ToyVision: A Toolkit to Support the Creation of Innovative Board-Games with Tangible Interaction

Javier Marco
Madeira-ITI
University of Madeira, Portugal
javier.marco@m-iti.org

Sandra Baldassarri, Eva Cerezo
GIGA Affective Lab
Computer Science Department,
Engineering Research Institute of Aragon (I3A)
Universidad de Zaragoza, Spain
{sandra, ecerezo}@unizar.es

ABSTRACT

This paper presents "ToyVision": a software toolkit developed to facilitate the implementation of tangible games in visual-based tabletop devices. Compared to other toolkits for tabletops which offer very limited and tag-centered tangible possibilities, ToyVision provides designers with tools for modeling and implementing innovative tangible playing pieces with a high level of abstraction from the hardware. For this purpose, a new abstraction layer (the Widget layer) has been included in an already existing tabletop framework (ReacTIVision), providing the host application with high processed data about each playing piece involved in the game. To support the framework application, a Graphic Assistant tool enables the designer to visually model all the playing pieces into tangible tokens that can be tracked and controlled by the framework software. As a practical example, the complete process of prototyping a tangible game is described.

Author Keywords
Tabletop; toolkit; tangible; games; playing pieces; widget; design.

ACM Classification Keywords
H.5.2 Information Interfaces and Presentation: User Interfaces—Input devices and strategies, Interaction styles

General Terms
Design.

INTRODUCTION

The increasing popularity of tabletop devices is bringing in a new generation of entertainment and game applications that mix traditional face to face gaming with computer augmentation on the active surface [29]. At present, most of these tabletop games are based on multitouch interaction [5], [1] and players manipulate virtual representations of the playing pieces by dragging their fingers on the table. However, several tabletop devices are not only capable of detecting user fingers and hands, but also of supporting the identification and tracking of conventional objects placed on the surface. This also enables the use of physical playing pieces in tabletop games [13] [22], reinforcing the emotional impact that the activity has on the players [17] [15].

Although the tabletop hardware can be used to detect and track fingers and objects on the surface, the development of an application is not easy since it usually involves having to "hardcode" complex algorithms to process raw data from tabletop in order to detect and track each playing piece manipulated on the active surface. This situation results in a gap between the tangible interaction design process and the corresponding implementation tasks, i.e., between designers and developers. To tackle the problem, several toolkits have emerged with the aim of isolating the hardware complexities of a tabletop system. These toolkits offer high processed data of user interactions on the table, both tactile and through objects, but unfortunately in a very basic form: tangible interaction is described through simple events (object placed, moved or removed). This simplistic approach constrains the designer to using playing pieces that can merely be moved on the table, limiting the exploration of richer tangible interaction possibilities.

This paper proposes a toolkit for the prototyping of tangible tabletop games involving playing pieces that can be manipulated by the players and also controlled by the system in a great variety of ways. ToyVision lowers the threshold of implementing a tangible game, and so enables designers to access tasks that previously required greater engineering and coding skills.

The paper first examines the current state of the art in tabletop and tangible toolkits. An application scenario is then presented, and the problems involved in building complex tangible interactions with playing pieces using current tabletop toolkits are identified. Next, the ToyVision tools (a Graphic Assistant and a Framework application) are detailed. The coding stage of the application scenario is then sketched. Finally, the conclusions and proposals for future work are outlined.

RELATED WORK

Due to the recent success of multitouch devices, several software frameworks have been created to be used as toolkits for the rapid development of tabletop applications independent of the hardware. While earlier multitouch frameworks merely informed developers about raw-tactile events (finger added, moved, taken away from the table) [33] [2] [23] [32] [24], recent frameworks also inform about finger gestures by sending high abstraction events (zoom, rotation, delete…) [14] [11] [27].

The addition of tangible interaction to tabletop surfaces requires conventional objects placed on the interactive surface area to be identified and tracked. In visual based multitouch surfaces [31], this can be achieved by attaching a printed visual tag (fiducial) [6] on the base of the object. Fiducial recognition and tracking is based on a simple principle: a fiducial is composed of a unique distribution of reflective and non-reflective to infrared light (IR) areas, and the visual software detects the reflective areas as white blobs. Using this technique, several multitouch tabletop frameworks also support tangible interaction with tagged objects [3] [28] [4] [34].

The software architecture of tabletop frameworks has been described by Echtler and Klinker [9] using four layers, from the lowest to the highest abstraction: Hardware, Calibration, Event Interpretation and Widget layers. A framework that follows a layered architecture (see fig. 1) offers at least the Hardware Layer in order to hide the visual hardware and blob recognition algorithms. Optionally, the framework can add the Calibration layer to correct the position coordinates of each detected blob due to camera optics aberrations. In the Event Interpretation layer, the framework keeps track of multitouch events ("finger" added, moved or removed from the tabletop surface) and tangible events ("fiducial" added, moved or removed from the tabletop surface). Finally, by adding the Widget layer, the framework may associate sequences of events in tabletop regions with predefined actions in the tabletop application.

