Unifying Events from Multiple Devices for Interpreting User Intentions through Natural Gestures

Pablo Llinás, Manuel García-Herranz, Pablo A. Haya, and Germán Montoro

Dept. Ingeniería Informática, Universidad Autónoma de Madrid
C. Fco. Tomás y Valiente, 11, 28049 Madrid, Spain
{Pablo.Llinas, Manuel.GarciaHerranz, Pablo.Haya, German.Montoro}@uam.es

Abstract. As technology evolves (e.g. 3D cameras, accelerometers, multitouch surfaces, etc.) new gestural interaction methods are becoming part of the everyday use of computational devices. This trend forces practitioners to develop applications for each interaction method individually. This paper tackles the problem of interpreting gestures in a multiple ways of interaction scenario, by focusing on the abstract gesture rather than on the technology or technologies used to generate it. This article describes the Flash Library for Interpreting Natural Gestures (FLING), a framework for developing multi-gestural applications integrated and running in different gestural-platforms. By offering an architecture for the integration and unification of different types of interaction, FLING eases scalability while presenting an environment for rapid prototyping by novice multi-gestural programmers. Throughout the article we analyse the benefits of this approach, comparing it with state of the art technologies, describe the framework architecture, and present several examples of applications and experiences of use.

Keywords: FLING framework, Multi-touch interface, multiple input peripherals, application development

1 Introduction

The evolving progression of HCI interfaces promises a brilliant future to seamless control and handling of intelligent devices. New ways of interaction are invented and integrated in the search for the best and most natural method of interaction between humans and computers. Their goal is to allow for a more intuitive and natural way of expression when communicating with computers. An immediate consequence of this evolution is the creation of new input devices with which we can operate. Today, intelligent homes can be fitted with all sorts of sensors (temperature, movement, pressure, fingerprint…). Also, existing devices are enhanced with extra sensing capabilities [1], such as multi-touch screens, accelerometers and voice recognition in smartphones like Apple's iPhone or Google's Nexus S.

As computers integrate deeper into our daily lives, the most natural way to use them becomes dependent to each particular scenario. A single application will have to allow different interaction methods across different settings in order to offer the most

suitable experience each time. For example, for talking to a friend, video conferencing will be suitable at home, where you can give visual and hearing attention to the other person, while text chatting will be more adequate in a football stadium during a match, where being heard can be very difficult.

From a programmer's point of view, more input devices entail a higher complexity when treating input. Different peripherals offer different interaction possibilities, which generate input events that need to be processed in order to interact with the applications. To reduce the difficulty of dealing with a large (and sometimes unknown) number of input devices, cross-platform frameworks are used to take care of these devices, and for developing applications using a set of abstract input events.

The difficulty of programming for cross-platform ubiquitous control has been shifted from treating each input device, to learning how to work with cross-platform frameworks. In a world of constant upgrades in hardware gadgets and increasing intelligent sensing devices, developers need unification and standardization of user intention recognition regardless of the technology employed.

2 Related work

The advantages of allowing the user to choose the most adequate input device for interaction with applications are therefore being studied and proven profitable. Multimodal interaction frameworks [2, 3, 4] pose valuable precedents of frameworks for using different input devices for human-computer interaction, using input events either independently or combined. However, we lack a higher degree of flexibility when translating device actions (fixed for each input peripheral) into application actions (which trigger the available operations on the computer). One programmer could require a sequence of device events to carry out an operation, while another would be looking for a different one. Moreover, different users using the same application may require different responses to similar input patterns.

This issue is of particular importance in multitouch environments in which events are usually composed of several device inputs (e.g. multiple fingers touching a display) that have to be, furthermore, interpreted from complex raw data, such as blob identification and analysis. In order to ease the process of interpreting device inputs in camera-based multitouch systems, several low-level frameworks such as TouchLib [5], reacTIVision [6], Community Core Vision (CCV) [7] or Touchè [8] are available to identify significant blobs as screen touches ready to process. This interpretation results, in most cases, in TUIO messages codifying each finger and its evolution on the surface. This kind of frameworks provides a first level of abstraction, allowing applications to listen from a single channel to every finger interaction in a unified manner. Advanced examples of these are BBTouch [9] and LightTracker [10], which improve the aforementioned frameworks by allowing an advance tunning of the recognition parameters, or VVVV [11], which allows associating blob input with different visualization methods using a visual programming paradigm.

