2015 IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing

Separating Gesture Detection and Application Control Concerns with a Multimodal Architecture

Leonel Morgado
INESC TEC and Universidade Aberta
Coimbra, Portugal
leonel.morgado@uab.pt

Bernardo Cardoso, Fausto de Carvalho
PT Inovação
Aveiro, Portugal
{bernardo, cfausto}@telecom.pt

Luís Fernandes, Hugo Paredes, Luís Barbosa, Benjamim Fonseca, Paulo Martins, Ricardo Rodrigues Nunes
INESC TEC and UTAD
Vila Real, Portugal
{lfernandes,hparedes,lfb,benjaf,pmartins,rrnunes}@utad.pt

***Abstract*—Gesture-controlled applications typically are tied to specific gestures, and also tied to specific recognition methods and specific gesture-detection devices. We propose a concern-separation architecture, which mediates the following concerns: gesture acquisition; gesture recognition; and gestural control. It enables application developers to respond to gesture-independent commands, recognized using plug-in gesture-recognition modules that process gesture data via both device-dependent and device-independent data formats and callbacks. Its feasibility is demonstrated with a sample implementation.**

Keywords—gestural commands; concern independence; gesture-based interfaces; LeapMotion; Kinect

I. INTRODUCTION

In so-called "natural" user interfaces (NUI), gestural-interaction with the user environment is a forefront element, from tactile screens of smartphones and tablets to full-body somatic interaction using Microsoft Kinect or other devices [1]. They purport to be natural by leveraging users' preexisting skills **Error! Reference source not found.**, and aim to render the NUI invisible in cognitive terms **Error! Reference source not found.**. Recently, however, the term "natural" in such interfaces has been under critique, since the meaning associated with gestures varies across cultures, social groups, and sometimes even from person to person: "*Most gestures are neither natural nor easy to learn or remember. Few are innate or readily predisposed to rapid and easy learning. Even the simple headshake is puzzling when cultures intermix*" [4]. Actual sequences of gestures and their association with commands have not been under much scrutiny, and early proposals for evaluation of the "naturalness" of task sequences using gestures have now started to emerge [5]. The need to enable the customization of gestures, leveraging the semantic richness of gestures in specific cultures (gestural "emblems") a.k.a. as the "shamanic interface" concept, has also been proposed [6].

In coming years, as knowledge builds up on interaction design for gestural interfaces, current gestural command methods for applications will likely become obsolete. This obsolescence should not prevent current applications from being available under new gestural command modes. Our perspective on this problem is that it is simply a new application scenario for the classic software engineering paradigm of separation of concerns. In this paper, our contribution to the software engineering of gesture-based applications is an architecture proposal which separates the following concerns: gesture acquisition; gesture recognition; and gestural control. The actual code is open source, available at: https://bitbucket.org/Apidcloud/inmerse-framework/

In our view, this architecture may enable software development teams to approach each of these concerns separately. Hence, a team working with a specific gesture-acquisition device may provide the adapter modules that leverage the device's API to provide its data and services/callbacks in a standardized way, usable by other architectural modules. A team dealing with the recognition of specific gestures may leverage the standardized data and services to provide gesture-recognition services, and an application-development team may employ both kinds of modules (gesture-acquisition and gesture-recognition) to react to commands, simply swapping modules when necessary, avoiding fast obsolescence.

II. BACKGROUND

A. Separation of concerns in interactive applications

In interactive applications, the Model-View-Controller (MVC) architectural style is typically used to attain separation of concerns between visualization, application logic, and input control. However, it has been criticized for breaking separation of concerns in a key aspect: interpreting input and mapping it to an application's functionality [8]. Proposals for overcoming this issue have emerged in recent years. For instance, the Curry

978-1-5090-0154-5/15 $31.00 © 2015 IEEE
DOI 10.1109/CIT/IUCC/DASC/PICOM.2015.233

& Grace approach moves interaction concerns into the View module, a common approach in dialog-based applications [9]. But in gestural applications, where input and output take place in different media, this simply replaces a breakdown of separation by another. Angelopoulou et al. [11] proposed a NUI system that operates as a front-end framework for applications designed for traditional input, i.e., a wrapper approach for legacy software. A more adequate approach for development of new software is the MVIC extension to the MVC style [8], which inserts an Interaction module between the user and the Controller. A similar approach was followed by Carvalho et al. specifically for gestural-command applications, by proposing a three-tier approach to input processing: gesture acquisition & recognition; parameterized cultural mapping of gestures; and application control [10].

