Multimed Tools Appl
DOI 10.1007/s11042-014-2298-2

Lowering the threshold and raising the ceiling of tangible expressiveness in hybrid board-games

Javier Marco · Eva Cerezo · Sandra Baldassarri

Received: 28 February 2014 / Revised: 23 July 2014 / Accepted: 23 September 2014

Abstract Tabletop devices offer an attractive environment for the creation of hybrid board-games that seamlessly integrate physical and digital interaction. However, the prototyping of Tangible User Interfaces involves the integration of physical and virtual aspects, challenging both designers and developers, and hindering the rapid exploration of richer physical interactions with the game. Although the recent appearance of tabletop frameworks has helped to lower the threshold for coding tangible tabletop applications, the cost is a serious limitation on the degree of expressiveness of the tangible interaction. This paper presents "ToyVision", a software framework for the prototyping of tabletop games based on the manipulation of conventional playing pieces. ToyVision architecture is based on a Widget layer of abstraction that is able to express the manipulation of playing pieces in the context of the game. This Widget layer supports a wide range of passive and active manipulations with conventional objects. Designers and developers can visually model all passive and active manipulations involved in their game concepts in a Graphic Assistant, and test them immediately in a tabletop device. The paper also describes a set of evaluation sessions which demonstrate that ToyVision has been effective in lowering the threshold of developing applications for tabletop devices, while being sufficiently expressive to give support to innovative concepts for hybrid games that fully explore the tangible feasibilities of conventional playing pieces.

Keywords Tabletop · Framework · Tangible · Board-games · Hybrid games · Playing pieces · Design · Prototyping

J. Marco (✉)
GIGA Affective Lab Computer Science Department, Universidad de Zaragoza, Zaragoza, Spain
e-mail: jmarco2000@gmail.com

E. Cerezo · S. Baldassarri
GIGA Affective Lab, Computer Science Department, Engineering Research Institute of Aragon (I3A), Universidad de Zaragoza, Zaragoza, Spain

E. Cerezo
e-mail: ecerezo@unizar.es

S. Baldassarri
e-mail: sandra@unizar.es

Published online: 02 October 2014

1 Introduction and motivation

One of the emerging research fields in Human-Computer Interaction (HCI) is concerned with innovative interaction techniques that aim to provide a more seamless bridge between the physical and digital worlds. Tangible User Interfaces (TUI) aim to give physical form to digital information by coupling physical manipulations of conventional objects with computational systems [19].

Games and entertainment computer applications are especially prolific in putting into practice the TUI paradigm, giving rise to a new generation of hybrid games which combine traditional physical playing with the new possibilities of digitally augmenting the player's area with computer image and audio feedback. Large horizontal interactive surfaces (or tabletop devices) are emerging as an ideal environment for these innovative hybrid games [14, 25]. Traditionally, conventional tables are popular spaces for social board-games due to their physical affordances that engage face-to-face interaction [38]. Players sit around the table and interact with the game by dragging and manipulating physical playing pieces on the table surface. Tabletop computer systems provide the possibility of digitally augmenting the game area while keeping playing pieces in the player's physical environment, thus reinforcing the emotional impact of videogames [21, 16] and making digital technology accessible to other user profiles such as very young children [32], users with disabilities [26] and senior citizens [1].

Due to the relative youth of TUI research, designers and developers working in this field face a number of challenges which involve dealing with potentially complex manipulations with objects, non-standardized computer platforms and electronic sensors and actuators. In contrast to the design of traditional WIMP (Windows, Icon, Menu, Pointer) user interfaces, TUI involve dealing with both software and hardware components from the beginnings of the design process. Detecting physical manipulations requires embedding sensing hardware into the objects, receiving and sending raw electronic signals to the computer system, and coding hardware level algorithms to translate raw data from hardware into meaningful playing-piece manipulations. Interaction designers do not necessarily have all these technical skills, and so the prototyping of a TUI requires interdisciplinary teams including interaction designers, electronic engineers and specialized software developers. This situation can cause a breach between design decisions and the corresponding implementation tasks (i.e. between designers and developers) which results in prototypes that are difficult to adapt to design changes and in hindering designers from freely exploring all the feasibilities of coupling conventional playing pieces with a computer game.

To address this issue, a broad variety of heterogeneous and very specialized frameworks have evolved over the last few years giving support to the development of tabletop applications independent of the hardware. As stated by Myers et al. [34], frameworks are tools that aim to make the rapid prototyping of user interfaces easier and enable more iterations during the design process by lowering the threshold of developing the system while retaining a high ceiling of expressiveness. The threshold is the difficulty of learning and using a system, which is connected with the development process, while the ceiling captures the complexity of what can be built using the system, which is directly related with the design process.

Early tabletop frameworks only supported the development of very basic tactile tabletop applications by merely informing developers about raw-tactile events (finger added, moved, taken away from the table) [42, 3, 27, 40, 29], thus isolating developers from the tabletop hardware details. More recent tabletop frameworks have raised the ceiling of expressiveness of multitouch interaction by supporting more complex off-line tactile gestures (zoom, rotation, delete …) on the tabletop surface [15, 13, 35]. A few tabletop frameworks also support

interaction with conventional objects on the table surface [36, 6, 4, 43, 23], opening up the possibility of prototyping TUI for tabletops. However, as can be seen in the next section, current tabletop frameworks with TUI support, have low expressiveness for designers willing to create hybrid board-games, since they only support simple passive manipulations: object added, moved, taken away from the table by the user . Thus, the status of a playing piece in the game can only be dependent of its position on the board. In conclusion, the expressiveness of the games that can be created with current frameworks is limited to simple board-games in which players are restricted to arrange different playing pieces on the board (e.g. tic-tac-toe, snake-and-ladders, ludo, chess…). However, very frequently, board-games require from players other kinds of passive manipulations with the playing pieces in order to change their status in the game. For example, in the Trivial Pursuit™ game, players arrange small pie-shaped playing pieces inside their tokens to represent their status on the game (see Fig. 1-left). The same way, many board-games (e.g. Twister™) have spinner playing pieces which players manipulate in order to trigger random events on the game (see Fig. 1-centre). Current tabletop frameworks have only expressiveness to identify these playing pieces on the board and track their position; however, no current framework is giving support to identify the different statuses that the Trivial Pursuit and the spinner playing pieces can have during the game.

Furthermore, many conventional board-games have playing pieces that players must manipulate during the game but that, in fact, do not belong to any player. For example, in the classic board-game Cobra's Island™, players have to fight and slay a miniature cobra playing piece (see Fig. 1-right). This playing piece is passive; therefore, each turn, the player has to move his/her playing piece, and also the cobra playing piece according to game rules. By translating this game to a hybrid board-game, it would be possible to make the cobra playing piece active by embedding electronic actuators and connecting them to the computer system. However, this would not be an easy task with current frameworks, and it requires from designers advanced technical skills such as electronics, communication protocols, etc.

This paper presents ToyVision: a framework for the prototyping of hybrid videogames for tabletop devices. The main contribution of ToyVision is the inclusion of an abstraction layer on an existing tabletop framework. The new abstraction layer has been designed to support a wide range of passive and active manipulations of playing pieces involved in hybrid board-games. Being provided with this new layer, ToyVision not only raises the ceiling of expressiveness of tangible interaction in tabletop devices, but also lowers the threshold of implementing hybrid board-games. Designers can model customized passive and active manipulations in playing pieces and evaluate them without requiring advanced electronics and coding skills. Moreover, ToyVision simplifies the coding process by informing developers about the status of playing pieces, with meaningful events directly related to playing piece manipulations.

Fig. 1 Different conventional playing pieces. *Left*: Trivial Pursuit™ token. *Centre*: spinner. *Right*: passive cobra playing piece

This paper continues and extends the work previously presented in [31] and [30], containing a deep analysis of the state of the art of tabletop frameworks, detailing the complete process of prototyping a well-known tabletop tangible application with ToyVision and, more important, presenting a set of evaluations that qualitative and quantitatively assess the benefits of using ToyVision.

The paper first examines the current state of the art in frameworks and tools supporting tangible interaction in tabletop devices in Section 2. In section 3, the adoption of a tangible modelling language to express active and passive manipulations is explained. Section 4 details the ToyVision framework architecture. Section 5 describes the process of prototyping tabletop games through a practical example. The evaluation carried out with ToyVision is detailed in Section 6. Finally, the conclusions and proposals for future work are outlined.

2 Related work: tabletop frameworks for tangible interaction

Due to the recent success of tabletop active surfaces, several software frameworks have emerged for using in rapidly developing hardware-independent tabletop applications. However, as stated before, the majority focus on supporting tactile interaction and relatively few tabletop frameworks target the prototyping of tangible interaction.

Table 1 lists current tabletop frameworks that provide support for interaction with objects manipulated on the tabletop surface. It must be taken into account that the table analyses only tangible (not tactile) interaction functionalities.

The first column classifies the framework architecture. Echtler and Klinker [9] described the architecture of tabletop frameworks using four layers of abstraction from the lowest to the highest: Hardware, Calibration, Event Interpretation and Widget. A framework that follows a layered architecture (Fig. 2) offers at least the Hardware Layer in order to hide the low-level algorithms that obtain raw data from the tabletop hardware. Optionally, the framework can add the Calibration layer to correct the position coordinates of each detected finger and object due to hardware aberrations. In the Event Interpretation layer, the framework keeps track of basic tactile events ("finger" added, moved or removed from the tabletop surface) and tangible events ("object" 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 host application. Most current tabletop frameworks are based on an Event Interpretation layer [6, 10, 4, 41, 36, 43, 28] that triggers generic tangible events when objects are manipulated on the tabletop surface. This means that all tabletop host applications will receive the same generic events and consequently these have to be interpreted to give them meaning in the context of the tabletop application. On the other hand, Widget layer architectures are becoming common in frameworks supporting multitouch interaction [15, 13, 35]. In these frameworks, the Widget layer is responsible for processing off-line tactile gestures with a specific meaning in the application (zoom, delete, select…). However, this Widget approach is, at present, not being followed in tangible interaction. Only the Papier-Mâché [23] and the RoboTable [24] frameworks can be considered as Widget layered architectures. The Papier-Mâché Widget layer is capable of giving meaning to generic tangible events sent by the Event Interpretation layer, and this meaning is directly related to a specific action in the tabletop host application; e.g., placing a post-it note on the tabletop surface means opening a web page; thus, the Papier-Mâché Widget layer will inform the tabletop application to open the web page. The RoboTable Widget layer is exclusively aimed at managing autonomous movements on a robotic toy at a high level of abstraction; e.g., when the robot is placed on a specific tabletop area it performs a series of predefined movements.

Table 1 Comparison of different tabletop frameworks with support for developing TUI

Framework	Architecture	Platform		Tools	Manipulations		
		Development environment	Communication protocol		Passive		Active
					Type	Hardware	Type
CCV	Event Interpretation	Cross platform	TUIO	*x*	Place 2DMove Remove	Visual Fiducials	*x*
Graffiti	Event Interpretation	Cross platform	TUIO	*x*	Place 2DMove Remove	Visual Fiducials	*x*
PixelSense SDK	Event Interpretation	MS Visual C++	*x*	*x*	Place 2DMove Remove	Visual Fiducials	*x*
Papier-Mâché	Widget	MS Visual C++	*x*	Graphic Assistant	Place 2DMove Remove	Visual RFID	*x*
ProtoActive	Event Interpretation	MS Visual C++	*x*	*x*	Place 2DMove Remove	Visual Fiducials	*x*
Reactivision	Event Interpretation	Cross platform	TUIO	*x*	Place 2DMove Remove	Visual Fiducials	*x*
RoboTable	Widget	Own	*x*	Graphic Assistant	Place 2DMove Remove	Visual Fiducials	2DMove
TrackMate	Event Interpretation	Cross platform	OSC	*x*	Place 2DMove Remove	Visual Fiducials	*x*
LibTisch	Event Interpretation	Cross platform	OSC	*x*	Place 3DMove Remove	Visual Fiducials Kinect	*x*

