.8.0 [Personal Computing]: General—Games.

General Terms
Game Design; Augmented Reality; Pervasive Games; Petri Net.

Introduction

Pervasive Games merge into real life, combining physical, social, and virtual worlds, exploiting mobile technology - sensing, positioning and frequently touchscreens - as interaction devices. Their potential remains mostly untapped in today's society, even though this kind of technology has become ubiquitous and advanced devices follow us everywhere, especially in urban contexts. Nowadays, the smartphone is a device that has a vast array of ways to interface with our surroundings, and present information to us. We can take movement, location, orientation and camera data to layer visual and aural information onto the real world, to augment it. Mobile augmented reality is starting to populate the mobile market, mostly by means of small but rich applications. Still, the process of creating a mobile AR game can be technically demanding, with many unpredictable interactions, both technically and socially. Moreover, the process of designing a game involves several phases and challenges, from idea to implementation. In this work we approached a previously recognized need to solve [CHI'2013 Reusable??] the problem of designing a general purpose reusable software infrastructure for defining, deploying and testing a diverse set of Pervasive Games exploiting AR techniques.

Software, Game and Interaction Design disciplines all resort to diverse modeling techniques to get a better understanding of what to build, mostly involving some form of flow diagrams and state charts, along with models of the software logic. Some authors have shown that game related logic can be modeled to some advantage by using Petri Nets [??]. A Petri Net is a directed graph, where transitions can represent tasks or activities, and places are populated with tokens to define the state of the network representing the game system. Interconnections represent action dependencies on resources of certain subsets of the system state. (which consume a given amount of tokens and specify production of tokens on output). In this work we adopted Petri Net diagrams as a basis for specifying game designs, and used it to map these designs directly into a computable implementation that can be immediately deployed and tested. With this we studied the feasibility of a software system implementing this architecture and the practicality of defining and playtesting pervasive AR games based on a reusable infrastructure.

In this paper we propose the architecture for a reusable platform for the quick creation and agile deployment of pervasive AR games, using smartphone technology. We used the multitude of interaction means supported by the smartphone to exploit techniques such as visual and aural overlays, location and direction tracking, time and proximity-based, and QR tag reading, as a basis to define a set of general action types. After defining game interactions they are encoded and their dependencies modeled using a Petri Nets modeling interface. This approach significantly speeds up the process of defining and deploying game variants for shortening the playtesting cycle.

Over the next sections we will present the design and implementation of the proposed architecture and an evaluation of the platform that was performed by game design students. The evaluation involved executing field tests with realistic proof-of-concept game designs, that also provided interesting indicators on the quality, design spectrum and pervasiveness of the games supported.

Related Work

Augmented Reality Games

Augmented Reality has been applied in games since the early 2000s, with notorious seminal projects such as *ARQuake* [5] or the *Human Pac-Man* [7]. However, for several years, it was restricted to the use of complex (and heavy) contraptions of wearable machinery, such as head-mounted displays, large GPS trackers, and backpack computers. With the rising of stronger, lighter and more multifunctional devices such as smartphones and tablets, a new generation of AR games became more practical.

Still today AR games are being developed mostly for small scale indoor scenarios involving the physical presence of fiducial markers or other small items that map into virtual objects or characters on a game setting. For instance, we have *ARhrrrr* [4], an Augmented FPS with an aerial view of a 3D mini town (mapped onto a paper sheet) which is overrun with zombies. Marker-dependent AR games may have prematurely reached their peak potential. Marketization has been achieved with for instance, the PSP game *Invizimals* [13].

On the other hand, AR games are still taking their baby steps in the field of outdoor games, especially market-wise. These comprise mostly treasure hunting or other types of rather simplistic or obvious interaction designs, limited to the context of the player's location and use of the camera. These limitations are often compensated with incorporation of multiplayer aspects (either for competition or collaboration scenarios). *NBA: King of Courts* [17], is a smartphone game in which physical places become basketball hoops where points are scored in a multiplayer social environment.

Another reason for interface simplicity is that outdoor AR games nowadays are more directed to be played occasionally during mid-short intervals (ex: while commuting). Also, they are played on-the-move, with a smartphone, implying that the GPS, wireless connection, and video feed are already being used, leaving little battery power for such complex methods as contour recognition. Some projects managed to solve this restriction by using other computation devices, but sacrificing mobility. For instance, *Carcade* [6] is an in-car videogame for the passengers, which captures the landscape's silhouettes (via laptop and camera) and combines them into a racetrack.