Nevertheless, multitouch interactions are most of the times composed of several fingers and, therefore, multiple events have to be reinterpreted, according to the element of the UI over which they are acting, to form a high level global gesture (e.g.

two fingers moving are interpreted as moving apart from each other). Finally, the interpreted gestures have to be associated with a particular action. Some systems such as PyMT [12] or Grafiti [13] provide a transparent mechanism to distribute events among the elements of the UI as well as basic interpreters for the most common gestures such as move, resize or rotate.

However, the number of different gestures in multitouch systems can grow far beyond the basic ones and the actions associated with each of them may vary among applications or, in a single application, from component to component. Thus, a higher degree of abstraction, modularization and composition is needed. Systems such as Surface SDK [14] or DiamondTouch SDK [15] provide this kind of flexibility to a reasonable degree but are constrained to a particular platform, limiting their extensibility and scalability in an ever-increasing world of multitouch hardware solutions. Similarly, proprietary frameworks such as GestureWorks [16] constrain their extensibility, compared to their open source counterparts, in an ever-increasing world of gestures.

Nonetheless, while multitouch systems are gaining popularity, they are far from being an alternative to the standard mouse-keyboard paradigm and will have to further cohabitate with new interaction devices such as the Wii remote or Kinect. Thus, systems such as Squidy [17] provide a low-level alternative to unify various device drivers, frameworks, and tracking toolkits in one common library, overcoming the limitations of higher level solutions such as GestureWorks [16] or Grafiti [13] designed to work just with multitouch events.

From the high level open source alternatives designed to support different input mechanisms, define new gestures and dynamically associate actions with them, we can distinguish between language dependent and language independent frameworks. Those that rely on a separated event system, allowing to program in the language of our choice, force the programmer to provide a description of the UI and its components to the event interpreter layer. This is done either explicitly, therefore, adding an extra complexity to the programming process that prevents a rapid prototyping, as in SparshUI [18] and Midas [19], or implicitly through a widget library, as in libTISCH [20], simplifying the communication channel but making very difficult to interpret events outside the boundaries of the widget.

MT4j [21] (MultiTouch for Java) is, on the other hand, a language dependent framework designed for rapid prototyping and gesture extension. Making use of the well-known event-listener java architecture, it allows components to listen to particular gestures without modifying or adding complexity to the UI design and coding.

FLING, the framework described in this article, falls into this last category but relies on Adobe's Flash, instead, as a well-known platform to graphical designers. Thus, graphic design and program logic can be easily separated and distributed among designers and programmers, allowing for good-looking rapid prototyping. In addition, contrary to systems such as MT4j, FLING provides a double distribution mechanism. Through one channel, interpreted gestures are propagated to every component, allowing them to know what is happening to themselves as well as to the rest of the components. Through the second one, raw events are propagated too, allowing to program global or partial interpretations in any component of the UI, whether they fall in or outside its boundaries.

3 The FLING Framework

FLING (Flash Library for Interpreting Natural Gestures) is a cross-platform multi-gesture framework for developing Adobe Air and Flash applications using the ActionScript 3.0 programming language.

FLING has been developed under the following principles:

3.1 Platform-independent

FLING shares the "write once, run anywhere" philosophy. In order to run on the highest number of computing devices, a platform-independent programming language is a must. ActionScript 3.0, the language behind Adobe Flash and Air applications, was chosen for this purpose. Applications run on any desktop operating system (Windows, Mac and Linux) and also on Android smartphones. A special version, Flash Lite, can even be used on more basic mobile phones [22].

We chose this way of deploying cross-platform applications instead of using adapted interfaces because we believe that technologies such as Java, HTML and Adobe Flash, which in the past have been secluded to being used on standard PCs, will soon be running with the same capabilities on even the most basic portable devices. The processing gap between different computing architectures is closing-in, and today we can find smartphones[1], tablets[2], notebooks[3] and desktop computers sharing very similar dual-core processing computational power.

