

Jorge Cardoso ([jccardoso@porto.ucp.pt](mailto:jccardoso@porto.ucp.pt)) and Nuno Rodrigues ([nrodrigues@porto.ucp.pt](mailto:nrodrigues@porto.ucp.pt))

Research Center for Science and Technology in Art (CITAR), Universidade Católica Portuguesa –  
Campus da Foz, Rua Diogo Botelho 1327, 4169-005, Porto, Portugal

- Autor apresentador: Jorge Cardoso
- Autor para contacto: Jorge Cardoso
- Palavras-chave: “Interaction System”, “Digital Art”, “Bluetooth”, “Interactive Installation”

# DiABlu: Digital Arts' Bluetooth

Jorge Cardoso ([jccardoso@porto.ucp.pt](mailto:jccardoso@porto.ucp.pt)) and Nuno Rodrigues ([nrodrigues@porto.ucp.pt](mailto:nrodrigues@porto.ucp.pt))

Research Center for Science and Technology in Art (CITAR), Universidade Católica Portuguesa –  
Campus da Foz, Rua Diogo Botelho 1327, 4169-005, Porto, Portugal

**Abstract** – Digital art installations can often gain from the capability of detecting the presence of people observing them. With this information, the artists can enhance the experience of who interacts with their work. While this detection can be made by means of web cameras or sensors, these systems are generally difficult to implement for people with a low knowledge of programming. We propose a system that uses bluetooth to do this detection and allows easy integration with applications often used by digital artists. The system also allows users to interact with the installation using their mobile devices. It's intended to be used in art installations by digital artists who wish to give their audience a new way to interact with their pieces.

**Index Terms** – Interaction System, Digital Art, Bluetooth, Interactive Installation.

## I. INTRODUCTION

Digital art installations can often gain from the capability of detecting the presence of people observing the installation. With this information, artists can enhance the experience of who interacts with their work. Sometimes, detecting the presence of people is even the only way to implement the conceptual meaning of the work of art.

There are many ways to detect the presence of people near an installation. Web cameras with more or less advanced detection techniques can be used, or a wide range of general purpose sensors combined with sensor control interfaces like the iCubeX system [1]. Implementing these solutions, however, is a distraction to the artist from more important aspects of the installation. Often, these systems mean building special structures to position web cameras and sensors and have to be fine tuned to every location.

Sometimes, however, it's not really necessary to have a very precise detection system, i.e., it doesn't matter if the system only detects part of the audience. In some cases, the artist is only concerned with providing a dynamic piece that reacts to the presence of people in a room, but it's not important that the piece recognizes exactly how many people there are.

In this paper, we propose a system for detecting the presence of people by detecting the presence of Bluetooth enabled devices. Our system allows easy integration with

applications used for building digital art installations, namely by our students at the School of Arts of the Portuguese Catholic University. The system is called "Digital Arts' Bluetooth – DiABlu"<sup>1</sup>.

Our goal was to develop a system that was easy to use and integrate with other applications, like Flash [2], Processing [3], Max/MSP [4], Pure Data [5], etc, by using the widely used Open Sound Control [6] (OSC) protocol.

Besides allowing the detection of bluetooth devices, the DiABlu System also allows users to interact, using their mobile devices, with the installation.

Throughout this paper we use the names of the main components of the DiABlu system, with the following meaning:

**Target Application:** The application that is developed by the final user and that needs information about bluetooth devices. This application can be developed in Max/MSP, Pure Data, Processing, Flash, or any other environment that supports the Open Sound Control (OSC) protocol.

**DiABlu Server:** The base DiABlu application that connects to the Target Application and provides information about the nearby Bluetooth devices.

**DiABlu Client:** Mobile application that connects to the DiABlu Server and allows the user to input keystrokes and text messages that will be delivered to the Target Application.

The remainder of this paper is organized as follows: section II presents the goals we had in mind when designing the DiABlu System; section III introduces some basic Bluetooth concepts for better understanding the rest of the paper; section IV describes the DiABlu Server; section V describes the DiABlu Client application; section VI describes some types of Target Applications and gives some examples of applications built to use our system; section VII explains the OSC messages used by our system, in detail; section VIII presents the implementation status of the system and section IX concludes.

---

<sup>1</sup> More information about this project can be obtained at <http://soundserver.porto.ucp.pt/diablu>

## II. DESIGNING THE DIABLU SYSTEM

### A. Requirements/Goals

The DiABlu System started out from the need to incorporate interaction in installations programmed in Max/MSP and Processing, via a mobile phone. Basically, we had two requirements: to be able to detect the presence of mobile phones and to be able to receive input from those mobile phones. We also wanted to be able to simulate the presence of mobile phones, so that testing the target application would be easier.