By separating the framework software from the development environment, tabletop applications can be translated to other devices based on different hardware or even based on a different framework. This is achieved by the use of standard communication protocols between the framework and the tabletop host application. In this context, the TUIO protocol [19] has become very popular and has been adopted by most tabletop frameworks [28] [4] [33] [23]. However, the TUIO protocol is designed to transmit processed data from the Event Interpretation Layer (EIL): the framework sends, embedded in TUIO packets, multitouch and tangible events to the tabletop application (see fig. 1). TUIO support of tagged objects is limited to three simple events: add, remove and move/rotate.

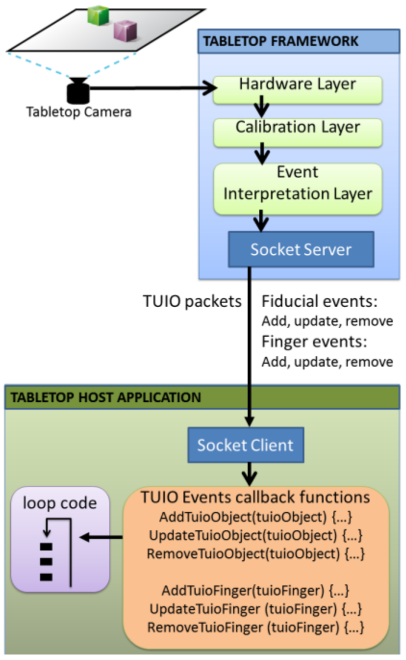

TABLETOP FRAMEWORK
Tabletop Camera
Hardware Layer
Calibration Layer
Event Interpretation Layer
Socket Server
TUIO packets
Fiducial events:
Add, update, remove
Finger events:
Add, update, remove
TABLETOP HOST APPLICATION
Socket Client
loop code
TUIO Events callback functions
AddTuioObject(tuioObject) {...}
UpdateTuioObject(tuioObject) {...}
RemoveTuioObject(tuioObject) {...}
AddTuioFinger(tuioFinger) {...}
UpdateTuioFinger (tuioFinger) {...}
RemoveTuioFinger (tuioFinger) {...}

Figure 1. Software architecture of tabletop frameworks based on an Event Interpretation Layer (EIL) .

More specific toolkits for supporting the design of Tangible User Interfaces (TUI) are mostly hardware-focused and aim to isolate designers and developers from the intrinsic complexity of managing several kinds of sensors and actuators embedded in objects. Commercial or open-source hardware toolkits such as Phydgets [10] or the Tower System [26] propose a limited set of electronic components and software libraries that can be easily assembled to provide any physical object with sense and action. In order to isolate developers from electronic embedded components, some other TUI toolkits [7] [12] [20] [8] [21] also include a more complex software layer supported by a Graphic Assistant tool, which introduces more of a "design thinking" approach to TUI toolkits as opposed to the "implement thinking" characteristic of previous toolkits.

The work presented here contributes to the state of the art of tabletop toolkits with the addition of a Widget layer in an already existing tabletop framework based on an EIL architecture, and a Graphic Assistant to model the tangible interaction. The new Widget layer and the Graphic Assistant help designers to implement advanced tangible interaction in playing pieces for tabletop games with a high level of abstraction from the hardware. The intrinsic difficulty of designing a tangible toolkit which includes a Widget abstraction layer lies in the huge variety of different existing objects and different manipulations that have to be modeled in the toolkit [30]. Nevertheless, limiting the scope to board games, the range of playing pieces should be manageable. Starting from the concept of "Token" defined by Holmquist et al. [16] as any object used to represent some stored digital information, we have proposed a classification of board-game playing pieces into four types of Tokens [25]:

- **Simple Tokens** are the most common in board games (Ludo, Go, Checkers…). They are pure symbolic objects (chips, marbles…) mostly used to represent players.
- **Named Tokens** have a very specific role in the game (e.g., the pieces in the Chess game have different roles). They are iconic playing pieces as their physical appearance is used to represent their function in the game.
- **Constraint Tokens** are a combination of a Named Token and one or more Simple Tokens. The Named Token acts as a physical constraint of the manipulations of the Simple Tokens. The way these simple tokens are manipulated within the constraint is related with the status of the playing piece in the game (e.g., the small pieces inside the big Trivial Pursuit™ playing piece are used to represent each player's achievements).
- **Deformable Tokens** are playing pieces without a constant shape (e.g. clay, fabrics…) which are mainly used in handcraft and building games.

The present work also roots on this classification to model each playing piece involved in a tabletop game. Moreover, the ToyVision toolkit also enables a Token to be made "active", i.e. to be manipulated not only by the player, but also by the computer system.

APPLICATION SCENARIO

In order to show how ToyVision facilitates the creation of tabletop games for designers, this section presents a game concept involving complex playing pieces, some of them active. This concept will illustrate some problems that current EIL tabletop toolkits are not able to tackle.

The game chosen is "Dragon's Cave", a board game for children based on the Dungeons & Dragons® role-playing games. In the "Dragon's Cave" game, the players play the role of a hero who has to find a sword to kill a dragon that lives in a cave (see fig.2). The game is composed of the following playing pieces:

- One or more **Heroes** (see fig.2-1). Players move their Heroes in turns. A virtual dice (see fig.2-2) represents the distance that the playing piece can move on the board in any turn.
- A **Dragon** (see fig.2-3). This playing piece slowly rotates during the game, looking for Heroes. When the Dragon looks straight at a Hero, it launches a virtual fireball and the player is removed from the game. The Dragon rotates by means of a small servo motor on the base of the toy.
- A **Chest** (see fig.2-4 and fig.3). This contains the Sword playing piece needed to kill the Dragon. The Chest automatically opens by means of a small servo motor on the back when a Hero gets sufficiently close. When the Chest is open, the Hero gets the sword and can move towards the Dragon to kill it.
- The **Sword** (see fig.3 right). This is a small metallic playing piece that remains inside the Chest until a Hero opens it. Then, the player places the Sword in the Hero's arms to represent that the Hero is able to kill the Dragon.