The Adobe family of products provides a very powerful and robust set of tools for the visual design of graphical interfaces. Also, when integrating these graphical elements into the Flash platform for adding programming logic, there is full compatibility and interoperability between multimedia contents. A drawing made in Photoshop can be imported into Illustrator to get a vector graphic that then can be inserted into Flash and get animated. The final symbol can be accessed and manipulated from code using the ActionScript language. And finally, the result will be an Adobe Air or Flash multi-platform application.

3.2 Useful for rapid prototyping

For first-time cross-platform multi-gesture application developers, FLING provides a basic manipulation of visual objects using the traditional mouse and keyboard, and multi-touch surfaces. Extending from the base FLING object class, object movement, rotation, resizing and physics engine (inertia, collisions, gravity…) can be enabled with a single line of code, as seen below.

[1] http://www.pcworld.com/article/204947/lg_announces_smartphones_with_dualcore_processor.html
[2] http://www.tgdaily.com/mobility-features/49854-nvidia-showcases-dual-core-tegra-2-tablet
[3] http://liliputing.com/2010/10/samsung-launches-nf310-dual-core-netbook-with-hd-display.html

Example of extending from the base FLING object class

```
public class NewObject extends FlingObj{
   public function NewObject():void{
      movable = resizable = rotatable = physics = true;
   }
}
```

The visual symbol of class "NewObject" will respond according to the activated capabilities upon standard input events. In the case of the mouse and keyboard, the object will respond as shown on Table 1. Using a multi-touch surface, the object will respond as shown on Table 2.

This initial functionality allows for basic application interaction without getting into event interpretation or gesture handling. It is oriented towards Adobe Flash and Air developers with minimal knowledge in device input processing, and allows them to create cross-platform multi-gesture applications which can run on multi-touch surfaces and standard computers without making the effort of learning a complex new framework.

FLING requires little adaptation for first-time programmers. One of the basic fundaments of its operation is a tree-structured hierarchy of symbols or visual objects (Figure 1). A unique FLING root object, representing the application itself, contains all device parsers and gesture interpreters. All symbols used in an application must be children to this root object (either directly, or further down in the tree of child objects). This is required because it represents the order in which visual objects are layered in an application. The root object is the background of the application, and it represents the lower-most layer. The next level in the hierarchy of children objects represents the layer on-top of the background layer, and so on.

Table 1. Default actions triggered by mouse and keyboard input events.

Input device gesture	Action performed on object
Drag with mouse	Object moves under mouse pointer
Control key + Drag with mouse	Object resizes in regard to its center and the mouse pointer
Shift key + Drag with mouse	Object rotates in regard to its center and the mouse pointer

Table 2. Default actions triggered by multi-touch finger input events.

Input device gesture	Action performed on object
Slide one finger over the object	Object moves according to the finger movement
Pose two fingers over object and separate or join them	Object resizes according to distance variation between fingers
Pose two fingers over object and move one around the other	Object rotates acording to angle variation of line joining fingers

The FLING framework relies on Flash's native visual hierarchy of objects for target identification, allowing disambiguating when objects overlap. As all Flash objects have one (and only one) parent and their insertion order decides among siblings, no inconsistencies can occur between FLING's object targeting and Flash's visual

representation. Using the native structure for object nesting presents a logical way to navigate through objects and can, therefore, be already found in most applications.