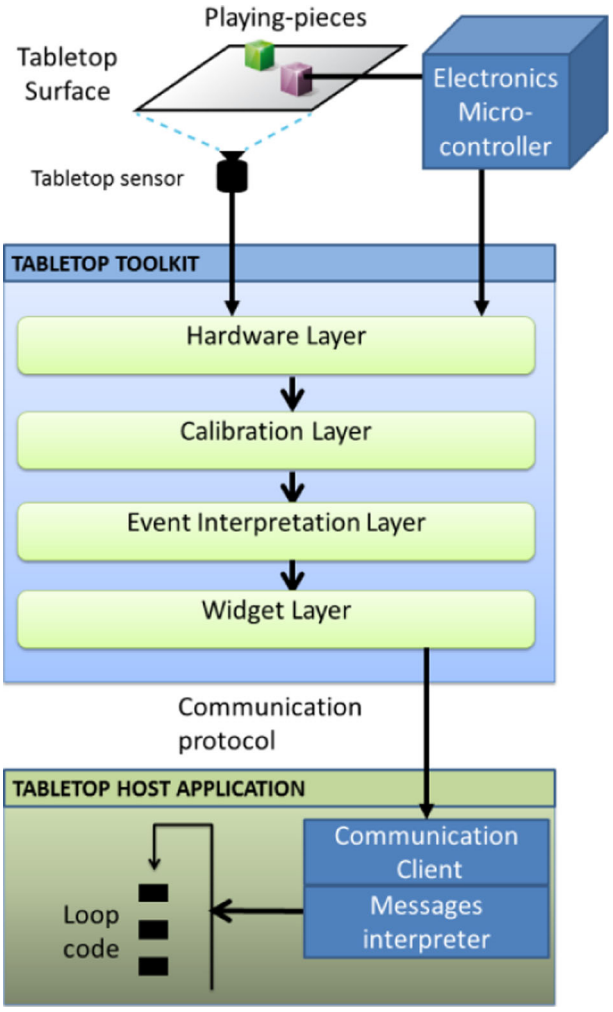

Fig. 2 Echtler and Klinker's layered framework architecture for tabletop applications. The tabletop framework acts as an intermediary application between the tabletop hardware and the host application

The second column in Table 1 analyses cross-platform support. Early frameworks targeted a specific development platform; Pixelsense SDK and Papier-Mâché target MS Visual C++. Nowadays, frameworks have adopted a client-server architecture in order to become independent from the development environment (see Fig. 2): tabletop events are thus transported via a network protocol. In this way, host applications can be developed in any programming environment provided by a network communication can be implemented. In this context, the OSC [45] and TUIO protocols [22] have become very popular and have been adopted by most tabletop frameworks [36, 6, 28, 43, 10]. Both protocols have similar characteristics since TUIO can be considered an evolution of OSC. Both aim to communicate generic events from a wide range of interactive devices (musical instruments, active surfaces…) to a host application. When used in a tabletop framework, these protocols can be used to transmit generic tactile and tangible events, embedded in binary packets using UDP network communication sockets.

The third column in Table 1 analyses the support tools that frameworks additionally provide to facilitate the prototyping of tangible tabletop applications. Most current frameworks are "non-interactive" applications. They run in background, dealing with the tabletop hardware, and sending tabletop events. During this process, they do not require user intervention. On the other hand, the Papier-Mâché and RoboTable frameworks include an interactive Graphic Assistant tool that designers may use to define specific application behaviours in response to the manipulations of objects and to test these behaviours immediately on the tabletop. Since Papier-Mâché and RoboTable are also the only frameworks that contain a Widget layer in their architecture, graphic assistants would appear to have an important role in supporting Widget layered architectures.

The last group of columns in Table 1 describes the kind of tangible manipulations supported by each framework. This column is divided into passive (carried out by users) and active (carried out by the system) manipulations.

Regarding passive manipulations, the Type column describes different feasible object manipulations. All current tabletop frameworks with tangible support are capable of tracking the position and orientation of objects on the tabletop surface. To do this, all frameworks are based on visual based hardware (conventional or infrared cameras) and on image processing algorithms. A few current frameworks (CCV, Papier-Mâché and LibTisch) are able to visually identify different objects from their shape; however, the visual algorithms used in these frameworks are not robust enough when the objects' shapes are very similar. For this reason all current frameworks (with the exception of Papier-Mâché) use visual based tags (fiducials) for the robust tracking of objects. A fiducial is a printed marker (generally black and white) with very simple topological designs than can be easily recognized by well-known image processing algorithms. When different fiducials are attached to the base of each object involved in the TUI, the framework is able to identify them when they are directly placed on or removed from the tabletop surface and to track their movements when they are dragged (2D position and orientation) on the tabletop surface. Fiducials have become very popular for the quick development of tangible tabletop application, since they are very easy to use and are hardware independent (any conventional video camera can be used to track fiducials). The only framework not supporting this fiducial technique, the Papier-Mâché framework, uses a combination of shape visual recognition algorithms and RFID tagging in order to identify objects. While RFID chips were very popular in tangible application prototypes some years ago, they have now been replaced by fiducials since the latter are more versatile and provide more robust identification and tracking of objects on a wide surface. Finally, all current frameworks are limited to tracking objects which are directly placed on the surface, with the exception of LibTisch which is also capable of tracking movements of objects manipulated over the tabletop surface, extracting also their relative height (3D coordinate); however, this functionality makes the LibTisch framework more hardware dependent as it is based on a Microsoft Kinect sensor.

Regarding active manipulations, no current tabletop framework provides support to create reactive objects with the single exception of the RoboTable framework. However, RobotoTable is very restricted in the kind of objects supported for reactive manipulations, being limited to an electronic robotic toy, specifically built for this framework, which is capable of autonomous 2D movements on the tabletop surface. RoboTable does not therefore support tangible manipulations with conventional objects. In the context of TUI prototyping, it is necessary to embed electronic sensors and actuators into the objects in order to design and implement active manipulations for them. Sensors provide context-aware information from an

object's environment (light, contact pressure, temperature…), while actuators can change some physical properties of the object (movement, colour change, shape change…). Also, the computer system has to be able to retrieve data from the sensors and to actuate the actuators. This requires an electronic microcontroller board which converts sensor and actuator raw signals into something manageable by the computer system. In conclusion, the prototyping of active manipulations requires advanced electronic skills. Several commercial and open-hardware toolkits have appeared to provide these processes [12, 33, 2]. These hardware toolkits contain a set of electronic microcontrollers, sensors and actuators which can be used by designers to easily assemble and embed the sensors and actuators into conventional objects, and to connect them to a computer system. The problem is that no current tabletop framework offers support to any hardware toolkit, so developers have to implement hardware level algorithms in their tabletop host applications in order to deal with raw data coming from microcontroller boards.

After having analyzed the tangible functionalities of current tabletop frameworks, several conclusions can be reached:

- The architecture of a tabletop framework has a strong impact both on the threshold of prototyping a tabletop application and on the ceiling of expressiveness of the tangible manipulations. Event Interpretation based architectures send generic events about "everything" that the tabletop hardware has detected on the tabletop surface. The developer has to code complex algorithms to process all those events, filtering them and finding relationships relevant in the context of the host application. Tangible expressiveness is, therefore, limited to the data provided by the event interpretation layer that entirely consists of generic manipulations completely out of context of the tabletop host application. On the other hand, current tabletop frameworks based on a Widget layer architecture are using the Widget layer in a very limited way, merely associating the position of an object on the tabletop with an action in the application, and thus missing other feasible object manipulations.
- Cross-platform frameworks help to lower the threshold of developing applications, since developers can choose the programing language more appropriate to their skills. However, client-server architectures based on transmitting binary packages also limit the expressiveness of the information that the framework sends to the host application.
- Including additional tools in a framework can help to lower the threshold of developing the host application. Graphic assistants have been successfully used in the context of WIMP interfaces [11] saving coding resources during the implementation of interactive forms. A similar approach is being followed by a few current tabletop frameworks that include a graphic assistant capable of automatically generating code from the information provided by the designer. The relationship between the graphic assistant and the Widget layer architecture is also relevant. The graphic assistant helps developers to define all the tangible manipulations to be interpreted by the Widget layer from the events coming from the Event Interpretation layer.
- Current frameworks consider objects as mere "moveable" entities, both for passive and active manipulations. Other feasible manipulations such as the manipulation of just part of an object (e.g. opening and closing the lid of a box) or the combination of different objects (e.g. placing an object inside a box) are not supported by current frameworks. It can be said that this is a direct consequence of the framework architecture and the limitations of the communication protocols that have been designed targeting only these specific manipulations, leaving out the possibility of considering other passive and active manipulations.

Considering this analysis of the state-of-art of tabletop frameworks supporting the prototyping of TUI, our objectives when designing ToyVision were:

- to follow a Widget Layer architecture approach, supported by a graphic assistant tool. The Widget layer should be capable of interpreting any feasible passive and active manipulations and expressing them in the context of the host tabletop application. The graphic assistant should support designers and developers to visually model all the active and passive manipulations involved in the host application.
- to be cross-platform, using a client-server approach, but with a communication protocol not constrained to the kind of data to be transmitted in order to avoid limiting the expressiveness of the messages transmitted by the ToyVision Widget layer. The ToyVision communication protocol could be based on XML-compliant messages: this way the ToyVision Widget layer would be able to adapt the data packed into the messages to the specific manipulations involved in the host application.
- to support a wide range of active and passive manipulations of conventional objects; in particular, the arrangement of different objects into a compound object (inserting/removing an object into/from another one), and the manipulation of objects with moveable pieces (moving/rotating part of an object). In order to detect and identify passive manipulations, ToyVision should support generic visual based tabletop hardware, using fiducials to track the movements of objects directly placed on the tabletop surface.

A direct consequence of these objectives will be the raising of the ceiling of the tangible expressiveness while keeping the threshold low for developing tabletop applications supporting tangible interaction.

The main challenge of creating a framework with a high ceiling of tangible expressiveness lies in being able to express all the feasible manipulations with conventional objects in a language understandable by a digital system. The following section describes the adoption of a modelling language in ToyVision to support designers in expressing all passive and active manipulations involved in their tabletop applications.

3 Modelling manipulations on tabletop games

3.1 Describing a TUI with the TUIML

Despite many attempts at defining and categorizing the expressiveness of object manipulations in a digital system [18], currently there is no single approach for exploiting the physical affordances of objects in a framework for the prototyping of TUIs. The “Token” approach [17] was one of the first and more popular approaches to formalize the link between object manipulations and the digital system. This was later updated by Shaer and Jacob [39] in the Token+Constraint+TAC paradigm. This paradigm introduces a compact set of vocabulary and means to systematically describe the structure and functionality of a large subset of TUIs. This set of constructs aims to allow any TUI designer to specify his or her designs while considering all the physical affordances of conventional objects. What is interesting about Shaer and Jacob’s proposal is that they also present the Tangible User Interface Modeling Language (TUIML) based on the Token+Constraint+TAC paradigm. Using the TUIML, designers can visually describe in terms of Tokens, Constraints and TACs the different relationships between physical objects and digital information in their TUI concepts. The main contribution of the

TUIML is that its visual representations can be automatically translated into a computational system using a computer language such as XML.

To visually describe any TUI, the TUIML uses a hierarchical structure of Tokens, Constraints and TACs (see Fig. 3).

A **token** is any graspable object involved in the TUI. A token's physical properties may dictate how it is to be manipulated in the TUI.

Tokens can also act as containers of other tokens. The container token has one or more physical **constraints** associated to it which limit the manipulation of the other tokens. Both user and system (passive and active interaction) can manipulate tokens inside a constraint in two possible ways:

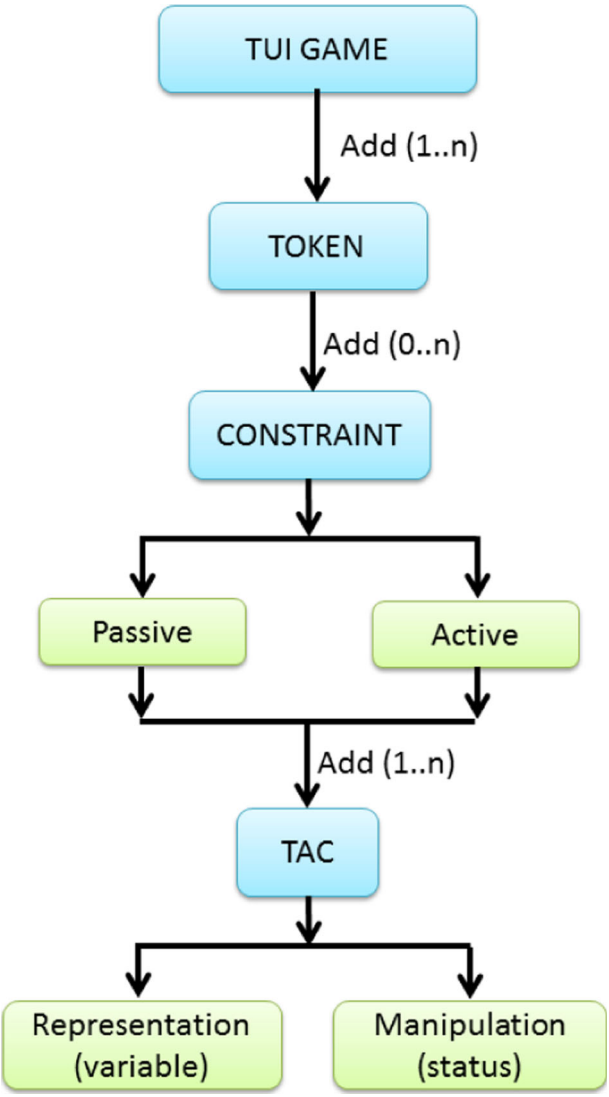

Fig. 3 TUIML visual representation based on the Token+Constraint+TAC paradigm

- Associating digital data. By filling, emptying, or reassigning tokens inside the constraint, different digital states are assigned to the container token. For example, the playing piece used in the Trivial Pursuit board game is divided into six compartments (constraints) in which little colored tokens can be inserted to represent the player's status in the game.
- Manipulating digital data. By moving or rotating tokens in the constraint, the user varies the digital state of the container token in the TUI. For example, a kitchen timer has a knob attached that, when rotated, varies the alarm time.