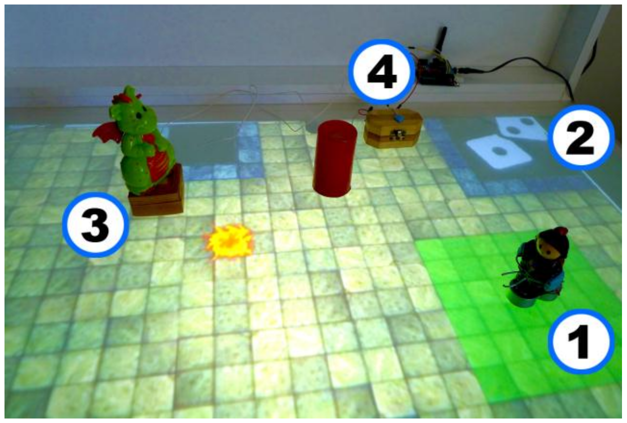


Figure 2. The "Dragon's Cave" game and its playing pieces. 1: The Hero. 2: Dice. 3: Dragon. 4: Chest.

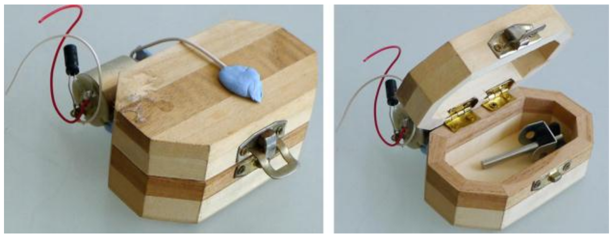


Figure 3. Chest toy. Left: closed. Right: open with the miniature Sword inside.

So that the manipulations of the toys may be tracked by the tabletop software toolkit, a fiducial has to be glued to the base of each playing piece. However, when a tabletop software framework based on EIL architecture is used to implement the game, the fiducial technique cannot solve the design of the Hero and the Sword playing pieces because the framework merely sends events of a playing piece when it is placed, moved, or removed from the tabletop. There is no such event for "the Hero has taken the Sword", and thus a hardware-specific coding solution needs to be created for this situation, which may be beyond the capability of the designer.

Moreover, the designer also has to implement mechanical and electronic solutions using actuators in order that the Dragon can rotate and the Chest be automatically opened by the system during the game. An EIL framework does not have hardware abstractions for toy sensors and actuators, and so it will be very difficult for a designer to create the specific code required to control each electronic device. Once again, this task should probably be delegated to an engineer or a programmer.

The next section describes the ToyVision toolkit and the different approaches it provides towards modeling tangible interaction in playing pieces such as those involved in the "Dragon's Cave" game.

TOYVISION TOOLKIT

ToyVision is composed of two software tools: a tabletop Framework and a Graphic Assistant application (see figure 4). The ToyVision **Framework** takes an existing EIL tabletop framework (ReacTIVision in this case) and expands it with a new **Widget layer**. This layer is responsible for managing all tabletop finger and fiducial events and processing them into high abstract events directly related with each playing piece involved in the game. These events are coded in XML format and sent to the **Host Game application** through a TCP-IP socket. Furthermore, the Widget layer also manages high abstract commands received from the **Host Game application** to control active toys. These commands are translated into low level actuator orders and sent to an Analog/Digital (A/D) hardware converter (we use the open-source Arduino platform [18] for this function) connected to the Toolkit through a Bluetooth wireless connection. The Widget layer is able to translate low and high abstract events and actuator commands thanks to the information provided in the **Toys.XML configuration file**, previously generated by the **Graphic Assistant** tool. This Assistant is used by the designer to graphically model all the playing pieces as tangible Tokens during the design process of the game.

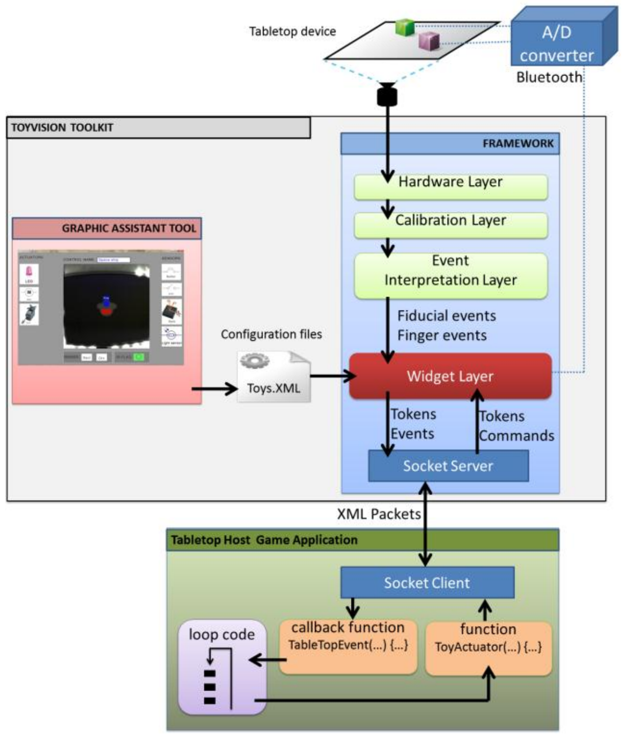


Figure 4. ToyVision toolkit architecture.