B. Low-cost gesture acquisition

The use of gestures to control computerized devices has long been a topic of research [12], even for fields such as mobile devices [13]. A generic goal is to leverage users' preexisting experience of bodily interaction in space and its associated rich semantics, to shrink the conceptual gap between users' actions and system response, and thus enable users to work with more complex mental models [14]. E.g., enable non-technical users to stage augmented-reality choreographies of stories, and put across conflicting perspectives of urban development [15].

There are currently many low-cost computational devices and sensors available, and not surprisingly a large diversity of low-cost gestural-interaction devices emerged, possibly brought to the public's attention due to the commercial success of Nintendo's Wii and its motion controller, the Wii Remote. Most notably, Microsoft's Kinect device has drawn much attention from the press and the research community, due to its ability to recognize full-body motion without requiring calibration, and being readily usable not only on the Xbox gaming system but also on PCs and other computational devices. More recent devices include Leap Motion, which tracks fingers, hands, wrists, and forearms (on both upper limbs simultaneously), and the Myo armband, which uses a combination of electromyography and kinetic sensors to identify arm, hand, and finger motions.

C. Devices used in the protype

To produce a concrete implementation of the architecture presented in this paper, we employed two distinct gesture-acquisition devices: the Leap Motion and the recent Kinect 2.

Fig. 1. The Leap Motion (left) and Kinect 2 (right) devices

As shown in Fig. 1, the Leap Motion is a small brick-shaped sensor, which samples the space above at regular intervals. It detects the position of forearms, hands, and fingers, and provides data in specialized structures over a sequence of time frames.

The Kinect 2, also shown in the same figure, acts in a similar way, but targeting the full body, therefore its data structures are distinct, focusing on providing positions of the body frame. A specific aspect of the Kinect 2 software services is their ability to provide not just the body position, but also detect higher order gestures (static and continuous) via machine learning algorithms. Hence, applications are able to process the sequence of body frame positions directly, and also respond to Kinect-detected gestural emblems.

III. The InMerse Multimodal Architecture proposal

The proposal herein was developed in the context of a corporate-funded innovation project, called InMerse (mentioned in the acknowledgments section). InMerse aimed to lay the grounds for creation and development of immersive and augmented applications and services in a corporate context. Rather than focus on specific application programming interfaces (APIs) or similar one-shot developer-oriented tools, an ongoing concern was for project outcomes to be made available for later use, by different departments or teams, as individuals and independent teams come up with ideas for prototype products/services. This made us consider early on that a team creating a prototype gesture-acquisition device could be dismantled and not readily available for software maintenance or customization at a later date, when a different department might come up with an idea for testing it as part of a different product/service prototype. Likewise, work done on recognition algorithms for specific gestures would be more readily available if it could be plugged-in into existing and future applications and benefit from data acquired through new devices, not just the ones available while the algorithms were being developed. Finally, an application-development team would be empowered if its operation could be independent from the specifics of how gesture acquisition and recognition is made. This perspective, alongside the overall concern with avoiding early obsolescence, was the inspiration for the current architecture.

A. Overview

The core structure of this architecture is the separation of the three team-oriented perspectives mentioned above: gesture acquisition, gesture recognition, and application command.

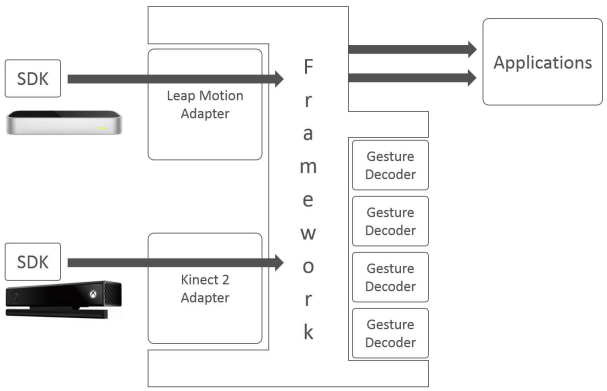

Fig. 2. The overview of the InMERSE multimodal architecture.