Depending on who carries out the manipulations (the user or the system), a constraint is defined as:

- A Passive Constraint. This defines a physical limitation on the user's manipulations within the playing piece. Tokens with passive constraints are composed of moveable or removable smaller pieces. By manipulating them (inserting, removing, rearranging or moving them), the user is effectively changing the status of the token in the TUI.
- An Active Constraint. In order to provide a token with active manipulations (autonomous movement, sound, colour change…), electronic actuators are embedded in the object. This can also represent a constraint since electronic components have a limited (constrained) range of actuation.

In order to describe passive and active manipulations on a constraint, one or more TACs (Token And Constraint relationship) have to be associated to the constraint. A TAC expresses a physical status of an object with some meaning in the tangible application. Activating a TAC involves whether the user is physically manipulating a token with respect to its constraint, or whether the system is electronically manipulating an actuator. Manipulation of a token outside a constraint has no effect on the tangible application; however, its manipulation inside the constraint has a very specific meaning in the application which is described by the associated TACs. Depending on how this meaning is described, a TAC can be defined either as a Representation or Manipulation TAC:

- Representation TAC: The meaning of a token manipulation is described by a numerical value (variable) limited within a range of values. This value describes a physical property of the token manipulated within a passive constraint or a value to be applied to an electronic actuator associated to an active constraint. For example, in a kitchen timer, a representation TAC will be associated to the knob in order to represent the numerical value of the alarm time (in minutes).
- Manipulation TAC: The meaning of a manipulation is described by a name. The name defines a very specific status of the token within its passive constraint or of an electronic actuator associated to an active constraint. For example, in the Trivial Pursuit playing piece, a list of manipulation TACs will represent different game achievements by a different name (science, arts, geography…).

In Shaer and Jacob's work, TUIML was used as a theoretical framework to describe a wide range of well-known TUIs. Despite the fact that TUIML was developed so that any TUI visual representation could be automatically translated into a computer language (e.g. XML), at present there is no practical implementation of TUIML. ToyVision is the first software framework that has adopted TUIML to express all tangible manipulations with conventional playing pieces in hybrid board games. However, some updates to the original TUIML have been required in order to adapt it to the board-game context. These are described below.

3.2 Extending TUIML to model hybrid tabletop games

3.2.1 ToyVision TUIML

In board games, the physical appearance of playing pieces defines their role in the game; for example, the color of the playing piece usually identifies the player to whom it belongs. In order to express the most common roles of playing pieces in board-games, ToyVision TUIML introduces three new categories of tokens: **Simple**, **Named and Deformable Tokens** (see Fig. 4).

- Simple Tokens (see Fig. 5a): Simple tokens are the most common playing pieces in board-games: checkers, marbles, chips, etc. Players arrange a limited amount of playing pieces on the board according to game rules (e.g. Checkers, Ludo, Snakes and Ladders, Roulette, etc.). In general, simple tokens can be physically described as small flat cylindrical pieces, all identical with the exception of the colour used to distinguish the piece of each player, or to give different values (money, points, etc.).
- Named Tokens (see Fig. 5b): Other kinds of playing pieces are assigned very specific roles in the rules of the game and have a unique name. These are perceived by the player through their physical appearance. A classic example is in chess: the castle has a different appearance than the pawn and they are subject to different rules.
- Deformable Tokens (see Fig. 5c): There are other playing pieces which do not have a constant shape, as they change with the manipulations of the player. These are made of malleable materials such as clay, cardboard or cloth. Children usually play with these materials in craft activities.

Due to the technical limitations of visual-based tabletop devices, ToyVision TUIML only supports the creation of constraints in named tokens. Simple tokens are usually small pieces in which it is physically difficult to embed passive and active constraints. Similarly, it is physically difficult to create a physical constraint in an object with an undefined shape. In any case, ToyVision TUIML is sufficiently expressive to model common passive and active manipulations in hybrid games for tabletop devices. In order to validate this assertion, a workshop session was organized in order to gain initial practical experience with interaction designers using ToyVision TUIML. Details of this workshop are now described.

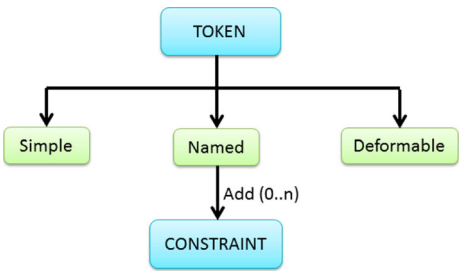

Fig. 4 TUIML extended to cover playing piece roles

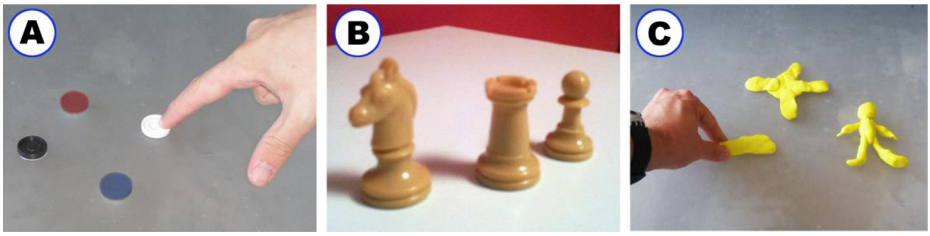

Fig. 5 Conventional board-game playing pieces classified as (**a**) Simple Tokens, (**b**) Named Tokens, (**c**) Deformable Tokens

3.2.2 Validating TUIML expressiveness

A workshop was organized at an international conference on HCI with the aim of validating the hypothesis that ToyVision TUIML is sufficiently expressive to model tangible interaction in tabletop devices. Seven interaction designers participated during a three and a half hour practical session. Six participants were Master students and one was an academic lecturer, all working in the field of Interaction Design. Five of the participants had some prior experience in designing TUI.

During the session, each participant had to ideate and create a playing piece suitable for use as a tangible token on a tabletop game. The session was divided into the following stages:

1. An informative introduction to tabletop tangible games (15 min). Participants were introduced to a visual based tabletop device, and some samples of hybrid games were given.
2. Brief description of ToyVision TUIML (30 min). Participants were introduced to the Token+Constraint+TAC paradigm. Each participant was provided with a printed TUIML template.
3. Concept creation (30 min). Participants worked individually ideating a new concept of a playing piece to be manipulated on a tabletop game. Each participant had to express their new concept drawing its TUIML representation on paper.
4. Prototyping (120 min): Participants had to build low-tech prototypes of their concepts. They had at their disposal hand-craft materials (pens, paper, foam, cardboard, scissors, glue, etc.) to build their prototypes (see Fig. 6).

Two coordinators assisted in the workshop session. One was in charge of introducing participants to the workshop tasks, explaining the use of the ToyVision TUIML and giving

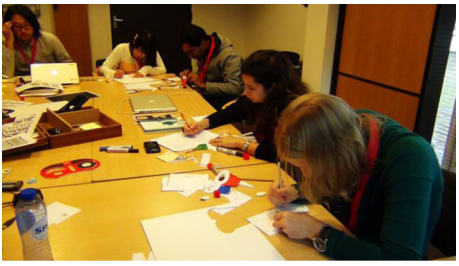

Fig. 6 ToyVision TUIML workshop session

technical support to participants during the prototyping. The second coordinator took observation notes, focusing on the participants' problems and questions.

By the end of the session, all participants had created their own playing pieces (see Fig. 7) together with their visual TUIML representations.

Three examples of playing pieces and their respective TUIML representations created during the workshop session are presented in Figs. 8, 9 and 10. Figure 8 shows a prototype of a tank playing piece for a tabletop videogame. Turret orientation is controlled by means of moving the small red piece along the circular groove (physical constraint). Shooting the missile is carried out by pressing on the circular hole. Figure 9 shows a prototype of a piano toy suitable for use in a tabletop musical game. The key movements are physically constrained in such a way that, when pressed, the keys make contact with the tabletop surface and the musical note is played. Figure 10 shows a prototype of a star-shaped musical instrument. Notes are played by pressing on the points of the star. The sound pitch can be varied by means of rotating the central knob.

All the playing pieces created during the workshop session were completely modelled using ToyVision TUIML, thus assessing its expressiveness. Since TUIML visual representations can be automatically translated into computer language, we argue that a software framework rooted in TUIML would have a high ceiling of expressiveness. Moreover, none of the passive manipulations involved in the new playing pieces created by the workshop participants were supported by current tabletop frameworks, but all could be supported by ToyVision, as explained in the following section.

4 ToyVision's architecture: the widget layer

ToyVision's architecture is based on an existing tabletop framework; ReacTIVision [36]. As well as its **Hardware**, **Calibration and Event Interpretation layers** (see Fig. 2), it has two additional features (see Fig. 11):

- Originally, the Reactivision hardware layer only supported the tabletop visual sensor (digital camera). In ToyVision, the **hardware layer** has been upgraded to support **communication with electronic components** embedded into the playing pieces. In order to communicate the tabletop computer system with electronic sensors and actuators, ToyVision's hardware layer supports the Arduino platform [30], an open-hardware platform which has become very popular for the development of tangible and ubiquitous

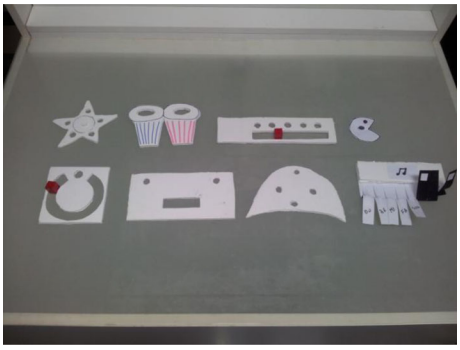

Fig. 7 Playing pieces created during the workshop session

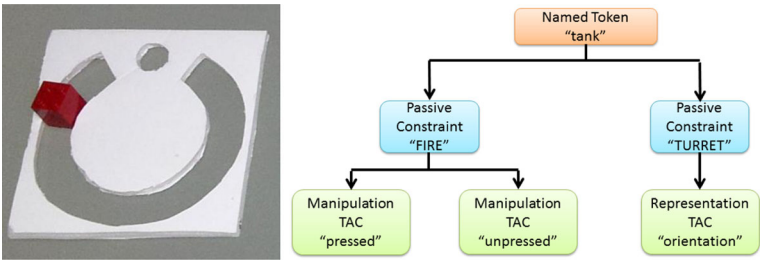

Fig. 8 Tank playing piece prototyped during the workshop and its corresponding TUIML

systems since it enables a wide range of electronic components to be easily connected to a computer system via USB or Bluetooth. ToyVision therefore offers support for implementing active manipulations of playing pieces.

- Originally, the Reactivision architecture was based on an Event Interpretation layer and the TUIO communication protocol. The ToyVision framework has upgraded the Reactivision architecture by including a new **Widget layer** which is composed of two elements: the Graphic Assistant and the Widget Manager.

The **Graphic Assistant** is a WIMP (Windows, Icon, Menu, Pointer) Graphic User Interface that enables the designer to visually model each playing piece in TUIML. These models provide all the data required by the Widget layer to perform its function. A first version of a Graphic Assistant not based on the TUIML has been already published [31]. The new version of Graphic Assistant presented in this paper fully implements ToyVision TUIML. In fact, The TUIML visual representation created with the graphic assistant is automatically translated into XML language and internally stored in ToyVision. This serves as a knowledge base for the ToyVision Widget Manager.