Pervasive Games

Pervasive gaming is a rather new research area and still in the phase of exploration. Of all the topics related to our research project, this had the most innovative published references we could find. However, one can consider that the most confined types of pervasive games (indoor with only basic sensors) have already reached the market (via Nintendo Wii). *Player Space Director* [11] is a framework to ease the creation of pervasive games, by portraying as an abstraction layer that integrates the inputs of a collection of sensors and the metadata retrieved (e.g. gesture processing). Some proof of concept games were created such as the treadmill-racing game Swan Boat [2].

Once we move outdoors, most researchers aim to put to test highly original concepts. For instance, these can either use the entire Wi-Fi network of a city into the capture-the-base multiplayer game *panOulu Conqueror* [20], or enhance cooperation by using only locally created ad-hoc networks on the game *Transhumance Team Exploration* [9].

One factor however is vastly common in research designs: Most outdoor concepts end up being loosely based on a treasure hunting mechanic, sometimes with a decent degree of narrative involved.

The *iPerg* [14] was a large-scale European project, that comprised designing and testing several pervasive games of diverse types. The initiative lasted almost four years and during that time it included treasure hunts and multiplayer alternate reality games, even featuring actor rosters and scripts.

Ingress [12] was Google's recent take on Pervasive Gaming through a complex alternate-reality narrative where a faction is "trying to establish portals around the world" and the other is "trying to stop them" via gathering a virtual resource ("Exotic Matter") and "hacking" "Portals" located at some city's points of interest.

Petri Net Modeling of Games

The feasibility of modeling games with Petri Nets has been proposed and studied previously. Since the use of UML at a conceptual or even specification level "lacks formal semantics that prevent them from being used in rigorous model analysis" [3], and Flowcharts "are limited to the modeling of sequential, non-concurrent systems", with little or no support for "conflict or concurrency for resources" [3] we explored Petri Nets as formal models that are expressive, easy to learn and can be used as computable models.

As proposed in [19], Petri Nets solve the gaps mentioned above. Petri Nets were invented in 1939 by Carl Petri with the purpose of describing chemical processes. The initial diagram concept has been extended in various ways and used in many and diverse domains. Petri Nets became an increasing asset during the rise of business modeling within the industry of Information Systems to represent workflow processes in a simple and accessible way [1]. The main reason behind this is that, with a few extensions, Petri Nets can model virtually any process. Since Petri Nets are graphic representations and composed by creative arrangements of repeated use of such a small set of elements, they become easy to learn and to use. Another advantage is that one can use a divide-and-conquer approach by subdividing the process model into smaller Petri Nets, by zooming in and out on parts of a larger network.

Definition of games with Petri Nets can be seen in [3], where a strategy game involving the Portuguese Maritime Discoveries was modeled through several subnets. These Petri Nets modeled both the game mechanics (such as the energy and state of the ship) and the flow of player action (the actions players could take at any given moment, considering the existing conditions). The ability to represent concurrency is a valuable asset for strategy games and massively multiplayer online RPGs. Also, as Petri Nets have a mathematically strict formality, the games flow can easily be simulated through existing tools, to help foreseeing issues before implementing the game.

With small expansions to the original concept of a place, the authors of [22] were able to design and simulate complex quests for *NeverWinter Nights* via Petri Nets, before inserting them into the game using the built-in *Plot Wizard*. The extended Petri Net design was able to model every aspect about Non Playable Characters interaction and item achievement.

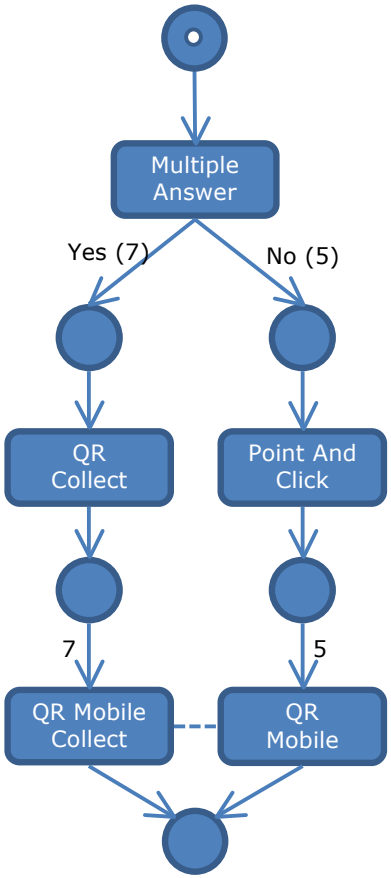


Image 1 – A Petri Net sample

Modeling AR Games with Petri Nets

The structuring of the games on our platform is based on an adapted Petri Net model. Here, for readability, transitions are named actions, places are dubbed as resources, and dependencies that link them are named bridges or connections. Resources, along with the Tokens that populate them, define the state of the Petri Net, i.e. the Actions that can be executed at a given moment. An action can be executed if all its input resources have the required amount of tokens to be consumed and, upon execution, produces the specified amount of tokens in all of its output resources (except for a few macro-actions that encode differentiated flows of execution).