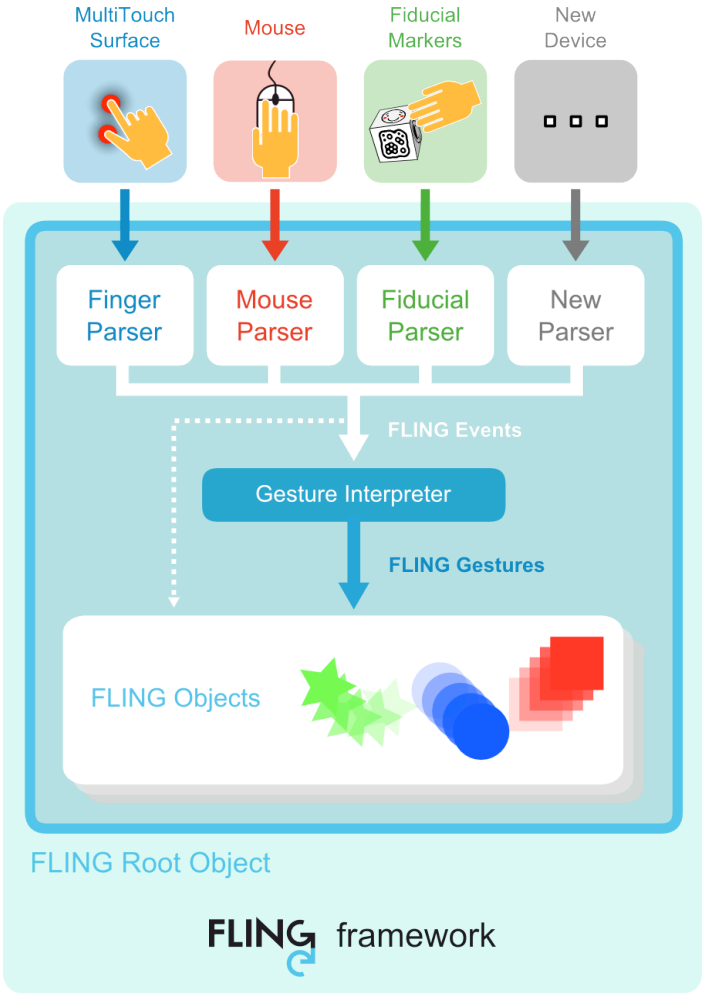

Fig. 1. Event parsing and interpretation inside the FLING framework.

3.3 Allows customization of event interpretation and triggered actions

To extend the default functionality offered as standard when working with FLING, developers can tune and enhance the interpretation of device events into gestures, and the reactions upon receiving interpreted gestures. This can be achieved with minimal code modification because of the predisposition to customization.

The three main elements with which FLING works are events, gestures and actions. Events are signals from input devices that are processed and homogenized

into a common stream of events by the device parsers. Each device has an associated event parser that in some cases makes use of external drivers to capture input signals.

Gestures are interpreted by another module which receives events parsed from all the input devices available to the user. The interpreter can use the combination of these input events and knows of all existing interface objects in order to make sense of the user's intentions. It outputs FLING Gestures, which consist of recognized user intentions. FLING is then responsible of propagating and making gesture events reach their correct target. Once an interactive object receives a gesture event (or FLING Gesture), it reacts according to the triggered action for that gesture. Some triggered actions are preconfigured by default (as is the case of the move, rotate and resize actions mentioned in section 3.2) and others are left blank, but all of them are customizable to fit the needs of each application.

As an example of triggered action customization, we will take the resizing gesture and change its default action so that instead of changing size, the target object will change its transparency. The code added to the target object for modifying this behaviour is as follows:

Example of triggered action customization

```
override public function onRotateGesture(gesture:FlingGesture):void{
     this.alpha += gesture.varAngle%360;
}
```

Normally, the "varAngle" property of the FLING event received is used to rotate the object accordingly. Instead, we are using this value, normalized between -1.0 and 1.0, to alter the alpha value of the object, hence changing its transparency. This is just an example of how easy it is to change the behavior associated to a particular gesture.

Another example of framework customization can be seen when modifying the way device events are interpreted into gestures. We will take the multi-touch gesture interpreter and add some code to recognize a new gesture. The new gesture will consist in a quick slide of two fingers (*cursor1* and *cursor2*) over a visual object, from top to bottom. This gesture will be called the "minimize" gesture, and could be linked to the minimizing action, but this is entirely up to the programmer.

The code needed inside the finger interpreter to recognize this gesture is the following:

Example of gesture customization

```
if(numCursors == 2){
     if(cursor1.type == cursor2.type == CURSOR-EXIT){
            if(slideDirection(cursorStream1)
            == slideDirection(cursorStream2) == SLIDE-DOWN){
                  flingGestures.push(new FlingGesture("MINIMIZE"));
            }
     }
}
```

3.4 Cross-platform multi-gestures

Another requirement for ubiquitous control is the possibility of multiple device input (Figure 2). FLING comes with parser modules for mouse & keyboard, multi-touch surfaces (such as Microsoft's Surface[4], MultiTouch Cell[5] and the reacTable [23]), pressure tokens (special objects recognizable by pressure marks) and fiducial markers [24]. These input devices can be used right away, and can be configured and customized to fit the application's needs.