The **Widget Manager** listens for fiducial, subtoken (can be small graspable objects or user's fingers, both visually identified as small circular white spots) and blob events coming from the Event Interpretation layer, and identifies how these events are related to playing piece manipulations (**passive interactions**). The Widget Manager uses the TUIML representations of playing pieces to find these relations and creates XML-compliant messages to express the playing piece status during the game. This XML message is formatted following the TUIML hierarchy (Token, Constraints and TACs) and contains the updated status of the playing piece as well as the kind of manipulation that triggered the event. Figure 12 shows the XML structure of messages created by the Widget Manager. The "TokenName" tag identifies the playing piece and its attributes describe its position and orientation on the table surface. If the playing piece has more than one copy involved in the game, the "copy" attribute helps to distinguish between them. The "TokenName" tag also contains an "event" attribute which

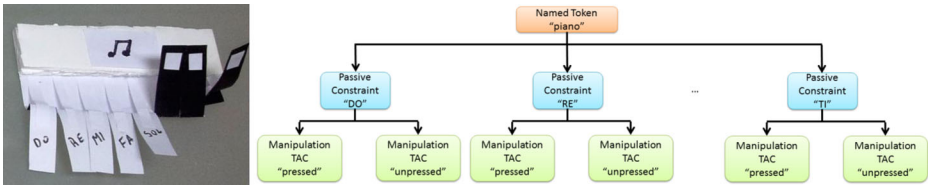

Fig. 9 Piano playing piece prototyped during the workshop and its TUIML

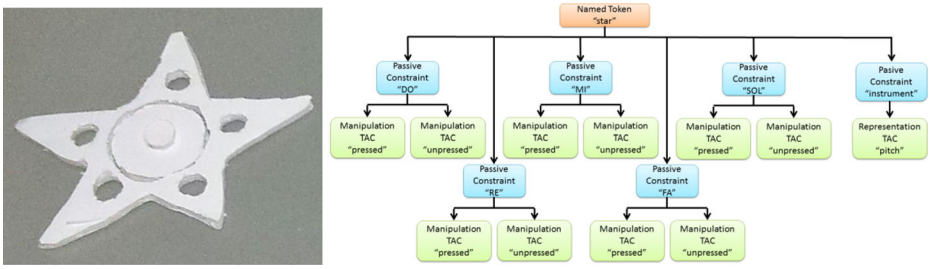

Fig. 10 Star-shaped music instrument prototyped during the workshop and its TUIML

indicates the kind of manipulation that triggered this event, which can be one of the following values:

- Added: the "TokenName" playing piece has been placed on the table surface
- Updated: the "TokenName" playing piece has been moved on the table surface
- Removed: the "TokenName" playing piece has been removed from the table surface
- A Constraint name: the named constraint has changed its status.

Each time a playing piece changes its status, the Widget manager sends an XML message to the host application through a TCP socket. The host game application only has to listen and parse them to extract an updated status of each playing piece placed on the tabletop surface. More details about coding the host application will be given in the next section.

The Widget manager also receives XML messages coming from the host application, containing commands to be carried out on reactive playing pieces (**active interactions**). These messages are translated into Arduino commands and sent to the Hardware Layer which is in wireless communication with the Arduino microcontroller.

This new Widget layer aims to keep the threshold low for developing tabletop games while raising the ceiling of expressiveness of passive and active manipulations compared to other similar software frameworks. As Table 1 shows, only a few of the current frameworks include a Widget layer of abstraction in their architectures, and these frameworks do not offer more tangible expressiveness than other frameworks based on an Event Interpretation layer architecture. The intrinsic difficulty of designing a Widget layer supporting tangible interaction lies in finding a language, understandable by a digital system, capable of expressing all the feasible manipulations with conventional objects. ToyVision is the first framework that has adopted a visual modelling language (the TUIML described in previous sections) to express all tangible manipulations involved in the application.

The following section shows, by means of a practical example, the process of prototyping a tangible tabletop application using the distinctive features of ToyVision.

5 Prototyping with ToyVision

In contrast with the development of conventional computer applications, generating a tangible application involves both physical and virtual tasks. Figure 13 sketches the complete process of developing a tangible tabletop game supported by ToyVision.

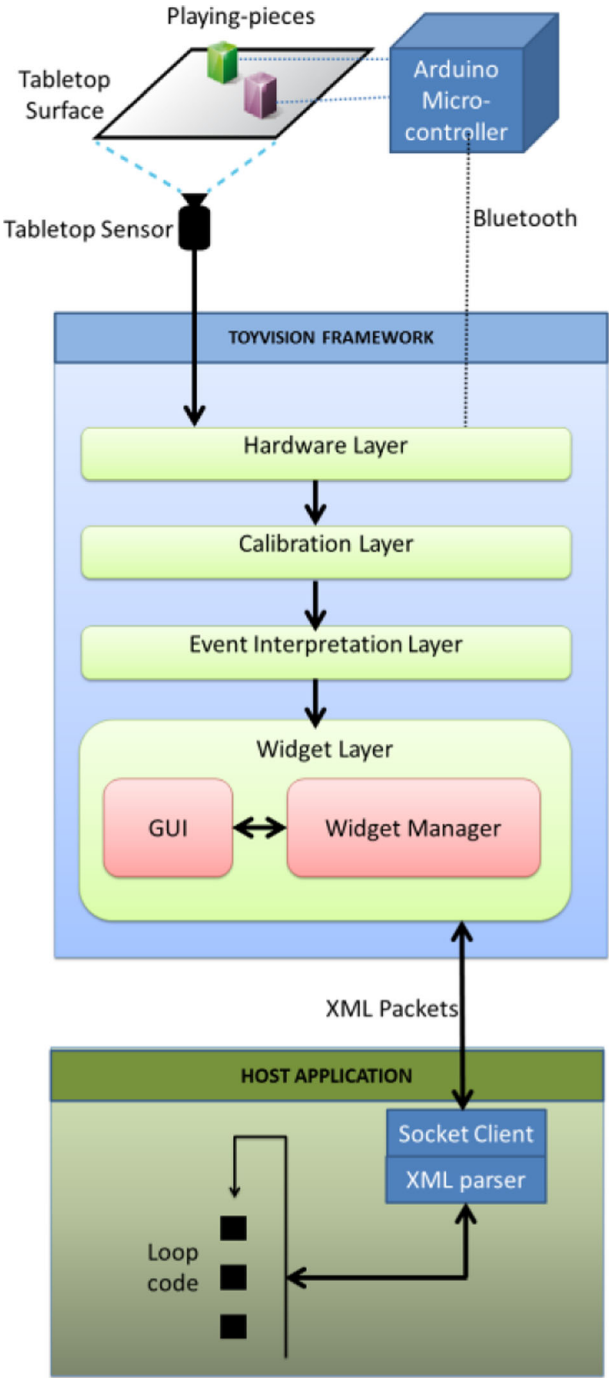

Fig. 11 ToyVision framework architecture

```
<TokenName copy="value" x="value" y="value" orientation="value" event="value">
        <ConstraintName_1>
                <TACName_1> {value} <TACName_1 />
                <TACName_2> {value} <TACName_2 />
                …
                <TACName_n> {value} <TACName_n />
        </ConstraintName_1>
        …
        <ConstraintName_n>
                …
        </ConstraintName_n>
</TokenName>
```

Fig. 12 XML structure of a message created by the Widget Manager

The next subsections give the details of each stage, exemplified with one of the pioneer tangible applications for tabletop devices and well-known through academic literature: MetaDesk [44]. MetaDesk (see Fig. 14) is a geographical application that enables users to navigate a city map projected on the tabletop surface by manipulating a set of objects on the same surface:

- Buildings (see Fig. 14a): When a building object is placed on the tabletop surface, the map automatically bounds to the object at its location on the map. We chose two representative buildings from our city (Zaragoza): the cathedral and the castle.
- Lens (see Fig. 14b): an object symbolizing a lens can be manipulated on the tabletop surface to visualize augmented data on the map. Two sub-tokens can be assigned to the lens object in order to get a photographic view of the map or to overlap graphic information about public transport on the lens area.
- Navigator (see Fig. 14c): this object can be dragged on the tabletop surface in order to navigate on the map. The navigator also has a moveable sub-token which can be used to adjust the zoom visualization factor.
- Compass (see Fig. 14d): this object was not in the original MetaDesk application, but it is added to our sample in order to exemplify the possibilities of including active manipulations with ToyVision. The compass is a reactive playing piece with helps users to orient themselves since it always points to the north of the map.

5.1 Physical prototyping

Prototyping a tangible tabletop application with ToyVision begins by handcrafting or adapting conventional objects to be used on a visual based tabletop device. This means that a printed

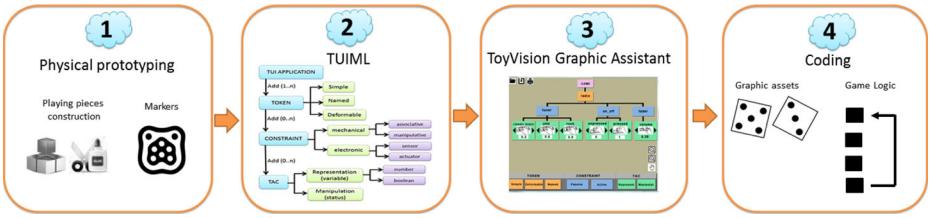

Fig. 13 The process of creating a tangible tabletop application with ToyVision

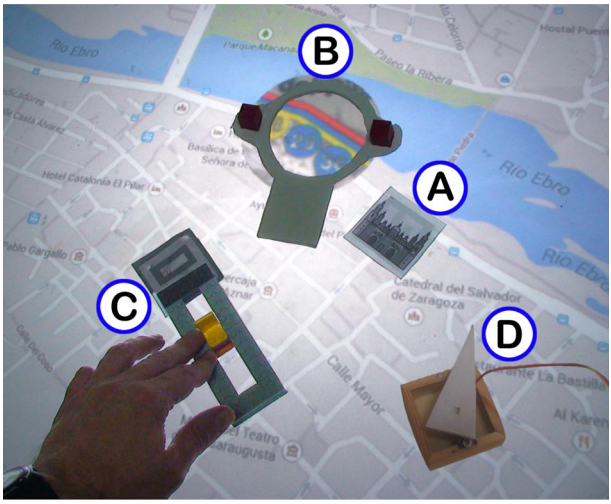

Fig. 14 MetaDesk: Geographic tangible application with building piece (**a**), lens (**b**), navigator (**c**) and compass (**d**)

marker (fiducial) has to be attached to the base of the playing piece in order to be identified and tracked by ToyVision. A collection of 18 different fiducials is provided with ToyVision.

MetaDesk playing pieces were handcrafted with cardboard, Lego™ pieces, and some pieces from other toys:

- Buildings (see Fig. 15): These pieces were built as cardboard cards, with a draw of the building on one side, and the fiducial on the other.
- Lens (see Fig. 16a): This piece was also built with cardboard. The lens required an area on its base to attach the fiducial. In order to change the different statuses of the lens piece, two holes were created on both sides of the piece, so that, small subpieces can be inserted/removed (red tokens in Fig. 16a). This way, user may activate/deactivate different visualization layers inside the lens. Subpieces to be manipulated inside the constraint holes need to have attached a printed white circle to their bases in order to be identified by ToyVision.
- Navigator (see Fig. 16b): This piece was built in Lego™ pieces. It requires an area to attach a fiducial, and a long transversal hole in which a subpiece (yellow token in Fig. 16b) can be translated vertically in order to adjust the map zoom value. Similar to the lens piece, this subpiece also requires a white circular marker in order to track its position.
- Compass (see Fig. 17): It was built with pieces from other toys and cardboard. Since this is a reactive playing piece, it requires to embed an electronic actuator (a servo motor), inside the playing piece. This actuator provides with autonomous orientation to the compass arrow. The computer system will be in charge of calculating the arrow orientation

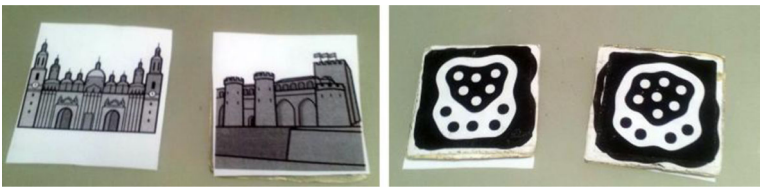

Fig. 15 Building pieces (*left*), and fiducials attached to base (*right*)

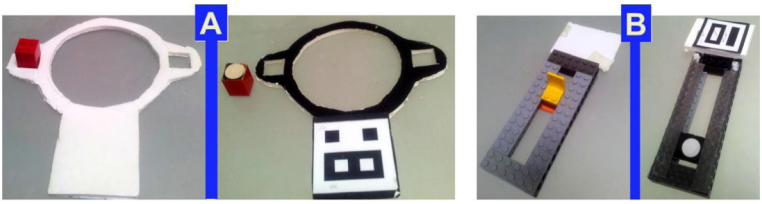

Fig. 16 MetaDesk controllers: **a**: Lens piece (*left*), and fiducials attached to base (*right*); **b**: Navigator (*left*), and fiducials attached to base (*right*)

depending on the map configuration, and send a command to ToyVision in order to reorient the compass.

5.2 TUIML

In order to model the application in ToyVision, first it must be visually represented in TUIML. Figure 18 shows the TUIML visual representation for the MetaDesk application.