A central core element provides intermediation and abstraction services between modules servicing these perspective. Fig. 2 provides this overview: different gesture-acquisition devices are interfaced by device-specific **Adapter** modules (left side of the picture), and multiple **Decoder** modules are plugged-in to provide gesture recognition services tuned to different requirements (right side, bottom). The core **Framework** module provides the intermediation and abstraction services and enables **Application** modules to react to abstract commands, rather than the specific gestures that elicited them.

This enables Applications and gesture recognition Decoders to focus on commands and device-independent data and callbacks, provided by the Framework module, rather than on the low-level issues concerning gesture acquisition. For instance, a typical concern of gesture acquisition is dealing with lack of precision and ambiguity in acquired data [16][17]. In the InMerse architecture, low-level approaches to such problems are dealt in Adapter modules, which can be debugged, updated or even replaced independently. Similarly, higher-level approaches are dealt with in Decoder modules, with the same level of independence.

B. Data perspective

This architecture provides Applications access to three different kinds of data, as shown in Fig. 3.

- **Commands**, which are gesture-independent information. The Framework module will associate a gesture identified by a Decoder module with a command, hence an application doesn't need to know (or care) if "Activate" was issued by moving a finger into a virtual button, focusing on it with one's eyes for 3 seconds, or doing a thumbs' up, for instance.
- **Gestures**, which is simply a transparent access to Decoders' output, if an Application requires at some point this level of detail (for instance, for user tutoring purposes).
- **Basic Data**, or transparent access to the Framework data structures containing the gestural data. Applications may require this, for instance, for displaying continual motion of the user's hand.

Providing access to lower-level data may seem to defeat the purpose of separation of concerns – and it does. However, it means an Application can use the concern-separation, command-based approach whenever possible, and only be tied to lower-level data when that is unavoidable in the current state of the architecture. For instance, suppose an application allows the user to "grab" a virtual object and "drag" it in space, until it is "released". The application can simply react to Commands for "Start drag" and "End drag", and benefit from separation of concerns, and only collect raw data for space-tracking purposes during dragging.

Hence, different gestures for "grabbing" and "releasing" could be associated with the "Start drag" and "End drag" commands, for different situations or users. For instance, a user that is unable to perform arm movements could use a specific Decoder to recognize gaze duration and blinking as "grab" and "release" gestures (and the Application can use the Framework module to associate these with "start drag" and "end drag"). In this same scenario, the acquisition device of that user can provide head-tracking data for the motion, which is stored in the Framework as Basic Data, accessible by the Application for providing feedback during the dragging.

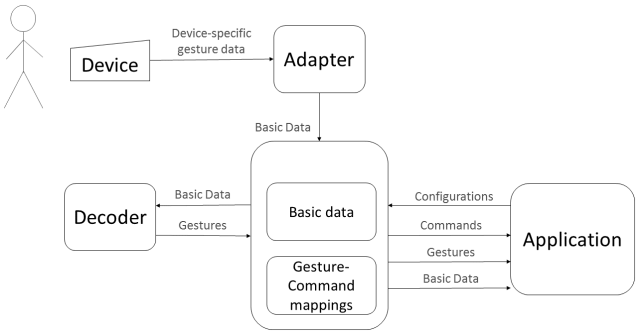

Fig. 3. InMerse architecture Data and Callback perspective (each flow can be implemented as either/both a function call or a function callback).

IV. AN IMPLEMENTATION OF THE INMERSE ARCHITECTURE

We have developed a prototype implementation of the InMerse architecture, shown in Fig. 4, to ascertain its feasibility. It is explained in the following sections and has enabled us to confirm that indeed this approach is feasible and warrants more in-depth exploration subsequently.