The Graphic Assistant tool is detailed below, together with the process that enables the playing pieces involved in the "Dragon's Cave" game to be modelled. This is followed by a description of the Framework, especially the functionalities of the Widget layer which represent the innovative content of the framework.

ToyVision Graphic Assistant Tool

The ToyVision Graphic Assistant has been designed following a similar approach to that of existing graphic tools included in most popular development environments oriented to coding WIMP-based applications. These tools enable interface designers to graphically arrange controls on an application frame and to define attributes for each control, thus facilitating the coding of the interface. ToyVision Graphic Assistant allows the designer to model, in a simple manner, all the data needed by the Toolkit (in particular by the Widget layer) to translate between low and high abstract data related with the tabletop and the playing pieces. The process of modeling playing pieces is now shown using the "Dragon's Cave" application scenario.

The Chest and the Dragon playing pieces belong to the Named Token category as they are iconic toys with a very specific function in the game. Thus, to model the Dragon toy, the designer places the toy on the table surface and clicks on the Named Token icon on the main menu of the application. The Token Configuration screen appears (see fig.5), letting the designer see the image sent by the tabletop camera (see fig.5-1). By activating the rectangular or circular buttons (see fig.5-2), the designer graphically draws the available area on which to glue a fiducial on the toy's base. The Token Configuration screen also provides a list of sensors (see fig.5-3) and a list of actuators (see fig.5-4) (new electronic components can be easily added to these lists by editing an external XML file included in the Graphic Assistant tool installation folder). The Dragon toy has a servo motor to rotate the figure during the game. To add this actuator, the designer drags the servo motor actuator icon into the visual camera feedback area. At that moment, a configuration window appears requiring data about the added component (see fig.6).

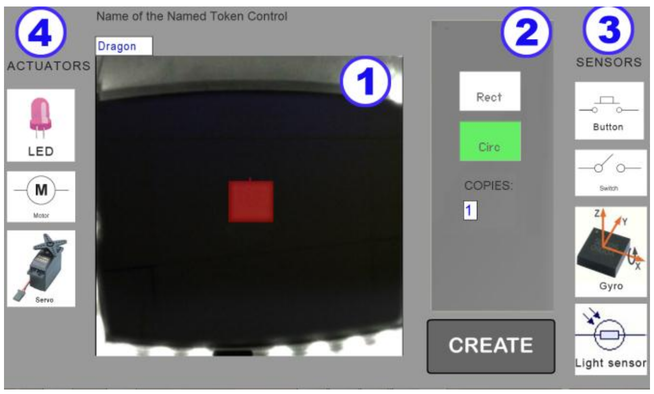
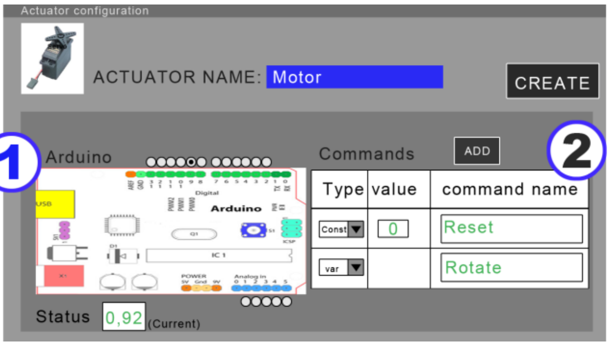


Figure 5. Named Token Configuration screen.



ACTUATOR NAME: Motor

Type	value	command name
Const	0	Reset
var		Rotate

Status 0,92 (Current)

Figure 6. Actuator Configuration screen.

The Actuator Configuration screen enables the designer to specify the low level details of the electronic component, and how these will be referred to at a high abstraction level. First, the designer gives a name to the actuator (in this case, “Motor”). Then, the designer links it with one of the analog/digital input/output terminals in the Arduino platform (see fig.6-1). At that moment, the designer can physically connect the Arduino device to the computer and check the actuator behavior using the “current status” text field by giving it a 0.0 to 1.0 value (all electric currents are normalized). Using these current values, the designer provides a list of different high level commands with a meaningful name in order to be referenced during the implementation stage. Figure 6-2 shows two commands for the Dragon toy: the “Reset” command will position the dragon in the initial orientation at the beginning of the game, and the “Rotate” command will be used to slowly rotate the toy during the game.

In the case of the Hero playing piece, the designer first has to solve technically how the Framework will “sense” whether the Sword is placed on the toy. This can be done by providing the playing piece with an electric switch component that closes a circuit when the Sword toy is placed on the arms of the Hero toy, causing an IR LED on the base of the toy to come on (see fig. 7). The light emitted by the LED can be detected by the Hardware layer of the Framework as a circular white blob and the Event Interpretation layer will trigger a “finger” event.