Since all the playing pieces involved in MetaDesk have a very specific role in the application, all have been modelled as Named Tokens. Lens and navigator areas in which subpieces can be passively manipulated have been modelled as passive constraints. The different statuses of each constrain have been modelled has TACs. Those statuses that can be expressed by a name (on/off statuses of lens playing piece) have been modelled as manipulation TAC. Those statuses that need to be expressed as a numerical variable (zoom coefficient at the navigator playing piece) have been modelled as representation TACs. Active manipulations in the compass playing piece have been modelled as an active constraint describing the reactive piece, and a representation TAC with a numerical variable containing the orientation (0°–360°).

5.3 Graphic assistant

The TUIML visual definition has to be transferred to ToyVision using the Graphic Assistant.

ToyVision's graphic interface is divided into two main areas (see Fig. 19). The left area shows real-time image feedback from the tabletop camera sensor; the right area represents a canvas where the designer can draw the TUIML visual representation of the playing pieces. A toolbox is placed at the bottom of the canvas which is used to drag tokens, constraints and TACs to the canvas area.

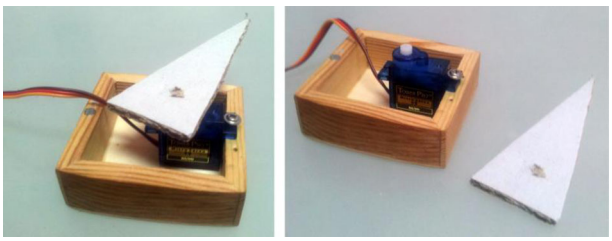

Fig. 17 Compass piece (*left*) and servo motor actuator embedded (*right*)

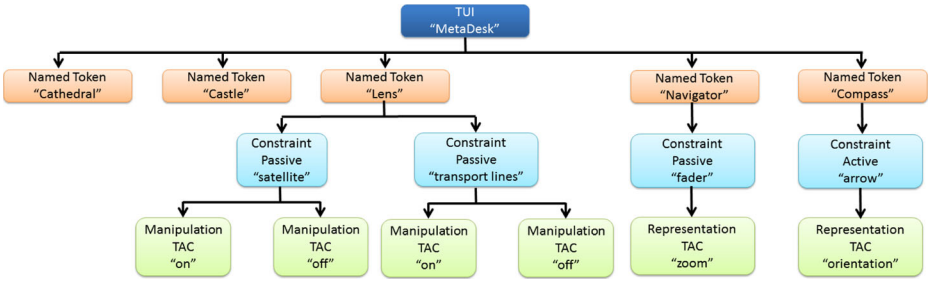

Fig. 18 Visual TUIML of the MetaDesk application

For the MetaDesk application, Fig. 20 shows the graphic interface after adding a named token that represents the cathedral building playing piece. In the visual feedback area, the designer has drawn an area to define the base of the playing piece in which the fiducial has been attached. Finally, the designer chooses the specific fiducial from the ToyVision's collection.

Once the Named Token has been created, the designer has the possibility of associating constraints to it. Figure 21 shows the fader constraint added to the navigator token to represent the fader constraint area. In order to model the fader constraint, the designers has drawn an area in the visual feedback area to delimit the hole in the base of the playing piece in which the subpiece is constrained to move.

Finally, the designer associates at least one TAC relationship to each constraint. TACs represent the different statuses that each constraint can have when the players or the system manipulates them. Figure 22 shows the "zoom" manipulation TAC added to the fader constraint. The designer has defined the kind of passive manipulation that has a meaning in the MetaDesk application. In the navigator playing piece, this manipulation consists on vertically displace the fader subpiece. This way, the zoom TAC will inform of the vertical position of the fader relative to the constraint area. ToyVision enables the designer to instantly test passive manipulations, updating and showing TAC values at the same time the playing piece is being manipulated.

Active manipulations are also modelled with constraints and TACs. An active constraint represents a reactive part of the playing piece. Figure 23 shows the ToyVision graphic assistant after adding the compass token. The "arrow" active constraint represents the servo-motor actuator embedded into the playing piece. ToyVision provides with graphic guidance to connect the actuator to the Arduino platform. The "zoom" manipulation TAC represents the status of the electronic actuator. In the compass case, this is an active status, i.e., it is the

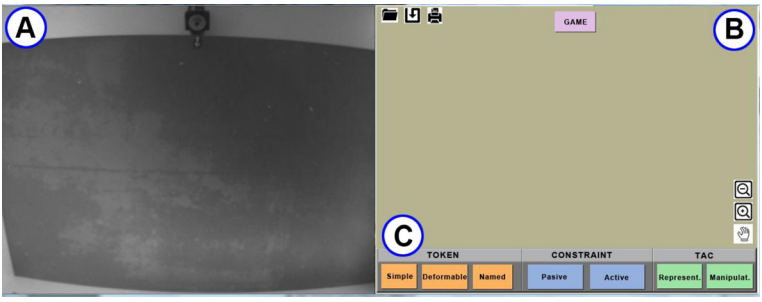

Fig. 19 ToyVision GUI. **a**: visual feedback from the tabletop camera sensor. **b**: TUIML canvas. **c**: Toolbar

Fig. 20 ToyVision GUI. Adding a named token

system, nor the user, which sets this status. By setting a numeric value to the TAC, and pressing the "test" button, the designer can test the active manipulations of the compass.

ToyVision is ready to identify all playing pieces manipulations that have a meaning in the tabletop application, once the TUIML model has been translated to the Graphic Assistant. ToyVision sends an XML message through a TCP/IP socket informing of each identified manipulation when this occurs. Thus, the host application just has to listen to this socket, and parse the messages to get knowledge of the playing pieces places on the tabletop surface. This is detailed in the next subsection exemplified with the coding of the MetaDesk host application.

5.4 Coding

In general, coding a game application mainly consists of providing the visual and audio feedback in response to the player's actions to some predefined game logic. In the case of developing a tabletop game with ToyVision, the player's actions are manipulations of playing pieces on the tabletop surface. ToyVision identifies those manipulations that have a meaning in the game and composes a XML-compliant message which contains complete information about the new status of the playing piece that has been manipulated. This message is sent to the host application through a TCP/IP socket. So, in order to implement a tabletop application, developers just need to implement a TCP/IP socket client that listens for text messages from ToyVision. Communication with ToyVision is bidirectional, which means that the host application can also send messages through the socket to ToyVision, related with active manipulations targeting reactive playing pieces.

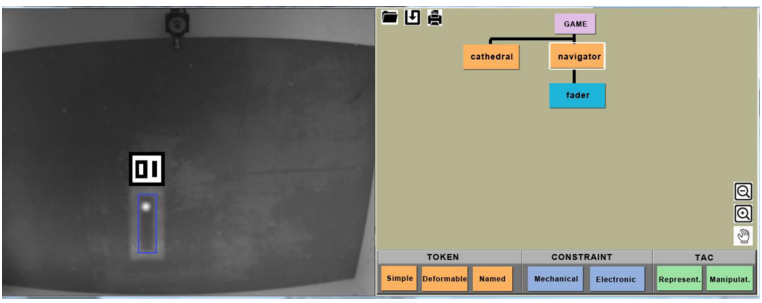

Fig. 21 ToyVision GUI. Adding a constraint to the navigator playing piece

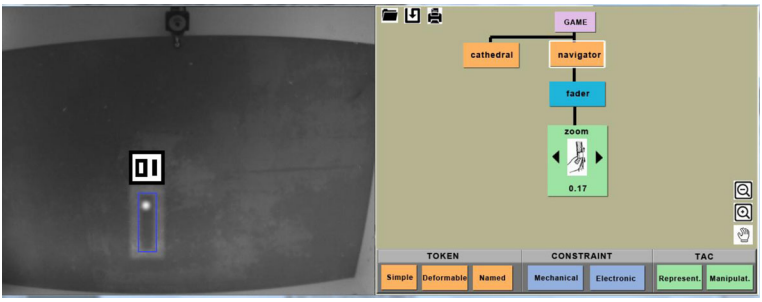

Fig. 22 ToyVision GUI. Adding a manipulation TAC to the navigator playing piece

Any development environment can be used to implement the host application, since it is only required support for implement a standard TCP/IP socket. To exemplifying the coding of a host application, following the code for the MetaDesk tabletop application is sketched. Code samples presented do not belong to any specific programing language; they aim to facilitate the understanding of the code needed.

In order to manage the visualization of the map to be projected on the tabletop surface, a Map class is created. The Map class implements the following methods:

- ShowMap(), draws the current view of the map on the tabletop surface
- ShowLensSatellite(boolean), sets the visibility of the photographic view on the lens area
- ShowLensTransportLines(), sets the visibility of the public transport lines view on the lens area
- MoveMap(mapCoordinate: 2DVector, tabletopCoordinate: 2DVector, angle: Number), repositions the map in a way that the "mapCoordinate" will be positioned in the "tabletopCoordinate". Also, the map reorients to "angle"
- MoveLens(tabletopCoordinate: 2DVector, angle: Number), sets the position and orientation of the lens graphic area in the tabletop surface
- SetPivot(tabletopCoordinate: 2DVector), set the centre of rotation of the map
- Navigate(tabletopCoordinate: 2DVector, angle: Number), moves and reorients the map relative to its pivot point
- Zoom(amount: Number), scales de map relative to amount.

Map class also implements the map coordinates of the Cathedral and Castle buildings as numeric constants: CathedralCoordinate and CastleCoordinate.

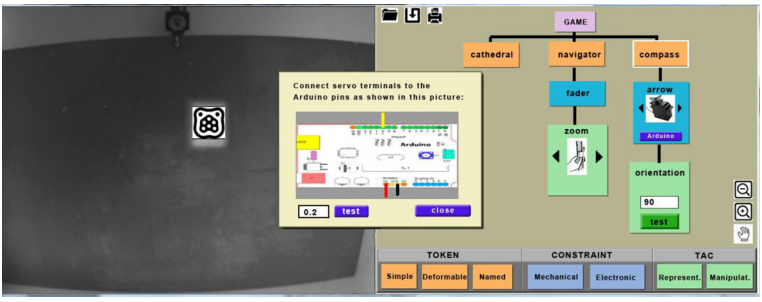

Fig. 23 ToyVision GUI. Adding the compass active playing piece

The logic of the MetaDesk application lies in two functions: Setup, and Callback. Setup function is summoned when the application launches. It is in charge of the initialization of the MetaDesk classes and variables: it instantiates the Map class, creates the TCP/IP socket and connects to the ToyVision port. It also associates this socket with the Callback function, so this is automatically summoned each time a new message arrives from ToyVision through the socket. The Callback function receives as a parameter the XML messaged retrieved from the socket. In general, for any new message that arrives from the ToyVision framework, the Callback function should:

1. Find out the name of the token that has been manipulated by the user.
2. Find out the kind of passive manipulation: user has added/moved/removed object, or a subpiece has been manipulated in a constraint.
3. If a constraint has changed its status, get the new active manipulation TAC, or get the value of the representation TAC.

Figure 24 sketches the Callback function for the MetaDesk application. For the building playing pieces, only its relative position on the tabletop surface is used to visualize its position on the map. For the lens playing piece, its TACs status determines the different visualization layers of the lens graphic area. The navigator playing piece sets the map pivot point position where it is placed. This pivot point is attached to the navigator object when it is dragged on the tabletop. When the fader subpiece is manipulated, the zoom value is extracted from its "Zoom" TAC. Finally, the compass arrow has to be reoriented when the map rotates. This can happen when the user moves the buildings and navigator playing pieces; thus, an XML command is send from the Callback function to ToyVision through the TCP socket, with the new orientation angle.