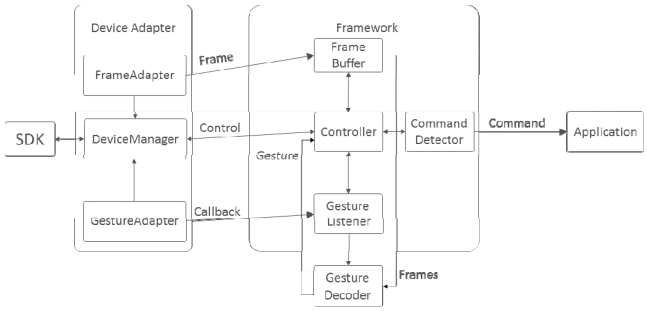

Fig. 4. Detail of the prototype implementation of the InMerse architecture

A. Gesture data acquisition: Basic Data and Adapters

As a first approach to storing Basic Data from gesture acquisition, we defined it as a buffer ("Frame Buffer"), which stores a sequence of timeframes (vd. Fig. 4). Each timeframe ("Frame") contains a static spatial position of the gesture data. The actual data represented in each frame can vary significantly from device to device, and it is each device-specific Adapter that needs to convert device-specific data into architecture-agnostic Frame data. While at the moment we are simply considering skeleton data with joint positions, and using traditional polymorphism to enable diverse data formats, subsequent evolutions should enable a more diverse set of data formats, using techniques such as ontology representation and transformations, which we are exploring in parallel research efforts [18], but are not the focus of this work. Another example of details that require further study is that of

differences in data production and consumption speeds, between adapters, decoders, and applications. A possible approach, which we've pursued in an Online Gymnastics project [19], is to drop some frames to enable quality-of-service responsiveness. These are just two examples of problem areas that we acknowledge need to be tackled in subsequent research, but that we are not pursuing in this paper, whose goal is to present and make the case for the overall InMerse architecture.

In the current prototype, Adapters have been implemented with the internal structure presented in Fig. 4. We used a core controller module, called Device Manager, which serves as a bridge, dealing with the specific API calls and data formats of the Software Development Kit provided by the manufacturer of each gesture-acquisition device. To translate device-specific data into the Frame data structure used by the current Framework implementation, we created a Frame Adapter module inside each Adapter. To provide the Framework with access to services provided by device-specific callbacks, we created an indirection module, inside each Adapter, called Gesture Adapter. The reason for this naming is that common services provided by device-specific APIs include preset recognition of some gestures. By providing indirect access to these via the Framework, where we created a Gesture Listener submodule for this purpose, the Adapters enable Gesture Decoder modules to benefit from leveraging low-level gesture recognition when it is available.

B. Gesture recognition: Decoders

With the Basic Data on gestures stored in the Framework module, Gesture Decoders access it to recognize gestures not just as three-dimensional positions, but as a sequence of positions in time. This means that it is the job of each Gesture Decoder to decide which features are relevant for gesture recognition from the Basic Data. In the scope of this architecture, a "gesture" can be a simple static position in space, a wide displacement of limbs or something as specific as a cultural-meaningful ritual, with specific rhythmic and amplitude requirements (in semiotics terminology, an emblem).

In the current prototype, we implemented this access to Basic Data via two submodules of the Framework module. As mentioned in the previous section, a Gesture Listener provides the Decoders with indirect access to the low-level device SDK callbacks. And the Frame Buffer submodule provides them with access to the time sequence of Frames containing the spatial position of skeleton joints.

C. Command issuing: Framework and Applications

When a Decoder recognizes a gesture, it reports it to the Framework module, which will decide whether or not it is a command to which an application needs to respond. This implies that Applications need to register in the Framework module which commands they are listening to, and the Framework needs to be configured to perform the matching of gestures to commands. In the current prototype, we implemented a simple Command Detector submodule that reads a configuration file with gesture-to-command pairings, alongside some parameterization of the matching, but we intend to explore more advanced methods. For instance, in an earlier work, Carvalho et al. had a global "culture" setting which was used to associate cultural-specific gestures with generic commands [10], something we could readily include in such a prototype, as well as more complex approaches. Again, such details are for subsequent work and not the focus of this paper. One variation we did implement was the ability to have different operation modes as part of the Command Detector operation, as described in the following section.