Each player has its own private Petri Net network marking (which tokens are at each place, repesenting player state in the game). To enable definition of team gameplaying and other player interactions, some resources can be shared by all game participants state or only among team members.

On top of this, the designer can define Goals that can be associated with resources to trigger when a specified condition or amount of tokens is reached. These goals can also be used to mimic the concept of obtainable or collectable items on an inventory. A point counter can also be linked to a resource, which supports the notion of a player's score or even a player ranking system.

Player Actions can be of several types, including:

- Point and Click: If the player enters a proximity zone for a GPS coordinate, she will see an AR overlay when pointing in a given direction. Action finishes by clicking the object displayed;
- Listen And Click: Player is invited to hunt for the source of a sound with only the hearing sense (louder as the player the closes in in the right direction);
- QR Collect: collect printed or on-screen QR codes from the environment or from other players;
- QR Mobile Collect (to be paired with QR Mobile): The first player reads a QR code directly from the second player's device. Used to model direct player interaction;
- Dialog Message: shows a message to the player, e.g. for narrative, orientation or other purposes, with an image and a textual component;
- Dialog (Single or Multiple) Answer: a question and answering interface (useful for narrative insertion and progression checking). Can be specified to produce tokens on different outputs depending on the answer given;
- Timed Event: an action that is triggered when a time condition is met, enabling other actions by releasing tokens satisfying their dependencies;
- Enter (and Exit) Proximity: this action fires when the player enters (or exits) the radius of a location;
- Player Router: automatically produces tokens on only one of its outputs, used to distribute or direct game flows based on random conditions;
- Player Selector: automatically produces tokens on only one of its outputs (based on some feature such as player's email address).

The example on figure 1 maps a game network model symbolizing the synergy of an ant colony. First, players would be asked "Do you like sugar?", and according to their answer they are redirect towards one of the specified branches. If a player likes sugar, he would

hunt for sugar heaps via QR Collect (this action will be available 7 times, the number of tokens loaded on its precondition place). If he answers no, he can search for virtual (augmented reality) sticks to build the colony, via Point and Click actions (available 5 times). Once he collects 7 sugar heaps, or 5 sticks, he can then interact with a player that took the other path, and both reach the end of the game. A goal could also be set on the last resource place symbolizing the reaching of a final achievement or prize.

The Proposed Architecture

Definition

The proposed architecture pictured in figure 2 is comprised of two servers (Back Office and Gaming) and an gameplay interface Client currently in Android. A client application runs on Android operating systems. The client communicates with the server via HTTP requests. Each time there is a change in game state affecting a player, the server pushes a notification (via Google Cloud Messaging) to the corresponding client application.

The Petri Net models for determining and executing available game Actions is processed in the Game Instance and Player Instance processes. When a Client requests the execution of a given Action, this message reaches the corresponding *Player Instance* (after being handled by the service layers above). The Action request and its private state dependencies are sent from the *Player Instance* to the *Game Instance* for verification and processing of the shared game state, to determine whether the action is executable at the moment and to generate its outcome. If enabled, the *Game Instance* process performs the removal of tokens from the input Resources, and generate the outputs into the Action's output Resources (both private and shared). It then messages the new private Resource states to the *Player Instance*. The whole change to the state is replicated to the database via the *Storage* process (in an asynchronous way, thus guaranteeing better performance).