It is interesting to point out the differences between coding passive manipulations of playing pieces using ToyVision or using other popular tabletop frameworks with an architecture based on an Event Interpretation Layer, for example Reactivision [36] or CCV [6], both based on the TUIO communication protocol [22]. Coding a host application based on the TUIO protocol requires six callback functions to be implemented, three to deal with object messages (playing pieces) and three to deal with finger messages:

- addTuioObject: this callback function is summoned when a new playing piece has been added to the tabletop surface
- updateTuioObject: this callback function is summoned when an already placed playing pieces has been moved or rotated on the tabletop surface
- removeTuioObject: this callback function is summoned when a playing piece is removed from the tabletop surface
- addTuioCursor: this callback function is summoned when a new finger is placed on the tabletop surface
- updateTuioCursor: this callback function is summoned when an already placed finger has moved on the tabletop surface
- removeTuioCursor: this callback function is summoned when a finger is removed from the tabletop surface

In the MetaDesk example, the implementation of the Map class methods and the management of the events related to placing, moving and removing playing pieces from the tabletop surface is very similar to use the ToyVision framework, with the exception of using different

```
Function CallBack(msg as XML) {
//msg contains the last message received in the socket from the ToyVision framework
//get name of the object that triggered the event
name=msg.name()
//get tabletop coordinates of the object
X=msg.@x; Y=msg.@y; angle=msg.@orientation;
If (name==”Cathedral”) {
        // user moves cathedral
       //Set position and orientation of the map
       Map.MoveMap( Map.CathedralCoordinate, (X,Y), angle)
       //update compass
       Socket.send(“<compass arrow=’orientation’ value=angle”)
}

If (name==”Castle”) {
        // user moves castle
       //Set position and orientation of the map
       Map.MoveMap( Map.CastleCoordinate, (X,Y), angle)
       //update compass
       Socket.send(“<compass arrow=’orientation’ value=angle”)
}

If (name==”lens”) {
       //find out which passive manipulation triggered the event
       If (msg.@event==”added” OR msg.@event==”moved”) {
              //user moves the lens on the tabletop
              Map.moveLens( (X,Y), angle)
       }
       If (msg.@event==”removed”) {
              //user removes lens
              Map.ShowLensSatellite(false)
              Map.ShowLensTransportLines(false)
       }
       If (msg.@event==”satellite”) {
              //user activates lens visualization of photographic view
              If (msg.satellite.child()==”on”) {
                     Map.ShowLensSatellite(true)
              }
              If (msg.satellite.child()==”off”) {
                     Map.ShowLensSatellite(false)
              }
       }
       If (msg.@event==”transportlines”) {
             //the “transport lines” Constraint has been manipulated
             If (msg.transporlines.child()==”on”) {
                     Map.ShowLensTransportLines(true)
              }
              If (msg.transportlines.child()==”off”) {
                     Map.ShowLensTransportLines(false)
              }
       }
If (name==”navigator”) {
       If (msg.@event==”added”) {
            //the user places the navigator playing piece on the tabletop
            //attach the map to the navigator
            Map.SetPivot( (X,Y) )
       }
       If (msg.@event==”moved”) {
              //user moves the navigator on the tabletop

              Map.Navigate( (X,Y), angle)
              //update compass
              Socket.send(“<compass arrow=’orientation’ value=angle”)
       }
       If (msg.@event==”fader”) {
              //uses manipulates fader subpiece
              //get the zoom value
              Map.Zoom(msg.fader.zoom.text)
       }
}
```

Fig. 24 Sketched code for the Callback function dealing with the radio playing piece manipulations

callback functions to deal with adding, moving or removing events. Also, the playing pieces in ToyVision are not identified by their names but by their printed fiducial identification numbers.

The main differences emerge when dealing with other passive manipulations on the playing pieces (changing the status of the playing piece). For example, changing the status of the lens and navigator objects implies to find relations between different TuioObject and TuioCursor events; i.e., each time a cursor event arrives, it is necessary to geometrically determine if this event is just a tactile event or if it is related with the manipulation of a subpiece on a playing piece. Figure 25 shows the TUIO six callback functions with sketched code to just deal with the navigator playing piece. As it can be seen, finding out the zoom status involves complex geometric calculations to determine the position and orientation of the subpiece in relation with constraint area of the navigator object.

Another important difference between the ToyVision framework and other event based frameworks is the impact that any change in the design of the playing piece might have on the code. In the case of ToyVision, changing the arrangement of any constraint area in the playing piece, for example, has no impact on the code. On the other hand, in an event based framework, all the geometrical calculations would have to be remade.

```
Function addTuioObject(obj as TuioObject) {
        //a new playing piece has been added
        If (obj.id==1) {    //id 1 is the identification number of the navigator fiducial
             Navigator.x= obj.x
             Navigator.y= obj.y
             Navigator.angle= obj.a
             Map.SetPivot(  (navigator.x, navigator.y))
        }
}
Function updateTuioObject(obj as TuioObject) {
        //the playing piece has moved
        If (obj.id==1) {
             Navigator.x= obj.x
             Navigator.y= obj.y
             Navigator.angle= obj.a
             Map.Navigate( (navigator.x, navigator.y), navigator.angle)
                       }
}
Function removeTuioObject(obj as TuioObject) {
}
Function addTUIOCursor(cur as TuioCursor) {
}


Function updateTUIOCursor(cur as TuioCursor) {
        //a new subpiece has been moved to the tabletop surface
        //find out if this cursor is inside the area of the "fader" constraint hole of the navigator
playing piece
        //calculate distance from the navigator fiducial to the new cursor
        Dist=Distance(vector(navigator.x, navigator.y), (cur.x, cur.y))
        //calculate angle from the radio marker to the new cursor
        Angle=arctang( (cur.y-navigator.y) / (cur.x-navigator.x) )
        //And get the difference with the orientation of the navigator fiducial
        diffAngle=navigator.angle-Angle;
        //determine if the distance and angle correspond to the "fader" constraint hole
// Hole distance and angle to navigator fiducial are constants which have been previously and manually
obtained
        If (Dist>HoleDistance-distTolerance) AND (Dist<HoleDistance+distTolerance)
               AND (diffAngle>HoleAngle-angleTolerance) AND (diffAngle<HoleAngle+angleTolerance) {
               //the cursor is inside the "fader" hole, so the zoom has a new status.
               Map.zoom(Dist)
        }
}

Function removeTUIOCursor(cur as TuioCursor) {
}
```

Fig. 25 Sketched code for the TUIO callback functions dealing with the navigator playing piece

Implementing active manipulations using other frameworks based on an Event Interpretation Layer and the TUIO protocol means that the TUIO protocol is unidirectional and the framework gives no support to communicate with electronic sensors and actuators embedded in the playing piece. Therefore, it is necessary to implement hardware level functions in the host application to deal with them.

To conclude, dealing with high abstract messages coded in XML helps to lower the threshold of developing tabletop game applications based on tangible interaction. This impact on the threshold has been quantitatively evaluated, as described in the next section.

6 Evaluation

This section presents two evaluations of the ToyVision framework. The first focuses on assessing the development threshold and the second on assessing the tangible expressiveness of our framework.

There are no standard methods for evaluating frameworks and user interfaces targeting the development of TUI. However, previous experience in the field of software engineering can be useful in finding valuable metrics [7] applicable to the evaluation of ToyVision.

Currently, usability measures are the most widely accepted method of assessing the appropriateness of a software tool for a particular context. The ISO 9126–1 [20] standard suggests that measures of usability should cover:

- Effectiveness. The ability of users to complete tasks using the tool. In the context of software frameworks, effectiveness can be measured by the completion rate of fully functional applications coded by users using the framework.
- Efficiency. The level of resources consumed in performing the tasks. In the context of software frameworks, measurable resources are the amount of code lines generated to complete the tasks and the time allocated.
- User Satisfaction. Users' subjective perceptions of the tool. Satisfaction is usually measured through the use of questionnaires and attitude scales.

As stated in the introduction to this paper, a low development threshold was one of the main objectives in creating ToyVision. The threshold is related with the difficulty of learning and using the system, and its usability in general. High completion rates (effectiveness), rapid development of shorter algorithms (efficiency), and users' perceived difficulty (user satisfaction) can be used as indicators of a low development threshold when developing applications using ToyVision. These indicators can be quantitatively measured.

There are no clear quantitative evaluation methods for assessing the raising of the ceiling of expressiveness of what can be implemented using the tool, the other main ToyVision design objective. However, one way of carrying out such an evaluation is to involve final users (designers and developers of TUI) in the use of ToyVision. Although we had already a successfully and initial assessment of TUIML (in which ToyVision is based) expressiveness (see section 3.2.2) we wanted to test whether our tool was able to translate users' concepts into functional prototypes. This would reveal whether the objective of a higher ceiling of tangible expressiveness has been achieved.

The next two sub-sections describe two different evaluations carried out with ToyVision. The first focused on quantitatively evaluating the threshold of ToyVision. The second focused on qualitatively evaluating the ceiling of expressiveness of our tool but some results related with the development threshold were also obtained.

6.1 "In Lab" evaluation

A group of students studying for the Master's Degree in Computer Sciences at the University of Zaragoza (Spain) participated in the evaluation of ToyVision. The hypothesis was that ToyVision lowers the threshold of developing tangible tabletop applications compared with other current tabletop frameworks. Therefore, the evaluation was planned as a comparison between ToyVision and another tabletop framework with tangible support. The Reactivision framework was chosen for this comparison since it is one of the most popular toolkits for developing tabletop applications with tangible interaction.

The tabletop application to be developed was the MetaDesk concept previously described. The compass playing piece was not included, since it requires active manipulations, and nor Reactivision, neither other tabletop framework except ToyVision is currently giving support to those kind of manipulations on conventional objects.

Four students participated in the evaluation, none of whom had any previous experience of ToyVision or Reactivision or any other tool for developing tabletop applications. They were free to come to our lab to work with ToyVision during several days. Some students needed 2 days to finish their tasks, while others needed 5 or 6 days.

The evaluation was organized in three stages: pre-test, test and post-test, as explained below.

6.1.1 Pre-test

In order to obtain a technical profile of the participants, each of them completed a survey requesting information about their age, academic degree and skills in three technical areas:

- knowledge of programming languages
- areas of expertise in development (web, management, multimedia, tangible…)
- areas of expertise in electronics (analog, digital, Arduino…).

The participants were between 22 and 25 years old. Two of them were industrial engineers and the other two computer engineers. They all had very similar technical experience:

- All participants had some knowledge of general purpose programing languages such as C++ and Pascal, and two of them of web languages such as JavaScript and PHP.
- Two participants had experience in developing scientific applications and the other two in web development. None of them had prior experience in tabletop or TUI development.
- All participants had only theoretical knowledge about electronics, acquired during their academic studies.

6.1.2 Test

The lab was arranged to accommodate two tabletop devices connected to two computer systems with ToyVision and Reactivision frameworks installed (see Fig. 26). Printed documentation about both frameworks was provided to the participants.

The participants had to complete two tasks:

- Task 1: to develop a tabletop application using the ToyVision framework
- Task 2: to develop the same tabletop application but using the Reactivision framework

Fig. 26 Lab evaluation participant working with a tabletop

The objects were provided to the participants, so that they only had to code the tabletop host application. Moreover, they did not have to develop algorithms to visualize the map on the tabletop. A pre-made library with high abstract map visualization and navigation functions and documentation was provided to the participants. Therefore, they only had to generate the code to deal with object manipulations on the tabletop.

At the beginning of each task, a coordinator introduced the participants to the framework used in the task. The coordinator then monitored the time taken for the task. Once the participant had finished the task, the host application was stored in order to count the lines of code generated afterwards.

6.1.3 Post-test

At the end of both tasks, the participants completed two surveys (one related to ToyVision and the other to Reactivision) to evaluate the user preferences. The survey comprised a System Usability Scale (SUS) [5] and an Intrinsic Motivation Inventory (IMI) questionnaire [8].

The SUS questionnaire aimed to give a global view of the subjective assessment of ToyVision and Reactivision usability. The SUS is composed of 10 items rated on a seven point Likert Scale rating from 1 (strongly disagree) to 7 (strongly agree), in response to statements such as “I thought ToyVision/Reactivision was easy to use”.

The IMI questionnaire was intended to assess the participants’ subjective experience related with the activity carried out in the lab. IMI questionnaires are composed of a varied number of items, rated on a seven point Likert Scale rating from 1 (strongly disagree) to 7 (strongly agree). Each item measures one of six sub-scales: interest/enjoyment, perceived competence, effort, value/usefulness, perceived pressure/tension and perceived choice. We selected a set of 20 items based on their relevance to our evaluation (see Table. 2).

6.1.4 Results

Quantitative measurements comparing ToyVision and Reactivision usability are now presented:

> Effectiveness: All participants were able to develop the MetaDesk application completely for both tasks. Therefore, it can be concluded that both ToyVision and Reactivision are effective tools for developing tabletop applications based on tangible interaction.

Table 2 Questionnaire to assess subjective experience when developing with ToyVision/Reactivision

Item	Sub-scale
1. While I was working with ToyVision/Reactivision, I was thinking about how much I enjoyed it	Interest/enjoyment Pressure/tension
2. I did not feel at all nervous about using ToyVision/Reactivision	Interest/enjoyment
3. This activity did not hold my attention at all	Perceived competence
4. I think I am pretty good at using ToyVision/Reactivision	Interest/enjoyment
5. I found using ToyVision/Reactivision very interesting	Interest/enjoyment
6. I enjoyed using ToyVision/Reactivision very much	Pressure/tension
7. I felt very tense while using ToyVision/Reactivision	Interest/enjoyment
8. Using ToyVision/Reactivision was fun	Perceived competence
9. I think I did pretty well in this activity, compared to using Reactivision/ToyVision	Perceived competence
10. When using ToyVision/Reactivision, I felt in control	Perceived competence
11. I could not do some of the things I wanted to do with ToyVision/Reactivision	Interest/enjoyment
12. When using ToyVision/Reactivision, I thought about other things	Interest/enjoyment
13. When using ToyVision/Reactivision, I was aware of distractions	Interest/enjoyment
14. When using ToyVision/Reactivision I was totally absorbed in what I was doing	Interest/enjoyment
15. Using ToyVision/Reactivision was intrinsically interesting	Interest/enjoyment
16. Interacting with ToyVision/Reactivision excited my curiosity	Interest/enjoyment
17. I was so involved in ToyVision/Reactivision that I lost track of time	Perceived competence
18. I have a positive opinion of using ToyVision/Reactivision	Interest/enjoyment
19. Using ToyVision/Reactivision aroused my imagination	Interest/enjoyment
20. I am willing to participate in a similar activity in the future	

Efficiency: Table 3 shows the resources (lines of code and time to complete the application) used in each task by each participant. On average, the lines of code generated when using ToyVision were 62 % fewer than those generated for Reactivision. The time required to implement the host application with ToyVision was 32 % less than with Reactivision.
Satisfaction: User satisfaction was measured with the post-test questionnaires by calculating the SUS and IMI scores. SUS and IMI scores have a range of 0 to 100, 100 being the perfect score.