D. Operation modes

As mentioned in the previous section, we did include an extra scenario – one might call it an intentional complication – in the command detection process, to further test the feasibility of the architecture. In this scenario, we considered the need for an application to perform continuous hand-tracking operations, such as dragging virtual items or pointing at virtual elements.

While this requires the application to have systematic access to the hand position, which it can have by accessing the Basic Data in the Framework, rather than device-specific data, we considered how such scenarios might be initiated and terminated. We considered two situations:

- **Application-determined tracking**, in which the application decides to access the hand position in response to its own internal state;
- **Framework-determined tracking**, in which the application configures the framework with start and end gestures for hand tracking.

Both situations are similar, in that the Application has to retrieve the hand position form the Basic Data in order to display its tracking to the user (e.g., moving/rendering the dragged virtual object or virtual pointer). The major difference is that in the first situation (application-determined) the Gesture Decoders are fully operational, and proceed to identify "gestures" as the tracked hands drag or point in the virtual space. Thus, the application keeps receiving command notices as those gestures are associated with commands, and it is the application's task to decide whether to ignore them or interpret them. This also means that the whole system is potentially being subjected to needless processing jobs. In the second situation (framework-determined), the application configures the framework with "start" and "end" gesture-command pairings for tracking. When the start command is identified, the application is notified, but no more command detections are reported to the Application, which is solely receiving or requesting Basic Data. When the end command is identified, the application is notified, and from then on starts to be notified of all Commands identified by the Framework.

These are thus two different operation modes of the Framework. We called the first **Acquisition Mode** and the second one **Detection Mode**. While we have not implemented further, this approach opens up the possibility of having Decoders identify the distinct mode in which the Framework is operating, and adjust their recognition algorithms/jobs accordingly.

V. TEST PROTOTYPE: GIBBO 2D DIGITAL SIGNAGE

A. The digital signage application

To test the operation of the implemented prototype, we developed a digital signage application with the open source Gibbo2D[1] game engine, which can be used seamlessly with Leap Motion or Kinect 2 gesture-acquisition devices. Fig. 5 shows this application in action. Against a world map backdrop, the user can pan, zoom in, and zoom out, using hand gestures. In this figure, an animated virtual hand is making a "pan left" gesture and the Portuguese-language text states "Reproduce this gesture to proceed." After presenting to the user each of the various gestures that can be used and requesting their reproduction as a tutorial, the user can use those gestures for panning and zooming.

Fig. 5. Digital Signage prototype implemented using Gibbo2D and the InMERSE framework

B. Implementation aspects

So that the Application could work seamlessly with Leap Motion and Kinect 2 devices, we implemented an Adapter for each. The user's gesture data is captured by the device he/she is using, and its Adapter translates this into Frame data structures, stored in the Frame Buffer inside the Framework module.

From the frames in the Frame Buffer, the Gesture Decoder module detects the valid gestures and passes them on to the Framework module, whose Command Detector submodule converts into Application-specific commands: pan left, pan right, zoom in, and zoom out (plus a "start application" command).

When the Digital Signage application receives a command notification, it reacts accordingly alongside the world map background (or, in the tutorial phase, reacts by either moving along the tutorial or ignoring a wrong command).

An interesting situation was found while studying the developer resources of Kinect 2. Time framing capture is feasible in Kinect 2, but development examples are geared towards defining gestures prior to execution, storing them in gesture files or databases, and then running gesture detection by direct access to the SDK to match frames with those stored gestures. Following the InMerse architecture, this latter approach should not be done entirely in a Gesture Detector module, since that would bypass the Framework and Adapter modules, and thus break the separation of concerns. Rather, loading a gesture database and launching events (callbacks, in this paper's terminology) upon detection should be done in the Adapter module for the Kinect 2. This Adapter can load the Framework module with time frames of the detected body, and make gesture-detection events available to the Framework as callbacks. This makes Kinect 2 data and events available to all Gesture Detector modules, which can query the Frame Buffer or listen to callbacks. But the Adapter needs to know which gesture database to use, which means that taking advantage of low-level Adapter services in Gestor Detectors may require extending the current architecture with an upstream configuration data flow. However, we have not included such a flow in this paper, for we have not yet conducted theoretical reflection on its impact nor have we prototyped its feasibility.