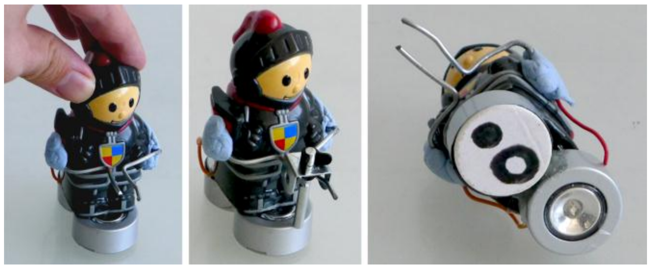
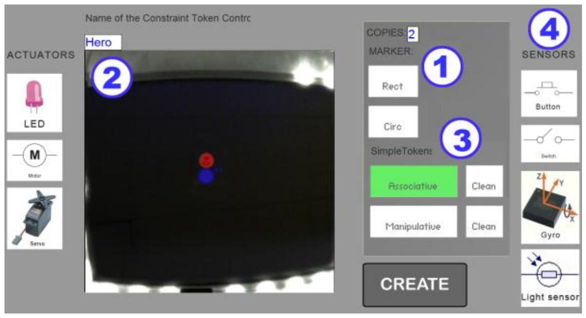


Figure 7. Hero playing piece. Left: without the Sword, the electric circuit is open. Center: placing the metal Sword between the two electric terminals closes the circuit. Right: The IR LED on the base of the toy comes on when the circuit is closed.

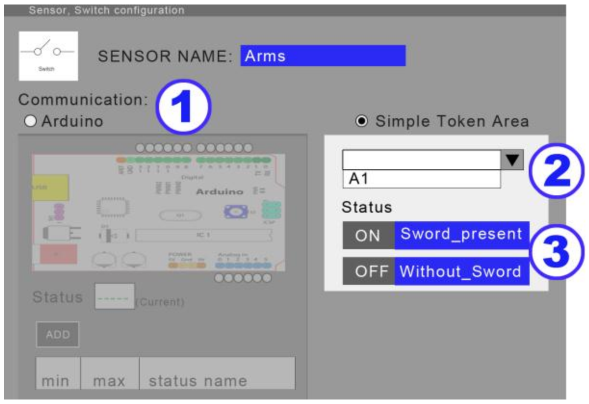
This design solution can be modeled in the Graphic Assistant using the Simple Token and the Constraint Token categories. The IR LED on the base of the toy is a constrained Simple Token that can simply appear or disappear on the base of the Hero playing piece (depending on the presence or absence of the Sword toy). Thus, to model the Hero playing piece, the designer first places the toy on the table surface and clicks the “Simple Token” icon on the Graphic Assistant main menu. The Simple Token Configuration screen appears. The designer gives a name to the Token (“Sword” in this case) and sets an approximate range of maximum and minimum sizes for the light spot created by the IR LED on the base of the Hero toy. Finally, the designer models the Hero playing piece by clicking on the “Constraint Token” icon on the application main menu. The Configuration screen appears (see fig.8). As in the case of a Named Token, the designer first uses the marker tools (see fig.8-1) to draw on the camera feedback image (see fig.8-2) the area on the base of the toy where the fiducial has to be glued. Then, using the “Simple Tokens” tools (see fig.8-3), the designer draws the area on the base of the toy that belongs to the IR LED. There are two kinds of Simple Token areas: Associative (representing areas where a Simple Token can only appear or disappear) and Manipulative (representing areas in which the Simple Token is always present but is able to move within the area). In the case of the Hero playing piece, the Simple Token area is an Associative area, as the IR LED makes the Simple Token on the base appear and disappear. Then, the designer adds a switch sensor by dragging its icon (see fig.9-4) to the central area of the interface, and the sensor configuration screen appears (see fig.9).



Name of the Constraint Token Contr: Hero

COPIES: 2

Figure 8. Constraint Token Configuration screen.



SENSOR NAME: Arms

Communication: ○ Arduino ● Simple Token Area

A1

Status

ON	Sword_present
OFF	Without_Sword

Status ----- (Current)

min	max	status name

Figure 9. Sensor Configuration screen.

The added sensor (named “Arms”) can send its status to the Framework through the Arduino platform (see fig.9-1), or by associating it with a Simple Token area (see fig.9-2), as in this case. The designer gives meaningful names to the different status that the sensor can have (see fig.9-3), which will be used during the game implementation stage.

Once all the playing pieces involved in the game have been modeled, the designer exports the project to a local folder. Two files are created during the export process: an Adobe PDF document containing all the fiducial markers ready to be printed, cut and glued on the base of each playing piece, and the *Toys.XML* configuration file which contains all the information needed by the Widget layer to be able to process all the low level information related with the tabletop hardware and the Arduino platform into high abstract data.

ToyVision Framework

The ToyVision Framework expands the open-source ReacTIVision framework by adding a new Widget layer to its EIL architecture. The Widget layer is responsible for processing all the events received from the ReacTIVision EIL and from the Arduino platform, finding relations between them and the playing pieces pre-modeled as Tokens in the Graphic Assistant tool. To support this functionality, the Widget layer uses all the information contained in the *Toys.XML* file. This process is now illustrated with the “Dragon’s Cave” application scenario.