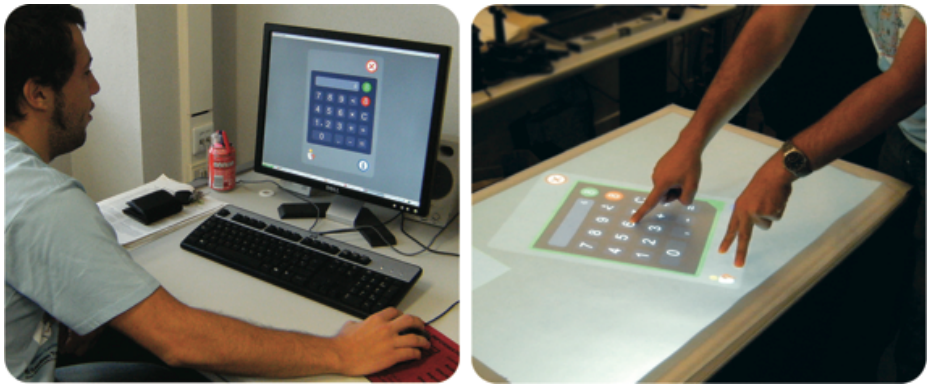

Fig. 2. A multi-platform running environment and multi-gesture input interaction enable applications run on the device which most suits each scenario.

For advanced necessities, it also allows for new input devices to be used when needed. The steps for adding a new input device are:

1. Identify the input data format in which the device sends events from the user's interaction. If not provided by the standard Adobe Flash / Air libraries, parse the device signals into atomic non-interpreted events. These atomic events should match the device's interaction possibilities. For example, a joystick will have coordinate events indicating its position, and button events indicating any change in the state of their manipulation.
2. Add a gesture interpretation module for the new input device, which groups atomic input events into recognized gestures. For example, using multi-touch surfaces, two fingers separating from each-other over a same object are recognized as a resize event by default. Gesture metrics (e.g.: resizing value) are also included in the interpreted gesture. The gesture interpretation module sends recognized gestures, called FLING events, as shown on Figure 1, which can be equivalent to those already handled, or new ones which are exclusive to each input device.
3. FLING will forward both atomic (or raw) events and gesture (or FLING) events to all objects, as described in section 3.2, for them to react consequently. All of the

[4] http://www.microsoft.com/surface/
[5] http://multitouch.fi/products/cell/

aspects of input device reading and interpreting can be openly customized to obtain the required functionality, as stated on section 3.3.

Multi-touch surfaces are one of the input devices currently supported by FLING (Figure 3). On a typical multi-touch table, fingers placed on-top of the touch panel (A) create light blobs that can be tracked by a video camera [25]. The camera sends a video stream (1) to a blob driver (B), such as reacTIVision [6], Touchlib [5] or Community Core Vision [7], which then outputs finger events using the TUIO protocol [26] through a data socket (2). FLING (C) connects to this data socket and uses the incoming finger events from the panel to begin the interpretation and propagation processes (3). Finally, the application reacts to the intentions expressed by the user, and evolves its visual interface (4) which is displayed on the same surface used as touch panel (D).

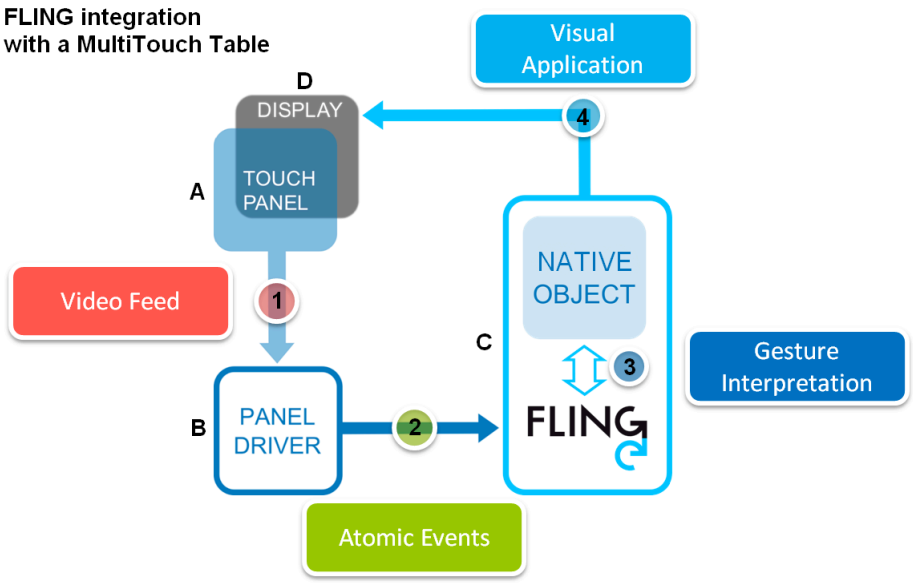

Fig. 3. Multi-touch surface input handling using the FLING framework.