[1] gibbo2d.anlagehub.com

VI. DISCUSSION AND FINAL THOUGHTS

The usage/efficiency of this prototype architecture needs to be tested in depth, as the results are very much preliminary. As mentioned throughout the paper, there are still several open issues and topics for reflection, requiring further work. We have not addressed in detail how to achieve adequate abstraction for multiple input data formats and diversified semantic content (e.g. different skeleton assumptions), albeit work done on ontology transformation in other research holds promise for achieving a solution for this problem [18]. Also mentioned was the issue of achieving adequate quality of service in processing time frame data across devices with very different capabilities in this regard. We have also not addressed this issue, but work done by other teams by dropping some frames to keep up with varying transmission and processing capabilities is indicative that this problem may also be reasonably addressed, albeit that team has not addressed precision or accuracy impacts [19]. This may also be a pathway to solve a related problem: the conversion between different coordinate systems among devices.

More likely to impact the architecture is the likelihood to find new constraints or requirements when conducting wider field trials, such as speed or performance issues. The Kinect 2 features mentioned at the end of the last section, which point towards the need for an upstream configuration flow from Detectors to Adapters, is but an example of the kind of issues such trials may bring to light.

An area which we find particularly interesting as a source of future requirements is that of gesture detection by machine learning. We have assumed programmatic gesture detectors, ready to run, and not differences of operation between learning and detection, or continuously improving detection by providing continuous feedback. Exploring this may prove to be a rich source of information for improving the architecture. In its current state, our approach to have different operating modes in the Framework (acquisition and detection) may offer a pathway for this, by enlarging it with more modes (e.g., learning mode).

In spite of acknowledging the above shortcomings, the results from this early prototype implementation of the architecture and the development of the test application

demonstrate the feasibility of the proposed architecture. Specifically, it is a feasible way to develop gestural input applications using separation of concerns between gesture acquisition, gesture recognition, and application command. It is the scope of its applicability and the mutations it must go through to widen that scope that remain to be explored.

We hope to contribute to diminish the early obsolescence of gesture-controlled applications and, by separating gestures from their interpretation as commands, render gesture-controlled applications more accessible to users with special motion needs and to users from diverse cultural backgrounds.

ACKNOWLEDGMENTS

This work has been developed at INESC TEC in cooperation and funded by PT Inovação. Part of this work has been financed by the FCT – Fundação para a Ciência e a Tecnologia (Portuguese Foundation for Science and Technology) within project UID/EEA/50014/2013.

REFERENCES

[1] A. Valli, "The design of natural interaction," Multimed. Tools Appl., vol. 38, no. 3, pp. 295–305, 2008.

[2] J. Blake, Natural User Interfaces in. NET: WPF 4, Surface 2, and Kinect. Manning, 2011.

[3] M. Roupé, P. Bosch-Sijtsema, and M. Johansson, "Interactive navigation interface for virtual reality using the human body," Comput. Environ. Urban Syst., vol. 43, pp. 42–50, 2014.

[4] D.A. Norman, "Natural user interfaces are not natural," Interactions, vol. 17, no. 3, pp. 6-10, June 2010.

[5] L- Gamberini, A. Spagnolli, L. Prontu, S. Furlan, F. Martino, B.R. Solaz, M. Alcañiz, and J.A. Lozano, "How natural is a natural interface? An evaluation procedure based on action breakdowns," Personal and Ubiquitous Computing, vol. 17, no. 1, pp. 69-79, January 2013.

[6] L. Morgado, "Cultural awareness and personal customization of gestural commands using a shamanic interface," Procedia Computer Science, no. 27, pp. 449-459, 2014.