Table 4 shows the results of the SUS survey. There was a significant difference in the perceived usability of each framework. On average, ToyVision scored 24 points more than Reactivision.

As regards Intrinsic Motivation (see Table 5), there was no notable difference between the frameworks. It seems that both tasks were equally motivating for participants. The reason for this is probably the fact that the IMI survey measures participants' motivation when carrying out tasks. Since both tasks were the same (developing the same tangible application), the IMI

Table 3 Efficiency measures

Participant	Code lines		TIME (minutes)	
	ToyVision	Reactivision	ToyVision	Reactivision
1	73	169	440	770
2	56	149	180	290
3	34	117	270	300
4	42	98	180	200
Average	51	133	267	390

Table 4 SUS questionnaire scores

Participant	SUS score	
	ToyVision	Reactivision
1	83	73
2	71	66
3	96	23
4	70	65
Average results	80	56

survey reflects a high motivation for developing a tangible application irrespective of the framework used.

As stated before, the threshold is related with usability, i.e. with the difficulty of learning and using the system: high effectiveness, efficiency (shorter algorithms developed in short times), and users' satisfaction can be used as indicators of a low development threshold. In that way and after analysing the usability measures obtained in the in-lab evaluation, it can be concluded that ToyVision has effectively lowered the threshold of developing tangible tabletop applications. Since ToyVision is based on the Reactivision framework, we may conclude that this impact on the threshold is caused by the addition of a Widget layer of abstraction (the Graphic Assistant and the Widget Manager). This impact was notable in the resources needed to track passive manipulations (the only ones supported in fact by Reactivision), with 64 % less code with ToyVision than with Reactivision. Furthermore, this was also clearly perceived by participants, who scored ToyVision 24 point easier to use than Reactivision.

6.2 Workshop session

A workshop session was organized during an international conference in the field of HCI. The goal of this workshop was to give designers and developers of computer interfaces hands-on experience of developing a functional prototype of a tangible tabletop game with the support of the ToyVision framework. The workshop session covered a complete design process from the concept creation to the development of a functional prototype. In this way, the ceiling of expressiveness of ToyVision could be validated by checking whether ToyVision was giving support to all the game concepts ideated by participants. But additionally, by using ToyVision in a real scenario with designers and developers with different technical profiles, also the threshold of prototyping hybrid games with ToyVision was evaluated.

Four participants took part in a full day workshop. One of them was a developer and electronic engineer specialized in TUI with a high technical profile. Two were students on a

Table 5 IMI questionnaire scores

Participant	IMI score	
	ToyVision	Reactivision
1	86	83
2	81	83
3	89	77
5	68	68
Average results	81	77

Master's computer interaction program with generic technical skills in developing interactive applications (not TUI). The last was a product designer with low technical skills. Three coordinators managed the workshop. One coordinator offered technical support to the participants while the other two took observation notes and video-recorded the session.

At the opening of the workshop session, the participants were introduced to a tabletop device, highlighting its affordances for TUI. The workshop was organized in the following stages: concept creation, building assets and coding the game.

6.2.1 Concept creation

All the participants, as a team, carried out a brain-storming activity around a new tangible game for the tabletop device. They were provided with DIN-A3 papers, pens and several conventional children's toys to create paper-prototypes from their ideas, and to discuss and iterate their design ideas while looking for a concept simple enough to be developed during the duration of the workshop. Once the concept was finished, they also had to express it using TUIML. A TUIML template was provided to the team with some brief instructions on how to model tangible playing pieces. The participants had to draw on paper a TUIML representation of each playing piece involved in their game concept.

The concept creation activity lasted for 2 h. The participants started by looking for inspiration from the toys at their disposal (see Fig. 27-left). They examined the physical affordances of the toys and tried to figure out a way to translate them into a game to be played on a tabletop device. This activity lasted for nearly half an hour without an agreement being reached on a complete game concept. Each participant tried to defend his/her idea inspired by his/her favorite toy, but it was not possible to unify all the ideas into one simple and coherent game. After realizing that the brain-storming session was getting stuck, the participant with product design experience started leading the concept creation process. He began by suggesting a game context attractive to children: popular current TV series or movies. After a short discussion, the team agreed on the "Lord of the Rings" as a setting, which served as a base for the plot-line of the game. Two players help the hobbits Frodo and Sam to pass through a labyrinth while collecting wood. When enough wood is collected, the players build a bridge to cross a river. On the other side of the river, three orcs are waiting. One of the orcs hides the Ring, the others attack the hobbits. The players have to guess which orc hides the Ring. In order to add speed to the game, a fish monster lives in the river and destroys the bridge at time intervals, forcing the players to rebuild it.

The team built a simple "paper" prototype using toys and drawing the board on a sheet of paper. They were able to check the feasibility of their concept by simulating how the game would be played (see Fig. 27-right).

Fig. 27 Concept creation activity. *Left*: Work area, with numerous toys provided. *Right*: Paper prototype of the tangible game concept

During the process of visually modeling each playing piece into TUIML, many questions arose from the participants. They had difficulties in classifying the playing pieces (i.e., taking decisions about which playing piece was a Named, Simple, or Deformable Token, or when a TAC was representative or manipulative). The general definitions of TUIML terms were not sufficient for them to understand, and the organizers had to give some practical samples before they were able to use TUIML by themselves. Finally, the participants were able to express all the tangible manipulations involved in the game concept in the TUIML visual representation shown in Fig. 28.

6.2.2 Building assets

The participants had to create all the materials needed to implement the game, both physical (playing pieces) and virtual (virtual game-board graphics to be projected on the tabletop surface). The participants had also to transfer each playing piece TUIML into ToyVision using the graphic assistant.

This activity lasted 3 h. From the beginning, the team was divided into two groups. One group was in charge of building or adapting conventional toys into tangible tokens to be used in the game, and the other group was in charge of drawing the game graphics. The first group had plenty of conventional toys and also LEGO™ construction block sets, foam core and cardboard, as well as a wide range of sensors and actuators for embedding into toys to provide them with active manipulations where necessary. The game pieces (Frodo and Sam) and the Orcs were created recycling other toys. Pieces of clay were modeled to build the bridge. The river fish required active manipulations (autonomous movements). This was built from LEGO pieces with an embedded servo motor actuator to control its speed (see Fig. 29-left). The construction of the fish took up most of the time available for this stage. The prototyping of this piece was led by the participant with experience in TUI. Nevertheless, it was difficult for him to embed a servo motor in the toy and to achieve optimal speed of movement required for the game. The other group was responsible for creating the graphic assets to be projected on the tabletop surface. They used vector graphic design software on their own portable computers (see Fig. 29-right). No problems were encountered during this process.

The process of translating the TUIML of each playing piece into ToyVision using the graphic assistant gave rise to many questions from the participants, especially during the creation of the TAC statuses for the fish monster since it required active manipulations. Defining and testing the TACs for the fish monster required many trials and errors until they were successfully adjusted to provide the playing piece with the desired speed in its movements.

6.2.3 Coding the game

Coding the game lasted 1 h and a half. The participants had to implement the game code. To code the host application, the Action Script 3 (AS3) language was chosen as it has a very simple

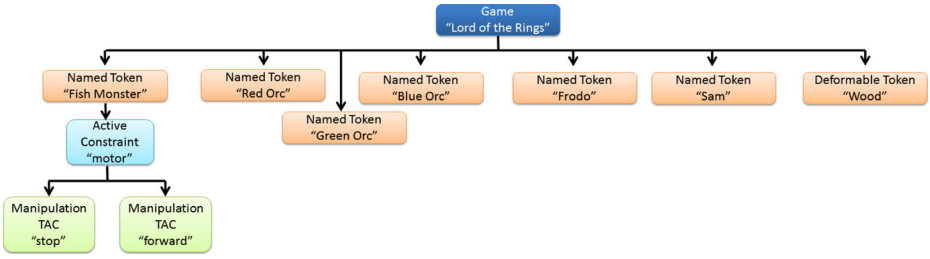

Fig. 28 Lord of the Rings: TUIML visual representation

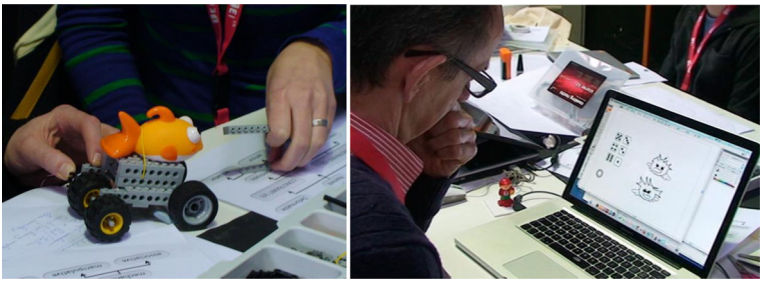

Fig. 29 *Left*: Prototyping an active playing piece. *Right*: Creating graphic assets

syntax to parse XML and to create graphic applications. Brief manuals on how to parse XML in AS3 and some AS3 code samples for dealing with graphics were provided to the participants.

The participant with development experience led the coding task. He had no experience in AS3, but after an initial introduction to the AS3 syntax he quickly started to implement the game logic. The other participants assisted in the testing of the application in the tabletop (see Fig. 30).

6.2.4 Results

As stated before, the focus of the evaluation was to assess the expressiveness of ToyVision.

The result was positive as ToyVision was proved to be sufficiently expressive to support the prototyping of the hybrid game concept ideated by the participants. All the playing pieces were completely modelled using TUIML, and they were translated to ToyVision through its graphic assistant. The concept of the "Lord of the Rings" game involved both passive and active manipulations in playing pieces. In particular:

- The three Orcs, Sam, and Frodo playing pieces (named tokens) do not require Constraints and TACs. Only their position on the board has a meaning on the game; thus, other current tabletop frameworks such as Reactivision, would have enough expressiveness to support these playing pieces.
- The wood pieces simulated with clay were supported by ToyVision as Deformable Tokens making it possible to identify the clay pieces placed on the board and their shapes and, therefore, to detect the building of a bridge. Current tabletop frameworks do not have expressiveness to support deformable objects. They are only able to support tagged objects, so they do not provide any info about object's shape. The "wood" concept would not have been possible to prototype with other current tabletop framework except ToyVision.

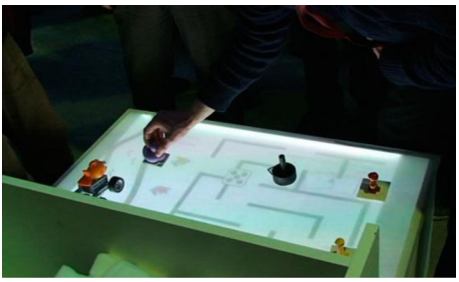

Fig. 30 Testing the game on the tabletop device

- The fish monster is a reactive playing piece with autonomous movements. Since ToyVision supports communication with the Arduino platform, the game was able to send commands to the fish monster player piece to make it move forward and backwards. No current framework is able to give support to that kind of manipulations.

Besides expressiveness, other issues were analyzed in the workshop session. At the end of the session, each participant filled in a satisfaction survey with their subjective perception of the difficulty of using ToyVision. The survey was composed of seven Likert questions (1 very easy – 5 very difficult) covering every stage of the prototyping process. The participants could also write their personal comments on each question. The survey results are summarized in Table 6. Two participants marked ToyVision as easy or very easy to use, and the other two as normal. The latter two also pointed out that some programming knowledge was required to use ToyVision, though one of them appreciated that the coding was focused on a small number of functions. All the participants highlighted the help that the ToyVision documentation provided during the session, though one participant would have appreciated more code examples. The ToyVision graphic assistant was the element perceived as the easiest in the survey, though the participants also pointed out some missing functionalities such as editing and deleting individual constraints or undoing actions. Those participants unfamiliar with TUI perceived the intrinsic difficulties of prototyping a TUI, though they also appreciated how ToyVision supports the rapid and easy prototyping of TUI concepts.