Inside the FLING framework, a parser module reads unprocessed finger events obtained from the hardware drivers, and homogenizes them into standard input device events. This homogenization consists in a translation of native signals to a common event class, which holds the same information but in a standard form. Then, the finger interpretation module is in charge of interpreting and generating gesture events that FLING will send to the appropriate application object. Interactive objects will react in response to these gesture events, and will also receive the unprocessed input events in case more information is required.

The format of gestures produced from a multi-touch surface is compatible with those generated using mouse, fiducial or token devices. Interactive objects from the application receive complete and descriptive gestures with the user's intentions, but don't have to worry about the input used to express those intentions. Additionally,

unprocessed events from the input devices are also delivered to application objects for the event of needing detailed information about raw input data.

3.5 Progressive learning curve

The idea behind offering both rapid prototyping and advanced customization capabilities is to create a progressive and smooth learning curve when working with the framework. Existing cross-platform frameworks offer extensive functionality and very advanced user expression recognition, but are difficult to use at first and require a lengthy learning period. We experienced these difficulties when trying commercial products such as Gestureworks[16] or the TUIO Flash client library [27] for receiving multi-touch events. It is normal to go through a number of learning steps while getting comfortable using a new framework, and a learning curve similar to the one described by Gaines [28] is typically experienced.

4 Sample Applications

FLING has been thoroughly tested and used for the development of full applications which have made their way into educational and experimental projects. Different working environments with adapted input devices have accommodated these applications, and people with varying levels of knowledge and expertise have given them a try.

4.1 Therapy applications

To serve as a complement for people with disorders, games such as Simon® and Gesture Hero (Figure 4) were developed under particular requirements. They were designed in cooperation with teachers and assistants of people with Down syndrome and Alzheimer's disease to serve as therapy work for these collectives.

This supervised use has served to gather information about interaction habits and difficulties that has been used in FLING's design.

Fig. 4. On the left, the Simon® game, a memory training application. On the right, the Gesture Hero game, a psychomotor skill game.

Specific gestures were added to the Gesture Interpreter module in order to supply these applications of their interaction requirements.

4.2 Educational applications

In addition to the therapy applications, other educational applications have been developed to be included in the regular activities of people with Down syndrome during their classrooms. Examples are the Postman Game and the Price is Right game shown on Figure 5.

The Postman Game consists in handing packages to the right recipient using the same procedure as real postmen do in our university. The user needs to search for the correct floor, office or desk assigned to the recipient from an address file. Then she must drag the package on to the correct mailbox. This application was made to train future university postmen, and it resembles the real procedure faithfully.

The Price is Right game simulates a typical trip to the cafeteria. The waiter offers the user a product at random, and informs her of its price. In the wallet the user has money represented by actual photographs of real Euro bills. She must drag money from the wallet onto the waiter's plate, and the waiter will give her the right change in return. Upon analysing the payment, the game awards the user with a rating which will be better as the payment gets closer to the right price. The change from the previous purchase is used to continue throughout the game.

Fig. 5. On the left, the Postman Game, a package delivery simulator. On the right, the Price is Right, a money management game.

These games are oriented to desktop computer usage, although they have also been tested on multi-touch tables with real-case users. This was possible thanks to the variety of input devices supported by FLING, which made it easy to shift from one platform to another without any modification.

4.3 Data visualization and control applications

One of the great advantages of using a development environment as Adobe Air / Flash is its inherited multimedia capabilities. Video and audio content can be accessed in many ways, and FLING adds a rich interaction experience that helps in the complex task of data visualization of large collections of multimedia content.