[7] E. Ernst, "Separation of concerns," in Proceedings of the AOSD 2003 Workshop on Software-Engineering Properties of Languages for Aspect Technologies (SPLAT), Boston, MA, USA, March 2003.

[8] M. Hesenius, and V. Gruhn, "MVIC – An MVC Extension for Interactive, Multimodal Applications," in Software Architecture - 7th European Conference, ECSA 2013, Montpellier, France, July 1-5, 2013. Proceedings, K. Drira, Ed. Berlin: Springer, 2013, pp 324-327.

[9] E. Curry, and P. Grace, "Flexible Self-Management Using the ModelView-Controller Pattern," IEEE Software, vol. 25, no. 3, pp. 84-90, May-June 2008.

[10] F. Carvalho, L. Morgado, and A. Coelho, "Shamanic interfaces for computers and gaming platforms," in INForum 2014 - Atas do 6º Simpósio de Informática, S. P. Abreu and J. P. Faria, Eds. Porto, Portugal: FEUP Edições, 2014, pp. 158-172.

[11] A. Angelopoulou, J. García-Rodríguez, A. Psarrou, M. Mentzelopoulos, B. Reddy, S. Orts-Escolano, J. A. Serra, and A. Lewis, "Natural User Interfaces in Volume Visualisation Using Microsoft Kinect," in New Trends in Image Analysis and Processing – ICIAP 2013, A. Petrosino, L. Maddalena, and P. Pála, Eds. Berlin, Germany: Springer, 2013, pp. 11-19.

[12] J. Bernardes, R. Nakamura, and R. Tori, "Design and implementation of a flexible hand gesture command interface for games based on computer vision," in Games and Digital Entertainment (SBGAMES), 2009 VIII Brazilian Symposium on, 2009, pp. 64–73.

[13] S. Spanogianopoulos, K. Sirlantzis, M. Mentzelopoulos, amd A. Protopsaltis, "Human computer interaction using gestures for mobile devices and serious games: A review," in 2014 International Conference on Interactive Mobile Communication Technologies and Learning (IMCL). Red Hook, NY, USA: IEEE, 2014, pp. 310-314.

[14] D. A. Bowman, E. Kruijff, J. J. LaViola Jr, and I. Poupyrev, 3D user interfaces: theory and practice. Addison-Wesley, 2004.

[15] L. Morgado, R. Rodrigues, A. Coelho, O. Magano, T. Calçada, P.T. Cunha, C. Echave, O. Kordas, S. Sama, J. Oliver, J. Ang, F. Deravi, R. Bento and L. Ramos, "Cities in citizens' hands," in Proceedings of the 6th International Conference on Software Development and Technologies for Enhancing Accessibility and Fighting Infoexclusion (DSAI 2015), in press.

[16] J. Alon, V. Athitsos, Q. Yuan, and S. Sclaroff, "A unified framework for gesture recognition and spatiotemporal gesture segmentation," Pattern Anal. Mach. Intell. IEEE Trans., vol. 31, no. 9, pp. 1685–1699, 2009.

[17] J. Suarez and R. R. Murphy, "Hand gesture recognition with depth images: A review," in RO-MAN, 2012 IEEE, 2012, pp. 411–417.

[18] E. Silva, N. Silva, and L. Morgado, "Model-Driven Generation of Multi-user and Multi-domain Choreographies for Staging in Multiple Virtual World Platforms," in Model and Data Engineering, 4th International Conference, MEDI 2014 Proceedings, Y.A. Ameur, L. Bellatreche, and G.A. Papadopoulos, Eds. Cham, Switzerland: Springer International Publishing, 2014, pp. 77 - 91.

[19] F. Cassola, H. Paredes, B. Fonseca, P. Martins, S. Ala, F. Cardoso, F. Carvalho, and L. Morgado, "Online-Gym: Multiuser Virtual Gymnasium Using RINIONS and Multiple Kinect Devices," in 2014 6th International Conference on Games and Virtual Worlds for Serious Applications (VS-GAMES), V. Camilleri, A. Dingli, and M. Montebello, Eds. Danvers, MA, USA: IEEE, 2014, pp. 25-28.