The following XML code specifies the Dragon playing piece as it is stored in the *Toys.XML* file.

```
<NamedToken name="Dragon" fidID="2">
        <actuator name="Motor" terminal="9">
                <command name="Reset" type="const"
                value="0"/>
                <command name="Rotate" type="var"/>
        </actuator>
</NamedToken>
```

The “NamedToken” tag gives the Token category and contains the attributes related with the Dragon playing piece name and its assigned fiducial (fidID). In this way, each time the EIL sends an event (add, move or remove) related with the fiducial with ID=2, the Widget layer recognizes that this event belongs to the Dragon. The next XML tag (“actuator”) informs that the Dragon playing piece has an embedded electronic actuator, named “Motor”, which is connected to the 9th terminal of the Arduino platform. The next two XML tags (“command”) inform that the Motor actuator can receive two possible orders from the game: “Reset” and “Rotate”. If the Widget layer receives an order to “Reset the Motor”, the Widget layer translates it to “set 9th Arduino terminal to 0 volts”; and if the Widget layer receives an order to “rotate (value=0.1) the Motor”, it sets the 9th Arduino terminal to 0.1 volts, which will slightly rotate the Dragon figure.

The following XML code specifies the Hero playing piece:

```
<SimpleTokens name="Sword" size="561"
tolerance="378"/>
<ConstraintToken name="Hero" fidID="0,1">
        <AssociativeArea name="a0" radius="0.04"
        distance="0.07" angle="3.12"/>
</ConstraintToken>
```

The first tag (“SimpleTokens”) gives instructions to the Widget layer to identify the IR Spot lights on the base of the Hero playing pieces. Each time the EIL sends a “finger” event (added, moved or removed), the Widget Layer compares its size and tolerance values provided in the XML tag attributes. If this is positive, the low abstract event can be translated into a high abstract one: “a Hero has got or lost a Sword”. The next tag helps to find relationships between the Heroes and the Sword. The “ConstraintToken” tag provides the name (“Hero”) and fiducial ID of the playing piece (in this case there are two IDs for two possible players). The next tags list all the Constraint areas. The “Hero” has one “AssociativeArea” (named “a0”), and the tag attributes inform about its size and position in polar coordinates in relation to the center of the fiducial area. With these data, the Widget layer finds spatial relations between “Swords” and “Heroes” events. When a relationship is found, the Widget layer composes a high abstract event directly related with a specific Hero playing piece (e.g., “Player 1’s Hero has got the Sword”).

Finally, the Widget layer codes the high level events in XML format and sends them through the socket to the Host Game application. The XML message has all the data needed for the Host Game application to extract which toy has triggered the event, and its new status. For example, the “Player 1’s Hero has got the Sword” event will be coded in XML this way:

```
<EVENT toyName="Hero" copy="1" eventype="sensor"
                               sensorName="Arms">
  <Hero posX="0.4" posY="0.2" angle="2.2">
    <Arms status="Sword_present"/>
  </Hero>
</EVENT>
```

Given the high processed events that the Widget Layer sends to the Host application, coding a tangible tabletop game is a very straightforward task that does not require the developer or even the designer to have advanced programming skills, as the next section will show.

CODING A TANGIBLE TABLETOP GAME

Figure 10 sketches the code structure of the Host Game for our application scenario, the “Dragon’s Cave” game, comparing the different code implementations required using an EIL toolkit and our ToyVision Toolkit. In both situations, as the Framework and the Host application are independent applications, practically any computer development environment can be used to code as it only requires a TCP-IP socket client in the Host to connect it with the Framework application.