One of the main goals was to design a system that was easy to use by our students and by digital artists, in general. This meant that the system should be easily used with applications like Max/MSP, Pure Data, Eyesweb, Flash and Processing and that it should run in the Mac OS X and Windows platforms, since these are the most used applications and platforms in our school.

### C. High-Level Architecture

The high-level architecture of the DiABlu system is presented in Figure 1.

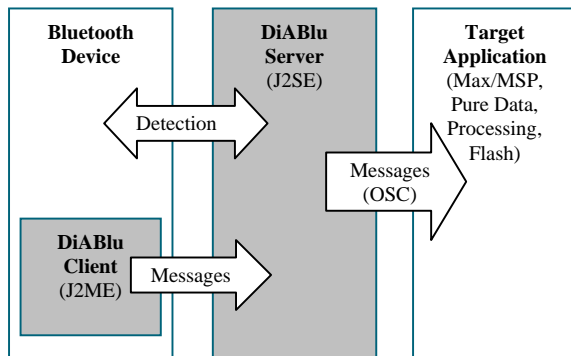


Figure 1 – High-level architecture of the DiABlu system

The shaded boxes represent the software components of the DiABlu System.

## III. BASIC BLUETOOTH CONCEPTS

Bluetooth is a wireless communication protocol intended to connect low power devices like portable digital assistants (PDA) and mobile phones. Bluetooth transmissions are omnidirectional, i.e., devices don't need an unobstructed line of sight to communicate, and have a nominal range of about 10 meters (class 3 devices).

Bluetooth devices are divided in three power classes. Class 1 is intended for larger devices, usually with AC power. Class 2 and 3 are intended for small, battery

powered devices. Table 1 lists the power rating and communications range of each power class. Mobile phones are usually class 3 devices.

| Class   | Power Rating | Range      |
|---------|--------------|------------|
| Class 1 | 100 mW       | 100 meters |
| Class 2 | 2.5 mW       | 20 meters  |
| Class 3 | 1 mW         | 10 meters  |

Table 1 – Bluetooth device power classes

Bluetooth devices are identified by their Universally Unique Identifiers (UUID) which are unique numbers associated with the Bluetooth hardware of the device. Besides having this identifier, Bluetooth devices may have (and generally do) “friendly names”, which are human readable names, normally configurable by the user.

When two Bluetooth devices communicate, three steps have to be accomplished: device discovery, service discovery and communication.

Before communication can occur, a device needs to find which devices are nearby. This process is called device discovery. In order to be discovered, devices need to be visible to other devices. This is usually user configurable, i.e., users can allow their devices to be discoverable or not.

After a device has found another to which it wishes to communicate with, it needs to know which services are offered by that device. There are several standard services like DialupNetworking, OBEXFileTransfer, Fax, BasicPrinting, etc. Applications can also define their own services. Services are identified by their UUID.

After finding a suitable service, communication can begin.

Besides the power classes, Bluetooth also defines types of devices (class of device, in the Bluetooth specification) which categorizes devices in classes like Computer, Phone, Network Access Point, Computer Peripheral, etc. Each class has a set of sub-classes. For example, the Computer class can be divided in Desktop, Server, Laptop, PDA, etc.

## IV. DIABLU SERVER

The DiABlu Server is the core of the DiABlu system. This application is responsible for detecting nearby bluetooth devices and informing the Target Application of the number of present devices and their UUIDs and names.

Basically, the DiABlu Server performs the following actions:

- Scan the environment for the presence of Bluetooth devices.

- Inform the Target Application of the nearby devices.
- Accepts Bluetooth connections from devices and receives data (keystrokes and text messages)
- Inform the Target Application of the data received.

All communication between the DiABlu Server and the Target Application is made using the Open Sound Control [6] (OSC) protocol.

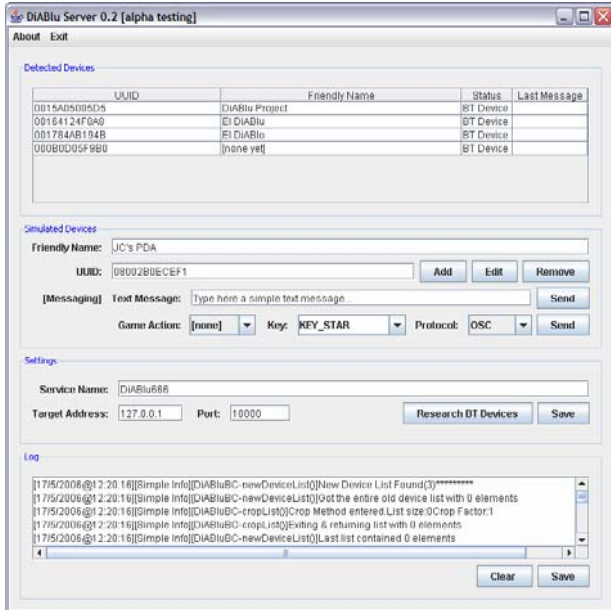


Figure 2 – Screenshot of the DiABlu Server interface

#### A. OSC Messages

The OSC messages sent by the DiABlu Server are described in more detail in section VII.