For these reasons, FLING was employed at Carnegie Mellon University to create a video discovery application that used a geographical information system based on Google Maps™. Maps can be browsed with multi-touch gestures, and embedded videos can be opened and played with full timeline controls, which are easily triggered with finger gestures. The typical hardware setup for this application is a vertical multi-touch panel on which the graphical interface can be projected and the interaction is executed using the hands. This application is an example of different interaction techniques seamlessly combined through FLING. While the Google Maps™ API listens to mouse events, the multitouch panel generates TUIO events. In addition, the rest of the elements of the interface, such as the opened videos, listen to finger events. FLING integrates all the interactions seamlessly, significantly reducing the programmer's effort. Google Maps™ are integrated naturally, as they do in traditional PC applications, and all the functions of their API can be used directly out of the box.

Fig. 6. On the left, a video discovery application using GIS developed at the Instinctive Computing Lab on Carnegie Mellon University. On the right, a control application for an intelligent room at the Universidad Autónoma de Madrid.

The intelligent environment control application is meant to be run on a multi-touch table that accepts fiducial marker interaction. These fiducial symbols can be placed over appliances drawn on the house map to control some of their properties. For example, the intensity fiducial marker can be placed over lamps, and can be rotated clockwise to increase light brightness, or anti-clockwise to reduce it. A selection fiducial can be placed over the TV to change the channel. This application was developed to show how new input devices (fiducials in this case) can be easily added to FLING, and how its functionality is automatically incorporated to the existing gestures.

5 Conclusions and Future Work

This paper has focused on the problem of developing applications for an increasing number of interaction mechanisms. In doing so, we have stressed the necessity to distinguish between events, gestures and actions. Events depend on the interaction mechanisms, gestures represent the users' manipulation intentions and actions depend on each particular application.

Following this distinction we have developed FLING, a framework that allows:

- To easily build interaction independent applications without having to deal with interaction events.
- To build new complex gestures combining events from multiple interaction sources.

This framework is designed to be:

- Scalable: so that new interaction mechanisms and gestures can be added and integrated with existing ones.
- Multi-platform: so that it can run in the varied number of platforms in which the new interaction mechanisms are emerging.

This framework has been tested, seamlessly and over a number of platforms, through a number of applications using both traditional mouse and keyboard, as well as novel multi-touch interaction mechanisms.

This has been achieved through a modular design, separating the parsing of each input device from the gesture interpreter process. Thus, new sensors can be added by just incorporating the corresponding parser to the framework.

We are currently working in extending the interaction experience through new interaction mechanisms and merging the interaction of multiple interfaces.

Finally, working in the Ambient Intelligence domain, we look forward to enriching the interactions with context-aware information in a multi-user, distributed, intelligent environment.

Acknowledgments. This work has been partially funded by the following projects: ASIES: Adapting Social & Intelligent Environments to Support people with special needs (Ministerio de Ciencia y Educación de España, TIN2010-17344), and Vesta (Ministerio de Industria, Turismo y Comercio de España, TSI-020100-2009-828).

References

1. Lester, J., Hurvitz, P., Chaudhri, R., Hartung, C., Borriello, G.: MobileSense-Sensing modes of transportation in studies of the built environment. In: UrbanSense 2008, pp. 46--50. (2008)
2. Dragicevic, P., Fekete, J.: Input device selection and interaction configuration with ICON. In: Blanford, A., Vanderdonkt, J., Gray, P. (eds.) People and Computers XV Interaction without Frontiers: Joint proceedings of IHM 2001 and HCI 2001 (IHM-HCI'01), pp. 543--558. Springer Verlag (2001)
3. Flippo, F., Krebs, A., Marsic, I.: A framework for rapid development of multimodal interfaces. In: 5th International Conference on Multimodal Interfaces (ICMI'03), pp. 109--116. ACM, New York (2003)
4. Serrano, M., Nigay, L., Lawson, J., Ramsay, A., Murray-Smith, R., Denef, S.: The openinterface framework: a tool for multimodal interaction. In CHI'08 extended abstracts on Human factors in computing systems (CHI EA '08), pp. 3501--3506. ACM, New York (2008).
5. Touchlib: an opensource multi-touch framework, http://www.whitenoiseaudio.com/touchlib
6. Kaltenbrunner, M., Bencina, R.: reacTIVision: a computer-vision framework for table-based tangible interaction. In: 1st international conference on Tangible and embedded interaction (TEI'07), pp. 69--74. ACM, New York (2007)
7. Community Core Vision, http://ccv.nuigroup.com/
8. Touchè, http://gkaindl.com/software/touche
9. Bederson, B. B., Grosjean, J. Meyer, J: Toolkit Design for Interactive Structured Graphics. IEEE Trans. Softw. Eng. 30(8) 535--546 (2004)
10. Gokcezade, A., Leitner, J., Haller, M:. LightTracker: An Open-Source Multitouch Toolkit. J. Comput. Entertain. 8, Article 19 (2010)
11. VVVV, http://vvvv.org/
12. Hansen, T. E., Hourcade, J. P., Virbel, M., Patali, S., Serra, T.: PyMT: a post-WIMP multi-touch user interface toolkit. In: ACM International Conference on Interactive Tabletops and Surfaces (ITS '09), pp. 17--24. ACM, New York (2009)