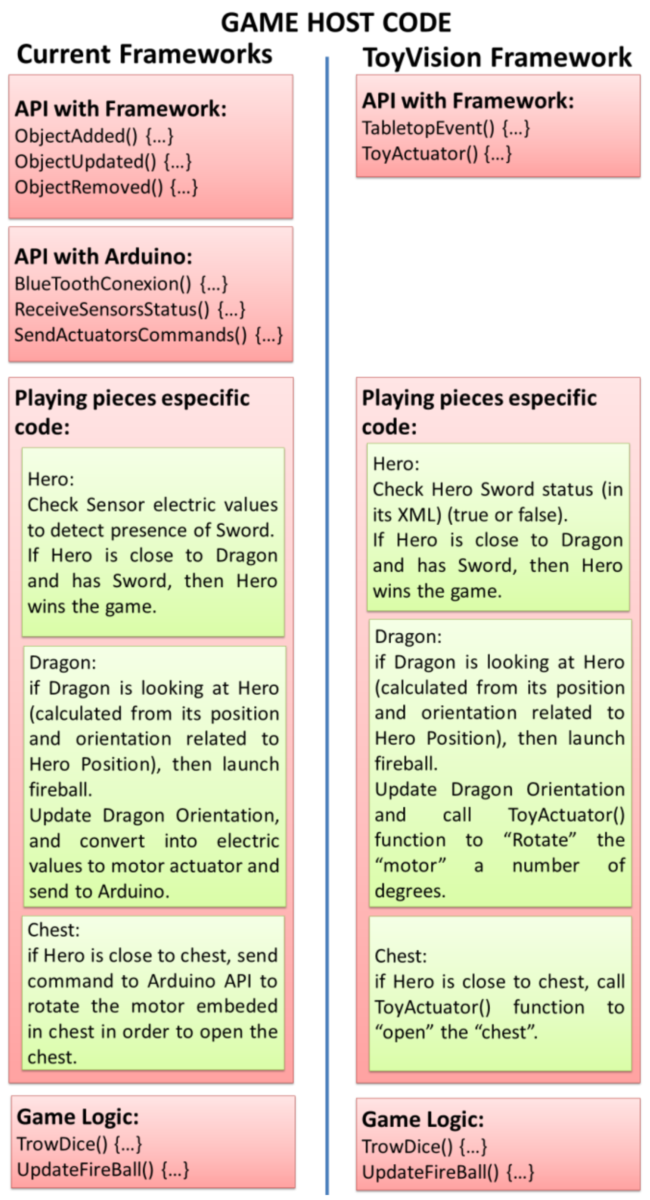

GAME HOST CODE
Current Frameworks
ToyVision Framework
API with Framework:
ObjectAdded() {...}
ObjectUpdated() {...}
ObjectRemoved() {...}
API with Arduino:
BlueToothConexion() {...}
ReceiveSensorsStatus() {...}
SendActuatorsCommands() {...}
Playing pieces especific code:
Hero: Check Sensor electric values to detect presence of Sword. If Hero is close to Dragon and has Sword, then Hero wins the game.
Dragon: if Dragon is looking at Hero (calculated from its position and orientation related to Hero Position), then launch fireball. Update Dragon Orientation, and convert into electric values to motor actuator and send to Arduino.
Chest: if Hero is close to chest, send command to Arduino API to rotate the motor embeded in chest in order to open the chest.
Game Logic:
TrowDice() {...}
UpdateFireBall() {...}
API with Framework:
TabletopEvent() {...}
ToyActuator() {...}
Playing pieces especific code:
Hero: Check Hero Sword status (in its XML) (true or false). If Hero is close to Dragon and has Sword, then Hero wins the game.
Dragon: if Dragon is looking at Hero (calculated from its position and orientation related to Hero Position), then launch fireball. Update Dragon Orientation and call ToyActuator() function to "Rotate" the "motor" a number of degrees.
Chest: if Hero is close to chest, call ToyActuator() function to "open" the "chest".
Game Logic:
TrowDice() {...}
UpdateFireBall() {...}

Figure 10. Code of the "Dragon's Cave" game. Left. Using an EIL frameworks. Right. Using ToyVision framework.

In an EIL toolkit, developers just receive from the Framework events triggered by fiducials being added, moved and removed from the table surface. Any other interactions with the playing pieces have to be dealt by implementing an Arduino API in charge of receiving low level data from electronic sensors, and sending low level commands to electronic actuators.

In the ToyVision toolkit, developers just have to deal with high abstract XML messages through an API composed of two functions:

The **toyActuator** function builds an XML command for a playing piece actuator and sends it through the socket.

The **tabletopEvent** is a callback function that is automatically summoned each time a message arrives from the Framework. This function reads the XML event, checks which playing piece triggered the event, and selects the specific code to run for each one.

Provided by the ToyVision Widget Layer, the task of coding a tangible game consists on reading and building XML messages straight related with the playing pieces, and it is completely isolated of playing pieces hardware implementation details.

CONCLUSION

The ToyVision Toolkit offers designers of computer board-games tools that help them to digitally enrich conventional playing pieces to support a great variety of manipulations and actions from the players and the computer system. This has been achieved by adding a new abstraction layer (the Widget layer) to an Event Interpretation Layered oriented framework (ReacTIVision) and the development of a Graphic Assistant tool in which the designers model each playing piece as tangible controls of the game application.

ToyVision contributes to the consolidation process that tabletop devices and the TUI paradigm have experienced during recent years by creating new "design-thinking" prototyping tools to create TUI applications.

A beta version of ToyVision can be downloaded from www.toyvision.org and can be used and modified under an open-source license. ToyVision's next future work is focused in its evaluation. For this purpose, we plan to carry out a user exploration workshop and to analyse its performance with objective final users. There are also plans to expand the tangible possibilities of playing pieces, even outside the tabletop surface. Different kinds of tangible tabletop pieces extending beyond board games will also come within the scope of our work.

ACKNOWLEDGMENTS

This work has been partly financed by the Spanish Government through the DGICYT contract TIN2011-24660.

REFERENCES

1. Antle, A.N., Bevans, A., Tanenbaum, J., Seaborn, K., and Wang, S. Futura: design for collaborative learning and game play on a multi-touch digital tabletop. Fifth international conference on Tangible, embedded, and embodied interaction (TEI '11). ACM, pp. 93-100.
2. Bespoke: http://www.bespokesoftware.org/multi-touch
3. Bollhoefer, K. W., Meyer, K., and Witzsche, R.. Microsoft surface und das Natural User Interface (NUI). Technical report, Pixelpark, Feb. 2009.
4. CCV: Community Core Vision Web: http://nuicode.com/
5. Cooper, N., Keatley, A., Dahlquist, M., Mann, S., Slay, H., Zucco, J., Smith, R., and Thomas, B. H. Augmented Reality Chinese Checkers. In Proceedings of the 2004

ACM SIGCHI international Conference on Advances in Computer Entertainment Technology (2004). ACE '04, vol. 74. pp.117-126.