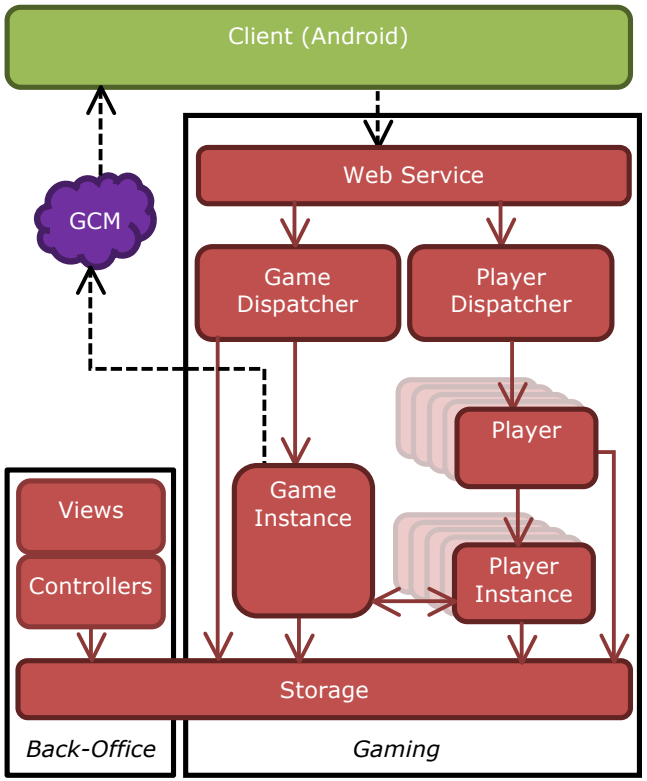
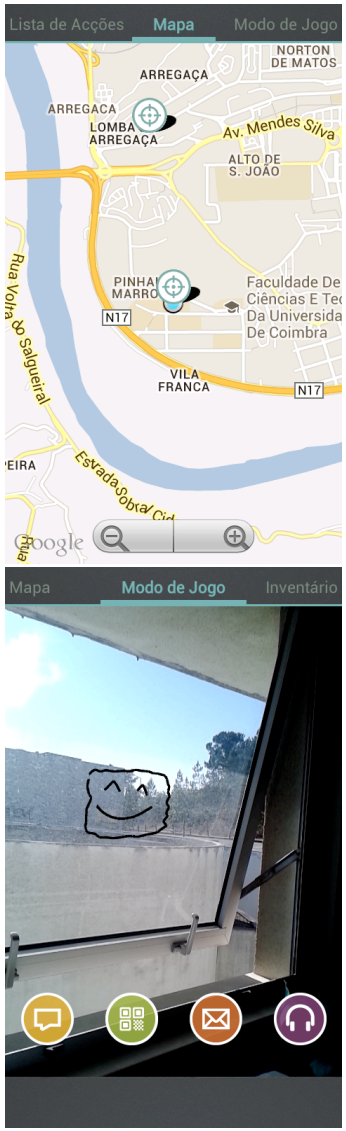


Figure 2. Proposed Software Architecture. Legend:
Red Components – Erlang Server processes;
Green Component – Android Client;
Purple component – Google Cloud Messaging Server;
Connectors: Continuous – messages, calls; Dotted – HTTP;



When states (Petri Nets markings) change, all affected players' device Clients are notified by the *Game Instance* process via *Google Cloud Messaging* (*GCM* – an Android push notification system) [10]. This approach also creates a useful abstraction for the Client, which only knows the playable Actions for one player at a given moment, and thus handles only the Action presentation and its interface with the player (example interfaces pictured in figure 3).

The service side architecture has two major functionality blocks. We just described the *Gaming* component that is responsible for managing all running games. The Back-Office functional block is a Website for creating and editing game definition PN models via a web Browser interface. It is coded in Nitrogen, an Erlang Web Framework. This framework only handles the VC (*View* and *Controller*) components of the MVC architecture, leaving the Model component to be handled in the *Storage* component, which is ideal for this scenario, where we have only one database definition.

Implementation

Our first attempt at implementing the architecture included a much less scalable Ruby service that managed game definitions and state as Petri Nets (PN) encoded and updated on a relational database. That implementation was abandoned due to a serious bottleneck in database access, that rendered it incompatible with real-time PN execution. Player connections were managed through TCP/IP connections and there were massive state pooling requests that rendered the system quite unstable under intermittent network conditions.

The current server is coded in Erlang [8], which is a functional language structured as concurrent processes that interact via messages (no locking mechanisms and no shared states). In this Erlang implementation, updating game state is done in memory, and therefore, with limited concurrency over the database stored state, that is managed by the Storage process. Dynamically, the runtime processing can be described as follows:

- The Erlang server receives and parses requests on its *WebServer* module. It uses MochiWeb [16] as the HTTP server library
- A message is then sent to the *Player Dispatcher* (if it is of a private nature – concerning just the player's state) or the *Game Dispatcher* (otherwise)
- Each online player has its own instance of the *Player* module, that logs locations and information that is not dependent on the specifics of each game he plays
- Management of the Petri Net begins at the *Player Instance* (one process instance for each player in each game – holds every player private resource state) and the *Game Instance* (one for each running game – holds everything shared between players of a game and manages game state changes)
- Every single piece of information, for both online games (running) and offline games (either stopped for editing or without any active players at the moment), is kept stored in the Mnesia [15] database (Erlang's built-in database module), that is accessed through *Storage*.