13. De Nardi, A: Grafiti: Gesture Recognition mAnagement Framework for Interactive Tabletop Interfaces. Diploma thesis. University of Pisa (2008)
14. Surface SDK, http://msdn.microsoft.com/en-us/library/ee804845.aspx
15. Esenther, A.; Forlines, C.; Ryall, K.; Shipman, S.: DiamondTouch SDK: Support for Multi-User, Multi-Touch Applications. Mitsubishi Electronics Research Laboratory, Report No. TF2002-48 (2002)
16. Gestureworks, http://gestureworks.com/
17. König, W. A., Rädle, R., Reiterer, H.: Squidy: a zoomable design environment for natural user interfaces. In: 27th international conference extended abstracts on Human factors in computing systems (CHI EA '09), pp. 4561--4566. ACM, New York (2009)
18. Ramanahally, P., Gilbert, S., Niedzielski, T., Velázquez, D., Anagnost, C.: Sparsh UI: A Multi-Touch Framework for Collaboration and Modular Gesture Recognition. In Proc. of WINVR 2009, Conference on Innovative Virtual Reality, pp 1–-6 (2009)
19. Scholliers, C., Hoste, L., Signer, B., De Meuter, W.: Midas: a declarative multi-touch interaction framework. In: 5th International Conference on Tangible, Embedded, and embodied Interaction (TEI '11), pp. 49--56. ACM, New York (2011)
20. Echtler, F., Klinker., G.: A Multitouch Software Architecture. In 5th Nordic Conference on Human-Computer Interaction (NordiCHI 2008), pp. 463--466 (2008)
21. Laufs, U., Ruff, C., Zibuschka, J.: MT4j - A Cross-platform Multi-touch Development Framework. In: Engineering Patterns for Multi-Touch Interfaces 2010, Workshop of the ACM EICS (2010)
22. Blom, S., Book, M., Gruhn, V., Hrushchak, R., Kohler, A.: Write Once, Run Anywhere A Survey of Mobile Runtime Environments. In: 3rd International Conference on Grid and Pervasive Computing – Workshops, pp. 132--137. IEEE Press, New York (2008)
23. Jordà, S., Geiger, G., Alonso, M., Kaltenbrunner, M.: The reacTable: exploring the synergy between live music performance and tabletop tangible interfaces. In: 1st International Conference on Tangible and Embedded Interaction (TEI'07), pp. 139--146 ACM, New York (2007)
24. Bencina, R., Kaltenbrunner, M.: The design and evolution of fiducials for the reactivision system. In: 3rd International Conference on Generative Systems in the Electronic Arts (3rd Iteration 2005), Melbourne, Australia (2005)
25. Wang, F., Ren, X., Liu, Z.: A Robust Blob Recognition and Tracking Method in Vision-Based Multi-touch Technique. In: International Symposium on Parallel and Distributed Processing with Applications (ISPA'08), pp. 971--974. IEEE Press, New York (2008)
26. Kaltenbrunner, M., Bovermann, T., Bencina, R., Costanza, E.: TUIO: A protocol for table-top tangible user interfaces. In: 6th Int'l Workshop on Gesture in Human-Computer Interaction and Simulation (2005)
27. TUIO Flash client library http://www.tuio.org/?flash
28. Gaines, B., Shaw, M.: A learning model for forecasting the future of information technology, J. Future Computing Systems 1, 31--69 (1986)