6. Costanza, E., Shelley, S. B., Robinson, J. Introducing audio d-touch: A tangible User Interface for Music Composition and Performance. DAFx '03 Conference.
7. Cottam, M., & Wray, K.. Sketching Tangible Interfaces: Creating an Electronic for the Design Community. IEEE Computer Society, (2009, June). Pp. 90-95.
8. Dey, A.K., Abowd, G.D., Salber, D. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. Human-Computer Interaction, 2001, v.16 n.2, pp. 97-166.
9. Echtler, F., Klinker G. A multitouch software architecture. In Proc of NordiCHI '08. pp. 463- 466.
10. Greenberg,S. and Fitchett, C. Phidgets: easy development of physical interfaces through physical widgets. In UIST '01, pages 209–218.
11. Hansen, T.E., Hourcade, J.P., Virbel, M., Patali, S. and Serra, T. PyMT: a post-WIMP multi-touch user interface toolkit. International Conference on Interactive Tabletops and Surfaces (ITS '09). ACM, pp. 17-24.
12. Hartmann, B., Klemmer, S.R., and Bernstein, M.d. tools: Integrated prototyping for physical interaction design. IEEE Pervasive Computing, 2005.
13. Heijboer M, and van den Hoven, E. Keeping up appearances: interpretation of tangible artifact design. 5th Nordic conference on Human-computer interaction: building bridges (NordiCHI '08). ACM, pp. 162-171.
14. Heng, X., Lao, S., Lee, H., and Smeaton, A. A touch interaction model for tabletops and PDAs. In Proc. PPD '08.
15. Hinske, S. and Langheinrich, M. W41K: digitally augmenting traditional game environments. 3rd international Conference on Tangible and Embedded interaction (2009). TEI '09, ACM. pp. 99- 106.
16. Holmquist L.E., Redström, J., Ljungstrand, P. Token-Based Access to Digital Information. 1st international symposium on Handheld and Ubiquitous Computing (1999), p.234-245
17. Iwata, T., Yamabe, T., Poloj, M., and Nakajima, T. Traditional games meet ICT: a case study on go game augmentation. Fourth international conference on Tangible, embedded, and embodied interaction (TEI '10). ACM, pp. 237-240.
18. Joliffe D. Arduino fever. MAKE V7, pp. 52–53, 2006
19. Kaltenbrunner, M., Bovermann, T., Bencina, R., and Costanza, E. TUIO: A protocol for table-top tangible user interfaces. In 6th Int'l Gesture Workshop, 2005.
20. Kimura, H., Tokunaga, E., Okuda, Y., and Nakajima, T. CookieFlavors: easy building blocks for wireless tangible input. In CHI '06 extended abstracts on Human factors in computing systems (CHI EA '06). ACM, pp. 965-970
21. Klemmer, S.R., Li, J., Lin, J., Landay J.A. Papier-Mache: Toolkit support for tangible input. SIGCHI conference on Human factors in Computing Systems (CHI'04). Pp. 399-406
22. Leitner, J., Haller, M., Yun, K., Woo, W., Sugimoto, M., Inami, M., Cheok, A. D., and Been-Lirn, H. D. Physical interfaces for tabletop games. Comput. Entertain. 7, 4, Article 61 (January 2010), 21 pages.
23. Libavg web http://www.libavg.de/
24. Lin H.-H., and Chang, T.-W. A camera-based multi-touch interface builder for designers. In Human-Computer Interaction. HCI Applications and Services, 2007.
25. Marco, J., Cerezo, E., and Baldassarri, S. ToyVision: A Toolkit for Prototyping Tabletop Tangible Games. The fourth ACM SIGCHI EICS 2012.
26. Mikhak, B., Lyon, C., & Gorton, T. The Tower system: A toolkit for prototyping tangible user interfaces. Submitted as a long paper to CHI 2003
27. Openexhibits web: http://openexhibits.org
28. Reactivision: http://reactivision.sourceforge.net/
29. Rogers, Y. and Rodden, T. Configuring spaces and surfaces to support collaborative interactions. In O'Hara, K., Perry, M., Churchill, E. and Russell, D. (eds.) Public and Situated Displays. Kluwer Publishers. 2004. pp. 45-79.
30. Shaer, O. and Jacob, R.J.K. A specification paradigm for the design and implementation of tangible user interfaces. ACM Trans. Comput.-Hum. Interact. 16, 4, Article 20 (November 2009), 39 pages.
31. Schöning, J., Hook, J., Motamedi, N., Olivier, P., Echtler, F., Brandl, P., Muller, L., Daiber, F., Hilliges, O., Löchtefeld, M., Roth, T., Schmidt, D. and von Zadow, U. Building Interactive Multi-touch Surfaces. JGT: Journal of Graphics Tools. 2009. Springer.
32. Shen, C., Vernier, F., Forlines, C., and Ringel, M. DiamondSpin: an extensible toolkit for around-the-table interaction. In Proc. CHI '04, pages 167–174, 2004.
33. TouchLib: http://nuigroup.com/touchlib/
34. Trackmate: http://trackmate.sourceforge.net/