#### B. Simulator

An important aspect of the DiABlu System is the ability to simulate the presence and the input from Bluetooth devices.

Developing and testing applications that use information about the presence Bluetooth devices can be a difficult task. Reproducing the dynamics of the final environment in which devices enter leave is very difficult to accomplish with real devices – because of the number of devices needed and because of the rate of visibility change.

In order to facilitate testing and development, the DiABlu Server application also incorporates a device simulator. The application allows the user to simulate the entering and exiting of Bluetooth devices and the input (text messages and keystrokes) from those devices. From the point of view of the Target Application, these simulated devices behave the same way as the real ones.

## V. DIABLU CLIENT

The DiABlu Client is a mobile application developed in Java ME for devices that support the MID profile plus the Bluetooth Java API (JSR-82) [7]. This application allows the handheld user to interact with the Target Application via the DiABlu Server.

The DiABlu Client is a general application, in the sense that it is independent of the Target Application. The application is the same for every Target Application; there is no way to customize it, at this moment. Basically, it allows the user to:

- Discover nearby DiABlu Servers and connect to one. This makes it possible for the user to choose to interact with one from a number of nearby installations.
- Send a text messages to the Target Application.
- Send keystrokes to the Target Application.

Figure 3 shows the screen diagram for the DiABlu Client application.

There are three main screens in the DiABlu Client Application: the Search Screen, the Msg Screen and the Keys Screen.

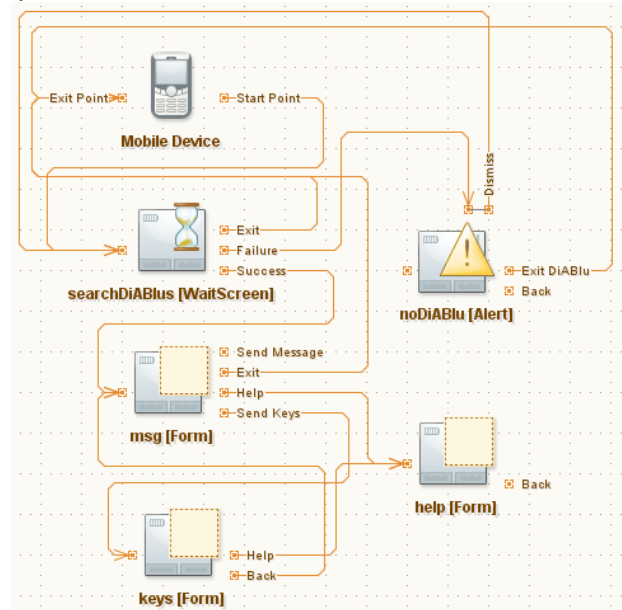


Figure 3 – Screen flow diagram for the DiABlu Client

The Search Screen is a waiting screen so that the DiABlu Servers can be discovered. To discover a DiABlu Server, the DiABlu Client first searches for Computer class devices. For all Computer devices, the application searches for a specific service UUID. If this service is

found, then the device has a DiABLu server running. The services names (set by the user in the DiABLu Server interface) are shown to the user in the Msg Screen.

The Msg Screen allows the user to send a text message to one of the DiABLu Servers discovered. The user can choose to which DiABLu Server to send the message (if there are more than one).

The Keys Screen allows the user to send keystrokes to the DiABLu Server.

## VI. TARGET APPLICATION

The Target Application is any application, developed by the final user of the DiABLu System, that is capable of receiving data via the OSC protocol. The Target Application receives updated information about the names, IDs and number of bluetooth devices near the computer running the DiABLu Server. It also receives the key codes that a given DiABLu Client's user pressed while connected to the DiABLu System.

### A. Usage Scenarios

There are three typical high-level use cases for the DiABLu System:

**No Interaction:** In this use case, the Target Application only needs to know how many devices there are in the vicinity and/or their names.

The installation does not have any direct interaction capability; it just reacts to the presence of bluetooth devices.