These results confirm the lowering of the prototyping threshold. ToyVision has enabled a multidisciplinary team to prototype a full functional tabletop game with tangible interaction in a short time and has been perceived by both designers and developers as easy to use. Although programing is still a barrier to designers without any programming skill, when experienced developers are involved in the prototyping process, the coding task can be carried out in short times, enabling designers to test the game in the tabletop device at an early stage and thus enabling faster design iterations. In a similar way, although ToyVision offers support for active manipulations in playing pieces, embedding electronic components in toys is still a task that requires some electronic skills and previous experience. Decisions such as which kind of actuator is suitable, how to embed it in the toy and how to model active manipulations are still too complicated for participants without experience with these kinds of devices. These issues would have to be carefully considered and improved.

Similarly, the observation notes taken during this evaluation session helped us to identify several usability problems, especially those related with missing functionalities in the Graphic Assistant. We are addressing these issues for the next version of ToyVision, as well as

Table 6 Perceived difficulty of creating a game with ToyVision

PERCEIVED DIFFICULTY OF…	DIFFICULTY (1 VERY EASY – 5 VERY DIFFICULT)				
	P1	P2	P3	P4	average
Using ToyVision	3	3	2	1	2.25
Finding functionalities	3	3	3	1	2.5
Using the graphic assistant	1	3	2	1	1.75
Coding the game	3	2	3	1	2.25
Consulting help documents	2	1	2	3	2
Developing TUI	3	4	3	2	3
Learning curve	3	2	3	1	2.25

improving the documentation by including more tutorials and practical samples. This is part of the future work outlined in the next section.

7 Conclusions and future work

In this paper we have presented ToyVision, a software framework that aims to raise the ceiling of expressiveness of tangible interaction in tabletop games while keeping the development threshold low. This goal was motivated by an analysis of current software frameworks supporting tangible interaction in tabletop applications. This analysis revealed that current frameworks are based on an Event Interpretation layered architecture which only provides for very simple manipulations of objects on the tabletop surface (place, move and remove). This situation is hindering designers from fully exploring the physical feasibilities of the manipulation of objects.

ToyVision adds a new Widget layer of abstraction over a framework with an Event Interpretation architecture (Reactivision), which is responsible for processing low abstract tabletop events into high abstract ones describing the status of playing pieces in the context of the host game. In order to describe the status of playing pieces, the ToyVision Widget layer implements a TUIML: a modelling language designed to express any tangible application in a way that can be understood by a computer system. The adoption of the TUIML has enabled ToyVision to support new passive and active manipulations on conventional objects. In particular, ToyVision, unlike current tabletop frameworks, provides support to passive manipulations of compound objects (inserting/removing sub-objects into/from another one), and the passive manipulation of moveable pieces (moving/rotating part of an object) as well as a wide range of active manipulations achieved by managing electronic actuators embedded into conventional objects.

The process of designing a tabletop application with ToyVision is supported by a Graphic Assistant which enables all the active and passive manipulations involved in the host application to be visually modeled in TUIML. On the other hand, the implementation process is supported by a client-server architecture, which enables ToyVision to be used with any development environment, and a communication protocol based on XML messages, which simplifies the task of parsing messages coming from the framework.

Evaluation sessions with HCI students, designers and developers of interactive applications have shown that ToyVision is effectively lowering the threshold of developing tabletop applications. Coding with ToyVision has proved to be more efficient and has been perceived by users as easier than with current frameworks. Moreover, ToyVision is sufficiently expressive to support new concepts of hybrid board games ideated by participants during the workshops and is unique in supporting deformable objects and active manipulations.

The ToyVision framework has been designed and implemented with reusability and extensibility in mind. For future work on ToyVision, new passive and active manipulations are planned such as 3d tabletop interaction (manipulation of objects over the tabletop surface). Also, we are considering ToyVision's suitability for bigger areas of interaction such us rooms and playgrounds. These environments may be attractive for creating new concepts of hybrid videogames, but they will also present new design and development challenges.

Further evaluations and workshops with designers and developers of interactive applications and videogames are planned. Moreover, the evaluation of ToyVision with children is also being considered, using development environments especially designed to teach children computer programing concepts such us Scratch [37].

ToyVision is open-source software, and it is available at http://www.toyvision.org.

Acknowledgments We thank Elisa Ubide for helping in the development of ToyVision. We also thank all the students, designers and developers who participated in our evaluation sessions. This work has been partly financed by the Spanish Government through the DGICYT contract TIN2011-24660.

References

1. Al Mahmud A, Mubin O, Shahid S, Martens JB (2008) Designing and Evaluating the Tabletop Game Experience for Senior Citizens. Proceeding: NordiCHI, October, 20–22
2. Arduino: http://www.arduino.cc/
3. Bespoke: http://www.bespokesoftware.org/multi-touch
4. Bollhoefer KW, Meyer K, and Witzsche R (2009) Microsoft surface und das Natural User Interface (NUI). Technical report, Pixelpark, Feb
5. Brooke J (1996) SUS-A quick and dirty usability scale. Usability evaluation in industry, 189, 194
6. CCV: Community Core Vision Web: http://nuicode.com/
7. Clements P, Kazman R, Klein M (2002) Evaluating software architectures: methods and case studies. Addison-Wesley, Boston, p 323
8. de Souza Alcantara T, Ferreira J, & Maurer F (2013) Interactive prototyping of tabletop and surface applications. In Proceedings of the 5th ACM SIGCHI symposium on Engineering interactive computing systems (pp. 229–238). ACM
9. Deci EL, Ryan RM (1985) Intrinsic motivation and self-determination in human behavior. Springer, New York
10. Echtler F, Klinker G (2008) A multitouch software architecture. In Proc of NordiCHI '08. pp. 463–466
11. Graffiti framework. https://code.google.com/p/grafiti/
12. Greenberg S (2002) Enhancing creativity with groupware toolkits. Groupware: Design, Implementation, and Use. Springer. 2003. pp. 1–9
13. Greenberg S and Fitchett C (2001) Phidgets: easy development of physical interfaces through physical widgets. In UIST '01, pages 209–218
14. Hansen TE, Hourcade JP, Virbel M, Patali S and Serra T (2009) PyMT: a post-WIMP multi-touch user interface toolkit. International Conference on Interactive Tabletops and Surfaces (ITS '09). ACM, pp. 17–24
15. Heijboer M and van den Hoven E (2008) Keeping up appearances: interpretation of tangible artifact design. Proc. of the 5th Nordic conference on Human-computer interaction: building bridges (NordiCHI '08) pp162–171
16. Heng X, Lao S, Lee H, & Smeaton A (2008) A touch interaction model for tabletops and PDAs. In Proc. PPD '08
17. Hinske S and Langheinrich M (2009) W41K: digitally augmenting traditional game environments. Proc. of the 3rd international Conference on tangible and Embedded interaction (2009). TEI '09, 99–106
18. Holmquist LE, Redström J, & Ljungstrand P (1999) Token-based access to digital information. In Handheld and Ubiquitous Computing (pp. 234–245). Springer Berlin Heidelberg
19. Hornecker E, & Buur J (2006) Getting a grip on tangible interaction: a framework on physical space and social interaction. In Proceedings of the SIGCHI conference on Human Factors in computing systems (pp. 437–446). ACM
20. Ishii H. & Ullmer B (1997) Tangible bits: towards seamless interfaces between people, bits and atoms. In Proceedings of the ACM SIGCHI Conference on Human factors in computing systems (pp. 234–241). ACM.
21. ISO 9126–1, Software engineering. 2001. ISO Press
22. Iwata T, Yamabe T, Poloj M, and Nakajima T (2010) Traditional games meet ICT: a case study on go game augmentation. Proc. of the fourth international conference on tangible, embedded, and embodied interaction (TEI '10). Pp. 237–240
23. Kaltenbrunner M, Bovermann T, Bencina R, and Costanza E (2005) TUIO: A protocol for table-top tangible user interfaces. In 6th Int'l Gesture Workshop
24. Klemmer SR, Li J, Lin J, & Landay JA (2004) Papier-Mache: toolkit support for tangible input. In Proceedings of the SIGCHI conference on Human factors in computing systems (pp. 399–406). ACM
25. Krzywinski A, Mi H, Chen W, Sugimoto M (2009) RoboTable: a tabletop framework for tangible interaction with robots in a mixed reality. In: In proceedings of the international conference on advances in computer enterntainment technology (ACE '09). ACM, New York, pp 107–114
26. Leitner J, Haller M, Yun K, Woo W, Sugimoto M, Inami, M, Cheok AD, and Been-Lirn HD (2010) Physical interfaces for tabletop games. Comput. Entertain. 7, 4, Article 61, 21 pages
27. Li Y, Fontijn W, and Markopoulos P (2008) A tangible Tabletop Game Supporting Therapy of Children with Cerebral Palsy. 2nd International Conference on Fun and Games, Springer-Verlag, pp. 182–193
28. Libavg web http://www.libavg.de/
29. LibTISCH: Library for tangible Interactive Surfaces for Collaboration between Humans. http://tisch.sourceforge.net/

30. Lin H.-H, and Chang T.-W (2007) A camera-based multi-touch interface builder for designers. In Human-Computer Interaction. HCI Applications and Services
31. Marco J, Baldassarri S, & Cerezo E (2013) ToyVision: a toolkit to support the creation of innovative board-games with tangible interaction. In Proceedings of the 7th International Conference on Tangible, Embedded and Embodied Interaction (pp. 291–298). ACM
32. Marco J, Cerezo E, & Baldassarri S (2012) ToyVision: a toolkit for prototyping tabletop tangible games. In Proceedings of the 4th ACM SIGCHI symposium on Engineering interactive computing systems (pp. 71–80). ACM
33. Marco J, Cerezo E, Baldassarri S (2012) Tangible Interaction and Tabletops: New Horizons for Children's Games International Journal of Arts and Technology (IJART). Vol. 5, Nos. 2/3/4. 2012. pp.151–176 ISSN: 1754–8853. Ed. Inderscience
34. Mikhak B, Lyon C, & Gorton T (2003) The Tower system: A toolkit for prototyping tangible user interfaces. Submitted as a long paper to CHI 2003
35. Myers B, Hudson SE, Pausch R (2000) Past, present, and future of user interface software tools. ACM Trans Comput Hum Interact 7(1):3–28
36. Openexhibits web: http://openexhibits.org
37. Reactivision: http://reactivision.sourceforge.net/
38. Resnick M, Maloney J, Monroy-Hernández A, Rusk N, Eastmond E, Brennan K, Kafai Y (2009) Scratch: programming for all. Commun ACM 52(11):60–67
39. Rogers Y and Rodden T (2004) 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. pp. 45–79
40. Shaer O and Jacob RJK (2009) A specification paradigm for the design and implementation of tangible user interfaces. ACM Trans. Comput.-Hum. Interact. 16, 4, Article 20
41. Shen C, Vernier F, Forlines C, and Ringel M (2004) DiamondSpin: an extensible toolkit for around-the-table interaction. In Proc. CHI '04, pages 167–174
42. TouchLib: http://nuigroup.com/touchlib/
43. Trackmate: http://trackmate.sourceforge.net
44. Ullmer B, & Ishii H (1997) The metaDESK: models and prototypes for tangible user interfaces. In Proceedings of the 10th annual ACM symposium on User interface software and technology (pp. 223–232). ACM
45. Wright M (2005) Open sound control: an enabling technology for musical networking. Organised Sound 10(03):193–200

Javier Marco obtained a Ph.D. in Computer Science Engineering in 2011 at the University of Zaragoza (Spain). His thesis researched the benefits of tangible interfaces for children and their involvement during design and evaluation stages.
He was a visiting researcher at the ChiCI Group at the University of Central Lancashire (UK) and at the M-ITI at the University of Madeira (Portugal), under a Carnegie Mellon post-doc program.
He is in the scientific committee of the Interaccion International Conference and Tangible and Embedded Interaction International Conference. He also has organized several workshops in different international symposiums dealing the design and implementation of tangible applications.

Eva Cerezo received a Ph.D. degree in Computer Science in 2002 from the University of Zaragoza. She is currently an Associate Professor at the Computer Sciences and Systems Engineering Department at the University of Zaragoza and a founding member of GIGA AffectiveLab. Her research areas are affective multimodal human computer interaction, tangible interaction and virtual humans. She is author of more than 70 international publications that take in conjunction more than 300 citations. She is editor of the Advances in Human-Computer Interaction Journal and regular reviewer of several national and international conferences like the "ACM CHI Conference On Human Factors in Computing Systems" and the "International Conference on Intelligent User Interfaces".

Sandra Baldassarri received a B.Sc. in Computer Science from University of Buenos Aires, Argentina, in 1992 and a Ph.D. in Computer Science Engineering from the University of Zaragoza, Spain, in 2004. She is Assistant Professor at the University of Zaragoza (Spain) and founder member of the AffectiveLab of the Advanced Computer Graphics Group (GIGA) at the University of Zaragoza. Her research interests includes virtual humans, affective computing, multimodal interfaces and tangible and natural interaction.