Player Interface

In figure 3, we can see two screenshots of the Client interface: Map view and camera View.

- Map – displays player's and visible Actions' locations on a map of the surroundings.
- Camera (a.k.a. Gaming mode) – comprises the almost all the visual content of the games. We can see that a total of 5 Actions is playable at the moment. In the bottom, there are 4 execution icons for (from left to right) *Dialog Answer, QR Collect, Dialog Message and Listen and Click*. The smiley on the center-left of the camera is a *Point And Click's* Augmented Reality object. It is tethered to a location a few meters away in that direction. If a player clicks the object it will be captured, completing the *Point And Click*.

There are also other Client views for Inventory (related to the Petri Net Goals), Ranking, Chat and Message History, as well as a specific interface for each kind of "activity" that can be coded in the game design.

Evaluation

To evaluate the reusability of the proposed architecture with realistic game designs, we resorted to 3 student design teams from a Game Design and Development course from an Engineering Masters Program and from a European Masters on Cultural Heritage Studies. The Informatics Engineering game concepts developed were more focused on promoting interaction between players. The EuroMACHS design was centered on adapting a Cultural Heritage scenario into a game format, with a rich multipath narrative.

To translate the concepts into the platform, some adaptations were made. These included mapping the game concept into the required Petri Nets models and rethinking some elements to take advantage of the proposed reusable action types. The adventure style narrative game concept became a Petri Net with over 70 Actions encoded, while the other games were significantly simpler and more "circular" or repetitive, with fewer actions and more cycles. After tuning the proposed game designs, the final games deployed were:

- *City by Night* - player complete a set of challenges spread throughout the nocturnal establishments of the City. The barmen also played to confirm (using the QRCollectMobile-QRMobile action pair) that certain challenges have been fulfilled (e.g. drinking a shot).
- *PIDE vs Revolucionários* - a multiplayer, team-based game inserted on a pre-revolutionary Portuguese context, where players take on one of two roles, either the regime police or the insurgents, and walk to explore the area searching for clues to the identity of their adversaries, whom they have to capture before being discovered themselves.
- *Mystery of Alta* - a singleplayer game with a dense narrative, where the player has to communicate with the game characters (helpers/opponents), to act or to decode clues, to explore the University surroundings, acquiring knowledge about its history and traditions.

This collaboration culminated with a game exhibit day where several users tried out the applications. This corresponded to our field test, since multiple game instances were running at the same time and it was possible to understand in loco the difficulties posed by some interfaces and the challenges associated with envisioning how the players might act at each specific location. To collect data, all requests to the server were logged, along with some important data, such as, location and player ids. Due to its high Petri Net

complexity paired with a large array of location-based Actions, the *Mystery of Alta* game was also used for a more extensive analysis leading an estimation of player mobility.

Results

The support of three different game designs acted as a first feasibility test for the platform and its reusability. Both *City by Night* and *Pide vs Revolutionários* escaped from the trend of treasure hunting that we have seen on *Pervasive Games*. They were more focused on player interaction than geolocation. These concepts were only made possible within the platform by the introduction of direct player interaction actions (i.e. *QRCollectMobile* and *QRMobile* pairs) and routing actions (e.g. *Player Selector*), thus pointing to the relevance of these action types.

Cooperation with the design teams revealed the importance of having a platform to quickly and iteratively deploy game concepts with less coding knowledge, which gave them more time to think on the game elements and scenarios. On the other hand, their usage of the platform served to validate existing functions and as participatory design opportunity as they requested changes/additions such as player routing, random flow, and multiple answer redirection actions that significantly enhanced the spectrum of possible game definitions.

The field tests during the exhibit day produced a hefty amount of log data. Some indicators could be drawn from the *Mystery of Alta* case, as shown in table 1. These values were defined to reflect an exploration rate (an approach to movement detected between actions), since it is based on the distances between locations of consecutive requests. The mobility rate was calculated to comprise all blocks of 10 seconds for which cumulative distances were more than 10 meters, signaling the players were active exploring the space (to discard small movements or GPS corrections).