**Shared Interaction:** This use case represents all applications that besides reacting to the presence of bluetooth devices, allow their users to directly interact with the application. Interaction is done by means of the DiABLu Client application, which must be installed in the device, and is limited to sending keystrokes and text messages. There are no restrictions, imposed by the Target Application, on the number of users that may be interacting simultaneously with it.

**Exclusive Interaction:** This is similar to the Shared Interaction use case, except that the Target Application limits the number of users directly interacting, to one. This is a typical use case for navigational interfaces in which at most one user may be navigating at a time.

### B. Example Target Applications

**Nulltidão (No Interaction)** – This is a video installation developed by João Cordeiro [8] that plays with the concepts of crowd and individuality. The installation uses only the information about the number of nearby Bluetooth devices as estimation of the number of people watching it. The installation consists of a video-wall

displaying a video captured by a web camera installed at the location. The video is manipulated so that it shows only regions of the current frame combined with an initial frame. This initial frame is taken from the location when there are no people around. The number of regions displayed depends on the number of devices present.

**Public Puzzle (Shared Interaction)** – This is a video installation that consists of a block puzzle that users can play with. Instead of using a still image for the puzzle, it uses frames taken from a web camera mounted at the location. Playing with this game is a matter of moving the black piece up, down, left or right, trying to put the nine pieces in the right order. Several users can play at the same time, issuing commands to the black piece. In order to play, users must have the DiABLu Client application installed.

**Jukebox (Exclusive Interaction)** – This application allows users to select a music file to play, just like a physical jukebox. The application's interface is displayed on a video-wall in a public place. Users can install the DiABLu Client application on their cell-phones and use it to control the jukebox. The jukebox application guarantees that only one user at a time can browse the music library and choose the file to play. This is done via timeouts – if a user starts controlling the interface, other users are not allowed until a fixed amount of time has passed since the last interaction.

## VII. OSC MESSAGES

The following are all OSC messages implemented by the DiABLu Server. Some of the messages are redundant, i.e., they transmit the same information. They differ only in the way that they must be handled by the target application.

We chose to provide redundant messages so that the target application programming could be facilitated.

We present the format of the OSC messages in the form [MessageName] [Type-0]:[ParameterName-0] [Type-1]:[ParameterName-1] (...) [Type-n]:[ParameterName-n], where [MessageName] is the OSC Address Pattern, [Type-x] is the OSC Type Tag String (“s” for string, “i” for int32, “f” for float) and [ParameterName-x] is the description of the parameter.

**/DeviceIn** – This message is sent for every new device that is detected by the server. If two devices enter at the same time, two messages will be sent. The format of the message is: /DeviceIn s:[UUID] s:[Friendly-Name].

**/DeviceListIn** – This message is similar to the previous, except that, if two, or more, devices enter at the same time, only one message is sent. The message contains the UUID and friendly names of all devices that entered. The format

of this message is: /DeviceListIn s:[UUID-0] s:[Friendly-Name-0] s:[UUID-1] s:[Friendly-Name-1] (...) s:[UUID-n] s:[Friendly-Name-n].

**/DeviceOut** – This message is sent for every device that ceases being detected by the server. If two devices leave at the same time, two messages will be sent. This message is the counterpart of the /DeviceIn message. The format of the message is: /DeviceOut s:[UUID] s:[Friendly-Name].

**/DeviceListOut** – This is the counterpart of /DeviceListIn. If two, or more, devices leave at the same time, only one message is sent. The message contains the UUID and friendly names of all devices that left the vicinity. The format of this message is: /DeviceListOut s:[UUID-0] s:[Friendly-Name-0] s:[UUID-1] s:[Friendly-Name-1] (...) s:[UUID-n] s:[Friendly-Name-n].

**/MessageIn** – This message is sent whenever a user sends a text message via the DiABlu Client application. The format is: /MessageIn s:[UUID] s:[Friendly-Name] s:[Message-Text].

**/KeyIn** – This message is sent when the user presses a key in the DiABlu Client application. This message contains also the game action associated with the key that was pressed, if any game action is associated. Game actions are actions like UP, DOWN, LEFT, RIGHT, FIRE, GAME\_A, GAME\_B, which different mobile phones map to different keys. This way, applications don't need to have a static association between key codes and game actions. The format of the /KeyIn message is: /KeyIn s:[UUID] s:[Friendly-Name] s:[Key-Pressed] s:[Game-Action]. The [Key-Pressed] parameter is a string representing the key from the ITU-T keyboard: "KEY\_NUM0", "KEY\_NUM1", (...), "KEY\_NUM\_9", "KEY\_STAR" or "KEY\_POUND". The [Game-Action] parameter is also a string with the name of the game action associated with the pressed key: "UP", "DOWN", "LEFT", "RIGHT", "FIRE", "GAME\_A", "GAME\_B" or "NONE" if no game action is associated with that key.

**/DeviceList** – The /DeviceList message is sent every time a device enters or leaves the vicinity of the server. This message contains the list of all devices that are currently visible by the server. The format is the following: /DeviceList s:[UUID-0] s:[Friendly-Name-0] s:[UUID-1] s:[Friendly-Name-1] (...) s:[UUID-n] s:[Friendly-Name-n].

**/NameChanged** – The /NameChanged message is sent when the friendly name of a device changes. This message is important because it allows devices that don't have the DiABlu Client application installed, to still be able to have some basic direct interaction capabilities. The Target Application can be programmed to react to certain friendly names, which means that users could interact with it by changing the name of their devices. The format of this

message is: /NameChanged s:[UUID] s:[Friendly-Name], where [Friendly-Name] is the new friendly name of the device identified by [UUID].

**/DeviceCount** – This message is sent every time a device enters or leaves the vicinity of the server. This message contains only the number of devices currently visible by the server. The format is: /DeviceCount i:[Number-Of-Devices].

Messages are sent only at the end of the Bluetooth discovery cycle, which can last a variable amount of time, depending on the number of nearby devices.

Allmost all messages (except for the /DeviceCount message) have the [UUID] and [Friendly-Name] parameters so that applications only have to maintain the minimum state information needed. The friendly name could be looked up by the target application, using the UUID, but this would mean that the application would have to maintain data arrays, which can be difficult to program in environments like Max/MSP, Pure Data and such.

## VIII. IMPLEMENTATION STATUS

The DiABlu Server has been implemented for the Microsoft Windows and Mac OS X platforms. We are finishing the implementation of the DiABlu Client application.

### A. Bypassing Bluetooth Idiosyncrasies

We tested the discovery of devices by the DiABlu server and found some Bluetooth related phenomena:

1. In some discovery cycles, some devices are not discovered, even if they were already discovered in a previous cycle and device hasn't moved. This has the effect of sending a /DeviceOut followed by a /DeviceIn message to the Target Application even in the case that the device hasn't moved.
2. The friendly name is only accessible the second time the device is discovered. This means that the Target Application receives a /DeviceIn followed by a /NameChanged message every time a new device is discovered.
3. Sometimes, passing devices are detected. In these cases, a /DeviceIn message followed immediately by a /DeviceOut message is sent to the Target Application.

To resolve these situations we had to adapt the server so that two discovery cycles are needed before a device is removed from the list. This means that it takes longer for the Target Application to be informed of a device out event, but we get less "false removals".

We also changed the server so that it waits for the friendly name to be accessible before sending a /DeviceIn message.

We chose not to address the third phenomenon because we believe that detecting passing devices can be an important functionality for the Target Application. Besides, it can easily be resolved within the Target Application.

#### IX. CONCLUSIONS AND FUTURE WORK

This paper presented the DiABlu System, a Bluetooth detection and interaction system for the digital arts community.

We have described the general functionality and architecture of the system and typical use cases for this kind of application.

We plan to use the the DiABlu System on projects developed at the School of Arts to gain experience and insight on the kind of functionality needed by our users in order to further develop and enhance the system.

In the short term, we plan to add bidirectional communication between the DiABlu Client and the Target Application. We also plan to extend the detection range by using the DiABlu Clients as detection nodes and

transmitting the information about the detected devices to the DiABlu Server.

#### REFERENCES

- [1] Infusion Systems, "I-CubeX", <http://infusionsystems.com/catalog/index.php> [accessed 06 June 2006].
- [2] Macromedia, "Flash", <http://www.macromedia.com> [accessed 06 June 2006].
- [3] B. Fry and C. Reas, "Processing", <http://processing.org> [accessed 06 June 2006].
- [4] Cycling74, "Max/MSP", <http://www.cycling74.com> [accessed 06 June 2006].
- [5] M. Puckette, "Pure Data: another integrated computer music environment", *Proc. the Second Intercollege Computer Music Concerts*, pp 37-41, 1996.
- [6] M. Wright and A. Freed, "OpenSound Control: A New Protocol for Communicating with Sound Synthesizers", *Proceedings of the 1997 International Computer Music Conference*, 1997.
- [7] Java Community Process, "JSR 82: Java™ APIs for Bluetooth", <http://www.jcp.org/en/jsr/detail?id=82> [accessed 06 June 2006].
- [8] J. Cordeiro, "Nulltidão", Multimedia Programming Course Project, 2006, <http://teaching.jorgecardoso.org/pm/> [accessed 06 June 2006].