Indicator	**Mean**
% Time exploring (player mobility rate)	61%
Average exploring speed	3 km/h

Table 1. Information drawn from Field Test results

In *Mystery of Alta,* with such large amounts of dialogues and questions (which draw on the player's attention, creating intervals without movement), it was rather interesting to verify that players dedicated 61% of their game time exploring the action's physical surroundings. An average moving speed of 3 km/h is close to human's default walking speed (5 km/h [21]), meaning that exploration was mostly made on-the-move, which we think might be representative of what to expect from pervasive games. Also, one should notice that the speed isn't too much on top of the human's walking speed. This means that players did not just walk straight to the Action's target location, but instead deviated from the optimal route between actions, thus also indicating exploration. Currently there is still a lack of reference values to understand what to expect while designing pervasive games, therefore, these indicators can contribute with a first useful approximation.

Acknowledgement

This research resulted from the AdVenture project developed at IPN, in partnership and with funding by Portugal Telecom Inovação in the scope the company's Plano de Inovação 2012-2012 and 2012-2013

Conclusions

In this paper we presented an architecture for a reusable platform for designing and playtesting pervasive AR games, validated through a set of design cases and field tests. We also presented our approach to modeling pervasive AR games resorting to Petri Net models and a definition of general purpose action types to be reused. We concluded that the approach for defining and computing games with Petri Net definitions is valid and agile, and that it could be well understood by game designers to an interesting spectrum of game models, that were quickly deployed and playtested. By calculating player mobility rates and speeds we validated indicators for the pervasiveness of the game designs.

References

[1] Aalst, W. and van Hee, K. Workflow Management: Models, Methods, and Systems. MIT Press (2004)

[2] Ahn, M., Choe, S., Kwon, S., Park, B., Park, T., Cho, S., Park, J., Rhee, Y. and Song, J. 2009. Swan Boat: Pervasive Social Game to Enhance Treadmill Running. *Proc. 17th ACM international conference on Multimedia*, ACM (2009), 997-998.

[3] Araújo, M. and Roque, L. (2009). Modeling Games with Petri Nets. *Proc. of the DIGRA2009, London.*

[4] Arhrrrr Zombies. http://ael.gatech.edu/lab/research/handheld-ar/arhrrrr

[5] ARQuake. http://wearables.unisa.edu.au/projects/arquake

[6] Carcade. http://www.ohgizmo.com/2008/10/08 /carcade-in-car-gaming

[7] Cheok, A., Fong, S., Goh, K., Yang, X., Liu, W. and Farzbiz, F. Human Pacman: a sensing-based mobile entertainment system with ubiquitous computing and tangible interaction. *Proc. NetGames 2003*, ACM (2003), 106-117.

[8] Erlang. http://www.erlang.org

[9] Gentes, A. Guyot-Mbodji, A. and Demeure, I.. Gaming on the Move: Urban Experience as a New Paradigm for Mobile Pervasive Game Design. *Proc. MindTrek 2008*, ACM (2008), 23-28.

[10] Google Cloud Messaging for Android. http://developer.android.com/google/gcm/index.html

[11] Hwang, I., Lee, Y., Park, T. and Song, J. Towards a Mobile Platform for Pervasive Games. *Proc. of the MobileGames 2012*, ACM (2012), 19-24.

[12] Ingress. http://www.ingress.com/

[13] Invizimals. http://www.invizimals.com/home.php?locale=pt_PT

[14] IPerG – Integrated Project of Pervasive Games. http://www.pervasive-gaming.org

[15] Mnesia http://www.erlang.org/doc/apps/mnesia/

[16] Mochiweb. https://github.com/mochi/mochiweb

[17] NBA King Of Courts. http://www.ogmento.com/games/nba-king-of-the-court

[18] Nitrogen. http://nitrogenproject.com/

[19] Petri, C., and Reisig, W. Petri Net. http://www.scholarpedia.org/article/Petri_net Games. *Ext. Abstracts CHI 2009*, ACM (2009), 4213-4218.

[20] Tiensyrjä, J., Ojala, T., Hakanen, T. and Salmi O. panOULU Conqueror: Pervasive Location-Aware Multiplayer Game for City-Wide Wireless Network. *Proc. of the Fun and Games 2010*, ACM (2010), 157-165.

[21] Transafety Study Compares Younger and Older Pedestrian Walking Speeds. http:// www.usroads.com/journals/p/rej/9710/re971001.htm

[22] Young-Seol L. and Sung-Bae C. 2012. Dynamic quest plot generation using Petri net planning. *Proc. of the* WASA 2012, ACM (2012), 47-52.